Emptiness of Multi-pushdown Automata Is 2ETIME-Complete

Mohamed Faouzi Atig[1], Benedikt Bollig[2], and Peter Habermehl[1,2]

[1] LIAFA, CNRS and University Paris Diderot, France
atig+haberm@liafa.jussieu.fr
[2] LSV, ENS Cachan, CNRS, Inria
bollig@lsv.ens-cachan.fr

Abstract. We consider *multi-pushdown automata*, a multi-stack extension of pushdown automata that comes with a constraint on stack operations: a pop can only be performed on the first non-empty stack (which implies that we assume a linear ordering on the collection of stacks). We show that the emptiness problem for multi-pushdown automata is 2ETIME-complete wrt. the number of stacks. Containment in 2ETIME is shown by translating an automaton into a grammar for which we can check if the generated language is empty. The lower bound is established by simulating the behavior of an alternating Turing machine working in exponential space. We also compare multi-pushdown automata with the model of bounded-phase multi-stack (visibly) pushdown automata.

1 Introduction

Various classes of pushdown automata with multiple stacks have been proposed and studied in the literature. The main goals of these efforts are twofold. First, one may aim at extending the expressive power of pushdown automata, going beyond the class of context-free languages. Second, multi-stack systems may model recursive concurrent programs, in which any sequential process is equipped with a finite-state control and, in addition, can access its own stack to connect procedure calls to their corresponding returns. In general, however, multi-stack extensions of pushdown automata are Turing powerful and therefore come along with undecidability of basic decision problems. To retain desirable decidability properties of pushdown automata, such as emptiness, one needs to restrict the model accordingly. In [3], Breveglieri et al. define *multi-pushdown automata* (MPDA), which impose a linear ordering on stacks. Stack operations are henceforth constrained in such a way that a pop operation is reserved to the first non-empty stack. These automata are suitable to model client-server systems of processes with remote procedure calls. Another possibility to regain decidability in the presence of several stacks is to restrict the domain of input words. In [8], La Torre et al. define *bounded-phase multi-stack visibly pushdown automata* (bounded-phase MVPA). Only those runs are taken into consideration that can be split into a given number of phases, where each phase admits pop operations of one particular stack only. In the above-mentioned cases, the respective emptiness problem is decidable. In [9], the results of [8] are used to show decidability results for restricted queue systems.

M. Ito and M. Toyama (Eds.): DLT 2008, LNCS 5257, pp. 121–133, 2008.

In this paper, we resume the study of MPDA and, in particular, consider their emptiness problem. The decidability of this problem, which is to decide if an automaton admits some accepting run, is fundamental for verification purposes. We show that the emptiness problem for MPDA is 2ETIME-complete. Recall that 2ETIME is the class of all decision problems solvable by a deterministic Turing machine in time $2^{2^{dn}}$ for some constant d. In proving the upper bound, we correct an error in the decidability proof given in [3].[1] We keep their main idea: MPDA are reduced to equivalent *depth-n-grammars*. Deciding emptiness for these grammars then amounts to checking emptiness of an ordinary context-free grammar. For proving 2ETIME-hardness, we borrow an idea from [10], where a 2ETIME lower bound is shown for bounded-phase pushdown-transducer automata. We also show that $2m$-MPDA are strictly more expressive than m-phase MVPA providing an alternative proof of decidability of the emptiness problem for bounded-phase MVPA.

The paper is structured as follows: In Section 2, we introduce MPDA formally, as well as depth-n-grammars. Sections 3 and 4 then establish the 2ETIME upper and, respectively, lower bound of the emptiness problem for MPDA, which constitutes our main result. In Section 5, we compare MPDA with bounded-phase MVPA. We conclude by identifying some directions for future work. Missing proofs can be found in [1].

2 Multi-pushdown Automata and Depth-n-grammars

In this section we define *multi-pushdown automata* with $n \geq 1$ pushdown stacks and their corresponding grammars. We essentially follow the definitions of [3].

Multi-pushdown Automata. Our automata have one read-only left to right input tape and $n \geq 1$ read-write memory tapes (stacks) with a last-in-first-out rewriting policy. In each move, the following actions are performed:

- read one or zero symbol from the input tape and move past the read symbol
- read the symbol on the top of the first non-empty stack starting from the left
- switch the internal state
- for each $i \in \{1, \ldots, n\}$, write a finite string α_i on the i-th pushdown stack

Definition 1. *For $n \geq 1$, an (n-)*multi-pushdown automaton *(n-MPDA or MPDA) is a tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, F, Z_0)$ where:*

- *Q is a finite non-empty set of* internal states*,*
- *Σ (input) and Γ (memory) are finite disjoint alphabets,*
- *$\delta : Q \times (\Sigma \uplus \{\epsilon\}) \times \Gamma \rightarrow 2^{Q \times (\Gamma^*)^n}$ is a* transition mapping*,*
- *q_0 is the* initial state*,*
- *$F \subseteq Q$ is the set of* final states*, and*
- *$Z_0 \in \Gamma$ is the* initial memory symbol.

[1] A similar correction of the proof has been worked out independently by the authors of [3] themselves [4]. They gave an explicit construction for the case of three stacks that can be generalized to arbitrarily many stacks.

Table 1. A 2-MPDA for $\{\epsilon\} \cup \{a^{i_1}b^{i_1}c^{i_1}a^{i_2}b^{i_2}c^{i_2}\cdots a^{i_k}b^{i_k}c^{i_k} \mid k \geq 1$ and $i_1, \ldots, i_k > 0\}$

$M = (\{q_0, \ldots, q_3, q_f\}, \{a, b, c\}, \{A, B, Z_0, Z_1\}, \delta, q_0, \{q_f\}, Z_0)$	
$\delta(q_0, \epsilon, Z_0) = \{(q_f, \epsilon, \epsilon)\}$	$\delta(q_2, b, A) = \{(q_2, \epsilon, \epsilon)\}$
$\delta(q_0, a, Z_0) = \{(q_1, AZ_0, BZ_1)\}$	$\delta(q_2, \epsilon, Z_0) = \{(q_3, \epsilon, \epsilon)\}$
$\delta(q_1, \epsilon, A) = \{(q_2, A, \epsilon)\}$	$\delta(q_3, \epsilon, Z_1) = \{(q_0, Z_0, \epsilon)\}$
$\delta(q_1, a, A) = \{(q_1, AA, B)\}$	$\delta(q_3, c, B) = \{(q_3, \epsilon, \epsilon)\}$

A *configuration* of M is an $(n+2)$-tuple $\langle q, x; \gamma_1, \ldots, \gamma_n\rangle$ with $q \in Q$, $x \in \Sigma^*$, and $\gamma_1, \ldots, \gamma_n \in \Gamma^*$. The *transition relation* $\vdash_M^*$ is the transitive closure of the binary relation $\vdash_M$ over configurations, defined as follows:

$$\langle q, ax; \epsilon, \ldots, \epsilon, A\gamma_i, \ldots, \gamma_n\rangle \vdash_M \langle q', x; \alpha_1, \ldots, \alpha_{i-1}, \alpha_i\gamma_i, \ldots, \alpha_n\gamma_n\rangle$$

if $(q', \alpha_1, \ldots, \alpha_n) \in \delta(q, a, A)$, where $a \in \Sigma \cup \{\epsilon\}$.

The *language of M accepted by final state* is defined as the set of words $x \in \Sigma^*$ such that there are $\gamma_1, \ldots, \gamma_n \in \Gamma^*$ and $q \in F$ with $\langle q_0, x; Z_0, \epsilon, \ldots \epsilon\rangle \vdash_M^* \langle q, \epsilon; \gamma_1, \ldots, \gamma_n\rangle$. The *language of M accepted by empty stacks*, denoted by $L(M)$, is defined as the set of words $x \in \Sigma^*$ such that there is $q \in Q$ with $\langle q_0, x; Z_0, \epsilon, \ldots \epsilon\rangle \vdash_M^* \langle q, \epsilon; \epsilon, \ldots, \epsilon\rangle$.

Lemma 2 ([3]). *The languages accepted by n-MPDA by final state are the same as the languages accepted by n-MPDA by empty stacks.*

Table 1 shows an example of a 2-MPDA. Notice that it accepts the same language by final state and by empty stacks.

We need the following normal form of n-MPDA for the proof of our main theorem. The normal form restricts the operation on stacks 2 to n: pushing one symbol on these stacks is only allowed while popping a symbol from the first stack, and popping a symbol from them pushes a symbol onto the first stack. Furthermore, the number of symbols pushed on the first stack is limited to two and the stack alphabets are distinct.

Definition 3. *A n-MPDA $(Q, \Sigma, \Gamma, \delta, q_0, F, Z_0)$ with $n \geq 2$ is in normal form if*

- *$\Gamma = \bigcup_{i=1}^{n} \Gamma^{(i)}$ where the $\Gamma^{(i)}$'s are pairwise disjoint memory alphabets whose elements are denoted by $A^{(i)}, B^{(i)}$, etc., and $Z_0 \in \Gamma^{(1)}$.*
- *Only the following transitions are allowed:*
  - *For all $A^{(1)} \in \Gamma^{(1)}$ and $a \in \Sigma \cup \{\epsilon\}$, $\delta(q, a, A^{(1)}) \subseteq \{(q', \epsilon, \ldots, \epsilon) \mid q' \in Q\} \cup \Delta_1 \cup \Delta_2$ with*
    - $\Delta_1 = \{(q', B^{(1)}C^{(1)}, \epsilon, \ldots, \epsilon) \mid q' \in Q \wedge B^{(1)}, C^{(1)} \in \Gamma^{(1)}\}$,
    - $\Delta_2 = \{(q', \epsilon, \ldots, \epsilon, A^{(i)}, \epsilon, \ldots, \epsilon) \mid q' \in Q \wedge A^{(i)} \in \Gamma^{(i)} \wedge 2 \leq i \leq n\}$.
  - *For all i with $2 \leq i \leq n$ and $a \in \Sigma \cup \{\epsilon\}$,* $\delta(q, a, A^{(i)}) \subseteq \{(q', B^{(1)}, \epsilon, \ldots, \epsilon) \mid q' \in Q \wedge B^{(1)} \in \Gamma^{(1)}\}$.

Lemma 4. *An n-MPDA M can be transformed into an n-MPDA M' in normal form with linear blowup in its size such that $L(M) = L(M')$.*

Proof. The proof makes use of the ideas from [3], where a proof for a normal form for D^n-grammars (see below) is given. Notice, however, that we do not use the same normal form as the one of [3] for MPDA. □

Next, we recall some properties of the class of languages recognized by n-MPDA. We start by defining a renaming operation: A *renaming* of Σ to Σ' is a function $f : \Sigma \rightarrow \Sigma'$. It is extended to strings and languages in the natural way: $f(a_1 \ldots a_k) = f(a_1) \cdot \ldots \cdot f(a_k)$ and $f(L) = \bigcup_{x \in L} f(x)$. The following can be shown following [3].

Lemma 5. *(Closure Properties) The class of languages recognized by n-MPDA is closed under union, concatenation, and Kleene-star. Moreover, given an n-MPDA M over the alphabet Σ and a renaming function $f : \Sigma \rightarrow \Sigma'$, it is possible to construct an n-MPDA M' over Σ' such that $L(M') = f(L(M))$.*

Depth-n-grammars. We now define the notion of a depth-n-grammar. Let V_N and V_T be finite disjoint alphabets and let "(" and ")$_i$" for $i \in \{1, \ldots, n\}$ be $n+1$ characters not in $V_N \cup V_T$. An *n-list* is a finite string of the form $\overline{\alpha} = w(\alpha_1)_1(\alpha_2)_2 \ldots (\alpha_n)_n$ where $w \in V_T^*$ and $\alpha_i \in V_N^*$ for all i with $1 \leq i \leq n$.

Definition 6. *A depth-n-grammar (D^n-grammar) is a tuple $G = (V_N, V_T, P, S)$ where V_N and V_T are the finite disjoint sets of* non-terminal *and* terminal symbols*, respectively, $S \in V_N$ is the axiom, and P is a finite set of* productions *of the form $A \rightarrow \overline{\alpha}$ with $A \in V_N$ and $\overline{\alpha}$ an n-list.*

For clarity, we may drop empty components of n-lists in the productions as follows: $A \rightarrow w(\epsilon)_1 \ldots (\epsilon)_n$ is written as $A \rightarrow w$, $A \rightarrow (\epsilon)_1 \ldots (\epsilon)_n$ is written as $A \rightarrow \epsilon$, and $A \rightarrow w(\epsilon)_1 \ldots (\epsilon)_{i-1}(\alpha_i)_i(\epsilon)_{i+1} \ldots (\epsilon)_n$ is written as $A \rightarrow w(\alpha_i)_i$.

We define the *derivation relation* on n-lists as follows. Let $i \in \{1, \ldots, n\}$ and let $\overline{\beta} = (\epsilon)_1 \ldots (\epsilon)_{i-1}(A\beta_i)_i(\beta_{i+1})_{i+1} \ldots (\beta_n)_n$ be an n-list, where $\beta_j \in V_N^*$ for all $j \in \{i, \ldots, n\}$. Then,

$$x\overline{\beta} \Rightarrow xw(\alpha_1)_1(\alpha_2)_2 \ldots (\alpha_{i-1})_{i-1}(\alpha_i\beta_i)_i(\alpha_{i+1})_{i+1} \ldots (\alpha_n\beta_n)_n$$

if $A \rightarrow w(\alpha_1)_1(\alpha_2)_2 \ldots (\alpha_n)_n$ is a production and $x \in V_T^*$. Notice that only leftmost derivations are defined. As usual we denote by $\Rightarrow^*$ the reflexive and transitive closure of $\Rightarrow$. A terminal string $x \in V_T^*$ is *derivable* from S if $(S)_1(\epsilon)_2 \ldots (\epsilon)_n \Rightarrow^* x(\epsilon)_1 \ldots (\epsilon)_n$. This will be also denoted by $S \Rightarrow^* x$. The language generated by a D^n-grammar G is $L(G) = \{x \in V_T^* \mid S \Rightarrow^* x\}$.

Definition 7. *Let $G = (V_N, V_T, P, S)$ be a D^n-grammar. Then, the underlying context-free grammar is $G_{cf} = (V_N, V_T, P_{cf}, S)$ with $P_{cf} = \{A \rightarrow w\alpha_1 \ldots \alpha_n \mid A \rightarrow w(\alpha_1)_1 \ldots (\alpha_n)_n \in P\}$.*

The following lemma from [3] is obtained by observing that the language generated by a D^n-grammar is empty iff the language generated by its underlying context-free grammar G_{cf} is empty. Furthermore, it is well-known that emptiness of context-free grammars can be decided in time linear in its size.

Lemma 8. *The emptiness problem of D^n-grammars is decidable in linear time.*

3 Emptiness of MPDA is in 2ETIME

In this section, we show that the emptiness problem of n-MPDA is in 2ETIME. We first show that n-MPDA correspond to D^n-grammars with a double exponential number of non-terminal symbols. To do so, we correct a construction given in [3]. Then, emptiness of D^n-grammars is decidable using the underlying context-free grammar (Lemma 8).

Theorem 9. *A language L is accepted by an n-MPDA iff it is generated by a D^n-grammar.*

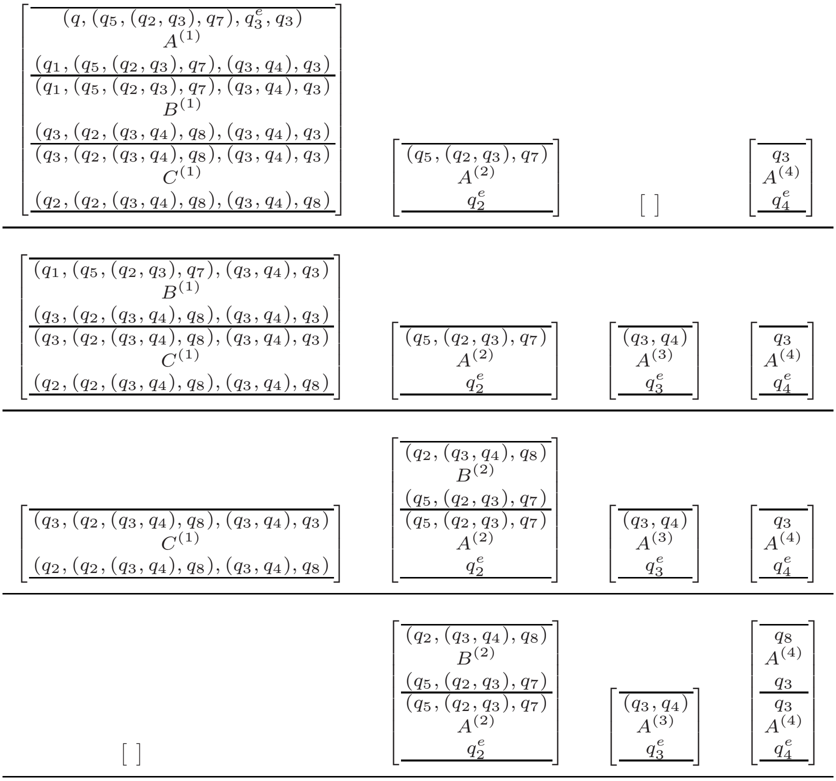
In the following we give a sketch of the proof. The “if”-direction is obvious, since a grammar is just an automaton with one state. For the “only if”-direction, let L be a language accepted by empty stacks by an n-MPDA $M = (Q, \Sigma, \Gamma, \delta, q_0, F, Z_0)$. By Lemma 4, we assume, without loss of generality, that M is in normal form. We will construct a D^n-grammar $G_M = (V_N, \Sigma, P, S)$ such that $L(G_M) = L$.

Intuitively, we generalize the proof for the case of 2-MPDA [7]. In [3], an incorrect proof was given for the case of n-MPDA. Recently, the authors of [3] independently gave a generalizable proof for 3-MPDA, which is similar to ours [4]. The general proof idea is the same as for the corresponding proof for pushdown automata. To eliminate states, one has to guess the sequence of states through which the automaton goes by adding pairs of state symbols to the non-terminal symbols of the corresponding grammar. We do this for the first stack. However, when the first stack gets empty, the other stacks may be not empty and one has to know the state in which the automaton is in this situation. For this, we have to guess for all the other non-empty stacks and each of their non-terminal symbols the state in which the automaton will be when reading these symbols. [2]

To do this for the n-th stack, a pair of state symbols is enough. For the $(n-1)$-th stack, in addition to guessing the state, we also have to know the current state on top of the n-th stack to be able to push correctly symbols onto the n-th stack. Therefore, a pair of pairs of states (4 in total) is needed. For the $(n–2)$-th stack, we need to remember the current state and the states on top of the $(n-1)$-th stack and on top of the n-th stack (in total 8 states) and so on. Therefore, there will be 2^n state symbols to be guessed in the first stack. Furthermore we have special state symbols (denoted q_i^e) to indicate that the i-th stack is empty. In Fig. 1 we give an intuitive example illustrating the construction.

Now we define the grammar $G_M = (V_N, \Sigma, P, S)$ formally. To define V_N, we first provide symbols of level i denoted by V_i. For i with $2 \leq i \leq n$, let q_i^e be states pairwise different and different from any state of Q (these are the symbols indicating that the corresponding stack is empty). States of level i are denoted by Q_i and defined as follows : $Q_n = Q \cup \{q_n^e\}$ and for all i such that $2 \leq i < n$, $Q_i = (Q \times Q_{i+1} \times \cdots \times Q_n) \cup \{q_i^e\}$, and $Q_1 = Q \times Q_2 \times \cdots \times Q_n$. We denote by $\boldsymbol{q_i}$ states of Q_i. Then, $V_i = Q_i \times \Gamma \times Q_i$ and $V_N = \{S\} \cup \bigcup_{i=1}^{n} V_i$. Notice that a state in Q_i different from q_i^e has exactly 2^{n-i} components. Therefore $|V_N| \leq (|Q|+1)^{2^{n+1}}|\Gamma|$. The set P contains exactly the following productions, which are partitioned into five types ($a \in \Sigma \cup \{\epsilon\}$):

[2] The proof in [3] incorrectly assumes that this state is the same for each stack when the first stack gets empty.



$(q, (q_5, (q_2, q_3), q_7), q_3^e, q_3)$
$A^{(1)}$
$(q_1, (q_5, (q_2, q_3), q_7), (q_3, q_4), q_3)$
$B^{(1)}$
$(q_3, (q_2, (q_3, q_4), q_8), (q_3, q_4), q_3)$
$C^{(1)}$
$(q_2, (q_2, (q_3, q_4), q_8), (q_3, q_4), q_8)$
$(q_5, (q_2, q_3), q_7)$
$A^{(2)}$
q_2^e
(q_3, q_4)
$A^{(3)}$
q_3^e
q_3
$A^{(4)}$
q_4^e
$(q_2, (q_3, q_4), q_8)$
$B^{(2)}$
q_8

Fig. 1. A sketch of a partial derivation (from top to bottom) of a depth-4-grammar corresponding to a run of a 4-MPDA where three symbols are popped from the first stack while three symbols are pushed onto the other stacks. In each configuration, if the first stack is non-empty, then the state symbols on top of the other stacks can be found on top of the first stack as well. In the last configuration, the top symbols of the other stacks can be found on top of the second stack.

T1 $S \rightarrow ([(q_0, q_2^e, \ldots, q_n^e), Z_0, (q^1, \boldsymbol{q_2^1}, \ldots, \boldsymbol{q_n^1})])_1$
if there is k with $2 \leq k \leq n+1$ such that
- for all i with $2 \leq i < k$ we have $\boldsymbol{q_i^1} = q_i^e$
- if $k \leq n$, then $\boldsymbol{q_k^1} = (q^1, \boldsymbol{q_{k+1}^1}, \ldots, \boldsymbol{q_n^1})$

T2 $[(q^1, \boldsymbol{q_2^1}, \ldots, \boldsymbol{q_n^1}), A^{(1)}, \boldsymbol{q_1^2}] \rightarrow a([(q^4, \boldsymbol{q_2^1}, \ldots, \boldsymbol{q_n^1}), B^{(1)}, \boldsymbol{q_1^3}][\boldsymbol{q_1^3}, C^{(1)}, \boldsymbol{q_1^2}])_1$
if $(q^4, B^{(1)}C^{(1)}, \epsilon, \ldots, \epsilon) \in \delta(q^1, a, A^{(1)})$

T3 $[(q^1, \boldsymbol{q_2^1}, \ldots, \boldsymbol{q_{j-1}^1}, \boldsymbol{q_j^1}, \boldsymbol{q_{j+1}^1}, \ldots, \boldsymbol{q_n^1}), A^{(1)}, (q^2, \boldsymbol{q_2^1}, \ldots, \boldsymbol{q_{j-1}^1}, \boldsymbol{q_j^2}, \boldsymbol{q_{j+1}^1}, \ldots, \boldsymbol{q_n^1})]$
$\rightarrow a([\boldsymbol{q_j^2}, B^{(j)}, \boldsymbol{q_j^1}])_j$ if $\boldsymbol{q_j^2} \neq q_j^e$ and $(q^2, \epsilon, \ldots, \epsilon, B^{(j)}, \epsilon, \ldots, \epsilon) \in \delta(q^1, a, A^{(1)})$

T4 $[(q^1, \boldsymbol{q_{j+1}^1}, \ldots, \boldsymbol{q_n^1}), A^{(j)}, \boldsymbol{q_j^1}]$
$\rightarrow a([(q^4, q_2^e, \ldots, q_{j-1}^e, \boldsymbol{q_j^1}, \boldsymbol{q_{j+1}^1}, \ldots, \boldsymbol{q_n^1}), B^{(1)}, (q^2, \boldsymbol{q_2^2}, \ldots, \boldsymbol{q_n^2})])_1$
if $(q^4, B^{(1)}, \epsilon, \ldots, \epsilon) \in \delta(q^1, a, A^{(j)})$, and there is k with $2 \leq k \leq n+1$ such that
- for all i with $2 \leq i < min(k, j)$ we have $\boldsymbol{q_i^2} = q_i^e$
- for all i with $min(k, j) \leq i < k$ we have $\boldsymbol{q_i^1} = \boldsymbol{q_i^2} = q_i^e$
- if $k \leq n$, then $\boldsymbol{q_k^2} = (q^2, \boldsymbol{q_{k+1}^2}, \ldots, \boldsymbol{q_n^2})$

T5 $[(q^1, \boldsymbol{q}_2^1, \ldots, \boldsymbol{q}_n^1), A^{(1)}, (q^2, \boldsymbol{q}_2^1, \ldots, \boldsymbol{q}_n^1)] \rightarrow a$ if $(q^2, \epsilon, \ldots, \epsilon) \in \delta(q^1, a, A^{(1)})$

The grammar corresponding to the example in Table 1 can be found in [1]. The following key lemma formalizes the intuition about derivations of the grammar G_M by giving invariants satisfied by them (illustrated in Fig. 1). This lemma is the basic ingredient of the full proof of Theorem 9, which can be found in [1]. Intuitively, condition 1 says that the first element of the first stack contains the state symbols on top of the other stacks. Condition 2 says that the last state symbols in the first stack are of the form allowing condition 3 to be true when the corresponding symbol is popped. Condition 3 says that if the first stack is empty, then the top of the first non-empty stack contains the same state symbols as the top of the other stacks. Conditions 4 and 5 say that the state symbols guessed form a chain through the stacks.

Lemma 10. *Let $w(\gamma_1)(\gamma_2)\ldots(\gamma_n)$ be an n-list different from $(\epsilon)_1\ldots(\epsilon)_n$ appearing in a derivation of the grammar G_M.*

1. *If $\gamma_1 = [(q^1, \boldsymbol{q_2^1}, \ldots, \boldsymbol{q_n^1}), A^{(1)}, (q^2, \boldsymbol{q_2^2}, \ldots, \boldsymbol{q_n^2})]\gamma_1'$ with $\gamma_1' \in V_1^*$, then for all i with $2 \leq i \leq n$, if γ_i is empty, then $\boldsymbol{q_i^1} = q_i^e$, else $\gamma_i = [\boldsymbol{q_i^1}, B^{(i)}, \boldsymbol{q_i^3}]\gamma_i'$ with $\gamma_i' \in V_i^*$.*
2. *If $\gamma_1 = \gamma_1'[(q^1, \boldsymbol{q_2^1}, \ldots, \boldsymbol{q_n^1}), A^{(1)}, (q^3, \boldsymbol{q_2^3}, \ldots, \boldsymbol{q_n^3})]$ with $\gamma_1' \in V_1^*$, then there exists k with $2 \leq k \leq n+1$ such that we have both for all i with $2 \leq i < k$, $\boldsymbol{q_i^3} = q_i^e$ and $k \leq n$ implies $\boldsymbol{q_k^3} = (q^3, \boldsymbol{q_{k+1}^3}, \ldots, \boldsymbol{q_n^3})$.*
3. *Suppose that $\gamma_1 = \epsilon$. Let i be the smallest k such that γ_k is not empty and let $\gamma_i = [(q^1, \boldsymbol{q_{i+1}^1}, \ldots, \boldsymbol{q_n^1}), A^{(i)}, \boldsymbol{q_i^2}]\gamma_i'$ with $\gamma_i' \in V_i^*$. Then, for all $j > i$, we have: if γ_j is empty, then $\boldsymbol{q_j^1} = q_j^e$, else $\gamma_j = [\boldsymbol{q_j^1}, A^{(j)}, \boldsymbol{q_j^3}]\gamma_j'$ with $\gamma_j' \in V_j^*$.*
4. *For all i with $2 \leq i \leq n$, if γ_i is not empty then for some $j \geq 1$, $\gamma_i = [\boldsymbol{q_i^1}, A_1^{(i)}, \boldsymbol{q_i^2}][\boldsymbol{q_i^2}, A_2^{(i)}, \boldsymbol{q_i^3}]\ldots[\boldsymbol{q_i^{j-1}}, A_{j-1}^{(i)}, \boldsymbol{q_i^j}][\boldsymbol{q_i^j}, A_j^{(i)}, q_i^e]$ and for all l with $1 \leq l \leq j$, $\boldsymbol{q_i^l} \neq q_i^e$.*
5. *If γ_1 is not empty, then for some $j \geq 1$, $\gamma_1 = [\boldsymbol{q_1^1}, A_1^{(1)}, \boldsymbol{q_1^2}][\boldsymbol{q_1^2}, A_2^{(1)}, \boldsymbol{q_1^3}]\ldots[\boldsymbol{q_1^{j-1}}, A_{j-1}^{(1)}, \boldsymbol{q_1^j}][\boldsymbol{q_1^j}, A_j^{(1)}, \boldsymbol{q_1^{j+1}}]$.*

By observing that the size of the grammar G_M corresponding to an MPDA M in the construction used in the proof of Theorem 9 is double exponential in the number of stacks and using Lemma 8 we obtain the following corollary.

Corollary 11. *The emptiness problem of MPDA is in 2ETIME.*

In the next Section, it is shown that the double exponential upper bound is tight.

4 Emptiness of MPDA Is 2ETIME-Hard

In this section, we prove that the emptiness problem of MPDA is 2ETIME-hard. This is done by adapting a construction in [10], where it is shown that certain bounded-phase pushdown-transducer automata capture precisely the class 2ETIME.

Theorem 12. *The emptiness problem for MPDA is 2ETIME-hard under logspace reductions.*

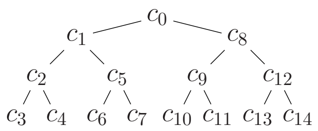


Fig. 2. A run of an alternating Turing machine

Proof. It is well-known that the class of problems solvable by alternating Turing machines in space bounded by 2^{dn} for some d (call it AESPACE) equals 2ETIME [5]. Thus, it is sufficient to show that any problem in AESPACE can be reduced, in logarithmic space, to the emptiness problem for MPDA.

So let T be an alternating Turing machine working in space bounded by 2^{dn}. Let furthermore w be an input for T of length n. We construct (in logarithmic space) from T and w an MPDA M with $2dn + 4$ stacks such that the language of M is non-empty iff w is accepted by T. The simulation of T proceeds in two phases: (1) M guesses a possible accepting run of T on w; (2) M verifies if the guess is indeed a run.

Without loss of generality, we can assume that a transition of T is basically of the form $c \longrightarrow (c_1 \wedge c_2) \vee (c_3 \wedge c_4) \vee \ldots \vee (c_{h-1} \wedge c_h)$ (where configuration changes are local), i.e., from configuration c, we might switch to both c_1 and c_2 or both c_3 and c_4 and so on. This allows us to represent a run of T as a complete finite binary tree, as shown in Fig. 2, whose nodes are labeled with configurations. Note that each configuration will be encoded as a string, as will be made precise below. The run is accepting if all leaf configurations are accepting. Following the idea of [10], we write the labeled tree as the string (let c^r denote the reverse of c)

$$\begin{array}{l} c_0|c_1|c_2|c_3 \parallel c_3^r \parallel c_4 \parallel c_4^r|c_2^r \parallel c_5|c_6 \parallel c_6^r \parallel c_7 \parallel c_7^r|c_5^r|c_1^r \parallel \\ c_8|c_9|c_{10} \parallel c_{10}^r \parallel c_{11} \parallel c_{11}^r|c_9^r \parallel c_{12}|c_{13} \parallel c_{13}^r \parallel c_{14} \parallel c_{14}^r|c_{12}^r|c_8^r|c_0^r \end{array}$$

It is generated by the (sketched) context-free grammar

$$\begin{array}{lcl} A & \longrightarrow & \alpha_i A \alpha_i \;+\; \alpha_i B \alpha_i \;+\; \alpha_i \| \alpha_i \\ B & \longrightarrow & |A \parallel A| \end{array}$$

where the α_i are the atomic building blocks of an encoding of a configuration of T. This string allows us to access locally those pairs of configurations that are related by an edge in the tree and thus need to agree with a transition. Finally, the grammar can make sure that all leafs are accepting configurations and that the initial configuration corresponds to the input w. Using two stacks, we can generate such a word encoding of a (possible) run of T and write it onto the second stack, say with c_0 at the top, while leaving the first stack empty behind us (cf. Fig. 3(a)).

The MPDA M now checks if the word written onto stack 2 stems from a run of T. To this aim, we first extract from stack 2 any pair of configurations that needs to be compared wrt. the transition relation of T. For this purpose, some of the configurations need to be duplicated. Corresponding configurations are written side by side as follows: By means of two further stacks, 3 and 4, we transfer the configurations located on stack 2 and separated by the symbol "|" onto the third stack (in reverse order), hereby copying some configuration by writing it onto the fourth stack (cf. Fig. 3(b)).

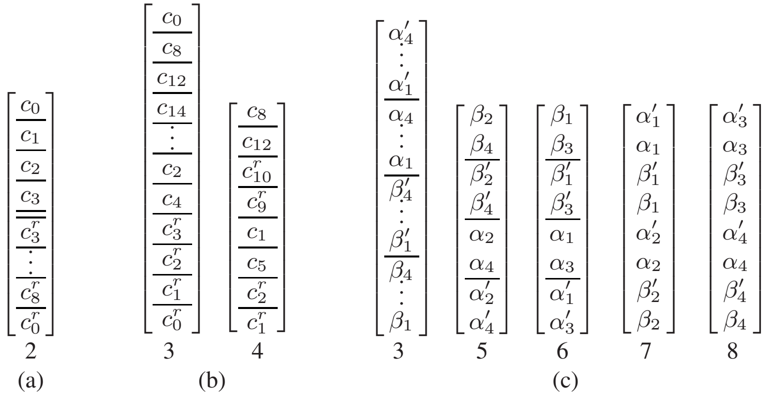

c_0 c_1 c_2 c_3 c_3^r c_8^r c_0^r
2
(a)
c_0 c_8 c_{12} c_{14} c_2 c_4 c_3^r c_2^r c_1^r c_0^r
3
c_8 c_{12} c_{10}^r c_9^r c_1 c_5 c_2^r c_1^r
4
(b)
α_4' α_1' α_4 α_1 β_4' β_1' β_4 β_1
3
β_2 β_4 β_2' β_4' α_2 α_4 α_2' α_4'
5
β_1 β_3 β_1' β_3' α_1 α_3 α_1' α_3'
6
α_1' α_1 β_1' β_1 α_2' α_2 β_2' β_2
7
α_3' α_3 β_3' β_3 α_4' α_4 β_4' β_4
8
(c)

Fig. 3. Guessing and verifying a run of an alternating Turing machine

It still remains to verify that c_0 and c_8 belong to a transition of T, as well as c_{12} and c_{14}, etc. The encoding of one single configuration $a_1 \ldots (q, a_i) \ldots a_{2^{dn}}$ will now allow us to compare two configurations letter by letter. It has the form $(-, a_1, a_2, e)$ $(a_1, a_2, a_3, e) \ldots (a_{i-1}, (q, a_i), a_{i+1}, e) \ldots (a_{2^{dn}-1}, a_{2^{dn}}, -, e)$ where the component e denotes a "transition" $c \rightarrow c' \wedge c''$, which has been selected to be executed next and which has been guessed in the above grammar. We would like to compare the k-th letter of one with the k-th letter of another configuration. To access corresponding letters simultaneously, we divide the configurations on stacks 3 and 4 into two, using two further stacks, 5 and 6. We continue this until corresponding letters are arranged one below the other. This procedure, which requires $2dn$ additional stacks, is illustrated in Fig. 3(c) where each α_i and β_i stands for an atomic symbol of the form (a_1, a_2, a_3, e). Note that, in some cases, we encounter pairs of the form (c, c') whereas in some other cases, we face pairs of the form $(c^r, (c')^r)$. Whether we deal with the reverse of a configuration or not can be recognized on the basis of its border symbols (i.e., $(-, a_1, a_2, e)$ or $(a_{2^{dn}-1}, a_{2^{dn}}, -, e)$). Consider, for example, stacks 3 and 4 in Fig. 3(b). We want to compare c_0 and c_8 where c_0 is of the form $(-, a_1, a_2, e) \ldots$, i.e., it is read in the correct order. Suppose e is of the form $c_0 \rightarrow c \wedge c'$. Then, locally comparing c_0 and c_8, we can check whether $c' = c_8$. If, at the bottom of stack 3, we compare $c_1^r = (a_{2^{dn}-1}, a_{2^{dn}}, -, e) \ldots$ with c_0^r and e is of the form $c_0 \rightarrow c \wedge c'$, then we need to check if $c = c_1$. In other words, the order in which a configuration is read indicates if we follow the right or left successor in the (tree of the) run. □

From Corollary 11 and Theorem 12, we deduce our main result:

Theorem 13. *The emptiness problem of MPDA is 2ETIME-complete under logspace reductions.*[3]

[3] The emptiness problem of MPDA is 2EXPTIME-complete, too. Hereby, 2EXPTIME denotes the class of all decision problems solvable by a deterministic Turing machine in time $exp(exp(n^d))$ for some constant d ($exp(x)$ denoting 2^x). Note that 2EXPTIME is a robust complexity class. On the other hand, 2ETIME is not robust, as it is not closed under logspace reductions.

5 Comparison to Bounded-Phase Multi-stack Pushdown Automata

In this section, we recall m-phase *multi-stack (visibly) pushdown automata* ($m \geq 1$) defined in [8] and show that they are strictly less expressive than $2m$-MPDA.

Multi-stack Visibly Pushdown Automata. For $n \geq 1$, an *n-stack call-return alphabet* is a tuple $\widetilde{\Sigma}_n = \langle \{(\Sigma_c^i, \Sigma_r^i)\}_{i \in \{1,\ldots,n\}}, \Sigma_{int} \rangle$ of pairwise disjoint finite alphabets. For $i \in \{1, \ldots, n\}$, Σ_c^i is the set of *calls of the stack* i, Σ_r^i is the set of *returns of the stack* i, and Σ_{int} is the set of *internal actions*. For any such $\widetilde{\Sigma}_n$, let $\Sigma_c = \bigcup_{i=1}^n \Sigma_c^i$, $\Sigma_r = \bigcup_{i=1}^n \Sigma_r^i$, $\Sigma^i = \Sigma_c \cup \Sigma_r^i \cup \Sigma_{int}$, for every $i \in \{1, \ldots, n\}$, and $\Sigma = \Sigma_c \cup \Sigma_r \cup \Sigma_{int}$.

Definition 14. *A* multi-stack visibly pushdown automaton (MVPA) *over the n-stack call-return alphabet $\widetilde{\Sigma}_n = \langle \{(\Sigma_c^i, \Sigma_r^i)\}_{i \in \{1,\ldots,n\}}, \Sigma_{int} \rangle$ is a tuple $N = (Q, \Gamma, \Delta, q_0, F)$ where Q is a finite set of* states*, Γ is a finite* stack alphabet *containing a distinguished stack symbol $\bot$, $\Delta \subseteq (Q \times \Sigma_c \times Q \times (\Gamma \setminus \{\bot\})) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_{int} \times Q)$ is the* transition relation*, $q_0 \in Q$ is the* initial state*, and $F \subseteq Q$ is the set of* final states.

A configuration of N is an $(n+2)$-tuple $\langle q, x; \gamma_1, \ldots, \gamma_n \rangle$ where $q \in Q$, $x \in \Sigma^*$, and for all $i \in \{1, \ldots, n\}$, $\gamma_i \in \Gamma^*$ is the content of stack i. The *transition relation* $\vdash_N^*$ is the transitive closure of the binary relation $\vdash_N$ over configurations, defined as follows: $\langle q, ax; \gamma_1, \ldots, \gamma_n \rangle \vdash_N \langle q', x; \gamma_1', \ldots, \gamma_n' \rangle$ if one of the following cases holds:

1. **Internal move:** $a \in \Sigma_{int}$, $(q, a, q') \in \Delta$, and $\gamma_i = \gamma_i'$ for every $i \in \{1, \ldots, n\}$.
2. **Push onto stack i:** $a \in \Sigma_c^i$, $\gamma_j' = \gamma_j$ for every $j \neq i$, and there is $A \in \Gamma \setminus \{\bot\}$ such that $(q, a, q', A) \in \Delta$ and $\gamma_i' = A\gamma_i$.
3. **Pop from stack i:** $a \in \Sigma_r^i$, $\gamma_j' = \gamma_j$ for every $j \neq i$, and there is $A \in \Gamma$ such that $(q, a, A, q') \in \Delta$ and either $A \neq \bot$ and $\gamma_i = A\gamma_i'$, or $A = \bot$ and $\gamma_i = \gamma_i' = \bot$.

A string $x \in \Sigma^*$ is *accepted* by N if there are $\gamma_1, \ldots, \gamma_n \in \Gamma^*$ and $q \in F$ such that $\langle q_0, x; \bot, \ldots, \bot \rangle \vdash_N^* \langle q, \epsilon; \gamma_1, \ldots, \gamma_n \rangle$. The language of N, denoted $L(N)$, is the set of all strings accepted by N.

Definition 15. *For $m \geq 1$, an m-phase multi-stack visibly pushdown automaton (m-MVPA) over the n-stack call-return alphabet $\widetilde{\Sigma}_n$ is a tuple $K = (m, Q, \Gamma, \Delta, q_0, F)$ where $N = (Q, \Gamma, \Delta, q_0, F)$ is an* MVPA *over $\widetilde{\Sigma}_n$. The language accepted by K is $L(K) = \bigcup_{i_1, \ldots, i_m \in \{1,\ldots,n\}} \big(L(N) \cap \big((\Sigma^{i_1})^* \cdots (\Sigma^{i_m})^* \big) \big)$.*

Finally, we recall that the class of languages accepted by m-MVPA is closed under union, intersection, renaming, and complementation [8]. However, one easily shows:

Lemma 16. *The class of languages of m-MVPA is not closed under Kleene-star.*

$2m$-MPDA are Strictly More Expressive than m-MVPA. We now show that, for any $m \geq 1$, $2m$-MPDA are strictly more expressive than m-MVPA. Let us fix an m-MVPA $K = (m, Q, \Gamma, \Delta, q_0, F)$ over $\widetilde{\Sigma}_n = \langle \{(\Sigma_c^i, \Sigma_r^i)\}_{i \in \{1,\ldots,n\}}, \Sigma_{int} \rangle$, with $N = (Q, \Gamma, \Delta, q_0, F)$ an MVPA.

Proposition 17. *For every sequence* $i_1, \ldots, i_m \in \{1, \ldots, n\}$, *it is possible to construct a* $2m$-MPDA M *such that* $L(M) = L(N) \cap \big((\Sigma^{i_1})^* \cdots (\Sigma^{i_m})^*\big)$.

In the following, we sketch the proof. Intuitively, any computation of N accepting a string $x \in L(N) \cap \big((\Sigma^{i_1})^* \cdots (\Sigma^{i_m})^*\big)$ can be decomposed into m phases, where in each phase (say j), N can only pop from the stack i_j (but it can push onto all stacks).

Let $j \in \{1, \ldots, m\}$ be the current phase of N and for every $l \in \{1, \ldots, n\}$, let $k_l^j = min\big(\{k \,|\, j \leq k \leq m \wedge i_k = l\} \cup \{m{+}1\}\big)$ denote the closest phase in $\{j, \ldots, m\}$ such that N can pop from the l-th stack if the phase is k_l^j (note that $k_{i_j}^j = j$), if such phase does not exist, then $k_l^j = m + 1$.

We construct a $2m$-MPDA M over Σ such that the following invariant is preserved during the simulation of N when its current phase is j: the content of the l-th stack of N is stored in the $(2k_l^j - 1)$-th stack of M if $k_l^j \neq m + 1$. Then, an internal move (labeled by $a \in \Sigma_{int}$) of N is simulated by an internal move (labeled by a) of M; a pop rule (labeled by $a \in \Sigma_r^{i_j}$) of N from the i_j-th stack corresponds to a pop rule (labeled by a) of M from the $(2j - 1)$-th stack; and a push rule (labeled by $a \in \Sigma_c^l$) onto the l-th stack of N is simulated by a push rule (labeled by a) of M onto the $(2k_l^j - 1)$-th stack if $k_l^j \neq (m + 1)$, else by an internal move (labeled by a) of M .

On switching phase from j to $(j + 1)$ if $k_{i_j}^{j+1} \neq m + 1$, when N is able once again to pop from the (i_j)-th stack, M moves the content of the $(2j - 1)$-th stack onto the $(2k_{i_j}^{j+1} - 1)$-th stack using the $(2j)$-th stack as an intermediary one, else it removes the content of the $(2j-1)$-th stack. Observe that all the above described behaviors maintain the stated invariant since $k_l^{j+1} = k_l^j$ for every $l \neq i_j$.

We are now ready to present the main result of this section.

Theorem 18. $2m$*-MPDA are strictly more expressive than* m*-MVPA.*

Proof. For every m-MVPA K over the stack alphabet $\widetilde{\Sigma}_n$ one can construct a $2m$-MPDA M over Σ such that $L(M) = L(K)$ by considering all possible orderings of phases (fixing for each phase the stack which can be popped) and using Proposition 17. To prove *strict* inclusion, we notice that the class of languages recognized by $2m$-MPDA is closed under Kleene-star (Lemma 5) but the class of languages of m-MVPA is not (Lemma 16). □

$2m$-MPDA are Strictly More Expressive than m-MPA. In the following, we extend the previous result to m-phase multi-stack pushdown automata over non-visible alphabets (defined in [8]). A multi-stack pushdown automaton (called MPA) over (non-visible) alphabet Σ is simply an n-stack automaton with ϵ-moves, that can push and pop from any stack when reading any letter. Also, we define m-phase version of these (called m-MPA). An m-MPA is an MPA using at most m-phases, where in each phase one can pop from one distinguished stack, and push on any other stack.

Theorem 19. $2m$*-MPDA are strictly more expressive than* m*-MPA.*

The idea behind proving inclusion is that for any m-MPA K over Σ, it is possible to construct an m-MVPA K' over $\widetilde{\Sigma'}_n = \langle\{(\Sigma'^i_c, \Sigma'^i_r)\}_{i \in \{1,\ldots,n\}}, \Sigma'_{int}\rangle$, with $\Sigma'^i_c =$

$\big(\Sigma \cup \{\epsilon\}\big) \times \{c\} \times \{i\}$, ${\Sigma'}^i_r = \big(\Sigma \cup \{\epsilon\}\big) \times \{r\} \times \{i\}$, and $\Sigma'_{int} = \big(\Sigma \cup \{\epsilon\}\big) \times \{int\}$, such that every transition on $a \in \Sigma \cup \{\epsilon\}$ that pushes onto the stack i is transformed to a transition on (a, c, i), transitions on a that pop the stack i are changed to transitions on (a, r, i), and the remaining a-transitions are changed to transitions over (a, int). Let f be a renaming function that maps each symbol (a, c, i), (a, r, i), and (a, int) to a. Then, $w \in L(K)$ iff there is some $w' \in L(K')$ such that $w = f(w')$. It follows that $L(K) = f\big(L(K')\big)$. Consider now the $2m$-MPDA M' over Σ' constructed from K' such that $L(M') = L(K')$, thanks to Theorem 18. Then, it is possible to construct from M' a $2m$-MPDA M over Σ such that $L(M) = f\big(L(M')\big)$ (Lemma 5) which implies that $L(M) = L(K)$. To prove the *strict* inclusion we use the easy to see fact that m-MPA are not closed under Kleene-star whereas $2m$-MPDA are (Lemma 5).

6 Conclusion

We have shown that the emptiness problem for multi-pushdown automata (MPDA) is 2ETIME-complete. The study of the emptiness problem is the first step of a comprehensive study of verification problems for MPDA. For standard pushdown automata, a lot of work has been done recently (see for example [2]) concerning various model-checking problems. It will be interesting to see how these results carry over to MPDA and at which cost. A basic ingredient of model-checking algorithms is typically to characterize the set of successors or predecessors of sets of configurations. For MPDA, this problem remains to be studied. Another class of extended pushdown automata has recently been studied extensively: the class of higher-order pushdown automata (HPDA, see for example [6]). It is quite easy to see that HPDA of order n can simulate MPDA with n stacks (which allows us to use all verification results for HPDA also for MPDA). However, the converse is wrong, since emptiness of pushdown automata of order n is $(n-1)$-EXPTIME-complete [6]. Therefore, it is interesting to study dedicated algorithms for the verification of MPDA.

References

1. Atig, M.F., Bollig, B., Habermehl, P.: Emptiness of multi-pushdown automata is 2ETIME-complete. Research Report LSV-08-16, LSV, ENS Cachan (May 2008), `http://www.lsv.ens-cachan.fr/Publis/RAPPORTS_LSV/PDF/rr-lsv-2008-16.pdf`
2. Bouajjani, A., Esparza, J., Maler, O.: Reachability analysis of pushdown automata: Application to model-checking. In: Mazurkiewicz, A., Winkowski, J. (eds.) CONCUR 1997. LNCS, vol. 1243, pp. 135–150. Springer, Heidelberg (1997)
3. Breveglieri, L., Cherubini, A., Citrini, C., Crespi Reghizzi, S.: Multi-push-down languages and grammars. International Journal of Foundations of Computer Science 7(3), 253–292 (1996)
4. Breveglieri, L., Cherubini, A., Crespo Reghizzi, S.: Personal communication
5. Chandra, A.K., Kozen, D.C., Stockmeyer, L.J.: Alternation. J. ACM 28(1), 114–133 (1981)
6. Engelfriet, J.: Iterated stack automata and complexity classes. Information and Computation 95(1), 21–75 (1991)

7. San Pietro, P.: Two-stack automata. Technical Report 92-073, Dipartimento di elettronica e informazione, Politechnico di Milano (1992)
8. La Torre, S., Madhusudan, P., Parlato, G.: A robust class of context-sensitive languages. In: Proceedings of LICS, pp. 161–170. IEEE, Los Alamitos (2007)
9. La Torre, S., Madhusudan, P., Parlato, G.: Context-bounded analysis of concurrent queue systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 299–314. Springer, Heidelberg (2008)
10. La Torre, S., Madhusudan, P., Parlato, G.: An infinite automaton characterization of double exponential time. In: Proceedings of CSL 2008. LNCS. Springer, Heidelberg (to appear, 2008)