A Robust Class of Context-Sensitive Languages

Salvatore La Torre[*]
Università di Salerno, Italy
latorre@dia.unisa.it

P. Madhusudan
University of Illinois
Urbana-Champaign, USA
madhu@uiuc.edu

Gennaro Parlato
Università di Salerno, Italy
University of Illinois (UC), USA
parlato@uiuc.edu

Abstract

*We define a new class of languages defined by multi-stack automata that forms a robust subclass of context-sensitive languages, with decidable emptiness and closure under boolean operations. This class, called multi-stack visibly pushdown languages (*MVPL*s), is defined using multi-stack pushdown automata with two restrictions: (a) the pushdown automaton is* visible*, i.e. the input letter determines the operation on the stacks, and (b) any computation of the machine can be split into k stages, where in each stage, there is at most one stack that is* popped*.* MVPL*s are an extension of visibly pushdown languages that captures non-context free behaviors, and has applications in analyzing abstractions of multithreaded recursive programs, significantly enlarging the search space that can be explored for them. We show that* MVPL*s are closed under boolean operations, and problems such as emptiness and inclusion are decidable. We characterize* MVPL*s using monadic second-order logic over appropriate structures, and exhibit a Parikh theorem for them.*

1. Introduction

Linear labeled structures play an important role in modeling information in computer science. Explicit linear representations of data (like a text document) and implicit linear representations of behavior (like the linear-time behavior of a system) motivate the study of robust classes of linear structures. Regular word languages have played an important role in this regard, and have found applications in searching documents, defining logics for behaviors, and algorithms for model-checking systems (see [8, 11, 19, 20, 22]).

In certain linear models, there is an implicit relation on the positions in the model that can be inferred from its labels. A standard example of these are XML/SGML documents where open- and close-tag annotations form a *nesting relation* on the linear document, capturing the *hierarchical tree structure* of the data it represents. When studying the linear behaviors of a recursive sequential program, there is again a nesting relation naturally present, which relates calls to procedures with their corresponding returns. Viewing the nesting structure as *explicit* edges in the document led to the recently studied theory of nested words ([1, 2]). The notion of regularity for nested words captures a different class of languages than regular word languages (i.e. regular nested word languages can be context-free when the nesting edge relation is ignored), but forms a robust class closed under boolean operations and with decidable decision problems.

In this paper, we delve deeper into defining tractable classes of word languages, where the tractability stems from explicit relations between positions that simplifies the internal complexity of the word. The class of languages we discover in this paper is a robust and tractable subclass of context-sensitive languages, defined using a restricted class of *multiple* nesting relations on words. The class is closed under all boolean operations, and admits tractable decision procedures for the problems of emptiness and inclusion.

From an automata-theoretic perspective, the regular class of words with a single nesting relation is captured using *visibly pushdown automata* (VPA) [1]. A VPA works on words over an alphabet Σ that is partitioned into three parts, called the call-alphabet, the return-alphabet, and the internal-alphabet. A word over such a partitioned alphabet implicitly defines a nesting relation, formed by matching the call-letters with the corresponding return-letters in a nested fashion. A VPA working over such a word is constrained to push exactly one symbol onto its stack on call-letters, to pop exactly one symbol on return-letters, and to not touch the stack on internal-letters. The class of languages accepted by these automata are called visibly pushdown languages, and corresponds exactly to the class of regular languages over nested words ([1, 2]).

The class of languages we study in this paper is defined using visibly pushdown automata working with *multiple* stacks. These automata read words and manipulate a finite set of stacks (say n of them), where the input alpha-

[*]MIUR grants ex-60% 2005 and 2006 Università degli Studi di Salerno.

bet determines the operations allowed on the stack. More precisely, the input alphabet is partitioned into n sets of call alphabets (Σ_c^i) and return alphabets (Σ_r^i), and an internal alphabet (Σ_{int}), and the automaton pushes or pops from the i'th stack only when it reads a call from Σ_c^i or a return in Σ_r^i, respectively. The automaton hence retrieves, using one stack for each nesting relation, the nesting edges present in the word.

Multi-stack visibly pushdown automata can be seen as encoding the *runs* of a multi-stack automaton, and it is easy to see that the emptiness problem for them is undecidable (since a Turing machine can be simulated using two stacks). Consequently, any naturally-defined notion of regularity on multiple nested words is bound to be intractable. We restrict the class of words in the following way. We fix a uniform bound k, and only consider words that can be decomposed into k sub-words, where each sub-word has at most one kind of return nodes. In other words, the automaton working over such a word proceeds in k phases, where in each phase it pops at most from one stack (but it can push onto all stacks). We dub these automata and the languages they accept as k-phase multi-stack visibly pushdown automata (languages).

The class of k-phase MVPLs (k-MVPLs) is a subclass of context-sensitive languages. It includes non-context-free languages such as $\{(a_1a_2)^n\, b_1^n\, b_2^n \mid n \in \mathbb{N}\}$, which can be accepted using only two phases, if a_1 and a_2 are calls corresponding to two different stacks, and b_1 and b_2 are the returns from these stacks. The MVPA accepting this language pushes the a_1's and a_2's onto the two stacks, and then when reading b_1's, pops symbols from the first stack, and then in the second phase, reading b_2's pops symbols from the second stack.

The main theorem of the paper is that the *emptiness* problem for MVPAs is decidable. The proof lies in a fairly complex encoding of k-phase words into trees, and hence reducing it to tree automata emptiness. As a consequence, we also obtain the result that for normal (not visible) multi-stack pushdown automata that use at most k-phases on any word, the emptiness problem is decidable.

We show that k-phase MVPLs are effectively closed under union, intersection, and complement (with respect to k-phase words), and using decidability of emptiness, it follows that the inclusion and equivalence problems are decidable for MVPLs. While closure of MVPLs under union and intersection are easy, closure under complement is more involved as MVPLs are *not* determinizable. The closure under complement is shown by mapping any MVPL into a regular language of trees, complementing the automaton with respect to the class of all trees that represent k-phase words, and translating the resulting automaton back to an MVPA.

Delving further into the language theory of this class, we show that it corresponds exactly to the class of monadic second-order logic on n-nested words with k phases, where the logic has n binary relations corresponding to the n nesting relations on the word.

In order to understand which subclass of context-sensitive languages MVPLs capture, we note that our mapping of k-phase words into trees essentially rearranges the letters in the word so that the language becomes tree-regular. We formalize this by proving a Parikh's theorem for MVPLs, namely that the commutative image of every MVPL is the commutative image of a regular language. This can be used to show that certain languages are not accepted by multi-stack automata; for example, we can show that $\{a^{2^n} \mid n \in \mathbb{N}\}$ is not accepted by *any* multi-stack automaton with a bounded number of phases (over the non-visible alphabet $\{a\}$), although it is a context-sensitive language.

One application of multi-stack automata with bounded phases is for the *model-checking of concurrent recursive programs*. When verifying concurrent programs, a common paradigm is to abstract the program into a finite model (say using predicate abstraction), and subject the model to algorithmic verification. The model however preserves the control flow accurately, and hence preserves recursion, leading to essentially a multi-stack pushdown automaton. Reachability checking for this model is of course undecidable. However, recent papers in verification make the observation that many errors are already found after a few number of *context-switches* (switches between stacks), and in fact for any k, checking if an error state is reached within k context switches is decidable [16]. Context-bounded checking of concurrent recursive programs has shown to discover several errors in programs (see [5, 16, 17]). MVPLs generalize this result by showing that checking whether an error state is reached within k *phases*, where in each phase *all* processes can evolve, but only one of them is allowed to return from procedures, is decidable. Technically, our result is more sophisticated as we can show that the set of configurations reached in k-phases is *not* regular. Other results on deciding reachability, including bounded-context switching and nested locks, crucially rely on the fact that the reachable sets are regular ([3, 4, 12, 16]). (In [4] the forward reachable sets are not regular, but backward reachable sets are.) We believe that analyzing k-phase executions of a program model explore a much larger space of configurations than k-context switches do, and may lead to more thorough model-checking algorithms.

The paper is structured as follows. Bounded phase multi-stack visibly pushdown languages are defined in Section 2. Section 3 is devoted to proving that the emptiness problem is decidable for MVPLs. Closure under boolean operations and non-determinazability of MVPLs are shown in Section 4, while decision problems are discussed in Section 5. In Section 6, we present a logical characterization of k-phase MVPLs and prove a Parikh theorem for them, and conclude with a few remarks in Section 7.

2. Preliminaries

Multi-Stack Visibly Pushdown Languages.
Given two positive integers i and j, $i \leq j$, we denote with $[i, j]$ the set of integers k with $i \leq k \leq j$, and denote by $[j]$ the set $[1, j]$. Let w be a word on some alphabet. We denote with $|w|$ the length of w, and hence $[|w|]$ denotes the set of positions of w. Given two positions j', j of w, $j' \leq j$, we denote with $w[j]$ the j-th symbol of w, and with $w[j', j]$ the sub-word of w formed from position j' to position j, both inclusive (when $j' > j$, $w[j', j] = \epsilon$).

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

A multi-stack visibly pushdown automaton over such an alphabet must push on the i'th stack exactly one symbol when it reads a call of the i'th alphabet, and pop exactly one symbol from the i'th stack when it reads a return of the i'th alphabet. Also, it cannot touch any stack when reading an internal letter.

Definition 1 *(*MULTI-STACK VISIBLY PUSHDOWN AUTOMATON*) A* multi-stack visibly pushdown automaton *(*MVPA*) over the n-stack call-return alphabet $\widetilde{\Sigma}_n = \langle\{(\Sigma_c^i, \Sigma_r^i)\}_{i\in[n]}, \Sigma_{int}\rangle$, is a tuple $M = (Q, Q_I, \Gamma, \delta, Q_F)$ where Q is a finite set of states, $Q_I \subseteq Q$ is the set of initial states, Γ is a finite stack alphabet that contains a special bottom-of-stack symbol $\perp$, $\delta \subseteq (Q \times \Sigma_c \times Q \times (\Gamma \setminus \{\perp\})) \cup (Q \times \Sigma_r \times \Gamma \times Q) \cup (Q \times \Sigma_{int} \times Q)$, and $Q_F \subseteq Q$ is the set of final states.*

Let us fix an n-stack alphabet $\widetilde{\Sigma}_n$ for the rest of the paper.

A transition (q, a, q', γ), where $a \in \Sigma_c^i$ and $\gamma \neq \perp$, is a push-transition with the meaning that on reading a, γ is pushed onto stack i and the control changes from state q to q'. Similarly, (q, a, γ, q') with $a \in \Sigma_r^i$ is a pop-transition where γ is read from the top of the stack i and popped (if the top of stack i is $\perp$, then it is read and not popped), and the control changes from q to q'.

A *stack* is a nonempty finite sequence over Γ where the bottom-of-stack symbol $\perp$ appears always and only in the end; let us denote the set of stacks as $St = (\Gamma \setminus \{\perp\})^* \cdot \{\perp\}$. A *configuration* of an MVPA M over $\widetilde{\Sigma}_n$ is a tuple $\langle q, \sigma_1, \ldots, \sigma_n\rangle$ with $q \in Q$ and each $\sigma_i \in St$. For a word $w = a_1 \ldots a_m$ in Σ^*, a run of M on w is a sequence of $m + 1$ configurations $\rho = \langle q_0, \sigma_1^0, \ldots, \sigma_n^0\rangle, \ldots, \langle q_m, \sigma_1^m, \ldots, \sigma_n^m\rangle$, where $q_0 \in Q_I$, $\sigma_i^0 = \perp$ for every $i \in [n]$, and for every $j \in [m]$ the following holds:

[Push] If $a_j \in \Sigma_c^i$ (i.e a_j is a call of stack i), then $\exists \gamma \in \Gamma$ such that $(q_j, a_j, q_{j+1}, \gamma) \in \delta$, $\sigma_i^{j+1} = \gamma \cdot \sigma_i^j$, and $\sigma_h^{j+1} = \sigma_h^j$ for every $h \in ([n] \setminus \{i\})$.

[Pop] If $a_j \in \Sigma_r^i$ (i.e. a_j is a return of stack i), then $\exists \gamma \in \Gamma$ such that $(q_j, a_j, \gamma, q_{j+1}) \in \delta$, $\sigma_h^{j+1} = \sigma_h^j$ for every $h \in ([n] \setminus \{i\})$, and either $\gamma \neq \perp$ and $\sigma_i^j = \gamma \cdot \sigma_i^{j+1}$, or $\sigma_i^j = \sigma_i^{j+1} = \perp$.

[Internal] If $a_j \in \Sigma_{int}$ is an internal action, then $(q_j, a_j, q_{j+1}) \in \delta$, and $\sigma_h^{j+1} = \sigma_h^j$ for every $h \in [n]$.

A run $\rho = \langle q_0, \sigma_1^0, \ldots, \sigma_n^0\rangle, \ldots, \langle q_m, \sigma_1^m, \ldots, \sigma_n^m\rangle$ is accepting if the state in the last configuration is final, i.e., if $q_m \in Q_F$. A word $w \in \Sigma^*$ is accepted by an MVPA M if there is an accepting run of M on w. The language of M, denoted $L(M)$, is the set of all words accepted by M.

Given a word $w \in \Sigma^*$, we denote with $Ret(w)$ the set of all returns in w. A word w is a *phase* if $Ret(w) \subseteq \Sigma_r^i$, for some $i \in [n]$, and we say that w is a *phase of stack* i. Let us now define k-phase words, which are words formed by concatenating at most k phases.

Definition 2 *(*PHASES*) For any k, a k-phase word [1] is a word $w \in \Sigma^+$ such that w can be factorized as $w = w_1 w_2 \ldots w_{k'}$ where $k' \leq k$ and w_h is a phase, for every $h \in [k']$. Such a factorization $w_1, w_2, \ldots w_{k'}$ is called a* k-factorization *of w w.r.t. $\widetilde{\Sigma}_n$. Let $Phases(\widetilde{\Sigma}_n, k)$ denote the set of all k-phase words over $\widetilde{\Sigma}_n$.*

Definition 3 *(*MULTI-STACK VISIBLY PUSHDOWN LANGUAGES WITH PHASES*) For any k, a* k-phase multi-stack visibly pushdown automaton *(*k-MVPA*) over $\widetilde{\Sigma}_n$ is a tuple $A = (k, Q, Q_I, \Gamma, \delta, Q_F)$ where $M = (Q, Q_I, \Gamma, \delta, Q_F)$ is an* MVPA *over $\widetilde{\Sigma}_n$. The language accepted by A is $L(A) = L(M) \cap Phases(\widetilde{\Sigma}_n, k)$. A language accepted by a* k-MVPA *is called a* k-phase multi-stack visibly pushdown language *(*k-MVPL*).*

Example 1 *Figure 1 gives a formal definition of a* 2-MVPA *A over $\widetilde{\Sigma}_2$ that accepts the language $\{(a_1a_2)^t\, b_1^t\, b_2^t | t \geq 0\}$. The automaton A checks whether the input word has the form $(a_1a_2)^*\, b_1^*\, b_2^*$ using its control states. For $i = 1, 2$: when A reads a call a_i, it pushes onto stack S_i symbol \$, if it is the first occurrence of a_i, and symbol #, otherwise; then, reading the return symbol b_i, A pops a symbol from stack S_i until symbol \$ is popped; when \$ is popped form stack S_2, it enters the accepting state.*

Multi-stack pushdown automata.
A *multi-stack pushdown automaton* (MPA) over a normal (non-visible) alphabet Σ is simply an n-stack automaton, with ϵ moves, that can push and pop from any stack reading any letter. Also, we define the k-phase version of these

[1] We ignore the empty word ϵ to simplify the presentation.

$\Sigma_c^1 = \{a_1\}, \Sigma_r^1 = \{b_1\}, \Sigma_c^2 = \{a_2\}, \Sigma_r^2 = \{b_2\}, \Sigma_{int} = \emptyset$; $A = (2, \{q_i | i \in [0,6]\}, \{q_0\}, \{\#, \$\}, \delta, \{q_0, q_6\})$

$$\begin{aligned}\delta \;:= \{\; & (q_0, a_1, q_1, \$), (q_1, a_2, q_2, \$), (q_2, a_1, q_3, \#),\\ & (q_2, b_1, \#, q_4), (q_2, b_1, \$, q_5), (q_3, a_2, q_2, \#),\\ & (q_4, b_1, \#, q_4), (q_4, b_1, \$, q_5), (q_5, b_2, \#, q_5),\\ & (q_5, b_2, \$, q_6)\}.\end{aligned}$$

Figure 1. A 2-MVPA for $\{(\mathbf{a_1a_2})^\mathbf{t}\, \mathbf{b_1^t}\, \mathbf{b_2^t} | \mathbf{t} \geq \mathbf{0}\}$.

(called k-MPAs). A k-MPA is an MPA that uses at most k-phases on any word (i.e. any run on any word can be decomposed into at most k phases, where in each phase the MPA pops at most from one stack). For example, the language $\{a^n b^n c^n \mid n \in \mathbb{N}\}$ is accepted by a 2-phase 2-stack MPA that pushes a symbol onto both stacks while reading a's (using ϵ-moves) and matches the b's against one stack and the c's with the other. It is easy to see that the language is not a k-MVPL for any partition of the letters a, b and c into call-return alphabets over multiple stacks.

Trees and Monadic Second-Order Logic.
We briefly recall trees and some well known results on trees and monadic second-order logic.

The trees we deal with are finite, binary, and labeled by a finite alphabet Υ. An Υ-*labeled tree* is a structure $T = (V, \lambda)$, where $V \subseteq \{0,1\}^*$ is a finite prefix-closed set, and $\lambda : V \to \Upsilon$ is a labeling function. The set V represents the nodes of T, where ϵ is the *root* of T, denoted $root(T)$. The edge-relation of T is implicit: edges are pairs $(v, v.i)$, where $v, v.i \in V, i \in \{0,1\}$. The node $v.0$ ($v.1$) is called the left-child (right-child) of v. Let $\mathcal{T}_\Upsilon$ denote the set of all Υ-labeled trees.

We assume standard nondeterministic top-down automata on trees [6].

Definition 4 *(*REGULAR TREES*) A set of Υ-labeled trees $\mathcal{L}$ is regular if there is some tree automaton $\mathcal{A}$ with $\mathcal{L} = \mathcal{L}(\mathcal{A})$.*

We will use the standard monadic-second order logic on trees, which allows quantification over nodes and sets of nodes of a tree, with interpreted relations for the left-child and right-child, and boolean operators [20].

The following theorem relates the class of regular trees and the class of trees definable by MSO.

Theorem 1 *([7, 18]) A tree language is regular iff it is MSO definable.*

3. The Emptiness Problem

Given a k-MVPA A over $\widetilde{\Sigma}_n$, the *emptiness problem* for A is to decide whether $L(A)$ is empty. In this section, we show that the emptiness problem for k-MVPLs is decidable in double exponential time by reducing it to the emptiness problem for finite tree automata.

We need some definitions. A word $w \in \Sigma^+$ is *i-well matched* if it is generated by the context-free grammar:

$$S \to S\,S \mid a\,S\,b \mid d \mid \epsilon$$

where S is the only variable, $a \in \Sigma_c^i, b \in \Sigma_r^i$, and $d \in \Sigma \setminus \Sigma^i$. Let $w \in \Sigma^+$ and j, j' be two positions of w with $j < j'$. The pair (j, j') is a *matching pair* of w iff there exists $i \in [n]$ such that $w[j] \in \Sigma_c^i$, $w[j'] \in \Sigma_r^i$, and $w[j+1, j'-1]$ is i-well matched.

A k-phase word can be factored in several ways; we fix a unique factorization that will simplify the technical exposition. We say that a k-factorization $w_1, \ldots, w_{k'}$ of a word w w.r.t. $\widetilde{\Sigma}_n$ is *tight* if: (1) the first symbol of w_h is a return, for every $h \in [2, k']$, (2) if $k' > 1$ then $Ret(w_1) \neq \emptyset$, and (3) w_h and w_{h+1} are phases of different stacks, for every $h \in [k'-1]$. It is easy to see that, for every word $w \in Phases(\widetilde{\Sigma}_n, k)$, there is a unique tight k-factorization of w w.r.t. $\widetilde{\Sigma}_n$.

Given a word $w \in Phases(\widetilde{\Sigma}_n, k)$, we define the map $phase_w : [|w|] \to [k]$ as follows: $phase_w(j) = h$ iff $w_1, \ldots, w_{k'}$ is the tight k-factorization of w w.r.t. $\widetilde{\Sigma}_n$ and j is a position of w_h.

Our proof is structured as follows. We first define a representation of k-phase words using trees labeled over the alphabet $(\Sigma \times [k])$. The vertices in the tree correspond to positions in the word such that the returns in the word are the right-children of the matching calls. While the call-return matching relation of the k-phase word is immediately recoverable in the tree, the positions in the word get arranged haphazardly in the tree, and retrieving the *linear* ordering of the word in the tree gets considerably complex. We exhibit how to recover the linear order $\prec$ of the word from the tree using MSO on trees. The correctness of this relation is not shown immediately. We first show that it defines *some* linear order on the vertices of any tree in which the phase numbers are monotonically non-decreasing along any path. Using this linear order, we characterize the set of stack trees: intuitively, the characterization looks at the word w formed using the linear order, and checks whether the nesting relations defined by the right-child relation agree with the call-return matching in w. We show that if a tree T satisfies the characterizing conditions of the stack tree, then the word w obtained is indeed such that its stack tree is T. This formally establishes the correctness of the linear order (Lemma 2).

Along with the above proofs, we establish that the complexity of checking the $\prec$ relation (using tree automata) is exponential, and the complexity of checking whether a tree is a stack-tree is double exponential in the number of phases. Finally, using the $\prec$-ordering, we show how to simulate an MVPA on the word obtained from the tree using a tree automaton of size double exponential in k. The complexity of checking emptiness of MVPA then follows from emptiness checking for tree automata.

The embedding of words into trees and the recovery of the linear relation forms the main technical crux of the paper, and is the hardest part of our proofs. The linear order is not recoverable unless the *phase number* of each letter is also encoded in the tree, and this is precisely why our embedding does not work for unboundedly many phases (understandably so, since unbounded number of phases leads to an undecidable emptiness problem).

Mapping nested structures into trees.
We define a function that maps every word $w \in \mathit{Phases}(\widetilde{\Sigma}_n, k)$ into a $(\Sigma \times [k])$-labeled tree that has a node for each position of w. The node corresponding to position j of w encodes in its label the symbol and the phase of position j in w. The root corresponds to position 1, and for all positions j, if (j', j) is a matching pair of w then the node corresponding to position j is the right-child of the node corresponding to position j'; otherwise, the node corresponding to position j is the left-child of the node corresponding to position $j-1$. The formal definition below simultaneously defines a 1-to-1 correspondence *pos* that relates positions in the word with the nodes in the tree.

Definition 5 (TREE REPRESENTATION) *For any word $w \in \mathit{Phases}(\widetilde{\Sigma}_n, k)$ with $|w| = m$, the word-to-tree map of w, $wt(w)$, which is a $(\Sigma \times [k])$-labeled tree (V, λ), and the bijection map $pos_w : V \to [m]$ are inductively (on $|w|$) defined as follows:*

- *If $m=1$, then $V=\{\epsilon\}$, $\lambda(\epsilon)=(w,1)$, and $pos_w(\epsilon)=1$.*
- *Otherwise, let $w' = w[1, m-1]$ and $wt(w') = (V', \lambda')$. Then:*
  - *$V = V' \cup \{v\}$ with $v \notin V'$.*
  - *$\lambda(v) = (w[m], phase_w(m))$;*
    $\lambda(v') = \lambda'(v')$, for every $v' \in V'$.
  - *If there is some $j \in [m-1]$ such that (j, m) is a matching pair in w, then v is the right-child of $pos^{-1}_{w'}(j)$.*
    Otherwise v is the left-child of $pos^{-1}_{w'}(m-1)$.
  - *$pos_w(v) = m$ and $pos_w(v') = pos_{w'}(v')$, for every $v' \in V'$.*

A tree T such that $T = wt(w)$ for some $w \in \mathit{Phases}(\widetilde{\Sigma}_n, k)$ is called a *k-stack tree*, and the set of k-stack trees is denoted by $\mathit{STree}(\widetilde{\Sigma}_n, k)$.

Example 2 *Consider $\Sigma^1_c = \{a\}, \Sigma^1_r = \{\bar{a}\}, \Sigma^2_c = \{c\}, \Sigma^2_r = \{\bar{c}\}$ and $\Sigma_{int} = \{e\}$, and $w = aeca\bar{a}\bar{a}\bar{a}ca\bar{c}e\bar{c}\bar{a}$. For $w_1 = aeca\bar{a}\bar{a}\bar{a}ca$, $w_2 = \bar{c}e\bar{c}$, and $w_3 = \bar{a}$, w_1, w_2, w_3 is a tight 3-factorization of w w.r.t. $\widetilde{\Sigma}_2$. Thus, $phase_w(j) = 1$ for every $j \in [9]$, $phase_w(j) = 2$ for every $j \in [10, 12]$, and $phase_w(13) = 3$. The set of all matching pairs of w w.r.t. $\widetilde{\Sigma}_2$ is $\{(1,6), (3,12), (4,5), (8,10), (9,13)\}$. Figure 2 shows the $(\Sigma \times [3])$-labeled tree $wt(w)$. Observe that, $pos(v_i) = i$, for every $i \in [13]$. Note that the linear ordering on the word is very non-local on the tree. For example, v_{12} is the successor of v_{11} and yet they are far away in the tree.*

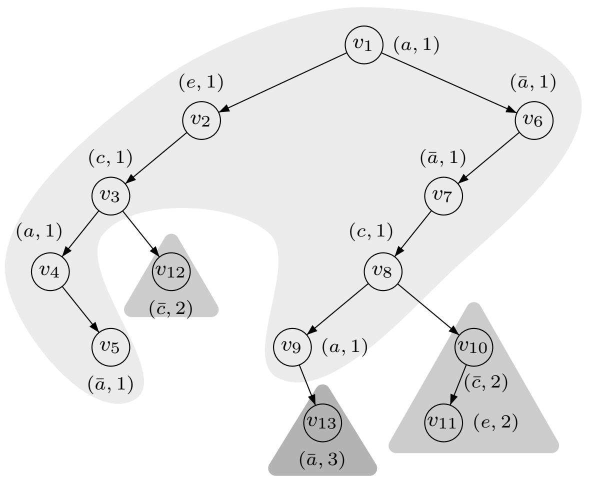

Figure 2. The tree $wt(\mathrm{aeca\bar{a}\bar{a}\bar{a}ca\bar{c}e\bar{c}\bar{a}})$.

Recovering the linear order $\prec$
In this section, we define a relation on nodes of $(\Sigma \times [k])$-labeled trees, denoted $\prec$, which (we will eventually show) recovers the linear order in w from the stack tree $wt(w)$.

Let $T = (V, \lambda)$ be a $(\Sigma \times [k])$-labeled tree. Given $x, y \in V$, we say that $x <_{\mathit{prefix}} y$ iff x precedes y in the prefix traversal of T. We define two maps, $sym : V \to \Sigma$ and $phase_T : V \to [k]$ as: $sym(x) = a$ and $phase_T(x) = h$ iff $\lambda(x) = (a, h)$. For a node $x \in V$, we denote with T_x the largest subtree of T containing x and such that all its nodes are labeled with the same phase number.

Definition 6 *Let $T = (V, \lambda)$ be a $(\Sigma \times [k])$-labeled tree with $phase_T(x) \geq phase_T(parent(x))$, for every $x \in V \setminus \{\epsilon\}$. For every $h \in [k]$, we inductively define the relations $<_h \subseteq V \times V$ and $<_{[h]} \subseteq V \times V$, as follows:*

- *$x <_h y$ iff $phase_T(x) = phase_T(y) = h$ and either (1) $T_x = T_y$ and $x <_{\mathit{prefix}} y$, or (2) $T_x \neq T_y$, $h > 1$ and $parent(root(T_y)) <_{[h-1]} parent(root(T_x))$.*
- *$x <_{[h]} y$ iff either (1) $phase_T(x), phase_T(y) < h$ and $x <_{[h-1]} y$, or (2) $phase_T(x) = phase_T(y) = h$ and $x <_h y$, or (3) $phase_T(x) < h$ and $phase_T(y) = h$.*

We define the relation $\prec$ as $<_{[k]}$.

Consider the tree T from Figure 2. On this tree, relation $<_1$, and thus $<_{[1]}$, simply corresponds to the ordering resulting from a prefix traversal of the subtree of all nodes of phase 1, i.e. the sequence $v_1, v_2, \ldots, v_9$. To determine relation $<_2$, observe that there are two separate subtrees which are formed by nodes of phase 2. Within each subtree the relation $<_2$ is defined according to the prefix traversal. To relate nodes of different subtrees we compare the parents of the roots of the respective subtrees. For example, consider nodes v_{11} and v_{12}. We have that v_{10} is the root of v_{11}'s subtree and v_{12} itself is the root of its subtree. Since $parent(v_{12}) <_{[1]} parent(v_{10})$ (i.e., $v_3 <_{[1]} v_8$), we can conclude that $v_{11} <_2 v_{12}$ (note that we have inverted the direction of the inequality moving from two nodes x, y of different subtrees to $parent(root(T_x)), parent(root(T_y))$). Thus, $<_{[2]}$ is given by the ordering $v_1, v_2, \ldots, v_{12}$. Since we only have one node of phase 3, the relation $<_{[3]}$, and thus $\prec$, is given by the linear ordering $v_1, v_2, \ldots, v_{13}$.

We now show that $\prec$ is a regular relation on the tree, i.e. expressible in MSO over the tree, using the inductive definition of $\prec$ given in Definition 6 above.

The MSO formula for the relation $\prec$, called $\psi_\prec$, is a formula with two free variables x and y, and holds precisely when $x \prec y$. We define it below using an inductive definition of formulas $less_{[h]}$

$$\begin{aligned}
less_{[h]}(x,y) := {} & LessPhase(x,y) \\
& \vee(InSamePhaseSubTree(x,y) \wedge less_{prefix}(x,y)) \\
& \vee(\neg LessPhase(x,y) \wedge \neg InSamePhaseSubTree(x,y) \\
& \quad \wedge \exists z \exists z' (ParentRootInPhaseSubTree(z,x) \\
& \qquad \wedge ParentRootInPhaseSubTree(z',y) \\
& \qquad \wedge less_{[h-1]}(z',z)))
\end{aligned}$$

$$\psi_\prec(x,y) = less_{[k]}(x,y)$$

with the understanding that $less_0(x,y)$ is `false`.

In the above, the auxiliary subformulas are defined as follows: $LessPhase(x,y)$ holds iff the phase of x (read from its label) is less than the phase of y; $InSamePhaseSubTree(x,y)$ holds if x and y are of the same phase and belong to the same-phase subtree (i.e. $T_x = T_y$); $less_{prefix}(x,y)$ holds iff x occurs before y in the prefix-order of the tree; $ParentRootInPhaseSubTree(z,x)$ holds iff z is the parent of the root of the subtree T_x.

A brief note on complexity. The auxiliary sub-formulas above can be written easily in MSO using only disjunctions and existential quantifications, and a number of conjunctions that is only polynomially many (in k, n, and Σ) atomic formulas. Consequently, they can be implemented using nondeterministic tree automata with a polynomial number of states. The nondeterministic tree automaton finally obtained for $\psi_\prec$ is however exponential, because of the two conjunctions before the recursive application to $less_{[h-1]}$. These conjunctions contribute a series of $2k$ conjunctions, and causes the automaton to be exponential in k ($2^{O(k \log k)}$).

Though we cannot establish the correctness of $\prec$ yet, we can show now that $\prec$ always defines *some* linear order on a tree, provided phase numbers are monotonically non-decreasing along any path in the tree. Intuitively, $<_{prefix}$ linearly orders all elements of a subtree that corresponds to the same phase, elements of two different phases get ordered uniquely, and nodes of the same phase but in different subtrees get ordered according to the reverse ordering of the nodes where their subtrees hang.

Lemma 1 *Let $T = (V, \lambda)$ be a $(\Sigma \times [k])$-labeled tree with $phase_T(x) \geq phase_T(parent(x))$, for every $x \in (V \setminus \{\epsilon\})$. Then the relation $\prec$ is a linear ordering on V.*

Regularity of stack trees.

All trees obviously do not correspond to stack trees of words. In this section, we establish the regularity of the set of all stack trees. We give first a characterization of stack trees, and then argue that it can be expressed in MSO. Intuitively, if a tree T is a stack tree, then we should be able to recover the word w corresponding to it using the $\prec$ relation on the tree, and we expect that the right-child relation in the tree captures the nesting edges between calls and returns in w. Apart from other checks on w, we would need to make sure that w with the phase numbers inherited from corresponding vertices in T gives a tight factorization. Let T-*tight* denote the (regular) set of words over $(\Sigma \times [k])$ that correspond to a tight k-factorization.

Lemma 2 *Let $T = (V, \lambda)$ be a $(\Sigma \times [k])$-labeled tree. Then $T \in STree(\widetilde{\Sigma}_n, k)$ iff the following hold:*

1. *Phase numbers are monotonically non-decreasing along any path, i.e. $phase_T(x) \geq phase_T(parent(x))$, for every $x \in (V \setminus \{\epsilon\})$.*
2. *Right-children are always returns and their parent is always a call: i.e. if y is a right-child of x, then y is a return and x is a call of the same stack as y.*
3. *Unmatched calls cannot be followed by unmatched returns (from the same stack):*
   i.e. there is no x, y such that $x \prec y$, $\lambda(x)$ is a call of stack i, y is a return of stack i, x does not have a right-child, and y is not a right-child of any node.
4. *Nesting relations are proper: Let y be a right-child of x and z be a node between x and y in the $\prec$ order.*
   - *If z is a call of the same stack as x (and y), then z must have a right-child which is between z_1 and y in the $\prec$ order.*
   - *Analogously, if z is a return of the same stack as x and y, then z must be the right-child of a node that is between x and z in the $\prec$ order.*
5. *The word formed according to the $\prec$ order is T-tight.*

Furthermore, if a tree T satisfies the conditions above, then the word w obtained using the $\prec$ order is such that the stack-tree of w is T.

Let tw denote the function that maps any stack tree T to the word w obtained using the $\prec$ ordering. The above lemma shows that if $tw(T) = w$, then $wt(w) = T$, i.e. wt and tw are inverses of each other. This establishes that $\prec$ is indeed the correct linear order on stack trees.

We can now show the regularity of $STree(\widetilde{\Sigma}_n, k)$.

Theorem 2 *The set $STree(\widetilde{\Sigma}_n, k)$ is MSO definable. Moreover, there is a tree automaton with a number of states double exponential in k, that accepts $STree(\widetilde{\Sigma}_n, k)$.*

Proof The proof proceeds by expressing the characterizing properties of stack-trees in MSO. Properties (1),(2),(3) and (5) are easy to express. The first property in (4) (the second property is similar) can be expressed as:

$$\begin{aligned}\neg\ \exists x, y, z_1, z_2 :\ & \mathit{RightChild}(x, y) \wedge \mathit{IsCall}(z_1) \\ & \wedge \psi_{\prec}(x, z_1) \wedge \psi_{\prec}(z_1, y) \\ & \wedge [(\neg\ \exists z_3.\mathit{RightChild}(z_1, z_3)\ ) \vee \\ & (\mathit{RightChild}(z_1, z_2) \wedge (\psi_{\prec}(z_2, z_1) \vee \psi_{\prec}(y, z_2)\ )\ )]\end{aligned}$$

Since $x \prec y$ takes an exponential-sized automaton to check, the above checks can be effected by a tree automaton of size double exponential in k. □

Solving emptiness

We are now ready to prove that for any class of words accepted by a k-MVPA, the class of trees corresponding to them forms a regular tree language.

Theorem 3 *If L is a k-MVPL, then $wt(L)$ is regular. Moreover, if A is a k-MVPA accepting L, then there is a tree automaton that accepts $wt(L)$ with number of states at most exponential in the size of A and double exponential in the number of phases k (more precisely, $exp(|A| 2^{O(k log k)})$ states).*

Proof The MSO sentence φ defining $wt(L)$ consists of a conjunction of two MSO sentences φ_1 and φ_2, where φ_1 enforces the trees T to belong to $STree(\widetilde{\Sigma}_n, k)$ and φ_2 guaranties that $tw(T)$ is a word of L. By Theorem 2, we just need to show φ_2.

For a k-MVPA A, φ_2 simply guesses the transitions that A takes along an accepting run. Let $\delta = \{\delta_1, \ldots, \delta_t\}$ be the set of A transitions. Then, φ_2 is of the form

$$\exists Y_1 \ldots \exists Y_t\ \ (\mathit{unique} \wedge \mathit{init} \wedge \mathit{trans} \wedge \mathit{acc} \wedge \mathit{nonempty}).$$

We use the variable Y_i to guess all tree nodes where transition δ_i is taken (along a run). Checking whether such a guess corresponds to a valid run is along standard lines (see [20] for similar proofs). The formula unique checks that each node is associated with exactly one transition, init checks that the transition labeling the root starts from an initial state, and acc ensures that the last state on the run is a final state. The formula trans is more interesting. It verifies the validity of each transition in the run by checking that: (i) at each node labeled by the symbol a is associated a transition on a, (ii) when a symbol is pushed onto the stack then at the corresponding return (if it occurs) the same symbol is popped (this can be easily accomplished as the matching return for a call is its right-child in any stack tree), and (iii) for each pair of nodes x, y of $wt(w)$ corresponding to two consecutive symbols of w, the target of the transition taken at x is the source of the transition taken at y. This last property can be stated as:

$$\neg\exists x \exists y \left(\mathit{succ}(x, y) \wedge \neg \bigvee_{(i,j)\in I} (x \in Y_i \wedge y \in Y_j) \right),$$

where I is the set of all pairs (i, j) such that the target of δ_i coincides with the source of δ_j. Expressing $\mathit{succ}(x, y)$ using $\psi_{\prec}$ would lead to an extra exponential blow-up than needed. However, we can show that there is a nondeterministic automata of size double exponential in k that traverses the tree according to the linear ordering and checks if all the state transitions are correct.

The formula φ_2 can be translated to a corresponding tree automaton of size double exponential in k and exponential in the size of A (see [20]), and hence also the automaton for $\varphi_1 \wedge \varphi_2$. □

We can now show the main result of this section, which follows from the above theorem and the fact that tree automata emptiness is solvable in linear time [6].

Theorem 4 *(EMPTINESS FOR k-MVPLS) The emptiness problem for k-MVPLs is decidable in double exponential time.*

We can generalize the emptiness result to k-phase multi-stack automata over non-visible alphabets as well:

Theorem 5 *(EMPTINESS OF k-MPAS) The emptiness problem is decidable in double-exponential time for k-MPAs.*

Proof Consider the runs of the multi-stack automaton, by taking each transition as a new letter, and associating with it a type as to whether it is a call/return/internal depending on what it does to the stack. The set of runs is accepted by a multi-stack visibly pushdown automaton with k-phases, and the original automaton accepts some word iff the latter does. Hence emptiness can be decided using Theorem 4. □

4. Closure Properties

We now show closure properties of k-MVPLS. We start by defining a renaming operation.

A renaming of $\widetilde{\Sigma}_n$ to $\widetilde{\Omega}_n$ is a function $f : \Sigma \to \Omega$ such that $f(\Sigma^i_c) \subseteq \Omega^i_c$, $f(\Sigma^i_r) \subseteq \Omega^i_r$, for every $i \in [k]$, and $f(\Sigma_{int}) \subseteq \Omega_{int}$. A renaming f is extended to words over Σ in the natural way: $f(a_1 \dots a_m) = f(a_1) \dots f(a_m)$.

Theorem 6 (CLOSURE) *Let L_1 and L_2 be two k-MVPLs over $\widetilde{\Sigma}_n$. Then, $L_1 \cup L_2$ and $L_1 \cap L_2$ are k-MVPLs over $\widetilde{\Sigma}_n$. Moreover, if f is a renaming of $\widetilde{\Sigma}_n$ to $\widetilde{\Omega}_n$, then $f(L_1)$ is also a k-MVPL over $\widetilde{\Omega}_n$.*

Proof Let A_1, A_2 be two k-MVPAs such that $L(A_1) = L_1$ and $L(A_2) = L_2$.

Closure under union follows by taking the union of the state and transitions of A_1 and A_2 (assuming they are disjoint) and taking the new set of initial states (final states) to be the union of the initial states (final states) of A_1 and A_2.

$L_1 \cap L_2$ can be accepted by an MVPA A that has as its set of states the product of the states of A_1 and A_2, and as its stack alphabet the product of the stack alphabets of A_1 and A_2. When reading a call of stack i, if A_1 pushes γ_1 and A_2 pushes γ_2, respectively on their i-th stack, then A pushes (γ_1, γ_2) onto its i-th stack. The set of initial (final) states is the product of the initial (final) states of A_1 and A_2.

Given L_1 accepted by the k-MVPA A and a renaming f, $f(L_1)$ is accepted by the k-MVPA obtained from A by transforming each transition on a to a transition on $f(a)$. $\square$

An MVPA $M = (Q, Q_I, \Gamma, \delta, Q_F)$ is *deterministic* if $|Q_I| = 1$, and $|\{(q, a, q') \in \delta\} \cup \{(q, a, q', \gamma) \in \delta\} \cup \{(q, a, \gamma, q') \in \delta\}| = 1$, for every $q \in Q$ and $a \in \Sigma$. A k-MVPA $A = (k, Q, Q_I, \Gamma, \delta, Q_F)$ is deterministic if the MVPA $(Q, Q_I, \Gamma, \delta, Q_F)$ is deterministic. MVPAs *cannot* be determinized. In fact we can show that for $\widetilde{\Sigma}_2 = (\Sigma^1_c, \Sigma^1_r, \Sigma^2_c, \Sigma^2_r, \emptyset)$, with $\Sigma^1_c = \{a\}$, $\Sigma^1_r = \{c, d\}, \Sigma^2_c = \{b\}, \Sigma^2_r = \{x, y\}$, the language $L = \{(ab)^i c^j d^{i-j} x^j y^{i-j} | i \in \mathbb{N}, j \in [i]\}$ is accepted by a non-deterministic 2-MVPA but not by any deterministic k-MVPA.

Theorem 7 (NON-DETERMINIZABILITY) *The class of k-MVPLs is not closed under dertermінization.*

Let L be a k-MVPL over $\widetilde{\Sigma}_n$; the *complement* of L is defined with respect to all k-phase words: $\overline{L} = (\Sigma^* \setminus L) \cap Phases(\widetilde{\Sigma}_n, k)$. We show that though k-MVPLs cannot be determinized, they can be complemented. First we give a technical result for translating any regular language of stack trees to an MVPA that accepts the corresponding words.

Lemma 3 *If $\mathcal{L} \subseteq STree(\widetilde{\Sigma}_n, k)$ is regular, then $tw(\mathcal{L})$ is a k-MVPL. Moreover, if $\mathcal{L}$ is accepted by an automaton of size s, then the corresponding k-MVPA is of size polynomial in s.*

We can now show complementation of k-MVPAs:

Theorem 8 (COMPLEMENTABILITY) *The class of k-MVPLs is closed under complementation. That is, if L is a k-MVPL over $\widetilde{\Sigma}_n$, then $\overline{L}$ is also a k-MVPL over $\widetilde{\Sigma}_n$.*

Proof By Theorem 3, the tree set $wt(L)$ is regular. Since tree automata are closed under complementation, $\overline{wt(L)}$ is also regular (see [19]), and by Theorem 1 there is an MSO sentence φ defining it. Now, if ψ is an MSO formula defining $STree(\widetilde{\Sigma}_n, k)$ (the existence of ψ is guaranteed by Theorem 2), then it is simple to show that the MSO sentence $\varphi \wedge \psi$ defines the set of trees $wt(\overline{L})$. To conclude the proof, observe that by Lemma 3 $tw(wt(\overline{L}))$ is a k-MVPL, and by Lemma 2, $tw(wt((\overline{L})) = \overline{L}$. $\square$

The following table summarizes and compares closure properties for CSLs, CFLs, VPLs, MVPLs and regular languages (see [11]).

	Closure properties			
	$\cup$	$\cap$	Complement	Determ.
Regular	Yes	Yes	Yes	Yes
VPL	Yes	Yes	Yes	Yes
CFL	Yes	No	No	No
CSL	Yes	Yes	Yes	Not known
MVPL	**Yes**	**Yes**	**Yes**	**No**

5. Decision Problems

The *membership problem* for k-MVPAs is to check, for any fixed k-MVPA A over $\widetilde{\Sigma}_n$, whether a given word $w \in \Sigma^*$ is accepted by A.

Theorem 9 (MEMBERSHIP) *The membership problem for k-MVPLs is NP-complete.*

Proof Showing membership in NP is trivial. A nondeterministic polynomial-time algorithm just need to guess a run and simulate it on a given input word. To show NP-hardness we reduce satisfiability of boolean formulae. Fix a formula ϕ over variables $\{x_1, \dots, x_k\}$. We define a 2-stack k-MVPA A that accepts a particular word in $Phases(\widetilde{\Sigma}_2, k)$ iff ϕ holds. Automaton A, starts in phase 1 reading the formula from the tape and storing it in stack 1. Then, in phase h, $h \geq 1$, A guesses a truth value for variable x_h, pops the content of the stack of the current phase and pushes it onto the other stack rewriting each occurrence of x_h with the guessed value. Finally, in phase k, while rewriting the stack content, A also evaluates the obtained expression (all variables have been substituted with a truth value) and thus accepts iff it evaluates to true. We remark that the $O(|\phi|.k)$ operations concerning the stacks are driven by the input word to the MVPA. $\square$

The *universality problem* for k-MVPLs is to check whether a given k-MVPA accepts all the strings of

$Phases(\widetilde{\Sigma}_n, k)$. The *inclusion problem* is to find whether, given two k-MVPAs A_1 and A_2 over $\widetilde{\Sigma}_n$, $L(A_1) \subseteq L(A_2)$.

Theorem 10 (UNIVERSALITY, INCLUSION) *The universality and the inclusion problems are decidable.*

Proof Using closure of k-MVPAs under complement, it follows that universality and inclusion of k-MVPAs reduce to the emptiness problem, and thus are decidable. □

We can show an EXPTIME lower bound for the emptiness problem and a 2EXPTIME lower bound for the universality and inclusion problems. We conjecture that emptiness is complete for double exponential time.

The following table summarizes the results we have shown on the complexity of the main decision problems for MVPLs, and recalls known results for CSLs, CFLs, VPLs and regular languages. In this table, NLOG stands for NLOG-complete, and so on for the other complexity classes.

	Decision Problems		
	Membership	Emptiness	Univ./ Equiv./Incl.
Reg.	NLOG	NLOG	PSPACE
VPL	PTIME	PTIME	EXPTIME
CFL	PTIME	PTIME	Undecidable
CSL	NLINSPACE	Undecidable	Undecidable
MVPL	**NP**	**IN 2EXPTIME** **EXPTIME-HARD**	**IN 3EXPTIME** **2EXPTIME-HARD**

6. Language Theoretic Properties

6.1. A Logical Characterization

In this section, we define a logic on words over an n-stack call-return alphabet $\widetilde{\Sigma}_n$ which has in its signature relations that capture the n nesting relations.

In this context, a word w over Σ is a structure over the universe $U = \{1, \ldots, |w|\}$, the set of positions in w. We use unary predicates $P_a(i)$, for $a \in \Sigma$, which stand for $w[i] = a$. Also, we have n binary relations μ_j ($j \in [n]$) over U, where μ_j corresponds to the matching relation of calls and returns according to the j'th nesting relation. Let us fix a countable infinite set of first-order variables $x, y, \ldots$ and a countable infinite set of monadic second-order (set) variables $X, Y, \ldots$.

The *monadic second-order logic* (MSO$_\mu$) over $\widetilde{\Sigma}_n$ is defined as:

$\varphi := P_a(x) | x \in X | x \leq y | \mu_j(x, y) | \neg\varphi | \varphi \vee \varphi | \exists x \varphi | \exists X \varphi$

where $j \in [n]$, $a \in \Sigma$, x, y are a first-order variables and X is a set variable.

The models are words over Σ. The semantics is the natural semantics on the structure for words defined above. The first order variables are interpreted over the positions of w, and the second-order variables range over sets of positions. We recall that a sentence is a formula which has no free variables. The set of all words of $Phases(\widetilde{\Sigma}_n, k)$ that satisfy a sentence φ is denoted $L_k(\varphi)$ and we say φ defines this language. Using standard techniques to convert MSO to automata (given that the automata are closed under boolean operations and projection), we get (see [20]):

Theorem 11 *A language L is a k-MVPL over $\widetilde{\Sigma}_n$ iff there is an MSO$_\mu$ sentence φ over $\widetilde{\Sigma}_n$ with $L_k(\varphi) = L$.*

6.2. A Parikh Theorem

The Parikh mapping, introduced by Parikh [15], associates a word with the vector of natural numbers that reflect the number of occurrences of the symbols in the word. Formally, the Parikh image of a word w, over the alphabet $\{a_1, \ldots, a_\ell\}$, denoted by $\Phi(w)$, is the tuple $\Phi(w) = (\#a_1, \ldots, \#a_\ell)$ where $\#a_i$ is the number of occurrences of the symbol a_i in w. We extend the Parikh image to languages in the natural way: $\Phi(L) = \{\Phi(w) | w \in L\}$. Parikh's theorem [15] states that for each context-free language L over Σ there is a regular language L' over Σ such that the Parikh image of L and L' coincide, that is, $\Phi(L) = \Phi(L')$. For example, the language $L = \{a^i b^i | i \in \mathbb{N}\}$ has the same Parikh image of the regular language $L' = (ab)^*$, i.e. $\Phi(L) = \Phi(L') = \{(i, i) | i \in \mathbb{N}\}$. Moreover, given a context-free language L, it is *effective* to find a regular language L' such $\Phi(L) = \Phi(L')$; that is, there is an algorithm that takes as input L and gives as output L'. The next theorem shows that the same properties also hold for k-MVPLs.

Theorem 12 *For every k-MVPL L over $\widetilde{\Sigma}_n$, there exists a regular language L' over Σ such that $\Phi(L') = \Phi(L)$. Moreover, L' can be effectively computed.*

Proof Recall that for any context-free grammar G, we can construct a right-linear grammar G' such that $\Phi(L(G)) = \Phi(L(G'))$ [15]. Thus, for any k-MVPL L, we just need to construct a context-free grammar G such that $\Phi(L) = \Phi(L(G))$. From Theorem 3, we can construct a tree automaton $\mathcal{A}$ accepting $wt(L)$. Viewing the transition rules of a tree automaton as productions of a context free grammar, we can construct starting from $\mathcal{A}$ a context free grammar that generates a language L' such that $\Phi(L) = \Phi(L')$. □

The above result can be used to show that certain languages are not recognized by k-MVPAs or even k-phase MPAs. For instance, consider the language over the (non-visible) alphabet $\Sigma = \{a\}$, $L = \{a^{2^n} \mid n \in \mathbb{N}\}$. We can show that L is not accepted by any k-phase n-stack MPA. Assume by contradiction that it is accepted by a k-phase n-stacks MPA. Let us first change the MPA so that it does stack moves only on ϵ-transitions. Then, let's augment the alphabet with symbols $\{(c, i), (r, i) \mid i \leq n\} \cup \{int\}$, and let us transform the MPA by relabeling each ϵ push-transition onto stack i to (c, i), each ϵ pop-transition from stack i to (r, i) and each ϵ transition that doesn't touch the stack to int. The resulting MPA is an MVPA whose language projected to $\{a\}$ results in L. But by the Parikh theorem above, the Parikh image of the language accepted by the MVPA is equivalent to the image of a regular language R. Projecting R to $\{a\}$ is L and must be regular, which is a contradiction.

7. Discussion

We have defined and studied a robust and tractable subclass of context-sensitive languages. There are various other extensions of context-free languages that have been shown to be tractable. Bounded context-switching reachability of multistack automata [16] and concurrent pushdown automata interacting using nested locks [12] are known to admit decidable reachability problems, but crucially rely on the fact that the set of reachable configurations of such automata are *regular*. MVPAs with bounded phases strictly generalize both these classes of languages, and the reachability algorithm is more sophisticated. The set of reachable configurations of k-phase MVPAs are, for example, not regular or even context-free. For instance, consider the deterministic 2-phase MVPA that reads $(abc)^n.(\bar{b}a)^n.(\bar{c}a)^n$, pushes the symbols a, b and c onto three different stacks, and then transfers the b's and c's back to the first stack reading $\bar{b}a$ and $\bar{c}a$ repeatedly. The resulting set of configurations has $a^n b^n c^n$ on the first stack with the other two stacks empty, which is a non-context-free set of configurations.

Other extensions of context-free languages include the class of languages accepted by *stack automata* [9] and those accepted by higher-order pushdown automata [13]. While these classes have a decidable emptiness problem, they are not robust (for example, they are not closed under complement). We do not know whether k-MVPLs are contained in these classes, but we conjecture that they indeed are.

Several future directions are interesting. First, the class of multiple nested word languages with a bounded number of phases is of bounded tree-width (this is the property that allows us to embed them in trees). It would be interesting to characterize naturally the exact class of multiple nested words that have bounded tree-width. Secondly, we believe that our results have applications to other areas in verification, for instance in checking parallel programs that communicate with each other using unbounded FIFO queues, as multiple stacks can be used to simulate queues. Thirdly, propositional dynamic logic (PDL) has been shown to be decidable when regular expressions in the logic are replaced with particular classes of non-regular languages. The non-regular but context-free language extensions that were known to be decidable [10] were recently generalized in [14], where it was shown that PDL with visibly-pushdown language modalities is decidable. Whether one can obtain a generalization of the small number of decidable extensions of PDL with non-context-free languages, using the class of automata presented in this paper, is an interesting open problem.

References

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

[2] R. Alur and P. Madhusudan. Adding nesting structure to words. In *DLT*, LNCS 4036, pages 1–13, 2006.

[3] A. Bouajjani, J. Esparza, S. Schwoon, and J. Strejcek. Reachability analysis of multithreaded software with asynchronous communication. In *FSTTCS*, LNCS 3821, pages 348–359, 2005.

[4] A. Bouajjani, M. Müller-Olm, and T. Touili. Regular symbolic analysis of dynamic networks of pushdown systems. In *CONCUR*, LNCS 3653, pages 473–487, 2005.

[5] S. Chaki, E. M. Clarke, N. Kidd, T. W. Reps, and T. Touili. Verifying concurrent message-passing C programs with recursive calls. In *TACAS*, LNCS 3920, pages 334–349, 2006.

[6] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi. Tree automata techniques and applications, 1997.

[7] J. Doner. Tree acceptors and some of their applications. *J. Comput. Syst. Sci.*, 4(5):406–451, 1970.

[8] E. A. Emerson. Temporal and modal logic. In [21], pages 995–1072.

[9] S. Ginsburg, S. A. Greibach, and M. A. Harrison. One-way stack automata. *J. ACM*, 14(2):389–418, 1967.

[10] D. Harel and D. Raz. Deciding properties of nonregular programs. *SIAM J. Comput.*, 22(4):857–874, 1993.

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

[12] V. Kahlon, F. Ivancic, and A. Gupta. Reasoning about threads communicating via locks. In *CAV*, LNCS 3576, pages 505–518, 2005.

[13] T. Knapik, D. Niwinski, and P. Urzyczyn. Higher-order pushdown trees are easy. In *FoSSaCS*, LNCS 2303, pages 205–222, 2002.

[14] C. Löding and O. Serre. Propositional dynamic logic with recursive programs. In *FoSSaCS*, LNCS 3921, pages 292–306, 2006.

[15] R. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.

[16] S. Qadeer and J. Rehof. Context-bounded model checking of concurrent software. In *TACAS*, LNCS 3440, pages 93–107, 2005.

[17] S. Qadeer and D. Wu. Kiss: keep it simple and sequential. In *PLDI*, pages 14–24. ACM, 2004.

[18] J. W. Thatcher and J. B. Wright. Generalized finite automata theory with an application to a decision problem of second-order logic. *Math. Sys. Theory*, 2(1):57–81, 1968.

[19] W. Thomas. Automata on infinite objects. In [21], pages 133–192.

[20] W. Thomas. Languages, automata, and logic. In *Handbook of Formal Languages, Volume 3*, pages 389–455. Springer-Verlag New York, Inc., New York, NY, USA, 1997.

[21] J. van Leeuwen, editor. *Handbook of Theoretical Computer Science, Volume B*. Elsevier and MIT Press, 1990.

[22] M. Y. Vardi and P. Wolper. Automata-theoretic techniques for modal logics of programs. *J. Comput. Syst. Sci.*, 32(2):183–221, 1986.