Descriptional complexity of unambiguous input-driven pushdown automata[☆]

Alexander Okhotin[a,1], Kai Salomaa[b,2]

[a] *Department of Mathematics, University of Turku, 20014 Turku, Finland*
[b] *School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada*

Abstract

It is known that a nondeterministic input-driven pushdown automaton (IDPDA) (a.k.a. visibly pushdown automaton; a.k.a. nested word automaton) of size n can be transformed to an equivalent deterministic automaton of size $2^{\Theta(n^2)}$ (B. von Braunmühl, R. Verbeek, "Input-driven languages are recognized in $\log n$ space", FCT 1983), and that this size is necessary in the worst case (R. Alur, P. Madhusudan, "Adding nesting structure to words", J.ACM, 2009). This paper demonstrates that the same worst-case $2^{\Theta(n^2)}$ size blow-up occurs when converting a nondeterministic IDPDA to an unambiguous one, and an unambiguous IDPDA to a deterministic one. In addition, the methods developed in this paper are used to demonstrate that the descriptional complexity of complementation for nondeterministic IDPDAs is $2^{\Theta(n^2)}$, and that the descriptional complexity of homomorphisms for deterministic IDPDAs is $2^{\Theta(n^2)}$ as well.

Keywords: nondeterminism, unambiguity, input-driven pushdown, visibly pushdown automata, nested word automata, descriptional complexity

1. Introduction

An input-driven pushdown automaton, first considered by Mehlhorn [23] in 1980, has an input alphabet split into three classes, and the type of the current symbol determines whether the automaton must push onto the stack, pop from the stack, or ignore the stack. Mehlhorn [23] showed that the languages recognized by such automata, called *input-driven languages*, have space complexity $O\big(\frac{\log^2 n}{\log\log n}\big)$. This bound was further improved to $O(\log n)$ by von Braunmühl and Verbeek [8], and later Rytter [32] obtained a different algorithm with the same space requirements. Input-driven languages were proved to be in NC^1 by Dymond [10]. Von Braunmühl and Verbeek [8] have also demonstrated that the nondeterministic variant of the model is equal in power to the deterministic one.

Input-driven automata were rediscovered and further studied by Alur and Madhusudan [3] in 2004 under the name of *visibly pushdown automata*. In particular, they showed that a deterministic automaton simulating a nondeterministic automaton with n states and stack symbols needs in the worst case $2^{\Theta(n^2)}$ states, and that the class of input-driven languages has strong closure properties. Various aspects of visibly pushdown automata were researched in further papers [2, 3, 7, 9, 25]. Later, Alur and Madhusudan [4] suggested to regard this model as automata operating on *nested words*, which provide a natural data model

[☆] Appeared in *Theoret. Comput Sci.* doi.org/10.1016/j.tcs.2014.11.015 (2015). A preliminary version of this paper, entitled "Descriptional complexity of unambiguous nested word automata", was presented at the Languages and Automata Theory and Applications (LATA 2011) conference held in Tarragona, Spain, May 26–31, 2011, and its extended abstract appeared in the conference proceedings.

Email addresses: alexander.okhotin@utu.fi (Alexander Okhotin), ksalomaa@cs.queensu.ca (Kai Salomaa)

[1] Supported by the Academy of Finland under grants 134860 and 257857.
[2] Supported by the Natural Sciences and Engineering Research Council of Canada.

for such applications as XML document processing, where the data has a dual linear-hierarchical structure [1, 5]. The nested word automaton model has been studied in a number of papers [1, 5, 14, 29, 33].

Yet another mathematically equivalent model are the *pushdown forest automata* of Neumann and Seidl [24]. A pushdown forest automaton is a tree automaton that traverses a tree in depth-first left-to-right order and is equipped with a stack, which is manipulated as follows: whenever the machine goes down to the leftmost child, it pushes a symbol onto the stack, and as it returns from the rightmost child, it pops a symbol off the stack. The class of tree languages recognized by pushdown forest automata coincides with the regular tree languages. Pushdown forest automata were found to be equivalent to input-driven pushdown automata by Gauwin et al. [11].

Of all the different terminology for this machine model, this paper sticks to the original name of *input-driven pushdown automata* (IDPDA) and to their pushdown automata semantics. Though the name "visibly pushdown automaton" has been more widespread in the recent literature, the authors believe that the original name better describes this model: for example, Bollig [7] uses the term "input-driven" to describe the operation of these automata, in spite of being unaware of their original name. In the following, when citing results from papers using any equivalent automaton model, the terminology shall be translated to that of input-driven pushdown automata without separate mention.

It is known from Alur and Madhusudan [4] that a deterministic input-driven pushdown automaton (DIDPDA) equivalent to a nondeterministic one (NIDPDA) with n states and stack symbols needs in the worst case $2^{\Theta(n^2)}$ states. Later Okhotin et al. [27] refined the result by giving a lower bound construction that is tight within a multiplicative constant both for the number of states and the number of stack symbols. A similar result for finite automata, that an n-state nondeterministic automaton (NFA) requires up to exactly 2^n states in a deterministic one (DFA), is well-known, and there exists a vast literature on descriptional complexity of different kinds of finite automata [16]. For instance, precise succinctness tradeoffs between two-way and one-way finite automata were determined by Kapoutsis [19] for an unbounded alphabet, and by Kunc and Okhotin [20] in the case of a one-letter alphabet. There exists an important intermediate class of finite automata located between NFAs and DFAs: the *unambiguous finite automata* (UFA), which are NFAs, where every accepted string must have a unique accepting computation. The first state complexity studies on UFAs [30, 34] and on automata employing different degrees of ambiguity [12, 21] led to the following tradeoffs. Leung [22] proved that simulating an n-state UFA requires, in the worst case, a DFA with 2^n states, while the NFA-to-UFA tradeoff is $2^n - 1$. In the case of a one-letter alphabet, the UFA-to-DFA and NFA-to-UFA tradeoffs are known from Okhotin [26], and are estimated as $e^{\Theta(\sqrt[3]{n \ln^2 n})}$ and $e^{(1+o(1))\sqrt{n \ln n}}$, respectively.

An unambiguous subclass of nondeterministic input-driven pushdown automata would be interesting to investigate for the following two reasons. First, understanding the power of unambiguous nondeterminism, as an intermediate mode of computation between determinism and general nondeterminism, is a recurring question in the computation theory, which has been intensively studied for various devices, such as finite automata and resource-bounded Turing machines. Since the languages recognized by input-driven pushdown automata share many desirable properties of the regular languages, understanding the descriptional complexity trade-offs between the nondeterministic, unambiguous and deterministic input-driven automata is a well-motivated theoretical question. Secondly, as input-driven automata have applications to representing XML syntax, the unambiguous variant represents those automata that assign a unique syntactic structure to any syntactically well-formed string, and can thus be compared to unambiguous context-free grammars.

This paper introduces *unambiguous input-driven pushdown automata* (UIDPDA) and investigates their descriptional complexity tradeoffs with DIDPDAs and NIDPDAs. In Sections 4–5 it is shown that converting an unambiguous automaton of size n to a deterministic one, or nondeterministic automaton of a size n to an unambiguous automaton, requires size $2^{\Theta(n^2)}$ in the worst case. By the size of an IDPDA we mean the sum of the number of states and of stack symbols. That is, in both cases, the size explosion is the same as for determinizing an NIDPDA.

To establish lower bounds on the size of NIDPDAs, this paper employs fooling set methods [14, 29], that have been originally introduced for proving lower bounds for NFAs [6, 35]. The general lower bound method for UFAs is based on the rank of a fooling set matrix; it was developed by E. M. Schmidt [34] already in

1978, and a good self-contained presentation was given by Leung [22]. Also Hromkovič et al. [17] have given an alternative proof using communication complexity. The lower bound technique based on the rank of a fooling set matrix is extended to unambiguous input-driven automata in Section 3.

The ideas used to establish the tradeoffs in Sections 4–5 are next applied to obtain further results on the complexity of operations on input-driven automata, in line with some recent work [14, 28, 29, 33]. The first operation to be investigated is *complementation*. It is known that the complement of an n-state NFA needs in the worst case 2^n states [15], and the same bound for languages over a binary alphabet is known from Jirásková [18]. In other recent work, the complementation of two-way finite automata was studied by Geffert et al. [13], and complementation of unary UFAs was investigated by Okhotin [26]. For NIDPDAs, Han and Salomaa [14] gave a lower bound of $\sqrt{n!}$ states for complementing nondeterministic nested word automata, leaving the precise state complexity open. This paper demonstrates, in Section 6, that the complexity of complementation for NIDPDAs is $2^{\Theta(n^2)}$: in other words, in the worst case, one essentially has to determinize the automaton in order to represent its complement.

In the last Section 7, the complexity of homomorphisms for DIDPDAs is determined to be $2^{\Theta(n^2)}$ as well. The lower bound construction again relies on a modification of the languages used for the lower bound for the NIDPDA-to-UIDPDA conversion.

2. Input-driven pushdown automata

We assume that the reader is familiar with the basics of formal languages and finite automata [31, 35, 36]. The original reference on input-driven pushdown automata is the paper by Mehlhorn [23], and more details on this model and on its applications, as well as recent references, are given by Alur and Madhusudan [4].

In the following, Σ always denotes a finite alphabet. The set of strings over Σ is Σ^*, and Σ^+ is the set of nonempty strings. Let $\Sigma^{\leqslant m}$ with $m \geqslant 0$ denote the set of all strings over Σ of length at most m.

For $m \in \mathbb{N}$, denote $[1, m] = \{1, \ldots, m\}$. For every binary string $w = b_{\ell-1} \ldots b_1 b_0 \in \{0,1\}^*$, denote its numerical value by $(w)_2 = \sum_{i=0}^{\ell-1} b_i 2^i$.

Next, we recall and introduce definitions and notation concerning input-driven pushdown automata. An *action alphabet* is a triple $\widetilde{\Sigma} = (\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$, in which the components Σ_{+1}, Σ_{-1} and Σ_0 are finite disjoint sets. Unless otherwise mentioned, Σ_{+1}, Σ_{-1} and Σ_0 always refer to components of an action alphabet, and their union is denoted by Σ. In the following, when referring to a string $w \in \Sigma^*$, unless otherwise mentioned Σ is assumed to be the underlying alphabet of an action alphabet.

The distinguishing property of input-driven pushdown automata is that the type of the stack operation is always determined by the input symbol, and in the definition of an NIDPDA it is convenient to use three separate transition functions, one for each type of input symbol.

Definition 1. *A* nondeterministic input-driven pushdown automaton, NIDPDA, *is a tuple*

$$A = (\widetilde{\Sigma}, \Gamma, Q, q_0, F, \delta_0, \delta_{+1}, \delta_{-1})$$

where $\Sigma = \Sigma_{+1} \cup \Sigma_{-1} \cup \Sigma_0$ *is the input alphabet,* Γ *is the finite set of stack symbols,* Q *is the finite set of internal states,* $q_0 \in Q$ *is the start state,* $F \subseteq Q$ *is the set of final states,* $\delta_0 : Q \times \Sigma_0 \to 2^Q$ *is the internal transition function, and* $\delta_{+1} : Q \times \Sigma_{+1} \to 2^{Q \times \Gamma}$ *and* $\delta_{-1} : Q \times (\Gamma \cup \{\bot\}) \times \Sigma_{-1} \to 2^Q$ *are the transition functions determining the push and pop operations, respectively. The symbol* $\bot \notin \Gamma$ *is used to denote the empty stack.*

A *configuration of* A is a tuple (q, w, u), where $q \in Q$ is the state, $w \in \Sigma^*$ is the remaining input and $u \in \Gamma^*$ is the stack contents. A stack ε is, for the purposes of the transition relation δ_{-1}, interpreted to contain the bottom of stack symbol $\bot$. The *height* of the stack of a configuration (q, w, u) is $|u|$, and hence the height of the empty stack is zero. The set of configurations of A is $\mathcal{C}(A)$, and we define the single step computation relation $\vdash_A \subseteq \mathcal{C}(A) \times \mathcal{C}(A)$ as follows.

For $a \in \Sigma_0$ we have: $(q, aw, u) \vdash_A (q', w, u)$, for all $q' \in \delta_0(q, a)$, $w \in \Sigma^*$ and $u \in \Gamma^*$.

For $a \in \Sigma_{+1}$ we have: $(q, aw, u) \vdash_A (q', w, \gamma u)$, for all $(q', \gamma) \in \delta_{+1}(q, a)$, $\gamma \in \Gamma$, $w \in \Sigma^*$ and $u \in \Gamma^*$.

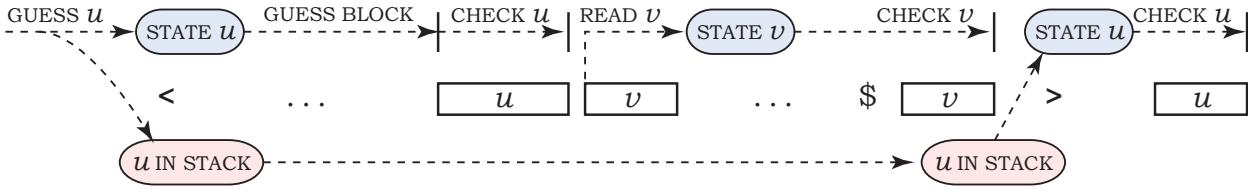

Figure 1: Recognizing the language L_n from Proposition 1 by an NIDPDA with $O(n)$ states.

For $a \in \Sigma_{-1}$ we have: $(q, aw, \gamma u) \vdash_A (q', w, u)$ for all $q' \in \delta_{-1}(q, \gamma, a)$, $\gamma \in \Gamma$, $w \in \Sigma^*$ and $u \in \Gamma^*$; furthermore, $(q, aw, \varepsilon) \vdash_A (q', w, \varepsilon)$, for all $q' \in \delta_{-1}(q, \bot, a)$ and $w \in \Sigma^*$.

According to the last case, when the automaton A encounters a symbol $a \in \Sigma_{-1}$ with an empty stack, it can make any transition chosen from $\delta_{-1}(q, \bot, a)$, where $q \in Q$ is the current state of A, and the stack remains empty. When reading a symbol from $\Sigma_0 \cup \Sigma_{+1}$, the behavior does not depend on the top stack symbol. Naturally, given an NIDPDA A, it would be possible to construct an NIDPDA B that simulates the computation of A and keeps track (in its state) of the topmost stack symbol in the corresponding computation of A, however, the transformation would need to increase the number of states and stack symbols of A. In order to update correctly the stack symbol stored in the state after a pop operation, each stack symbol of B would need to keep track of the previous stack symbol in the simulated computation of A. Thus, if A has k states and h stack symbols, then $k \cdot h$ states and h^2 stack symbols will be sufficient for B.

The initial configuration of A on an input $w \in \Sigma^*$ is $C_A^{\text{init}}(w) = (q_0, w, \varepsilon)$. The language recognized by A is defined as

$$L(A) = \{w \in \Sigma^* \mid C_A^{\text{init}}(w) \vdash_A^* (q, \varepsilon, u) \text{ for some } q \in F, u \in \Gamma^*\}.$$

An IDPDA accepts by final state only. By an *input-driven language* we mean a language recognized by some IDPDA.

As special cases of the automata of Definition 1, we obtain the unambiguous and the deterministic input-driven pushdown automata. An input-driven pushdown automaton A is *unambiguous* (a UIDPDA) if it has exactly one accepting computation for any $w \in L(A)$, that is, a unique sequence of configurations $C_1, C_2, \ldots, C_m$ such that $C_1 = C_A^{\text{init}}(w)$, $C_i \vdash_A C_{i+1}$, $1 \leq i \leq m-1$, and C_m is of the form (q, ε, u) where $q \in F$, $u \in \Gamma^*$.

An input driven pushdown automaton A is said to be *deterministic* (a DIDPDA) if its transition functions δ_0, δ_{+1} and δ_{-1} give at most one action in each configuration, that is, are defined as partial functions $\delta_0 \colon Q \times \Sigma_0 \to Q$, $\delta_{+1} \colon Q \times \Sigma_{+1} \to Q \times \Gamma$ and $\delta_{-1} \colon Q \times (\Gamma \cup \{\bot\}) \times \Sigma_{-1} \to Q$.

When speaking about the descriptional complexity of input-driven pushdown automata, by the *size* of an NIDPDA A as in Definition 1 we mean the sum of the number of its states and the number of its stack symbols, that is, $|Q| + |\Gamma|$.

The following result gives an upper bound on the size blow-up of determinizing an NIDPDA. Note that in Proposition 1, as well as in the following sections, when citing results from papers using the nested word automaton formalism, we translate them to the terminology of input-driven pushdown automata, that is, we speak about states and stack symbols instead of linear and vertical states.

Proposition 1 (Alur and Madhusudan [4])**.** *An NIDPDA with k states and h stack symbols can be simulated by a DIDPDA with $2^{k \cdot h}$ states and 2^{h^2} stack symbols. There exist languages L_n for $n \geqslant 1$, recognized by NIDPDAs of size $O(n)$ such that any DIDPDA for L_n needs size 2^{n^2}.*

A matching lower bound for both the number of states and the number of stack symbols was given by Okhotin, Piao and Salomaa [27].

The second part of Proposition 1, where the languages L_n are constructed, deserves an explanation, because this paper develops two new constructions of similar languages with different properties. The

witness languages of Alur and Madhusudan [4] are defined as

$$L_n = \big\{ <u_1\#v_1\#u_2\#v_2\#\ldots\#u_\ell\#v_\ell\$v>u \;\big|\; \ell \geqslant 1,\ u_i, v_i \in \{0,1\}^+ \text{ for all } i \in \{1,\ldots,\ell\},\ u, v \in \{0,1\}^{\lceil \log n \rceil}, \text{ and there exists } t \in \{1,\ldots,\ell\} \text{ with } u = u_t,\ v = v_t \big\},$$

and an NIDPDA with $O(n)$ states recognizes this language as follows. At the first step of the computation, upon reading the left bracket $<$, the automaton guesses the string u and pushes a symbol representing this string to the stack, as well as stores it in the internal state. Then the automaton nondeterministically decides to skip any even number of blocks, verifies that the next string is u, and then stores the following string v in its internal state. After skipping until the dollar sign, the automaton verifies that the last string inside the brackets is exactly v. Finally, upon reading the right bracket $>$ the automaton pops u from the stack and compares this string to the remaining symbols of the input string, to verify that they are the same. The data flow in such a computation is illustrated in Figure 1.

2.1. Stack height, well nested strings and homomorphisms

Since the type of the stack operation is always uniquely determined by the input symbol, the height of the stack that any NIDPDA reaches after reading an input string w is uniquely determined by w. Thus we can introduce the following notion of *stack height.*

A configuration (q, ε, u), with $q \in Q$, $u \in \Gamma^*$ and with the remaining input empty, is called a *terminal configuration.* If $\mathcal{C}_A^{\text{init}}(w) \vdash_A^* C$, where C is a terminal configuration with stack contents u, we say that the *stack height* of w is $|u|$. As observed above, the stack height of a string w is a property of w that does not depend on the nondeterministic choices made by A during the computation.

A string w over an action alphabet $\widetilde{\Sigma}$ defines a natural matching relation between occurrences of elements of Σ_{+1} and of Σ_{-1}, respectively, by matching each occurrence of an element of Σ_{+1} with the next unmatched occurrence of an element of Σ_{-1} [4]. In an arbitrary computation of an NIDPDA on a string w, a stack symbol pushed at an occurrence of a symbol of Σ_{+1} will always be popped at a matching occurrence of a symbol from Σ_{-1}, assuming that there is a matching symbol of type -1 in w. (If there is no matching occurrence of a symbol of type -1, the stack symbol will never be popped.) The stack height of w is then defined as the number of occurrences of symbols of type $+1$ in w that do not have a matching symbol of type -1.

We say that $w \in \Sigma^*$ is *well nested*, if every occurrence of a symbol of Σ_{+1} has a matching occurrence of a symbol of Σ_{-1}, and vice versa. The computations of an input-driven pushdown automaton on a well nested string w have the following property: if an NIPDPA A begins reading w with stack contents u, the computation ends with the same stack contents u without ever touching the initial contents of the stack.

Since the stack operations of an NIDPDA are determined by the type of the input symbols, arbitrary homomorphisms defined on $\Sigma = \Sigma_{+1} \cup \Sigma_{-1} \cup \Sigma_0$ do not preserve the class of input-driven languages: for instance, if a symbol in Σ_{+1} is mapped to a string of two symbols from Σ_{+1}, then it breaks the nestedness of brackets, and may map such an input-driven language as $\{ <^n>^n \mid n \geqslant 0 \}$ to a non-input-driven language $\{ <^{2n}>^n \mid n \geqslant 0 \}$. Below we define a class of homomorphisms that respect the types of symbols and, thus, maintain the nesting of the correspondence between occurrences of symbols of type $+1$ and -1, respectively.

Definition 2. *Let $\widetilde{\Sigma}$ be an action alphabet. Let $h : \widetilde{\Sigma} \to \Sigma^*$ be a mapping such that*

(i) *for each $\sigma \in \Sigma_0$, $h(\sigma)$ is well nested,*

(ii) *for each $\sigma \in \Sigma_{+1}$, $h(\sigma)$ contains exactly one unmatched symbol of type $+1$ and no unmatched symbols of type -1,*

(iii) *for each $\sigma \in \Sigma_{-1}$, $h(\sigma)$ contains exactly one unmatched symbol of type -1 and no unmatched symbols of type $+1$.*

The mapping h determines a homomorphism that respects nesting *$\overline{h} : \Sigma^* \to \Sigma^*$ by setting $\overline{h}(\varepsilon) = \varepsilon$, and, for $\sigma \in \Sigma$, $w \in \Sigma^*$, $\overline{h}(\sigma w) = h(\sigma)\overline{h}(w)$. For simplicity we use h in place of $\overline{h}$.*

We say that h is a Σ_0-relabeling *if h is the identity on Σ_{+1} and Σ_{-1}, and maps symbols of Σ_0 into Σ_0 (not necessarily injectively).*

In the following, unless otherwise mentioned, by a *homomorphism* we mean always a homomorphism that respects nesting. Note that Alur and Madhusudan [4] used a slightly different definition for the above notion: they consider a multiple-valued substitution, in which all strings in each set $h(\sigma)$ with $\sigma \in \Sigma_{+1}$, have an unmatched symbol of type $+1$ as their first symbols, while all images of $\sigma \in \Sigma_{-1}$ must end with a unique unmatched symbol, A one-valued homomorphism that respects nesting according to Alur and Madhusudan [4] is a special case of a homomorphism of Definition 2. As in the cited paper [4], it can be shown that the family of input-driven languages is closed under homomorphisms (that respect nesting).

3. Lower bounds on the size of input-driven automata

We recall some techniques for establishing lower bounds on the size of deterministic and nondeterministic input-driven pushdown automata [14, 29]. These results are straightforward extensions of the well-known fooling set method for NFAs [6, 35]. However, already in the case of *deterministic* input-driven pushdown automata, these methods do not always yield a precise lower bound [33].

Let $\widetilde{\Sigma} = (\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$ be an action alphabet and, as usual, we denote $\Sigma = \Sigma_{+1} \cup \Sigma_{-1} \cup \Sigma_0$. For $k \geqslant 1$, a *set of depth* k is a finite set S of strings over Σ each having stack height k. A finite set of pairs of strings $F = \{(x_i, y_i) \mid i = 1, 2, \ldots, m\}$ is said to be a *set of pairs of depth* k, where $k \geqslant 1$, if each string x_i has stack height k.

Definition 3. *Let $\widetilde{\Sigma} = (\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$ be an action alphabet and let $L \subseteq \Sigma^*$.*

(i) *A set S of depth k is a* separator set of depth k *for L, if every element of S is a prefix of some string in L, and for any two-element set $\{u, v\} \subseteq S$ with $u \neq v$, there exists $x \in \Sigma^*$, such that one of the strings ux, vx is in L, while the other is not.*

(ii) *A set of pairs $F = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ of depth k is said to be a* fooling set of depth k *for L, if*

(ii-a) *$x_i y_i \in L$ for all $i \in \{1, 2, \ldots, m\}$, and*

(ii-b) *for all i, j with $1 \leqslant i < j \leqslant m$, $x_i y_j \notin L$ or $x_j y_i \notin L$.*

Lemma 1 ([14, 29])**.** *Let A be a (deterministic or nondeterministic) input-driven pushdown automaton with a set of states Q and a set of stack symbols Γ.*

(i) *If A is a DIDPDA and S is a separator set of depth k for $L(A)$ then $|\Gamma|^k \cdot |Q| \geqslant |S|$.*

(ii) *If $L(A)$ has a fooling set F of depth k, then $|\Gamma|^k \cdot |Q| \geqslant |F|$.*

The above definition of a fooling set of depth k is similar to the fooling sets of Birget [6] used for NFAs. In the case of DIDPDAs, there is a unique computation on a given prefix, which means that instead of pairs of strings, it is sufficient to consider only individual strings to be separated. The requirement that all strings of a separator set of depth k (or first components of a fooling set of depth k) must have the same stack height limits the use of Lemma 1.

We need a stronger lower bound condition for UIDPDAs to establish a trade-off for converting a nondeterministic automaton to an unambiguous one. In the domain of ordinary finite automata without a pushdown, the first lower bound argument for UFAs was given by Schmidt [34, Thm. 3.9] in his proof of a $2^{\Omega(\sqrt{n})}$ lower bound on the NFA-to-UFA tradeoff. We recall here a general statement of Schmidt's lower bound method due to Leung [22]:

Schmidt's Theorem [34, 22]. *Let $L \subseteq \Sigma^*$ be a regular language and let $F = \{(u_1, v_1), \ldots, (u_n, v_n)\}$ with $n \geqslant 1$ and $u_i, v_i \in \Sigma^*$ be a finite set of pairs of strings. Consider the integer matrix $M(F, L) \in \mathbb{Z}^{n \times n}$ defined by $M(F, L)_{i,j} = 1$ if $u_i v_j \in L$, and $M(F, L)_{i,j} = 0$ otherwise. Then every UFA recognizing L has at least* $\operatorname{rank} M(F, L)$ *states.*

We translate Schmidt's Theorem to the case of unambiguous input-driven pushdown automata. Let L be an input-driven language and let $F = \{(x_i, y_i) \mid i = 1, \ldots, n\}$ be a set of pairs of depth $k \geqslant 1$. Analogously to the above, we define an integer matrix $M(F, L) \in \mathbb{Z}^{n \times n}$ by setting $M(F, L)_{i,j} = 1$ if $x_i y_j \in L$ and $M(F, L)_{i,j} = 0$ otherwise.

Lemma 2. *Let $F = \{(x_i, y_i) \mid i = 1, \ldots, n\}$ be a set of pairs of depth k. Suppose that an input-driven language L is recognized by a UIDPDA A with a set of states Q and a set of stack symbols Γ. Then*

$$|\Gamma|^k \cdot |Q| \geqslant \operatorname{rank} M(F, L).$$

Proof. The proof is analogous to the one given by Leung [22, Thm. 2], except that instead of considering the states reached by A after reading each string x_i, we now consider both the states and the stack contents that the computations of A reach after reading each string x_i. For completeness, we include here a short proof.

We define a matrix $\widehat{M}(F, A)$, where the rows are indexed by elements $\mathbf{f} \in \Gamma^k \times Q$ and the columns are indexed by the strings y_i with $1 \leqslant i \leqslant n$, by setting $\widehat{M}(F, A)_{\mathbf{f}, y_i} = 1$ if $\mathbf{f} = (u, q)$ and A has an accepting computation starting from the configuration (q, y_i, u), and $\widehat{M}(F, A)_{\mathbf{f}, y_i} = 0$ otherwise.

For each $i \in \{1, \ldots, n\}$, let $H_i \subseteq \Gamma^k \times Q$ be the set of pairs of stack contents and states that A can reach after reading one of the inputs $x_1, \ldots, x_n$. Recall that since x_i has stack height k, the stack contents of A after reading x_i is in Γ^k. We note that the unambiguity of A guarantees that if $\widehat{M}(F, A)_{\mathbf{f}, y_j} = 1$ for some $\mathbf{f} \in H_i$, $1 \leqslant j \leqslant n$, then $\widehat{M}(F, A)_{\mathbf{f}', y_j} = 0$ for all $\mathbf{f}' \in H_i$, $\mathbf{f}' \neq \mathbf{f}$. This means that the ith row of $M(F, L)$ (corresponding to x_i) is a sum of the rows of $\widehat{M}(F, A)$ indexed by elements of H_i. Since each row of $M(F, L)$ is a sum of rows of $\widehat{M}(F, A)$,

$$\operatorname{rank} M(F, L) \leqslant \operatorname{rank} \widehat{M}(F, A).$$

On the other hand, $\operatorname{rank} \widehat{M}(F, A) \leqslant |\Gamma|^k \cdot |Q|$, which is the number of rows of $\widehat{M}(F, A)$. □

4. From unambiguous to deterministic

The size blow-up of converting a UIDPDA to a DIDPDA turns out to be, in the worst case, the same as for determinizing a general nondeterministic nested word automaton. Recall that, as elsewhere in this paper, the size of an input-driven pushdown automaton means the sum of the number of states and the number of stack symbols.

Theorem 1. *For every $n \geqslant 1$ there exists an input-driven language K_n recognized by a UIDPDA of size $O(n)$, such that every DIDPDA for K_n needs size $2^{\Theta(n^2)}$.*

Proof. Let $\Sigma_0 = \{0, 1, \#, \$\}$, $\Sigma_{+1} = \{<\}$, $\Sigma_{-1} = \{>\}$. For $n \geqslant 1$ define the language

$$K_n = \big\{\, <x_0 \# x_1 \# \ldots \# x_\ell \$ v > u \;\big|\; x_i \in \{0,1\}^* \text{ for all } i \in \{0, 1, \ldots, \ell\},\ u, v \in \{0,1\}^{\lceil \log n \rceil}, \\ \text{and the bit number } (v)_2 \text{ in } x_{(u)_2} \text{ is } 1 \big\}$$

The language K_n is recognized unambiguously as follows. In the following discussion, we assume that the input string is in the language $L_0 = <(\{0,1\}^*\#)^*\{0,1\}^*\$\{0,1\}^*>\{0,1\}^*$, that is, of the general form given in the definition of the language K_n. By increasing the number of states of the UIDPDA with a multiplicative constant, it is easy to guarantee that all computations reject otherwise. This is done by adding (by a standard cross-product construction) a second component to the states of the UIDPDA, which is used for simulating a DFA recognizing the regular language L_0.

The computation of a UIDPDA A recognizing the language K_n is illustrated in Figure 2. At the first input symbol $< \in \Sigma_{+1}$, the automaton guesses a string $u \in \{0,1\}^{\lceil \log n \rceil}$ and stores it both in its state and in the symbol pushed to the stack. Using the state, the computation counts the separation markers # to locate the $(u)_2$-th binary substring $x_{(u)_2}$, forgetting $(u)_2$ in the process. Then, the automaton nondeterministically chooses any true bit of $x_{(u)_2}$ and remembers the number of this bit in the internal state; let this be the

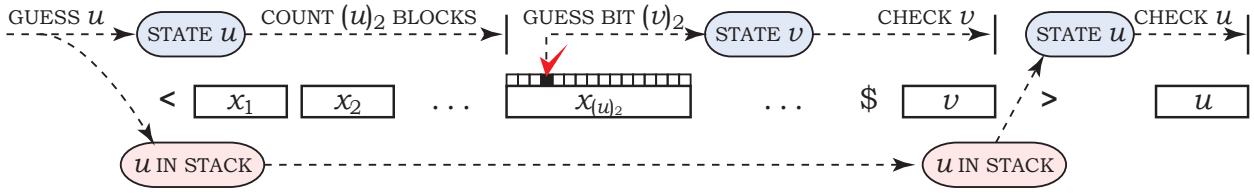

Figure 2: Recognizing the language K_n in the proof of Theorem 1 by a UIDPDA with $\Theta(n)$ states.

$(v)_2$-th bit of $x_{(u)_2}$, with $v \in \{0,1\}^{\lceil \log n \rceil}$. The next task carried out by the automaton is checking that the binary string occurring after the marker \$ equals to v. Finally, after popping the stack at the symbol $> \in \Sigma_{-1}$, the computation verifies, using the information in the symbol popped from the stack, that the binary string after the symbol $>$ is equal to u. All nondeterministic choices in a successful computation are pre-determined by the suffix of the input $v{>}u$, and for each input there can be only one accepting computation. This computation is illustrated in Figure 2.

It remains to show the lower bound on the size of any DIDPDA recognizing K_n. For every set $R \subseteq [0, n-1] \times [0, n-1]$, consider the unique string

$$w_R = {<}x_0 \# x_1 \# \ldots \# x_{n-1}\$,$$
$$\text{where } x_i \in \{0,1\}^n \text{ and for each } i,j \in \{0,1,\ldots,n-1\}, \text{ the } j\text{th bit of } x_i \text{ is 1 iff } (i,j) \in R.$$

It is claimed that the set of strings w_R, with $R \subseteq [0, n-1] \times [0, n-1]$, is a separator set of depth 1 for K_n. Indeed, for any distinct sets $R_1 \neq R_2$ there is a pair (i,j) belonging to one of them but not to the other; assume, without loss of generality, that $(i,j) \in R_1 \setminus R_2$. Let $u, v \in \{0,1\}^{\lceil \log n \rceil}$ be the binary notations of these numbers, with $i = (u)_2$ and $j = (v)_2$. Then $w_{R_1} v{>}u \in K_n$ and $w_{R_2} v{>}u \notin K_n$.

By Lemma 1 (i), if A is a DIDPDA for K_n with a state set Q and a stack alphabet Γ, then $|Q| \cdot |\Gamma| \geq |R|$. Thus, any DIDPDA for K_n needs a size of at least $2^{\frac{n^2}{2}}$. □

Note that, in the proof of Theorem 1, all strings in K_n contain only one pair of matching symbols of Σ_{+1} and Σ_{-1}, and thus any automaton recognizing this language makes only one push and one pop operation during its computation. Consequently, when establishing a lower bound on the size of a DIDPDA recognizing K_n, in Lemma 1 (i), one can always choose $k = 1$. This property is shared with the lower bound on the NIDPDA determinization blowup due to Alur and Madhusudan [4]. This is possible in both results, because the size of an input-driven pushdown automaton is measured as the sum of the number of states and the number of stack symbols.

Recently, Okhotin et al. [27] gave a more refined lower bound construction for the NIDPDA determinization blowup, which measures the size of an automaton as a pair of integers, consisting of the number of states and the number of stack symbols. In that construction, the witness languages require an unbounded number of matching pairs of symbols from Σ_{+1} and Σ_{-1}. However, for the simpler descriptional complexity measure adopted in this paper, there is no need to use more than one pair of brackets.

5. From nondeterministic to unambiguous

We show that transforming an NIDPDA to an equivalent UIDPDA entails, in the worst case, the same $2^{\Omega(n^2)}$ size blow-up as for the conversion of UIDPFAs to DIDPDAs.

We first define a class of languages L_n, with $n \geqslant 1$, that will be used for the lower bound construction, as well as for establishing lower bounds in the later sections. Choose

$$\Sigma_0 = \{a, b, \#\},\ \Sigma_{+1} = \{<\}, \Sigma_{-1} = \{>\}$$

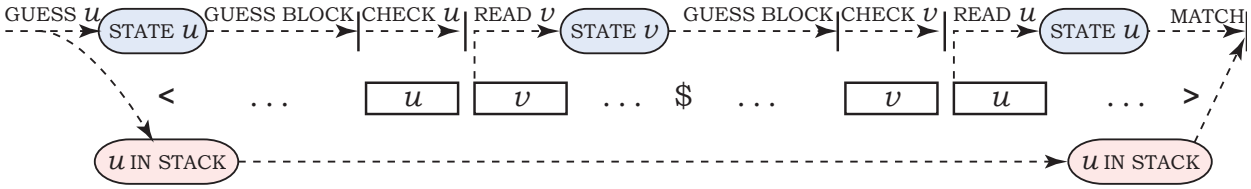

Figure 3: Recognizing the language L_n in the proof of Theorem 2 by an NIDPDA with $\Theta(n)$ states.

and define, for every $n \geqslant 1$,

$$L_n = \big\{ < a^{k_1} b^{\ell_1} a^{k_2} b^{\ell_2} \ldots a^{k_m} b^{\ell_m} \# b^{\ell'_1} a^{k'_1} b^{\ell'_2} a^{k'_2} \ldots b^{\ell'_{m'}} a^{k'_{m'}} > \mid m, m' \geqslant 1,\ k_i, \ell_i, k'_j, \ell'_j \geqslant 1,\ \exists s, t:\ k_s = k'_t \leqslant n,\ \ell_s = \ell'_t \leqslant n \big\}. \quad (1)$$

Theorem 2. *For every $n \geqslant 1$, the language L_n is recognized by an NIDPDA of size $O(n)$, but every UIDPDA for L_n needs size at least $2^{\lfloor \frac{n^2}{2} \rfloor - 1}$.*

Proof. An NIDPDA recognizing this language is defined as follows. Its computation on any well-formed input in $<\{a,b\}^*\#\{a,b\}^*>$ aims to detect a pattern of the form $<\ldots uv\ldots\#\ldots vu\ldots>$. Upon reading the first symbol $< \in \Sigma_{+1}$, the NIDPDA guesses an integer $i \in \{1,\ldots,n\}$ representing the string $u = a^i$, pushes a symbol representing i to the stack and remembers it in the state. Then the computation proceeds as illustrated in Figure 3. First, it nondeterministically selects a substring of the form $w = ba^r b^s a$ in the part preceding the "middle marker" $\#$, and verifies that r is equal to the number i stored in the state. If the selected substring w occurs directly after $<$, the first b is missing; similarly, if w occurs directly before $\#$, then the last a is missing. Next, the computation memorizes the integer s in the state and proceeds to the part after the symbol $\#$, where another substring in b^*a^* encoding a pair of integers is nondeterministically selected. The computation verifies that the first component of this pair is equal to b^s, and then memorizes the second component in the state. Finally, upon the transition at the last symbol $> \in \Sigma_{-1}$, this memorized length of the second component is compared to the value of i stored in the stack.

The computation uses exactly n stack symbols. The state of the automaton needs to store an integer of size at most n, count the length of a nondeterministically chosen unary string of length at most n and remember whether or not the computation has passed the "middle marker" $\#$. For this, $O(n)$ states are clearly sufficient.

For each set $R \subseteq [1,n] \times [1,n]$, define the strings $x_R = a^{i_1} b^{j_1} \cdots a^{i_m} b^{j_m}$ and $y_R = b^{j_1} a^{i_1} \cdots b^{j_m} a^{i_m}$, where $R = \{(i_1,j_1),\ldots,(i_m,j_m)\}$. For each $R \subseteq [1,n] \times [1,n]$, the elements of R are listed in an arbitrary but fixed order, so that the strings x_R and y_R are fixed.

The lower bound on the size of every UIDPDA recognizing this language is obtained by using a set of pairs F of depth 1. The set F is of size $2^{\lfloor \frac{n^2}{2} \rfloor}$ consisting of some pairs $(<x_R\#,\ y_{f(R)}>)$, where R is a subset of $[1,n] \times [1,n]$ of size $\lfloor \frac{n^2}{2} \rfloor$ and $f(R)$ is another subset of $[1,n] \times [1,n]$ of the same size depending on R. We will need that for all $R, S \subseteq [1,n] \times [1,n]$, the sets R and $f(S)$ are disjoint if and only if $R = S$.

The set F with the above properties is constructed as follows. Denote $z = \lfloor \frac{n^2}{2} \rfloor$. First consider the case where n is even, and let

$$p_{(1,0)}, p_{(1,1)}, p_{(2,0)}, p_{(2,1)}, \ldots, p_{(z,0)}, p_{(z,1)} \quad (2)$$

be an arbitrary but fixed enumeration of $[1,n] \times [1,n]$. If n is odd, define the list (2) to contain all elements of $[1,n] \times [1,n] \setminus \{(n,n)\}$, again listed in any order. For $\mathbf{v} = b_1 b_2 \cdots b_z$ with $b_i \in \{0,1\}$, denote by $\overline{\mathbf{v}} = b'_1 b'_2 \cdots b'_z$ the inverted string, where $b'_i = 1$ if $b_i = 0$ and $b'_i = 0$ if $b_i = 1$ for all $i \in \{1,\ldots,z\}$. Also for $\mathbf{v} = \{0,1\}^z$, define the set $R_{\mathbf{v}} \subseteq [1,n] \times [1,n]$ by setting

$$R_{\mathbf{v}} = \{\, p_{(1,b_1)},\ p_{(2,b_2)},\ \ldots,\ p_{(z,b_z)} \,\}.$$

Now define

$$F = \{\, (<x_{R_{\mathbf{v}}}\#,\ y_{R_{\overline{\mathbf{v}}}}>) \mid \mathbf{v} \in \{0,1\}^z \,\},$$

which is a set of $2^z = 2^{\lfloor \frac{n^2}{2} \rfloor}$ pairs.

The above choices guarantee that each string $<x_{R_{\mathbf{v}}} \# y_{R_{\overline{\mathbf{v}}}}>$ is not in L_n, while $<x_{R_{\mathbf{v}}} \# y_{R_{\overline{\mathbf{v}'}}}> \in L_n$ for all $\mathbf{v} \neq \mathbf{v}'$. Therefore, the corresponding matrix $M(F, L_n)$, as defined in Lemma 2, has a zero diagonal and ones as the rest of its elements. Hence, this $2^z \times 2^z$ matrix has a full rank, and according to Lemma 2, each UIDPDA with a set of states Q and a stack alphabet Γ recognizing the language L_n must satisfy $|\Gamma| \cdot |Q| \geqslant 2^{\lfloor \frac{n^2}{2} \rfloor}$. □

6. Complementing nondeterministic automata

We show that complementing an NIDPDA of size n requires, in the worst case, size $2^{\Theta(n^2)}$. We consider again the languages L_n with $n \geqslant 1$, defined in (1), and the following lemma establishes the lower bound.

Lemma 3. *Let A be an NIDPDA with a set of states Q and a pushdown alphabet Γ, which recognizes the language $\overline{L_n}$, where L_n is the language (1) defined in Section 5. Then $|Q| \cdot |\Gamma| \geqslant 2^{n^2}$.*

Proof. The construction of a fooling set is similar to the construction used in Theorem 2, however, now the fooling set can be based on arbitrary binary relations on $[1, n]$.

As in the proof of Theorem 2, for each set $R \subseteq [1, n] \times [1, n]$ we define the strings $x_R \in (a^+b^+)^*$ and $y_R \in (b^+a^+)^*$ that "list" the elements of R in an arbitrary order. We define a set of pairs S of depth 1 to consist of all pairs $(<x_R \#, y_{\overline{R}}>)$, where $R \subseteq [1, n] \times [1, n]$. Since every $R_1, R_2 \subseteq [1, n] \times [1, n]$ with $R_1 \neq R_2$ satisfy $R_1 \cap \overline{R_2} \neq \emptyset$ or $R_2 \cap \overline{R_1} \neq \emptyset$, one of the strings $<x_{R_1} \# y_{\overline{R_2}}>$, $<x_{R_2} \# y_{\overline{R_1}}>$ must be in L_n. This means that S is a fooling set of depth 1 for $\overline{L_n}$, and the claim follows by Lemma 1 (ii). □

As a consequence, we get a descriptional complexity bound for the complementation of NIDPDAs that is tight within a multiplicative constant in the exponent.

Theorem 3. *The worst-case size needed to recognize the complement of a language recognized by an NIDPDA of size n is $2^{\Theta(n^2)}$.*

Proof. Proposition 1 gives an upper bound $2^{O(n^2)}$. By Theorem 2, the language L_n can be recognized by an NIDPDA of size $O(n)$, and hence the result follows by Lemma 3. □

7. Homomorphic images of deterministic automata

We establish a tight bound for the deterministic state complexity of homomorphisms that respect nesting. Recall that, as observed in Section 2, the input-driven languages are not closed under homomorphisms defined in the usual way that are not required to respect the types of symbols. For the worst-case lower bound, it is sufficient to consider Σ_0-relabelings.

We use a modification of the language (1), where, roughly speaking, every place that requires a nondeterministic guess is annotated with special symbols that communicate the guess to a deterministic automaton Thus, the pairs of substrings that need to be compared are marked by the new symbols $a_\natural$ and $b_\natural$. Let $\Sigma_0 = \{0, 1, a, b, a_\natural, b_\natural, \#\}$, $\Sigma_{+1} = \{<\}$, $\Sigma_{-1} = \{>\}$. For $n \geqslant 1$ we define the language L'_n to consist of all strings of the form

$$w<(a^+b^+)^* a_\natural a^{(w)_2 - 1} b^\ell (a^+b^+)^* \# (b^+a^+)^* b_\natural b^{\ell-1} a^{(w)_2} (b^+a^+)^*>, \tag{3}$$

where $w \in \{0,1\}^{\lceil \log n \rceil}$ and $1 \leqslant \ell \leqslant n$.

Lemma 4. *The language L'_n is recognized by a DIDPDA A with $O(n)$ states and n stack symbols.*

Proof. At the beginning of its computation, A memorizes the number $(w)_2$ in its state, and, when reading the input symbol $< \in \Sigma_{+1}$, stores this number in the stack as a single stack symbol. Then the computation checks that the length of the sequence of as following the first occurrence of $a_\natural$ equals to $(w)_2 - 1$. After this, the computation memorizes the number of bs immediately following the as (that is, ℓ) and checks that

the number of bs occurring after the symbol $b_\natural$ is $\ell - 1$. Then A stores in the state the length of the following sequence of a's, and finally, while popping the stack at the last symbol $>$, the computation checks that the number stored in the internal state is equal to the number $(w)_2$ popped from the stack. □

Let h be the homomorphism that maps $a_\natural$ to a, $b_\natural$ to b and 1 to 0, while mapping the rest of the symbols to themselves. Note that h is a Σ_0-relabeling.

Lemma 5. *Every DIDPDA for $h(L'_n)$ has to be of size at least $2^{\frac{n^2}{2}}$.*

Proof. We note that

$$h(L'_n) = \{\ 0^{\lceil \log n \rceil} u \ \mid\ u \in L_n\ \},$$

where L_n is defined in (1). For $R \subseteq [1,n] \times [1,n]$, we define the strings x_R and y_R as in the proof of Lemma 3. Let

$$S_1 = \{\ 0^{\lceil \log n \rceil} < x_R \#\ \mid\ R \subseteq [1,n] \times [1,n]\ \}.$$

Consider arbitrary relations $R_1, R_2 \subseteq [1,n] \times [1,n]$, where $R_1 \neq R_2$. Without loss of generality we assume that $R_1 - R_2 \neq \emptyset$. Now

$$0^{\lceil \log n \rceil} < x_{R_1} \# y_{\overline{R_2}} > \in h(L'_n) \text{ and } 0^{\lceil \log n \rceil} < x_{R_2} \# y_{\overline{R_2}} > \notin h(L'_n).$$

This means that S_1 is a separator set of depth 1 for $h(L'_n)$ and, by Lemma 1 (i), any DIDPDA for $h(L'_n)$ needs size $\sqrt{2^{n^2}}$. □

Now we can state a tight bound for the descriptional complexity of homomorphism for deterministic input-driven pushdown automata. The upper bound is established straightforwardly by constructing an NIDPDA to recognize the homomorphic image.

Theorem 4. *Let h be a homomorphism and A a DIDPDA of size n. The input-driven language $h(L(A))$ can be recognized by a DIDPDA of size $2^{O(n^2)}$.*

There exists a Σ_0-relabeling h and input-driven languages L'_n, with $n \geqslant 1$, recognized by a DIDPDA with $O(n)$ states and stack symbols, such that any DIDPDA for $h(L'_n)$ needs size at least 2^{n^2}.

Proof. Given h and A, the method of Alur and Madhusudan [4, Thm. 3.8] can be used to construct an NIDPDA B recognizing the homomorphic image[3] $h(L(A))$. The NIDPDA B nondeterministically guesses and verifies a decomposition of its input w as $h(a_1)h(a_2)\cdots h(a_m)$ with $a_i \in \widetilde{\Sigma}$, and in parallel simulates the computation of A on $a_1 \cdots a_m$. The construction relies on the property that if $a_i \in \Sigma_{+1}$, then $h(a_i)$ has exactly one unmatched symbol of type $+1$, and the stack symbol pushed to the stack in the simulated computation of A is pushed at the unmatched symbol; and symmetrically, for each $a_i \in \Sigma_{-1}$, its image $h(a_i)$ has a unique unmatched symbol of type -1, and the stack symbol popped in the simulated computation of A is popped at this unmatched symbol. If A is of size n, then the size of B will be $c \cdot n$, where c is a constant depending on h but independent of A. The upper bound follows from Proposition 1.

The lower bound was established in Lemma 5. □

8. Conclusion

We have shown that both the conversion of an unambiguous input-driven pushdown automaton of size n to a deterministic one and the conversion of a nondeterministic automaton of size n to an unambiguous nondeterministic automaton causes in the worst case $2^{\Theta(n^2)}$ size blow-up. Both descriptional complexity bounds are tight within a multiplicative constant in the exponent. As a future work, one can try to determine the associated multiplicative constants more precisely. It can be noted that for the cost of determinizing

[3]Even though the definition of a homomorphism used by Alur and Madhusudan [4] differs slightly from ours, the same construction works here.

a general nondeterministic input-driven pushdown automaton, the precise multiplicative constant in the exponent is also not known [4].

Not long ago, the state complexity of operations on input-driven pushdown automata was investigated by Piao and Salomaa [29] and by Okhotin and Salomaa [28]. Another subject for future research is the state complexity of operations on unambiguous input-driven pushdown automata, of which nothing is yet known. Not much is known about the operational state complexity of unambiguous finite automata either [26], and finding out these properties would lead to a better understanding of the power of unambiguous nondeterminism in automata.

References

[1] R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, L. Libkin, First-order and temporal logics for nested words. In: Proc. of 22nd IEEE Symposium on Logic in Computer Science, (2007) 151–160.

[2] R. Alur, V. Kumar, P. Madhusudan, M. Viswanathan, Congruences for visibly pushdown languages. In: Proc. 32nd ICALP. Lect. Notes Comput. Sci., Vol. 3580. Springer (2005) 1102–1114.

[3] R. Alur, P. Madhusudan, Visibly pushdown languages. In: Proc. ACM Symposium on Theory of Computing, STOC 2004, Chicago, USA (2004) 202–211

[4] R. Alur, P. Madhusudan, Adding nesting structure to words. J. Assoc. Comput. Mach. 56(3) (2009).

[5] M. Arenas, P. Barceló, L. Libkin, Regular languages of nested words: Fixed points, automata and synchronization. In: Proc. 34th ICALP. Lect. Notes Comput. Sci., Vol. 4596, Springer (2007) 888–900.

[6] J.-C. Birget, Intersection and union of regular languages and state complexity. Inform. Process. Lett. 43 (1992) 185–190.

[7] B. Bollig, On the expressive power of 2-stack visibly pushdown automata. Logical Methods in Computer Science 4:4 (2008), paper 16.

[8] B. von Braunmühl, R. Verbeek, Input-driven languages are recognized in $\log n$ space. *North-Holland Mathematics Studies*, 102 (1985), 1–19.

[9] P. Chervet, I. Walukiewicz, Minimizing variants of visibly pushdown automata. In: Proc. MFCS 2007. Lect. Notes Comput. Sci., Vol. 4708. Springer (2007) 135–146.

[10] P. Dymond, Input-driven languages are in $\log n$ depth. Inform. Process. Lett. 26 (1988) 247–250.

[11] O. Gauwin, J. Niehren, Y. Roos, Streaming tree automata. Inform. Proc. Lett. 109 (2008) 13–17.

[12] J. Goldstine, H. Leung, D. Wotschke, On the relation between ambiguity and nondeterminism in finite automata. Inform. Comput. 100 (1992) 261–270

[13] V. Geffert, C. Mereghetti, G. Pighizzini, Complementing two-way finite automata. Inform. Comput. 205:8 (2007) 1173–1187.

[14] Y.-S. Han, K. Salomaa, Nondeterministic state complexity of nested word automata. Theoret. Comput. Sci. 410 (2009) 2961–2971.

[15] M. Holzer, M. Kutrib, Nondeterministic descriptional complexity of regular languages. Internat. J. Foundations of Comput. Sci. 14 (2003) 1087–1102.

[16] M. Holzer, M. Kutrib, Descriptional and computational complexity of finite automata—A survey. Inform. Comput. 209 (2011) 456–470.

[17] J. Hromkovič, S. Seibert, J. Karhumäki, H. Klauck, G. Schnitger, Communication complexity method for measuring nondeterminism in finite automata. Inform. Comput. 172 (2002) 202–217.

[18] G. Jirásková, State complexity of some operations on binary regular languages. Theoret. Comput. Sci. 330 (2005) 287–298.

[19] C. A. Kapoutsis, Removing bidirectionality from nondeterministic finite automata. In: Prof. *Mathematical Foundations of Computer Science,* MFCS, Gdańsk, Poland, August 29–September 2, 2005), Lect. Notes Comput. Sci. 3618, Springer (2005) 544–555.

[20] M. Kunc, A. Okhotin, Describing periodicity in two-way deterministic finite automata using transformation semigroups. In: Proc. *Developments in Language Theory,* DLT, Milan, Italy, 19–22 July 2011, Lect. Notes Comput. Sci. 6795, Springer (2011) 324–336.

[21] H. Leung, Separating exponentially ambiguous finite automata from polynomially ambiguous finite automata. SIAM J. Comput. 27:4 (1998) 1073–1082.

[22] H. Leung, Descriptional complexity of NFA of different ambiguity. Internat. J. Foundations Comput. Sci. 16:5 (2005) 975–984.

[23] K. Mehlhorn, Pebbling mountain ranges and its application to DCFL-recognition. In: Proc. 7th ICALP, Lect. Notes Comput. Sci. 85, Springer (1980) 422-435.

[24] A. Neumann, H. Seidl, Locating matches of tree patterns in forests. In: Proc. 18th Foundations of Software Technology and Theoretical Computer Science 1998. Lect. Notes Comput. Sci. 1530, Springer (2004) 134–146

[25] A. Okhotin, Comparing linear conjunctive languages to subfamilies of the context-free languages. In: Proc. SOFSEM 2011: Theory and Practice of Computer Science. (Nový Smokovec, Slovakia, 22–28 January 2011), Lect. Notes Comput. Sci. 6543, Springer (2011) 431–443.

[26] A. Okhotin, Unambiguous finite automata over a unary alphabet. Inform. Comput. 212 (2012) 15–36.

[27] A. Okhotin, X. Piao, K. Salomaa, Descriptional complexity of input-driven pushdown automata. In: H. Bordihn, M. Kutrib, B. Truthe (Eds.) *Dassow Festschrift 2012,* Lect. Notes Comput. Sci. 7300, Springer-Verlag (2012) 186–206.

[28] A. Okhotin, K. Salomaa, State complexity of operations on input-driven pushdown automata. In: Mathematical Foundations of Computer Science, MFCS, Lect. Notes Comput. Sci. 6907, Springer-Verlag (2011) 485–496.
[29] X. Piao, K. Salomaa, Operational state complexity of nested word automata. Theoret. Comput. Sci. 410 (2009) 3290–3302.
[30] B. Ravikumar, O.H. Ibarra, Relating the type of ambiguity of finite automata to the succinctness of their representation. SIAM J. Comput. 18:6 (1989) 1263–1282.
[31] G. Rozenberg, A. Salomaa (Eds.), *Handbook of Formal Languages,* Vol. I–III, Springer-Verlag, 1997.
[32] W. Rytter, An application of Mehlhorn's algorithm for bracket languages to $\log n$ space recognition of input-driven languages. Inform. Process. Lett. 23 (1986) 81–84.
[33] K. Salomaa, Limitations of lower bound methods for deterministic nested word automata. Inform. Comput. 209 (2011) 580–589.
[34] E.M. Schmidt, *Succinctness of Description of Context-Free, Regular and Unambiguous Languages*. Ph. D. thesis, Cornell University, 1978.
[35] J. Shallit, *A Second Course in Formal Languages and Automata Theory.* Cambridge University Press (2009)
[36] S. Yu, Regular languages. In [31], Vol. I, pp. 41–110.