



# Descriptive complexity of unambiguous input-driven pushdown automata <sup>☆</sup>



Alexander Okhotin <sup>a,1</sup>, Kai Salomaa <sup>b,\*,2</sup>

<sup>a</sup> Department of Mathematics and Statistics, University of Turku, 20014 Turku, Finland

<sup>b</sup> School of Computing, Queen's University, Kingston, Ontario K7L 3N6, Canada

## ARTICLE INFO

### Article history:

Received 28 August 2012

Received in revised form 3 April 2014

Accepted 14 November 2014

Available online 20 November 2014

Communicated by G. Ausiello

### Keywords:

Nondeterminism

Unambiguity

Input-driven pushdown

Visibly pushdown automata

Nested word automata

Descriptive complexity

## ABSTRACT

It is known that a nondeterministic input-driven pushdown automaton (IDPDA) (a.k.a. visibly pushdown automaton; a.k.a. nested word automaton) with  $n$  states can be transformed to an equivalent deterministic automaton with  $2^{\Theta(n^2)}$  states (B. von Braunmühl and R. Verbeek, 1983 [8]), and that this size is necessary in the worst case (R. Alur and P. Madhusudan, 2009 [4]). 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 descriptive complexity of complementation for nondeterministic IDPDAs is  $2^{\Theta(n^2)}$ , and that the descriptive complexity of homomorphisms for deterministic IDPDAs is  $2^{\Theta(n^2)}$ .

© 2014 Elsevier B.V. All rights reserved.

## 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  $O(\frac{\log^2 n}{\log \log n})$ . 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  $\text{NC}^1$  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 provides a natural

<sup>☆</sup> A preliminary version of this paper, entitled “Descriptive 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.

\* Corresponding author.

E-mail addresses: alexander.okhotin@utu.fi (A. Okhotin), ksalomaa@cs.queensu.ca (K. Salomaa).

<sup>1</sup> Supported by the Academy of Finland under grants 134860 and 257857.

<sup>2</sup> Supported by the Natural Sciences and Engineering Research Council of Canada Grant OGP0147224.

data model 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 is the *pushdown forest automaton* 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 with any number of stack symbols needs in the worst case  $2^{\Theta(n^2)}$  states. Later Okhotin, Piao and Salomaa [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 descriptorial 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 descriptorial complexity trade-offs between the nondeterministic, unambiguous and deterministic input-driven automata is a well-motivated theoretical question. Secondly, as nondeterministic input-driven automata can be determinized and deterministic IDPDA 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 descriptorial complexity tradeoffs with DIDPDAs and NIDPDAs. In Sections 4–5 it is shown that converting an  $n$ -state unambiguous automaton to a deterministic one, or a nondeterministic automaton with  $n$  states to an unambiguous automaton, requires  $2^{\Theta(n^2)}$  states in the worst case. That is, in both cases, the size explosion is the same as for determinizing an NIDPDA.

To establish lower bounds on the number of states in 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,  $\Sigma$  always denotes a finite alphabet. The set of strings over  $\Sigma$  is  $\Sigma^*$ , and  $\Sigma^+$  is the set of nonempty strings. Let  $\Sigma^{\leq m}$  with  $m \geq 0$  denote the set of all strings over  $\Sigma$  of length at most  $m$ .

For  $m \in \mathbb{N}$ , denote  $[1, m] = \{1, \dots, m\}$ . For every binary string  $w = b_{\ell-1} \dots b_1 b_0 \in \{0, 1\}^*$ , 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  $\tilde{\Sigma} = (\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$ , in which the components  $\Sigma_{+1}$ ,  $\Sigma_{-1}$  and  $\Sigma_0$  are finite disjoint sets. Unless otherwise mentioned,  $\Sigma_{+1}$ ,  $\Sigma_{-1}$  and  $\Sigma_0$  always refer to components of an action alphabet, and their union is denoted by  $\Sigma$ . In the following, when referring to a string  $w \in \Sigma^*$ , unless otherwise mentioned,  $\Sigma$  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 IDPDA 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

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

where  $\Sigma = \Sigma_{+1} \cup \Sigma_{-1} \cup \Sigma_0$  is the input alphabet,  $\Gamma$  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,  $\delta_0: Q \times \Sigma_0 \rightarrow 2^Q$  is the internal transition function, and  $\delta_{+1}: Q \times \Sigma_{+1} \rightarrow 2^{Q \times \Gamma}$  and  $\delta_{-1}: Q \times (\Gamma \cup \{\perp\}) \times \Sigma_{-1} \rightarrow 2^Q$  are the transition functions determining the push and pop operations, respectively. The symbol  $\perp \notin \Gamma$  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  $\varepsilon$  is, for the purposes of the transition relation  $\delta_{-1}$ , interpreted to contain the bottom of stack symbol  $\perp$ . 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 the single step computation relation  $\vdash_A \subseteq \mathcal{C}(A) \times \mathcal{C}(A)$  is defined 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  $u \in \Gamma^*$ .

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  $q' \in \delta_{-1}(q, \perp, a)$  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  $\delta_{-1}(q, \perp, a)$ , 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 IDPDA  $A$ , it would be possible to construct an IDPDA  $B$  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  $\mathcal{C}_A^{\text{init}}(w) = (q_0, w, \varepsilon)$ . The language recognized by  $A$  is defined as

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

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  $C_1, C_2, \dots, C_m$  such that  $C_1 = \mathcal{C}_A^{\text{init}}(w)$ ,  $C_i \vdash_A C_{i+1}$ ,  $1 \leq i \leq m-1$ , and  $C_m$  is of the form  $(q, \varepsilon, u)$  where  $q \in F$ ,  $u \in \Gamma^*$ .

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

The size of an input-driven pushdown automaton  $A$  is determined by the number of its states  $|Q|$  and the number of its stack symbols  $|\Gamma|$ . As it is often difficult to estimate both of these values individually, their sum  $|Q| + |\Gamma|$  is frequently used in the literature as the descriptive complexity measure for IDPDAs. This measure is adopted in two of the results of this paper (Theorems 2 and 3), whereas the other two results (Theorems 1 and 4) give bounds directly on the number of states.

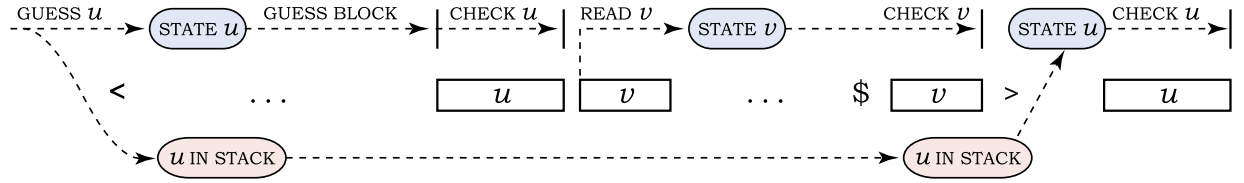


Fig. 1. Recognizing the language  $L_n$  from Proposition 2 by an NIDPDA with  $O(n)$  states.

The following result gives an upper bound on the size blow-up of determinizing an NIDPDA. It was first obtained for the formalism of IDPDAs [8], and later reformulated in terms of nested word automata [4]. A close lower bound on the complexity of this transformation was discovered using nested word automata [4]. In this paper, all results are translated to the terminology of input-driven pushdown automata (that is, we speak about states and stack symbols rather than about “linear states” and “vertical states”).

**Proposition 1.** (See von Braunmühl and Verbeek [8]; Alur and Madhusudan [4].) An NIDPDA over an alphabet  $\tilde{\Sigma} = (\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$  with  $n$  states and with any number of stack symbols can be simulated by a DIDPDA with  $2^{n^2}$  states and  $|\Sigma_{+1}| \cdot 2^{n^2}$  stack symbols.

**Proposition 2.** (See Alur and Madhusudan [4].) Let  $\tilde{\Sigma} = (\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$  with  $\Sigma_{+1} = \{\}$ ,  $\Sigma_{-1} = \{\}$  and  $\Sigma_0 = \{0, 1, \#\}$ , be an alphabet. Then, for each  $n \geq 1$ , there exists a language  $L_n$  over  $\tilde{\Sigma}$ , which is recognized by an NIDPDA with  $C \cdot n$  states and  $n$  stack symbols, for some constant  $C \geq 1$ , while every DIDPDA for  $L_n$  needs at least  $2^{n^2}$  states.

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 construction of the languages  $L_n$  in Proposition 2 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

$$L_n = \{ \langle u_1 \# v_1 \# u_2 \# v_2 \# \dots \# u_\ell \# v_\ell \$ v \rangle u \mid \ell \geq 1, u_i, v_i \in \{0, 1\}^+ \text{ for all } i \in \{1, \dots, \ell\}, u, v \in \{0, 1\}^{\lceil \log n \rceil}, \text{ and there exists } t \in \{1, \dots, \ell\} \text{ with } u = u_t, v = v_t \},$$

and an NIDPDA with  $O(n)$  states recognizes this language as follows. At the first step of the computation, upon reading the left bracket  $\langle$ , 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  $\rangle$  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 Fig. 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  $(q, \varepsilon, u)$ , with  $q \in Q$ ,  $u \in \Gamma^*$  and with the remaining input empty, is called a *terminal configuration*. The *stack height* of an input string  $w$  is defined as the number of symbols in the stack after reading  $w$ : if  $C_A^{\text{init}}(w) \vdash_A^* C$ , where  $C$  is a terminal configuration with stack contents  $u$ , then 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  $\tilde{\Sigma}$  defines a natural matching relation between occurrences of elements of  $\Sigma_{+1}$  and of  $\Sigma_{-1}$ , respectively, by matching each occurrence of an element of  $\Sigma_{+1}$  with the next unmatched occurrence of an element of  $\Sigma_{-1}$  [4]. In an arbitrary computation of an NIDPDA on a string  $w$ , a stack symbol pushed at an occurrence of a symbol of  $\Sigma_{+1}$  will always be popped at a matching occurrence of a symbol from  $\Sigma_{-1}$ , 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  $-1$ .

A string  $w \in \Sigma^*$  is said to be *well nested*, if every occurrence of a symbol of  $\Sigma_{+1}$  has a matching occurrence of a symbol of  $\Sigma_{-1}$ , and vice versa. The computations of an input-driven pushdown automaton on a well nested string  $w$  have the following property: if an NIDPDA  $A$  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  $\Sigma_{+1}$  is

mapped to a string of two symbols from  $\Sigma_{+1}$ , then it breaks the nestedness of brackets, and may map such an input-driven language as  $\{\langle^n\rangle^n \mid n \geq 0\}$  to a non-input-driven language  $\{\langle^{2n}\rangle^n \mid n \geq 0\}$ . 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  $\tilde{\Sigma} = (\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$  be an action alphabet, let  $\Sigma = \Sigma_{+1} \cup \Sigma_{-1} \cup \Sigma_0$ , let  $h: \tilde{\Sigma} \rightarrow \Sigma^*$  be a mapping that satisfies the following conditions:

- (i) for each neutral symbol  $c \in \Sigma_0$ , the image string  $h(c)$  is well nested,
- (ii) for each left bracket  $\langle \in \Sigma_{+1}$ , its image  $h(\langle)$  contains exactly one unmatched symbol of type +1 and no unmatched symbols of type -1,
- (iii) for each right bracket  $\rangle \in \Sigma_{-1}$ , the image  $h(\rangle)$  contains exactly one unmatched symbol of type -1 and no unmatched symbols of type +1.

The mapping  $h$  determines a homomorphism that respects nesting  $\bar{h}: \Sigma^* \rightarrow \Sigma^*$  by setting  $\bar{h}(\varepsilon) = \varepsilon$ , and, for  $\sigma \in \Sigma$ ,  $w \in \Sigma^*$ ,  $\bar{h}(\sigma w) = h(\sigma)\bar{h}(w)$ . For simplicity, this homomorphism  $\bar{h}$  shall be denoted by the symbol  $h$ .

A homomorphism  $h$  is called a  $\Sigma_0$ -relabeling, if  $h$  is the identity on  $\Sigma_{+1}$  and  $\Sigma_{-1}$ , and maps symbols of  $\Sigma_0$  into  $\Sigma_0$  (not necessarily injectively).

In the following, unless otherwise mentioned, a homomorphism always means 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 left bracket as their first symbols, while all images of  $\sigma \in \Sigma_{-1}$  must end with a unique unmatched right bracket. 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  $\tilde{\Sigma} = (\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$  be an action alphabet and, as usual, we denote  $\Sigma = \Sigma_{+1} \cup \Sigma_{-1} \cup \Sigma_0$ . For  $k \geq 1$ , a set of depth  $k$  is a finite set  $S$  of strings over  $\Sigma$  each having stack height  $k$ . A finite set of pairs of strings  $F = \{(x_i, y_i) \mid i = 1, 2, \dots, m\}$  is said to be a set of pairs of depth  $k$ , where  $k \geq 1$ , if each string  $x_i$  has stack height  $k$ .

**Definition 3.** Let  $\tilde{\Sigma} = (\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$  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  $F = \{(x_1, y_1), \dots, (x_m, y_m)\}$  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  $i \in \{1, 2, \dots, m\}$ , and
  - (ii-b) for all  $i, j$  with  $1 \leq i < j \leq m$ ,  $x_i y_j \notin L$  or  $x_j y_i \notin L$ .

**Lemma 1.** (See [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  $\Gamma$ .

- (i) If  $A$  is a DIDPDA and  $S$  is a separator set of depth  $k$  for  $L(A)$  then  $|\Gamma|^k \cdot |Q| \geq |S|$ .
- (ii) If  $L(A)$  has a fooling set  $F$  of depth  $k$ , then  $|\Gamma|^k \cdot |Q| \geq |F|$ .

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  $2^{2(\sqrt{n})}$  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.** (See [34,22].) Let  $L \subseteq \Sigma^*$  be a regular language and let  $F = \{(u_1, v_1), \dots, (u_n, v_n)\}$  with  $n \geq 1$  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  $\text{rank } M(F, L)$  states.

We translate Schmidt's Theorem to the case of unambiguous input-driven pushdown automata. Let  $L$  be an input-driven language and let  $F = \{(x_i, y_i) \mid i = 1, \dots, n\}$  be a set of pairs of depth  $k \geq 1$ . 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  $F = \{(x_i, y_i) \mid i = 1, \dots, n\}$  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  $\Gamma$ . Then

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

**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.

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  $1 \leq i \leq n$ , 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  $i \in \{1, \dots, n\}$ , 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  $x_1, \dots, x_n$ . Recall that since  $x_i$  has stack height  $k$ , the stack contents of  $A$  after reading  $x_i$  is in  $\Gamma^k$ . We note that the unambiguity of  $A$  guarantees that if  $\widehat{M}(F, A)_{\mathbf{f}, y_j} = 1$  for some  $\mathbf{f} \in H_i$ ,  $1 \leq j \leq n$ , 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  $i$ th 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  $\widehat{M}(F, A)$ ,

$$\text{rank } M(F, L) \leq \text{rank } \widehat{M}(F, A).$$

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

#### 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. The upper bound in Proposition 1 applies here, whereas the following lower bound theorem can be regarded as a sharpened version of Proposition 2.

**Theorem 1.** For every  $n \geq 1$ , there exists an input-driven language  $K_n$  recognized by a UIDPDA with  $C \cdot n$  states and  $n$  stack symbols, where  $C$  is a constant independent of  $n$ , such that every DIDPDA for  $K_n$  needs at least  $2^{n^2}$  states.

**Proof.** Let  $\Sigma_0 = \{0, 1, \#, \$\}$ ,  $\Sigma_{+1} = \{ \langle \rangle \}$ ,  $\Sigma_{-1} = \{ \rangle \}$ . For  $n \geq 1$  define the language

$$K_n = \{ \langle x_0 \# x_1 \# \dots \# x_\ell \$ v \rangle u \mid x_i \in \{0, 1\}^* \text{ for all } i \in \{0, 1, \dots, \ell\}, u, v \in \{0, 1\}^{\lceil \log n \rceil}, \text{ and the bit number } (v)_2 \text{ in } x_{(u)_2} \text{ is } 1 \}$$

The language  $K_n$  is recognized unambiguously as follows. In the following discussion, we assume that the input string is in the language  $L_0 = \{ \langle \{0, 1\}^* \# \{0, 1\}^* \$ \{0, 1\}^* \rangle \{0, 1\}^* \}$ , 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 Fig. 2. At the first input symbol  $\langle \in \Sigma_{+1}$ , 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  $(v)_2$ -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  $\rangle \in \Sigma_{-1}$ , the computation verifies, using the information in the symbol popped from the stack, that the binary string after the symbol  $\rangle$  is equal to  $u$ . All nondeterministic choices in a successful computation are pre-determined by the suffix of the input  $v \rangle u$ , and for each input there can be only one accepting computation. This computation is illustrated in Fig. 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

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

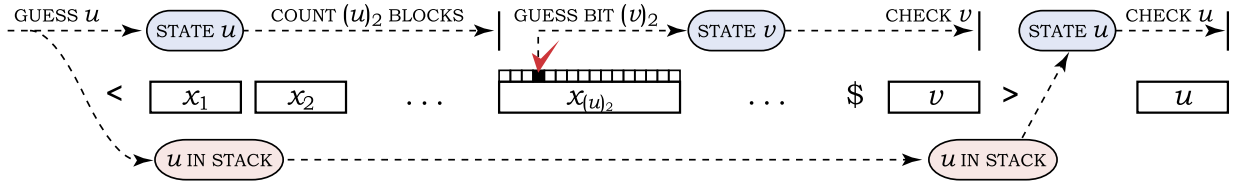


Fig. 2. Recognizing the language  $K_n$  in the proof of Theorem 1 by a UIDPDA with  $\Theta(n)$  states.

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, which is constructed as follows. 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  $\Gamma$ , then  $|Q| \cdot |\Gamma| \geq |R|$ . Furthermore, as all strings in  $K_n$  begin with a left bracket and contain no other left brackets, the automaton  $A$  has a chance to push a symbol to the stack only in its initial state, and so one can assume that  $|\Gamma| = 1$ . Therefore,  $|Q| \geq |R|$ , that is, any DIDPDA recognizing  $K_n$  needs at least  $2^{n^2}$  states.  $\square$

Similarly to Proposition 2, the above Theorem 1 does not establish any lower bounds on the number of stack symbols needed for determinizing a UIDPDA. In fact, the witness languages  $K_n$  used in the proof are recognized using a single dummy stack symbol.

Recently, Okhotin et al. [27] gave a more refined construction of witness languages for the NIDPDA determinization blowup, proving that every DIDPDA recognizing those languages must have a certain number of states and a certain number of stack symbols at the same time. The strings in those witness languages contain an unbounded number of matching brackets from  $\Sigma_{+1}$  and  $\Sigma_{-1}$ . Proving such a refined lower bound for the UIDPDA to DIDPDA transformation is left as an open problem.

## 5. From nondeterministic to unambiguous

The purpose of this section is to prove 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. This time, this will be a lower bound of a weaker form, that is, on the sum of the number of states and the number of stack symbols.

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

$$\Sigma_0 = \{a, b, \#\}, \quad \Sigma_{+1} = \{ \langle \}, \quad \Sigma_{-1} = \{ \rangle \}$$

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

$$L_n = \left\{ \langle a^{k_1} b^{\ell_1} a^{k_2} b^{\ell_2} \dots a^{k_m} b^{\ell_m} \# b^{\ell'_1} a^{k'_1} b^{\ell'_2} a^{k'_2} \dots b^{\ell'_{m'}} a^{k'_{m'}} \rangle \mid \right. \\ \left. m, m' \geq 1, k_i, \ell_i, k'_j, \ell'_j \geq 1, \exists s, t: k_s = k'_t \leq n, \ell_s = \ell'_t \leq n \right\}. \quad (1)$$

**Lemma 3.** For every  $n \geq 1$ , the language  $L_n$  is recognized by an NIDPDA with  $C \cdot n$  states and  $n$  stack symbols, where  $C \geq 1$  is a constant independent of  $n$ . At the same time, for every UIDPDA recognizing  $L_n$ , the product of the number of its states by the number of its stack symbols must be at least  $2^{\lfloor \frac{n^2}{2} \rfloor}$ .

**Proof.** An NIDPDA recognizing this language is defined as follows. Its computation on any well-formed input in  $\langle \{a, b\}^* \# \{a, b\}^* \rangle$  aims to detect a pattern of the form  $\langle \dots uv \dots \# \dots vu \dots \rangle$ . Upon reading the left bracket  $\langle \in \Sigma_{+1}$ , the NIDPDA guesses an integer  $i \in \{1, \dots, n\}$  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 Fig. 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 left bracket  $\langle$ , then the first  $b$  is missing; similarly, if  $w$  occurs directly before the middle marker  $\#$ , then the last  $a$  is missing. Next, the computation memorizes the integer  $s$  in the state and proceeds to the part after the middle marker  $\#$ , where another substring in  $b^* a^* \rangle$  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 right bracket  $\rangle \in \Sigma_{-1}$ , 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.

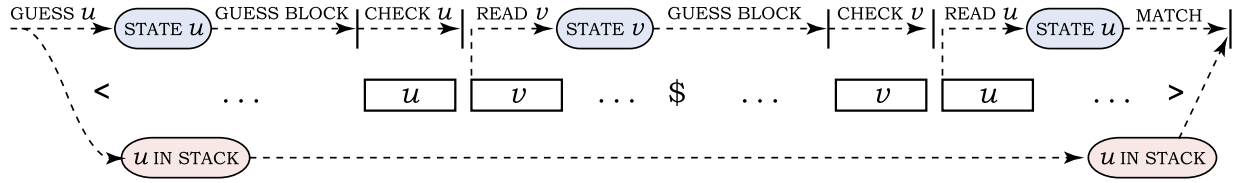


Fig. 3. Recognizing the language  $L_n$  in the proof of Lemma 3 by an NIDPDA with  $\Theta(n)$  states.

For each set  $R \subseteq [1, n] \times [1, n]$ , define the strings  $x_R = a^{i_1}b^{j_1} \dots a^{i_m}b^{j_m}$  and  $y_R = b^{j_1}a^{i_1} \dots b^{j_m}a^{i_m}$ , where  $R = \{(i_1, j_1), \dots, (i_m, j_m)\}$ . 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  $(\langle x_R \#, y_{f(R)} \rangle)$ , 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

$$P(1,0), P(1,1), P(2,0), P(2,1), \dots, P(z,0), P(z,1) \quad (2)$$

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  $\mathbf{v} = b_1b_2 \dots b_z$  with  $b_i \in \{0, 1\}$ , denote by  $\bar{\mathbf{v}} = b'_1b'_2 \dots b'_z$  the inverted string, where  $b'_i = 1$  if  $b_i = 0$  and  $b'_i = 0$  if  $b_i = 1$  for all  $i \in \{1, \dots, z\}$ . Also for  $\mathbf{v} = \{0, 1\}^z$ , define the set  $R_{\mathbf{v}} \subseteq [1, n] \times [1, n]$  by setting

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

Now define

$$F = \{(\langle x_{R_{\mathbf{v}}} \#, y_{R_{\bar{\mathbf{v}}}} \rangle) \mid \mathbf{v} \in \{0, 1\}^z\},$$

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

The above choices guarantee that each string  $\langle x_{R_{\mathbf{v}}} \#, y_{R_{\bar{\mathbf{v}}}} \rangle$  is not in  $L_n$ , while  $\langle x_{R_{\mathbf{v}}} \#, y_{R_{\bar{\mathbf{v}}}} \rangle \in L_n$  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  $\Gamma$  recognizing the language  $L_n$  must satisfy  $|Q| \cdot |\Gamma| \geq 2^{\lfloor \frac{n^2}{2} \rfloor}$ .  $\square$

If the product of the number of states  $|Q|$  and the number of stack symbols  $|\Gamma|$  in any NIDPDA recognizing this language is at least  $2^{\lfloor \frac{n^2}{2} \rfloor}$ , then one can infer a lower bound on their sum. Using the sum  $|Q| + |\Gamma|$  as a descriptiveness complexity measure, called the *combined size*, leads to the following succinctness tradeoff between UIDPDAs and NIDPDAs.

**Theorem 2.** Every NIDPDA of combined size  $n$  has an equivalent UIDPDA of combined size  $O(2^{n^2})$ , where the constant factor depends on the alphabet. At the same time, for every  $n \geq 1$ , there exists a language recognized by an NIDPDA of combined size  $C \cdot n$ , where  $C \geq 1$  is a constant, while every UIDPDA recognizing this language has to be of combined size at least  $2^{\frac{n^2}{4}}$ .

**Proof.** The upper bound is by Proposition 1. The witness languages for the lower bound are given by Lemma 3, which asserts that for every UIDPDA recognizing  $L_n$ , if  $Q$  is its set of states and  $\Gamma$  is its pushdown alphabet, then  $|Q| \cdot |\Gamma| \geq 2^{\lfloor \frac{n^2}{2} \rfloor}$ . In order to obtain a lower bound on the sum  $|Q| + |\Gamma|$ , consider that the arithmetic mean of two numbers is greater than or equal to their geometric mean, that is,  $\frac{x+y}{2} \geq \sqrt{xy}$  for all  $x, y \geq 1$ . Then,

$$|Q| + |\Gamma| \geq 2\sqrt{|Q| \cdot |\Gamma|} \geq 2\sqrt{2^{\lfloor \frac{n^2}{2} \rfloor}} \geq 2^{\lfloor \frac{n^2}{4} \rfloor + 1} > 2^{\frac{n^2}{4}},$$

as desired.  $\square$

## 6. Complementing nondeterministic automata

The next result concerns the complementation operation for NIDPDA. Consider again the languages  $L_n$  with  $n \geq 1$ , defined in (1), which can be recognized by NIDPDA with  $O(n)$  states and  $n$  stack symbols. The following lemma shows that for every



NIDPDA recognizing the complement of this language, the product of the number of its states and the number of its stack symbols must be at least  $2^{\Theta(n^2)}$ .

**Lemma 4.** *Let  $A$  be an NIDPDA with a set of states  $Q$  and a pushdown alphabet  $\Gamma$ , which recognizes the language  $\overline{L_n}$ , where  $L_n$  is the language (1) defined in Section 5. Then  $|Q| \cdot |\Gamma| \geq 2^{n^2}$ .*

**Proof.** The proof is by constructing a fooling set of depth 1 for the language  $\overline{L_n}$ , as in Definition 3(ii). The construction of the fooling set is similar to the construction used in Lemma 3, however, now the fooling set can be based on arbitrary binary relations on  $[1, n]$ .

As in the proof of Lemma 3, for each set  $R \subseteq [1, n] \times [1, n]$ , define the strings  $x_R \in (a^+b^+)^*$  and  $y_R \in (b^+a^+)^*$  that list the elements of  $R$  in an arbitrary order. Define a set of pairs  $S$  to consist of all pairs  $(\langle x_R \#, y_{\overline{R}} \rangle)$ , where  $R \subseteq [1, n] \times [1, n]$ .

To see that  $S$  is a fooling set for  $\overline{L_n}$ , first, consider the concatenation of every pair, which is  $\langle x_R \# y_{\overline{R}} \rangle$ , for some set  $R \subseteq [1, n] \times [1, n]$ . Since the sets  $R$  and  $\overline{R}$  have no common pairs, the concatenation is not in  $L_n$ , that is, it belongs to the complement  $\overline{L_n}$ , as required by Definition 3(ii-a).

Let  $R_1, R_2 \subseteq [1, n] \times [1, n]$ , be any two distinct sets, and consider the two cross-concatenations of the pairs  $(\langle x_{R_1} \#, y_{\overline{R_2}} \rangle)$  and  $(\langle x_{R_2} \#, y_{\overline{R_1}} \rangle)$ . As  $R_1 \neq R_2$  at least one of the intersections  $R_1 \cap \overline{R_2}$  and  $R_2 \cap \overline{R_1}$  must be non-empty. Then, at least one of the strings in question,  $\langle x_{R_1} \# y_{\overline{R_2}} \rangle$  and  $\langle x_{R_2} \# y_{\overline{R_1}} \rangle$ , belongs to the language  $L_n$ , and, accordingly, is not in the complement  $\overline{L_n}$ . Thus, the condition in Definition 3(ii-b) is satisfied.

This means that  $S$  is a fooling set of depth 1 for  $\overline{L_n}$ , which contains  $2^{n^2}$  pairs. The claim follows by Lemma 1(ii).  $\square$

Similarly to Theorem 2, this lower bound on the product of the number of states  $|Q|$  and the number of stack symbols  $|\Gamma|$  in any NIDPDA recognizing the complement  $\overline{L}$  leads to a lower bound on their sum  $|Q| + |\Gamma|$ . Using this sum as a descriptonal complexity measure, called the *combined size*, one can obtain the following bounds on the complexity of complementing NIDPDAs.

**Theorem 3.** *For every language recognized by an NIDPDA of combined size  $n$ , its complement is recognized by an NIDPDA of combined size  $O(2^{n^2})$ . At the same time, for every  $n \geq 1$ , there exists a language recognized by an NIDPDA of combined size  $C \cdot n$ , for some constant  $C \geq 1$ , such that every NIDPDA recognizing its complement is of combined size at least  $2^{\frac{n^2}{2}+1}$ .*

**Proof.** Proposition 1 gives an upper bound  $2^{O(n^2)}$ . By the first part of Lemma 3, the language  $L_n$  can be recognized by an NIDPDA of combined size  $O(n)$ . At the same time, every NIDPDA recognizing its complement must have  $|Q| \cdot |\Gamma| \geq 2^{n^2}$  by Lemma 4. This lower bound on the product of the number of states by the number of stack symbols implies the following lower bound on their sum, by the same argument as in the proof of Theorem 2:

$$|Q| + |\Gamma| \geq 2\sqrt{|Q| \cdot |\Gamma|} \geq 2\sqrt{2^{n^2}} = 2^{\frac{n^2}{2}+1}. \quad \square$$

## 7. Homomorphic images of deterministic automata

The last result of this paper is a tight bound on the deterministic state complexity of homomorphisms. Throughout this section, all homomorphisms are assumed to *respect nesting*, as per Definition 2.

The witness languages are defined as a modification of the languages (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  $a'$  and  $b'$ . Let  $\Sigma_0 = \{0, 1, a, b, a', b', \#\}$ ,  $\Sigma_{+1} = \{\langle \rangle, \Sigma_{-1} = \{\rangle\}$ . For  $n \geq 1$ , define the language  $L'_n$  to consist of all strings of the form

$$w \langle (a^+b^+)^* a' a^{(w)_2-1} b^\ell (a^+b^+)^* \# (b^+a^+)^* b' b^{\ell-1} a^{(w)_2} (b^+a^+)^* \rangle, \quad (3)$$

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

**Lemma 5.** *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  $\langle \in \Sigma_{+1}$ , stores this number in the stack as a single stack symbol. Then the computation checks that the length of the sequence of  $a$ s following the first occurrence of  $a'$  is equal to  $(w)_2 - 1$ . After this, the computation memorizes the number of  $b$ s immediately following the  $a$ s (that is,  $\ell$ ) and checks that the number of  $b$ s occurring after the symbol  $b'$  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  $\rangle$ , the computation checks that the number stored in the internal state is equal to the number  $(w)_2$  popped from the stack.  $\square$

Let  $h$  be the homomorphism that maps  $a'$  to  $a$ ,  $b'$  to  $b$  and 1 to 0, while mapping the rest of the symbols to themselves. Note that  $h$  is a  $\Sigma_0$ -relabeling.

**Lemma 6.** *Every DIDPDA recognizing  $h(L'_n)$  has to have at least  $2^{n^2}$  states.*

**Proof.** The homomorphic image of  $L'_n$  under  $h$  equals

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

where  $L_n$  is defined in (1). For each relation  $R \subseteq [1, n] \times [1, n]$ , define the strings  $x_R$  and  $y_R$  as in the proof of Lemma 4. Let

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

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

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

This means that  $S_1$  is a separator set of depth 1 for  $h(L'_n)$  and, by Lemma 1(i), for every DIDPDA  $A$  recognizing  $h(L'_n)$ , if  $Q$  is its set of states and  $\Gamma$  is its stack alphabet, then  $|Q| \cdot |\Gamma| \geq 2^{n^2}$ .

As in the proof of Theorem 1, consider that all strings in the language  $h(L'_n)$  are of the form  $0^{\lceil \log n \rceil} \langle x \rangle$ , with  $x \in \Sigma_0$ , that is, contain a single pair of brackets, which is preceded by the same constant string. Thus,  $A$  shall push the same stack symbol on all input strings beginning with the valid prefix, and accordingly, the stack alphabet can be assumed to be a singleton. Substituting  $|\Gamma| = 1$  in the above expression leads to the desired upper bound  $|Q| \geq 2^{n^2}$ .  $\square$

Now we can state a tight bound for the descriptonal 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.** *For every homomorphism  $h$  and for every DIDPDA  $A$  over an alphabet  $(\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$ , with  $n$  states and with any number of stack symbols, the language  $h(L(A))$  can be recognized by a DIDPDA with  $2^{C \cdot n^2}$  states and  $|\Sigma_{+1}| \cdot 2^{C \cdot n^2}$  stack symbols, where  $C > 0$  is a constant that depends only on  $h$ .*

*There exists a  $\Sigma_0$ -relabeling  $h$  and input-driven languages  $L'_n$ , with  $n \geq 1$ , each recognized by a DIDPDA with  $O(n)$  states and  $n$  stack symbols, such that any DIDPDA for  $h(L'_n)$  needs at least  $2^{n^2}$  states.*

**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  $h(L(A))$ ; even though the definition of a homomorphism used by Alur and Madhusudan [4] differs slightly from the one adopted in this paper, the same construction works here. 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  $a_i \in \tilde{\Sigma}$ , 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  $A$  has  $n$  states, then  $B$  will have  $C \cdot n$  states, where  $C$  is a constant depending on  $h$  but independent of  $A$ . Then the upper bound on the size of the equivalent DIDPDA follows from Proposition 1.

The lower bound is established in Lemma 6.  $\square$

## 8. Conclusion

It has been shown that 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. The unambiguous-to-deterministic blow-up has been established with respect to the number of states in the automaton, whereas the nondeterministic-to-unambiguous lower bound uses a weaker descriptonal complexity measure: the sum of the number of states and the number of stack symbols. Both descriptonal 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 a general nondeterministic input-driven pushdown automaton, the precise multiplicative constant in the exponent is also not known [4]. It would also be interesting to determine the blowups in terms of states and in terms of stack symbols separately.

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, pp. 151–160.
- [2] R. Alur, V. Kumar, P. Madhusudan, M. Viswanathan, Congruences for visibly pushdown languages, in: Proc. 32nd ICALP, in: Lect. Notes Comput. Sci., vol. 3580, Springer, 2005, pp. 1102–1114.
- [3] R. Alur, P. Madhusudan, Visibly pushdown languages, in: Proc. ACM Symposium on Theory of Computing, STOC 2004, Chicago, USA, 2004, pp. 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, in: Lect. Notes Comput. Sci., vol. 4596, Springer, 2007, pp. 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, Log. Methods Comput. Sci. 4 (4) (2008), paper 16.
- [8] B. von Braunmühl, R. Verbeek, Input-driven languages are recognized in  $\log n$  space, North-Holl. Math. Stud. 102 (1985) 1–19.
- [9] P. Chervet, I. Walukiewicz, Minimizing variants of visibly pushdown automata, in: Proc. MFCS 2007, in: Lect. Notes Comput. Sci., vol. 4708, Springer, 2007, pp. 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. Process. Lett. 109 (2008) 13–17.
- [12] J. Goldstine, H. Leung, D. Wotschke, On the relation between ambiguity and nondeterminism in finite automata, Inform. and Comput. 100 (1992) 261–270.
- [13] V. Geffert, C. Mereghetti, G. Pighizzini, Complementing two-way finite automata, Inform. and 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 descriptonal complexity of regular languages, Internat. J. Found. Comput. Sci. 14 (2003) 1087–1102.
- [16] M. Holzer, M. Kutrib, Descriptonal and computational complexity of finite automata—a survey, Inform. and 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. and 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: Proc. Mathematical Foundations of Computer Science, MFCS, Gdańsk, Poland, August 29–September 2, 2005, in: Lect. Notes Comput. Sci., vol. 3618, Springer, 2005, pp. 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, in: Lect. Notes Comput. Sci., vol. 6795, Springer, 2011, pp. 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, Descriptonal complexity of NFA of different ambiguity, Internat. J. Found. Comput. Sci. 16 (5) (2005) 975–984.
- [23] K. Mehlhorn, Pebbling mountain ranges and its application to DCFL-recognition, in: Proc. 7th ICALP, in: Lect. Notes Comput. Sci., vol. 85, Springer, 1980, pp. 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, in: Lect. Notes Comput. Sci., vol. 1530, Springer, 2004, pp. 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, in: Lect. Notes Comput. Sci., vol. 6543, Springer, 2011, pp. 431–443.
- [26] A. Okhotin, Unambiguous finite automata over a unary alphabet, Inform. and Comput. 212 (2012) 15–36.
- [27] A. Okhotin, X. Piao, K. Salomaa, Descriptonal complexity of input-driven pushdown automata, in: H. Bordihn, M. Kutrib, B. Truthe (Eds.), Dassow Festschrift 2012, in: Lect. Notes Comput. Sci., vol. 7300, Springer-Verlag, 2012, pp. 186–206.
- [28] A. Okhotin, K. Salomaa, State complexity of operations on input-driven pushdown automata, in: Mathematical Foundations of Computer Science, MFCS, in: Lect. Notes Comput. Sci., vol. 6907, Springer-Verlag, 2011, pp. 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, vols. 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. and 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.