Input-Driven Stack Automata

Suna Bensch, Markus Holzer, Martin Kutrib, Andreas Malcher

Input-Driven Stack Automata

Suna Bensch[1], Markus Holzer[2], Martin Kutrib[2], and Andreas Malcher[2]

[1] Department of Computing Science,
Umeå University, 90187 Umeå, Sweden
email: suna@cs.umu.se
[2] Institut für Informatik, Universität Giessen,
Arndtstr. 2, 35392 Giessen, Germany
email: {holzer,kutrib,malcher}@informatik.uni-giessen.de

Abstract. We introduce and investigate input-driven stack automata, which are a generalization of input-driven pushdown automata that recently became popular under the name visibly pushdown automata. Basically, the idea is that the input letters uniquely determine the operations on the pushdown store. This can nicely be generalized to stack automata by further types of input letters which are responsible for moving the stack pointer up or down. While visibly pushdown languages share many desirable properties with regular languages, input-driven stack automata languages do not necessarily so. We prove that deterministic and nondeterministic input-driven stack automata have different computational power, which shows in passing that one cannot construct a deterministic input-driven stack automaton from a nondeterministic one. We study the computational capacity of these devices. Moreover, it is shown that the membership problem for nondeterministic input-driven stack automata languages is NP-complete.

1 Introduction

Finite automata have intensively been studied and, moreover, have been extended in several different ways. Typical extensions in the view of [8] are pushdown tapes [4], stack tapes [9], or Turing tapes. The investigations in [8] led to a rich theory of abstract families of automata, which is the equivalent to the theory of abstract families of languages (see, for example, [18]). On the other hand, slight extensions to finite automata such as a one-turn pushdown tape lead to machine models that can no longer be determinized, that is, the nondeterministic machine model is more powerful than the deterministic one. Moreover, fundamental problems such as membership become more complicated than for languages accepted by finite automata. For example, the equivalence problem turns out to be undecidable, while for regular languages this problem is decidable, and its complexity depends on the machine type used (deterministic or nondeterministic finite automata).

Recently a pushdown automaton model, called visibly pushdown automaton, was popularized by [1], which shares many desirable properties with regular

languages, but still is powerful enough to describe important context-free-like behavior. The idea on visibly pushdown automata is that the input letters uniquely determine whether the automaton pushes a symbol, pops a symbol, or leaves the pushdown unchanged. Such devices date back to the seminal paper [15] and its follow-ups [2] and [6], where this machine model is called input-driven pushdown automaton. One of the most important properties on visibly pushdown automata languages or, equivalently, input-driven pushdown automata languages is that deterministic and nondeterministic automata are equally powerful. Moreover, the language class accepted is closed under almost all basic operations in formal language theory. Since the recent paper [1], visibly pushdown automata are a vivid area of research, which can be seen by the amount of literature, for example, [1, 3, 5, 10, 16, 17]. In some of these papers yet another name is used for visibly pushdown automata, namely nested word automata, which may lead to some confusion.

Here we generalize the idea of input-driven pushdown automata to input-driven stack automata. Since the main difference between a pushdown and a stack is that the latter storage type is also allowed to read information from the inside of the stack and not only from the top, the idea that input letters control the stack behavior easily applies. Hence, in addition to the letters that make the automaton push, pop, or leave the stack unchanged, two new types of letters that allow the movement of the stack pointer up or down are introduced. This leads us to the strong version of a input-driven stack automaton. Relaxing the condition on being input-driven when reading the stack contents, gives the basic idea of a weak input-driven stack automaton. We compare both automata models and show that the strong version is strictly less powerful than the corresponding weak version for deterministic devices. Moreover, when staying with the same model, nondeterminism turns out to be more powerful than determinism, which shows in passing that in both cases determinization is not possible. This sharply contrasts the situation for input-driven pushdown automata. Concerning decidability questions, we would like to note that the results in [9] imply that emptiness is decidable for nondeterministic input-driven stack automata and equivalence with regular languages is decidable for deterministic input-driven stack automata. Finally, we also show that the fixed membership problem for input-driven stack automata languages, even for the strong automaton model, has the same complexity as for languages accepted by ordinary stack automata, namely it is NP-complete, and therefore intractable. This again is in sharp contrast to the situation on input-driven pushdown automata languages, whose membership problem is NC^1-complete [6] while ordinary pushdown automata languages are LOGCFL-complete [20].

2 Preliminaries and Definitions

We write Σ^* for the set of all words over the finite alphabet Σ. The empty word is denoted by λ, and we set $\Sigma^+ = \Sigma^* \setminus \{\lambda\}$. The reversal of a word w is denoted

by w^R and for the length of w we write $|w|$. We use $\subseteq$ for inclusions and $\subset$ for strict inclusions.

A nondeterministic one-way stack automaton is a classical nondeterministic pushdown automaton which is enhanced with an additional pointer that is allowed to move inside the stack without altering the stack contents. In this way it is possible to read but not to change information which is stored on the stack. Such a stack automaton is called input-driven, if the next input symbol defines the next action on the stack, that is, pushing a symbol onto the stack, popping a symbol from the stack, changing the internal state without changing the stack, or moving inside the stack by going up or down. To this end, we assume that the input alphabet Σ is partitioned into the sets Σ_c, Σ_r, Σ_i, Σ_u, and Σ_d, that control the actions push (call), pop (return), state change without changing the stack (internal), and up and down movement of the stack pointer. A formal definition reads as follows.

Definition 1. *A* strong nondeterministic one-way input-driven stack automaton (1sNVSA) *is a system* $M = \langle Q, \Sigma, \Gamma, \bot, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$*, where*

1. Q *is the finite set of* internal states*,*
2. Σ *is the finite set of* input symbols *consisting of the disjoint union of sets* Σ_c*,* Σ_r*,* Σ_i*,* Σ_u*, and* Σ_d*,*
3. Γ *is the finite set of* stack symbols*,*
4. $\bot \in \Gamma$ *is the* initial stack *or* bottom-of-stack symbol*,*
5. Γ' *is a marked copy of* Γ*, that is* $\Gamma' = \{\, a' \mid a \in \Gamma \,\}$*, where the symbol* $\bot'$ *is denoted by* $\bot$*,*
6. $q_0 \in Q$ *is the* initial state*,*
7. $F \subseteq Q$ *is the set of* accepting states*, and*
8. δ_c *is the partial transition function mapping* $Q \times \Sigma_c$ *into the subsets of* $Q \times (\Gamma' \setminus \{\bot\})$*,*
9. δ_r *is the partial transition function mapping* $Q \times \Sigma_r \times \Gamma'$ *into the subsets of* Q*,*
10. δ_i *is the partial transition function mapping* $Q \times \Sigma_i$ *into the subsets of* Q*,*
11. δ_d *is the partial transition function mapping* $Q \times \Sigma_d \times (\Gamma \cup \Gamma')$ *into the subsets of* Q*,*
12. δ_u *is the partial transition function mapping* $Q \times \Sigma_u \times (\Gamma \cup \Gamma')$ *into the subsets of* Q*.*

A *configuration* of a 1sNVSA $M = \langle Q, \Sigma, \Gamma, \bot, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$ at some time $t \geq 0$ is a quadruple $c_t = (q, w, s, p)$, where $q \in Q$ is the current state, $w \in \Sigma^*$ is the unread part of the input, $s \in \Gamma'\Gamma^*\bot \cup \{\bot\}$ gives the current stack contents, and $1 \leq p \leq |s|$ gives the current position of the stack pointer. Let $s = s_n s_{n-1} \cdots s_1$ denote the stack contents. Consider the projection $[\cdot] : (\Gamma \cup \Gamma')^+ \to (\Gamma \cup \Gamma')$ such that $s[p] = s_p$, for $1 \leq p \leq n$. Furthermore, let φ be a mapping which marks the first letter of a string in Γ^+, that is, $\varphi : \Gamma^+ \to \Gamma'\Gamma^*$ such that $\varphi(a_1 a_2 \cdots a_n) = a'_1 a_2 \cdots a_n$. By definition, $\varphi(\bot) = \bot$.

The *initial configuration* for input w is set to $(q_0, w, \bot, 1)$. During its course of computation, M runs through a sequence of configurations. One step from a

configuration to its successor configuration is denoted by $\vdash$. Let $a \in \Sigma$, $w \in \Sigma^*$, $s \in \Gamma'\Gamma^*\bot \cup \{\bot\}$, $t \in \Gamma^*\bot$, $1 \leq p \leq |s|$, and $Z \in \Gamma$. We set

1. $(q, aw, s, p) \vdash (q', w, Z'\varphi^{-1}(s), |s| + 1)$, if $a \in \Sigma_c$ and $(q', Z) \in \delta_c(q, a)$,
2. $(q, aw, Z't, p) \vdash (q', w, \varphi(t), |t|)$, if $a \in \Sigma_r$ and $q' \in \delta_r(q, a, Z')$,
3. $(q, aw, \bot, 1) \vdash (q', w, \bot, 1)$, if $a \in \Sigma_r$ and $q' \in \delta_r(q, a, \bot)$,
4. $(q, aw, s, p) \vdash (q', w, s, p)$, if $a \in \Sigma_i$ and $q' \in \delta_i(q, a)$,
5. $(q, aw, s, p) \vdash (q', w, s, p + 1)$, if $a \in \Sigma_u$, $q' \in \delta_u(q, a, s[p])$, and $s[p] \notin \Gamma'$,
6. $(q, aw, s, p) \vdash (q', w, s, p)$, if $a \in \Sigma_u$, $q' \in \delta_u(q, a, s[p])$, and $s[p] \in \Gamma'$,
7. $(q, aw, s, p) \vdash (q', w, s, p - 1)$, if $a \in \Sigma_d$, $q' \in \delta_d(q, a, s[p])$, and $s[p] \neq \bot$,
8. $(q, aw, s, p) \vdash (q', w, s, p)$, if $a \in \Sigma_d$, $q' \in \delta_d(q, a, s[p])$, and $s[p] = \bot$.

As usual, we define the reflexive, transitive closure of $\vdash$ by $\vdash^*$.

So, the pushing of a symbol onto the stack is described by Σ_c and δ_c, and the popping of a symbol is described by Σ_r and δ_r. With the help of the mappings φ and φ^{-1} it is possible to mark the new topmost symbol suitably. The internal change of the state without altering the stack contents is described by Σ_i and δ_i. We remark that δ_c and δ_i do not depend on the topmost stack symbol, but only on the current state and input symbol. This is not a serious restriction since every automaton can be modified in such a way that the topmost stack symbol is additionally stored in the state set. In this context, the question may arise of how a state can store the new topmost stack symbol in case of popping. This can be solved by a similar construction as given in [12], where every pushdown automaton is converted to an equivalent pushdown automaton such that every stack symbol is a pair of stack symbols consisting of the symbol on the stack and its immediate predecessor.

The moves inside the stack are described by Σ_d, δ_d and Σ_u, δ_u, respectively. Up-moves at the top of the stack and down-moves at the bottom of the stack can only change the state, but do not affect the position of the stack pointer. So, the pointer can never go below the bottom and beyond the top of the stack. To ensure the latter the topmost stack symbol is suitably marked. By definition, transition functions δ_c and δ_r can only be applied if the stack pointer is at the topmost stack symbol. Thus, we stipulate the following behavior: if δ_c or δ_r have to be applied and the stack pointer is inside the stack, then the stack pointer is set to the topmost symbol, and the new symbol is pushed onto the stack or the topmost symbol is popped off the stack. The bottom-of-stack symbol $\bot$ can neither be pushed onto nor be popped from the stack.

The language accepted by a 1sNVSA is precisely the set of words w such that there is some computation beginning with the initial configuration and ending in a configuration in which the whole input is read and an accepting state is entered:

$$L(M) = \{\, w \in \Sigma^* \mid (q_0, w, \bot, 1) \vdash^* (q, \lambda, s, p) \text{ with } q \in F, \\ s \in \Gamma'\Gamma^*\bot \cup \{\bot\}, \text{ and } 1 \leq p \leq |s| \,\}.$$

If in any case each δ_r, δ_c, δ_i, δ_u, and δ_d is either undefined or a singleton, then the stack automaton is said to be *deterministic*. Strong deterministic stack

automata are denoted by 1sDVSA. In case that no symbol is ever popped from the stack, that is, $\delta_r = \emptyset$, the stack automaton is said to be *non-erasing*. Strong nondeterministic and deterministic non-erasing stack automata are denoted by 1sNENVSA and 1sNEDVSA. The family of all languages accepted by an input-driven stack automaton of some type X is denoted by $\mathscr{L}(X)$.

In order to clarify our notion we continue with an example.

Example 2. The non-context-free language $\{ a^n b^n c^{n+1} \mid n \geq 1 \}$ is accepted by the 1sNEDVSA $M = \langle \{q_0, q_1, q_2, q_3\}, \Sigma, \{A, \bot\}, \bot, q_0, \{q_3\}, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$, where $\Sigma_c = \{a\}$, $\Sigma_u = \{c\}$, $\Sigma_d = \{b\}$, and $\Sigma_r = \Sigma_i = \emptyset$. The transition functions δ_r and δ_i are undefined, and δ_c, δ_u, and δ_d are as follows.

(1) $\delta_c(q_0, a) = (q_0, A')$	(4) $\delta_u(q_1, c, \bot) = q_2$
(2) $\delta_d(q_0, b, A') = q_1$	(5) $\delta_u(q_2, c, A) = q_2$
(3) $\delta_d(q_1, b, A) = q_1$	(6) $\delta_u(q_2, c, A') = q_3$

Since δ_r is undefined, M is non-erasing. An input is accepted only if M eventually enters state q_3. To this end, it must be in state q_2. Similarly, to get into state q_2 it must be in state q_1, and the only possibility to change into state q_1 is from q_0.

If the input does not begin with an a, the computation blocks and rejects immediately. So, any accepting computation starts with a sequence of transitions (1) reading a prefix of the form a^n, for $n \geq 1$. This yields a configuration $(q_0, w_1, A'A^{n-1}\bot, n+1)$. Since for input symbol c no transition is defined from q_0, the remaining input w_1 must have a prefix of the form b^m, for $m \geq 1$. Therefore, M applies one transition (2) and, subsequently, tries to apply $m-1$ transitions (3). If $m < n$, this yields a configuration $(q_1, w_2, A'A^{n-1}\bot, p)$, where $2 \leq p \leq n$, and M blocks and rejects if an a or a c follows. Similarly, if $m > n$, after reading b^n a configuration $(q_1, w_2, A'A^{n-1}\bot, 1)$ is reached, on which M blocks and rejects when trying to read the next b. Therefore, in any accepting computation w_1 must begin with exactly n copies of b. Since for input symbol a no transition is defined from q_1, the remaining input w_2 must have a prefix of the form c^ℓ, for $\ell \geq 1$. Therefore, M applies one transition (4) and, subsequently, applies transitions (5). If $\ell < n$, this yields a configuration $(q_2, w_3, A'A^{n-1}\bot, p)$, where $2 \leq p \leq n$, and M blocks and rejects if an a or a b follows, and does not accept if the input has been consumed. If $\ell \geq n$, a configuration $(q_2, w_3, A'A^{n-1}\bot, n+1)$ is reached. Next, if $\ell = n+1$ then M applies transition (6) and accepts the input $a^n b^b c^{n+1}$. For $\ell > n+1$ the computation blocks. □

Next, we introduce weak variants of input-driven stack automata, for which moves inside the stack are not necessarily input-driven. To this end, we have to extend the domain of δ_d and δ_u appropriately and to adapt the derivation relation $\vdash$ accordingly. First we explain how to modify the transition functions in the definition of input-driven stack automata, where items 11 and 12 are changed to

11′. δ_d is the partial transition function mapping $Q \times \Sigma \times (\Gamma \cup \Gamma')$ into the subsets of Q,

12′. δ_u is the partial transition function mapping $Q \times \Sigma \times (\Gamma \cup \Gamma')$ into the subsets of Q.

In the weak mode, the stack can only by entered by reading an input symbol from Σ_d. Being inside the stack, the pointer may move up and down for any input symbol. When the top of the stack is reached, the stack is left and any new entering needs another input symbol from Σ_d. So, Σ_u is not necessary, but we keep it for the sake of compatibility. For the weak mode, the relation $\vdash$ is adapted by replacing items 5 to 8 by the following ones:

5′. $(q, aw, s, p) \vdash (q', w, s, p+1)$, if $a \in \Sigma$, $q' \in \delta_u(q, a, s[p])$, and $s[p] \notin \Gamma'$,
6′. $(q, aw, s, p) \vdash (q', w, s, p)$, if $a \in \Sigma_u$, $q' \in \delta_u(q, a, s[p])$, and $s[p] \in \Gamma'$,
7′a. $(q, aw, s, p) \vdash (q', w, s, p-1)$, if $a \in \Sigma_d$, $q' \in \delta_d(q, a, s[p])$, $s[p] \in \Gamma'$, and $s[p] \neq \bot$,
7′b. $(q, aw, s, p) \vdash (q', w, s, p-1)$, if $a \in \Sigma$, $q' \in \delta_d(q, a, s[p])$, $s[p] \in \Gamma$, and $s[p] \neq \bot$,
8′a. $(q, aw, s, p) \vdash (q', w, s, p)$, if $|s| = 1$, $a \in \Sigma_d$, $q' \in \delta_d(q, a, s[p])$, and $s[p] = \bot$.
8′b. $(q, aw, s, p) \vdash (q', w, s, p)$, if $|s| \geq 2$, $a \in \Sigma$, $q' \in \delta_d(q, a, s[p])$, and $s[p] = \bot$.

If for every $q \in Q$, $a \in \Sigma$, and $Z \in \Gamma' \cup \Gamma$ each of the three sums $|\delta_r(q, a, Z)| + |\delta_u(q, a, Z)| + |\delta_d(q, a, Z)|$, $|\delta_c(q, a)| + |\delta_u(q, a, Z)| + |\delta_d(q, a, Z)|$, and $|\delta_i(q, a)| + |\delta_u(q, a, Z)| + |\delta_d(q, a, Z)|$ is at most one, then the weak stack automaton is said to be *deterministic*. Weak deterministic stack automata are denoted by 1wDVSA. Moreover, weak nondeterministic and deterministic non-erasing stack automata are denoted by 1wNENVSA and 1wNEDVSA, respectively.

3 Computational Capacity

We investigate the computational capacity of input-driven stack automata working in strong and weak mode, and prove that these machines induce a strict hierarchy of language families with respect to the following three features: (i) strong and weak mode, (ii) determinism and nondeterminism, and (iii) non-erasing and erasing stack. First we consider deterministic machines and compare weak non-erasing mode with strong erasing mode.

Lemma 3. $L_1 = \{\, a^n b a^{n-1} c \mid n \geq 1 \,\} \in \mathscr{L}(\textit{1wNEDVSA}) \setminus \mathscr{L}(\textit{1sDVSA})$.

Proof. The main idea for a 1wNEDVSA accepting L_1 is to push some symbol on the stack for every input symbol a. When the b appears in the input the stack pointer is moved one position down, that is, to position n. At the same time a certain state is entered which ensures that the stack pointer moves down for subsequent symbols a. In this way, the pointer reaches the bottom of the stack after reading a^{n-1}. If in this situation a c follows, the input is accepted with an up move, otherwise the input is rejected. Clearly, the automaton constructed is deterministic and non-erasing.

To show that $L_1 \notin \mathscr{L}(1\text{sDVSA})$, we assume in contrast to the assertion that L_1 is accepted by a 1sDVSA $M = \langle Q, \Sigma, \Gamma, \bot, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$. Suppose that $a \notin \Sigma_c$. Since the stack is initially empty up to $\bot$, any application

of δ_i, δ_u, δ_d, or δ_r can only alter the current state. Therefore, there are two sufficiently large natural numbers $n_1 \neq n_2$ and a state $q \in Q$, so that

$$(q_0, a^{n_1}ba^{n_1-1}c, \bot, 1) \vdash^* (q, ba^{n_1-1}c, \bot, 1) \vdash^* (f_1, \lambda, s_1, p_1)$$

and

$$(q_0, a^{n_2}ba^{n_2-1}c, \bot, 1) \vdash^* (q, ba^{n_2-1}c, \bot, 1) \vdash^* (f_2, \lambda, s_2, p_2),$$

for $f_1, f_2 \in F$, $s_1, s_2 \in \Gamma'\Gamma^*\bot \cup \{\bot\}$, and $1 \leq p_i \leq |s_i|$, for $i = 1, 2$. This implies

$$(q_0, a^{n_1}ba^{n_2-1}c, \bot, 1) \vdash^* (q, ba^{n_2-1}c, \bot, 1) \vdash^* (f_2, \lambda, s_2, p_2)$$

and, thus, $a^{n_1}ba^{n_2-1}c \in L_1$, which is a contradiction.

Now suppose that $a \in \Sigma_c$. Then for every symbol a, some stack symbol is pushed onto the stack. Since Q is finite, the sequence of states passed through while reading a large number of a's is eventually periodic. Therefore, the sequence of symbols pushed onto the the stack is also eventually periodic. Say, it is of the form uv^*w, where u is a suffix of v, $v \in \Gamma^+$, $w \in \Gamma^*\bot$, and $|w|, |v| \leq |Q|$. Therefore, there are two sufficiently large natural numbers $n_1 \neq n_2$, a state $q \in Q$, strings u, v, w, where u is a suffix of v, $v \in \Gamma^+$, $w \in \Gamma^*\bot$, and natural numbers $k_2 > k_1 > 0$ so that

$$(q_0, a^{n_1}ba^{n_1-1}c, \bot, 1) \vdash^* (q, ba^{n_1-1}c, \varphi(u)v^{k_1}w, n_1 + 1) \vdash^* (f_1, \lambda, s_1, p_1)$$

and

$$(q_0, a^{n_2}ba^{n_2-1}c, \bot, 1) \vdash^* (q, ba^{n_2-1}c, \varphi(u)v^{k_2}w, n_2 + 1) \vdash^* (f_2, \lambda, s_2, p_2)$$

for $f_1, f_2 \in F$, $s_1, s_2 \in \Gamma'\Gamma^*\bot$, and $1 \leq p_i \leq |s_i|$ for $i = 1, 2$. Since the input symbol b forces M to push or to pop a symbol, or to leave the stack as it is, the stack is decreased by at most one symbol when the b is read. The subsequent input symbols a increase the stack again. So, the bottommost n_1 (n_2) stack symbols are not touched again. Since $\varphi(u)v^{k_1}w$ and $\varphi(u)v^{k_2}w$ have a common prefix of length at most 2, we derive

$$(q_0, a^{n_1}ba^{n_2-1}c, \bot, 1) \vdash^* (q, ba^{n_2-1}c, \varphi(u)v^{k_1}w, n_1 + 1) \vdash^* (f_2, \lambda, s_3, p_3)$$

for some $s_3 \in \Gamma'\Gamma^*\bot$ and $p_3 = |s_3|$. This implies that $a^{n_1}ba^{n_2-1}c \in L_1$ which is a contradiction and, hence, $L_1 \notin \mathscr{L}(\text{1sDVSA})$. □

Now we turn to show how input-driven stack automata languages are related to some important context-free language families. Let CFL refer to the family of context-free languages. Then Example 2 shows the following result.

Lemma 4. *$L_2 = \{ a^n b^n c^{n+1} \mid n \geq 1 \} \in \mathscr{L}(\text{1sNEDVSA}) \setminus \text{CFL}$.* □

The next lemma proves a converse relation, namely that a deterministic and linear context-free language is not accepted by any deterministic weak input-driven stack automaton. Let DCFL refer to the family of deterministic context-free and LIN to the family of linear context-free languages, which are both strict sub-families of CFL.

Lemma 5. *Let* $L_3 = \{ a^n b^m a b^m a^n \mid n, m \geq 1 \} \cup \{ b^n a^n \mid n \geq 1 \}$. *Then,* $L_3 \in (DCFL \cap LIN) \setminus \mathscr{L}(1wDVSA)$.

Proof. Clearly, L_3 belongs to DCFL $\cap$ LIN. Now assume that L_3 is accepted by some 1wDVSA $M = \langle Q, \Sigma, \Gamma, \bot, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$. Similarly as in the proof of Lemma 3, we conclude $a \in \Sigma_c$ and $b \in \Sigma_c$, since otherwise the words from L_3 of the form b^+a^+ could not be accepted. Thus, every input from $\{a, b\}^+$ forces M to only push symbols onto the stack. Continuing similar as in the second part of the proof of Lemma 3 shows that then words not belonging to L_3 are accepted. This contradiction shows the lemma. □

For the proof of the following lemma that compares deterministic weak and strong input-driven stack automata (conversely to Lemma 3), we use an incompressibility argument. General information on Kolmogorov complexity and the incompressibility method can be found in [14]. Let $w \in \{0,1\}^+$ be an arbitrary binary string of length n. Then the plain Kolmogorov complexity $C(w|n)$ of w denotes the minimal size of a program that knows n and describes w. It is well known that there exist binary strings w of arbitrary length n such that $C(w|n) \geq n$ (see [14], Theorem 2.2.1). Similarly, for any natural number n, $C(n)$ denotes the minimal size of a program that describes n. It is known that there exist infinitely many natural numbers n such that $C(n) \geq \log(n)$.

Lemma 6. *Let* $\hat{\ } : \{0,1\}^* \to \{\hat{0}, \hat{1}\}^*$ *be the homomorphism that maps* 0 *to* $\hat{0}$ *and* 1 *to* $\hat{1}$. $L_4 = \{ a^{n+m} b^m w \hat{w}^R b^n \mid m, n \geq 1, w \in \{0,1\}^+ \} \cup \{ a^n b^n \mid n \geq 1 \} \in \mathscr{L}(1sDVSA) \setminus \mathscr{L}(1wNEDVSA)$.

Proof. The rough idea of a 1sDVSA M for L_4 is to push a symbol A onto the stack for every input symbol a, and to pop symbol A from the stack for every input symbol b read. In order to check the infix $w\hat{w}^R$, M pushes a Z for every 0 and an O for every 1 while reading w. Subsequently, it pops a Z for every $\hat{0}$ and an O for every $\hat{1}$ while reading $\hat{w}$. If eventually the stack is empty up to $\bot$, the input is to be accepted and otherwise rejected. The concrete construction on M is straightforward except for one detail. The machine must be able to recognize when the stack is empty. To be more precise, it must know when the stack pointer is moved at the bottom-of-stack symbol even if there are no more moves, as for accepting computations. In order to implement this detail, M marks the first symbol on the stack. When this symbol is popped again, M knows that the stack is empty even without reading $\bot$.

Next, we show that $L_4 \notin \mathscr{L}(1\text{wNEDVSA})$. Contrarily, assume that L_4 is accepted by a 1wNEDVSA $M = \langle Q, \Sigma, \Gamma, \bot, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$. Similar as in the proof of Lemma 3, we conclude that $a \in \Sigma_c$. Moreover, since $\{ a^n b^n \mid n \geq 1 \}$ is a subset of L_4, we conclude that $b \in \Sigma_d$. Otherwise, $b \in \Sigma_c$ or $b \in \Sigma_i$ which leads to a contradiction as shown before. Now we consider an accepting computation K on an input of the form $z = a^{n+m} b^m w \hat{w}^R b^n$, for $w \in \{0,1\}^+$, $n \geq 2$, $k = \lfloor \log(n) \rfloor$, and $m = 2^{2^k} - n$. We distinguish two cases.

1. First, we assume that in K nothing is pushed onto the stack while reading the infix w, and consider the configuration $c = (q, \hat{w}^R b^n, s, p)$ with $s \in \Gamma' \Gamma^* \bot$

and $1 \leq p \leq |s|$ after reading $a^{n+m}b^m w$. We claim that the knowledge of M, n, m, $|w|$, q, and p suffices to write a program that outputs w. This is seen as follows.
Since n and m are known, the stack contents s can be computed by simulating M on input a^{n+m}. Furthermore, due to our assumption that nothing is pushed while reading w, and since q and the pointer position p are known, it is possible to simulate M starting with a situation as configuration c but with arbitrary input. This is done successively on all inputs $\hat{v}b^n$ with $|\hat{v}| = |w|$. If $\hat{v} = \hat{w}^R$, then the simulation ends accepting. On the other hand, if the simulation is accepting, then $\hat{v} = \hat{w}^R$, since otherwise $a^{n+m}b^m w\hat{v}b^n$ with $\hat{v} \neq \hat{w}^R$ would belong to L_4 which is a contradiction. This suffices to identify and output w. The size of the program is a constant for the code itself plus the sizes of M, n, m, $|w|$, q, and p. The size of M and, thus, of q is also a constant, while $p \leq n + m$. So, the size of the program is of order $O(\log(n) + \log(m) + \log(n + m) + \log(|w|))$. Fixing n and m and choosing $|w|$ large enough, we obtain $C(w||w|) \in o(|w|)$ which is a contradiction to the above-cited result that there exist binary strings w of arbitrary length such that $C(w||w|) \geq |w|$.

2. Second, we assume that in K something is pushed onto the stack while reading the infix w, and consider the configuration $c = (q, x, s, |s|)$ with $s \in \Gamma'\Gamma^*\bot$ and $x \in \{0,1\}^*\{\hat{0},\hat{1}\}^+b^+$ is the remaining input when the first symbol has been pushed onto the stack while reading w. We claim that the knowledge of M, k, $|w|$, the length ℓ of the prefix w' of w which has been already read, q, and the last symbol B pushed suffices to write a program that outputs n.
The stack height $|s|$ is $m + n + 1 = 2^{2^k} + 1$. Therefore, the stack contents s can be computed by simulating M on input $a^{|s|-1}$ using solely the knowledge of k, and adding B on the top. Next the simulation of M beginning in state q while having the stack pointer on the top of the stack contents s is started successively on all inputs $u\hat{v}b^i$ with $u \in \{0,1\}^*$, $|u| = |w| - \ell$, $|\hat{v}| = |w|$, and $1 \leq i \leq 2^{2^k}$. If $w'u\hat{v} = w\hat{w}^R$ and $i = n$, then the simulation ends accepting. On the other hand, if the simulation is accepting, then $w'u\hat{v} = w\hat{w}^R$ and $i = n$. This suffices to identify and output n. Again, the size of the program is a constant for the code itself plus the sizes of M, k, $|w|$, ℓ, q, and $|\Gamma|$. The size of M and, thus, of q and $|\Gamma|$ is also a constant, while $\ell \leq |w|$. So, the size of the program is of order $O(\log(k) + \log(|w|)) = O(\log(\log(n)) + \log(|w|))$. Fixing w and choosing n large enough, we obtain $C(n) \in o(\log(n))$ which is a contradiction since there are infinitely many natural numbers n such that $C(n) \geq \log(n)$.

This proves the lemma. □

With the help of the previous four lemmata we can show the following strict inclusion relations and incomparability results.

Theorem 7. *All inclusions shown in Figure 1 are strict. Moreover, language families that are not linked by a path are pairwise incomparable.*

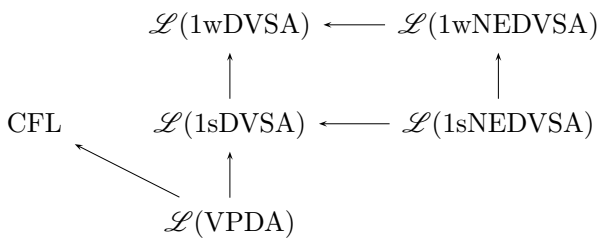

Fig. 1. Inclusion structure of deterministic language families. The arrows indicate strict inclusions. All families not linked by a path are pairwise incomparable.

Proof. The inclusions of the language families induced by the input-driven stack automata are clear by definition. Moreover, $\mathscr{L}(\text{VPDA}) \subseteq \text{CFL}$ is immediate and $\mathscr{L}(\text{VPDA}) \subseteq \mathscr{L}(\text{1sDVSA})$ follows since every visibly (deterministic) pushdown automaton is also a deterministic strong input-driven stack automaton (that is not allowed to *read* the internal contents of the stack). First, we show that these inclusions are strict. The strictness of the inclusion $\mathscr{L}(\text{1sDVSA}) \subseteq \mathscr{L}(\text{1wDVSA})$ as well as the inclusion $\mathscr{L}(\text{1sNEDVSA}) \subseteq \mathscr{L}(\text{1wNEDVSA})$ is ensured by language L_1 of Lemma 3. Furthermore, a language similar to L_1 was used in [1] to show that the inclusion $\mathscr{L}(\text{VPDA}) \subseteq \text{CFL}$ is proper. Finally, the strictness of both inclusions $\mathscr{L}(\text{1sNEDVSA}) \subseteq \mathscr{L}(\text{1sDVSA})$ and $\mathscr{L}(\text{1wNEDVSA}) \subseteq \mathscr{L}(\text{1wDVSA})$ follows by language L_4 of Lemma 6. Next, we show the incomparability results. The languages L_2 and L_3 from Lemmata 4 and 5 imply the incomparability of CFL with the all language families of deterministic variants of input-driven stack automata, namely $\mathscr{L}(\text{1sNEDVSA})$, $\mathscr{L}(\text{1wNEDVSA})$, $\mathscr{L}(\text{1sDVSA})$, and $\mathscr{L}(\text{1wDVSA})$. The incomparability of $\mathscr{L}(\text{VPDA})$ with both language families $\mathscr{L}(\text{1sNEDVSA})$ and $\mathscr{L}(\text{1wNEDVSA})$ follows by the languages L_2 from Lemma 4 and L_4 from Lemma 6, and the obvious fact that L_4 is a visibly pushdown language. Finally, $\mathscr{L}(\text{1sDVSA})$ and $\mathscr{L}(\text{1wNEDVSA})$ are incomparable due to the languages L_1 and L_4 from Lemmata 3 and 6. This proves the stated claim. □

In the remainder of this section we investigate the relation between deterministic and nondeterministic input-driven stack automata. To this end consider the language $L_5 = T_1 \cup T_2 \cup T_1' \cup T_2' \cup T_3 \cup T_4$ with

$$
\begin{aligned}
T_1 &= \{\, a^n d_1^n u_1^{n+1} \mid n \geq 1 \,\} \\
T_2 &= \{\, a^n d_2^n u_2^{n+1} \mid n \geq 1 \,\} \\
T_1' &= \{\, a^n u_1^m d_1^n u_1^{n+1} \mid m, n \geq 1 \,\} \\
T_2' &= \{\, a^n u_2^m d_2^n u_2^{n+1} \mid m, n \geq 1 \,\} \\
T_3 &= \{\, a^n d_1^{n+1} \mid n \geq 1 \,\}
\end{aligned}
$$

and

$$
T_4 = \{\, a^{n+m} d_1^m d_2^n u_2^{n+1} u_1^m w \hat{w}^R d_1^m d_2^{n+1} \mid m, n \geq 1, w \in \{0,1\}^+ \,\},
$$

where $\hat{}$ is the mapping introduced in Lemma 6. Then we can prove the following result, which allows us to categorize the different symbols used in the definition of language L_5.

Lemma 8. *Let $M = \langle Q, \Sigma, \Gamma, \bot, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$ be a 1wDVSA.*

1. *If T_1 is accepted by M, then $d_1 \notin \Sigma_c \cup \Sigma_r$.*
2. *If T_2 is accepted by M, then $d_2 \notin \Sigma_c \cup \Sigma_r$.*
3. *If T_1' is accepted by M, then $u_1 \notin \Sigma_c \cup \Sigma_r$.*
4. *If T_2' is accepted by M, then $u_2 \notin \Sigma_c \cup \Sigma_r$.*

Proof. We only prove the first claim. The other claims can be shown by similar arguments. So assume that T_1 is accepted by M. In order to show $d_1 \notin \Sigma_c \cup \Sigma_r$, we assume in contrast to the assertion that d_1 is in Σ_c or Σ_r. Thus, we distinguish two cases—note that we make no assumption on the containment of the letter a within the sub-alphabets that come from the partition of Σ:

1. Assume that $d_1 \in \Sigma_r$. Note that after reading the word $a^n d_1^n$ the stack of M is empty up to the bottom of stack symbol $\bot$. This is due to the fact, that M is deterministic and $d_1 \in \Sigma_r$. Since Q is finite, there are two distinct sufficiently large numbers $n_1 \neq n_2$ and a state $q \in Q$, such that

   $$(q_0, a^{n_1} d_1^{n_1} u_1^{n_1+1}, \bot, 1) \vdash^* (q, u_1^{n_1+1}, \bot, 1) \vdash^* (f_1, \lambda, s_1, p_1)$$

   and

   $$(q_0, a^{n_2} d_1^{n_2} u_1^{n_2+1}, \bot, 1) \vdash^* (q, u_1^{n_2+1}, \bot, 1) \vdash^* (f_2, \lambda, s_2, p_2),$$

   for $f_1, f_2 \in F$, $s_1, s_2 \in \Gamma'\Gamma^*\bot \cup \{\bot\}$, and $1 \leq p_i \leq |s_i|$, for $i = 1, 2$. This implies

   $$(q_0, a^{n_1} d_1^{n_1} u_1^{n_2+1}, \bot, 1) \vdash^* (q, u_1^{n_2+1}, \bot, 1) \vdash^* (f_2, \lambda, s_2, p_2)$$

   and, thus, $a^{n_1} d_1^{n_1} u_1^{n_2+1} \in T_1$, which is a contradiction.
2. Now suppose that $d_1 \in \Sigma_c$. Then we argue as follows. Since M is deterministic and $d_1 \in \Sigma_c$, we know that the stack height is at least $n+1$ after reading word $a^n d_1^n$ and the stack pointer is on the topmost symbol. Then further reading of u_1^{n+1}—here we make no assumption on letter u_1 and its containment in Σ_c Σ_r, Σ_i, Σ_u, or Σ_d—may only touch the topmost $n+1$ symbols of the stack. Since Q and Γ are finite we find two sufficiently large numbers $n_1 \neq n_2$, a state $q \in Q$ and a stack symbol $Z \in \Gamma$, such that

   $$(q_0, a^{n_1} d_1^{n_1} u_1^{n_1+1}, \bot, 1) \vdash^* (q, d_1^{n_1} u_1^{n_1+1}, Z'\gamma_1, |\gamma_1| + 1) \vdash^* (f_1, \lambda, s_1, p_1)$$

   and

   $$(q_0, a^{n_2} d_1^{n_2} u_1^{n_2+1}, \bot, 1) \vdash^* (q, d_1^{n_2} u_1^{n_2+1}, Z'\gamma_2, |\gamma_2| + 1) \vdash^* (f_2, \lambda, s_2, p_2),$$

   for $\gamma_i = \lambda$, if $Z = \bot$, and $\gamma_i \in \Gamma^*\bot$, if $Z \neq \bot$, for $i = 1, 2$, and $f_1, f_2 \in F$, $s_1, s_2 \in \Gamma'\Gamma^*\bot \cup \{\bot\}$, and $1 \leq p_i \leq |s_i|$, for $i = 1, 2$. Since both topmost stack symbols after processing a^{n_1} and a^{n_2}, respectively, are identical, and

the stack contents below that particular symbol is never touched while processing the remaining part $d_1^{n_1}u_1^{n_1+1}$ and $d_1^{n_2}u_1^{n_2+1}$, respectively, we obtain the accepting computation

$$(q_0, a^{n_1}d_1^{n_2}u_1^{n_2+1}, \bot, 1) \vdash^* (q, d_1^{n_2}u_1^{n_2+1}, Z'\gamma_1, |\gamma_1|+1) \vdash^* (f_2, \lambda, s_3, p_3),$$

for some $s_3 \in \Gamma'\Gamma^*\bot \cup \{\bot\}$ with $|s_3| \geq |\gamma_1|$, and $p_3 \geq |\gamma_1|$. This implies that the word $a^{n_1}d_1^{n_2}u_1^{n_2+1} \in T_1$, which is a contradiction. □

Next we need some notation. Let Σ_c, Σ_r, Σ_i, Σ_u, and Σ_d be a partitioning of Σ. Then we say that a partitioning Σ'_c, Σ'_r, Σ'_i, Σ'_u, and Σ'_d of $\Sigma' \subseteq \Sigma$ is *compatible with* the partitioning of Σ, if $\Sigma'_c \subseteq \Sigma_c$, $\Sigma'_r \subseteq \Sigma_r$, $\Sigma'_i \subseteq \Sigma_i$, $\Sigma'_u \subseteq \Sigma_u$, and $\Sigma'_d \subseteq \Sigma_d$. Then the next lemma, which can be shown by an easy adaptation of the well-known Cartesian product construction for pushdown automata, reads as follows:

Lemma 9. *Let $M = \langle Q, \Sigma, \Gamma, \bot, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d\rangle$ be a 1wDVSA (1sDVSA) and $R \subseteq \Sigma^*$ be a regular language. Then the language $L(M) \cap R$ is accepted by a 1wDVSA (1sDVSA) with a compatible partitioning of the alphabet Σ.* □

Now we are ready to show that there exists a language that belongs to the class induced by the most restricted form of nondeterministic input-driven automata, namely 1sNENVSA, but is not a member of the larger deterministic class 1wDVSA.

Lemma 10. $L_5 \in \mathscr{L}(1sNENVSA) \setminus \mathscr{L}(1wDVSA)$.

Proof. The idea for a 1sNENVSA accepting L_5 is first to guess whether the input belongs to T_1, T_2, T'_1, T'_2, T_3, or T_4. The construction to accept inputs from T_1, T_2, T'_1, T'_2, and T_3 is similar to the construction in Lemma 4. In the constructions for T'_1 and T'_2 we observe that u_1 and u_2 belong to Σ_u. Since the stack pointer is on the top after reading a^n, the processing of u_1 and u_2 only affects the current state, but not the stack or the position of the stack pointer. The construction to accept T_4 is as follows. While reading a's some symbol A is pushed onto the stack up to some moment in which it is nondeterministically decided to push some different symbol B onto the stack for every remaining input symbol a. Then, while reading d_1's and seeing B's the stack pointer moves down and continues moving down while reading d_2's and seeing A's. If the bottom of the stack is reached, the stack pointer moves up while reading u_2's and seeing A's and continues moving up, while reading u_1's and seeing B's on the stack. If the top of the stack is reached, the processing of the infix $w\hat{w}^R$ is done in a similar way as in the proof of Lemma 6. The only difference is that symbols from $\{\hat{0}, \hat{1}\}$ now ensure that the stack pointer moves down instead of popping off the topmost symbol. If the first B of the stack is again reached, the stack pointer continues moving down while reading d_1's and seeing B's and continues to move down while reading d_2's and seeing A's. The input is accepted if the bottom of the stack is eventually reached and rejected otherwise. We observe from the

constructions that $a, 0, 1 \in \Sigma_c$, $d_1, d_2, \hat{0}, \hat{1} \in \Sigma_d$, and $u_1, u_2 \in \Sigma_u$. Thus, the automaton constructed is non-erasing and works in the strong mode.

Next, we show by way of contradiction that $L_5 \notin \mathscr{L}(\text{1wDVSA})$. Assume that L_5 is accepted by a 1wDVSA $M = \langle Q, \Sigma, \Gamma, \bot, q_0, F, \delta_c, \delta_r, \delta_i, \delta_u, \delta_d \rangle$. Similar as in the proof of Lemma 3, we conclude that $a \in \Sigma_c$. Now, we assume that $d_1 \in \Sigma_c \cup \Sigma_r$. Due to Lemma 9 with $R = a^+ d_1^+ u_1^+$ we obtain that T_1 is accepted by some 1wDVSA having $d_1 \in \Sigma_c \cup \Sigma_r$. This is a contradiction to Lemma 8. Thus, $d_1 \notin \Sigma_c \cup \Sigma_r$. Similarly, it can be shown that also $d_2, u_1, u_2 \notin \Sigma_c \cup \Sigma_r$. Finally, we claim that $d_1 \in \Sigma_d$. Otherwise, due to Lemma 9 with $R = a^+ d_1^+$, language T_3 would be accepted by some 1wDVSA having $d_1 \in \Sigma_i \cup \Sigma_u$ which is a contradiction.

The rest of the proof is similar to the proof of Lemma 6 and we leave out some details here. We consider an accepting computation K on an input of the form $z = a^{n+m} d_1^m d_2^n u_2^{n+1} u_1^m w \hat{w}^R d_1^m d_2^n$, for $w \in \{0,1\}^+$, $n \geq 2$, $k = \lfloor \log(n)) \rfloor$, and $m = 2^{2^k} - n$. Due to the above considerations, we know that nothing is pushed onto or popped off the stack while reading the infix $d_1^m d_2^n u_2^{n+1} u_1^m$. We distinguish now two cases.

1. First, we assume that nothing is pushed onto or popped off the stack while reading the infix w, and consider the configuration $c = (q, \hat{w}^R d_1^m d_2^n, s, p)$ with $s \in \Gamma' \Gamma^* \bot$ and $1 \leq p \leq |s|$ after reading $a^{n+m} d_1^m d_2^n u_2^{n+1} u_1^m w$. Then, the knowledge of $M, n, m, |w|, q$, and p is sufficient to write a program that outputs w. The size of this program is bounded by $O(\log(n) + \log(m) + \log(n+m) + \log(|w|))$. Hence, $C(w||w|) \in o(|w|)$ which is a contradiction for $|w|$ large enough.
2. Second, we assume that something is pushed onto or popped off the stack while reading the infix w. We consider the configuration $c = (q, x, s, |s|)$ with $s \in \Gamma' \Gamma^* \bot$ and $x \in \{0,1\}^* \{\hat{0}, \hat{1}\}^+ d_1^+ d_2^+$ being the remaining input when the first symbol has been pushed onto or popped off the stack while reading w. Again, the knowledge of M, k, $|w|$, the length ℓ of the prefix w' of w which has been already read, q, and the last pushed or popped symbol $B \in \Gamma'$ is sufficient to write a program that outputs n. The size of this program is bounded by $O(\log(k) + \log(|w|)) = O(\log(\log(n)) + \log(|w|))$. Thus, $C(n) \in o(\log(n))$ which is a contradiction since there are infinitely natural numbers n such that $C(n) \geq \log(n)$.

Hence, there cannot be any 1wDVSA that accepts the language L_5. □

As an immediate corollary of the previous lemma we obtain the following strict inclusions.

Corollary 11. *1.* $\mathscr{L}(\textit{1sNEDVSA}) \subset \mathscr{L}(\textit{1sNENVSA})$.
2. $\mathscr{L}(\textit{1sDVSA}) \subset \mathscr{L}(\textit{1sNVSA})$.
3. $\mathscr{L}(\textit{1wNEDVSA}) \subset \mathscr{L}(\textit{1wNENVSA})$.
4. $\mathscr{L}(\textit{1wDVSA}) \subset \mathscr{L}(\textit{1wNVSA})$. □

These strict inclusions show a large difference between input-driven pushdown automata languages or equivalently visibly pushdown languages, where determinism coincides with nondeterminism on the underlying automaton model.

For input-driven stack automata, even for the most restrictive version, the strong machines, nondeterminism is more powerful than determinism. These strictness results are also reflected in the forthcoming result on the complexity of the fixed membership problem. Since the family of languages accepted by ordinary deterministic stack automata belongs to deterministic polynomial time P [13] it follows that this is also true for every language family induced by a deterministic input-driven stack automaton, regardless whether we have a strong or weak machine, or whether the device is non-erasing or not. On the other hand, when changing from ordinary deterministic stack automata to nondeterministic ones, we obtain an NP-complete language family [11, 19]. This is also the case for the nondeterministic versions of input-driven stack automata, even for strong non-erasing machines, which is shown next.

Theorem 12. *Each of the language families $\mathscr{L}$(1sNENVSA), $\mathscr{L}$(1sNVSA), $\mathscr{L}$(1wNENVSA), and $\mathscr{L}$(1wNVSA) has an NP-complete fixed membership problem.*

Proof. The containment in NP follows immediately from the fact that ordinary nondeterministic stack automata have an NP-complete fixed membership problem [11, 19]. It remains to show NP-hardness. To this end, it suffices to show that the (with respect to set inclusion) smallest language family $\mathscr{L}$(1sNENVSA) has an NP-hard membership problem. We reduce the well-known NP-complete 3SAT problem [7] to the problem under consideration. We encode a Boolean formula $F = c_1 \wedge c_2 \wedge \cdots \wedge c_m$ with variables $X = \{x_1, x_2, \ldots, x_n\}$, where each clause c_i with $1 \leq i \leq m$ is a disjunction of three literals, by a word $\langle F \rangle \in 1^n(\{0, +, -\}^*\#^*\$)^*$. The prefix encodes the number of input variables. Then each clause c_i, for $1 \leq i \leq m$, of the formula F is encoded by a word w_{c_i} in $\{0, +, -\}^n\#^n$, where at position j, for $1 \leq j \leq n$, there is a 0 ($+$, $-$), if the variable x_j does not appear (positively appears, negatively appears) in c_i. These words are separated by \$-symbols and are placed in sequence. Then the language

$$L = \{ \langle F \rangle \mid F \text{ is a Boolean formula that evaluates to } 1 \}$$

is NP-hard. We informally describe how a 1wNENVSA automaton M accepts L. Set $\Sigma_c = \{1\}$, $\Sigma_r = \emptyset$, $\Sigma_i = \{\$\}$, $\Sigma_u = \{\#\}$, and $\Sigma_d = \{0, +, -\}$. On prefix 1^n the automaton pushes either the symbol 0 or 1. In this way, the automaton guesses an assignment to the n variables of the formula. Then the sequence that encodes a clause is used to read into the stack in order to determine the assignments of the involved variables. In passing the automaton checks whether this clause evaluates to 1. Then the block of # symbols is used to reset the stack pointer to the top of the stack, and after reading \$ the checking procedure for the next clause is restarted. If all clauses evaluate to 1, the whole encoding is accepted, otherwise it is rejected. It is easy to see that the automaton is non-erasing. This shows that already the language family $\mathscr{L}$(1sNENVSA) contains an NP-hard language. □

References

1. Alur, R., Madhusudan, P.: Adding nesting structure to words. J. ACM 56, Article 16 (2009)
2. von Braunmühl, B., Verbeek, R.: Input-driven languages are recognized in $\log n$ space. In: Fundamentals of Computation Theory (FCT 1983). LNCS, vol. 158, pp. 40–51. Springer (1983)
3. Chervet, P., Walukiewicz, I.: Minimizing variants of visibly pushdown automata. In: Mathematical Foundations of Computer Science (MFCS 2007). LNCS, vol. 4708, pp. 135–146. Springer (2007)
4. Chomsky, N.: Handbook of Mathematic Psychology, vol. 2, chap. Formal Properties of Grammars, pp. 323–418. Wiley & Sons (1962)
5. Crespi-Reghizzi, S., Mandrioli, D.: Operator precedence and the visibly pushdown property. In: Language and Automata Theory and Applications (LATA 2010). LNCS, vol. 6031, pp. 214–226. Springer (2010)
6. Dymond, P.W.: Input-driven languages are in $\log n$ depth. Inform. Process. Lett. 26, 247–250 (1988)
7. Garey, M.R., Johnson, D.S.: Computers and Intractability, A Guide to the Theory of NP-Completeness. Freeman (1979)
8. Ginsburg, S.: Algebraic and Automata-Theoretic Properties of Formal Languages. North-Holland (1975)
9. Ginsburg, S., Greibach, S.A., Harrison, M.A.: One-way stack automata. J. ACM 14, 389–418 (1967)
10. Han, Y.S., Salomaa, K.: Nondeterministic state complexity of nested word automata. Theoret. Comput. Sci. 410, 2961–2971 (2009)
11. Hunt, H.: On the complexity of finite, pushdown and stack automata. Math. Systems Theory 10, 33–52 (1976)
12. Kutrib, M., Malcher, A.: Reversible pushdown automata. In: Language and Automata Theory and Applications (LATA 2010). LNCS, vol. 6031, pp. 368–379. Springer (2010)
13. Lange, K.J.: A note on the P-completeness of deterministic one-way stack languages. J. Univ. Comput. Sci. 16, 795–799 (2010)
14. Li, M., Vitányi, P.: An Introduction to Kolmogorov Complexity and its Applications. Springer (1993)
15. Mehlhorn, K.: Pebbling mountain ranges and its application of DCFL-recongnition. In: International Colloquium on Automata, Languages and Programming (ICALP 1980). LNCS, vol. 85, pp. 422–435. Springer (1980)
16. Okhotin, A., Salomaa, K.: State complexity of operations on input-driven pushdown automata. In: Mathematical Foundations of Computer Science (MFCS 2011). LNCS, vol. 6907, pp. 485–496. Springer (2011)
17. Piao, X., Salomaa, K.: Operational state complexity of nested word automata. Theoret. Comput. Sci. 410, 3290–3302 (2009)
18. Salomaa, A.: Formal Languages. ACM Monograph Series, Academic Press (1973)
19. Shamir, E., Beeri, C.: Checking stacks and context-free programmed grammars accept P-complete languages. In: International Colloquium of Automata, Languages, and Programming (ICALP 1974). LNCS, vol. 14, pp. 27–33. Springer (1974)
20. Sudborough, I.H.: On the tape complexity of deterministic context-free languages. J. ACM 25, 405–414 (1978)