Further Closure Properties of Input-Driven Pushdown Automata

Alexander Okhotin, Kai Salomaa

Further closure properties of input-driven pushdown automata

Alexander Okhotin[1,⋆] and Kai Salomaa[2]

[1] St. Petersburg State University, 7/9 Universitetskaya nab., Saint Petersburg 199034, Russia alexander.okhotin@spbu.ru
[2] School of Computing, Queen's University, Kingston, Ontario K7L 2N8, Canada, ksalomaa@cs.queensu.ca

Abstract. The paper investigates the closure of the language family defined by input-driven pushdown automata (IDPDA) under the following operations: insertion $ins(L,K) = \{xyz \mid xz \in L,\ y \in K\}$, deletion $del(L,K) = \{xz \mid xyz \in L,\ y \in K\}$, square root $\sqrt{L} = \{w \mid ww \in L\}$, and the first half $\frac{1}{2}L = \{u \mid \exists v : |u| = |v|,\ uv \in L\}$. For K and L recognized by nondeterministic IDPDA, with m and with n states, respectively, insertion requires $mn+2m$ states, as long as K is well-nested; deletion is representable with $2n$ states, for well-nested K; square root requires $n^3-O(n^2)$ states, for well-nested L; the well-nested subset of the first half is representable with $2^{O(n^2)}$ states. Without the well-nestedness constraints, non-closure is established in each case.

1 Introduction

Input-driven pushdown automata (IDPDA), also known under the name of *visibly pushdown automata*, are an important special class of pushdown automata, introduced by Mehlhorn[16] and later studied, in particular, by Alur and Madhusudan [1,2]. In these automata, the input symbol determines whether the automaton should push a stack symbol, pop a stack symbol or leave the stack untouched. These symbols are called *left brackets*, *right brackets* and *neutral symbols*, and the symbol pushed at each left bracket is always popped when reading the corresponding right bracket. Input-driven automata are important as a model of hierarchically structured data, such as XML documents or computation traces for recursive procedure calls.

Input-driven automata exist in deterministic (DIDPDA) and nondeterministic (NIDPDA) variants, which, as shown by von Braunmühl and Verbeek [3], are equivalent in power. Input-driven automata are also notable for their appealing closure properties, which almost rival those of finite automata. For instance, they are closed under all Boolean operations, concatenation, Kleene star, reversal [1,21], quotient [23]. and edit distance neighbourhood [22] (for binary operations it is assumed that the automata defining its two arguments use the same

⋆ Supported by Russian Science Foundation, project 18-11-00100.

partition of the alphabet into classes), Besides the closure results, the descriptional complexity of these operations has also been estimated in the literature. For instance, the concatenation of an m-state and an n-state DIDPDA in the worst case requires a DIDPDA with $m2^{\Theta(n \log n)}$ states [21], whereas both for the union and for the intersection its state complexity is $\Theta(mn)$ [24]. For NIDPDA, the state complexity of concatenation and of union is $m+n+O(1)$ [1], and for the intersection it is $\Theta(mn)$ [10]. Further state complexity results were established for an intermediate family of *unambiguous input-driven automata* (UIDPDA) [20]. For more details on input-driven automata and their complexity, the readers are directed to a fairly recent survey [19].

The purpose of this paper is to investigate the closure and the state complexity of input-driven pushdown automata with respect to several further operations on languages, which are fairly standard in the theoretical research in formal language theory—yet their application to IDPDA has not yet been considered.

The first operation, investigated in Section 3, is **insertion**, $\mathrm{ins}(L, K) = \{\, xyz \mid xz \in L,\ y \in K \,\}$. For finite automata, the closure under this operation is folklore, and its precise state complexity has recently been determined by Han et al. [8]. For input-driven automata, the closure is currently known only for singleton K [22]. This paper demonstrates the closure under the assumption that K consists only of well-nested strings, and determines the worst-case number of states in an NIDPDA recognizing $\mathrm{ins}(L, K)$ as $mn + 2m$, where m is the number of states in the NIDPDA for K, while the NIDPDA for L has n states. For unrestricted K, the non-closure is established.

Another related operation is **deletion**, $\mathrm{del}(L, K) = \{\, xz \mid \exists y \in K : xyz \in L \,\}$. It is well-known that if L is regular, then the result is regular for an arbitrary K. The state complexity of this operation for finite automata has recently been studied by Han et al. [9]. For NIDPDA, as shown in Section 4, $\mathrm{del}(L, K)$ is representable with $2n$ states, as long as K consists only of well-nested strings; without this assumption, there is a non-closure result.

The third operation, studied in Section 5, is the **square root**, $\sqrt{L} = \{\, w \mid ww \in L \,\}$. Assuming that L consists of only well-nested strings and is recognized by an n-state DIDPDA, its square root can be represented by a DIDPDA with n^n states, and this bound is tight; this proof uses the *behaviour functions* of DIDPDA [18,21] in the same way as in the similar result for deterministic finite automata (DFA) [15]. For NIDPDA, as well as for nondeterministic finite automata (NFA), it is shown that this operation requires at most n^3 states and at least $(n-1)(n-2)(n-3)$ states in the worst case. If L is not restricted to well-nested strings, a non-closure is established.

Section 6 considers the operation of taking the **first half** of all even-length strings in a language, $\frac{1}{2}L = \{u \mid \exists v : |u| = |v|,\ uv \in L\}$. This is a particular case of *proportional removals*, studied by Maslov [15] and by Domaratzki [4] for DFA, and by Goč et al. [6] for NFA. For NIDPDA, if L consists of well-nested strings, then $\frac{1}{2}L$ is recognized by an NIDPDA with $2^{O(n^2)}$ states. For not necessarily well-nested L, there is again a non-closure.

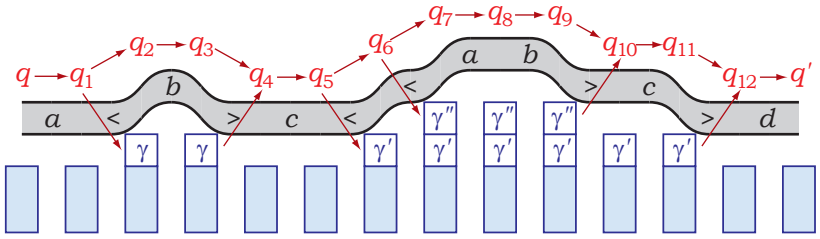

Fig. 1. The computation of an IDPDA on a well-nested string.

The last Section 7 proposes an open problem on the state complexity of **scattered substrings** for IDPDA.

2 Input-driven automata

A *deterministic input-driven pushdown automaton* (DIDPDA) [1,16] is a special case of a deterministic pushdown automaton, in which the input alphabet Σ is split into three disjoint sets of *left brackets* Σ_{+1}, *right brackets* Σ_{-1} and *neutral symbols* Σ_0. If the input symbol is a left bracket from Σ_{+1}, then the automaton always pushes one symbol onto the stack. For a right bracket from Σ_{-1}, the automaton must pop one symbol. Finally, for a neutral symbol in Σ_0, the automaton may not use the stack. In this paper, symbols from Σ_{+1} and Σ_{-1} shall be denoted by left and right angled brackets, respectively ($<$, $>$), whereas lower-case Latin letters from the beginning of the alphabet ($a, b, c, \ldots$) shall be used for symbols from Σ_0.

A *nondeterministic input-driven pushdown automaton* (NIDPDA) [1,3] is a similarly restricted case of a nondeterministic pushdown automaton, in which the type of the action on the stack is determined by the input symbol, whereas the actual next state and the symbol pushed onto the stack may be selected nondeterministically.

Input-driven automata are often restricted to operate on input strings, in which the brackets are *well-nested*. When an input-driven automaton reads a left bracket $< \in \Sigma_{+1}$, it pushes a symbol onto the stack. This symbol is popped at the exact moment when the automaton encounters the matching right bracket $> \in \Sigma_{-1}$. Thus, a computation of an input-driven automaton on any well-nested substring leaves the stack contents untouched, as illustrated in Figure 1.

Alur and Madhusudan [1] also considered input-driven automata operating on potentially ill-nested input strings. For every unmatched left bracket, the symbol pushed to the stack when reading this bracket is never popped, and remains in the stack to the end of the computation. An unmatched right bracket is read with an empty stack: instead of popping a stack symbol, the automaton merely detects that the stack is empty and makes a special transition, which leaves the stack empty.

Definition 1 (von Braunmühl and Verbeek [3]; Alur and Madhusudan [1]). *A nondeterministic input-driven pushdown automaton (NIDPDA) over an alphabet $\widetilde{\Sigma} = (\Sigma_{+1}, \Sigma_{-1}, \Sigma_0)$ consists of*

- *a finite set Q of states, with set of initial states $Q_0 \subseteq Q$ and accepting states $F \subseteq Q$;*
- *a finite stack alphabet Γ, and a special symbol $\bot \notin \Gamma$ for the empty stack;*
- *for a neutral symbol $c \in \Sigma_0$, a transition function $\delta_c \colon Q \to 2^Q$ gives the set of possible next states;*
- *for each left bracket symbol $< \in \Sigma_{+1}$, the behaviour of the automaton is described by a function $\delta_< \colon Q \to 2^{Q \times \Gamma}$, which, for a given current state, provides a set of pairs (q, γ), with $q \in Q$ and $\gamma \in \Gamma$, where each pair means that the automaton enters the state q and pushes γ onto the stack;*
- *for every right bracket symbol $> \in \Sigma_{-1}$, there is a function $\delta_> \colon Q \times (\Gamma \cup \{\bot\}) \to 2^Q$ specifying possible next states, assuming that the given stack symbol is popped from the stack (or that the stack is empty).*

A configuration is a triple (q, w, x), with the current state $q \in Q$, remaining input $w \in \Sigma^*$ and stack contents $x \in \Gamma^*$. Possible next configurations are defined as follows.

$$\begin{aligned}
(q, cw, x) &\vdash_A (q', w, x), && c \in \Sigma_0,\ q \in Q,\ q' \in \delta_c(q)\\
(q, <w, x) &\vdash_A (q', w, \gamma x), && < \in \Sigma_{+1},\ q \in Q,\ (q', \gamma) \in \delta_<(q)\\
(q, >w, sx) &\vdash_A (q', w, x), && > \in \Sigma_{-1},\ q \in Q,\ s \in \Gamma,\ q' \in \delta_>(q, \gamma)\\
(q, >w, \varepsilon) &\vdash_A (q', w, \varepsilon), && > \in \Sigma_{-1},\ q' \in \delta_>(q, \bot)
\end{aligned}$$

The language recognized by A is the set of all strings $w \in \Sigma^*$, on which the automaton, having begun its computation in the configuration (q_0, w, ε), eventually reaches a configuration of the form (q, ε, x), with $q \in F$ and with any stack contents $x \in \Gamma^*$ (for well-nested inputs, the stack is empty at this point).

An NIDPDA is deterministic (DIDPDA), if there is a unique initial state and every transition provides exactly one action.

Von Braunmühl and Verbeek [3], proved that every n-state NIDPDA operating on well-nested strings can be transformed to a 2^{n^2}-state DIDPDA. Alur and Madhusudan [1] extended this construction to allow ill-nested inputs, so that a DIDPDA has 2^{2n^2} states; in the worst case, $2^{\Omega(n^2)}$ states are necessary.

The known closure results under Boolean operations, concatenation, star, reversal, quotient and edit distance neighbourhood are all valid for input-driven automata operating on ill-nested strings.

3 Insertion

The insertion operation is a binary operation on languages: $\mathrm{ins}(L, K)$ is the set of all strings obtained by taking any string from L and any position in this string, and inserting any string from K at that position.

$$\mathrm{ins}(L, K) = \{\, xyz \mid xz \in L,\ y \in K \,\}$$

The question is: when both K and L are recognized by IDPDA, shall $\mathrm{ins}(L, K)$ always be recognized by an IDPDA? Provided that K does not contain any ill-nested strings, the answer is positive, established by the following effective construction.

Lemma 1. *Let L be a language recognized by an NIDPDA $\mathcal{B}$ with the set of states Q and with a stack alphabet Γ. Let K be a set of well-nested strings, which is recognized by an NIDPDA $\mathcal{A}$ with the set of states P and with a stack alphabet Θ. Then, the language $\mathrm{ins}(L, K)$ is recognized by an NIDPDA with the set of states $Q \cup \widetilde{Q} \cup (P \times Q)$, where $\widetilde{Q} = \{ \widetilde{q} \mid q \in Q \}$, and with the stack alphabet $\Gamma \cup \Theta$.*

Proof. The new automaton begins by simulating the computation of $\mathcal{B}$ in the states $\widetilde{Q}$.

At any moment, while in a state $\widetilde{q} \in \widetilde{Q}$, it may nondeterministically guess that a substring accepted by $\mathcal{A}$ begins at the next symbol. If the next symbol is $c \in \Sigma_0$, the new automaton may make a transition from $\widetilde{q}$ by c to a pair (p, q), where p is any state, to which $\mathcal{A}$ may go from its initial state p_0 by reading c. If the next symbol is a left bracket $< \in \Sigma_{+1}$, the new automaton may choose to go from $\widetilde{q}$ by the left bracket $<$ to a pair (p, q), pushing θ, where $\mathcal{A}$ may go from p_0 to p by this bracket $<$, pushing θ.

Thus, the simulation of $\mathcal{A}$ on a substring begins. It proceeds in the states from $P \times Q$, with the simulation carried out in the first component, whereas the state of $\mathcal{B}$ remembered in the second component remains unchanged. Whenever the simulating automaton is in a state (p, q) with p accepting, it may decide that the well-nested substring being inserted has now ended, and accordingly make simulate $\mathcal{B}$'s transition from the state q on the next input symbol. After that, the simulation continues as $\mathcal{B}$ in the states from Q. □

This construction is optimal in the worst case, because of the recent result by Han et al. [8] for NFA. Han et al. [8] constructed a pair of witness languages, K and L, recognized by an m-state NFA and by an n-state, respectively, and proved that every NFA recognizing the language $\mathrm{ins}(L, K)$ must have at least $mn + 2m$ states. Assuming that the partition of the alphabet is fixed, with all symbols in Σ_0, and with no brackets, an NIDPDA cannot do anything more than an NFA, and for that reason, the lower bound by Han et al. [8] also applies to NIDPDA. This yields the following result.

Theorem 1. *The state complexity of inserting a language recognized by an m-state NIDPDA into a language of well-nested strings recognized by an n-state NIDPDA is exactly $mn + 2m$.*

On the other hand, if the strings being inserted are not required to be well-nested, then the closure property no longer holds, as shown in the next example.

Example 1. The following language is recognized by an input-driven automaton.

$$L = \{ <^n>^n \mid n \geqslant 1 \}$$

However, no input-driven automaton recognizes the language $\mathrm{ins}(L, <>^*<^*>)$.

Proof. If $\mathrm{ins}(L, <>^*<^*>)$ is recognized by some input-driven automaton, then so is its intersection with the regular language $<^+>^+<^+>^+$. The intersection ensures that the insertion is made exactly in the middle of a string $<^n>^n$.

$$\mathrm{ins}(L, <>^*<^*>) \cap <^+>^+<^+>^+ = \{\, <^{n+1}>^i<^j>^{n+1} \mid n, i, j \geqslant 1 \,\}$$

Let the latter intersection be recognized by a DIDPDA with a set of states Q. Let $n = |Q|^2+1$, and consider the accepting computation on the string $<^n>^n<^n>^n$. In the first half of this computation, that is, on the prefix $<^n>^n$, for every $i \in \{1, \ldots, n\}$, let p_i and q_i be the states reached after reading $<^i$ and $<^n>^{n-i}$, respectively. When the automaton reads the $(i+1)$-th left bracket in the state p_i, it pushes the stack symbol $s_i = \gamma_<(p_i)$, which is popped when reading the $(n-i)$-th right bracket in the state q_{j+1}, so that $q_j = \delta_>(q_{j+1}, s_i)$. There are $|Q|^2$ different pairs (p_i, q_i), and therefore, for some i and j, these pairs coincide: $(p_i, q_i) = (p_j, q_j)$, with $i < j$. Then the two segments of computation between p_{i+1} and p_j and between q_j and q_{i+1}, along with the stack symbols linking them to each other, can be cut, obtaining an accepting computation on the string $<^{n+j-i}>^{n+j-i}<^n>^n$, which does not belong to the language. □

4 Deletion

The deletion operation is a binary operation on languages: $\mathrm{del}(L, K)$ is the set of all strings obtained by taking any string from L and removing any substring belonging to K from that string.

$$\mathrm{del}(L, K) = \{\, xz \mid \exists y \in K : xyz \in L \,\}$$

If the set of strings being deleted consists of well-nested strings, there is the following closure result.

Lemma 2. *Let a language L be recognized by an NIDPDA $\mathcal{B}$ with states Q and stack alphabet Γ. Let K be a language containing only well-nested strings, recognized by an NIDPDA $\mathcal{A}$. Then, the language* $\mathrm{del}(L, K)$ *is recognized by an NIDPDA with the set of states $Q \cup \widetilde{Q}$, where $\widetilde{Q} = \{\, \widetilde{q} \mid q \in Q \,\}$, and with the stack alphabet Γ.*

Proof. The new automaton first simulates $\mathcal{B}$ in the states from $\widetilde{Q}$. At some point, while in some state $\widetilde{q}$, it guesses that a string accepted by $\mathcal{A}$ has been deleted beginning from the next position. Let $q' \in Q$ be any such state of $\mathcal{B}$, that there exists a well-nested string y, which is accepted by $\mathcal{A}$, whereas $\mathcal{B}$ may read y beginning from the state q, and finish reading it in the state q'. Then the simulating automaton may guess such a state q' and continue its simulation, as if it is currently in the state q'.

The construction is effective, because all pairs (q, q') satisfying the above conditions can be determined by first constructing an IDPDA $\mathcal{D}_{q,q'}$ that recognizes the set of all strings y as defined above, and then applying the known emptiness test [2] to this IDPDA. □

If ill-nested strings may be deleted, then, as in the case of insertion, the family is no longer closed under this operation.

Example 2. The following language is recognized by an input-driven automaton.

$$L = \{ <^m \ll^n \gg^n >^m \mid m, n \geqslant 1 \}$$

However, no input-driven automaton recognizes the language $\mathrm{del}(L, \ll^*)$.

Proof. If $\mathrm{del}(L, \ll^*)$ is recognized by some input-driven automaton, then so is its intersection with the regular language $<^+ \gg^+ >^+$, which ensures that *all* double left brackets ($\ll$) are erased.

$$\mathrm{del}(L, \ll^*) \cap <^+ \gg^+ >^+ = \{ <^m \gg^i >^m \mid m, i \geqslant 1 \}$$

If the latter intersection is recognized by a DIDPDA with a set of states Q, then let $m = |Q|$. Then, in the computation on $<^m \gg^m >^m$, the stack must be empty after reading the prefix $<^m \gg^m$. In the last part of the computation, while reading the suffix $>^m$, the automaton passes through a sequence of states $p_0, \ldots, p_m$, behaving like a DFA, with every next state determined by a transition by the empty stack as $p_{i+1} = \delta_>(p_i, \bot)$. Since $m + 1 > |Q|$, two of these states must coincide: $p_i = p_j$, with $0 \leqslant i < j \leqslant m$. Then this segment of the computation can be cut out without affecting the acceptance, and the automaton accepts the string $<^m \gg^m >^{m-(j-i)}$, which is not in the language. □

5 Square root

For a string of the form ww, the string w is its *square root*, denoted by $\sqrt{ww} = w$. The square root of a language is defined as the set of square roots of all its applicable elements.

$$\sqrt{L} = \{ w \mid ww \in L \}$$

The regular languages are closed under this operation. Indeed, for every n-state DFA recognizing L, one can construct a DFA with n^n states that computes the *behaviour function* of the original DFA on the string w: this is a function $f_w \colon Q \to Q$ that maps each state $q \in Q$ to the state reached by the DFA after reading w beginning from the state q. Denote by Q^Q the set of all such functions. Then, the set of states of the constructed DFA is Q^Q.

This construction generalizes to input-driven automata, under the assumption that L contains only well-nested strings. Notably, in such a case, the language $\sqrt{L}$ also contains only well-nested strings.

Lemma 3. *If a language L contains only well-nested strings, and is recognized by a DIDPDA with the set of states Q, then the language $\sqrt{L}$ is recognized by a DIDPDA with the set of states Q^Q and with the stack alphabet $\Sigma_{+1} \times Q^Q$.*

The new DIDPDA processing w constructs the *behaviour function* [18,21] of the original DIDPDA on w. This is one of the basic constructions for DIDPDA, based on the following observation: when a DIDPDA with a set of states Q processes a well-nested string w and begins in a state q, it finishes reading that string in some state $f_w(q)$, where $f_w \colon Q \to Q$ is its *behaviour function on* w, and the stack is left untouched. Thus, f completely characterizes the behaviour of a DIDPDA on w. For any given DIDPDA $\mathcal{A}$, it is possible to construct an n^n-state DIDPDA, where $n = |Q|$, that reaches the end of an input w in a state representing the behaviour of $\mathcal{A}$ on the longest well-nested suffix of w. This construction is necessary for optimal constructions representing operations on DIDPDA [21].

Proof (of Lemma 3). This is a known DIDPDA construction for the behaviour function $f_w \colon Q \to Q$ of $\mathcal{A}$ on the input string w. Then the function $f_w \circ f_w$ is its behaviour of $\mathcal{A}$ on ww, and the simulating automaton accepts in a state $f \in Q^Q$ whenever $(f \circ f)(q_0) \in F$.

A matching lower bound for the state complexity of the square root on DFA is known. It immediately applies to DIDPDA, showing that n^n is the exact complexity of the square root for this model as well.

Lemma 4 (Maslov [15]). *For every $n \geqslant 1$, there exists such an n-state DFA over a 3-symbol alphabet (equivalently, an n-state DIDPDA over the alphabet $\Sigma_{+1} = \Sigma_{-1} = \varnothing$, $\Sigma_0 = \{a, b, c\}$) that every DFA (as well as a DIDPDA) recognizing the language $\sqrt{L}$ requires at least n^n states.*

For NIDPDA, the state complexity of the square root is quite different. As to the authors' knowledge, the number of states in an NFA recognizing the square root of an n-state NFA is not yet known, and it is natural to establish it first.

Theorem 2. *Square root of an n-state NFA is recognized by an n^3-state NFA.*

For every n, there is a language L_n over a three-symbol alphabet, which is recognized by an n-state DFA, whereas every NFA recognizing $\sqrt{L_n}$ must have at least $(n-1)(n-2)(n-3)$ states.

Proof. Let $\mathcal{A} = (\Sigma, Q, Q_0, \delta, F)$ be any n-state NFA. The NFA $\mathcal{B}$ recognizing $\sqrt{L(\mathcal{A})}$ uses the set of triples $Q \times Q \times Q$ as its states. In the beginning of its computation on a string w, it guesses the state p reached by $\mathcal{A}$ after reading w, remembers this state as the last component on the triple, and begins simulating two computations of $\mathcal{A}$, one beginning from an initial state and the other beginning from p. Accordingly, its set of initial states is $\{ (q_0, p, p) \mid q_0 \in Q_0,\ p \in Q \}$, and it uses the following transitions.

$$\delta'\big((q, r, p), a\big) = \{ (q', r', p) \mid q' \in \delta(q, a),\ r' \in \delta(r, a) \}$$

When $\mathcal{B}$ finishes reading w in a state (q, r, p), it has verified that $\mathcal{A}$, upon reading w, can move from q_0 to q, as well as from p to r. If $q = p$, this confirms that A can move from q_0 to r upon reading ww, and if r is accepting, then $\mathcal{B}$ should accept

its input string w. Accordingly, the set of accepting states of $\mathcal{B}$ is $\{\,(p,r,p) \mid p \in Q,\ r \in F\,\}$.

Turning to the lower bound, the language L_n is the standard "universal" witness language, as in Lemma 4. It is recognized by a DFA with the states $\{0,\ldots,n-1\}$, and has the following property: for every function $f\colon \{0,\ldots,n-1\} \to \{0,\ldots,n-1\}$, there is such a string x_f, that, upon reading x_f beginning from a state i, the DFA finishes reading it in the state $f(i)$. The state 0 is initial and $n-2$ is accepting.

Now the lower bound is proved using the standard *fooling set* method. Let (i,j,k) be a triple of states, with $i,k \in \{0,\ldots,n-2\}$, $j \in \{1,\ldots,n-3\}$ and $i \neq k$. Define the string $u_{i,j,k} = x_f$, where f is a function defined by $f(0) = i$, $f(j) = k$ and $f(t) = n-1$ for all remaining arguments. The other string is $v_{i,j,k} = x_g$, where $g(i) = j$, $g(k) = n-2$ and $g(t) = n-1$ for the rest of the arguments. Then, the concatenation $u_{i,j,k}v_{i,j,k}$ maps the initial state to j, and j to the accepting state. This means that the square $(u_{i,j,k}v_{i,j,k})^2$ maps the initial state to the accepting state, and therefore $u_{i,j,k}v_{i,j,k}$ belongs to $\sqrt{L(A)}$. On the other hand, for any two different pairs on this list, $(u_{i,j,k}, v_{i,j,k})$ and $(u_{i',j',k'}, v_{i',j',k'})$, at least one of the mismatched concatenations $u_{i,j,k}v_{i',j',k'}$ and $u_{i',j',k'}v_{i,j,k}$ is not in $\sqrt{L(\mathcal{A})}$. Then, every NFA recognizing the square root must have at least as many states as there are pairs in this fooling set, and there are $(n-1)(n-2)(n-3)$ such pairs. □

The construction for NFA easily generalizes to NIDPDA, whereas the lower bound applies to NIDPDA as it is.

Theorem 3. *Assume that a language L contains only well-nested strings, and let it be recognized by an NIDPDA with set of states Q, and with stack alphabet Γ. Then the language $\sqrt{L}$ is recognized by an NIDPDA with the set of states Q^3 and with the stack alphabet Γ^2.*

For every n, there is a language L_n over an alphabet that consists of four neutral symbols, which is recognized by an n-state DFA, whereas every NIDPDA recognizing $\sqrt{L_n}$ must have at least $(n-1)(n-2)(n-3)$ states.

Thus, the state complexity of the square root has been established as $n^3 - O(n^2)$ both for NFA and for NIDPDA. For the latter, the construction relies on all strings' being well-nested.

In the general case without the well-nestedness assumption, input-driven automata again demonstrate a non-closure.

Example 3. The following language is recognized by an input-driven automaton.

$$L = \{\, >^i <^n >^n <^j \mid i,n,j \geqslant 1 \,\} \cup \{\, <^m >^m <^n >^n \mid m,n \geqslant 1 \,\}$$

Its square root has the following form.

$$\sqrt{L} = \{\, >^n <^n \mid n \geqslant 1 \,\} \cup \{\, <^n >^n \mid n \geqslant 1 \,\}$$

It is not recognized by any input-driven automaton, for any partition of the alphabet.

6 Propotional removals

For a string of even length, $w = a_1 \dots a_{2\ell}$, its *first half*, denoted by $\frac{1}{2}w$, is the string $a_1 \dots a_\ell$, obtained by discarding the second half of w. This operation is extended to languages element-wise, as follows.

$$\tfrac{1}{2}(L) = \{\, \tfrac{1}{2}w \mid w \in L,\ |w| \text{ is even} \,\}$$

As reported by Seiferas and McNaughton [26], Yamada, Stearns and Hartmanis were the first to prove that the regular languages are closed under this operation. Maslov [15] determined its state complexity for DFA as $2^{\Theta(\sqrt{n \log n})}$, relying on the known determinization of unary NFAs using $e^{(1+o(1))\sqrt{n \ln n}}$ states. Later Domaratzki [4] has further investigated its state complexity, and Goč et al. [6] proved that for NFA, the state complexity of "one half" is $\Theta(n^2)$.

For input-driven automata, there is a construction somewhat similar to those used for finite automata. Unfortunately, it cannot anymore rely on determinizing unary NFA, and for that reason is much less efficient with respect to the number of states.

Lemma 5. *For an n-state NIDPDA recognizing a language L, the well-nested subset of the language $\frac{1}{2}(L)$ is recognized by an NIDPDA with $2^{O(n^2)}$ states.*

This time no assumptions are made on the well-nestedness of strings in L, but the construction produces an automaton that defines the intersection $\frac{1}{2}L$ with the set of well-nested strings.

Proof. The states of the new automaton are of the form $(q, \widehat{q}, p)$, where q is the state of the original automaton's ongoing simulation, $\widehat{q}$ is the guessed state of the original automaton in the end of the simulation, whereas p is a state of a certain finite automaton.

For each state q_i of the given input-driven automaton, there exists an NFA $\mathcal{A}_i$ that accepts a string a^ℓ if and only if there exists any string of length ℓ accepted by the input-driven automaton, beginning in the state q_i with the empty stack. Such a finite automaton exists by Parikh's theorem, and is effectively obtained as follows. First, the NIDPDA is transformed to a grammar of size $O(n^2)$. Then, the efficient construction for Parikh's theorem by Esparza et al. [5] is applied to this grammar, producing an NFA of size $2^{O(n^2)}$.

Given the well-nested first half u of some string, the new automaton begins its computation in any state of the form $(q_0, q_i, p_0^{(i)})$, where q_0 is the original automaton's initial state, q_i is any of its states, and $p_0^{(i)}$ is the initial state of the finite automaton $\mathcal{A}_i$. Then, the new automaton simulates the behaviour of the original automaton on u, along with running $\mathcal{A}_i$ on the same string. In the end, the new automaton accepts if its first component reaches the state q_i guessed in the beginning, while the simulated $\mathcal{A}_i$ accepts the string. □

The construction in Lemma 5 should be taken as a rough upper bound on the state complexity of one half. Improving this construction is proposed as an open problem.

If the exact value of $\frac{1}{2}L$ is required, that is, without the intersection with the set of well-nested strings, then there is yet another non-closure result.

Example 4. The following language consists only of well-nested strings and is recognized by an input-driven automaton.

$$L = \{ <^m>^m<^n(cc>)^n cc \mid m, n \geqslant 1 \}$$

Proof. Each string is of length $2m+4n+2$, and its first half is of length $m+2n+1$. This first half belongs to the set $<^*>^*<^*c$ if and only if $m = n$. For this reason, the set of first halves of strings in L, under intersection with the regular language $<^*>^*<^*c$, has the following form.

$$\tfrac{1}{2}L \cap <^*>^*<^*c = \{ <^n>^n<^n c \mid n \geqslant 1 \}$$

The latter language is certainly not recognized by any input-driven automaton, and therefore neither is $\frac{1}{2}L$. □

7 Scattered substrings

For each string $w = a_1 \dots a_n$, any string $a_{i_1} \dots a_{i_\ell}$, with $1 \leqslant i_1 < \ldots < i_\ell \leqslant n$, is its *scattered substring*. Denote by $sub(w)$ the set of all scattered substrings of w, and let $sub(L) = \bigcup_{w \in L} sub(w)$ for a language L. By the Higman–Haines theorem [11,7], the language $sub(L)$ is regular for an arbitrary language L, and can therefore be recognized by an IDPDA without paying attention to the bracket structure, by pushing and popping dummy stack symbols on the brackets.

For regular languages, the state complexity of scattered substrings is $2^{\Theta(n)}$ [12,17], with recent further results by Karandikar et al. [13].

What is the state complexity of this operation for IDPDA? As proved by van Leeuwen [14], for L given by a grammar, the language $sub(L)$ is *effectively* regular. However, van Leeuwen's [14] constructive proof does not include any estimation of the size of the regular language; it uses the finiteness of the basis to prove that the construction terminates. For that reason, no upper bound on the state complexity of scattered substrings for IDPDA is known.

References

1. R. Alur, P. Madhusudan, "Visibly pushdown languages", *ACM Symposium on Theory of Computing* (STOC 2004, Chicago, USA, 13–16 June 2004), 202–211.
2. R. Alur, P. Madhusudan, "Adding nesting structure to words", *Journal of the ACM*, 56:3 (2009).
3. B. von Braunmühl, R. Verbeek, "Input driven languages are recognized in $\log n$ space", *Annals of Discrete Mathematics*, 24 (1985), 1–20.
4. M. Domaratzki, "State complexity of proportional removals", *Journal of Automata, Languages and Combinatorics*, 7:4 (2002), 455–468.

5. J. Esparza, P. Ganty, S. Kiefer, M. Luttenberger, "Parikh's theorem: A simple and direct automaton construction", *Information Processing Letters*, 111:12 (2011), 614–619.
6. D. Goč, A. Palioudakis, K. Salomaa, "Nondeterministic state complexity of proportional removals", *International Journal of Foundations of Computer Science*, 25:7 (2014), 823–836.
7. L. H. Haines, "On free monoids partially ordered by embedding", *Journal of Combinatorial Theory*, 6 (1969), 94–98.
8. Y.-S. Han, S.-K. Ko, T. Ng, K. Salomaa, "State complexity of insertion", *International Journal of Foundations of Computer Science*, 27:7 (2016), 863–878.
9. Y.-S. Han, S.-K. Ko, K. Salomaa, "State complexity of deletion and bipolar deletion", *Acta Informatica*, 53:1 (2016), 67–85.
10. Y.-S. Han, K. Salomaa, "Nondeterministic state complexity of nested word automata", *Theoretical Computer Science*, 410 (2009), 2961–2971.
11. G. Higman, "Ordering by divisibility in abstract algebras", *Proceedings of London Mathematical Society*, s3-2:1 (1952), 326–336.
12. H. Gruber, M. Holzer, M. Kutrib, "More on the size of Higman–Haines sets: effective constructions", *Fundamenta Informaticae*, 91:1 (2009), 105–121.
13. P. Karandikar, M. Niewerth, Ph. Schnoebelen, "On the state complexity of closures and interiors of regular languages with subwords and superwords", *Theoretical Computer Science*, 610:A (2016), 91–107.
14. J. van Leeuven, "Effective construction in well-partially-ordered free monoids", *Discrete Mathematics*, 21:3 (1978), 237–252.
15. A. N. Maslov, "Estimates of the number of states of finite automata", *Soviet Mathematics Doklady*, 11 (1970), 1373–1375.
16. K. Mehlhorn, "Pebbling mountain ranges and its application to DCFL-recognition", *Automata, Languages and Programming* (ICALP 1980, Noordweijkerhout, The Netherlands, 14–18 July 1980), LNCS 85, 422-435.
17. A. Okhotin, "On the state complexity of scattered substrings and superstrings", *Fundamenta Informaticae*, 99:3 (2010), 325–338.
18. A. Okhotin, "Input-driven languages are linear conjunctive", *Theoretical Computer Science*, 618 (2016), 52–71.
19. A. Okhotin, K. Salomaa, "Complexity of input-driven pushdown automata", *SIGACT News*, 45:2 (2014), 47–67.
20. A. Okhotin, K. Salomaa, "Descriptional complexity of unambiguous input-driven pushdown automata", *Theoretical Computer Science*, 566 (2015), 1–11.
21. A. Okhotin, K. Salomaa, "State complexity of operations on input-driven pushdown automata", *Journal of Computer and System Sciences*, 86 (2017), 207–228.
22. A. Okhotin, K. Salomaa, "Edit distance neighbourhoods of input-driven pushdown automata", *Computer Science in Russia* (CSR 2017, Kazan, Russia, 8–12 June 2017), LNCS 10304, 260–272.
23. A. Okhotin, K. Salomaa, "The quotient operation on input-driven pushdown automata", *Descriptional Complexity of Formal Systems* (DCFS 2017, Milan, Italy, 3–5 July 2017), LNCS 10316, 299–310.
24. X. Piao, K. Salomaa, "Operational state complexity of nested word automata", *Theoretical Computer Science*, 410 (2009), 3290–3302.
25. K. Salomaa, "Limitations of lower bound methods for deterministic nested word automata", *Information and Computation*, 209 (2011), 580–589.
26. J. I. Seiferas, R. McNaughton, "Regularity-preserving relations", *Theoretical Computer Science*, 2:2 (1976), 147–154.