Fundamenta Informaticae 154 (2017) 185–199
DOI 10.3233/FI-2017-1560
IOS Press

State Complexity of k-Parallel Tree Concatenation

Yo-Sub Han[*†]
Department of Computer Science
Yonsei University
50 Yonsei-Ro, Seodaemun-Gu
Seoul 120-749, Republic of Korea
emmous@yonsei.ac.kr

Sang-Ki Ko[‡]
Department of Computer Science
University of Liverpool
Ashton Street, Liverpool
L69 3BX, United Kingdom
sangkiko@liverpool.ac.uk

Kai Salomaa[§]
School of Computing
Queen's University
Kingston, Ontario K7L 3N6, Canada
ksalomaa@cs.queensu.ca

Abstract. We give an optimized construction of a tree automaton recognizing the k-parallel, $k \geq 1$, tree concatenation of two regular tree languages. For tree automata with m and n states, respectively, the construction yields an upper bound $(m + \frac{1}{2})(n + 1) \cdot 2^{nk} - 1$ for the state complexity of k-parallel tree concatenation. We give a matching lower bound in the case $k = 2$. We conjecture that the upper bound is tight for all values of k. We also consider the special case where one of the tree languages is the set of all ranked trees and in this case establish a different tight state complexity bound for all values of k.

Keywords: tree automata, state complexity, regular tree languages, tree concatenation

[*]Han was supported by the Basic Science Research Program through NRF funded by MEST (2015R1D1A1A01060097), the Yonsei University Future-leading Research Initiative of 2016 and the IITP grant funded by the Korea government (MSIP) (R0124-16-0002).

[†]Address for correspondence: Department of Computer Science, Yonsei University, 50 Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea

[‡]Ko was partially supported by EPSRC grant "Reachability problems for words, matrices and maps" (EP/M00077X/1)
[§]Salomaa was supported by the Natural Sciences and Engineering Research Council of Canada Grant OGP0147224.

1. Introduction

The state complexity of regular languages is one of the fundamental questions of automata theory and has been studied for over half a century. The state complexity of basic operations on regular languages was considered first by Maslov [1] but the work remained little known. Recent work on operational state complexity was initiated by Yu et al. [2] and A. Salomaa et al. [3] first considered the state complexity of combined operations. Further information on descriptional complexity can be found in the survey by Holzer and Kutrib [4], and Gao et al. [5] review operational state complexity in more detail.

Research on state complexity is not limited to finite automata. The descriptional complexity of other formal systems such as tree automata [6, 7, 8], and input-driven pushdown automata (a.k.a. nested word automata) [9, 10] has been studied. Piao and Salomaa [11] considered the state complexity of sequential and parallel tree concatenation, Han et al. [7] studied the state complexity of star operations on regular tree languages and Ko et al. [12, 13] studied the operational state complexity for subclasses of regular tree languages.

It can be noted that while many aspects of the state complexity of tree languages, including the operational state complexity results for the Boolean operations, are similar to the corresponding results for regular string languages, the state complexity of sequential concatenation of tree languages differs from the string case by an order of magnitude [11]. Since a tree is a two-dimensional structural object whereas a string is a sequence of characters, the concatenation of trees is more complicated than string concatenation and the analogy of the concatenation operation is tree substitution. Different types of tree substitutions have been considered in the literature [14, 15, 11]. Since people rely on trees to represent structural data [16, 17, 18], such as XML schemas or XML instances, tree concatenation is a crucial operation in XML migration or XML schema integration. This observation motivates us to consider a new tree concatenation operation.

The sequential concatenation of two trees is defined by substituting the first tree for a leaf node of the second tree, and the parallel concatenation of two trees is defined by substituting the first tree for all leaves of the second tree. We define the *k-parallel concatenation,* $k \in \mathbb{N}$, of tree languages L_1 and L_2, to consist of trees of L_2 where exactly k leaves have been replaced by some trees of L_1. The k-parallel concatenation can be viewed as an intermediate form between the sequential ($k = 1$) and parallel tree concatenation operations.

We study the state complexity of the k-parallel tree concatenation. We show that $(m + \frac{1}{2}) \cdot (n + 1) \cdot 2^{nk} - 1$ states are sufficient to recognize the k-parallel tree concatenation of tree languages recognized by deterministic bottom-up tree automata with, respectively, m and n states. We present a matching lower bound when $k = 2$. With $k = 1$ the operation coincides with the sequential tree concatenation and a matching lower bound is known [11]. We also consider the state complexity of the same operation in the special case where the second tree language is the set of all trees, and give a significantly improved state complexity upper bound compared with the general bound with $n = 1$. We show that this bound is tight for all values of k.

We consider state complexity in terms of deterministic bottom-up tree automata [14, 19, 15] and consider automata operating on ranked trees where the label of the node determines the number of children. Also the state complexity of unranked tree automata has been considered [8]. However, the

notations become essentially more complicated in the unranked case and, furthermore, the state complexity results for ranked tree automata are similar to the bounds on the numbers of vertical states for unranked tree automata. The previous work on the operational state complexity of tree automata [7, 11] uses ranked tree automata.

2. Tree automata and k-parallel tree concatenation

First we briefly recall basics of finite tree automata and regular tree languages. For all unexplained notions and more information on tree automata we refer the reader to the books [14, 19].

The cardinality of a finite set S is $|S|$. A ranked alphabet Σ is a finite set where each element is assigned with a nonnegative integer rank. We denote the set of elements of rank m by $\Sigma_m \subseteq \Sigma$ for $m \geq 0$. The set of ranked trees over Σ is defined as the smallest set S satisfying the condition: if $m \geq 0, \sigma \in \Sigma_m$ and $t_1, \ldots, t_m \in S$, then $\sigma(t_1, \ldots, t_m) \in S$. The set of all ranked trees over Σ is denoted F_Σ. If $u_1, \ldots, u_k$ are nodes of a tree $t \in F_\Sigma$, and $s_1 \ldots, s_k \in F_\Sigma$, we denote by $t(u_1 \leftarrow s_1, \ldots, u_k \leftarrow s_k)$ the tree obtained from t by replacing the subtree at node u_i with s_i, $i = 1, \ldots, k$. Two nodes of a tree are said to be *independent* if neither one is a predecessor of the other. For a set U of pairwise independent nodes of t and $S \subseteq F_\Sigma$, we denote by $t(U \leftarrow S)$ the set of trees obtained from t by replacing the subtree at each node of U by some tree in S.

A *nondeterministic bottom-up tree automaton* (NTA) is specified by a tuple $A = (\Sigma, Q, Q_f, g)$, where Σ is a ranked alphabet, Q is a finite set of states, $Q_f \subseteq Q$ is a set of final states and g associates to each $\sigma \in \Sigma_m$ a mapping $\sigma_g : Q^m \longrightarrow 2^Q$, where $m \geq 0$. For a tree $t = \sigma(t_1, \ldots, t_m) \in F_\Sigma$, we define inductively the set $t_g \subseteq Q$ by setting $q \in t_g$ if there exist $q_i \in (t_i)_g$, for $1 \leq i \leq m$, such that $q \in \sigma_g(q_1, \ldots, q_m)$. Intuitively, t_g consists of the states of Q that A may reach at the root after reading the tree t. The tree language accepted by A is defined as $L(A) = \{t \in F_\Sigma \mid t_g \cap Q_f \neq \emptyset\}$.

The intermediate stages of a computation of A, or configurations of A, are trees where some leaves may be labeled by states of A. Thus, the set of configurations of A consists of Σ'-trees, where $\Sigma'_0 = \Sigma_0 \cup Q$ and $\Sigma'_m = \Sigma_m$ when $m \geq 1$. The automaton A is a *deterministic bottom-up tree automaton* (DTA) if, for each $\sigma \in \Sigma_m$, where $m \geq 0$, σ_g is a partial function $Q^m \longrightarrow Q$.

The NTAs and DTAs define the class of *regular tree languages.* A regular tree language (over a ranked alphabet) has a unique state minimal DTA and minimality of a DTA can be checked similarly as for ordinary finite automata by verifying that all states are reachable and pairwise inequivalent [14, 19].

Note that we allow a deterministic tree automaton to be incomplete, that is, some transitions may be undefined. Naturally when dealing with ranked tree automata, the optimal sizes of a complete and an incomplete DTA differ by at most one state. On the other hand, descriptional complexity results for unranked tree automata change significantly if one would require all transitions to be defined. The transitions of an unranked tree automaton A are defined in terms of regular languages, called horizontal languages. Each horizontal language is specified by a an incomplete deterministic finite automaton (DFA) that processes strings of states of the bottom-up computation, or *vertical states,* and the size of A is the sum of the number of vertical states and the number of states of the DFAs specifying the horizontal languages. Requiring all transitions to be defined changes, in the worst case, dramatically the number of horizontal states [7, 11]. In order to keep our results compatible with the bounds for the

numbers of vertical states in unranked tree automata, we use here incomplete deterministic tree automata.

Let $A = (\Sigma, Q_A, Q_{A,F}, g_A)$ be a DTA. Denote $Q'_A = Q_A \cup \{d_A\}$, where d_A is a new element. The *completion of* A is the DTA $\overline{A} = (\Sigma, Q'_A, Q_{A,F}, g'_A)$ where for $\sigma \in \Sigma_m$, $m \geq 0$, and $q_1, \ldots, q_m \in Q'_A$ we set

$$\sigma_{g'_A}(q_1, \ldots, q_m) = \begin{cases} \sigma_{g_A}(q_1, \ldots, q_m) & \text{if } \sigma_{g_A}(q_1, \ldots, q_m) \text{ is defined,} \\ d_A & \text{otherwise.} \end{cases}$$

Note that when all transitions of A are defined, the state d_A of $\overline{A}$ is useless. If A is a minimal DTA with some transitions undefined, then $\overline{A}$ is the equivalent minimal complete DTA.

Here we consider a tree concatenation where the substitutions occur at exactly k leaves. We denote the set of leaves of a tree t by $\texttt{leaf}(t)$. For $k \geq 1$, we define the *k-parallel tree concatenation* of a set of trees $T_1 \subseteq F_\Sigma$ and $t_2 \in F_\Sigma$ as follows:

$$T_1 \cdot^p_k t_2 = \{\, t_2(S \leftarrow T_1) \mid S \subseteq \texttt{leaf}(t_2), |S| = k \,\}.$$

The operation is extended in the natural way for two sets of trees $T_1, T_2 \subseteq F_\Sigma$ by setting $T_1 \cdot^p_k T_2 = \bigcup_{t_2 \in T_2} T_1 \cdot^p_k t_2$. Figure 1 illustrates 4-parallel tree concatenation.

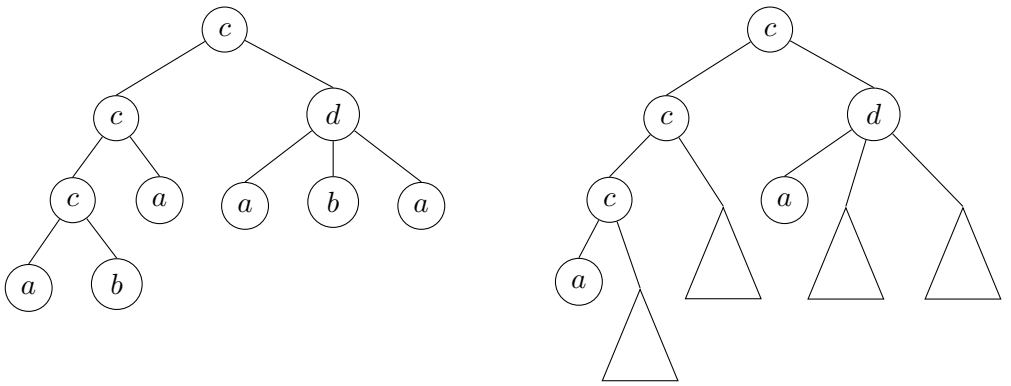

Figure 1: The left figure shows a tree t and the right figure shows a tree obtained as a result of 4-parallel tree concatenation of a tree language T and the tree t. The triangles depict trees in T.

The 1-parallel concatenation, roughly speaking, coincides with sequential tree concatenation [11] with the minor technical difference that we allow substitutions to occur at all leaves while the sequential concatenation of [11] specifies also the labels of leaves to be substituted.

We consider also an "at least k"-parallel concatenation operation $T_1 \cdot^p_{\geq k} T_2 = \bigcup_{m \geq k} T_1 \cdot^p_m T_2$, where at least k leaves of a tree of T_2 are replaced by some trees of T_1.

The following technical notion will be used later. Let $t \in F_\Sigma$ and $L \subseteq F_\Sigma$. For $j \in \mathbb{N}$, the *L-j-quotient of* t is $t_{L\text{-}j\text{-quot}} = \{\, r \in F_\Sigma \mid t \in L \cdot^p_j r \,\}$. Intuitively, the L-j-quotient of t consists of trees that can be completed to t by replacing exactly j leaf nodes by trees of L.

3. Results

We begin by giving an upper bound for the state complexity of k-parallel tree concatenation. The upper bound for 1-parallel concatenation coincides with the corresponding bound for sequential σ-concatenation [11] although the definitions have a minor technical difference.

Lemma 3.1. Let $k \in \mathbb{N}$ and let A and B be DTAs with m and n states, respectively. The tree language $L(A) \cdot_k^p L(B)$ can be recognized by a DTA of size $(m + \frac{1}{2}) \cdot (n + 1) \cdot 2^{nk} - 1$.

Proof:
Let $\overline{A} = (\Sigma, Q'_A, Q_{A,F}, g_A)$ and $\overline{B} = (\Sigma, Q'_B, Q_{B,F}, g_B)$ be the completions of A and B. (We denote the transition relations of the completions simply as g_A and g_B.) Thus, if Q_A and Q_B are the state sets of A and B, respectively, we have $Q'_A = Q_A \cup \{d_A\}$ and $Q'_B = Q_B \cup \{d_B\}$ and $|Q'_A| = m + 1$, $|Q'_B| = n + 1$. We construct a DTA $C = (\Sigma, P, P_F, h)$ for the tree language $L(A) \cdot_k^p L(B)$ where $P = Q'_A \times Q'_B \times (2^{Q_B})^k$, and P_F consists of tuples

$$(q, p, R_1, \ldots, R_k),\ q \in Q'_A, p \in Q'_B, R_i \subseteq Q_B,\ 1 \le i \le k,\ \text{where } R_k \cap Q_{B,F} \neq \emptyset,$$

and the transitions of h are defined as follows: For $\sigma \in \Sigma_0$ we define

$$\sigma_h = \begin{cases} (\sigma_{g_A}, \sigma_{g_B}, \underbrace{\emptyset, \ldots, \emptyset}_{k \text{ copies}}) & \text{if } \sigma_{g_A} \in Q'_A - Q_{A,F}, \\ (\sigma_{g_A}, \sigma_{g_B}, \{\tau_{g_B} \in Q_B \mid \tau \in \Sigma_0\}, \underbrace{\emptyset, \ldots, \emptyset}_{k-1 \text{ copies}}) & \text{if } \sigma_{g_A} \in Q_{A,F}. \end{cases} \tag{1}$$

Note that the set $\{\tau_{g_B} \in Q_B \mid \tau \in \Sigma_0\}$ is nonempty unless $L(B) = \emptyset$. To define the transitions of C on r-ary symbols, $r \ge 1$, we denote an arbitrary state of P as

$$(q, R_0, R_1, \ldots, R_k),\ \ q \in Q'_A,\ R_0 = \{p\},\ p \in Q'_B,\ R_1, \ldots, R_k \subseteq Q_B.$$

For notational convenience, the state of Q'_B appearing as the second component of an element of P is denoted as a singleton subset with subindex 0. The idea is that the states of the set R_i, $0 \le i \le k$, are all states that the DTA B may be in assuming that in the current subtree exactly i leaf nodes have been replaced by a tree of $L(A)$. The set R_0 consists of d_B if the computation of B is undefined when no leaf has been replaced by a tree of $L(A)$.

In the following, elements of a set R_i are called *elements of index i,* $0 \le i \le k$.

Now consider $\sigma \in \Sigma_r$, $r \ge 1$, and let $\mathbf{v_i} = (q_i, R_{i,0}, R_{i,1}, \ldots, R_{i,k})$, $1 \le i \le r$, be any r states of P, where $q_i \in Q'_A$, $R_{i,0} = \{r_i\}$, $r_i \in Q'_B$, $R_{i,j} \subseteq Q_B$, $1 \le j \le k$. Now define

$$\sigma_h(\mathbf{v_1}, \ldots, \mathbf{v_r}) = (\sigma_{g_A}(q_1, \ldots, q_r), \sigma_{g_B}(r_1, \ldots, r_r), S_1, \ldots, S_k),$$

where the sets $S_i \subseteq Q_B$, $1 \le i \le k$, are defined as follows. For $i \in \{2, \ldots, k\}$,

$$S_i = \{\ \sigma_{g_B}(u_1, \ldots, u_r) \in Q_B \mid u_j \in R_{j,\ell_j},\ j = 1, \ldots, r, \text{ and } \sum_{j=1}^{r} \ell_j = i\ \}. \tag{2}$$

Note that the definition requires that the elements $\sigma_{g_B}(u_1, \ldots, u_r)$ must be in Q_B and, in particular, this means that a choice $\ell_j = 0$ can be made only when the only element of R_{j,ℓ_j} is not the dead state d_B. Note also that in the above definition of S_i some indices ℓ_{j_1} and ℓ_{j_2} may be equal for $j_1 \neq j_2$.

Finally, for $i = 1$ define

$$S_1 = \{\ \sigma_{g_B}(u_1, \ldots, u_r) \in Q_B \mid (\exists 1 \leq j \leq r)\, u_j \in R_{j,1} \text{ and } u_\ell \in R_{\ell,0} \text{ when } \ell \neq j\ \} \cup X, \quad (3)$$

$$\text{where} \quad X = \begin{cases} \{\tau_{g_B} \in Q_B \mid \tau \in \Sigma_0\} & \text{if } \sigma_{g_A}(q_1, \ldots, q_r) \in Q_{A,F}, \\ \emptyset & \text{if } \sigma_{g_A}(q_1, \ldots, q_r) \notin Q_{A,F}. \end{cases}$$

The set S_1 is obtained by "combining" exactly one element of index one (from one of the preceding states of C) with other elements of index zero. Additionally, when the first component simulating the computation of the DTA A reaches a final state, the set S_1 contains all elements the DTA B may reach at any leaf node.

Consider a computation of the DTA C in a state $(q, R_0, R_1, \ldots, R_k)$. The intuitive meaning of the components of the state is a follows. The first component q simply simulates the computation of the DTA A. The elements in a component R_i, $0 \leq i \leq k$, consist of all possible states of Q_B that the DTA B may reach assuming exactly i leaves below have been substituted by a tree of $L(A)$. Recall that, according to our convention, R_0 is always a singleton subset of Q'_B, where the added state d_B represents the possibility that the computation of B has failed.

The claim below verifies that the states appearing in computations of C actually satisfy the properties described in the above paragraph.

Claim 1. Let $t \in F_\Sigma$ and $t_h = (q, R_0, R_1, \ldots, R_k) \in P$. Then
(i) $q = t_{g_A}$, and $R_0 = \{t_{g_B}\}$ (which allows the possibility $t_{g_B} = d_B$),
(ii) $R_i = \{\ r_{g_B} \in Q_B \mid r \in t_{L(A)\text{-}i\text{-quot}}\ \}$, $1 \leq i \leq k$.

Proof of the claim. Item (i) follows directly from the definition of the transition function h which on the first (respectively, second) component of the states of C directly simulates the computation of the DTA A (respectively, B). Below we verify claim (ii) using structural induction on the input tree t.

When $t = \sigma \in \Sigma_0$, the transition rules in (1) make R_1 to be the set of all states that B can reach at a leaf node exactly when σ_{g_A} is a final state of A and R_1 to be $\emptyset$ otherwise. Note also that when t is an element of Σ_0, for all $j \geq 2$, the $L(A)$-j-quotient of t is the empty set.

Next consider $t = \sigma(t_1, \ldots, t_r)$, $\sigma \in \Sigma_r$, $r \geq 1$, where the claim holds for trees t_i for $1 \leq i \leq r$.

(a) We note that, for $2 \leq j \leq k$, the $L(A)$-j-quotient of t consists of all trees that can be completed to t by replacing exactly j leaves of the trees $t_1, \ldots, t_r$ by a tree of $L(A)$, that is,

$$t_{L(A)\text{-}j\text{-quot}} = \{\ \sigma(s_1, \ldots, s_r) \mid s_\ell \in (t_\ell)_{L(A)\text{-}i_\ell\text{-quot}},\ \ell = 1, \ldots, r,\ 0 \leq i_\ell \leq k,\ \sum_{\ell=1}^{r} i_\ell = j\ \}.$$

Since inductively the claim holds for the trees $t_1, \ldots, t_r$, this means that the rule (2) selects the correct states for the set R_j.

(b) In the case $j = 1$, we note that $t_{L(A)\text{-1-quot}}$ contains all trees $\sigma(s_1, \ldots, s_r)$, where for some $1 \leq \ell \leq r$, $s_\ell \in (t_\ell)_{L(A)\text{-1-quot}}$ and $s_x = t_x$ when $x \neq \ell$ and $1 \leq x \leq r$. Additionally when $t \in L(A)$, the set $t_{L(A)\text{-1-quot}}$ contains all elements of $\tau \in \Sigma_0$ such that $\tau_{g_B} \neq d_B$. This means that again the rule (3) assigns the correct states to the set R_1. This concludes the proof of Claim 1. ◁

Claim 1 together with the choice of the set of final states P_F of C implies that the DTA C recognizes the tree language $L(A) \cdot_k^p L(B)$. The number of states of C is $(m+1) \cdot (n+1) \cdot 2^{nk}$. It remains to verify that some states $(q, R_0, R_1, \ldots, R_k) \in P$ of C must always be unreachable or cannot be used in any accepting computation. We note that when $q \in Q_{A,F}$ and the state $(q, R_0, R_1, \ldots, R_k)$ is reached either according to rule (1) or according to rule (2), then R_1 must contain all states that B can reach at leaf nodes. Since the statement of Lemma 3.1 holds trivially when $L(A) = \emptyset$ or $L(B) = \emptyset$, we can assume that the DTAs A and B both accept a nonempty tree language. In this case $|Q_{A,F}| \geq 1$ and B must reach at least one state on leaf nodes, which gives at least $(n+1) \cdot 2^{nk-1}$ unreachable states. Furthermore, the state $(d_A, d_B, \emptyset, \ldots, \emptyset)$ is a dead state of C and can be omitted. By omitting the unreachable states and the dead state, C has the claimed number of states. □

3.1. Matching lower bound for 2-parallel concatenation

Next we give a lower bound construction that reaches the upper bound of Lemma 3.1 when $k = 2$. We choose the ranked alphabet $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$, where $\Sigma_0 = \{d\}, \Sigma_1 = \{a, b, e\}$, and $\Sigma_2 = \{c\}$. Let $M_A = (\Sigma, Q_A, Q_{A,F}, g_A)$ be the DTA, where $Q_A = \{0, 1, \ldots, m-1\}$, $Q_{A,F} = \{m-1\}$, and the transition function g_A is defined by setting

- $d_{g_A} = 0, \quad a_{g_A}(i) = i,\ 0 \leq i \leq m-1,$
- $b_{g_A}(i) = i+1,\ 0 \leq i \leq m-2$, and $b_{g_A}(m-1) = 0,$
- $e_{g_A}(i)$ is undefined for $0 \leq i \leq m-1,$

$$c_{g_A}(i,j) = \begin{cases} i & \text{if } i = j, \\ 0 & \text{if } i \neq j, \text{ and } i = 0 \text{ or } j = 0, \\ \text{undefined} & \text{otherwise,} \end{cases} \qquad 0 \leq i, j \leq m-1. \tag{4}$$

Let $M_B = (\Sigma, Q_B, Q_{B,F}, g_B)$ be a DTA, where $Q_B = \{0, 1, \ldots, n-1\}$, $Q_{B,F} = \{n-1\}$, and the transition function g_B is defined by setting

- $d_{g_B} = 0, \quad a_{g_B}(i) = i+1,\ 0 \leq i \leq n-2$, and $a_{g_A}(n-1) = 0,$
- $b_{g_B}(i) = e_{g_B}(i) = i,\ 0 \leq i \leq n-1$, and,
- $c_{g_B}(i,j)$ is defined as the right side of equation (4) for $0 \leq i, j \leq n-1$.

Roughly, speaking the DTA A counts unary symbols b modulo m and B counts unary symbols a modulo n. The operation on the binary symbols defined by conditions (4) is less intuitive and is designed to guarantee that all states of the DTA constructed in the proof of Lemma 3.1 are reachable and pairwise inequivalent.

In the following, we establish that the DTA $M_C = (\Sigma, Q_C, Q_{C,F}, g_C)$ constructed in the proof of Lemma 3.1 to recognize the 2-parallel concatenation of the tree languages $L(M_A)$ and $L(M_B)$ is minimal. This implies then that $L(M_A)$ and $L(M_B)$ provide a lower bound that reaches the upper bound in Lemma 3.1 with $k = 2$.

Let $M_C = (\Sigma, Q_C, Q_{C,F}, g_C)$ be the DTA for the tree language $L(M_A) \cdot_2^p L(M_B)$ constructed as in the proof of Lemma 3.1. We denote the dead state added to the completion of M_A (respectively, completion of M_B) as m (respectively n). Then, the set of states Q_C consists of all quadruples

$$(p, q, S_1, S_2), \qquad 0 \leq p \leq m, \;\; 0 \leq q \leq n, \;\; S_1, S_2 \subseteq \{0, 1, \ldots, n-1\},$$

where if $p = m - 1$ then $0 \in S_1$, and if $S_1 = \emptyset$ and $S_2 = \emptyset$ then $p \neq m$ or $q \neq n$. Therefore, the number of states of M_C is $(m + \frac{1}{2}) \cdot (n + 1) \cdot 2^{2n} - 1$. In the following two lemmas, we show that all states of M_C are reachable and pairwise inequivalent.

Lemma 3.2. All states of M_C are reachable.

Proof:

Case 1: Using induction on $|S_1|$, we first show that all the states $(p, q, S_1, \emptyset)$ where $0 \leq p \leq m$, $S_1 \subseteq \{0, 1, \ldots, n-1\}$, $0 \leq q \leq n$ and $p = m - 1$ implies $0 \in S_1$ are reachable. Note that if $p = m - 1$ then $0 \in S_1$ by the construction of M_C. The DTA M_C assigns the state $(0, 0, \emptyset, \emptyset)$ to a leaf symbol. When $|S_1| = 0$, the state $(i, j, \emptyset, \emptyset)$, for $0 \leq i \leq m - 2$ and $0 \leq j \leq n - 1$, is reachable from $(0, 0, \emptyset, \emptyset)$ by reading a sequence of unary symbols $a^j b^i$. Recall that $(m - 1, j, \emptyset, \emptyset)$ is not a state of Q_C. The state $(m, j, \emptyset, \emptyset)$ is reached by reading the unary symbol e in state $(0, j, \emptyset, \emptyset)$. The state $(i, n, \emptyset, \emptyset)$ is reachable by reading a binary symbol c with two states $(i, j_1, \emptyset, \emptyset)$ and $(i, j_2, \emptyset, \emptyset)$ such that $j_1 \neq j_2$, $j_1 \neq 0$ and $j_2 \neq 0$.

For an integer $x \geq -n$, we denote $\overline{x} = x$ if $x \geq 0$, and $\overline{x} = n + x$ if $0 \geq x \geq -n$.

Let us inductively assume that for $|S_1| \leq x$, all the states $(i, j, S_1, \emptyset)$ where $0 \leq i \leq m, 0 \leq j \leq n$ are reachable. Then, we show that any state $(i', j', S_1', \emptyset)$, for $0 \leq i' \leq m, 0 \leq j' \leq n$ and $|S_1'| = x + 1$, is reachable.

(i) First, consider the case when $i' \neq m - 1$. Let $S_1' = \{s_1, s_2, \ldots, s_{x+1}\}$ where $s_1 > s_2 > \cdots > s_x > s_{x+1}$. Let $P = \{s_1 - s_{x+1}, s_2 - s_{x+1}, \ldots, s_x - s_{x+1}\}$. There are two cases to consider as follows:

(ia) Case $0 \leq j' \leq n - 1$: According to the inductive assumption, the state $(0, \overline{j' - s_{x+1}}, P, \emptyset)$ is reachable. Then the state $(m-1, \overline{j' - s_{x+1}}, P \cup \{0\}, \emptyset)$ is reachable from $(0, \overline{j' - s_{x+1}}, P, \emptyset)$ by reading a sequence of unary symbols b^{m-1}. We reach the state $(i', j', S_1', \emptyset), 0 \leq i' \leq m-2$ from $(m-1, \overline{j' - s_{x+1}}, P \cup \{0\}, \emptyset)$ by reading a sequence of unary symbols $b^{i'+1} a^{s_{x+1}}$. Lastly, the state $(m, j', S_1', \emptyset)$ is reachable by reading a unary symbol e from $(m - 1, j', S_1', \emptyset)$.

(ib) Case $j' = n$: According to the inductive assumption, the state $(0, n, P, \emptyset)$ is reachable. Then the state $(m - 1, n, P \cup \{0\}, \emptyset)$ is reachable from $(0, n, P, \emptyset)$ by reading a sequence of unary symbols b^{m-1}. Now the state $(i', n, S_1', \emptyset), 0 \leq i' \leq m - 2$ is reachable from $(m - 1, n, P \cup \{0\}, \emptyset)$ by reading a sequence of unary symbols $b^{i'+1} a^{s_{x+1}}$.

(ii) Now consider the case when $i' = m-1$. By the construction, $0 \in S'_1$ if $i' = m-1$. We know that the state $(m-2, j', S'_1 - \{0\}, \emptyset)$ is reachable by the inductive assumption. Then, the state $(m-1, j', S'_1, \emptyset)$ is reachable by reading a unary symbol b from $(m-2, j', S'_1 - \{0\}, \emptyset)$.

Case 2: Using induction on $|S_2|$, we show that all the states $(p, q, \emptyset, S_2)$ where $0 \le p \le m, p \ne m-1$, $S_2 = \{0, 1, \ldots, n-1\}$ and $0 \le q \le n$ are reachable. From Case 1 we know that all states $(p, q, \emptyset, \emptyset)$ for $0 \le p \le m, p \ne m-1, 0 \le q \le n$ are reachable. Let us assume that for $|S_2| \le x$, all the states $(i, j, \emptyset, S_2), 0 \le i \le m, 0 \le j \le n, S_2 \subseteq \{0, 1, \ldots, n-1\}$ are reachable. We show that any state $(i', j', \emptyset, S'_2), 0 \le i' \le m, 0 \le j' \le n, |S'_2| = x+1$ is reachable.

(iii) We first consider the case when $i' \ne m-1$. Let $S'_2 = \{s_1, s_2, \ldots, s_{x+1}\}$ where $s_1 > s_2 > \cdots > s_x > s_{x+1}$. We separate the following subcases:

(iii-a) Case $0 \le j' \le n-1$ and $j' \in S'_2$: Let

$$P = \{s_1 - s_l, s_2 - s_l, \ldots, s_{l-1} - s_l, \overline{s_{l+1} - s_l}, \ldots, \overline{s_x - s_l}, \overline{s_{x+1} - s_l}\}.$$

It is clear that $0 \notin P$ and $|P| = x$. Assume that $j' = s_l$ such that $1 \le l \le x+1$ since $j' \in S'_2$. We already have shown that the state $(i', 0, P, \emptyset)$ is reachable. We choose a tree $t = a^{j'}c(x, x) \in F_{\Sigma'}$ and consider the computation of M_C on $t(x \leftarrow (i', 0, P, \emptyset))$. Here Σ' consists of elements of Σ with an auxiliary nullary symbol x and, thus, $t(x \leftarrow (i', 0, P, \emptyset))$ is a configuration of M_C. The DTA M_C reaches $(i', 0, \emptyset, P \cup \{0\})$ after computing the binary symbol c and moves to $(i', j', \emptyset, S'_2)$ by reading a sequence of unary symbols $a^{j'}$. Now we know the state $(i', j', \emptyset, S'_2)$ is reachable when $j' \in S'_2$.

(iii-b) Case $0 \le j' \le n-1$ and $j' \notin S'_2$: First note that $|S'_2| \le n-1$ in this case since $j \notin S'_2$. Let

$$P = \{s_1 - s_{x+1}, s_2 - s_{x+1}, \ldots, s_x - s_{x+1}\}.$$

From Case 1 above, we know that the state $(i', \overline{j' - s_{x+1}}, P, \emptyset)$ is reachable. Choose a tree $t = a^{s_{x+1}}c(x, x) \in F_{\Sigma'}$. Then, the computation of M_C on $t(x \leftarrow (i', \overline{j' - s_{x+1}}, P, \emptyset))$ reaches the state $(i', j', \emptyset, S'_2)$.

(iii-c) Case $j' = n$: Choose the same tree $t = a^{s_{x+1}}c(x, x) \in F_{\Sigma'}$ as in (iii-b) and consider the computation on $t(x \leftarrow (i', n, P, \emptyset))$. We can verify that M_C assigns the state $(i', n, \emptyset, S'_2)$ to the root of t.

(iv) Now consider the case when $i' = m-1$. Again, according to the construction, in this case 0 must be in the third component of the state and hence $(m-1, j', \emptyset, S'_2)$ is not a state of M_C.

Case 3. Finally, we move to the last step of the proof. Above in Case 2 we have shown that any state $(i, j, \emptyset, S_2), 0 \le i \le m, 0 \le j \le n, S_2 \subseteq \{0, 1, \ldots, n-1\}$ is reachable. Inductively assume that for $|S_1| \le x$, all the states (i, j, S_1, S_2) where $0 \le i \le m, 0 \le j \le n, S_2 \subseteq \{0, 1, \ldots, n-1\}$ are reachable. Then, we show that any state $(i', j', S'_1, S'_2), 0 \le i' \le m, 0 \le j' \le n, |S'_1| = x+1, S'_2 \subseteq \{0, 1, \ldots, n-1\}$ is reachable using induction on $|S'_1|$.

(v) Analogously with Case 1 (i) and Case 2 (iii), we first consider the situation where $i' \neq m-1$. Let $S'_1 = \{s_1, s_2, \ldots, s_{x+1}\}$ where $s_1 > s_2 > \cdots > s_x > s_{x+1}$ and $S'_2 = \{t_1, t_2, \ldots, t_y\}$ where $t_1 > t_2 > \cdots > t_y$ and $0 \leq y \leq n$. We also denote

$$P_1 = \{s_1 - s_{x+1}, s_2 - s_{x+1}, \ldots, s_x - s_{x+1}\} \text{ and } P_2 = \{\overline{t_1 - s_{x+1}}, \overline{t_2 - s_{x+1}}, \ldots, \overline{t_y - s_{x+1}}\}.$$

There are two subcases to consider as follows:

(v-a) Case $0 \leq j' \leq n-1$: According to the inductive assumption, the state $(0, \overline{j' - s_{x+1}}, P_1, P_2)$ is reachable. Then the state $(m-1, \overline{j' - s_{x+1}}, P_1 \cup \{0\}, P_2)$ is reachable from $(0, \overline{j' - s_{x+1}}, P_1, P_2)$ by reading a sequence of unary symbols b^{m-1}. We reach the state $(i', j', S'_1, S'_2), 0 \leq i' \leq m-2$ from $(m-1, \overline{j' - s_{x+1}}, P_1 \cup \{0\}, P_2)$ by reading a sequence of unary symbols $b^{i'+1}a^{s_{x+1}}$. The state (m, j', S'_1, S'_2) is reachable by reading a unary symbol e from $(m-1, j', S'_1, S'_2)$.

(v-b) Case $j' = n$: According to the inductive assumption, the state $(0, n, P_1, P_2)$ is reachable. Then the state $(m-1, n, P_1 \cup \{0\}, P_2)$ is reachable from $(0, n, P_1, P_2)$ by reading a sequence of unary symbol b^{m-1}. Now the state $(i', n, S'_1, S'_2), 0 \leq i' \leq m-2$ is reachable from $(m-1, n, P_1 \cup \{0\}, P_2)$ by reading a sequence of unary symbols $b^{i'+1}a^{s_{x+1}}$.

(vi) Now consider the case when $i' = m-1$. Recall that $0 \in S'_1$ if $i' = m-1$. The state $(m-2, j', S'_1 - \{0\}, S'_2)$ is reachable by the inductive assumption. Then, the state $(m-1, j', S'_1, S'_2)$ is reachable by reading a unary symbol b from $(m-2, j', S'_1 - \{0\}, S'_2)$.

□

Note that in the proof of Lemma 3.2, Case 3 has similarities with Case 1. We need to prove Case 1 first because it is needed to establish Case 2, which in turn is used for Case 3.

Lemma 3.3. All states of M_C are pairwise inequivalent.

Proof:
Let $(i_1, j_1, S_{1,1}, S_{1,2})$ and $(i_2, j_2, S_{2,1}, S_{2,2})$ be any distinct states of M_C.

(i) First, we consider the case when $j_1 \neq j_2$. We establish that $(i_1, j_1, S_{1,1}, S_{1,2})$ and $(i_2, j_2, S_{2,1}, S_{2,2})$ are inequivalent by the computation of M_C on the tree $t = a^{n-1-j_1}c(q, (m, \emptyset, \emptyset, \{j_1\})) \in F_{\Sigma'}$. Let us consider the computation of M_C on $t(q \leftarrow (i_1, j_1, S_{1,1}, S_{1,2}))$. The DTA M_C reaches the state $(m, n, \emptyset, \{n-1\})$, which is a final state of M_C, after the computation. However, the computation of M_C on $t(q \leftarrow (i_2, j_2, S_{2,1}, S_{2,2}))$ does not.

(ii) Next consider the case $S_{1,1} \neq S_{2,1}$. Without loss of generality, we have a state $s \in S_{1,1} - S_{2,1}$. Then, we choose a tree $t = a^{n-1-s}c(q, (m, n, \{s\}, \emptyset))$ and consider the computations of M_C on $t(q \leftarrow (i_1, j_1, S_{1,1}, S_{1,2}))$ and $t(q \leftarrow (i_2, j_2, S_{2,1}, S_{2,2}))$. The former computation reaches the state $(m, n, \emptyset, \{n-1\})$, which is a final state of M_C whereas the latter does not.

(iii) Third assume that $S_{1,2} \neq S_{2,2}$ and consider $s \in S_{1,2} - S_{2,2}$. Choose a tree $t = a^{n-1-s}c(q, (m, s, \emptyset, \emptyset))$. Then, the computation on $t(q \leftarrow (i_1, j_1, S_{1,1}, S_{1,2}))$ reaches a final state whereas the computation on $t(q \leftarrow (i_2, j_2, S_{2,1}, S_{2,2}))$ does not.

(iv) Finally, assume $i_1 \neq i_2$. Choose a tree $t = c(q, (i_1, n, \emptyset, \emptyset))$ and consider two computations $t(q \leftarrow (i_1, j_1, S_{1,1}, S_{1,2}))$ and $t(q \leftarrow (i_2, j_2, S_{2,1}, S_{2,2}))$ in the DTA M_C; we have $(i_1, n, \emptyset, \emptyset)$ and $(m, n, \emptyset, \emptyset)$, respectively, as the results of the computations. After reading a sequence of unary symbols b^{m-1-i_1}, the former state reaches $(m-1, n, \{0\}, \emptyset)$ while the latter state is still a sink state. □

We can now state our main result.

Theorem 3.4. If A and B are DTAs with m and n states, respectively, then the tree language $L(A) \cdot_k^p L(B)$ can be recognized by a DTA with $(m + \frac{1}{2}) \cdot (n+1) \cdot 2^{nk} - 1$ states. For every $m, n \geq 2$ and $k = 1$ or $k = 2$, there exist DTAs A and B with m and n states, respectively, such that any DTA recognizing $L(A) \cdot_k^p L(B)$ needs $(m + \frac{1}{2}) \cdot (n+1) \cdot 2^{2nk} - 1$ states.

Proof:
The upper bound follows from Lemma 3.1. The lower bound for $k = 2$ follows by Lemmas 3.2 and 3.3.

The lower bound for $k = 1$ follows from the tight lower bound for sequential concatenation [11]. Note that the sequential σ-concatenation of [11] differs from 1-parallel concatenation in the sense that the former specifies a particular nullary symbol where the substitutions must be performed. However, the corresponding lower bound construction ([11] Theorem 5) uses as the second component a tree language defined over a ranked alphabet with only one nullary symbol, which means that the same lower bound applies for 1-parallel concatenation. □

We conjecture that the upper bound of Lemma 3.1 is tight for general values of k, but we do not have a proof for this claim.

4. Concatenation of a regular tree language with F_Σ

As a special case we consider the k-parallel tree concatenation of a regular tree language and the set of all trees F_Σ. We derive for all $k \geq 1$ a tight state complexity bound that is significantly lower than the upper bound of Theorem 3.4 with $n = 1$.

The tree language $L(A) \cdot_k^p F_\Sigma$ consists of Σ-trees with at least k "independent" subtree occurrences of the trees in $L(A)$. The independence of the subtree occurrences means that we do not count twice if a large subtree belonging to $L(A)$ contains subtrees that are also in $L(A)$. This observation is formalized in the lemma below.

Lemma 4.1. Let L be a tree language defined over Σ and $k \in \mathbb{N}$. Then, $L \cdot_{\geq k}^p F_\Sigma = L \cdot_k^p F_\Sigma$.

Proof:
From the definition of the operations, it is clear that, for any L_1 and L_2, $L_1 \cdot_k^p L_2 \subseteq L_1 \cdot_{\geq k}^p L_2$. For the converse inclusion, consider a tree $t_0 \in L \cdot_{\geq k}^p F_\Sigma$ that is obtained by substituting j trees $t_1, \ldots, t_j \in L$ for leaf nodes $u_1, \ldots, u_j$ of a tree $s \in F_\Sigma$, where $j > k$. (If $j = k$ we have nothing to prove.) Denote $r = s(u_{k+1} \leftarrow t_{k+1}, \ldots, u_j \leftarrow t_j)$. Now $t_0 = r(u_1 \leftarrow t_1, \ldots, u_k \leftarrow t_k) \in L \cdot_k^p r \subseteq L \cdot_k^p F_\Sigma$. □

The set F_Σ is recognized by a one state DTA and with $n = 1$ Lemma 3.1 yields an upper bound $(m + \frac{1}{2}) \cdot 2^{k+1} - 1$ for the k-parallel concatenation of an m-state DTA and F_Σ. The following lemma, together with Lemma 4.1, yields an improved upper bound that is then shown to be tight.

Lemma 4.2. Let $k \in \mathbb{N}$ and let A be a DTA with m states. The tree language $L(A) \cdot^p_{\geq k} F_\Sigma$ can be recognized by a DTA of size $m + k$.

Proof:
Let $\overline{A} = (\Sigma, Q', Q_F, g)$ be the completion of A. Thus, if Q is the state set of A, we have $Q' = Q \cup \{d\}$ and $|Q'| = m + 1$. We construct a DTA $B = (\Sigma, P, P_F, h)$ for the tree language $L(A) \cdot^p_{\geq k} F_\Sigma$, where

$$P = (Q' - Q_F) \cup \{1, \ldots, k\}, \quad P_F = \{k\},$$

where without loss of generality $Q' \cap \{1, \ldots, k\} = \emptyset$. The transitions of h are defined as follows. For $\sigma \in \Sigma_0$, we define $\sigma_h = \sigma_g$ if $\sigma_g \notin Q_F$ and $\sigma_h = 1$ if $\sigma_g \in Q_F$.

For $\sigma \in \Sigma_r, r \geq 1$, and $p_i \in P,\ i = 1, \ldots, r$, we define $\sigma_h(p_1, \ldots, p_r)$ to be

$$\sigma_h(p_1, \ldots, p_r) = \begin{cases} \sigma_g(p_1, \ldots, p_r) & \text{if } p_1, \ldots, p_r \in Q', \text{ and } \sigma_g(p_1, \ldots, p_r) \notin Q_F, \\ 1 & \text{if } p_1, \ldots, p_r \in Q' \text{ and } \sigma_g(p_1, \ldots, p_r) \in Q_F, \\ \min\{k, \sum_{i=1}^r x_i\} & \text{if } \{p_1, \ldots, p_r\} \cap \{1, \ldots, k\} \neq \emptyset \\ & \text{where } x_i = \begin{cases} p_i \text{ if } p_i \in \{1, \ldots, k\}, \\ 0 \text{ if } p_i \in Q' - Q_F. \end{cases} \end{cases}$$

Note that $\sigma_h(p_1, \ldots, p_r)$, for $p_i \in P$, $i = 1, \ldots, r$, is never in Q_F (as required by the choice of states of B). The computation of the DTA B directly simulates the computation of the DTA A until a final state of A is reached. Once a final state of A is reached, the state of B becomes 1, and the integers from 1 to k are used to count the number of independent subtrees of $L(A)$ that have been encountered. When the computation of B reaches a node labeled by σ in states $p_1, \ldots, p_r$ some of which are in $\{1, \ldots, k\}$ and others are states of A, according to the definition of $\sigma_h(p_1, \ldots, p_r)$ only the elements of $\{1, \ldots, k\}$ are summed up and the states of $Q' - Q_F$ have no effect in the remainder of the computation. However, for this purpose we needed to add the dead state d to the original DTA A because in such a situation the computation should not become blocked at an earlier undefined transition. Notice also that if the input tree has more than k independent subtrees of $L(A)$, then the DTA B just remembers the number as k since, by the definition of the operation $\cdot^p_{\geq k}$ there is no need to count beyond the number k. The DTA B accepts the tree language $L(A) \cdot^p_{\geq k} F_\Sigma$ by arriving at the final state k.

Since $|P| = |Q'| - |Q_F| + k$ and (except when $L(A) = \emptyset$), $|Q_F| \geq 1$, an upper bound for the number of states of B is $m + k$. □

We give a lower bound construction that matches the upper bound of Lemma 4.2. We choose a ranked alphabet $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$, where $\Sigma_0 = \{c\}, \Sigma_1 = \{a\}$, and $\Sigma_2 = \{b\}$. Let $M_A = (\Sigma, Q_A, Q_{A,F}, g_A)$ be a DTA, where $Q_A = \{0, 1, \ldots, m-1\}, Q_{A,F} = \{m-1\}$, and the transition

function g_A is defined by setting

$$c_{g_A} = 0, \quad a_{g_A}(i) = i + 1,\ 0 \leq i \leq m - 2, \text{ and } a_{g_A}(m-1) = 0.$$

The DTA M_A recognizes a unary tree language $L(M_A) = \{\ a^s(c) \mid s \equiv -1 \pmod m\ \}$.

Let $M_B = (\Sigma, Q_B, Q_{B,F}, g_B)$ be the DTA for the tree language $L(M_A) \cdot^p_{\geq k} F_\Sigma$ constructed as in the proof of Lemma 4.2. In the construction of M_B, we denote the dead state added to M_A as m. The state set of M_B is then chosen to be $\{1, \ldots, m, m+1, \ldots, m+k\}$, where the states used for counting the subtree occurrences of $L(M_A)$ are the integers from $m+1$ to $m+k$. In the following lemma, we show that the DTA M_B is minimal.

Lemma 4.3. All states of M_B are reachable and pairwise inequivalent.

Proof:
First we show that all the states i where $0 \leq i \leq m, i \neq m-1$ are reachable. The DTA M_B assigns the state 0 to a leaf symbol. Then, the state i, where $0 \leq i \leq m-2$, is reachable from 0 by reading a sequence of unary symbols a^i. Since the computation of the binary symbol b is not defined in g_A, the state m is reachable by reading a binary symbol b with two states that are both 0.

We show that all the states i for $m+1 \leq i \leq m+k$ are reachable using induction on i. The state $m+1$ is reachable from $m-2$ by reading a unary symbol a. Let us inductively assume that all the states $m+1, m+2, \ldots i$, where $i < m+k$ are reachable. Then the state $i+1$ is reachable by reading a binary symbol b with states i and $m+1$. Thus, all states of M_B are reachable.

Now we show that any two states of M_B are pairwise inequivalent. Without loss of generality, let i and j be any distinct states of M_B such that $i > j$.

(a) Case $i > m$: Consider a tree $t = b(x, 2m + k - i)$ with one leaf labeled by symbol x and consider the computations of M_B on $t(x \leftarrow i)$ and $t(x \leftarrow j)$. The former computation reaches the final state $m+k$ of M_B whereas the latter does not. Therefore, any two states i and j are inequivalent if $i > m$.

(b) Case $i \leq m$: By reading a sequence of unary symbols a^{m-1-i}, the state i reaches the state $m+1$ while j reaches $m-1-i+j$. Now $m+1 > m-1-i+j$ and, by the previous step, the states $m+1$ and $m-1-i+j$ are inequivalent. □

Combining Lemmas 4.1, 4.2 and 4.3 we have:

Theorem 4.4. Let A be a DTA with m states. The tree language $L(A) \cdot^p_{\geq k} F_\Sigma = L(A) \cdot^p_k F_\Sigma$ is recognized by a DTA with $m+k$ states, and this bound can be reached in the worst-case.

Acknowledgement. We thank the anonymous referees for a careful reading of the paper and many pertinent and useful suggestions.

References

[1] Maslov A. Estimates on the number of states of finite automata. Soviet Mathematics Doklady, 1970;11:1373–1375.

[2] Yu S, Zhuang Q, Salomaa K. The state complexities of some basic operations on regular languages. Theoretical Computer Science, 1994;125(2):315–328. doi:10.1016/0304-3975(92)00011-F.

[3] Salomaa A, Salomaa K, Yu S. State complexity of combined operations. Theoretical Computer Science, 2007;383(2-3):140–152. URL https://doi.org/10.1016/j.tcs.2007.04.015.

[4] Holzer M, Kutrib M. Descriptional and computational complexity of finite automata. Information and Computation, 2011;209(3):456–470. URL https://doi.org/10.1016/j.ic.2010.11.013.

[5] Gao Y, Moreira N, Reis R, Yu S. A survey on operational state complexity. URL https://arxiv.org/abs/1509.03254.

[6] Cristau J, Löding C, Thomas W. Deterministic automata on unranked trees. In: Proceedings of the 15th International Symposium on Fundamentals of Computation Theory, pp. 68–79. 2005. doi:10.1007/11537311_7.

[7] Han YS, Ko SK, Piao X, Salomaa K. State complexity of Kleene-star operations on regular tree languages. Acta Cybernetica, 2015;22(2):403–422. URL http://dblp.uni-trier.de/db/journals/actaC/actaC22.html#HanKPS15.

[8] Piao X, Salomaa K. Transformations between different models of unranked bottom-up tree automata. Fundamenta Informaticae, 2011;109(4):405–424. dooi:10.3233/FI-2011-519.

[9] Han YS, Salomaa K. Nondeterministic state complexity of nested word automata. Theoretical Computer Science, 2009;410(30-32):2961–2971. URL https://doi.org/10.1016/j.tcs.2009.01.004.

[10] Okhotin A, Salomaa K. State complexity of operations on input-driven pushdown automata. Journal of Computer and System Sciences, 2017;86:207–228. URL Statecomplexityofoperationsoninput-drivenpushdownautomata..

[11] Piao X, Salomaa K. State complexity of the concatenation of regular tree languages. Theoretical Computer Science, 2012;429:273–281. URL https://doi.org/10.1016/j.tcs.2011.12.048.

[12] Ko SK, Eom HS, Han YS. Operational state complexity of subtree free regular tree languages. International Journal of Foundations of Computer Science, 2016;27(6):705–724. URL https://doi.org/10.1142/S0129054116500246.

[13] Ko SK, Lee HR, Han YS. State complexity of regular tree languages for tree matching. International Journal of Foundations of Computer Science, 2016;27(8):965–980. URL https://doi.org/10.1142/S0129054116500398.

[14] Comon H, Dauchet M, Jacquemard F, Lugiez D, Tison S, Tommasi M. Tree Automata Techniques and Applications. 2007. Electronic book available at http://www.tata.gforge.inria.fr.

[15] Gécseg F, Steinby M. Tree languages. In: Rozenberg G, Salomaa A (eds.), Handbook of Formal Languages, Vol. 3: Beyond Words, pp. 1–68. Springer-Verlag New York, Inc., 1997. ISBN:3-540-60649-1.

[16] Brüggemann-Klein A, Wood D. Caterpillars: A context specification technique. Markup Languages, 2000;2(1):81–106. doi:10.1162/109966200750410613.

[17] Neven F. Automata theory for XML researchers. ACM SIGMOD Record, 2002;31(3):39–46. doi:10.1145/601858.601869.

[18] Schwentick T. Automata for XML—A survey. Journal of Computer and System Sciences, 2007; 73(3):289–315. URL https://doi.org/10.1016/j.jcss.2006.10.003.

[19] Gécseg F, Steinby M. Tree Automata. Akadémiai Kiadó, 1984. URL arxiv.org/abs/1509.06233.