State complexity of the concatenation of regular tree languages∗

Xiaoxue Piao, Kai Salomaa∗

School of Computing, Queen’s University, Kingston, Ontario, K7L 3N6, Canada

Abstract

We consider the state complexity of basic concatenation operations for regular tree languages. We show that the sequential (respectively, parallel) concatenation of tree languages recognized by deterministic bottom-up automata with \(m\) and \(n\) states can be recognized by an automaton with \((n + 1) \cdot (m \cdot 2^n + 2^{n-1}) - 1\) (respectively, \(m \cdot 2^n + 2^{n-1} - 1\)) states, and establish matching state complexity lower bounds. The bound for sequential concatenation of tree languages differs by an order of magnitude from the corresponding bound for regular string languages.

© 2011 Elsevier B.V. All rights reserved.

1. Introduction

The descriptional complexity of finite automata and regular languages has been extensively studied in recent years, and tight bounds for the state complexity of basic operations and many combined operations have been established; see, e.g., [4,7,8,10,11,23]. The descriptional complexity of extensions of finite automata, such as tree automata [12,19], or input-driven pushdown automata and nested word automata [1,16,17], has also been considered.

While the state complexity of Boolean operations is similar in the tree case as for ordinary finite automata operating on strings, the situation becomes more involved when considering concatenation operations in which, roughly, we substitute a leaf node of some tree by another tree. It is possible to extend the concatenation operation from strings to trees as either a sequential or a parallel concatenation operation. In the sequential (respectively, parallel) concatenation of trees \(t_1\) and \(t_2\), one leaf (respectively, all leaves) of \(t_2\) having a designated label is (are) replaced by \(t_1\). The operations are extended in the natural way for sets of trees. Parallel concatenation is the same as the concatenation defined in [3] and the operation is called in [6] the \((z-)\) product of tree languages.

In order to keep the connection with string operations more transparent, we define the substitution operation by replacing a leaf (or leaves) of \(t_2\) by \(t_1\). In the context of trees, one could define more general substitutions where a node (or nodes) of \(t_2\) with a given label are replaced by \(t_1\); however, this would not change the worst-case state complexity bounds for the sequential or parallel substitutions, respectively.

We consider the state complexity of concatenation operations for regular tree languages, that is, the question how many states are sufficient, and necessary in the worst case, to recognize the concatenation of tree languages recognized by deterministic bottom-up tree automata with \(m\) and \(n\) states, respectively. We give tight state complexity bounds both for sequential and parallel concatenation. Interestingly, the state complexity of sequential concatenation of tree languages turns out to be of a different order of magnitude than the corresponding bound for regular string languages. The results for parallel concatenation are more similar to those for the string case. It should be noted that, in order to keep our state complexity bounds consistent with the corresponding bounds for the number of vertical states in unranked tree automata,1

1. This will be discussed in more detail in Section 2.
we consider incomplete deterministic tree automata, that is, automata for which some transitions need not be defined. The results for concatenation of string languages known in the literature are stated in terms of complete deterministic finite automata (DFAs) [8,13,27], and the bounds are slightly different for incomplete DFAs.

Much of the recent work on tree automata uses automata operating on unranked trees that are used in modern applications such as XML document processing [3,14,15,24]. An early reference on unranked tree automata is [2]. The transitions of an unranked tree automaton are defined in terms of horizontal languages that are specified by a DFA [3]. Thus, in addition to a set of vertical states used in the bottom-up computation, the size of an unranked tree automaton depends also on the sizes of the DFAs used to specify the horizontal languages [12,18,19,21].

Here, for the sake of conciseness, we restrict consideration to automata operating on ranked trees. The main justification is that, while the notations needed for automata operating on ranked trees are much simpler, by using tree languages over a ranked alphabet we can construct worst-case examples that match the general upper bound for the number of vertical states for the sequential concatenation of unranked tree languages [18]. This bound is of a different order of magnitude than the known state complexity of concatenation of regular string languages.

The general upper bound construction for concatenation of unranked tree languages is given in [18]. While the idea is similar to the one used in Lemma 1 below, the notations are considerably more complicated with unranked tree automata. On the other hand, establishing lower bounds for the sizes of horizontal DFAs in unranked tree automata is a challenging question [18,19,21], and a topic for further research.

2. Preliminaries

We assume the reader to be familiar with finite tree automata and only briefly recall and introduce some definitions needed here. More information on tree automata can be found in [3,6]. An excellent general reference on automata theory is the handbook by Rozenberg and Salomaa [22].

The set of positive integers is \( \mathbb{N} \). The cardinality of a finite set \( S \) is \( |S| \), and the power set of \( S \) is \( 2^S \). When there is no danger of confusion, a singleton set \( \{s\} \) is denoted simply by \( s \). For a cartesian product \( S = S_1 \times \cdots \times S_n \), the \( i \)th projection, \( 1 \leq i \leq n \), is the mapping \( \pi_i : S \rightarrow S_i \) defined by setting \( \pi_i(s_1, \ldots, s_n) = s_i, s_j \in S_j, j = 1, \ldots, n \).

A ranked alphabet is a pair \( (\Sigma, r) \), where \( \Sigma \) is a finite set and \( r : \Sigma \rightarrow \mathbb{N} \cup \{0\} \) is a function that associates with each element \( \sigma \in \Sigma \) its rank \( r(\sigma) \). The set of elements of rank \( m \) is \( \Sigma_m, m \geq 0 \). Usually, instead of \( (\Sigma, r) \), we speak of the ranked alphabet \( \Sigma \) and assume that \( r \) is known. The set of trees over \( \Sigma \), or \( \Sigma \)-trees, \( F_\Sigma \), is the smallest set \( S \) satisfying the following 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 \( F_\Sigma \) consists of \( \Sigma \)-labeled trees, where a node labeled by \( \sigma \in \Sigma_m, m \geq 0 \), always has \( m \) children.

We assume that notions such as the root, a leaf, a subtree and the height of a tree are known. We use the convention that the height of a single node tree is \( 0 \). By the height of a node \( u \) of a tree \( t \) we mean the height of the subtree rooted at \( u \). For \( \sigma \in \Sigma \) and \( t \in F_\Sigma \), \( \text{leaf}(t, \sigma) \) denotes the set of leaves of \( t \) with label \( \sigma \). Let \( t \) be a tree and \( u \) some node of \( t \). The tree obtained from \( t \) by replacing the subtree at node \( u \) with a tree \( s \) is denoted \( t(u \leftarrow s) \). The notation is extended in the natural way for a set of pairwise independent nodes \( U \) of \( t \) and \( S \subseteq F_\Sigma \): \( t(U \leftarrow S) \) is the set of trees obtained from \( t \) by replacing the subtree at each node of \( U \) by some tree in \( S \).

The set of \( \Sigma \)-trees where exactly one label is denoted by a special symbol \( x (x \notin \Sigma) \) is \( F_\Sigma[x] \). For \( t \in F_\Sigma[x] \) and \( t' \in F_\Sigma \), \( t(x \leftarrow t') \) denotes the tree obtained from \( t \) by replacing the unique occurrence of variable \( x \) by \( t' \).

A nondeterministic bottom-up tree automaton (NTA) is a four-tuple \( A = (\Sigma, Q, Q_e, \sigma) \), where \( \Sigma \) is a ranked alphabet, \( Q \) is a finite set of states, \( Q_e \subseteq Q \) is a set of accepting states, and \( \sigma \) associates to each \( \sigma \in \Sigma_m \) a mapping \( \sigma_g : \Sigma^m \rightarrow 2^Q, m \geq 0 \). For each \( t = \sigma(t_1, \ldots, t_m) \in F_\Sigma \), we define inductively the set \( t_Q \subseteq Q \) by setting \( q \in t_Q \) if and only if there exist \( q_i \in (t_i)_Q, i = 1, \ldots, m \), such that \( q \in \sigma_g(q_1, \ldots, q_m) \). Intuitively, \( t_Q \) consists of the states of \( Q \) that \( A \) may reach at the root of \( t \). The tree language accepted by \( A \) is \( L(A) = \{ t \in F_\Sigma \mid t_Q \mid \emptyset \} \). The intermediate stages of a computation, 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 \( \Sigma^t \)-trees where \( \Sigma^t_0 = \Sigma_Q \cup \{Q\} \) and \( \Sigma^t_m = \Sigma_m \) when \( m \geq 1 \). The set of configurations is denoted as \( F_\Sigma[Q] \).

The automaton \( A \) is deterministic (a DTA) if, for each \( \sigma \in \Sigma_m (m \geq 0) \), \( \sigma_g \), is a partial function \( \sigma_g : \Sigma^m \rightarrow Q \). The nondeterministic (bottom-up or top-down) and deterministic bottom-up tree automata accept the family of regular tree languages [3,6].

We allow a deterministic automaton to have undefined transitions, that is, \( \sigma_g, \sigma \in \Sigma_m \), may be undefined for some \( m \)-tuples of states. Note that, while adding a dead state to an incomplete ranked tree automaton (or ordinary DFA) changes the number of states only by \( 1 \), for deterministic tree automata operating on unranked trees [3,18,19], the sizes of an incomplete deterministic automaton and the corresponding completed version may be significantly different. Adding a dead state for the bottom-up computation requires adding, corresponding to an input symbol \( \sigma \), a horizontal language \( L \) that is the complement of a finite disjoint union \( L(A_1) \cup \cdots \cup L(A_m) \), where \( A_i, i = 1, \ldots, m \), are the DFAs recognizing the horizontal languages corresponding to symbol \( \sigma \) and the states of the incomplete automaton. The size of the minimal DFA for \( L \) may be considerably larger than the sum of the sizes of \( A_i, i = 1, \ldots, m \). When considering the state complexity of tree automata operating on trees, it is convenient to allow the use of incomplete automata, and in order to keep
our state complexity bounds consistent with the bounds for unranked tree languages (e.g., those in [18,19]), we also use incomplete DTAs here.

2.1. Concatenation of tree languages

Concatenation of strings can be extended to trees as a sequential operation, where one occurrence of a leaf with a given label is replaced by a tree, or as a parallel operation, where all occurrences of a leaf with a given label are replaced.

For $\sigma \in \Sigma_0$ and $T_1 \subseteq F_\Sigma$, $T_2 \subseteq F_\Sigma$, we define their sequential $\sigma$-concatenation as follows:

$$ T_1 \cdot_\sigma T_2 = \{ t_2(u \leftarrow t_1) \mid u \in \text{leaf}(t_2, \sigma), t_1 \in T_1 \}. $$

That is, $T_1 \cdot_\sigma T_2$ is the set of trees obtained from $t_2$ by replacing one occurrence of a leaf labeled by $\sigma$ with some tree of $T_1$. In order to get concatenation of individual trees we can choose $T_1$ as a singleton set.\footnote{The first argument in (1) and (2) is a set of trees, because otherwise the extension of parallel concatenation to tree languages would be somewhat cumbersome.}

The parallel $\sigma$-concatenation of $T_1$ and $T_2$ is

$$ T_1 \circ_\sigma T_2 = t_2(\text{leaf}(t_2, \sigma) \leftarrow T_1). $$

Note that, when $T_1 = \{t_1\}$ consists of one tree, $T_1 \circ_\sigma T_2$ is an individual tree while $T_1 \cdot_\sigma T_2$ is a set of trees. In the case when no leaf of $T_2$ is labeled by $\sigma$, $T_1 \circ_\sigma T_2 = \emptyset$ and $T_1 \cdot_\sigma T_2 = T_2$.

In the natural way we extend $\circ \in \{ \cdot, \circ \}$ for tree languages $T_1, T_2 \subseteq F_\Sigma$ by setting

$$ T_1 \circ T_2 = \bigcup_{t_2 \in T_2} T_1 \circ t_2. $$

The parallel concatenation $T_1 \circ_\sigma T_2$ is called the $\sigma$-product of $T_1$ and $T_2$ in [6]. When considering bottom-up tree automata operating on unary trees, the above definition of $T_1 \circ T_2$ reduces to the usual concatenation of string languages: the automaton reads first an element of $T_1$ and then an element of $T_2$.

The parallel concatenation operation is associative: however, sequential concatenation is non-associative. For example, with $\Sigma_2 = \{\tau\}$, $\Sigma_0 = \{\sigma\}$ and $t = \tau(\sigma, \sigma)$, we have $\tau(\tau(\sigma, \sigma), \tau(\sigma, \sigma)) \not\in T_1^{\circ_\tau} (T_1^{\circ_\tau}, t)$; however, all trees in $(t^{\circ_\sigma})^{\circ_\tau} t$ have height 3.

We can define powers of a tree language based on sequential or parallel concatenation in the natural way, and then define a Kleene-star operation by setting $T_1^{\star_\sigma}$ to be the infinite union of all the $i$th powers $(i \geq 0)$ of $T$. With a Kleene-star operation based on parallel concatenation, it is easy to see that, with $\Sigma$ as above, the set $(\tau(\sigma, \sigma))^{\star_\tau}$ consists of all balanced trees over $\Sigma$ and, thus, a (usual) Kleene-star operation based on parallel concatenation would not preserve recognizability. In fact, the iteration of the parallel concatenation of tree languages is defined slightly differently in [3,6]. On the other hand, since sequential concatenation is non-associative, there will be two different ways to define an iterated version of sequential concatenation [20].

Finally, we note that instead of (sequential or parallel) concatenation where we replace occurrences of $\sigma \in \Sigma_0$, we could define a more general substitution operation where a subtree with root labeled by $\sigma \in \Sigma_m$, $m \geq 0$, can be replaced by another tree. In the parallel case, the selected nodes labeled by $\sigma \in \Sigma_m$ should be independent, and there is more than one way to define the parallel operation. The state complexity of such generalized operations is the same as the state complexity of tree concatenation considered here; see Remark 2.

3. State complexity of sequential concatenation

Note that, for DTAs $A_1$ and $A_2$, the difficulty in constructing a DTA $B$ for the (sequential) concatenation of $A_1$ and $A_2$ is caused by the fact that $B$ has no way to “know” where a substitution may have occurred, and consequently $B$ has to simulate multiple computations in its state. It turns out that, for sequential concatenation of tree languages, the size blow-up of $B$ differs by an order of magnitude from the known state complexity of concatenation of regular string languages [26].

First, we give an upper bound for the state complexity of sequential concatenation.

**Lemma 1.** Let $A_i$ be a DTA with $m_i$ states, $i = 1, 2$. For $\sigma \in \Sigma_0$, the tree language $L(A_1)^{\cdot_\sigma} L(A_2)$ can be recognized by a DTA with

$$(m_2 + 1) \cdot (m_1 \cdot 2^{m_2} + 2^{m_2-1}) - 1$$

states.
**Proof.** Denote \( A_i = (\Sigma, Q_i, Q_i, f_i, g_i) \), and let \( Q'_i = Q_i \cup \{ \text{dead} \} \), \( i = 1, 2 \). The symbol "dead" will be used to denote a simulated computation that is undefined. Without loss of generality, we assume that \( \sigma_{g_2} \) is defined. Note that, otherwise, trees of \( L(A_2) \) cannot contain leaves labeled by \( \sigma \) and \( L(A_1) \).

We define \( B = (\Sigma, Q_B, Q_B, f_B, g_B) \), where
\[
Q_B = Q'_2 \times 2^Q \times Q'_1,
\]
and the transitions of \( g_B \) are determined below. For \( \tau \in \Sigma_0, m \geq 0, q_1, \ldots, q_m \in Q_i, i = 1, 2 \), we denote
\[
\tau_{g_B}(q_1, \ldots, q_m) = \begin{cases} \tau_{g_1}(q_1, \ldots, q_m) & \text{if } \tau_{g_1}(q_1, \ldots, q_m) \text{ is defined}, \\ \text{dead} & \text{otherwise}. \end{cases}
\]

For \( \tau \in \Sigma_0, \) define
\[
\tau_{g_B} = \begin{cases} (\tau_{g_1}, \sigma_{g_1}, \tau_{g_1}) & \text{if } \tau_{g_1} \in Q_{1,\tau}, \\ (\tau_{g_2}, \emptyset, \tau_{g_1}) & \text{if } \tau_{g_2} \text{ or } \tau_{g_1} \text{ is defined, } \tau_{g_1} \notin Q_{1,\tau}, \\ \text{undefined} & \text{if } \tau_{g_2} \text{ and } \tau_{g_1} \text{ are both undefined}. \end{cases}
\]

For \( \tau \in \Sigma_m, m \geq 1, (p_1, p_2, q_1) \in Q_B, p_i \in Q'_i, p_i \subseteq Q_2, q_i \in Q'_1, i = 1, \ldots, m \), define
\[
\tau_{g_B}(p_1, p_1, q_1, \ldots, (p_m, p_m, q_m))
\]
to be equal to
\[
\text{(i) } (\tau_{g_1}(p_1, \ldots, p_m), X, \tau_{g_1}(q_1, \ldots, q_m)) \text{ if } \tau_{g_1}(q_1, \ldots, q_m) \in Q_{1,\tau}, \text{ where}
\]
\[
X = \bigcup_{i=1}^{m} \left( \bigcup_{x \in P_i} \tau_{g_2}(p_1, \ldots, p_{i-1}, x, p_{i+1}, \ldots, p_m) \right) \cup \{\sigma_{g_1}\}.
\]

\[
\text{(ii) } (\tau_{g_2}(p_1, \ldots, p_m), Y, \tau_{g_1}(q_1, \ldots, q_m)) \text{ if } \tau_{g_1}(q_1, \ldots, q_m) \notin Q_{1,\tau} \text{ and } [\tau_{g_2}(p_1, \ldots, p_m) \text{ or } \tau_{g_1}(q_1, \ldots, q_m) \text{ is defined, or } Y \neq \emptyset]. \text{ Here,}
\]
\[
Y = \bigcup_{i=1}^{m} \left( \bigcup_{x \in P_i} \tau_{g_2}(p_1, \ldots, p_{i-1}, x, p_{i+1}, \ldots, p_m) \right).
\]

\[
\text{(iii) undefined, otherwise.}
\]

The computation of \( B \) is as follows. The first component of the state simulates the computation of \( A_2 \), assuming that no \( \sigma \)-substitution has occurred below the current node. Similarly the third component of the state simulates the computation of \( A_1 \) on the current subtree.

Finally, the second component of the state of \( B \) consists of the set of states \( S \subseteq Q_2 \) that \( A_2 \) could be in, assuming that a \( \sigma \)-substitution has been done below the current node; that is, \( S \) consists of all states that \( A_2 \) would reach if exactly one subtree of the current node belonging to \( L(A_1) \) is replaced by a leaf labeled by \( \sigma \). Both rules (4) and (5) add the state \( \sigma_{g_2} \) to the second component exactly when the current subtree is in \( L(A_1) \). Rules (5)(i) and (ii) simulate all computations in which for exactly one \( 1 \leq i \leq m \) we take a state of \( A_2 \) corresponding to a computation in which a \( \sigma \)-substitution was done below the current node and for all \( j \neq i \) we take the state of \( A_2 \) that corresponds to a computation in which no substitution has occurred.

Thus, by induction on the height of an input tree \( t = \tau(t_1, \ldots, t_m) \), it follows that, assuming that \( B \) reaches the root of \( t_i \) in a state \( (p_i, P_i, q_i) \), where \( P_i \) consists of all states that \( A_2 \) can reach assuming that in \( t_i \) exactly one subtree belonging to \( L(A_1) \) would be replaced by a leaf labeled by \( \sigma \) (and \( p_i, q_i \) are as described above), the second component of the state (5) again consists of all states that \( A_2 \) can reach at the root of \( t \) assuming that exactly one subtree of \( t \) belonging to \( L(A_1) \) had been replaced by the symbol \( \sigma \).

A state of \( B \) is final exactly when the second component contains a final state of \( A_2 \). This means that \( B \) accepts exactly the trees that are obtained from some tree of \( L(A_2) \) by replacing exactly one \( \sigma \)-labeled leaf by a leaf of \( L(A_1) \).

We note that \( |Q_B| = (m_2 + 1) \cdot 2^{m_2} \cdot (m_1 + 1) \); however, not all states of \( Q_B \) are reachable. According to the definition of \( g_B \), a state \( (p, P, q) \) where \( q \in Q_{1,\tau} \) and \( \sigma_{g_2} \notin P \) cannot be reached in any computation of \( B \) and, furthermore, the state \( \text{dead}, \emptyset, \text{dead} \) is omitted as the sink state. Thus, \( Q_B \) has (at least) \( (m_2 + 1) \cdot 2^{m_2} + 1 \) unreachable states. Subtracting this number from \( |Q_B| \) gives the upper bound for the size of \( B \) given in the statement of the lemma. \( \square \)

\footnote{We use a new symbol "dead" (instead of \( \emptyset \)) to make a more transparent distinction between components of \( B \) that are, respectively, a state or a set of states of \( A_i, i = 1, 2 \).}
Remark 2. Suppose that, instead of tree concatenation, where the substitutions occur only at leaves, we consider a more general sequential tree substitution $t_1 \circ t_2$ that substitutes in $t_2$ the subtree at some node labeled by $\sigma \in \Sigma_m$, $m \geq 0$, by the tree $t_1$. For $\sigma \in \Sigma_m$ and DTAs $A_1$ and $A_2$, the $\sigma$-substitution of $L(A_1)$ into $L(A_2)$, $L(A_1) \circ_\sigma^m L(A_2)$, could be recognized by a DTA $C$ with the same set of states as the DTA $B$ in the proof of Lemma 1; however, the transitions of $C$ would be defined slightly differently. In (4) and (5)(i), when the third component is a final state of $A_1$, the transition would add to the second component all states that $A_2$ may reach at the root of an arbitrary tree with the root labeled by $\sigma$. From the description of $A_2$, this set can be easily computed.

On the other hand, since tree concatenation (as considered here) is a special case of $\sigma$-substitution, the state complexity lower bound established below applies also for $\sigma$-substitution, for an $m$-ary symbol $\sigma$. Hence the state complexity of $\sigma$-substitution for an $m$-ary symbol $\sigma$, $m \geq 0$, coincides with the state complexity of tree concatenation.

Similarly, the upper bound construction of Theorem 6 could be modified, without changing the set of states, for a parallel substitution operation that replaces subtrees at nodes labeled by an $m$-ary symbol $\sigma$ by another tree. Since $\sigma \in \Sigma_m$, may label nodes that are descendants of each other, there are various ways to define a parallel substitution operation, and we leave the details to the interested reader.

The upper bound of Lemma 1 is of a different order of magnitude than the tight state complexity bound for concatenation of string languages [26], and it remains to be verified that there exists a worst-case example matching the upper bound.

For our lower bound construction, we use tree languages consisting, roughly speaking, of trees where each branch belongs to the worst-case languages $L(A)$ and $L(B)$ for string concatenation [26] and, furthermore, the DFA $A$ (or $B$, respectively) reaches the same state at an arbitrary node $u$ in computations starting from any two leaves below $u$. Although the construction is based on the worst-case string languages, the extension is non-trivial, and additional technical modifications are required in order to establish a lower bound matching the upper bound of Lemma 1.

Let $A$ and $B$ be the DFAs from Figs. 1 and 2, respectively. Note that $A$ and $B$ are modified variants of the automata used for the worst-case construction for concatenation in [26]. In the DFA $B$ we have added a new alphabet symbol $d$ and a self-loop on $d$ for each state of $B$. With the modified alphabet, $A$ is an incomplete DFA in which the $d$-transition is undefined in each state.

We choose the ranked alphabet $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$, where $\Sigma_0 = \{e\}$, $\Sigma_1 = \{a, b, c, d\}$ and $\Sigma_2 = \{a_2, b_2, c_2, d_2\}$. Denote by $h_\Sigma$ the morphism $(\Sigma_1 \cup \Sigma_2)^* \to \{a, b, c, d\}^*$ defined by the conditions $h_\Sigma(z_2) = h_\Sigma(z) = z$, $z \in \{a, b, c, d\}$; that is, the morphism $h_\Sigma$ simply erases the subscript from elements of $\Sigma_2$.

Using the DFAs $A$ and $B$ of Figs. 1 and 2, we define the tree languages $T_A$, $T_B \subseteq F_\Sigma$. First, $T_B$ is defined to consist of all $\Sigma$-trees $t$ such that we have the following.

(i) The following holds for any node $u$ of $t$ and any nodes $v_1$ and $v_2$ of height 1, located below $u$. If $w_i$ is the string of symbols labeling the path from $v_i$ to $u$, $i = 1, 2$, then $B$ reaches the same state after reading the strings $h_\Sigma(w_1)$ and $h_\Sigma(w_2)$. Furthermore, if $u = e$, $B$ accepts the strings $h_\Sigma(w_1)$ and $h_\Sigma(w_2)$.

(ii) Suppose that node $u$ of $t$ is labeled by $a_2, b_2, c_2, d_2$, and that $u$ has a child that is a leaf (labeled by $e$) and another child $u'$ that is not a leaf, and $w$ is the string of symbols labeling a path from a node of height 1 below $u'$ to $u'$. Then $B$ reaches the state 0 after reading the input $h_\Sigma(w)$.

Intuitively, condition (i) means that, for a tree in $T_B$, when the DFA $B$ reads strings of symbols (with subscripts omitted) labeling paths starting from nodes of height 1 upwards, the computations corresponding to different paths “agree” at each node, and the computations accept at the root. The technical condition (ii) is just used to simplify the definition of the DTA $M_B$ below.
Note that the simulated computations of $B$ on path of the tree are started from the nodes of height 1 and they ignore the leaf symbols. This is done for technical reasons, because in tree concatenation a leaf symbol is replaced by a tree, i.e., the original symbol labeling the leaf will not appear in the resulting tree.

The tree language $T_B$ is recognized by a DTA $M_B = (\Sigma, Q_B, Q_{B,F}, \delta_B)$ where $Q_B = \{0, 1, \ldots, n-1\}$ and the transition function is defined by setting

- $e_{g_B} = 0$,
- $\delta_{g_B}(i) = (a_2)_{g_B}(i, i) = i, 0 \leq i \leq n - 1$,
- $b_{g_B}(i) = (b_2)_{g_B}(i, i) = i + 1, 0 \leq i \leq n - 2$, and $b_{g_B}(n - 1) = (b_2)_{g_B}(n - 1, n - 1) = 0$.
- $c_{g_B}(i) = (c_2)_{g_B}(i, j) = 1, 0 \leq i, j \leq n - 1$,
- $d_{g_B}(i) = (d_2)_{g_B}(i, i) = 1, 0 \leq i \leq n - 1$.

Note that the transitions of $g_B$ on the binary symbol $c_2$ allow different states as the arguments, while the transitions on $a_2, b_2,$ and $d_2$ can be used only for a pair of identical states. The reason is that the transition function of $B$ on each of the symbols $a, b, d, c$ is injective, while the transition function on $e$ is not. It is clear that $M_B$ recognizes the tree language $T_B$ defined previously. Note that condition (ii) implies that, for $t \in T_B$, if a node $u$ of $t$ is labeled by $a_2, b_2,$ or $d_2$, and $u$ has a child $u \cdot i, 1 \leq i \leq 2$, that is a leaf, the computation of $B$ started from a node of height 1 below $u \cdot j, j \neq i$, arrives at node $u \cdot i$ in state 0, which is the state assigned by $M_B$ to the leaf node $u \cdot i$. Thus, the transitions of $M_B$ for a symbol of rank 2 (labeling $u$) continue to correctly simulate the computation of $B$ on one path of the tree.

The tree language $T_A$ and a DTA $M_A$, with $m$ states, recognizing $T_A$ are defined completely analogously based on the DFA $A$ from Fig. 1. Note that, since all $d$-transitions of $A$ are undefined, trees of $T_A$ have no nodes labeled by $d$ or $d_2$ and all $d$- and $d_2$-transitions of $M_A$ are undefined.

In the following, we establish that the DTA constructed from $M_A$ and $M_B$ to recognize the sequential concatenation of $T_A$ and $T_B$ is minimal, and thus gives a worst-case example that matches the upper bound of Lemma 1. Let $M_C = (\Sigma, Q_C, Q_{C,F}, \delta_C)$ be the DTA for the tree language $T_A \times T_B$ constructed as in the proof of Lemma 1. We make the convention that the “dead” state added to $M_A$ (respectively, to $M_B$) is denoted by $m$ (respectively, $n$). That is, the set of states $Q_C$ consists of all triples

$$(p, s, q). \quad 0 \leq p \leq n, \quad S \subseteq \{0, 1, \ldots, n - 1\}, \quad 0 \leq q \leq m,$$

where $q = m - 1$ then $0 \in S$, and if $S = \emptyset$ then $p \neq n$ or $q \neq m$. The number of states of $M_C$ is $(n + 1)(m + 1)2^n - 2^{n-1} - 1$.

In the following two lemmas, we establish that all states of $M_C$ are reachable and pairwise inequivalent with respect to the Myhill–Nerode equivalence relation[25,26] (extended to trees). We still introduce the following notation. For a unary tree where the leaf is an element of $\Sigma_0$ or a state of $Q_C$, $t = z_1(z_2(\ldots z_m(x)\ldots)) \in F_2[Q_C]$, we define word$(t) = z_mz_{m-1}\ldots z_1$.

Note that word$(t)$ consists of the sequence of symbols labeling the nodes of $t$ bottom-up, and the label of the leaf is not included. In the following, when we refer to word$(t)$ of a tree $t$, without further mention, this implies that $t$ is a unary tree (with the leaf possibly labeled by a state of $Q_C$).

**Lemma 3. All states of $M_C$ are reachable.**

**Proof.** Using induction on $|S|$, we establish that all the states (6) are reachable. The DTA $M_C$ assigns to a leaf symbol $e$ the state $(0, \emptyset, 0)$. When $|S| = 0, (i, \emptyset, j), 0 \leq i \leq n - 1, 0 \leq j \leq m - 2$ is reachable from $(0, \emptyset, 0)$ by reading a unary tree $t$ where word$(t) = b^j d$. (Note that $(i, m, m) = 0$ is not a state of $Q_C$.) The state $(n, \emptyset, j), 1 \leq j \leq m - 2,$ is reachable by reading tree $a_2(t_1, t_2)$, where word$(t_1) = b^j d^{-1}$ and word$(t_2) = b^j d$ and the leaves of $t_1$ and $t_2$ are labeled by $(0, \emptyset, 0)$. The state $(n, 0, 0)$ is reached by reading a unary symbol $b$ from state $(n, \emptyset, j), 1 \leq j \leq m - 2.$ The state $(i, \emptyset, m), 0 \leq i \leq n - 1$ is reached from state $(i, \emptyset, 0)$ by reading a unary symbol $d$.

In the following, for an integer $x \geq -n$, denote

$$\bar{x} = \begin{cases} x & \text{if } x \geq 0 \\ n + x & \text{if } x < 0. \end{cases}$$

Consider $z \geq 0$, and inductively assume that, for $|S| \leq z$, all the states $(i, S, j)$ as in (6), $0 \leq i \leq n, 0 \leq j \leq m, S \subseteq \{0, \ldots, n - 1\}$ are reachable. We will show that any state $(x, S', y), 0 \leq x \leq n, 0 \leq y \leq m, |S'| = z + 1$, is reachable.

First, consider the case when $y \neq m - 1$. Let $s_1 > s_2 > \cdots > s_z$ be the elements in $S'$. Let $P = \{s_1 - s_2 + 1, s_2 - s_2 + 1, \ldots, s_z - s_{z+1}\}$.

(i) When $0 \leq x \leq n - 1$, according to the inductive assumption, the state $(\bar{x} - s_2 + 1, P, 0)$ is reachable. Then the state $(\bar{x} - s_2 + 1, P \cup \{0\}, m - 1)$ is reachable from $(\bar{x} - s_2 + 1, P, 0)$ by reading a sequence of unary symbols $a^{m-1}$. The state $(x, S', y), 0 \leq y \leq m - 2$ is reachable from $(\bar{x} - s_2 + 1, P \cup \{0\}, m - 1)$ by reading a sequence of unary symbols $b^{z+1}d$.

The state $(x, S', m)$ is reachable from $(\bar{x} - s_2 + 1, P \cup \{0\}, m - 1)$ by reading a sequence of unary symbols $b^{z+1}d$.

As at the end of the proof of Lemma 1, we have omitted from $Q_C$, the unreachable states.
(ii) When \( x = n \), according to the inductive assumption, the state \((n, P, 0)\) is reachable. Then, the state \((n, P \cup \{0\}, m−1)\) is reachable from \((n, P, 0)\) by reading a sequence of unary symbols \(a^m\). The state \((n, S', y)\), \(0 ≤ y ≤ m−2\) is reachable from \((n, P \cup \{0\}, m−1)\) by reading a sequence of unary symbols \(b^{n−1}d\). The state \((n, S', m)\) is reachable from \((n, P \cup \{0\}, m−1)\) by reading a sequence of unary symbols \(b^{n−1}d\).

Finally, consider the case when \( y = m−1 \). According to the definition of (6), \( 0 ∈ S' \). By the inductive assumption, the state \((x, S' \cup \{0\}, m−2)\) is reachable. Then, the state \((x, S', m−1)\) is reachable by reading the unary symbol \(a\).

This concludes the proof of the inductive step and the proof of the lemma. \(\Box\)

It remains to be established that the DTA \(M_C\) has no two equivalent states.

**Lemma 4.** All states of \(M_C\) are pairwise inequivalent.

**Proof.** Let \((i_1, S_1, j_1)\) and \((i_2, S_2, j_2)\) be any distinct states as in (6). First, we consider the case when \(S_1 \neq S_2\) or \(j_1 \neq j_2\). We get from the DTA \(M_C\) a bottom-up tree automaton recognizing the restriction of \(T_A \cup T_B\) to unary trees simply by ignoring the first component of the states, and making all transitions undefined on elements of \(Σ_2\). Note that, for unary trees \(t_1 ∈ T_A\), \(t_2 ∈ T_B\), \(\text{word}(t_1 \cdot t_2)\) is simply the string concatenation of \(\text{word}(t_1)\) and \(\text{word}(t_2)\).

Let \(B'\) be the DFA obtained from \(B\) (in Fig. 2) by deleting all \(d\)-transitions. In [26,27], it is established that the minimal DFA for the concatenation of the string languages \(L(A)\) and \(L(B')\) needs \(m^2 − 2m−1\) states, which means that the elements \((S, i), S \subseteq \{0, \ldots, n−1\}, 0 ≤ i ≤ m−1\) where \(0 ∈ S\) always when \(i = m−1\), correspond to states of a minimal DFA for \(L(A)\cup L(B')\), and also to states of a minimal DTA for \(T_A \cup T_B\) restricted to unary trees without occurrences of the symbol \(d\). Note that, in the construction of \(M_C\), the \(U\) transitions operate on the second and third components in the same way as in the DFA constructed in [26,27] to recognize \(L(A)\cup L(B')\).

This means that, when \((S_1, j_1) \neq (S_2, j_2)\) and \(0 ≤ j_1, j_2 ≤ m−1\), the states \((i_1, S_1, j_1)\) and \((i_2, S_2, j_2)\) can be distinguished using a unary tree. Note that \(j = m−1\) \((1 ≤ i ≤ 2)\) corresponds to a dead state of \(M_C\), and this dead state does not occur in the construction of [26,27], and we need to consider the cases \(j = m, 1 ≤ i ≤ 2\), separately.

First, consider the case \(j_1 = m, 0 ≤ j_2 ≤ m−1\) (and \(S_1\) and \(S_2\) may or may not be equal). Choose a sequence of unary symbols \(ca^{m−j_1−1}b^{n−1}\). From state \((i_2, S_2, j_2)\), state \((1, \{1\}, j_2)\) is reached after reading \(c\), state \((1, \{1\}, 0)\) is reached after reading \(a^{m−j_2−1}\), and a final state \((0, \{0, n−1\}, 0)\) is reached after reading \(b^{n−1}\). On the other hand, from state \((i_1, S_1, m)\), state \((1, \{1\}, m)\) is reached after reading \(c\), state \((1, \{1\}, m)\) is reached after reading \(a^{m−j_2−1}\), and state \((0, \{0\}, m)\) is reached after reading \(b^{n−1}\). The latter is not a final state.

Next, consider the case when \(S_1 \neq S_2\) and \(j_1 = j_2 = m\). Without loss of generality, choose \(S \in S_1 \setminus S_2\) (the other possibility being symmetric). Choose a sequence of unary symbols \(w = b^{n−1}a\). After reading \(w\), \(M_C\) reaches a final state when the computation begins from state \((i_1, S_1, m)\), while the computation beginning with \((i_2, S_2, m)\) does not reach a final state.

So far, we have shown that any two states \((i_1, S_1, j_1)\) and \((i_2, S_2, j_2)\) can be distinguished when \(S_1 \neq S_2\) or \(j_1 \neq j_2\), \(0 ≤ j_1, j_2 ≤ m\). It remains to show the case when \(S_1 = S_2 = S, j_1 = j_2 = j\) and \(i_1 \neq i_2\). Since \(i_1 \neq i_2\), one of \(i_1, i_2\) has to be distinct from \(n\), and, without loss of generality, we assume that \(0 ≤ i_1 ≤ n−1, 0 ≤ i_2 ≤ n\). In order to establish that \((i_1, S, j)\) (and \((i_2, S, j)\) are inequivalent), it is sufficient to give a tree \(t \in F_{Σ'}[x]\) such that the computation of \(M_C\) on \(t(x \leftarrow (i_1, S, j))\) (respectively, \(t(x \leftarrow (i_2, S, j))\)) visits a final state (respectively, a non-final state). Here, \(Σ'_0 = Σ_0 \cup Q_c\) and \(Σ'_k = Σ_k\), when \(k ≥ 1\). Above, we use the fact that by **Lemma 3** all states of \(Q_c\) are reachable.

Denote \(q_u = (i_1 + 1, \{i_1\}, j)\) and as the tree \(t \in F_{Σ'}[x]\) we choose \(t = b^{n−2−i_1}b_{2x}(x, q_u)\).

First, consider the computation of \(M_C\) on \(t(x \leftarrow (i_1, S, j))\). Since the second component of \(q_u\) contains \(i_1\) and the first components of \(q_u\) and \((i_1, S, j)\) are different, the computation assigns \((n, \{i_1\}, 0)\) to the root of \(b_2((i_1, S, j), q_u)\). After reading the remaining \(b_2\)'s on the unary path, the final state \((n, \{n−1\}, 0)\) is reached.

Now, consider the computation on \(t(x \leftarrow (i_2, S, j))\). Denote by \((y, U, z)\) the state assigned to the root of \(b_2((i_2, S, j), q_u)\). We note that \(z = 0\) and

\[
y = \begin{cases} 
(i_1 + 2) \mod n & \text{if } i_2 = i_1 + 1 \text{ and } i_1 + 1 \neq n, \\
0 & \text{otherwise.}
\end{cases}
\]

\[
U = \begin{cases} 
\{(i_1 + 2) \mod n\} & \text{if } i_1 + 1 \in S, \\
\emptyset & \text{otherwise.}
\end{cases}
\]

Above, we use \(x \mod n\) as an element of \(\{0, 1, \ldots, n−1\}\). Recall that the number of \(b\)'s after the root of \(b_2((i_2, S, j), q_u)\) (the binary symbol \(b_2\) is not counted) to the root of \(t(x \leftarrow (i_2, S, j))\) is \(2n−i−2\).

(i) If \(y = (i_1 + 2) \mod n\) and \(U = \{k\}, k = (i_1 + 2) \mod n\), the computation beginning from state \((y, U, z)\) after reading the sequence of unary symbols \(b^{2n−i−2}\) reaches the state \((0, \{0\}, 0)\).

(ii) If \(y = n\), the first component of the resulting state will change to \(n\), and if \(U = \emptyset\), the second component of the resulting state will change to \(\emptyset\).

In all cases, the computation of \(M_C\) reaches a non-final state at the root of \(t(x \leftarrow (i_2, S, j))\).

This concludes the proof showing that all the states of (6) are pairwise inequivalent. \(\Box\)
The following is now a consequence of Lemmas 1, 3 and 4.

Theorem 5. Suppose that $A_i$ is a DTA with $m_i$ states, $i = 1, 2$, and $\sigma \in \Sigma_0$. The sequential $\sigma$-concatenation $L(A_1) \cdot_\sigma L(A_2)$ can be recognized by a DTA with

$$ (m_2 + 1) \cdot (m_1 \cdot 2^{m_2} + 2^{m_2 - 1}) - 1 $$

states.

For any integers $m_1, m_2 \geq 2$, there exist DTAs $A_i$ with $m_i$ states, $1 \leq i \leq 2$, such that the minimal DTA for $L(A_1) \cdot_\sigma L(A_2)$ has $(7)$ states.

4. State complexity of parallel concatenation

In this section, we give a tight state complexity bound for the parallel concatenation of tree languages. As can perhaps be expected, the bounds are similar to those for regular string languages. We give a short construction for the upper bound because we are considering incomplete automata, and the bounds differ slightly for complete and incomplete DTAs, respectively. The well-known state complexity bounds for concatenation of string languages are stated in terms of complete DTAs [9,13,27]. The transition complexity of incomplete DTAs has been considered in [5].

Theorem 6. Let $A_1$ and $A_2$ be DTAs with $m$ and $n$ states, respectively $(m, n \geq 2)$. For $\sigma \in \Sigma_0$, the tree language $L(A_1) \cdot_\sigma L(A_2)$ is recognized by a DTA with $m \cdot 2^n + 2^{n - 1} - 1$ states, and this bound can be reached in the worst case.

Proof. Denote $A_i = (i, Q_i, Q_{i,F}, g_i), i = 1, 2,$ and let $Q_i' = Q_i \cup \{\text{dead}\}$. Without loss of generality, $\sigma_{g_2}$ is defined (because otherwise $L(A_1) \cdot_\sigma L(A_2) = L(A_2)$). We define $D = (\Sigma, Q_{D,F}, g_{D,F}),$ where $Q_{D,F} = 2^{Q_2} \times Q_1', Q_{D,F} = \{q \in Q_D | \tau_1(q) \cap Q_{2,F} \neq \emptyset\}$, and the transitions of $g_D$ are determined below. For $\tau \in \Sigma_0,$ define

$$ g_D((\tau_{g_1}, q_1), \ldots, (\tau_{g_k}, q_k)) = (\tau_{g_2}, P_{g_2}) \cup X, $$

where $X = \{q_{g_2} \text{ if } \tau_{g_1}(q_1, \ldots, q_k) \in Q_{1,F} \text{ and } X = \emptyset \text{ otherwise.}$$

Above, the overline notation is as in (3). When $\tau_{g_2}$ is undefined, $\{\tau_{g_2}, \sigma_{g_2}\} = \{\sigma_{g_2}\}$.

For $\tau \in \Sigma_k, k \geq 1$, and $(P_i, q_i) \in Q_{D,F}, i = 1, \ldots, k$, define

$$ g_D((\tau_{g_1}, q_1), \ldots, (\tau_{g_k}, q_k)) = (\tau_{g_2}, (P_1, \ldots, P_k) \cup X, $$

where $X = \{q_{g_2} \text{ if } \tau_{g_1}(q_1, \ldots, q_k) \in Q_{1,F} \text{ and } X = \emptyset \text{ otherwise.}$$

Denote by $B'$ the DTA obtained from $B$ by omitting all transitions on $d$. Denote by $A'$ the DTA used in [26,27] to establish the tight lower bound for concatenation of complete DTAs$^6$ implies that all states of $D$ belonging to

$$ Z = \{(P, i) | P \subseteq \{0, \ldots, n - 1\}, 0 \leq i \leq m - 1, \text{ where } i = m - 1 \text{ implies } 0 \in P\} $$

are reachable and pairwise inequivalent. Furthermore, each state of the form $(P, d)$, $\emptyset \neq P \subseteq \{0, \ldots, n - 1\}$ is reachable in $D$ from state $(P, 0)$ by reading the symbol $d$. (Recall that $d$-transitions are undefined in $A$.)

In order to complete the proof, it is sufficient to show that states of the form $(P, d)$ are all pairwise inequivalent, and no state of this form can be equivalent with a state of $Z$.

First, consider two states $(P_1, d)$ and $(P_2, d)$, where $i \in P_1 - P_2$. Now, after reading $b^{n-i}$ from state $(P_1, d)$ (respectively, from state $(P_2, d)$), $D$ reaches a final (respectively, non-final) state.

Second, consider states $(P_1, d)$ and $(P_2, i)$, $0 \leq i \leq m - 1$ (where $P_1$ and $P_2$ are not required to be distinct). Choose $w = c^{m-1-i} b^{i-1}$. After reading $c$ in state $(P_1, d)$, $D$ goes to state $((1), \text{dead})$ and $a$ is the identity on states of $B$. After this, $n - 1$ symbols $b$ give the non-final state $(0, \text{dead})$. On the other hand, the symbol $c$ yields from state $(P_2, i)$ the state $((1), i)$, and reading the sequence $a^{m-1-i}$ yields then $((0), m - 1)$. After this, the sequence $b^{i-1}$ yields the accepting state $((0, n - 1), 0)$. Thus, the computation of $D$ on input $w$ reaches a non-final and a final state from states $(P_1, d)$ and $(P_2, i)$, respectively. □
5. Conclusion

We have established tight state complexity bounds for both sequential and parallel tree concatenation. For ease of presentation, our constructions are based on the lower bound construction of [26] that uses an alphabet of size 3. The alphabet size could be reduced by basing the constructions on the lower bound example of Jiráskova [9] over a binary alphabet. However, even in the simpler case of parallel concatenation, our construction requires the addition of a new symbol, and we do not know whether the lower bound of Theorem 6 holds for incomplete DFAs over a binary alphabet. The question of minimal alphabet size is more involved for sequential concatenation because there the lower bound construction needs to use a non-unary ranked alphabet.

Finally, note that, in the natural way, based on the concatenation operations, we can define the \( i \)-th powers, \( i \geq 0 \), of a tree language \( T \), and then we can define the Kleene-star of \( T \) as the infinite union of all \( i \)-th powers of \( T \), \( i \geq 0 \). With this definition, the Kleene-star based on parallel concatenation would not preserve regularity, and the iterated parallel concatenation is defined slightly differently in [3,6]. Since sequential concatenation is non-associative, there are two different ways to define the Kleene-star based on sequential concatenation, depending on how we define the powers of a tree language, and we call these the top-down star and the bottom-up star, respectively [20]. The top-down iterated sequential concatenation coincides with the iterated parallel concatenation as defined in [3,6]. In future work, we will consider the state complexity of iterated concatenation of tree languages [20].

References