State Complexity of Regular Tree Languages for Tree Matching

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

Ha-Rim Lee† and Yo-Sub Han‡
Department of Computer Science, Yonsei University
50 Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea
†hrlee@yonsei.ac.kr
‡emmous@yonsei.ac.kr

Received 6 November 2014
Accepted 3 May 2016
Communicated by Arto K. Salomaa

We study the state complexity of regular tree languages for tree matching problem. Given a tree $t$ and a set of pattern trees $L$, we can decide whether or not there exists a subtree occurrence of trees in $L$ from the tree $t$ by considering the new language $L'$ which accepts all trees containing trees in $L$ as subtrees. We consider the case when we are given a set of pattern trees as a regular tree language and investigate the state complexity. Based on the sequential and parallel tree concatenation, we define three types of tree languages for deciding the existence of different types of subtree occurrences. We also study the deterministic top-down state complexity of path-closed languages for the same problem.

Keywords: State complexity; tree matching; regular tree languages; path-closed languages.

1. Introduction

State complexity is one of the most interesting topics in automata and formal language theory [10, 12, 26, 27]. The state complexity of finite automata has been studied since the 60’s [13, 16, 17]. Maslov [15] initiated the problem of finding the operational state complexity and Yu et al. [27] investigated the state complexity for basic operations. Later, Yu and his co-authors [4, 7, 24, 25] initiated the study on the state complexity of combined operations such as star-of-union, star-of-intersection

†Corresponding author.
and so on. Moreover, Yu and his co-authors studied the state complexity of combined Boolean operations including multiple unions and multiple intersections [4–6].

Recently, the state complexity problem has been extended to regular tree languages. Regular tree languages and tree automata theory provide a formal framework for XML schema languages such as XML DTD, XML Schema, and Relax NG [18]. XML schema languages can process a set of XML documents by specifying the structural properties. Martens and Niehren [14] considered the problem of efficiently minimizing unranked tree automata. Piao and Salomaa [20,21] considered the state complexity between different models of unranked tree automata. They also investigated the state complexity of concatenation [23] and star [22] for regular tree languages. Two of the authors studied the state complexity of subtree-free regular tree languages, which are a proper subclass of regular tree languages [11].

Since a regular tree language is a set of trees, it is suitable for representing a set of structural documents such as XML documents, web documents, or RNA secondary structures. This implies that a regular tree language can be used as a theoretical toolbox for processing of the structured documents. When it comes to the string case, many researchers often use regular languages to process a set of strings efficiently. Consider the case that we have a set of strings which is a regular language $L$. Now we want to find any occurrence of strings in $L$ from a text $T$. The most common way is to construct an FA $A$ that accepts a regular language $\Sigma^*L$ [3]. Then, we read $T$ using $A$ and check whether or not $A$ reaches a final state. When $A$ reaches a final state, we find that there is an occurrence of a matching string of $L$ in $T$. We extend this approach to the tree matching problem [9]. First, we formally define the tree matching problem to be the problem of finding subtree occurrences of a tree in $L$ from a set of trees $T$. Since a tree can be processed in a bottom-up or a top-down fashion, we need to consider different types of tree languages for the tree matching problem.

Here we consider three types of tree substructures called a subtree, a topmost subtree and an internal subtree. Given a tree language $L$, we construct three types of tree languages recognizing trees which contain the trees in $L$ as subtrees, topmost subtrees and internal subtrees. Note that these tree languages can be used for the tree matching problem as we have used $\Sigma^*L$ for the string pattern matching problem. In particular, we tackle the deterministic state complexity of regular tree languages and path-closed languages. Interestingly, the tree language consisting of trees that have a subtree belonging to a path-closed language language need not be path-closed and therefore cannot recognized by deterministic top-down tree automata (DTTAs).

We give basic notations and definitions in Sec. 2. We define the three types of tree languages for tree matching in Sec. 3. We present the results on the state complexity of regular tree languages and path-closed languages in Secs. 4 and 5, and conclude the paper in Sec. 6.
2. Preliminaries

We briefly recall definitions and properties of finite tree automata and regular tree languages. We refer the reader to the books [2,8] for more details on tree automata.

A ranked alphabet $\Sigma$ is a finite set of characters and we denote the set of elements of rank $m$ by $\Sigma_m \subseteq \Sigma$ for $m \geq 0$. The set $F_\Sigma$ consists of $\Sigma$-labeled trees, where a node labeled by $\sigma \in \Sigma_m$ always has $m$ children. We use $F_\Sigma$ to denote a set of trees over $\Sigma$ that 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 $\sigma(t_1, \ldots, t_m) \in S$. Let $t(u \leftarrow s)$ be the tree obtained from a tree $t$ by replacing the subtree at a node $u$ of $t$ with a tree $s$. The notation is extended for a set $U$ of nodes 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 $S$.

A nondeterministic bottom-up tree automaton (NBTA) is specified by a tuple $A = (\Sigma, Q, Q_f, g)$, where $\Sigma$ is a ranked alphabet, $Q$ is a finite set of states, $Q_f \subseteq Q$ is a set of final states and $g$ associates each $\sigma \in \Sigma_m$ to a mapping $\sigma_g : Q^m \to 2^Q$, where $m \geq 0$. Assume $A$ has a transition $\sigma_g(q_1, \ldots, q_m) = P$. In this case, $A$ moves to the set $P$ of states by reading a sequence $q_1, \ldots, q_m$ of states and a character $\sigma$ of rank $m$. We say that each element of $P$ is a target state of the sequence $q_1, \ldots, q_m$ of states. For each tree $t = \sigma(t_1, \ldots, t_m) \in F_\Sigma$, we define inductively the set $q_g \subseteq Q$ by setting $q \in t_q$ if and only 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, $t_q$ consists of the states of $Q$ that $A$ may reach by reading $t$. Thus, the tree language accepted by $A$ is defined as follows: $L(A) = \{ t \in F_\Sigma \mid t_q \cap Q_f \neq \emptyset \}$. The automaton $A$ is a deterministic bottom-up tree automaton (DBTA) if, for each $\sigma \in \Sigma_m$, where $m \geq 0$, $\sigma_g$ is a partial function $Q^m \to 2^Q$.

A nondeterministic top-down tree automaton (NTTA) is specified by a tuple $A = (\Sigma, Q, Q_0, g)$, where $\Sigma$ is a ranked alphabet, $Q$ is a finite set of states, $Q_0 \subseteq Q$ is a set of initial states, and $g$ associates each $\sigma \in \Sigma_m$, $m \geq 0$, a mapping $\sigma_g : Q \to 2Q^m$. As a convention, we denote the $m$-tuples $q_1, \ldots, q_m$ by $[q_1, \ldots, q_m]$. A top-down tree automaton $A$ is deterministic if $Q_0$ is a singleton set and for all $q \in Q$, $\sigma \in \Sigma_m$, and $m \geq 1$, $\sigma_g$ is a partial function $Q \to Q^m$.

The nondeterministic (bottom-up or top-down) and deterministic bottom-up tree automata accept the family of regular tree languages whereas the deterministic top-down tree automata accept a proper subfamily of regular tree languages — path-closed languages [2,8].

3. Tree Languages for Tree Pattern Matching

Pattern matching is the problem of finding occurrences of a pattern in a text. The regular expression matching problem is defined as follows: given a pattern regular expression $E$ and an input text $T$, we want to identify all substrings of $T$ that are in $L(E)$ [3]. Similar to the regular expression matching problem, we consider a pattern given as a set of trees with a tree automaton (TA) $A$ for the tree pattern matching problem. Here we construct a new TA $A'$ from $A$ as we put $\Sigma^*$ to the pattern.
language $L(A)$ to make the new FA $A'$ to simulate all the possible prefixes before simulating the matching substrings of $L(A)$ [1]. This means that, given $A$, we need to construct a new TA $A'$ that simulates all the possible prefixes before simulating the matching subtrees of $L(A)$. Since a tree can be processed in a bottom-up way with a bottom-up TA or a top-down way with a top-down TA, we need to consider three types of tree languages for the tree pattern matching problem.

First we define three different tree substructures. If a tree $t'$ consists of a node in a tree $t$ and all of its descendants, we call $t'$ a subtree of $t$. If a tree $t'$ is a subtree of $t$, then we call $t$ a supertree of $t'$. We also define the topmost subtree of a tree $t$ as a tree consisting of a set of nodes in $t$ including the root node such that from any node in the set, there exists a path to the root node through the nodes in the set. An internal subtree of a tree $t$ can be defined as a topmost subtree of a subtree of $t$. We give graphical examples for the definitions in Fig. 1.

![Graphical examples](image)

Fig. 1. We define three types of subtrees called a subtree, a topmost subtree and an internal subtree. These figures depict the examples.

Given a tree $t$ and a regular tree language $L$, we first compute a new regular tree language $L'$ that accepts all possible supertrees of trees in $L$. Then, we decide whether or not a tree in $L$ occurs as a subtree of the given tree $t$ by deciding $t \in L'$. Recall that we build a new FA that accepts $\Sigma^*L$, which is a concatenation of a universal language $\Sigma^*$ and a given language $L$, for matching a language $L$ of string patterns. For tree pattern matching problem, we need to consider how to define the concatenation of trees properly. Recently, Piao and Salomaa [23] studied the state complexity of the concatenation of regular tree languages. They defined the sequential $\sigma$-concatenation and parallel $\sigma$-concatenation where the substitutions can occur at $\sigma$-labeled leaves.

We consider a more generalized operation that allows substitution to occur at all leaves regardless of labels. We denote the set of leaves of a tree $t$ by $\text{leaf}(t)$. Then, for $T_1 \subseteq F_{\Sigma}$ and $t_2 \in F_{\Sigma}$, we define the sequential concatenation of $T_1$ and $t_2$ to be

$$T_1 \cdot \Sigma t_2 = \{t_2(u \leftarrow t_1) \mid u \in \text{leaf}(t_2), t_1 \in T_1\}.$$ 

In other words, $T_1 \cdot \Sigma t_2$ is a set of trees obtained from $t_2$ by replacing a leaf with a tree in $T_1$. We extend the sequential concatenation operation to the tree
languages \( T_1, T_2 \subseteq F_\Sigma \) as follows:
\[
T_1 \cdot^* T_2 = \bigcup_{t_2 \in T_2} T_1 \cdot^* t_2.
\]
The parallel concatenation of \( T_1 \) and \( t_2 \) is
\[
T_1 \cdot^p t_2 = \{ t_2(\text{leaf}(t_2) \leftarrow t_1) \mid t_1 \in T_1 \}.
\]
Thus, \( T_1 \cdot^p t_2 \) is a set of trees obtained from \( t_2 \) by replacing all leaves with a tree in \( T_1 \). We can also extend the parallel concatenation to tree languages. Note that we can say that a tree \( t_2 \) is a topmost subtree of \( t_1 \) if \( t_1 \in F_\Sigma \cdot^p t_2 \).

Relying on the sequential and parallel tree concatenations, we construct three types of tree languages from a regular tree language \( L \) for the tree pattern matching problem. See Fig. 2. Given a tree language \( L \),

1. \( L \cdot^* F_\Sigma \) is a set of trees where a tree in \( L \) occurs as a subtree of each tree in the set,
2. \( F_\Sigma \cdot^p L \) is a set of trees where a tree in \( L \) occurs as a topmost subtree of each tree in the set, and
3. \( F_\Sigma \cdot^p L \cdot^* F_\Sigma \) is a set of trees where a tree in \( L \) occurs as an internal subtree of each tree in the set.

Notice that a leaf node of a tree can be replaced with any other nodes for the topmost subtree occurrence and the internal subtree occurrence.

4. State Complexity of DBTAs
First we study the state complexity of \( F_\Sigma \cdot^p L \) which can be used for finding subtree occurrences of a tree in \( L \).

**Lemma 1.** Given a DBTA \( A = (\Sigma, Q, Q_F, g) \) with \( n \) states for a regular tree language \( L \), \( 2^{n-k} \) states are sufficient for recognizing \( F_\Sigma \cdot^p L \) if \( |\{\sigma_g \mid \sigma \in \Sigma_0\}| = k \).
Proof. Without loss of generality, we assume \( Q_F \cap \{ \sigma_g \mid \sigma \in \Sigma_0 \} = \emptyset \) because otherwise \( F_{\Sigma^*}^p L(A) = F_{\Sigma^*} \). We present an upper bound construction of a DBTA \( B \) for \( F_{\Sigma^*}^p L(A) \). We define \( B = (\Sigma, Q', Q_F', g') \), where

\[
Q' = \{ X \cup \{ \sigma_g \mid \sigma \in \Sigma_0 \} \mid X \in 2^{Q_{\Sigma^*}} \}, \quad Q_F' = \{ q \in Q' \mid q \cap Q_F \neq \emptyset \},
\]

and the transitions of \( g' \) are defined as follows:

For \( \tau \in \Sigma_0 \), \( \tau g' = \{ \sigma_g \mid \sigma \in \Sigma_0 \} \). For \( \tau \in \Sigma_m, m \geq 1 \), and \( P_1, P_2, \ldots, P_m \in Q' \),

\[
\tau g'(P_1, P_2, \ldots, P_m) = \tau g(P_1, P_2, \ldots, P_m) \cup \{ \sigma_g \mid \sigma \in \Sigma_0 \}.
\]

Now we explain how \( B \) recognizes the tree language \( F_{\Sigma^*}^p L \). Note that we define every target state of \( g' \) to be the union of the set of states reachable by \( g \) and the set of states reachable by reading leaf nodes. Since every target state of \( g' \) is not empty, a new DBTA \( B \) is complete although \( A \) may not be complete. Note that \( \{ \sigma_g \mid \sigma \in \Sigma_0 \} \) is a set of states that are reachable by reading a leaf node. This implies that all states of \( B \) contain the states in \( \{ \sigma_g \mid \sigma \in \Sigma_0 \} \). After reading any tree in \( F_{\Sigma^*} \), the state of \( B \) contains \( \{ \sigma_g \mid \sigma \in \Sigma_0 \} \) by the construction. Therefore, \( B \) can start a simulation of a tree in \( L(A) \) after reading any trees in \( F_{\Sigma^*} \) by regarding the trees as leaf nodes. Since \( F_{\Sigma^*}^p L(A) \) is a set of trees where all the leaf nodes of each tree can be substituted by any trees in \( F_{\Sigma^*} \), \( B \) accepts \( F_{\Sigma^*}^p L(A) \).

The upper bound in Lemma 1 is reachable when a DBTA accepts a set of unary trees. If a DBTA accepts a set of unary trees, then we can regard the DBTA as a DFA with multiple initial states. Since the upper bound reaches the maximum when \( k = 1 \), we consider the state complexity of catenation of \( L \) and \( \Sigma^* \). Let \( L \) be a regular language whose state complexity is \( n \). Then, the state complexity of \( \Sigma^* L \) is \( 2^{n-1} \) \cite{27} which is the same as the bound in Lemma 1. Furthermore, we show that the upper bound is tight for any \( 1 \leq k \leq n \).

Let \( \Sigma = \Sigma_0 \cup \Sigma_1 \), where \( \Sigma_0 = \{ \sigma_1, \sigma_2, \ldots, \sigma_k \} \) and \( \Sigma_1 = \{ a, b \} \). We define a DBTA \( C_1 = (\Sigma, Q_{C_1}, Q_{C_1,F}, g_{C_1}) \), where \( Q_{C_1} = \{ 0, 1, \ldots, n-1 \} \), \( Q_{C_1,F} = \{ n-1 \} \) and the transition function \( g_{C_1} \) is defined by setting:

- \((a)_{g_{C_1}}(i) = i - 1 \ (1 \leq i \leq k)\),
- \(a_{g_{C_1}}(i) = i + 1 \ \text{mod} \ n\),
- \(b_{g_{C_1}}(i) = i \ (0 \leq i < k)\),
- \(b_{g_{C_1}}(i) = i + 1 \ \text{mod} \ n \ (k \leq i < n)\).

Based on the construction of the proof of Lemma 1, we construct a DBTA \( D_1 = (\Sigma, Q_{D_1}, Q_{D_1,F}, g_{D_1}) \) recognizing \( F_{\Sigma^*}^p L(C_1) \), where \( Q_{D_1} = \{ P \mid \{ 0, 1, \ldots, k-1 \} \subseteq P, P \subseteq Q_{C_1} \} \), \( Q_{D_1,F} = \{ P \mid P \in Q_{D_1}, P \cap Q_{C_1,F} \neq \emptyset \} \), and the transition function \( g_{D_1} \) is defined as follows:

- \((a)_{g_{D_1}}(i) = \{ 0, 1, \ldots, k-1 \} \ (0 \leq i \leq k)\),
- \(a_{g_{D_1}}(P) = a_{g_{C_1}}(P) \cup \{ 0, 1, \ldots, k-1 \}\),
- \(b_{g_{D_1}}(P) = b_{g_{C_1}}(P) \cup \{ 0, 1, \ldots, k-1 \}\).
Notice that \( L(D_1) = F_{\Sigma}^{p} L(C_1) \). In the following lemma, we establish that \( D_1 \) is a minimal DBTA by showing that all states of \( D_1 \) are reachable and pairwise inequivalent.

**Lemma 2.** All states of \( D_1 \) are reachable and pairwise inequivalent.

**Proof.** First, we prove the reachability of all states of \( D_1 \). Note that each state of \( D_1 \) is a set of states in \( C_1 \). By the construction, the size of a state \( P \) in \( Q_{D_1} \) satisfies \( k \leq |P| \leq n \) since \( \{0, 1, \ldots, k-1\} \subseteq P \). Using the induction on \( |P| \), we show that all states of \( D_1 \) are reachable.

- **Basis:** We have a state \( \{0, 1, \ldots, k-1\} \) of size \( k \) that is reachable by reading a leaf node.
- **Inductive Hypothesis:** Assuming that all states \( P \) are reachable for \( |P| \leq x \), we will show that any state \( P' \) is reachable when \( |P'| = x + 1 \). Let \( P' = \{0, 1, \ldots, k-1, q_k, q_{k+1}, \ldots, q_x\} \) be a state of size \( x + 1 \). The state \( P' \) is reachable from a state \( \{0, 1, \ldots, k-1, q_k, q_{k+1}, \ldots, q_x\} \) by reading a sequence of unary symbols \( ab^{k-1} \). Therefore, all states are reachable by induction.

Next we prove that all states of \( D_1 \) are pairwise inequivalent. Pick any two distinct states \( P_1 \) and \( P_2 \). Assume \( p \in P_1 \setminus P_2 \). (The other possibility is completely symmetric.) After reading a sequence of unary symbols \( a_{n-p-1} \), a final state is reached from state \( P_1 \) whereas \( P_2 \) reaches a non-final state. Therefore, all states of \( D_1 \) are pairwise inequivalent.

Since we have shown that there exists a corresponding lower bound for the upper bound, the bound is tight.

**Theorem 3.** Given a DBTA \( A \) with \( n \) states for a regular tree language \( L \), \( 2^{n-k} \) states are necessary and sufficient in the worst-case for the minimal DBTA of \( F_{\Sigma}^{p} L \) if \( |\{\sigma_g \mid \sigma \in \Sigma_0\}| = k \).

Now we consider \( L^{s} F_{\Sigma} \) — a tree language consists of all trees that have trees in \( L \) as subtrees. In other words, for any tree \( t \) in \( L \), we have all possible supertrees of \( t \) in \( L' \). Given a regular tree language \( L \), it is known that \( L^{s} F_{\Sigma} \) is also a regular tree language [23]. We study the state complexity of \( L^{s} F_{\Sigma} \).

**Lemma 4.** Given a DBTA \( A = (\Sigma, Q, Q_F, g) \) with \( n \) states for a regular tree language \( L \), \( n + 1 \) states are sufficient for recognizing \( L^{s} F_{\Sigma} \).

**Proof.** We construct a new DBTA \( B = (\Sigma, Q', Q_F', g') \) for \( L^{s} F_{\Sigma} \), where \( Q' = Q \cup \{q_{\text{new}}\} \), \( Q_F' = Q_F \), and the transition function \( g' \) is defined as follows:

For \( \tau \in \Sigma_0 \),

\[
\tau g' = \begin{cases} 
\tau g & \text{if } \tau g \text{ is defined}, \\
q_{\text{new}} & \text{otherwise}.
\end{cases}
\]
For \( \tau \in \Sigma_m, m \geq 1, q_1, q_2, \ldots, q_m \in Q', \) and \( q_f \in Q'_f, \)

\[
\tau'_g(q_1, q_2, \ldots, q_m) = \begin{cases} 
\tau_g(q_1, q_2, \ldots, q_m) & \text{if } \tau_g(q_1, q_2, \ldots, q_m) \text{ is defined and } \\
\{q_1, q_2, \ldots, q_m\} \cap Q_f = \emptyset, & \\
q_f & \text{if } \{q_1, q_2, \ldots, q_m\} \cap Q_f \neq \emptyset, \\
q_{\text{new}} & \text{otherwise}.
\end{cases}
\]

Now we explain how \( B \) accepts a set of all trees that are supertrees of trees in \( L. \)
We define the transition function \( g' \) to be complete by setting the target state of the
undefined transition as a new state \( g_{\text{new}}. \) Then, \( B \) moves to \( q_{\text{new}} \) by reading
trees in the complement of \( L \) and moves to one of its final states by reading trees
in \( L. \) Assume that \( B \) accepts a tree in \( L \) and arrives at a final state \( q_f. \) After then,
\( B \) stays in \( q_f \) by reading any sequence of states including \( q_f. \) This implies that \( B \)
accepts all supertrees of trees in \( L(A). \)

We cannot reach the upper bound \( n + 1 \) with any DFA in this case since the
state complexity of \( L \Sigma^* \) is \( n, \) which is the same as that of \( L, \) even for incomplete
DFAs. Thus, we show that there exists a lower bound DBTA of \( n + 1 \) states for
accepting \( L \cdot \Sigma^* F_2, \) where the state complexity of \( L \) is \( n \) to prove the tightness of the
upper bound.

Let \( \Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2, \) where \( \Sigma_0 = \{c\}, \Sigma_1 = \{a\} \) and \( \Sigma_2 = \{b\}. \) We define a
DBTA \( C_2 = (\Sigma, Q_{C_2}, Q_{C_2,F}, g_{C_2}), \) where \( Q_{C_2} = \{0, 1, \ldots, n - 1\}, Q_{C_2,F} = \{n - 1\}, \)
and the transition function \( g_{C_2} \) is defined by setting:

- \( c_{g_{C_2}} = 0, \)
- \( a_{g_{C_2}}(i) = b_{g_{C_2}}(i, i) = i + 1 \mod n. \)

All transitions of \( g_{C_2} \) not listed above are undefined. Based on the construction of the
proof of Lemma 4, we construct a DBTA \( D_2 = (\Sigma, Q_{D_2}, Q_{D_2,F}, g_{D_2}) \) recognizing
\( L(C_2)^\cdot \Sigma^* F_2, \) where \( Q_{D_2} = Q_{C_2} \cup \{n\}, Q_{D_2,F} = Q_{C_2,F} \) and the transition function \( g_{D_2} \) is defined as follows:

- \( c_{g_{D_2}} = 0, \)
- \( a_{g_{D_2}}(i) = b_{g_{D_2}}(i, i) = i + 1 \quad (0 \leq i \leq n - 2), \)
- \( a_{g_{D_2}}(n - 1) = b_{g_{D_2}}(n - 1, i) = b_{g_{D_2}}(i, n - 1) = n - 1 \quad (0 \leq i \leq n - 1), \)
- \( a_{g_{D_2}}(n) = b_{g_{D_2}}(i, j) = n \quad (i \neq j, i \neq n - 1, j \neq n - 1). \)

Notice that \( L(D_2) = L(C_2)^\cdot \Sigma^* F_2. \) In the following lemma, we establish that \( D_2 \)
is a minimal DBTA by showing that all states in \( Q_{D_2} \) are reachable and pairwise
inequivalent.

**Lemma 5.** All states of \( D_2 \) are reachable and pairwise inequivalent.

**Proof.** First, we prove the reachability of all states of \( D_2. \) It is easy to verify that
the state \( i \) \((0 \leq i \leq n - 1)\) is reachable from the state \( c_{g_{C_2}} = 0 \) by reading a sequence
of unary symbols \( a^i. \) Then, the state \( n \) is reachable by reading a binary symbol \( b \)
with two states \( i \) and \( j \) \((0 \leq i, j \leq n - 2, i \neq j)\) since \( b_{g_{D_2}}(i, j) = n \) by construction.
We prove that all states are pairwise inequivalent. We consider two distinct states \(i\) and \(j\) such that \(i < j\). There are two possible cases:

- \(0 \leq i < j < n\): From the state \(j\), we arrive at a final state \(n - 1\) by reading a sequence of unary symbols \(a^{n-1-j}\). However, the state \(i\) arrives at \(n - 1 - j + i\) by reading the same sequence and the state \(n - 1 - j + i\) is not final.
- \(0 \leq i < n\) and \(j = n\): From the state \(i\), we arrive at a final state by reading a sequence of unary symbols \(a^{n-1-i}\) whereas the state \(j\) stays at the state \(n\), which is not final.

We have shown that all states are pairwise inequivalent in all possible cases. \(\Box\)

Based on Lemma 4 and Lemma 5, we establish the following statement.

**Theorem 6.** Given a DBTA \(A\) with \(n\) states for a regular tree language \(L\), \(n + 1\) states are necessary and sufficient in the worst-case for the minimal DBTA of \(L \cdot ^* F_\Sigma\).

We lastly consider the state complexity of \(F_{\Sigma} \cdot ^* L \cdot ^* F_\Sigma\). Note that the sequential catenation of trees is not associative whereas the parallel catenation of trees is associative. That means that there exist trees \(t_1, t_2\) and \(t_3\) such that \((t_1 \cdot ^* t_2) \cdot ^* t_3\) and \(t_1 \cdot ^* (t_2 \cdot ^* t_3)\) do not coincide. This also applies to the catenation of tree languages and thus, leads to \((L_1 \cdot ^* L_2) \cdot ^* L_3 \neq L_1 \cdot ^* (L_2 \cdot ^* L_3)\) for some regular tree languages \(L_1, L_2,\) and \(L_3\). However, for the case when \(L_1\) and \(L_3\) are \(F_\Sigma\),

\[
(F_\Sigma \cdot ^* L_2) \cdot ^* F_\Sigma = F_\Sigma \cdot ^* (L_2 \cdot ^* F_\Sigma)
\]

holds. Thus, we simply denote the language by \(F_{\Sigma} \cdot ^* L \cdot ^* F_\Sigma\) instead of \((F_\Sigma \cdot ^* L_2) \cdot ^* F_\Sigma\) or \(F_\Sigma \cdot ^* (L_2 \cdot ^* F_\Sigma)\). Now we tackle the state complexity of \(F_{\Sigma} \cdot ^* L \cdot ^* F_\Sigma\).

**Lemma 7.** Given a DBTA \(A = (\Sigma, Q, Q_F, g)\) with \(n\) states for a regular tree language \(L\), \(2^{n-k} + 1\) states are sufficient for recognizing \(F_{\Sigma} \cdot ^* L \cdot ^* F_\Sigma\) if \(|Q_F| = t\) and \(|\{\sigma_g \mid \sigma \in \Sigma_0\}| = k\).

**Proof.** Without loss of generality, we assume \(Q_F \cap \{\sigma_g \mid \sigma \in \Sigma_0\} = \emptyset\) because otherwise \(F_{\Sigma} \cdot ^* L(A) \cdot ^* F_\Sigma = F_\Sigma\). We give an upper bound construction of DBTA \(B\) that recognizes \(F_{\Sigma} \cdot ^* L(A) \cdot ^* F_\Sigma\). We define \(B = (\Sigma, Q', Q_F', g')\), where

\[
Q' = \{X \cup \{\sigma_g \mid \sigma \in \Sigma_0\} \mid X \in 2^{Q' \cup \{\sigma_0 \mid \sigma \in \Sigma_0\}}\} \cup \{Q_F\}, \quad Q_F' = \{Q_F\},
\]

and the transitions of \(g'\) are defined as follows:

For \(\tau \in \Sigma_0\), \(\tau_g' = \{\sigma_g \mid \sigma \in \Sigma_0\}\). For \(\tau \in \Sigma_m, m \geq 1,\) and \(P_1, P_2, \ldots, P_m \in Q'\),

\[
\tau_g' (P_1, P_2, \ldots, P_m) = \begin{cases} 
\tau_g (P_1, P_2, \ldots, P_m) & \text{if } \bigcup_{i=1}^{m} P_i \cap Q_F = \emptyset \\
\{\sigma_0 \mid \sigma \in \Sigma_0\} & \text{and } \tau_g (P_1, P_2, \ldots, P_m) \cap Q_F = \emptyset, \\
Q_F & \text{otherwise}.
\end{cases}
\]

Note that the sequential catenation is not associative whereas the parallel catenation of trees is associative. Therefore, \(F_{\Sigma} \cdot ^* L \cdot ^* F_\Sigma\) is not associative.
Now we explain how $B$ recognizes $F_{\Sigma}^{\cdot} L(A)^{\cdot} F_{\Sigma}$. Note that we define every target state of $g'$ to be the union of the set of states reachable by $g$ and the set of states reachable by reading leaf nodes. This implies that $B$ can start simulation of a tree in $L$ after reading any trees in $F_{\Sigma}$. Therefore, we know that $B$ arrives at a final state of $A$ by reading any trees in $F_{\Sigma}^{\cdot} L(A)$. By the construction, $B$ moves to $Q_F$ which is the single final state of $B$ by reading trees in $F_{\Sigma}^{\cdot} L(A)$. After then, $B$ stays in $Q_F$ by reading any sequence of states including $Q_F$. This implies that $B$ accepts all possible supertrees of trees in $F_{\Sigma}^{\cdot} L(A)$. Since a set of all supertrees of trees in $F_{\Sigma}^{\cdot} L(A)$ is $F_{\Sigma}^{\cdot} L(A)^{\cdot} F_{\Sigma}$, $B$ accepts $F_{\Sigma}^{\cdot} L(A)^{\cdot} F_{\Sigma}$. 

Next we present a lower bound example that reaches the upper bound $2^{n-t-k+1}$.

Let $\Sigma = \Sigma_0 \cup \Sigma_1$, where $\Sigma_0 = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$ and $\Sigma_1 = \{a, b, c\}$. We define a DBTA $C_3 = (\Sigma, Q_{C_3}, Q_{C_3,F}, g_{C_3})$, where $Q_{C_3} = \{0, 1, \ldots, n - 1\}, Q_{C_3,F} = \{n - t, n - t + 1, \ldots, n - 1\}$ and the transition function $g_{C_3}$ is defined by setting:

- $(\sigma_1)_{g_{C_3}} = i - 1 \ (1 \leq i \leq k)$,
- $a_{g_{C_3}}(i) = i + 1 \mod n$,
- $b_{g_{C_3}}(i) = i \ (0 \leq i \leq k)$,
- $b_{g_{C_3}}(i) = i + 1 \mod n \ (k \leq i < n)$,
- $c_{g_{C_3}}(i) = i + 1 \mod n \text{ if } i \neq n - t - 1, c_{g_{C_3}}(n - t - 1) = 0$.

Based on the construction in the proof of Lemma 7, we construct a DBTA $D_3 = (\Sigma, Q_{D_3}, Q_{D_3,F}, g_{D_3})$ recognizing $F_{\Sigma}^{\cdot} L(C_3)^{\cdot} F_{\Sigma}$, where $Q_{D_3} = \{P \mid \{0, 1, \ldots, k - 1\} \subseteq P, P \subseteq Q_{C_3} \setminus Q_{C_3,F}\}, Q_{D_3,F} = \{Q_{C_3,F}\}$, and the transition function $g_{D_3}$ is defined as follows:

- $(\sigma_1)_{g_{D_3}} = \{0, 1, \ldots, k - 1\}$,
- $a_{g_{D_3}}(P) = a_{g_{C_3}}(P) \cup \{0, 1, \ldots, k - 1\}$ if $a_{g_{C_3}}(P) \cap Q_{C_3,F} = \emptyset$ and $P \cap Q_{C_3,F} = \emptyset$,
- $a_{g_{D_3}}(P) = \{Q_{C_3,F}\}$ if $a_{g_{C_3}}(P) \cap Q_{C_3,F} \neq \emptyset$,
- $b_{g_{D_3}}(P) = b_{g_{C_3}}(P) \cup \{0, 1, \ldots, k - 1\}$ if $b_{g_{C_3}}(P) \cap Q_{C_3,F} = \emptyset$ and $P \cap Q_{C_3,F} = \emptyset$,
- $b_{g_{D_3}}(P) = \{Q_{C_3,F}\}$ if $b_{g_{C_3}}(P) \cap Q_{C_3,F} \neq \emptyset$,
- $c_{g_{D_3}}(P) = c_{g_{C_3}}(P) \cup \{0, 1, \ldots, k - 1\}$ if $c_{g_{C_3}}(P) \cap Q_{C_3,F} = \emptyset$ and $P \cap Q_{C_3,F} = \emptyset$,
- $c_{g_{D_3}}(\{Q_{C_3,F}\}) = b_{g_{D_3}}(\{Q_{C_3,F}\}) = c_{g_{D_3}}(\{Q_{C_3,F}\}) = \{Q_{C_3,F}\}$.

Notice that $L(D_3) = F_{\Sigma}^{\cdot} L(C_3)^{\cdot} F_{\Sigma}$. In the following lemma, we establish that $D_3$ is a minimal DBTA by showing that all states in $Q_{D_3}$ are reachable and pairwise inequivalent.

**Lemma 8.** All states of $D_3$ are reachable and pairwise inequivalent.

**Proof.** We prove the reachability of all non-final states of $D_3$ using induction on the size of $P$. Note that any non-final state $P \in Q_{D_3}$ satisfies $k \leq |P| \leq m - t$ because $Q_{C_3,F} \cap P = \emptyset$ and $\{\sigma \mid \sigma \in \Sigma_0\} \subseteq P$ by the construction. A state $\{0, 1, \ldots, k - 1\}$ of size $k$ is reachable by reading a leaf node. Assume that all states $P$ is reachable for $|P| \leq x$. Then, we show that any state $P'$ of size $x + 1$ is reachable.
Let \( P' = \{0, 1, \ldots, k-1, q_k, q_k+1, \ldots, q_x\} \) be a state of size \( x + 1 \). Then, the state \( P' \) is reached from a state \( \{0, 1, \ldots, k-1, q_k+1-k, q_k-k, q_k-k+1, \ldots, q_x-q_k+k-1\} \) after reading a sequence of unary symbols \( a^b q_0^k \). From the induction, it is easy to verify that all states except \( Q_{C_3,F} \) are reachable. Furthermore, the only final state \( Q_{C_3,F} \) is reachable from a non-final state \( \{0, 1, \ldots, n-t-1\} \) by reading a unary symbol \( a \).

Next we prove that all states of \( D_3 \) are pairwise inequivalent. Pick any two distinct states \( P_1 \) and \( P_2 \). Assume \( p \in P_1 \setminus P_2 \). (The other possibility is symmetric.) From \( P_1 \), a final state is reached by reading a sequence of unary symbols \( e^{n-t-1} \cdot \cdot \cdot \cdot a \) whereas \( P_2 \) does not reach a final state. Therefore, any two states in \( Q_{D_3} \) are pairwise inequivalent.

**Theorem 9.** Given a DBTA \( A = (\Sigma, Q, Q_F, g) \) with \( n \) states for a regular tree language \( L \), \( 2^{n-t-1} + 1 \) states are necessary and sufficient in the worst-case for the minimal DBTA of \( F_{\Sigma} \cdot L \cdot F_{\Sigma} \) if \( |Q_F| = t \) and \( |\{\sigma \mid \sigma \in \Sigma_0\}| = k \).

5. **State Complexity of DTTAs**

It is well known that every NBTA can be converted into an equivalent NTTA [2, 8]. On the other hand, not all regular tree languages are recognized by DTTAs. In other words, a class of regular tree languages accepted by DTTAs is a proper subclass of regular tree languages accepted by NBTA or NTTA. Note that DTTA recognize exactly the class of path-closed languages that is a proper subclass of regular tree languages [2, 8]. This leads us to study the state complexity of path-closed languages for tree matching — the state complexity of DTTAs.

We again consider three types of tree languages \( F_{\Sigma} \cdot L \cdot F_{\Sigma} \), \( L \cdot F_{\Sigma} \), and \( F_{\Sigma} \cdot L \cdot F_{\Sigma} \), where \( L \) is a regular tree language. However, given a path-closed language \( L \), \( L \cdot F_{\Sigma} \) and \( F_{\Sigma} \cdot L \cdot F_{\Sigma} \) are not necessarily path-closed languages. Nivat and Podelski [19] argued that path-closed languages can be characterized by a property called the subtree exchange property as follows:

**Proposition 10 (Nivat and Podelski [19]).** A regular tree language \( L \) is path-closed if and only if, for every \( t \in L \) and every node \( u \in t \), if \( t(u \leftarrow a(t_1, \ldots, t_m)) \in L \) and \( t(u \leftarrow a(s_1, \ldots, s_m)) \in L \), then \( t(u \leftarrow a(t_1, \ldots, s_i, \ldots, t_m)) \in L \) for each \( i = 1, \ldots, m \).

Using the subtree exchange property, we prove that given a tree language \( L \), \( L \cdot F_{\Sigma} \) and \( F_{\Sigma} \cdot L \cdot F_{\Sigma} \) are not path-closed languages.

**Proposition 11.** There exists a path-closed language \( L \) such that \( L \cdot F_{\Sigma} \) or \( F_{\Sigma} \cdot L \cdot F_{\Sigma} \) is not a path-closed language.

**Proof.** First we show that there exists a path-closed language \( L \) such that \( L \cdot F_{\Sigma} \) is not a path-closed language. Let \( \Sigma = \Sigma_2 \cup \Sigma_0 \), where \( \Sigma_2 = \{b\} \), and \( \Sigma_0 = \{a, c\} \). A singleton language \( L \) contains a single-node tree \( c \), namely \( L = \{c\} \). It is
straightforward to verify that $F_Σ$ contains every binary tree where leaf nodes are labeled by $a$ or $c$, and non-leaf nodes are labeled by $b$.

Then, $L \cdot ^s F_Σ$ is a set of binary trees where every tree contains at least one leaf labeled by $c$. Therefore, $b(a, c) \in L \cdot ^s F_Σ$, $b(c, a) \in L \cdot ^s F_Σ$, and $b(a, a) \notin L \cdot ^s F_Σ$ hold. However, if $L \cdot ^s F_Σ$ is path-closed, $b(a, a)$ should exist in $L \cdot ^s F_Σ$ by the subtree exchange property. This implies that $L \cdot ^s F_Σ$ is not a path-closed language.

Now let us prove that there exists a path-closed language $L$ such that $F_Σ \cdot ^p L \cdot ^s F_Σ$ is not a path-closed language. Let $Σ = Σ_2 ∪ Σ_0$, where $Σ_2 = \{a, b\}$, and $Σ_0 = \{c\}$. A singleton language $L$ contains a tree $a(c, c)$, namely $L = \{a(c, c)\}$. It is easy to verify that $F_Σ$ contains every binary tree where all leaf nodes are labeled by $c$ and non-leaf nodes are labeled by $a$ or $b$.

Then, $F_Σ \cdot ^p L \cdot ^s F_Σ$ is a set of binary trees where every tree contains at least one non-leaf node labeled by $a$. Therefore, $b(a(c, c), c) \in F_Σ \cdot ^p L \cdot ^s F_Σ$, $b(c, a(c, c)) \in F_Σ \cdot ^p L \cdot ^s F_Σ$, and $b(c, c) \notin F_Σ \cdot ^p L \cdot ^s F_Σ$. However, due to the subtree exchange property, $b(c, c)$ should be in $F_Σ \cdot ^p L \cdot ^s F_Σ$ if the language $F_Σ \cdot ^p L \cdot ^s F_Σ$ is path-closed. This means that $F_Σ \cdot ^p L \cdot ^s F_Σ$ is not a path-closed language.

We define the deterministic top-down state complexity of a path-closed language $L$ to be the number of states that are necessary and sufficient in the worst-case for the minimal DTTA recognizing $L$.

**Theorem 12.** Given a DTTA $A = (Σ, Q, Q_0, g)$ with $n$ states for a path-closed language $L$, $n$ states are necessary and sufficient in the worst-case for the minimal DTTA of $F_Σ \cdot ^p L$.

**Proof.** We construct a new DTTA $B = (Σ', Q', Q'_0, g')$ for $F_Σ \cdot ^p L$, where $Q' = Q$, $Q'_0 = Q_0$, and the transition function $g'$ is defined as follows:

For $τ ∈ Σ_m$, $m ≥ 0$ and $q ∈ Q'$, we define

$$τ_g(q) = \begin{cases} τ_q(q) & \text{if } σ_τ(q) \neq \lambda \text{ for any } σ ∈ Σ_0, \\ \underbrace{q, q, \ldots, q}_m & \text{otherwise.} \end{cases}$$

Now we explain how $B$ simulates $F_Σ \cdot ^p L$ with $n$ states. Since trees in $F_Σ \cdot ^p L$ have the same topmost parts with trees in $L$ and leaves can be substituted with any tree in $F_Σ$, $B$ simulates from the same initial state with $A$. Let us assume that a state $q ∈ Q'$ may end the top-down computation with generating a leaf node since $σ_τ(q) = \lambda$. Once $B$ arrives at $q$, the new transition function $g'$ continues the computation by reading a non-leaf label of rank $m$ and generating a sequence $[q, q, \ldots, q]$ of states whose length is $m$. This makes a new DTTA $B$ to generate any subtree in $F_Σ$ at the point where the computation may end with generating leaves and, thus, recognize the language $F_Σ \cdot ^p L$.

It is easy to see that $n$ states are necessary to recognize $F_Σ \cdot ^p L$. Consider a path-closed language of unary trees whose state complexity correspond to that of regular
string languages. Since the state complexity of $L\Sigma^*$ is $n$ if the state complexity of $L$ is $n$, this case can be a lower bound for the path-closed language $F\Sigma\cdot^p L$.

6. Conclusions

We have considered the state complexity of three types of tree languages $F\Sigma\cdot^p L$, $L\cdot^s F\Sigma$, and $F\Sigma\cdot^p L\cdot^s F\Sigma$ for tree pattern matching problem. Motivated from tree pattern matching problem, we have investigate the state complexity of these languages when they are described by DBTAs and DTTAs. Table 1 summarizes the established results. Especially, we have shown that $L\cdot^s F\Sigma$ and $F\Sigma\cdot^p L\cdot^s F\Sigma$ are not recognizable by DTTAs even when $L$ is a path-closed language since they are not necessarily path-closed languages. We have shown that $L\cdot^s F\Sigma$ and $F\Sigma\cdot^p L\cdot^s F\Sigma$ need not be path-closed and therefore cannot recognized by DTTAs.

Table 1. A summary table for the state complexity of DBTAs and DTTAs for the tree languages $F\Sigma\cdot^p L$, $L\cdot^s F\Sigma$, and $F\Sigma\cdot^p L\cdot^s F\Sigma$.

<table>
<thead>
<tr>
<th>languages</th>
<th>state complexity of DBTAs</th>
<th>state complexity of DTTAs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F\Sigma\cdot^p L$</td>
<td>$2^{n-k}$</td>
<td>$n$</td>
</tr>
<tr>
<td>$L\cdot^s F\Sigma$</td>
<td>$n + 1$</td>
<td>not recognizable</td>
</tr>
<tr>
<td>$F\Sigma\cdot^p L\cdot^s F\Sigma$</td>
<td>$2^{n-1-k} + 1$</td>
<td>not recognizable</td>
</tr>
</tbody>
</table>

A possible future direction is to investigate the descriptional complexity of unranked tree automata, which are a more generalized model than tree automata over ranked alphabet, for recognizing $L\cdot^s F\Sigma$ and $F\Sigma\cdot^p L\cdot^s F\Sigma$.

Acknowledgments

Ko was partially supported by EPSRC grant “Reachability problems for words, matrices and maps” (EP/M00077X/1), and Lee and Han were supported by the Basic Science Research Program through NRF funded by MEST (2015R1D1A1A01060097) and the Yonsei University Future-leading Research Initiative of 2015.

References


