State Complexity of Subtree-Free Regular Tree Languages

Hae-Sung Eom, Yo-Sub Han, and Sang-Ki Ko

Department of Computer Science, Yonsei University
50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea
{haesung,emmous,narame7}@cs.yonsei.ac.kr

Abstract. We introduce subtree-free regular tree languages that often appear in XML schemas and investigate the state complexity of basic operations on subtree-free regular tree languages. The state complexity of an operation for regular tree languages is the number of states that are sufficient and necessary in the worst-case for the minimal deterministic ranked tree automaton that accepts the tree language obtained from the operation. We establish the precise state complexity of (sequential, parallel) concatenation, (bottom-up, top-down) star, intersection and union for subtree-free regular tree languages.

Keywords: deterministic ranked tree automata, state complexity, subtree-free regular tree language, basic operations.

1 Introduction

State complexity problem is one of the most interesting topics in automata and formal language theory [2,8,11,12,21,26,27]. These results are mainly on the descriptional complexity of finite automata and regular languages. For example, Maslov [14] obtained the state complexity of catenation and later Yu et al. [27] investigated the state complexity for basic operations. Later, the state complexity of combined operations has been initiated by Yu et al. [5,6,24,25] such as star-of-union, star-of-intersection and so on. Researchers also considered the state complexity of multiple operations such as several catenations [4,5,23] or several intersections [5]. Han et al. [9,10] observed the state complexity of prefix-free and suffix-free regular languages that have unique structural properties in DFAs, which are crucial to obtain the state complexity. It turned out that the state complexities of catenation and Kleene-star are both at most linear for prefix-free regular languages due to the restrictions on the structures of DFAs.

Regular tree languages and tree automata theory provide a formal framework for XML schema languages such as XML DTD, XML Schema, and Relax NG [16]. XML schema languages can process a set of XML documents by specifying the structural properties formally. Recently, Marten and Niehren [13] considered the state complexity of tree automata for the purpose of minimization of XML schemas and unranked tree automata. Piao and Salomaa [17,18] also considered the state complexities between different models of unranked tree automata. They also investigated the state complexities of concatenation [20] and star [19] for regular tree languages.

We consider a proper subfamily of regular tree languages, called *subtree-free regular languages*. A *subtree* of a tree t is a tree consisting of a node in t and all of its descendants in t. We say that a tree t_1 is a supertree of a tree t_2 if t_2 is a subtree of t_1. Subtree-freeness means that a set of trees does not contain a tree that is a subtree of another tree in the set. This property is useful because many regular tree languages become subtree-free when they are used as XML schemas in practice. XML documents should have exactly one unique root element that is also known as the *document element*. The document element is, therefore, the very first element of an XML document and encloses all other elements. Therefore, all XML documents from a specific XML schema have the same root element. When viewed as a set of trees, it satisfies the subtree-freeness. We tackle the state complexities of basic operations for subtree-free regular tree languages.

In Section 2, we define some basic notions. We define the subtree-free regular tree languages in Section 3 and observe the structural properties of the languages. We obtain the state complexity for sequential and parallel concatenation in Section 4, bottom-up and top-down star in Section 5, and the intersection and union in Section 6. We conclude the paper in Section 7.

2 Preliminaries

We briefly recall definitions and properties of finite tree automata and regular tree languages. We refer the reader to the books [3,7] for more details on tree automata.

For a Cartesian product $S = S_1 \times \cdots \times S_n$, the ith projection, where $1 \leq i \leq n$, is the mapping $\pi_i : S \longrightarrow S_i$ defined by setting $\pi_i(s_1, \ldots, s_n) = s_i$. A ranked alphabet Σ 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_Σ consists of Σ-labeled trees, where a node labeled by $\sigma \in \Sigma_m$ always has m children. We use F_Σ to denote a set of trees over Σ 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* (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 each $\sigma \in \Sigma_m$ to a mapping $\sigma_g : Q^m \longrightarrow 2^Q$, where $m \geq 0$. For each 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 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_g consists of the states of Q that A may reach by reading the tree t. Thus, the tree language accepted by A is defined as follows: $L(A) = \{t \in F_\Sigma \mid t_g \cap Q_f \neq \emptyset\}$.

The intermediate states 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 Σ'-trees, where $\Sigma'_0 = \Sigma_0 \cup \{Q\}$ and $\Sigma'_m = \Sigma$ when $m \geq 1$. The set of configurations is denoted as $F_\Sigma[Q]$. 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 $Q^m \longrightarrow Q$.

For tree languages, there are two types of concatenations and two types of Kleene-star: sequential concatenation, parallel concatenation, bottom-up Kleene-star and top-down Kleene-star. We follow the definitions and notations of the operations from the prior work [19,20]. We denote the set of leaves of a tree t labeled σ by $\texttt{leaf}(t, \sigma)$. For $\sigma \in \Sigma_0$, $T_1 \subseteq F_\Sigma$ and $t_2 \in F_\Sigma$, we define the *sequential σ-concatenation* of T_1 and t_2 as follows: $T_1 \cdot^s_\sigma t_2 = \{t_2(u \leftarrow t_1) \mid u \in \texttt{leaf}(t_2, \sigma), t_1 \in T_1\}$. Therefore, $T_1 \cdot^s_\sigma t_2$ is the set of trees obtained from t_2 by replacing a leaf labeled by σ with a tree in T_1. We extend the sequential σ-concatenation operation to the tree languages $T_1, T_2 \subseteq F_\Sigma$ as follows: $T_1 \cdot^s_\sigma T_2 = \bigcup_{t_2 \in T_2} T_1 \cdot^s_\sigma t_2$. The *parallel σ-concatenation* of T_1 and t_2 is defined as $T_1 \cdot^p_\sigma t_2 = t_2(\texttt{leaf}(t_2, \sigma) \leftarrow T_1)$. Thus, $T_1 \cdot^p_\sigma t_2$ is the set of trees obtained from t_2 by replacing all leaves labeled by σ with a tree in T_1. Note that the parallel σ-concatenation also can be extended to the tree languages.

We observe that the sequential σ-concatenation is associative whereas the parallel version is not associative. Due to the non-associativity of the sequential concatenation, we have two variants of iterated sequential concatenations: *sequential top-down σ-star* and *sequential bottom-up σ-star*. We only consider the sequential σ-star operations since the iterated parallel concatenation does not preserve regularity [19]. For $\sigma \in \Sigma_0$ and $T \subseteq F_\Sigma$, we define the sequential top-down σ-star of T to be $T^{s,t,*}_\sigma = \bigcup_{k \geq 0} T^{s,t,k}_\sigma$ by setting $T^{s,t,0}_\sigma = \{\sigma\}$ and $T^{s,t,k}_\sigma = T \cdot^s_\sigma T^{s,t,k-1}_\sigma$ for $k \geq 1$. Similarly, we define the sequential bottom-up σ-star of T to be $T^{s,b,*}_\sigma = \bigcup_{k \geq 0} T^{s,b,k}_\sigma$ by setting $T^{s,b,0}_\sigma = \{\sigma\}$, $T^{s,b,1}_\sigma = T$ and $T^{s,b,k}_\sigma = T^{s,b,k-1}_\sigma \cdot^s_\sigma T$ for $k \geq 2$. Since we only consider the sequential σ-star operations, we call the sequential top-down (bottom-up, respectively) σ-star the top-down (bottom-up, respectively) σ-star and denote by $T^{t,*}_\sigma$ ($T^{b,*}_\sigma$, respectively) instead of $T^{s,t,*}_\sigma$ ($T^{s,b,*}_\sigma$, respectively) in the remaining sections.

3 Subtree-Free Regular Tree Language

There are several subfamilies of (regular) languages such as prefix-free, suffix-free and infix-free (regular) languages. For regular languages, some of these subfamilies have unique structural properties in their minimal DFAs and these properties often make the state complexity of the considered subfamilies different from that of general regular languages. For regular tree languages, we can similarly define proper subfamilies by adding some restrictions on the structure of minimal DTAs. We consider subtree-freeness in a tree language and define a *subtree-free tree language* as follows:

Definition 1. *A tree language L is subtree-free if, for any two trees t_1 and t_2 from L, t_1 is not a proper subtree of t_2.*

We can extend the subtree-freeness to the family of regular tree languages and define *subtree-free regular tree languages*. Then, the minimal DTAs recognizing the family have the following structural properties. It is interesting to note that the subtree-freeness of the tree language corresponds to the prefix-freeness of the string language since tree automata operate in the bottom-up direction.

Lemma 1. *A regular tree language L is subtree-free if and only if its minimal DTA A for L has only one final state and there is no transitions whose left-hand sides contain the final state.*

Recall that a regular language is prefix-free if and only if the unique final state of its minimal DFA does not have any out-transitions [1]. The properties of subtree-free regular tree languages in Lemma 1 are similar to that of prefix-free regular languages. This leads us to the following question: Are the state complexities for subtree-free regular tree languages similar to those for prefix-free regular languages considered by Han et al. [10]?

4 State Complexity of Concatenation

There are two types of concatenation operations when we consider the state complexity of regular tree languages. Piao and Salomaa [20] gave formal definitions of sequence and parallel concatenations and established two state complexities of regular tree languages for the operations. Note that the state complexity of regular tree languages considers incomplete minimal DTAs [19,20].

4.1 Sequential Concatenation

We first consider the state complexity of the sequential concatenation operation for subtree-free regular tree languages. We note that the state complexity of sequential concatenation obtained here differs from the state complexity of string concatenation since we need to remember the node where the σ-substitution has occurred.

Lemma 2. *Let A_1 and A_2 be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. For $\sigma \in \Sigma_0$, $(n_2+1)(n_1+n_2+1)-1$ states are sufficient for the minimal DTA of $L(A_1) \cdot_\sigma^s L(A_2)$.*

For the tight bound, we define subtree-free DTAs A and B such that state complexity of $L(A) \cdot_\sigma^s L(B)$ reaches the upper bound in Lemma 2. We choose $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$, where $\Sigma_0 = \{d\}, \Sigma_1 = \{a, b, c\}$ and $\Sigma_2 = \{a_2, b_2\}$. Let $A = (\Sigma, Q_A, q_{A,F}, g_A)$, where $Q_A = \{0, 1, \ldots, n_1 - 1\}, q_{A,F} = n_1 - 1$ and the transition function g_A is defined as follows:

- $d_{g_A} = 0$,
- $a_{g_A}(i) = (a_2)_{g_A}(i, i) = i + 1$, $0 \leq i \leq n_1 - 2$,
- $b_{g_A}(i) = (b_2)_{g_A}(i, i) = i$, $0 \leq i \leq n_1 - 2$,
- $c_{g_A}(i) = i$, $0 \leq i \leq n_1 - 2$.

Similarly, we define $B = (\Sigma, Q_B, q_{B,F}, g_B)$, where $Q_B = \{0, 1, \ldots, n_2-1\}$, $q_{B,F} = n_2 - 1$ and the transition function g_B is defined as follows:

- $d_{g_B} = 0$,
- $a_{g_B}(i) = (a_2)_{g_B}(i, i) = i$, $0 \leq i \leq n_2 - 2$,
- $b_{g_B}(i) = (b_2)_{g_B}(i, i) = i + 1$, $0 \leq i \leq n_2 - 2$,
- $c_{g_B}(i) = i + 1$, $0 \leq i \leq n_2 - 3$ and $c_{g_B}(n_2 - 2) = (c_2)_{g_B}(n_2 - 2, n_2 - 2) = 0$.

Note that both the final states of A and B do not have any out-transitions, thus, $L(A)$ and $L(B)$ are subtree-free regular tree languages. Now we show that the upper bound in Lemma 2 is reachable. Let $C = (\Sigma, Q_C, Q_{C,F}, g_C)$ be a new DTA constructed from A and B as in the proof of Lemma 2.

Lemma 3. *All states of C are reachable and pairwise inequivalent.*

From Lemma 2 and Lemma 3, we establish the following result.

Theorem 1. *Let A_1 and A_2 be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. For $\sigma \in \Sigma_0$, $(n_2 + 1)(n_1 + n_2 + 1) - 1$ states are sufficient and necessary in the worst-case for the minimal DTA of $L(A_1) \cdot_\sigma^s L(A_2)$.*

4.2 Parallel Concatenation

The parallel concatenation $L_1 \cdot_\sigma^p L_2$ is called the σ-product of L_1 and L_2 [7]. Piao and Salomaa obtained the state complexity of parallel concatenation [20], which is similar to that of catenation for regular string languages The state complexity of subtree-free regular tree languages for parallel concatenation turns out to be similar to the DFA state complexity for prefix-free regular languages [10].

Theorem 2. *Let A_1 and A_2 be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. For $\sigma \in \Sigma_0$, $n_1 + n_2 - 1$ states are sufficient and necessary in the worst-case for the minimal DTA of $L(A_1) \cdot_\sigma^p L(A_2)$.*

Proof. Let A_1 and A_2 be two subtree-free DTAs $A_i = (\Sigma, Q_i, q_{i,F}, g_i)$, for $i = 1, 2$. We denote the set containing the undefined state (q_{sink}) with Q_i by Q_i'; $Q_i' = Q_i \cup \{q_{sink}\}$. We construct a new DTA $B = (\Sigma, Q_B, q_{D,F}, g_B)$, where $Q_B = Q_2' \times Q_1'$, $Q_{B,F} = \{q \in Q_B \mid \pi_1(q) = q_{2,F}\}$, and g_B is defined as follows:

For $\tau \in \Sigma_0$, we define

$$\tau_{g_B} = \begin{cases} (\sigma_{g_2}, \tau_{g_1}) & \text{if } \tau_{g_1} = q_{1,F}, \\ (q_{sink}, \tau_{g_1}) & \text{if } \tau_{g_1} \text{ is defined and } \tau_{g_1} \neq q_{1,F}, \\ (\tau_{g_2}, q_{sink}) & \text{if } \tau_{g_2} \text{ is defined,} \\ \text{undefined,} & \text{if } \tau_{g_2} \text{ and } \tau_{g_1} \text{ are both undefined.} \end{cases}$$

For $\tau \in \Sigma_m$ and $(p_i, q_i) \in Q_B$, where $m \geq 1$ and $1 \leq i \leq m$, we define $\tau_{g_B}((p_1, q_1), \ldots, (p_m, q_m))$ to be

(i) $(\sigma_{g_2}, \tau_{g_1}(q_1, \ldots, q_m))$ if $\tau_{g_1}(q_1, \ldots, q_m) = q_{1,F}$.
(ii) $(\tau_{g_2}(p_1, \ldots, p_m), q_{sink})$ if $\tau_{g_2}(p_1, \ldots, p_m)$ is defined.
(iii) undefined, otherwise.

Now we consider the computation of the new DTA B. The first component of a state in Q_B simulates A_2 assuming that every leaf node labeled by σ is substituted with a tree in $L(A_1)$. The second component of a state in Q_B simulates A_1 and changes the first component into σ_{g_2} when it becomes the final state of A_1. Since each state of B is a pair of states from A_2 and A_1, the total number of states is $(n_1 + 1)(n_2 + 1)$. However, the states of D consist of two states from Q_1 and Q_2, respectively, are unreachable by the construction except $(\sigma_{g_2}, q_{1,F})$ that can be merged with (σ_{g_2}, q_{sink}). We have one more unreachable state $(q_{sink}, q_{1,F})$ since the first component should be σ_{g_2} when the second component is the final state of A_1. Furthermore, we remove the undefined state (q_{sink}, q_{sink}). Therefore, the upper bound is $n_1 + n_2 - 1$. The lower bound example for parallel concatenation can be given by unary tree languages. Let $\texttt{word}(t)$ denote the sequence of symbols labeling the nodes of a unary tree t in a bottom-up way. For instance, $\texttt{word}(t)$ for a unary tree $t = b(a_1(\ldots a_n(x)\ldots))$ is $a_n a_{n-1} \cdots a_1 b$. Note that the label of the leaf (which is x in t) is not included. Let L_1 and L_2 be subtree-free regular tree languages accepting unary trees t_1 and t_2 such that $\texttt{word}(t_1) = a^{n_1-1}$ and $\texttt{word}(t_2) = a^{n_2-1}$, respectively. It is easy to verify that n_1 (respectively, n_2) states are necessary for recognizing L_1 (respectively, L_2). By parallel concatenation, $L_1 \cdot_\sigma^p L_2$ recognizes a tree t' such that $\texttt{word}(t') = a^{n_1+n_2-2}$. It is easy to observe that $n_1 + n_2 - 1$ states are necessary for $L_1 \cdot_\sigma^p L_2$. Thus, we know that $n_1 + n_2 - 1$ is a tight bound for the parallel concatenation operation. □

5 State Complexity of Kleene-Star

Piao and Salomaa [19] gave definitions of two types of Kleene-star operations: bottom-up star and top-down star operations and obtained the tight state complexities for the operations. Note that they only considered the sequential variants of iterated concatenation as Kleene-star operation on trees since the parallel version does not preserve regularity [19]. We also observe that the same holds for subtree-free regular tree languages.

5.1 Bottom-Up Star

First we give an upper bound for the state complexity of subtree-free regular tree languages for bottom-up Kleene-star operation.

Lemma 4. *Let $A = (\Sigma, Q, q_F, g)$ be a subtree-free minimal DTA with n states, where $n \geq 2$. For $\sigma \in \Sigma_0$, $2n + 1$ states are sufficient for the minimal DTA of $L(A)_\sigma^{b,*}$.*

Proof. Let $Q' = Q \cup \{q_{sink}\}$. We construct a new DTA $B = (\Sigma, Q_B, Q_{B,F}, g_B)$ recognizing $L(A)_\sigma^{b,*}$, where $Q_B = Q' \times Q' \cup \{q_{new}\}$ and $Q_{B,F} = \{(q_F, q_{sink}), q_{new}\}$.

We reach q_{new} by reading a single node tree labeled by σ. Therefore, we define the transitions of q_{new} to be equal to those of (q_{sink}, σ_g) except that q_{new} is a final state and (q_{sink}, σ_g) is not necessarily a final state. We assume that σ_g is well defined without loss of generality because otherwise, $L(A)_\tau^{b,*} = L(A)_\tau^{b,0} \cup L(A)_\tau^{b,1} = \{\sigma\} \cup L(A)$. For $\tau \in \Sigma_0 \setminus \{\sigma\}$, we define

$$\tau_{g_B} = \begin{cases} (\sigma_g, \tau_g) & \text{if } \tau_g = q_F, \\ (q_{sink}, \tau_g), & \text{if } \tau_g \text{ is defined and } \tau_g \neq q_F, \\ \text{undefined}, & \text{if } \tau_g \text{ is undefined.} \end{cases}$$

For $\tau \in \Sigma_m$ and $(p_i, q_i) \in Q_B$, where $m \geq 1$ and $1 \leq i \leq m$, we define $\tau_{g_B}((p_1, q_1), \ldots, (p_m, q_m))$ to be

(i) $(\sigma_g, \tau_g(q_1, \ldots, q_m))$ if $\tau_g(q_1, \ldots, q_m) = q_F$.
(ii) $(q_{sink}, \tau_g(q_1, \ldots, q_m))$ if $\tau_g(q_1, \ldots, q_m) \neq q_F$ is defined.
(iii) (x, q_{sink}) if $\tau_g(q_1, \ldots, q_m)$ is undefined. Here, x is
  - $\tau_g(q_1, \ldots, q_{i-1}, p_i, q_{i+1}, \ldots, q_m)$ if $\tau_g(q_1, \ldots, q_{i-1}, p_i, q_{i+1}, \ldots, q_m)$ is defined and p_j is undefined for $1 \leq j \leq m$ and $i \neq j$.
  - q_{sink}, otherwise.

(iv) undefined, otherwise.

The second component of a state in Q_B simply simulates A while the first component of the state in Q_B simulates A under the assumption that at least a leaf labeled by σ has been replaced by a tree in $L(A)_\sigma^{b,k}$, where $k \geq 0$. Note that the total number of states in Q_B is $(n+1)^2+1$. When the second component reaches the final state q_F, we should have σ_g in the first state. After we reach (σ_g, q_F), the second component should be q_{sink}, since there is no transition defined for the final state. Thus, a state pair in Q_B cannot be in a form of $(p_i, q_i) \in Q \times Q$ except (σ_g, q_F). Therefore, we have $n^2 - 1$ unreachable states. We can merge two final states into one since (q_F, q_{sink}) has no out transitions. After we merge two states, the resulting state has the transitions of (q_{sink}, σ_g) and, thus, the resulting automaton is still deterministic. Furthermore, we remove one more state (q_{sink}, q_{sink}), which is undefined. Therefore, the sufficient number of states for $L(A)_\sigma^{b,*}$ is $2n+1$. □

We show a lower bound example whose state complexity corresponds to the upper bound in Lemma 4. Let $\Sigma = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2$, where $\Sigma_0 = \{e\}, \Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{a_2, b_2\}$. We define a subtree-free DTA $D = (\Sigma, Q_D, q_{D,F}, g_D)$, where $Q_D = \{0, 1, \ldots, n-1\}, q_{D,F} = n-1$ and the transition function g_D is defined as follows:

- $e_{g_D} = 0$,
- $a_{g_D}(i) = (a_2)_{g_D}(i, i) = i+1, 0 \leq i \leq n-3, a_{g_D}(n-2) = (a_2)_{g_D}(n-2, n-2) = 0$,
- $b_{g_D}(n-2) = (b_2)_{g_D}(n-2, n-2) = n-1$.

All transitions of g_D not defined above are undefined. Using D and the upper bound construction of the proof of Lemma 4, we construct a new DTA $E = (\Sigma, Q_E, Q_{E,F}, g_E)$ accepting $L(D)_e^{b,*}$, namely $L(D)_e^{b,*} = L(E)$.

Lemma 5. *All states of E are reachable and pairwise inequivalent.*

Theorem 3. *Let A be a subtree-free minimal DTA with n states, where $n \geq 2$. For $\sigma \in \Sigma_0$, $2n+1$ states are sufficient and necessary in the worst-case for the minimal DTA of $L(A)_\sigma^{b,*}$.*

5.2 Top-Down Star

Now we investigate the state complexity for top-down star of subtree-free regular tree languages. Note that the state complexity of regular tree languages for top-down star coincides with the state complexity of regular string languages for star [19]. We show that the state complexity of subtree-free regular tree languages for top-down star also coincides with that of prefix-free regular string languages for star.

Theorem 4. *Let $A = (\Sigma, Q, q_F, g)$ be a subtree-free minimal DTA with n states, where $n \geq 2$. For $\sigma \in \Sigma_0$, n states are sufficient and necessary in the worst-case for the minimal DTA of $L(A)_\sigma^{t,*}$.*

Proof. The upper bound construction for the top-down star operation is straightforward since it is similar to the construction of the Kleene-star operation for prefix-free languages [10]. We define $B = (\Sigma, Q_B, Q_{B,F}, g_B)$, where $Q_B = Q \cup \{q_{new}\}$ and $Q_{B,F} = \{q_{new}, q_F\}$. As in the proof of Lemma 4, q_{new} is defined as a state that is reached by reading a single node tree labeled by σ. Therefore, we define the transitions of q_{new} to be equal to those of σ_g except that q_{new} is a final state and σ_g is not necessarily a final state.

For $\tau \in \Sigma_0 \setminus \{\sigma\}$, we define τ_{g_B} to be equal to σ_{g_B} if $\tau_g = q_F$. Otherwise, we set $\tau_{g_B} = \tau_g$. For $\tau \in \Sigma_m$ and $q_i \in Q_B$, where $m \geq 1$ and $1 \leq i \leq m$, we define $\tau_{g_B}(q_1, \ldots, q_m)$ to be

(i) σ_g if $\tau_g(q_1, \ldots, q_m) = q_F$.
(ii) $\tau_g(q_1, \ldots, q_m)$ if $\tau_g(q_1, \ldots, q_m) \neq q_F$ is defined.
(iii) undefined, otherwise.

There are now $n+1$ states in Q_B and we merge two states q_F and q_{new} into one state while maintaining determinism since q_F does not have any out-transitions. Thus, the sufficient number of states for $L(A)_\sigma^{t,*}$ is n.

Now we show that n states are necessary for recognizing $L(A)_\sigma^{t,*}$ by a simple lower bound. Let L be the following unary tree language:

$$L(A) = \{t \mid \texttt{word}(t) = a^{n-1}\}.$$

It is easy to verify that a DTA A needs at least n states for recognizing L. Then, we construct a DTA B as described in the upper bound construction. Note that B accepts $L(A)_\sigma^{t,*}$ and has n states. Therefore, n is a tight bound for the minimal DTA of $L(A)_\sigma^{t,*}$. □

6 Intersection and Union

For regular languages, the state complexities of intersection and union are quite trivial. The upper bound construction is based on the Cartesian product of states and yields $n_1 n_2$ states. Yu et al. [27] showed that $n_1 n_2$ is tight. For regular tree languages, the tight bounds of intersection and union are similar to the string case. Since we consider incomplete DTAs, it is easy to verify that the state complexities for intersection and union are $n_1 n_2 + n_1 + n_2$. The state complexities of subtree-free regular tree languages for intersection and union operations are the same as those of prefix-free regular string languages [10]. The exact complexities are slightly different since we consider incomplete DTAs. First, we establish the tight bound of intersection for subtree-free regular tree languages as follows.

Theorem 5. *Let A_1 and A_2 be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. Then, $n_1 n_2 - n_1 - n_2 + 2$ states are sufficient and necessary in the worst-case for the minimal DTA of $L(A_1) \cap L(A_2)$.*

Proof. Let A_1 and A_2 be two subtree-free DTAs, where $A_i = (\Sigma, Q_i, q_{i,F}, g_i)$ for $i = 1, 2$. We construct a new DTA $B = (\Sigma, Q_B, Q_{B,F}, g_B)$, where $Q_B = Q_1 \times Q_2, Q_{B,F} = \{q \in Q_B \mid \pi_1(q) = q_{1,F}$ and $\pi_2(q) = q_{2,F}\}$, and g_B is defined as follows. For $\tau \in \Sigma_0$, we define $\tau_{g_B} = \tau_{g_1} \times \tau_{g_2}$. For $\tau \in \Sigma_m$ and $(p_i, q_i) \in Q_B$, where $m \geq 1$ and $1 \leq i \leq m$, we define $\tau_{g_B}((p_1, q_1), \ldots, (p_m, q_m))$ to be

(i) $(\tau_{g_1}(p_1, \ldots, p_m), \tau_{g_2}(q_1, \ldots, q_m))$ if $\tau_{g_1}(p_1, \ldots, p_m)$ and $\tau_{g_2}(q_1, \ldots, q_m)$ are both defined.
(ii) undefined, otherwise.

Now B has $n_1 n_2$ states. We assume that a state in Q_B contains $q_{1,F}$ or $q_{2,F}$, which is the final state of A_1 or A_2, respectively. Since A_1 and A_2 have no transitions defined for their final states, there are no transitions defined for the corresponding states in B, either. Note that the number of states containing $q_{1,F}$ or $q_{2,F}$ is $n_1 + n_2 - 1$. Among these states, $(q_{1,F}, q_{2,F})$ is the final state while the others are non-final. We remove $n_1 + n_2 - 2$ non-final states, which are the sink states. Then, the sufficient number of states for intersection is $n_1 n_2 - n_1 - n_2 + 2$. We give lower bound examples whose state complexity meets the upper bound. Let L_1 and L_2 be subtree-free unary tree languages as follows:

$$L_1 = \{t \mid \texttt{word}(t) = (a^{n_1 - 1})^*\} \text{ and } L_2 = \{t \mid \texttt{word}(t) = (a^{n_2 - 1})^*\},$$

Then, two DTAs A_1 and A_2 need at least n_1 and n_2 states for recognizing L_1 and L_2, respectively. Assume $n_1 - 1$ and $n_2 - 1$ are relatively prime. Then, $L_1 \cap L_2 = \{t \mid \texttt{word}(t) = (a^{(n_1-1)(n_2-1)})^*\}$ and thus, requires at least $n_1 n_2 - n_1 - n_2 + 2$ states. □

Next, we examine the state complexity of union.

Theorem 6. *Let A_1 and A_2 be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. Then, $n_1 n_2 + n_1 + n_2 - 2$ states are sufficient and necessary in the worst-case for the minimal DTA of $L(A_1) \cup L(A_2)$.*

Proof. Let A_1 and A_2 be two subtree-free DTAs $A_i = (\Sigma, Q_i, q_{i,F}, g_i)$ for $i = 1, 2$. Let $Q_i' = Q_i \cup \{q_{sink}\}$. We construct a new DTA $B = (\Sigma, Q_B, Q_{B,F}, g_B)$, where $Q_B = Q_1' \times Q_2', Q_{B,F} = \{q \in Q_B \mid \pi_1(q) = q_{1,F} \text{ or } \pi_2(q) = q_{2,F}\}$, and g_B is defined as follows. For $\tau \in \Sigma_0$, we define $\tau_{g_B} = \tau_{g_1} \times \tau_{g_2}$. For $\tau \in \Sigma_m$ and $(p_i, q_i) \in Q_B$, where $m \geq 1$ and $1 \leq i \leq m$, we define $\tau_{g_B}((p_1, q_1), \ldots, (p_m, q_m))$ to be

(i) $(\tau_{g_1}(p_1, \ldots, p_m), \tau_{g_2}(q_1, \ldots, q_m))$ if either $\tau_{g_1}(p_1, \ldots, p_m)$ or $\tau_{g_2}(q_1, \ldots, q_m)$ is defined.
(ii) undefined, otherwise.

Note that we have $(n_1 + 1)(n_2 + 1)$ states. First, we remove the sink state (q_{sink}, q_{sink}) and merge three final states $(q_{sink}, q_{2,F})$, $(q_{1,F}, q_{sink})$ and $(q_{1,F}, q_{2,F})$ into one final state since they are all equivalent. Thus, the cardinality of B is $n_1n_2 + n_1 + n_2 - 2$. Now we consider a lower bound example for the claimed upper bound. We choose $\Sigma = \Sigma_0 \cup \Sigma_1$, where $\Sigma_0 = \{e\}$ and $\Sigma_1 = \{a, b\}$. Let L_1 and L_2 be subtree-free unary tree languages as follows:

$$L_1 = \{t_1 \mid \texttt{word}(t_1) = w_1a \text{ and } |w_1|_a = n_1 - 2\},$$
$$L_2 = \{t_2 \mid \texttt{word}(t_2) = w_2b \text{ and } |w_2|_b = n_2 - 2\}.$$

Note that there are the minimal DTAs of size n_1 and n_2 for L_1 and L_2, respectively. Let M be a new DTA recognizing $L_1 \cup L_2$. Then, M should count both a's and b's simultaneously. Since the number of a's can be from 0 to $n_1 - 2$ and the number of b's can be from 0 to $n_2 - 2$, M requires $(n_1 - 1)(n_2 - 1)$ states. Assume that M reads $(n_1 - 1)$'th a, then M should be in one of $n_2 - 1$ final states depending on the number of b's that we have read. Similarly, M reaches $n_2 - 1$ non-final states by reading more a's from the final states. Analogously, M reaches $n_1 - 1$ final states and $n_1 - 1$ non-final states by reading $(n_2 - 1)$'th and n_2'th b. Now the number of necessary states is $n_1n_2 + n_1 + n_2 - 3$. We have one more final state that is reached by reading a unary tree such that $\texttt{word}(t) = a^{n_1-1}b^{n_2-1}$. Therefore, it follows that $n_1n_2 + n_1 + n_2 - 2$ states are necessary for union. □

Table 1. Comparison table among the state complexity of basic operations for subtree-free, general regular tree languages and prefix-free regular string languages

operations	subtree-free	general	prefix-free (string)
$L_1 \cdot_\sigma^s L_2$	$(n+1)(m+n+1)-1$	$(n+1)(m \cdot 2^n + 2^{n-1}) - 1$	$m+n-2$
$L_1 \cdot_\sigma^p L_2$	$m+n-1$	$m \cdot 2^n + 2^{n-1} - 1$	
$L_\sigma^{b,*}$	$2n+1$	$(n+1)2^{n-1} + 2^{n-2}$	n
$L_\sigma^{t,*}$	n	$2^{n-1} + 2^{n-2}$	
$L_1 \cap L_2$	$mn-m-n+2$	$mn+m+n$	$mn-2(m+n)+6$
$L_1 \cup L_2$	$mn+m+n-2$	$mn+m+n$	$mn-2$

7 Conclusions

Regular tree languages often appear to be subtree-free in practice. For instance, all XML documents from a specific XML schema have a unique root element and, thus, the set of such documents is a subtree-free tree language. We have defined the family of subtree-free regular tree languages, which is a proper subfamily of regular tree languages. Then, we have investigated the state complexity of subtree-free regular tree languages and obtained the tight bounds.

We have summarized the tight bounds and compared with that of general regular tree languages in Table 1. We have shown that the tight bounds of basic operations for subtree-free regular tree languages are linear (parallel concatenation, bottom-up star and top-down star) or at most quadratic (sequential concatenation, intersection and union) with respect to the sizes of input DTAs. We also have compared with the state complexity prefix-free regular string languages. Interestingly, the state complexity of subtree-free regular tree languages coincides with the state complexity of the incomplete DFAs for prefix-free regular string languages.

Acknowledgements. We wish to thank the referees for the careful reading of the paper and many valuable suggestions.

This research was supported by the Basic Science Research Program through NRF funded by MEST (2012R1A1A2044562).

References

1. Berstel, J., Perrin, D.: Theory of Codes. Academic Press, Inc. (1985)
2. Câmpeanu, C., Culik II, K., Salomaa, K., Yu, S.: State complexity of basic operations on finite languages. In: Boldt, O., Jürgensen, H. (eds.) WIA 1999. LNCS, vol. 2214, pp. 60–70. Springer, Heidelberg (2001)
3. Comon, H., Dauchet, M., Jacquemard, F., Lugiez, D., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications (2007), Electronic book available at `www.tata.gforge.inria.fr`
4. Domaratzki, M., Okhotin, A.: State complexity of power. Theoretical Computer Science 410(24-25), 2377–2392 (2009)
5. Ésik, Z., Gao, Y., Liu, G., Yu, S.: Estimation of state complexity of combined operations. Theoretical Computer Science 410(35), 3272–3280 (2009)
6. Gao, Y., Salomaa, K., Yu, S.: The state complexity of two combined operations: Star of catenation and star of reversal. Fundamenta Informaticae 83(1-2), 75–89 (2008)
7. Gécseg, F., Steinby, M.: Handbook of Formal Languages. Tree languages. In: vol. 3, pp. 1–68 (1997)
8. Han, Y.-S., Salomaa, K.: State complexity of union and intersection of finite languages. International Journal of Foundations of Computer Science 19(3), 581–595 (2008)
9. Han, Y.-S., Salomaa, K.: State complexity of basic operations on suffix-free regular languages. Theoretical Computer Science 410(27-29), 2537–2548 (2009)

10. Han, Y.-S., Salomaa, K., Wood, D.: State complexity of prefix-free regular languages. In: Proceedings of the 8th International Conference on Descriptional Complexity of Formal Systems, pp. 165–176 (2006)
11. Hricko, M., Jirásková, G., Szabari, A.: Union and intersection of regular languages and descriptional complexity. Proceedings of the 7th International Conference on Descriptional Complexity of Formal Systems 2005, 170–181 (2005)
12. Jirásek, J., Jirásková, G., Szabari, A.: State complexity of concatenation and complementation. International Journal of Foundations of Computer Science 16(3), 511–529 (2005)
13. Martens, W., Niehren, J.: On the minimization of XML schemas and tree automata for unranked trees. Journal of Computer System Sciences 73(4), 550–583 (2007)
14. Maslov, A.: Estimates of the number of states of finite automata. Soviet Mathematics Doklady 11, 1373–1375 (1970)
15. Nerode, A.: Linear automaton transformations. Proceedings of the American Mathematical Society 9(4), 541–544 (1958)
16. Neven, F.: Automata theory for XML researchers. ACM SIGMOD Record 31(3), 39–46 (2002)
17. Piao, X., Salomaa, K.: State trade-offs in unranked tree automata. In: Proceedings of the 13th International Conference on Descriptional Complexity of Formal Systems, pp. 261–274 (2011)
18. Piao, X., Salomaa, K.: Transformations between different models of unranked bottom-up tree automata. Fundamenta Informaticae 109(4), 405–424 (2011)
19. Piao, X., Salomaa, K.: State complexity of kleene-star operations on trees. In: Dinneen, M.J., Khoussainov, B., Nies, A. (eds.) Computation, Physics and Beyond. LNCS, vol. 7160, pp. 388–402. Springer, Heidelberg (2012)
20. Piao, X., Salomaa, K.: State complexity of the concatenation of regular tree languages. Theoretical Computer Science 429, 273–281 (2012)
21. Pighizzini, G., Shallit, J.: Unary language operations, state complexity and Jacobsthal's function. International Journal of Foundations of Computer Science 13(1), 145–159 (2002)
22. Rabin, M.O., Scott, D.: Finite automata and their decision problems. IBM Journal of Research and Development 3(2), 114–125 (1959)
23. Rampersad, N.: The state complexity of L^2 and L^k. Information Processing Letters 98(6), 231–234 (2006)
24. Salomaa, A., Salomaa, K., Yu, S.: State complexity of combined operations. Theoretical Computer Science 383(2-3), 140–152 (2007)
25. Salomaa, K., Yu, S.: On the state complexity of combined operations and their estimation. International Journal of Foundations of Computer Science 18, 683–698 (2007)
26. Yu, S.: State complexity of regular languages. Journal of Automata, Languages and Combinatorics 6(2), 221–234 (2001)
27. Yu, S., Zhuang, Q., Salomaa, K.: The state complexities of some basic operations on regular languages. Theoretical Computer Science 125(2), 315–328 (1994)