International Journal of Foundations of Computer Science
Vol. 27, No. 6 (2016) 705–724
© World Scientific Publishing Company
DOI: 10.1142/S0129054116500246

Operational State Complexity of Subtree-Free Regular Tree Languages*

Sang-Ki Ko†, Hae-Sung Eom‡ and Yo-Sub Han§
Department of Computer Science, Yonsei University
50, Yonsei-Ro, Seodaemun-Gu, Seoul 120-749, Republic of Korea
†*narame7@cs.yonsei.ac.kr*
‡*haesung@cs.yonsei.ac.kr*
§*emmous@cs.yonsei.ac.kr*

Received 19 March 2015
Accepted 2 December 2015
Communicated by Kai Salomaa

We introduce subtree-free regular tree languages that are closely related to 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 languages.

1. Introduction

State complexity is one of the most interesting topics in automata and formal language theory [2, 13, 17, 18, 29, 35, 36]. We can use the state complexity for measuring the descriptional complexity of finite automata and regular languages. Maslov [20] obtained the state complexity of catenation and Yu *et al.* [36] investigated the state complexity for basic operations. Later, Yu and his co-authors [7, 11, 32, 33] initiated the study on the state complexity of combined operations such as star-of-union, star-of-intersection and so on. Gao *et al.* [7, 9, 10] studied the state complexity of combined Boolean operations including multiple unions and multiple intersections. Researchers also considered the state complexity of multiple operations such as several catenations, intersections, unions or intersections [4, 6, 7, 31].

*A preliminary version of this paper appeared in Proceedings of the 15th International Workshop on Descriptional Complexity of Formal Systems, DCFS 2013, LNCS 8031, Springer-Verlag, 2013, pp. 66–77.
§Corresponding author.

Han *et al.* [5, 14, 15] observed that prefix-free and suffix-free regular languages have unique structural properties in their DFAs, which are crucial to obtain the precise state complexity. Based on the structural properties for such DFAs, they obtained the precise state complexity for prefix-free and suffix-free regular languages. For instance, 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 [15].

Regular tree languages and tree automata theory provide a formal framework for XML schema languages such as XML DTD, XML Schema, and Relax NG [21, 23]. In other words, a set of XML documents generated from a given XML schema is a regular tree language since a XML document is a tree-based structural document. Piao and Salomaa [25, 26] examined the state complexities between different models of unranked tree automata. They also investigated the state complexities of concatenation [28] and star [27] for regular tree languages.

We consider a proper subfamily of regular tree languages, called *subtree-free regular tree 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. We define a set T of trees to be *subtree-free* if a tree in T is not a subtree of another tree in T. We observe that the subtree-freeness is often well-preserved in practice: XML documents always have a single root element called the *document element*. The document element is, therefore, the very first element of an XML document and encloses all other elements [19]. Moreover, the document element does not appear elsewhere in many XML documents. For example, consider an XML Schema instance, which itself is an XML document. All XML Schemas should have the following start and end tags according to the XML Schema specification [8]:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
...
</xs:schema>
```

Note that all XML Schemas should have the `xs:schema` element as its root element and `xs:schema` cannot be used elsewhere in XML Schemas [8]. Namely, a set of XML Schemas is a subtree-free regular tree language.

We can estimate the space requirement for manipulating sets of XML documents under basic operations that satisfy the subtree-freeness by investigating the state complexity of regular tree languages with the same property. Notice that we consider state complexity of regular tree languages on ranked trees whereas XML schemas correspond to unranked tree automata. This is because it is difficult to compute the state complexity of unranked tree automata directly and, thus, researchers often rely on the state complexity of ranked tree automata for estimating the state complexity for unranked tree automata [24].

In Sec. 2, we define some basic notions. We define the subtree-free regular tree languages in Sec. 3 and identify the structural properties of their tree automata.

We obtain the state complexity for sequential and parallel concatenation in Sec. 4, bottom-up and top-down star in Sec. 5, and the intersection and union in Sec. 6. We conclude the paper in Sec. 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, 12] 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 stages of a computation, or configurations, of A are trees where some leaves may be labeled by states of A. We denote the set of Σ-labeled trees where exactly one leaf is labeled by a special symbol $x \notin \Sigma$ by $F_\Sigma[x]$. Thus the set of configurations of A consists of Σ'-trees, where $\Sigma'_0 = \Sigma_0 \sqcup \{Q\}$ and $\Sigma'_m = \Sigma_m$ when $m \geq 1$. Note that we use $\sqcup$ for denoting the disjoint union. 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 $Q^m \longrightarrow Q$. The nondeterministic (bottom-up or top-down) and deterministic bottom-up tree automata accept the family of *regular tree languages*.

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 [27, 28]. For any $\sigma \in \Sigma_0$ and $t \in F_\Sigma$, $\texttt{leaf}(t, \sigma)$ is the set of leaf nodes labeled by σ. 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, $T_1 \cdot_\sigma^s t_2$ 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:

$$T_1 \cdot_\sigma^s T_2 = \bigcup_{t_2 \in T_2} T_1 \cdot_\sigma^s t_2.$$

The *parallel σ-concatenation* of T_1 and t_2 is defined as

$$T_1 \cdot_\sigma^p t_2 = t_2(\texttt{leaf}(t_2, \sigma) \leftarrow T_1).$$

Thus, $T_1 \cdot_\sigma^p t_2$ 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 can also be extended to the tree languages.

We observe that the sequential σ-concatenation is not associative whereas the parallel version is 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 [27]. For $\sigma \in \Sigma_0$ and $T \subseteq F_\Sigma$, we define the sequential top-down σ-star of T to be

$$T_\sigma^{s,t,*} = \bigcup_{k \geq 0} T_\sigma^{s,t,k}$$

by setting $T_\sigma^{s,t,0} = \{\sigma\}$ and $T_\sigma^{s,t,k} = T \cdot_\sigma^s T_\sigma^{s,t,k-1}$ for $k \geq 1$. Similarly, we define the sequential bottom-up σ-star of T to be

$$T_\sigma^{s,b,*} = \bigcup_{k \geq 0} T_\sigma^{s,b,k}$$

by setting $T_\sigma^{s,b,0} = \{\sigma\}$, $T_\sigma^{s,b,1} = T$ and $T_\sigma^{s,b,k} = T_\sigma^{s,b,k-1} \cdot_\sigma^s T$ 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 it by $T_\sigma^{t,*}$ ($T_\sigma^{b,*}$, respectively) instead of $T_\sigma^{s,t,*}$ ($T_\sigma^{s,b,*}$, 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. *We define a tree language L to be* subtree-free *if, for any two distinct trees t_1 and t_2 in L, t_1 is not a 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.

Lemma 2. *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 are no transitions whose left-hand sides contain the final state.*

Proof. We say that a DTA A is a subtree-free DTA if $L(A)$ is subtree-free. Suppose that $A = (\Sigma, Q, q_f, g)$ is a subtree-free minimal DTA. We first prove that if A has no transitions whose left-hand sides contain the final state, then $L(A)$ is subtree-free. Assume that $L(A)$ recognizes two trees t and t', where t' is a subtree of t. Since A accepts $t = \sigma(t_1, \ldots, t_m)$, $\sigma_g(q_1, \ldots, q_m)$ should be defined as q_f, where $q_i = (t_i)_g$ for $1 \leq i \leq m$. Note that t' must be t_i itself or a subtree of t_i for $1 \leq i \leq m$. By the assumption, A also accepts t' and, thus, $q_f = (t')_g$ holds. This follows that $t' \neq t_i$ since there are no transitions containing q_f on the left-hand side. Using a similar argument, it is easy to show that t' cannot be a subtree of t_i.

Now we prove the other direction: If $L(A)$ is subtree-free, then A has no transitions whose left-hand sides contain the final state. Assume that there is a transition $q' = \sigma_g(q_1, \ldots, q_f, \ldots, q_m)$ and A accepts t. This means that A reaches q_f by recognizing t and then A reaches another state q' by reading $\sigma(q_1, \ldots, t, \ldots, q_m)$, which is a supertree of t. Note that every state in Q except q_f can move to the final state. Thus, A can reach the final state q_f from q' as well and accept a supertree of t — contradiction. □

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. 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].[a] The properties of subtree-free regular tree languages in Lemma 2 are similar to that of prefix-free regular languages. This leads us to study the following question: Are the state complexities for subtree-free regular tree languages similar to those for prefix-free regular languages.

4. State Complexity of Concatenation

There are two types of concatenation operations for regular tree languages. Piao and Salomaa [28] gave formal definitions of sequential and parallel concatenations and established two state complexities of regular tree languages for both operations. Note that the state complexity of regular tree languages considers incomplete minimal DTAs [27, 28].

[a]We assume that there is no sink state in a minimal DFA.

4.1. *Sequential concatenation*

We first consider the state complexity of the sequential concatenation operation for subtree-free regular tree languages.

Lemma 3. *Let A and B be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. For $\sigma \in \Sigma_0$, $(n_2+1)(n_1+2^{n_2}-1)-1$ states are sufficient for the minimal DTA of $L(A) \cdot_\sigma^s L(B)$.*

Proof. We show the sufficient number of states for the minimal DTA of $L(A) \cdot_\sigma^s L(B)$ by the DTA construction for the desired operation. Let $A = (\Sigma, Q_A, q_{A,F}, g_A)$ and $B = (\Sigma, Q_B, q_{B,F}, g_B)$ be two subtree-free DTAs. Let $Q'_A = Q_A \sqcup \{q_{\text{sink}}\}$ and $Q'_B = Q_B \sqcup \{p_{\text{sink}}\}$. Note that $q_{A,F}$ and $q_{B,F}$ are the unique final states of A and B, respectively. Without loss of generality, assume that σ_{g_B} is defined. We define a new DTA $C = (\Sigma, Q_C, Q_{C,F}, g_C)$, where

$$Q_C = Q'_B \times 2^{Q_B} \times Q'_A \text{ and } Q_{C,F} = \{q \in Q_C \mid \pi_2(q) \cap Q_{B,F} \neq \emptyset\}.$$

For $\tau \in \Sigma_m, m \geq 0, q_1, \ldots, q_m \in Q'_A$, we define

$$\overline{\tau_{g_A}(q_1,\ldots,q_m)} = \begin{cases} \tau_{g_A}(q_1,\ldots,q_m) & \text{if } \tau_{g_A}(q_1,\ldots,q_m) \text{ is defined,} \\ q_{\text{sink}}, & \text{otherwise.} \end{cases}$$

We define $\overline{\tau_{g_B}(p_1,\ldots,p_m)}$ analogously. For $\tau \in \Sigma_0$, we define

$$\tau_{g_C} = \begin{cases} (\overline{\tau_{g_B}}, \{\sigma_{g_B}\}, \tau_{g_A}) & \text{if } \tau_{g_A} = q_{A,F}, \\ (\overline{\tau_{g_B}}, \emptyset, \overline{\tau_{g_A}}) & \text{if } \tau_{g_A} \text{ or } \tau_{g_B} \text{ is defined, } \tau_{g_A} \neq q_{A,F}, \\ \text{undefined,} & \text{if } \tau_{g_B} \text{ and } \tau_{g_A} \text{ are both undefined.} \end{cases}$$

For $\tau \in \Sigma_m$ and $(p_i, P_i, q_i) \in Q_C$, where $m \geq 1, p_i \in Q'_B, P_i \subseteq Q_B$, $q_i \in Q'_A$ and $1 \leq i \leq m$, we define $\tau_{g_C}((p_1, P_1, q_1), \ldots, (p_m, P_m, q_m))$ to be one of the following three:

(i) $(\overline{\tau_{g_B}(p_1,\ldots,p_m)}, X, \tau_{g_A}(q_1,\ldots,q_m))$ if $\tau_{g_A}(q_1,\ldots,q_m) = q_{A,F}$, where

$$X = \bigcup_{i=1}^{m} \left(\bigcup_{x \in P_i} \tau_{g_B}(p_1,\ldots,p_{i-1},x,p_{i+1},\ldots p_m) \right) \cup \{\sigma_{g_B}\}.$$

(ii) $(\overline{\tau_{g_B}(p_1,\ldots,p_m)}, Y, \overline{\tau_{g_A}(q_1,\ldots,q_m)})$ if $\tau_{g_A}(q_1,\ldots,q_m) \neq q_{A,F}$ and $[Y \neq \emptyset$, or at least one of $\tau_{g_B}(p_1,\ldots,p_m)$ and $\tau_{g_A}(q_1,\ldots,q_m)$ is defined$]$. Here,

$$Y = \bigcup_{i=1}^{m} \left(\bigcup_{x \in P_i} \tau_{g_B}(p_1,\ldots,p_{i-1},x,p_{i+1},\ldots p_m) \right).$$

(iii) undefined, otherwise.

The first component of a state in Q_C simulates the computation of B assuming that there is no occurrence of σ-substitution below the current node and the third component simulates the computation of A. The second component simulates B assuming that a σ-substitution has occurred below the current node. When the third component arrives at the final state of A, we add the state that can be reached by reading a leaf labeled by $\sigma \in \Sigma_0$ to the second component. Then, the DTA C simulates the computation of B with a state from the second component and the other states from the first component. If the second component has a final state of B, this implies that the DTA C has simulated a tree of B, where a σ-substitution has occurred. Therefore, the construction guarantees that C recognizes the sequential concatenation of A and B.

Now from the construction, we know that $(n_2 + 1) \cdot 2^{n_2} \cdot (n_1 + 1)$ states are sufficient for simulating the sequential concatenation. However, any two states $(p, P, q_1), (p, P, q_2) \in Q_C$ are always equivalent if q_1 is final and q_2 is q_{sink} since there is no transition defined for the only final state of Q_A. This implies that we can merge $(n_2 + 1) \cdot 2^{n_2}$ states. Now we consider the states where the third component is neither the sink state q_{sink} nor the final state of A. For a state q that is not the sink state or the final state of A, we observe that a state $(p, P, q) \in Q_C$ is unreachable if $P \neq \emptyset$. Furthermore, we remove one more state $(p_{\text{sink}}, \emptyset, q_{\text{sink}})$ and establish an upper bound of

$$(n_2+1) \cdot 2^{n_2} \cdot (n_1+1) - (n_2+1) \cdot 2^{n_2} - (n_2+1)(2^{n_2}-1)(n_1-1) - 1 = (n_2+1)(n_1+2^{n_2}-1) - 1$$

states. □

For the tight bound, we present two subtree-free DTAs A and B whose state complexity of $L(A) \cdot_{\sigma}^{s} L(B)$ reaches the upper bound in Lemma 3. We choose $\Sigma = \Sigma_0 \sqcup \Sigma_1 \sqcup \Sigma_2$, where $\Sigma_0 = \{d\}, \Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{c, e, f\}$. 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:

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

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:

- $d_{g_B} = 0$,
- $a_{g_B}(i) = i, 0 \leq i \leq n_2 - 2$,
- $b_{g_B}(i) = i + 1$, $0 \leq i \leq n_2 - 2$,
- $c_{g_B}(i, j) = (i - j) \pmod{n_2}$, $0 \leq i, j \leq n_2 - 1$,
- $e_{g_B}(0, 0) = 0$ and $e_{g_B}(i, j) = n_2 - 1$, $i \neq j, 0 \leq i, j \leq n_2 - 2$,
- $f_{g_B}(i, 0) = 0, 0 \leq i \leq n_2 - 2$ and $f_{g_B}(i, 1) = i + 1, 0 \leq i \leq n_2 - 2$.

Note that both the final states of A and B do not have any out-transitions and, thus, $L(A)$ and $L(B)$ are subtree-free regular tree languages by Lemma 2.

We show that the upper bound in Lemma 3 is reached using A and B. 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 3. Note that here a symbol d is used to sequentially concatenate two trees.

Lemma 4. *All states of C are reachable and pairwise inequivalent.*

Proof. First, we show that all states of C are reachable. Note that a state n_1 is a sink (undefined) state of A and a state n_2 is a sink state of B. Before we consider the reachability of states, we remind that any two states $(i, P, n_1 - 1)$ and (i, P, n_1) are equivalent since there is no transition defined for the final state $n_1 - 1$.

The computation of C assigns the state $(0, \emptyset, 0)$ to a leaf labeled by d. From $(0, \emptyset, 0)$, we reach $(i, \emptyset, j)$ by reading a sequence of unary symbols $a^j b^i$, where $0 \leq i \leq n_2$ and $0 \leq j \leq n_1{-}2$. Choose a state $q = (i, \emptyset, 0), 0 \leq i \leq n_2{-}1$ and a tree $t = c(q, x) \in F_{\Sigma'}[x]$. Then, the computation of C on $t(x \leftarrow (0, \emptyset, 0))$ assigns the state $(i, \emptyset, n_1)$.

Now we show that all the states $(i, P, j), 0 \leq i \leq n_2, 0 \leq j \leq n_1, P \subseteq \{0, \ldots, n_2{-}1\}$ are reachable by induction. First we show that any state $(i, P, n_1), 0 \leq i \leq n_2{-}1, |P| = 1$ is reachable. Note that if the second component P is not empty, namely $|P| \neq 0$, then the third component of states of C is always n_1. Consider a state $(i_1, \{i_2\}, n_1), 0 \leq i_1, i_2 \leq n_2{-}1$. For $n_2 - 1 > i_1 - i_2 \geq 0$, any state $(i_1, \{i_2\}, n_1), 0 \leq i_2 \leq i_1 \leq n_2{-}2$ is reachable from $(i_1{-}i_2, \emptyset, n_1{-}2)$ by reading a sequence of unary symbols ab^{i_2}. For the case when $i_1 - i_2 = n_2 - 1$, the state $(n_2, \{0\}, n_1)$ is reachable from $(n_2, \emptyset, n_1{-}2)$ by reading a unary symbol a. For $n_2 - 1 > i_2 - i_1 > 0$, choose a state $q = (i_2{-}i_1, \{0\}, n_1)$ which is already shown to be reachable and a tree $t = b^{i_1}(c(x, q)) \in F_{\Sigma'}[x]$. Then, the computation of C on $t(x \leftarrow (i_2{-}i_1, \emptyset, n_1))$ reaches the state $(i_1, \{i_2\}, n_1)$. Lastly, the state $(0, \{n_2{-}1\}, n_1)$ is the case when $i_2 - i_1 = n_2 - 1$. Choose a state $q = (0, \emptyset, n_1)$ and a tree $t = e(x, q)$. Then, the computation of C on $t(x \leftarrow (0, \{1\}, n_1))$ assigns the state $(0, \{n_2{-}1\}, n_1)$.

Consider a non-negative integer $z \geq 1$, and inductively assume that all the states $(i, P, n_1), 0 \leq i \leq n_2, |P| \leq z$ are reachable. Now we prove that any state $(i', P', n_1), 0 \leq i' \leq n_2, |P'| = z{+}1$ is reachable. Let $P' = \{s_1, s_2, \ldots, s_z, s_{z+1}\}$ and $P = P' \setminus \{s_{z+1}\}$. From the inductive assumption, we know that the state $q = (i', P, n_1)$ is reachable. We choose a tree $t = c(q, x) \in F_{\Sigma'}[x]$. First we consider the case when $i' < n_2 - 1$. If $i' \geq s_{z+1}$, the computation of C on $t(x \leftarrow (0, \{i'{-}s_{z+1}\}, n_1))$ reaches the state (i', P', n_1). Otherwise, we consider the computation of C on $t(x \leftarrow (0, \{n_2{+}i'{-}s_{z+1}\}, n_1))$ that assigns the state (i', P', n_1) to the root of t. Now we consider the case when $i' = n_2 - 1$, namely $(n_2 - 1, P', n_1)$. Suppose that $0 \notin P'$ and let $P'' = \{s_1 - 1, s_2 - 1, \ldots, s_z - 1, s_{z+1} - 1\}$. We choose a state $q = (n_2{-}2, P'', n_1)$ and a tree $t = f(q, x)$. Then, the computation of C on $t(x \leftarrow (1, \emptyset, n_1))$ reaches the state $(n_2{-}1, P', n_1)$. If $0 \in P'$, let $s_1 = 0$ and $P'' = \{s_2 - 1, s_3 - 1 \ldots, s_z - 1, s_{z+1} - 1\}$. We choose a state $q = (n_2{-}2, P'', n_1)$ and a tree $t = f(q, x)$. Then, the computation of C on $t(x \leftarrow (1, \{0\}, n_1))$ reaches the state (n_2-1, P', n_1). This completes the proof of the inductive step and, therefore, all states of C are reachable.

Next, we show that all states of C are inequivalent with respect to the Myhill-Nerode equivalence relation [22, 30]. Let (i_1, P_1, j_1) and (i_2, P_2, j_2) be two distinct states of C. Then, there are three possible cases to consider:

(i) Case $P_1 \neq P_2$: Without loss of generality, we assume that $p \in P_1 \setminus P_2$. The state (i_1, P_1, j_1) reaches the final state by reading a sequence of unary symbols b^{n_2-1-p} whereas the state (i_2, P_2, j_2) does not.

(ii) Case $i_1 \neq i_2$: Since $i_1 \neq i_2$, one of i_1, i_2 has to be distinct from n_2 and we assume that $0 \leq i_1 < i_2 \leq n_2$ without loss of generality. In order to establish that (i_1, P_1, j_1) and (i_2, P_2, j_2) are inequivalent, it is sufficient to give a tree $t \in F_{\Sigma'}[x]$ such that the computation of C on $t(x \leftarrow (i_1, P_1, j_1))$ (respectively, $t(x \leftarrow (i_2, P_2, j_2)))$ reaches a final state (respectively, a non-final state). We can assume that $P_1 = P_2$ since otherwise, the two states should be inequivalent from the previous case.

Let $q' = (n_2, \{0\}, n_1)$ and choose a tree $t = c(b^{n_2-i_2}(x), q')$. First, consider the computation of C on $t(x \leftarrow (i_1, P_1, j_1))$. By reading a sequence of unary symbols $b^{n_2-i_2}$, the first component i_1 reaches $n_2 - i_2 + i_1$ and by reading a binary symbol c, the state reaches the state $(n_2, \{n_2 - i_2 + i_1\}, n_1)$. Now, consider the computation on $t(x \leftarrow (i_2, P_2, j_2))$. The first component becomes the sink state of B by reading a sequence of unary symbols $b^{n_2-i_2}$. Then, the state reaches $(n_2, \emptyset, n_1)$, which is the sink state of C, by reading a binary symbol c. Now the case reduces to the previous case.

(iii) Case $j_1 \neq j_2$: Without loss of generality, we assume that $0 \leq j_1 < j_2 \leq n_1$. Since any two states $(i, P, n_1), (i, P, n_1-1), 0 \leq i \leq n_2, P \subseteq \{0, \ldots, n_2-1\}$ are equivalent, we can assume that $0 \leq j_1 < j_2 < n_1$.

First, we empty P_1, P_2 and make i_1, i_2 to be sink states by reading a sequence of unary symbols b^{n_2}. Then, the final state is reached from $(n_2, \emptyset, j_1)$ after reading a sequence of unary symbols $a^{n_1-j_1-1}b^{n_2-1}$. On the other hand, we cannot reach the final state by reading a sequence of unary symbols $a^{n_1-j_1-1}b^{n_2-1}$ from $(n_2, \emptyset, j_2)$.

Therefore, all states of C are pairwise inequivalent. □

From Lemmas 3 and 4, we establish the following result.

Theorem 5. *Let A and B be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. For $\sigma \in \Sigma_0$, $(n_2 + 1)(n_1 + 2^{n_2} - 1) - 1$ states are sufficient and necessary in the worst-case for the minimal DTA of $L(A) \cdot_\sigma^s L(B)$.*

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.

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 [12]. Piao and Salomaa [28] investigate the state complexity of parallel concatenation, 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 of prefix-free regular languages for concatenation [15].

Theorem 6. *Let A and B be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. For $\sigma \in \Sigma_0$, $(n_1 - 1)(n_2 + 1) + 2^{n_2} - 1$ states are sufficient and necessary in the worst-case for the minimal DTA of $L(A) \cdot_{\sigma}^{p} L(B)$.*

Proof. Let $A = (\Sigma, Q_A, q_{A,F}, g_A)$ and $B = (\Sigma, Q_B, q_{B,F}, g_B)$ be two subtree-free DTAs. Let $Q'_A = Q_A \sqcup \{q_{\text{sink}}\}$. We construct a new DTA $C = (\Sigma, Q_C, Q_{C,F}, g_C)$, where $Q_C = 2^{Q_B} \times Q'_A, Q_{C,F} = \{q \in Q_C \mid \pi_1(q) \cap q_{B,F} \neq \emptyset\}$. For $\tau \in \Sigma_m, m \geq 0, q_1, \ldots, q_m \in Q'_A$, we define

$$\overline{\tau_{g_A}(q_1, \ldots, q_m)} = \begin{cases} \tau_{g_A}(q_1, \ldots, q_m) & \text{if } \tau_{g_A}(q_1, \ldots, q_m) \text{ is defined,} \\ q_{\text{sink}}, & \text{otherwise.} \end{cases}$$

The transitions of g_C are defined as follows. For $\tau \in \Sigma_0$, we define

$$\tau_{g_C} = \begin{cases} (\{\tau_{g_B}, \sigma_{g_B}\}, \tau_{g_A}) & \text{if } \tau_{g_A} = q_{A,F}, \\ (\{\tau_{g_B}\}, \overline{\tau_{g_A}}) & \text{if } \tau_{g_A} \neq q_{A,F}, \text{ and at least one of } \tau_{g_A} \text{ and } \tau_{g_B} \text{ is defined,} \\ \text{undefined,} & \text{otherwise.} \end{cases}$$

For $\tau \in \Sigma_m$ and $(P_i, q_i) \in Q_C$, where $m \geq 1$ and $1 \leq i \leq m$, we define

$$\tau_{g_C}((P_1, q_1), \ldots, (P_m, q_m)) = (\tau_{g_B}(P_1, \ldots, P_m) \cup X, \overline{\tau_{g_A}(q_1, \ldots, q_m)}),$$

where $X = \{\sigma_{g_B}\}$ if $\tau_{g_A}(q_1, \ldots, q_m) = q_{A,F}$ and $X = \emptyset$ otherwise.

Now consider the computation of the new DTA C. The first component of a state in Q_C simulates B assuming that every leaf node labeled by σ is substituted with a tree in $L(A)$. The second component of a state in Q_C simulates A and adds σ_{g_B} into the first component when it becomes the final state of A. Therefore, the construction guarantees that C simulates the parallel concatenation of A and B. Next we compute the upper bound by counting the number of sufficient states in C. Since each state of C is a pair of a set of states of B and a state of A, the total number of states is $(n_1 + 1) \cdot 2^{n_2}$.

First, any two states $(P, q_{A,F}), (P, q_{\text{sink}}), P \subseteq Q_B$, can be merged into one state since they are equivalent. Therefore, we can subtract 2^{n_2} states. By the construction, the second component always becomes the sink state q_{sink} when we add a state into the first component. Thus all the states $(P, q), q \in Q_A, P \subseteq Q_B$ are unreachable if $q \neq q_{\text{sink}}$ and $|P| > 1$. Furthermore, we remove the sink state $(\emptyset, q_{\text{sink}})$ because we consider the incomplete state complexity. Therefore, the upper bound is

$$(n_1+1)\cdot 2^{n_2} - 2^{n_2} - (n_1-1)(2^{n_2}-n_2-1) - 1 = (n_1-1)(n_2+1) + 2^{n_2} - 1.$$

For the tight bound, we present two subtree-free DTAs A and B whose state complexity of $L(A) \cdot_\sigma^p L(B)$ reaches the upper bound. We choose $\Sigma = \Sigma_0 \sqcup \Sigma_1 \sqcup \Sigma_2$, where $\Sigma_0 = \{d\}, \Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{c\}$. 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:

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

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:

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

Note that both the final states of A and B do not have any out-transitions and, thus, $L(A)$ and $L(B)$ are subtree-free regular tree languages by Lemma 2. We show that the upper bound is reached using A and B. 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 3. Note that here a symbol d is used to concatenate two trees.

We show that all states of C are reachable. Note that n_1 and n_2 are sink states of A and B, respectively. Before we consider the reachability of states, we remind that any two states $(i, n_1{-}1)$ and (i, n_1) are equivalent since there is no transition defined for the final state $n_1{-}1$. The computation of C assigns the state $(\{0\}, 0)$ to a leaf labeled by d. The state $(\emptyset, j), 0 \leq j \leq n_1 - 1$ can be reached by reading a sequence of unary symbols $a^j b^{n_2}$ from $(\{0\}, 0)$.

Now we show that all the states $(P, j), P \subseteq Q_B, 0 \leq j \leq n_1$ are reachable by induction. From $(\{0\}, 0)$, we reach the state $(\{i\}, j)$, $0 \leq i \leq n_2 - 1, 0 \leq j \leq n_1$ by reading a sequence of unary symbols $a^j b^i$.

Consider a non-negative integer $z \geq 1$, and inductively assume that all the states $(P, n_1), |P| \leq z, 0 \leq j \leq n_1$ are reachable. Now we prove that any state $(P', n_1), |P'| = z+1$ is reachable. Let $P' = \{s_1, s_2, \ldots, s_z, s_{z+1}\}, s_1 > s_2 > \cdots > s_z > s_{z+1}$ and $P = \{s_1 - s_{z+1}, s_2 - s_{z+1}, \ldots, s_z - s_{z+1}\}$. From the inductive assumption, we know that the state $q = (P, n_1)$ is reachable. We choose $t = b^{s_{z+1}}(c(x, q)) \in F_{\Sigma'}[x]$. Then, the computation of C on $t(x \leftarrow (\{0, s_z - s_{z+1}\}, n_1))$ reaches the state (P', n_1). Note that the state $(\{0, s_z - s_{z+1}\}, n_1)$ is reachable from $(\{s_z - s_{z+1}\}, n_1 - 2)$ by reading a unary symbol a. This completes the proof of the inductive step and, therefore, all states of C are reachable.

Next, we show that all states of C are pairwise inequivalent. Let (P_1, j_1) and (P_2, j_2) be two distinct states of C. There are two possible cases to consider:

(i) Case $P_1 \neq P_2$: Without loss of generality, we assume that $p \in P_1 - P_2$. The state (P_1, j_1) reaches the final state by reading a sequence of unary symbols b^{n_2-1-p} whereas the other state (P_2, j_2) does not.

(ii) Case $j_1 \neq j_2$: Without loss of generality, we assume that $0 \leq j_1 < j_2 \leq n_1$. Note that any two states $(P, n_1 - 1), (P, n_1), P \subseteq Q_B$ are equivalent. Therefore, we assume that $0 \leq j_1 < j_2 < n_1$.

First, we empty P_1 and P_2 by reading a sequence of unary symbols b^{n_2}. Then, we reach two distinct states $(\{0\}, n_1 - 1)$ and $(\emptyset, n_1 - j_2 + j_1 - 1)$ by reading a sequence of unary symbols $a^{n_1-j_2-1}$. Since the first component of two states are different, this case reduces to the first case.

Therefore, all states of C are pairwise inequivalent. □

5. State Complexity of Kleene-Star

Piao and Salomaa [27] 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 [27].

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

Proof. Let $Q' = Q \sqcup \{q_{\text{sink}}\}$. We construct a new DTA $B = (\Sigma, Q_B, Q_{B,F}, g_B)$ recognizing $L(A)^{b,*}_{\sigma}$, where $Q_B = Q' \times Q' \sqcup \{q_{\text{new}}\}$ and $Q_{B,F} = \{(q_F, q_{\text{sink}}), (\sigma_g, q_F), q_{\text{new}}\}$. 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 $(q_{\text{sink}}, \sigma_g)$ except that q_{new} is a final state and $(q_{\text{sink}}, \sigma_g)$ is not necessarily a final state. We assume that σ_g is defined without loss of generality because otherwise, $L(A)^{b,*}_{\tau} = L(A)^{b,0}_{\tau} \cup L(A)^{b,1}_{\tau} = \{\sigma\} \cup L(A)$. For $\tau \in \Sigma_0 \setminus \{\sigma\}$, we define

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

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) $(q_{\text{sink}}, \tau_g(q_1, \ldots, q_m))$ 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 $\tau_g(q_1, \ldots, q_{i-1}, p_i, q_{i+1}, \ldots, q_m) \neq q_F$ and $p_j = q_{\text{sink}}$ for all $j, j \neq i$.
- σ_g if $\tau_g(q_1, \ldots, q_{i-1}, p_i, q_{i+1}, \ldots, q_m) = q_F$ and $p_j = q_{\text{sink}}$ for all $j, j \neq i$.

(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 q_{new} is one of the final states of B since a single node tree labeled by σ is in $L(B)$. Moreover, the DTA B accepts trees if either the first component or the second component is the final state of A. The first component enters the final state of A if and only if the input tree is a tree of $L(A)$ where at least a leaf of the tree has been replaced by a tree in $L(A)_\sigma^{b,k}$ for $k \geq 0$. The second component enters the final state of A if the input tree is in $L(A)$. Since $L(A) \subseteq L(A)_\sigma^{b,*}$, B accepts $L(A)_\sigma^{b,*}$.

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 component. 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). This implies that there are $n^2 - 1$ unreachable states. We also note that a state (q_{sink}, q_F) is unreachable since whenever we have a state q_F in the second component, the first component should be σ_g by the construction. Moreover, we merge two final states (q_F, q_{sink}) and (σ_g, q_F) into a single final state while maintaining the transitions of (σ_g, q_F). Furthermore, we remove one more state $(q_{\text{sink}}, q_{\text{sink}})$, which is a sink state. Therefore, we know that

$$(n+1)^2 + 1 - (n^2 - 1) - 1 - 1 - 1 = 2n$$

states are sufficient for $L(A)_\sigma^{b,*}$. □

Next we present a lower bound example whose state complexity reaches the upper bound in Lemma 7. Let $\Sigma = \Sigma_0 \sqcup \Sigma_1 \sqcup \Sigma_2$, where $\Sigma_0 = \{d\}, \Sigma_1 = \{a, b\}$ and $\Sigma_2 = \{c\}$. We define a subtree-free DTA $A = (\Sigma, Q, q_F, g)$, where $Q = \{0, 1, \ldots, n-1\}, q_F = n - 1$ and the transition function g is defined as follows:

- $d_g = 0$,
- $a_g(i) = i + 1, 0 \leq i \leq n - 3, a_g(n-2) = 0$,
- $b_g(n-2) = n - 1$,
- $c_g(n-2, n-2) = n - 1$.

All transitions of g not defined above are undefined. Using A and the upper bound construction of the proof of Lemma 7, we construct a new DTA $B = (\Sigma, Q_B, Q_{B,F}, g_B)$ accepting $L(A)_d^{b,*}$, namely $L(A)_d^{b,*} = L(B)$. Now we prove that the DTA B is the minimal DTA by showing all states of B are reachable and inequivalent.

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

Proof. First we show that all states of B are reachable. By the construction, we can reach q_{new} after reading a leaf labeled d. Since the transitions of q_{new} are defined to be equal to those of $(q_{\text{sink}}, d_g) = (n, 0)$, the state (n, k) is reached by reading a sequence of unary symbols a^k for $1 \leq k \leq n-2$. We can also reach $(n, 0)$ by reading a unary symbol a from $(n, n-2)$. The state $(0, n-1)$ is reachable from $(n, n-2)$ by reading a unary symbol b. From $(0, n-1)$, we can reach (k, n) by reading a sequence of unary symbols a^k for $1 \leq k \leq n-2$. We also reach $(0, n)$ by reading one more unary symbol a from $(n-2, n)$. Lastly, the state $(n-1, n)$ can be reached by reading a unary symbol b from $(n-2, n)$.

Next we show that all states of B are pairwise inequivalent by showing that any two distinct states (i_1, j_1) and (i_2, j_2) in Q_B are not equivalent with respect to the Myhill-Nerode equivalence relation [22, 30]. There are three cases to consider:

(i) Case $i_1 = i_2 = n$ and $j_1 \neq j_2$ for $0 \leq j_1, j_2 \leq n-2$: After reading a sequence of unary symbols a^{n-2-j_1} and a unary symbol b, we reach $(0, n-1)$ from (n, j_1). Then, we reach the final state $(n-1, n)$ by reading a sequence of unary symbols a^{n-2} and one unary symbol b. On the other hand, (n, j_2) does not reach the final state by reading the same sequence of symbols.

(ii) Case $j_1 = j_2 = n$ and $i_1 \neq i_2$ for $0 \leq i_1, i_2 \leq n-2$: We should read a sequence of unary symbols a^{n-2-i_1} and one unary symbol b to arrive at the final state from (i_1, n). However, (i_2, n) cannot reach the final state by reading the same sequence. Now we show that the states of (ii) are inequivalent to the states of (i). We denote $q' = (n-2, n)$ and choose a tree $t = c(a^{n-k-2}(x), q') \in F_{\Sigma'}[x]$. Consider the computation of B on $t(x \leftarrow (k, n))$ for $0 \leq k \leq n-2$. After the computation, we arrive at the state (n, n), which is a sink state. On the other hand, the computation of B on $t(x \leftarrow (n, k))$ for $0 \leq k \leq n-2$ arrives at the state $(n-1, n)$, which is a final state. Thus, the states of (i) and the states of (ii) are inequivalent.

(iii) Case $(n-1, n)$ and q_{new}: Note that these states are final, thus, they are inequivalent to the other states considered in (i) and (ii). We also notice that the state $(n-1, n)$ has the transitions of $(0, n-1)$ based on the construction. By the construction of B, the out-transitions from q_{new} are the same as the out-transitions of $(n, 0)$. We denote $q' = (n-2, n)$ and choose $t = b_2(a^{n-2}(x), q') \in F_{\Sigma'}[x]$. Then, the computation of B on $t(x \leftarrow (n, 0))$ reaches the final state $(n-1, n)$ while $t(x \leftarrow (n-1, n))$ goes to the sink state (n, n). Thus, $(n-1, n)$ and q_{new} are inequivalent.

Thus, all states are reachable and inequivalent. □

Lemmas 7 and 8 show that $2n$ is the tight bound for the bottom-up Kleene-star operation.

Theorem 9. *Let A be a subtree-free minimal DTA with $n \geq 2$ states. For $\sigma \in \Sigma_0$, $2n$ 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 [27]. 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 10. *Let $A = (\Sigma, Q, q_F, g)$ be a subtree-free minimal DTA with $n \geq 2$ states. 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 [15]. We define $B = (\Sigma, Q_B, Q_{B,F}, g_B)$, where $Q_B = Q \sqcup \{q_{\text{new}}\}$ and $Q_{B,F} = \{q_{\text{new}}, q_F\}$. As in the proof of Lemma 7, 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. Note that $L(B) = L(A)_\sigma^{t,*}$ since B simulates σ_g, which is the state reachable by reading a leaf node labeled by σ whenever A arrives at the final state. By the construction, the sufficient number of states for $L(A)_\sigma^{t,*}$ is n.

We show that n states are necessary for recognizing $L(A)_\sigma^{t,*}$ by a simple lower bound example. For a unary tree, where the leaf is an element of Σ_0, $t = z_1(z_2(\ldots z_m(d)\ldots))$, we define $\texttt{word}(t) = z_m z_{m-1} \ldots z_1$. Note that $\texttt{word}(t)$ is the sequence of symbols labeling the nodes of t except the leaf symbol in bottom-up direction. Let L be the following unary tree language:

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

Note that the DTA A needs at least n states for recognizing L. We construct a DTA B for recognizing $L(A)^{t,*}_{\sigma}$ according to the upper bound construction. Then, B accepts $L' = \{t \mid \texttt{word}(t) = (a^{n-1})^*\}$ and it is easy to verify that n states are necessary for accepting L'. Therefore, n is a tight bound for the minimal DTA of $L(A)^{t,*}_{\sigma}$. □

6. Intersection and Union

For regular languages, the state complexities of intersection and union are trivial. The upper bound construction is based on the Cartesian product of states and yields n_1n_2 states. Yu *et al.* [36] showed that n_1n_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 $n_1n_2+n_1+n_2$. 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 [15]. 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.

Theorem 11. *Let A and B be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. Then, $n_1n_2 - n_1 - n_2 + 2$ states are sufficient and necessary in the worst-case for the minimal DTA of $L(A) \cap L(B)$.*

Proof. Let $A = (\Sigma, Q_A, q_{A,F}, g_A)$ and $B = (\Sigma, Q_B, q_{B,F}, g_B)$ be two subtree-free DTAs. We construct a new DTA $C = (\Sigma, Q_C, Q_{C,F}, g_C)$, where $Q_C = Q_A \times Q_B, Q_{C,F} = \{q \in Q_C \mid \pi_1(q) = q_{A,F}$ and $\pi_2(q) = q_{B,F}\}$, and g_C is defined as follows. For $\tau \in \Sigma_0$, we define $\tau_{g_C} = \tau_{g_A} \times \tau_{g_B}$. For $\tau \in \Sigma_m$ and $(p_i, q_i) \in Q_C$, where $m \geq 1$ and $1 \leq i \leq m$, we define $\tau_{g_C}((p_1, q_1), \ldots, (p_m, q_m))$ to be

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

Note that C has n_1n_2 states. We assume that a state in Q_C contains $q_{A,F}$ or $q_{B,F}$, which is the final state of A or B, respectively. Since A and B have no out-transitions defined for their final states, there are no transitions defined for the corresponding states in C, either. Note that the number of states containing $q_{A,F}$ or $q_{B,F}$ is $n_1 + n_2 - 1$. Among these states, $(q_{A,F}, q_{B,F})$ is the final state while the others are non-final. We remove n_1+n_2-2 non-final states since they are all the sink states. Then, the sufficient number of states for intersection is $n_1n_2 - n_1 - n_2 + 2$.

For the lower bound, we choose $\Sigma = \Sigma_0 \sqcup \Sigma_1$, where $\Sigma_0 = \{d\}$ and $\Sigma_1 = \{a, b, c\}$. Let L_1 and L_2 be subtree-free unary tree languages as follows:

$$L_1 = \{t_1 \mid \texttt{word}(t_1) = wc, 0 \equiv |w|_a \pmod{n_1 - 1},\ \text{and } w \in \{a, b\}^*\},$$
$$L_2 = \{t_2 \mid \texttt{word}(t_2) = wc, 0 \equiv |w|_a \pmod{n_2 - 1},\ \text{and } w \in \{a, b\}^*\}.$$

Then the two DTAs A and B need at least n_1 and n_2 states for recognizing L_1 and L_2, respectively. Let C be a new DTA recognizing $L_1 \cap L_2$. and assume that $n_1 - 1$ and $n_2 - 1$ are relatively prime. Then,

$$L_1 \cap L_2 = \{t \mid \texttt{word}(t) = wc, 0 \equiv |w|_a \pmod{(n_1-1)(n_2-1)}, \text{ and } w \in \{a,b\}^*\}$$

and thus, requires at least $(n_1 - 1)(n_2 - 1) + 1 = n_1n_2 - n_1 - n_2 + 2$ states. It is easy to verify that all states are reachable and inequivalent. □

Next, we examine the state complexity of union.

Theorem 12. *Let A and B be subtree-free minimal DTAs with n_1 and n_2 states, respectively, where $n_1, n_2 \geq 2$. Then, $n_1n_2 + n_1 + n_2 - 2$ states are sufficient and necessary in the worst-case for the minimal DTA of $L(A) \cup L(B)$.*

Proof. Let $A = (\Sigma, Q_A, q_{A,F}, g_A)$ and $B = (\Sigma, Q_B, q_{B,F}, g_B)$ be two subtree-free DTAs. Let $Q'_A = Q_A \sqcup \{q_{\text{sink}}\}$ and $Q'_B = Q_B \sqcup \{q_{\text{sink}}\}$. We construct a new DTA $C = (\Sigma, Q_C, Q_{C,F}, g_C)$, where $Q_C = Q'_A \times Q'_B, Q_{C,F} = \{q \in Q_C \mid \pi_1(q) = q_{A,F}$ or $\pi_2(q) = q_{B,F}\}$, and g_C is defined as follows. For $\tau \in \Sigma_0$, we define $\tau_{g_C} = \tau_{g_A} \times \tau_{g_B}$. For $\tau \in \Sigma_m$ and $(p_i, q_i) \in Q_C$, where $m \geq 1$ and $1 \leq i \leq m$, we define $\tau_{g_C}((p_1, q_1), \ldots, (p_m, q_m))$ to be

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

Note that we have $(n_1 + 1)(n_2 + 1)$ states. First, we remove the sink state $(q_{\text{sink}}, q_{\text{sink}})$ and merge three final states $(q_{\text{sink}}, q_{B,F}), (q_{A,F}, q_{\text{sink}})$ and $(q_{A,F}, q_{B,F})$ into one final state since they are all equivalent. Thus, the number of states for C is $n_1n_2 + n_1 + n_2 - 2$.

For the lower bound, we choose $\Sigma = \Sigma_0 \sqcup \Sigma_1$, where $\Sigma_0 = \{d\}$ 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 C be a new DTA recognizing $L_1 \cup L_2$. Then, C 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 $n_2 - 2$, C requires $(n_1 - 1)(n_2 - 1)$ states. Assume that C reads $(n_1 - 1)$'th a, then C should be in one of $n_2 - 1$ final states depending on the number of b's that we have read. Similarly, C reaches $n_2 - 1$ non-final states by reading more a's from the final states. Analogously, C 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 t such that $\texttt{word}(t) = a^{n_1-1}b^{n_2-1}$. It is easy to verify that all states of C are inequivalent. Therefore, $n_1n_2 + n_1 + n_2 - 2$ states are necessary for union. $\square$

7. Conclusions

People rely on regular tree languages for defining a formal framework for XML schema languages [21, 23] and recently research revisited regular tree automata both from the practical side [16, 34] and from the theoretical side [25, 26].

We notice that regular tree languages in practice are often subtree-free, which is very similar to prefix-freeness in string languages. For instance, a set of XML Schemas is subtree-free since all XML Schemas should start with the unique root element `xs:schema` by the XML Schema specification [8]. Based on this observation, we have defined a set of trees T to be subtree-free if a tree $t \in T$ is not a subtree of any other trees in T. Then, we have introduced a family of subtree-free regular tree languages, which is a proper subfamily of regular tree languages and identified the structural properties of the family. We have investigated the state complexity of subtree-free regular tree languages and established the precise state complexity.

Table 1. Comparison between the state complexity of basic operations for subtree-free and general regular tree languages.

operations	subtree-free	general [27, 28]
$L_1 \cdot_\sigma^s L_2$	$(n+1)(m+2^n-1)-1$	$(n+1)(m \cdot 2^n + 2^{n-1})-1$
$L_1 \cdot_\sigma^p L_2$	$(m-1)(n+1)+2^n-1$	$m \cdot 2^n + 2^{n-1} - 1$
$L_\sigma^{b,*}$	$2n$	$(n+1) \cdot 2^{n-1} + 2^{n-2}$
$L_\sigma^{t,*}$	n	$2^{n-1} + 2^{n-2}$
$L_1 \cap L_2$	$mn - m - n + 2$	$mn + m + n$
$L_1 \cup L_2$	$mn + m + n - 2$	$mn + m + n$

Table 2. Comparison between the state complexity of basic operations for subtree-free regular tree languages and prefix-free regular string languages.

operations	subtree-free	prefix-free (string) [15]
$L_1 \cdot_\sigma^s L_2$	$(n+1)(m+2^n-1)-1$	$m+n-2$
$L_1 \cdot_\sigma^p L_2$	$(m-1)(n+1)+2^n-1$	
$L_\sigma^{b,*}$	$2n$	n
$L_\sigma^{t,*}$	n	
$L_1 \cap L_2$	$mn - m - n + 2$	$mn - 2(m+n) + 6$
$L_1 \cup L_2$	$mn + m + n - 2$	$mn - 2$

We have summarized the tight bounds and compared with that of general regular tree languages in Table. 1. We have also compared the state complexity of subtree-free regular tree languages with the state complexity of prefix-free regular string languages in Table. 2.

Acknowledgments

We wish to thank the referees for the care they put into reading the previous version of this manuscript. Their comments were invaluable in depth and detail, and the current version owes much to their efforts.

This research was supported by the Basic Science Research Program through NRF funded by MEST (2015R1D1A1A01060097) and the Yonsei University Future-leading Research Initiative of 2014.

References

[1] J. Berstel and D. Perrin, *Theory of Codes* (Academic Press, Inc., 1985).

[2] C. Câmpeanu, K. Culik II, K. Salomaa and S. Yu, State complexity of basic operations on finite languages. In *Proceedings of the 4th International Workshop on Implementing Automata*, 60–70, 2001.

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

[4] M. Domaratzki and A. Okhotin, State complexity of power, *Theoretical Computer Science*, 410(24-25):2377–2392, 2009.

[5] H.-S. Eom and Y.-S. Han, State complexity of combined operations for suffix-free regular languages, *Theoretical Computer Science*, 510:87–93, 2013.

[6] H.-S. Eom, Y.-S. Han and K. Salomaa, State complexity of k-union and k-intersection for prefix-free regular languages, *International Journal of Foundations of Computer Science*, 26(2):211–227, 2015.

[7] Z. Ésik, Y. Gao, G. Liu and S. Yu, Estimation of state complexity of combined operations, *Theoretical Computer Science*, 410(35):3272–3280, 2009.

[8] S. Gao, C. M. Sperberg-McQueen and H. S. Thompson, W3C XML schema definition language (XSD) 1.1 part 1: Structures, Technical report, W3C, 2012. http://www.w3.org/TR/xmlschema11-1.

[9] Y. Gao and L. Kari, State complexity of star of union and square of union on k regular languages, *Theoretical Computer Science*, 499:38–50, 2013.

[10] Y. Gao, L. Kari and S. Yu, State complexity of union and intersection of square and reversal on k regular languages, *Theoretical Computer Science*, 454:164–171, 2012.

[11] Y. Gao, K. Salomaa and S. Yu, The state complexity of two combined operations: Star of catenation and star of reversal, *Fundamenta Informaticae*, 83(1-2):75–89, 2008.

[12] F. Gécseg and M. Steinby, Tree languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages, Vol. 3: Beyond Words*, 1–68 (Springer-Verlag New York, Inc., 1997).

[13] Y.-S. Han and K. Salomaa, State complexity of union and intersection of finite languages, *International Journal of Foundations of Computer Science*, 19(3):581–595, 2008.

[14] Y.-S. Han and K. Salomaa, State complexity of basic operations on suffix-free regular languages, *Theoretical Computer Science*, 410(27-29):2537–2548, 2009.

[15] Y.-S. Han, K. Salomaa and D. Wood, Operational state complexity of prefix-free regular languages. In *Automata, Formal Languages, and Related Topics - Dedicated to Ferenc Gécseg on the Occasion of His 70th Birthday*, 99–115, 2009.

[16] H. Hosoya, *Foundations of XML Processing: The Tree-Automata Approach* (Cambridge University Press, New York, NY, USA, 1st edition, 2010).

[17] M. Hricko, G. Jirásková and A. Szabari. Union and intersection of regular languages and descriptional complexity. In *Proceedings of the* 7*th International Workshop on Descriptional Complexity of Formal Systems*, 170–181, 2005.

[18] J. Jirásek, G. Jirásková and A. Szabari, State complexity of concatenation and complementation, *International Journal of Foundations of Computer Science*, 16(3):511–529, 2005.

[19] E. Maler, J. Paoli, C. M. Sperberg-McQueen, F. Yergeau and T. Bray, Extensible markup language (XML) 1.0 (fifth edition), Technical report, W3C, 2008. http://www.w3.org/TR/2008/REC-xml-20081126.

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

[21] M. Murata, D. Lee, M. Mani and K. Kawaguchi, Taxonomy of XML schema languages using formal language theory, *ACM Transactions on Internet Technology*, 5(4):660–704, 2005.

[22] A. Nerode, Linear automaton transformations. In *Proceedings of the American Mathematical Society*, 9(4):541–544, 1958.

[23] F. Neven, Automata theory for XML researchers, *ACM SIGMOD Record*, 31(3):39–46, 2002.

[24] X. Piao, *State Complexity of Tree Automata*, PhD thesis, Queen's University, 2012.

[25] X. Piao and K. Salomaa, State trade-offs in unranked tree automata. In *Proceedings of the* 13*th International Workshop on Descriptional Complexity of Formal Systems*, 261–274, 2011.

[26] X. Piao and K. Salomaa, Transformations between different models of unranked bottom-up tree automata, *Fundamenta Informaticae*, 109(4):405–424, 2011.

[27] X. Piao and K. Salomaa, State complexity of Kleene-star operations on trees. In *Proceedings of the* 2012 *International Conference on Theoretical Computer Science: Computation, Physics and Beyond*, 388–402, 2012.

[28] X. Piao and K. Salomaa, State complexity of the concatenation of regular tree languages, *Theoretical Computer Science*, 429:273–281, 2012.

[29] G. Pighizzini and J. Shallit, Unary language operations, state complexity and Jacobsthal's function, *International Journal of Foundations of Computer Science*, 13(1):145–159, 2002.

[30] M. O. Rabin and D. Scott, Finite automata and their decision problems, *IBM Journal of Research and Development*, 3(2):114–125, 1959.

[31] N. Rampersad, The state complexity of L^2 and L^k, *Information Processing Letters*, 98(6):231–234, 2006.

[32] A. Salomaa, K. Salomaa and S. Yu, State complexity of combined operations, *Theoretical Computer Science*, 383(2-3):140–152, 2007.

[33] K. Salomaa and S. Yu, On the state complexity of combined operations and their estimation, *International Journal of Foundations of Computer Science*, 18:683–698, 2007.

[34] T. Schwentick, Foundations of XML based on logic and automata: A snapshot. In *Proceedings of the* 7*th International Symposium on Foundations of Information and Knowledge Systems*, 23–33. 2012.

[35] S. Yu, State complexity of regular languages, *Journal of Automata, Languages and Combinatorics*, 6(2):221–234, 2001.

[36] S. Yu, Q. Zhuang and K. Salomaa, The state complexities of some basic operations on regular languages, *Theoretical Computer Science*, 125(2):315–328, 1994.