Conversion of Finite Tree Automata to Regular Tree Expressions By State Elimination

Tomáš Pecka*, Jan Trávníček, and Jan Janoušek**

Department of Theoretical Computer Science
Faculty of Information Technology
Czech Technical University in Prague
{tomas.pecka,jan.travnicek,jan.janousek}@fit.cvut.cz

Abstract. Regular tree languages can be accepted and described by finite tree automata and regular tree expressions, respectively. We describe a new algorithm that converts a finite tree automaton to an equivalent regular tree expression. Our algorithm is analogous to the well-known state elimination method of the conversion of a string finite automaton to an equivalent string regular expression. We define a generalised finite tree automaton, the transitions of which read the sets of trees described by regular tree expressions. Our algorithm eliminates states of the generalised finite tree automaton, which is analogous to the elimination of states in converting the string finite automaton.

Keywords: regular tree languages, finite tree automata, regular tree expressions, state elimination method

1 Introduction

The theory of formal tree languages is an important part of computer science and has been extensively studied and developed since 1960s [5,6]. Trees are natural data structures for storing hierarchical data. Their applications range from areas such as natural language processing, interpretation of nonprocedural languages and code generation to processing markup languages such as XML.

Regular expressions are well-studied structures representing regular (string) languages in finite space [8,2]. The concept of expressions can be extended to regular tree languages as well. Regular tree expressions (RTEs) denote regular tree languages [5].

Standard computation models for problems on trees are various kinds of tree automata. An finite tree automaton (FTA) is an acceptor for the class of regular tree languages. The Kleene's theorem for tree languages states that the class of regular tree languages is equal to the class of languages that can be described by the RTEs [5]. Both formalisms are therefore equally powerful, but sometimes one of them is more convenient than the other one. For instance, language membership problem is easily solvable using the automaton. The expressions might be better in describing the language. Therefore it is suitable to find a way of converting one to another as we do in the area of strings.

The conversion of an RTE into an FTA was studied by several works. Algorithms presented in these papers are adaptations of well-known algorithms for the conversion of regular (string) expressions into finite (string) automata [9,10,3]. Note that there

* This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS20/208/OHK3/3T/18.

** The authors acknowledge the support of the OP VVV MEYS funded project CZ.02.1.01/0.0/0.0/16_019/0000765 "Research Center for Informatics".

Tomáš Pecka, Jan Trávníček, Jan Janoušek: Conversion of Finite Tree Automata to Regular Tree Expressions By State Elimination, pp. 11–22.
Proceedings of PSC 2020, Jan Holub and Jan Žďárek (Eds.), ISBN 978-80-01-06749-9 © Czech Technical University in Prague, Czech Republic

also exist conversion algorithms for the problem of converting RTEs into pushdown (string) automata that accept trees in their linearised form [13,12].

The possibility of conversion in the other way, i.e. the conversion of an FTA to an RTE was introduced in the proof of the Kleene's theorem in [5]. The proof proposes a State elimination-like of conversion but it was not put into the algorithm and it was not discussed much. Recently, Guellouma and Cherroun studied regular tree equations [7] which they used to construct a regular tree equation system from an FTA. Solving this equation system yields an RTE describing the same language that was accepted by the original FTA. Unfortunately, the time complexity of the algorithm is not stated in the article.

In this paper, we build upon the idea of eliminating states from an FTA presented in the proof of Kleene's Theorem [5]. We present a practical algorithm for the problem of converting FTAs to RTEs that is inspired by the classical State elimination algorithm for finite (string) automata [8]. We define the notion of generalized finite tree automaton (GFTA) which differs from the FTA mainly in the transition function. Instead of the input alphabet symbols, the transitions now involve sets of trees described by the RTE. We use this model for the process of eliminating states. States of the GFTA are then eliminated one-by-one and the transitions of the automaton are modified in such way that the language the automaton accepts does not change. The transitions to the last remaining final state then define the equivalent RTE to the original automaton. Such approach can convert an FTA to an equivalent RTE in $O(|Q|^2 \cdot (|\Delta| + |Q_F|)$ time where Q and Q_F are the sets of all states and final states of the FTA, respectively, and Δ is the set of transitions of the FTA. Furthermore, the implementation of the algorithm is really simple and straightforward.

The following parts of this paper are organised as follows: Section 2 recalls basic definitions and notations. The new conversion algorithm yielding the RTE is presented in Section 3. Finally, the achieved results and ideas for future work are presented in the concluding section.

2 Background

A *ranked alphabet* Σ is a finite nonempty set of symbols, each of which is assigned with non-negative integer *arity* denoted by $\mathrm{arity}(a)$. The set Σ_n denotes the set of symbols from Σ with arity n. Elements of arity $0, 1, 2, \ldots, n$ are called nullary (also constants), unary, binary, …, n-ary symbols, respectively. We assume that Σ contains at least one constant. We use numbers at the end of symbols for a short declaration of arity. For instance, $a2$ is a short declaration of a binary symbol a.

A labelled, ordered and ranked tree over a ranked alphabet Σ is defined on the concepts from graph theory [2]. A *directed ordered graph* G is a pair (N, R) where N is a set of nodes and R is a list of ordered pairs of edges. Elements of R are in the form $((f, g_1), (f, g_2), \ldots, (f, g_n))$, where $f, g_1, g_2, \ldots, g_n \in N$, $n \geq 0$. Such element denotes n edges leaving f with the first edge entering node g_1, the second entering g_2, and so forth. A sequence of nodes $(f_0, f_1, \ldots, f_n)$, $n \geq 1$ is a *path* of length n from node f_0 to f_n if there is an edge from f_i to f_{i+1} for each $0 \leq i < n$. A *cycle* is a path where $f_0 = f_n$. An *in-degree* of a node is a number of incoming edges. An *out-degree* is a number of outgoing edges. A node with out-degree 0 is called a *leaf.*

An ordered *directed acyclic graph (DAG)* is an ordered directed graph with no cycle. A *rooted DAG* is a DAG with a special node $r \in N$ called the *root.* The in-degree of r is 0, in-degree of every other node is 1 and there is just one path from the

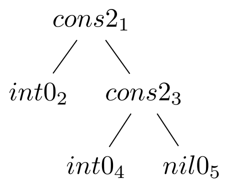


Figure 1. A directed, rooted, labelled, ranked, and ordered tree over $\Sigma = \{nil0, int0, cons2\}$.

root r to every $f \in N$, $f \neq r$. A *labelled ranked DAG* is a DAG where every node is labelled by a symbol $a \in \Sigma$ and the out-degree of a node $a \in \Sigma$ equals to arity(a). A *directed, ordered, rooted, labelled, and ranked tree* is rooted, labelled, and ranked DAG. All trees in this paper are considered to be directed, ordered, rooted, labelled, and ranked.

Due to simplicity we often represent trees in the well-known prefix notation (see Example 1).

Example 1. Let t from Figure 1 be a directed, rooted, labelled, ranked, and ordered tree with labels from ranked alphabet $\Sigma = \{nil0, int0, cons2\}$. Formally the tree is a graph $t = (\{cons2_1, int0_2, cons2_3, int0_4, nil0_5\}, \{(cons2_1, int0_2), (cons2_1, cons2_3), (cons2_3, int0_4), (cons2_3, nil0_5)\})$. The root of t is a $cons2_1$ node with an ordered pair of children $(int0_2, cons2_3)$. The prefix notation of t is $cons2(int0, cons2(int0, nil0))$. We omit the subscripted indexes and show the labels from Σ only.

A nondeterministic *bottom-up finite tree automaton (FTA)* over a ranked alphabet Σ is a 4−tuple $\mathcal{A} = (Q, \Sigma, Q_F, \Delta)$, where Q is a finite set of states, $Q_F \subseteq Q$ is a set of final states, and Δ is a mapping $\Sigma_n \times Q^n \mapsto \mathcal{P}(Q)$ (where $\mathcal{P}$ denotes the powerset function), i.e., the transitions are in the form $f(q_1, q_2, \ldots, q_n) \to q$ where $f \in \Sigma_n$, $n \geq 0$, and $q, q_1, q_2, \ldots, q_n \in Q$. An FTA is a *deterministic FTA* if for every left hand side of the transition there is at most one target state.

The computation of the FTA starts at leaves and moves towards the root. Each subtree is mapped to a state. A *run* of an automaton is defined inductively: The leaves are mapped to states q by the transitions of the form $a \to q \in \Delta$, $a \in \Sigma_0$. If a root of a subtree is labelled with $f \in \Sigma_n$, $n \geq 1$, and its children are mapped to states $q_1, \ldots, q_n$, this subtree is mapped to q, where $f(q_1, q_2, \ldots, q_n) \to q \in \Delta$. *Language of a state* q (denoted by $L(q)$) is a set of subtrees mapped to state q. Obviously, an FTA *accepts* such trees that have their roots mapped to any final state, i.e., $L(\mathcal{A}) = \bigcup_{q \in Q_F} L(q)$.

The tree language $L(\mathcal{A})$ *recognised* by an FTA $\mathcal{A}$ is the set of trees accepted by the FTA $\mathcal{A}$. A tree language is *recognisable* if it is recognised by some FTA. It is recognisable if and only if it is a regular tree language (see [4,5] for the definition). Every nondeterministic FTA can be transformed to an equivalent deterministic FTA [5].

The transition function of FTA can be depicted using a diagram in a similar way as the diagram of a finite automaton. However, transitions of an FTA can have an arbitrary amount of source states. Therefore a *join node* of source states is added into the diagram. The order of source states is specified by a number on edges from states to join nodes.

Example 2. An example of an FTA is $\mathcal{A} = (\{I, L\}, \{int0, nil0, cons2\}, \{L\}, \Delta)$, where Δ consists of the following transitions: $int0 \to I$, $nil0 \to L$, and $cons2(I, L) \to L$. $\mathcal{A}$

accepts the language $L(\mathcal{A}) = \{nil, cons(int, nil), cons(int, cons(int, nil)), \ldots\}$. The automaton is depicted in Figure 2.

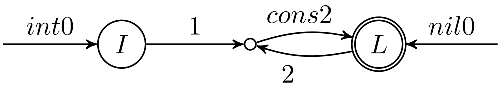


Figure 2. Visualisation of the FTA $\mathcal{A}$ from Example 2.

Regular tree expressions (RTEs) are defined (as in [5]) over two alphabets, $\mathcal{F}$ and $\mathcal{K}$. $\mathcal{F}$ is a ranked alphabet of symbols. $\mathcal{K}$ is a set of constants (special symbols with arity 0), $\mathcal{K} = \{\Box_1, \Box_2, \ldots, \Box_n\}$, $n \geq 0$, $\mathcal{F} \cap \mathcal{K} = \emptyset$. This alphabet is used to indicate the position where substitution operations take place.

Firstly, the substitution, i.e. replacing occurrences of $\Box_i$ by trees from a tree language L_j, is defined. Let $\mathcal{K} = \{\Box_1, \ldots, \Box_n\}$ and t be a tree over $\mathcal{F} \cup \mathcal{K}$, and $L_1, \ldots, L_n$ be tree languages. Then the *tree substitution* of $\Box_1, \ldots, \Box_n$ by $L_1, \ldots, L_n$ in t denoted by $t\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\}$ is the tree language defined by the following identities:

– $\Box_i\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\} = L_i$, for $i = 1, \ldots, n$,
– $a\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\} = \{a\}, \forall a \in \mathcal{F}_0 \cup \mathcal{K}$ and $a \neq \Box_1, \ldots, a \neq \Box_n$,
– $f(s_1, \ldots, s_n)\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\} = \{f(t_1, \ldots, t_n) \mid t_i \in s_i\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\}\}$.

The tree substitution can be generalized to languages: $L\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\} = \bigcup_{t \in L} t\{\Box_1 \leftarrow L_1, \ldots, \Box_n \leftarrow L_n\}$. The operation *alternation* of L_1 and L_2 is denoted by $L_1 + L_2$. It results in a set of trees obtained from the union of regular tree languages L_1 and L_2, i.e. $L_1 \cup L_2$. The operation *concatenation* of L_2 to L_1 through $\Box$, denoted by $\cdot\Box\ (L_1, L_2)$, is the set of trees obtained by substituting the occurrence of $\Box$ in trees of L_1 by trees of L_2, i.e. $\bigcup_{t \in L_1} t\{\Box \leftarrow L_2\}$. Given a tree language L over $\mathcal{F} \cup \mathcal{K}$ and $\Box \in \mathcal{K}$, the sequence $L^{n,\Box}$ is defined by the equalities $L^{0,\Box} = \{\Box\}$ and $L^{n+1,\Box} = \cdot\Box\ (L, L^{n,\Box})$. The operation *closure* is defined as $L^{*,\Box} = \bigcup_{n \geq 0} L^{n,\Box}$.

Finally, an RTE over alphabets $\mathcal{F}$ and $\mathcal{K}$ is defined inductively:

– the empty set ($\emptyset$) and a constant ($a \in \mathcal{F}_0 \cup \mathcal{K}$) are RTEs,
– if $E_1, E_2, \ldots, E_n$ are RTEs, $n > 0$, $f \in F_n$ and $\Box \in \mathcal{K}$, then: $E_1 + E_2$, $\cdot\Box\ (E_1, E_2)$, ${E_1}^{*,\Box}$, and $f(E_1, \ldots, E_n)$ are RTEs.

RTE E represents a language denoted by $L(E)$ and defined by the following equalities:

– $L(\emptyset) = \emptyset$,
– $L(a) = \{a\}$ for $a \in F_0 \cup K$,
– $L(f(E_1, \ldots, E_n)) = \{f(s_1, \ldots, s_n) \mid s_1 \in L(E_1), s_2 \in L(E_2), \ldots, s_n \in L(E_n)\}$,
– $L(E_1 + E_2) = L(E_1) \cup L(E_2)$,
– $L(\cdot\Box\ (E_1, E_2)) = L(E_1)\{\Box \leftarrow L(E_2)\}$,
– $L(E^{*,\Box}) = L(E)^{*,\Box}$.

We define the set $RTE(\mathcal{F}, \mathcal{K})$ to be a set of all RTEs over $\mathcal{F}$ and $\mathcal{K}$ alphabets. A regular tree language is recognisable if and only if it can be denoted by an RTE [5]. For the sake of simplicity we allow the alternation to act as an n-ary operator ($n \geq 2$), e.g., the RTE $(((E_1 + E_2) + E_3) + \cdots + E_n)$ can be written as $E_1 + E_2 + E_3 + \cdots + E_n$.

Example 3. Let $\mathcal{F} = \{nil0, int0, cons2\}$ and let $\mathcal{K} = \{\Box_1, \Box_2\}$. Then the RTE E from Figure3 denotes the language of lists of integers in LISP. $L(E) = \{nil0, cons2(int0, nil0), cons2(int0, cons2(int0, nil0)), \ldots\}$.

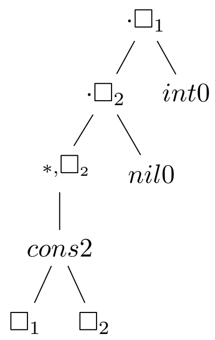


Figure 3. An RTE from Example 3 denoting the language of integer lists in LISP.

3 State elimination algorithm

The proof of Kleene's Theorem in Comon et al. [5] hinted that the conversion of FTAs to RTEs can be done by elimination of states. However, the algorithm itself is not presented in the book.

The following approach is inspired by the well-known state elimination algorithm for finite automata [8]. We eliminate states one by one and all paths that went through the eliminated state will be replaced by new transitions. Also, the transitions will now involve sets of trees described by an RTE (in the places where the classical string version would involve regular expressions) rather than individual symbols of the alphabet.

3.1 Generalized finite tree automaton

In order to represent a tree automaton whose transitions involve trees rather than symbols of the input alphabet we define an extension of the FTA called generalized finite tree automaton (GFTA). The GFTA differs from FTA mainly in the transition function which is defined over RTEs rather than over the individual symbols of a ranked alphabet.

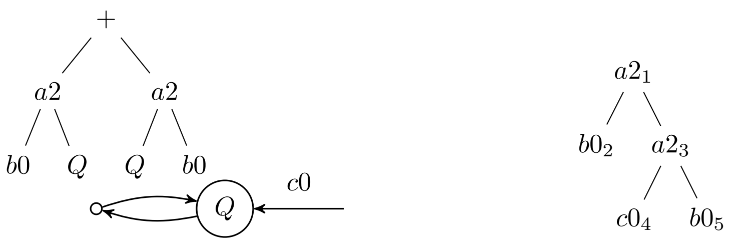
Definition 4. *Let $A' = (Q, \Sigma, Q_F, \Gamma)$ be a generalized finite tree automaton (GFTA). The meaning of Q, Σ and Q_F sets is the same as in an FTA and Γ is a mapping $RTE(\Sigma, Q) \times \mathcal{P}(Q) \mapsto Q$. The Q_F set is a singleton.*

Transition function Γ of a GFTA is in the form $E\{q_1, \ldots, q_n\} \to q$ where E is an $RTE(\Sigma, Q)$ and $q, q_1, \ldots, q_n \in Q$. The substitution symbols of RTEs act as references to the languages of the corresponding states and the constant alphabet of RTEs is therefore equal to the set of states, i.e., Q.

The order of transition's source states is no longer important as it is defined in the RTE because the children in RTEs are ordered. For this reason, it is no longer necessary to maintain the vector of source states of a transition ordered, and it can be converted to a set.

The run of GFTA is defined similarly to the run of FTA. A subtree t is labelled with the state q only if there exists a transition $E\{q_1 \ldots q_n\} \to q$ and $t \in L(E)$. Recall that the symbols from Q set in the RTE act as the references to the states of the automaton. A tree t is accepted by GFTA if t is labelled with the final state of the automaton. Language of a GFTA is the set of trees accepted by the automaton.

We use almost the same rules for depicting GFTAs as for FTAs. Only the edges leading to *join nodes* are no longer labelled with their position because they are no longer ordered.



(a) Example GFTA.

(b) Example tree.

Figure 4. Example GFTA and a sample tree it accepts.

Example 5. Consider the simple GFTA depicted in Figure 4a and the input tree t from Figure 4b. Subtree $c0_4$ of t is trivially labelled with state Q. Subtrees $a2_3$ and $a2_1$ are also labelled with Q state because both subtrees correspond to the RTE leading to the state Q. Note that the Q symbol in the RTE corresponds to the subtree labelled with state Q. Subtrees $b0_2$ and $b0_5$ are not labelled with a state.

Lemma 6. *An FTA* $A = (Q, \Sigma, Q_F, \Delta)$ *can be converted to an equivalent GFTA* $A' = (Q \cup q_f, \Sigma, \{q_f\}, \Gamma)$, $q_f \notin Q$.

Proof. Every transition $f_n(q_1, \ldots, q_n) \to q \in \Delta$ is transformed to $E\{q_1, \ldots, q_n\} \to q \in \Gamma$ where E is an RTE $f_n(q_1, \ldots, q_n)$. In order to have only one final state we also add a new state q_f and we create a transition $E\{q\} \to q_f$ where $E = q$ from every old final state $q \in Q_F$ to q_f. This is similar to adding ε-transitions in the string variant. It is easy to see that the languages accepted by A and A' are equal. □

The transformation of an FTA to a GFTA is obvious from the proof of Lemma 6, but we still formalise it in Algorithm 1. We also formulate Lemma 8, which states that every single-state GFTA can be converted to an RTE.

Algorithm 1: FTA to GFTA

```
input  : FTA $A = (Q, \Sigma, Q_F, \Delta)$
output : GFTA $A' = (Q', \Sigma, Q'_F, \Gamma)$ corresponding to $A$
function ExtendFTA($A = (Q, \Sigma, Q_F, \Delta)$):
    $Q' = Q \cup \{q_f\}$ $(q_f \notin Q)$ // new final state
    $Q'_F = \{q_f\}$
    $\Gamma = \{E\{q_1, \ldots, q_n\} \to q$ where $E = f_n(q_1, \ldots, q_n) \mid \forall f_n(q_1, \ldots, q_n) \to q \in \Delta\}$
    $\Gamma = \Gamma \cup \{E'\{q\} \to q_f$ where $E' = q \mid \forall q \in Q_F\}$
    return $A' = (Q', \Sigma, Q'_F, \Gamma)$
```

Example 7. Figure 5 shows the tree automaton from Example 2 converted to GFTA by Algorithm 1.

Lemma 8. *Let* $A = (Q, \Sigma, Q_F, \Gamma)$ *be a GFTA with only 1 state* ($|Q| = 1$) *that is also the final state* ($Q = Q_F$) *and transitions in the form* $E_i \to q$*. One can generate an RTE* E *such that* $L(E) = L(A)$.

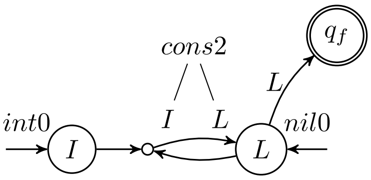


Figure 5. Visualisation of the GFTA created from the FTA from Example 2.

Proof. Multiple transitions of the form $E_i' \to q$ can be transformed into a single transition in the form $E \to q$ where $E = E_1' + \cdots + E_n'$ and E_i' corresponds to the RTE of the i-th original transition.

Now it is clear that any tree accepted by the automaton must also be in the language denoted by the RTE corresponding to the only transition of the automaton and vice versa. Therefore the language denoted by the RTE E is equivalent to the language of the GFTA. □

3.2 Elimination of a single state

Before we state the full algorithm, we must define the process of eliminating a single non-final state from a GFTA. The elimination of a single state q, $q \in Q \setminus Q_F$ from a GFTA modifies the GFTA in such a way that the state q is no longer present. Therefore the transitions involving the state q are removed as well. However, the language of the automaton must remain unchanged. To compensate, some new transitions between the remaining states are added.

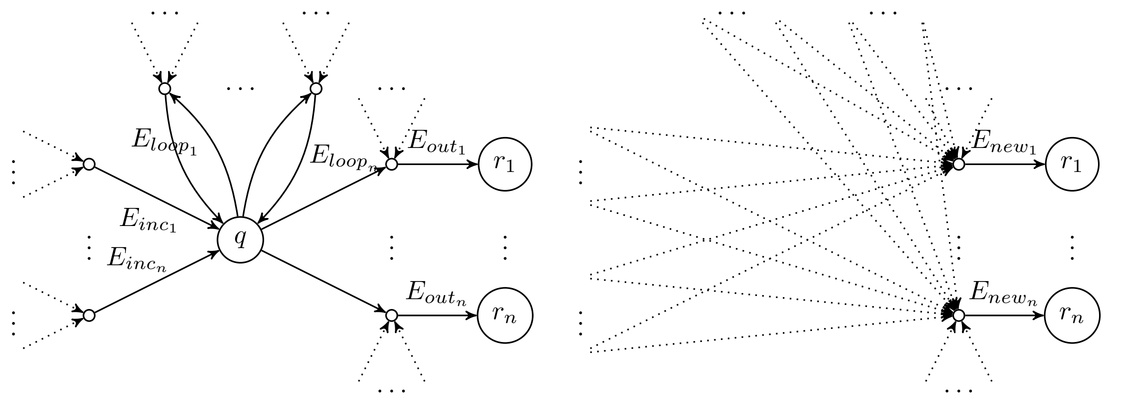
With respect to the state q we classify the transitions of the automaton into the following four groups: A transition is *incoming* if q is only a target state but not a source. If q is among the source states of a transition and also a target state, then the transition is classified as *looping*. If q is only a source state but not a target, it is called an *outgoing* transition. If q has no part in the transition, it is classified as an *other* transition. It is obvious that *other* transitions are left intact when q is eliminated. Definition 9 formalises the classification.

Definition 9. *Function* trtype *classifies the transition of a GFTA w.r.t. the state* $q \in Q$.

$$\mathrm{trtype}(E\{p_1,\ldots,p_n\} \to p, q) = \begin{cases} \text{incoming} & \text{if } q \notin \{p_1,\ldots,p_n\} \wedge q = p \\ \text{outgoing} & \text{if } q \in \{p_1,\ldots,p_n\} \wedge q \neq p \\ \text{looping} & \text{if } q \in \{p_1,\ldots,p_n\} \wedge q = p \\ \text{other} & \text{if } q \notin \{p_1,\ldots,p_n\} \wedge q \neq p \end{cases}$$

Example 10. Let $A = (Q, \Sigma, Q_F, \Gamma)$ be the GFTA from Figure 5. According to the Definition 9 the transitions from Γ with respect to state L are classified as follows: $\mathrm{trtype}(int0 \to I, L) = \mathrm{other}$, $\mathrm{trtype}(nil0 \to L, L) = \mathrm{incoming}$, $\mathrm{trtype}(cons2\{I, L\} \to L, L) = \mathrm{looping}$, $\mathrm{trtype}(L \to q_f, L) = \mathrm{looping}$.

Now consider the fragment of GFTA visualised in Figure 6a. Arbitrary non-final state q of the GFTA may have incoming transitions (such transitions are labelled in the picture by the RTEs E_{inc_i}), looping transitions (E_{loop_i}) and outgoing transitions (E_{out_i}). The other transitions are obviously left intact by the process of eliminating q



(a) Before the elimination of state q.

(b) After the elimination of state q by Lemma 12.

Figure 6. Fragment of a GFTA before and after the elimination of state q.

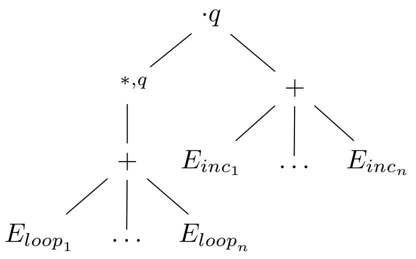


Figure 7. RTE equivalent to the language of a state of GFTA.

because q is not involved in those transitions. Let us state the following lemmas that will define the elimination process.

Lemma 11. *The concatenation operation in RTEs is associative, i.e., for RTEs x, y, z the following holds: $\cdot\Box\ (\cdot\Box\ (x, y), z) = \cdot\Box\ (x, \cdot\Box\ (y, z))$.*

Proof. No mater the order of application of the concatenation operation in either RTEs $\cdot\Box\ (\cdot\Box\ (x, y), z)$ and $\cdot\Box\ (x, \cdot\Box\ (y, z))$, the occurrences of $\Box$ in the RTE x are the place of substitution of a language given by RTE y and the occurrences of $\Box$ in the RTE y are the place of substitution of a language given by RTE z. □

Lemma 12. *Let $A = (Q, \Sigma, Q_F, \Gamma)$ be a GFTA and $q \in Q \backslash Q_F$. Let $E_{loop_1}, \dots, E_{loop_n}$ be RTEs collected from looping transitions w.r.t. the state q, let $E_{inc_1}, \dots, E_{inc_n}$ be RTEs from incoming transitions w.r.t. the state q and let $E_{out_1}, \dots, E_{out_n}$ be RTEs collected from outgoing transitions w.r.t. the state q. Then the language of state q can be represented as an RTE $E_q = \cdot q\ ((E_{loop_1} + \dots + E_{loop_n})^{*,q}, (E_{inc_1} + \dots + E_{inc_n}))$ (for clarity, the RTE fragment is visualised in Figure 7). Then the references to q in E_{out_i} can be replaced (using the concatenation operation) with an RTE denoting the language of state q. The E_{out_i} transition then becomes $E_{new_i} = \cdot q\ (E_{out_i}, E_q)$. Note that empty alternation equals to $\emptyset$.*

Proof. The proof is essentially the same as the proof of Kleene's Theorem in [5, Prop. 2.2.7]. We only use different RTE fragment for the E_{new_i} because we find it more intuitive. The fragment proposed in [5], i.e., $E_{new_i} = \cdot q\ (\cdot q\ (E_{out_i}, (E_{loop_1} + \dots + E_{loop_n})^{*,q}), (E_{inc_1} + \dots + E_{inc_n}))$, is equivalent as follows from Lemma 11. □

Lemma 13. *After eliminating state $q \in Q$, all occurrences of symbol q in the automaton's transition function are bounded by some concatenations over q.*

Proof. References to state q can appear either at outgoing and looping transitions of state q or at incoming and looping transitions of a state $p \in Q$, $p \neq q$ of the automaton. Firstly, by eliminating the state q the references at outgoing and looping transitions are surely bounded under concatenation nodes (by Lemma 12). Now for the state p. If q appears at a incoming or a looping transition of the state p then (by Definition 9) the transition is also an outgoing transition of state q. Therefore the reference is properly bounded by eliminating state q. □

The state elimination algorithm is intentionally designed not to replace the references inside the RTE but rather to utilize the concatenation operation. Using this approach, there is no need to traverse the RTEs to replace all the occurrences. Furthermore, when multiple references are present, the replacement is not copied to multiple places. This approach is formalised in Algorithm 2. Function *Alternate* is used in Algorithm 2. For a set of transitions of labelled with RTEs E_i the function returns an alternation of those, i.e., an RTE $E_1 + \cdots + E_n$.

Algorithm 2: Elimination of a single state in a GFTA

```
input    : GFTA $A = (Q, \Sigma, Q_F, \Gamma)$, $q \in Q$
output   : GFTA $A' = (Q \setminus \{q\}, \Sigma, Q_F, \Gamma')$, i.e., $A$ without state $q$ and $L(A) = L(A')$
function EliminateState($A = (Q, \Sigma, Q_F, \Gamma)$, $q$):
    $incoming, looping, sources, \Gamma' = \emptyset, \emptyset, \emptyset, \emptyset$
    foreach $E\{q_1, \dots, q_n\} \to r \in \Gamma$ do
        if trtype($E\{q_1, \dots, q_n\} \to r, q$) = incoming then
            $incoming = incoming \cup \{E\{q_1, \dots, q_n\} \to r\}$
            $sources = sources \cup \{q_1, \dots, q_n\}$
        else if trtype($E\{q_1, \dots, q_n\} \to r, q$) = looping then
            $looping = looping \cup \{E\{q_1, \dots, q_n\} \to r\}$
            $sources = sources \cup \{q_1, \dots, q_n\}$
        else if trtype($E\{q_1, \dots, q_n\} \to r, q$) = other then
            $\Gamma' = \Gamma' \cup \{E\{q_1, \dots, q_n\} \to r\}$                // not involved
    foreach $E\{q_1, \dots, q_n\} \to r \in$ outgoing do
        $E_{new} = \cdot q\ (E, (\cdot q\ (Alternate(looping)^{*,q}, Alternate(incoming))))$      // Lemma 12
        $\Gamma' = \Gamma' \cup \{E_{new}\{sources \setminus \{q\} \cup \{q_1, \dots, q_n\}\} \to r\}$
    return $A' = (Q \setminus \{q\}, \Sigma, Q_F, \Gamma')$
```

Lemma 14. *Applying Algorithm 2 to a GFTA $A = (Q, \Sigma, Q_F, \Gamma)$ does not increase the cardinality of Γ.*

Proof. The claim follows directly from Algorithm 2. For every other and outgoing transition one transition is added to Γ'. For every incoming and looping transition no new transitions are added. Therefore it always holds that $|\Gamma'| \leq |\Gamma|$. □

Lemma 15. *Algorithm 2 runs in $O(|Q| \cdot |\Gamma|)$ time for input GFTA $A = (Q, \Sigma, Q_F, \Gamma)$.*

Proof. Both for loops obviously iterates over at most $|\Gamma|$ elements and also require some work for merging two sets of size at most Q. Therefore, the upper bound of running time is $O(|Q| \cdot |\Gamma|)$. □

3.3 State elimination algorithm

The previous subsection stated an algorithm for the elimination of a single non-final state from a GFTA. This process can be repeated (in arbitrary order of states) until we obtain a single-state automaton with such transitions that allow us to directly apply Lemma 8. Algorithm 3 formalises this simple process.

Algorithm 3: State elimination of a GFTA

```
input  : FTA $A = (Q, \Sigma, Q_F, \Delta)$
output : RTE $E$ such that $L(A) = L(E)$
function StateElimination($A = (Q, \Sigma, Q_F, \Delta)$):
    GFTA $A' = (Q', \Sigma', \{q_f\}, \Gamma)$ = ExtendFTA($A$)                 // Algorithm 1
    foreach $q \in Q' \setminus \{q_f\}$ do
        $A'$ = EliminateState($A', q$)                                        // Algorithm 2
    return Alternate($\{E_i \mid \forall E_i\{q_1, \ldots, q_n\} \to q_f\} \in \Gamma$)   // remaining transitions
```

Theorem 16. *Algorithm 3 converts an FTA $A = (Q, \Sigma, Q_F, \Delta)$ to an RTE E such that $L(A) = L(E)$.*

Proof. The algorithm creates a single-state GFTA. Therefore the claim immediately follows from Lemmas 8, 12 and 13.

Theorem 17. *The total running time of Algorithm 3 is $O(|Q|^2 \cdot (|\Delta| + |Q_F|))$ time.*

Proof. The running time consists of converting the original FTA to GFTA and $|Q|$ invocations of Algorithm 2. The largest value for $|\Gamma|$ parameter of Algorithm 2 is in the first iteration where $|\Gamma| = |\Delta| + |Q_F|$ (Lemma 6). After the elimination of a single state, the number of transitions can only decrease or remain the same (Lemmas 12 and 14). Using time complexity of Algorithm 2 stated in Lemma 15, the total upper bound on the complexity of Algorithm 3 is $O(|Q| \cdot |Q| \cdot (|\Delta| + |Q_F|))$. □

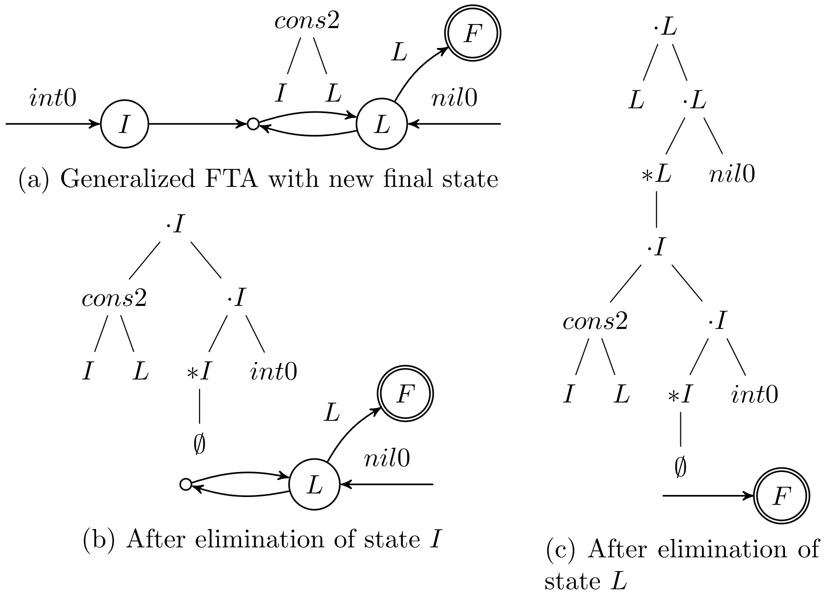
Example 18. Let A be the GFTA from Figure 2. The trace run of the algorithm is shown in Figure 8. The states of A are eliminated in lexicographical order, i.e., I, L.

4 Conclusion

We presented a simple full algorithm for the construction of a regular tree expression (RTE) equivalent to given finite tree automaton (FTA) by eliminating states. Both the idea and implementation are also very similar to the original State elimination for finite (string) automata and regular expressions [8]. This construction was originally hinted in [5] to prove the Kleene Theorem, i.e., the equivalence between languages of RTEs and FTAs. We showed that the idea of eliminating states one by one from an FTA forms an easy and intuitive algorithm that can be easily implemented.

The presented algorithm runs in $O(|Q|^2 \cdot (|\Delta| + |Q_F|))$ time, i.e., it is proportional to the number of states and the size of the transition function of the converted automaton. Also, different order of elimination may create different RTE but all of them denote the same language. However, this also holds for the original string algorithm [8,11].

Future work may focus on the Algorithm 2. Better data structures may help in improving the time complexity upper bound. Another interesting problem is finding



(a) Generalized FTA with new final state

(b) After elimination of state I

(c) After elimination of state L

Figure 8. Example run of the algorithm on the FTA from Figure 2.

the best elimination order. Different orderings give different resulting RTEs and some of them are smaller than others. Similar experimental research was done in the string elimination method [11].

You can find the `C++` implementation of the presented algorithm in the latest versions of Algorithms Library Toolkit project [1]. We also tested the implementation by converting various random FTAs to RTEs using this algorithm and then back using the algorithm from [12] adapted to FTAs.

References

1. *Algorithms Library Toolkit*: `https://alt.fit.cvut.cz`.
2. A. V. Aho and J. D. Ullman: *The theory of parsing, translation, and compiling. 1: Parsing*, Prentice-Hall, 1972.
3. A. Belabbaci, H. Cherroun, L. Cleophas, and D. Ziadi: *Tree pattern matching from regular tree expressions*. Kybernetika, 54(2) 2018, pp. 221–242.
4. L. Cleophas: *Tree Algorithms: Two Taxonomies and a Toolkit*, PhD thesis, Department of Mathematics and Computer Science, Eindhoven University of Technology, Eindhoven, Apr. 2008.
5. H. Comon, M. Dauchet, R. Gilleron, C. Löding, F. Jacquemard, D. Lugiez, S. Tison, and M. Tommasi: *Tree automata techniques and applications*, 2007, Release October 2007.
6. F. Gécseg and M. Steinby: *Tree Languages*, vol. 3 of Handbook of Formal Languages, Springer, 1997, pp. 1–68.
7. Y. Guellouma and H. Cherroun: *From tree automata to rational tree expressions*. Int. J. Found. Comput. Sci., 29(6) 2018, pp. 1045–1062.
8. J. E. Hopcroft, R. Motwani, and J. D. Ullman: *Introduction to automata theory, languages, and computation (2. ed)*, Addison-Wesley, 2003.
9. D. Kuske and I. Meinecke: *Construction of tree automata from regular expressions*, in Developments in Language Theory, 12th International Conference, DLT 2008, Kyoto, Japan, September 16-19, 2008. Proceedings, 2008, pp. 491–503.
10. É. Laugerotte, N. O. Sebti, and D. Ziadi: *From regular tree expression to position tree automaton*, in Language and Automata Theory and Applications - 7th International Conference, LATA 2013, Bilbao, Spain, April 2-5, 2013. Proceedings, 2013, pp. 395–406.
11. N. Moreira, D. Nabais, and R. Reis: *State elimination ordering strategies: Some experimental results*, vol. 31, 08 2010, pp. 139–148.

12. T. Pecka, J. Trávníček, R. Polách, and J. Janoušek: *Construction of a pushdown automaton accepting a postfix notation of a tree language given by a regular tree expression.* 2018, pp. 6:1–6:12.
13. R. Polách, J. Janoušek, and B. Melichar: *Regular tree expressions and deterministic pushdown automata*, in Proceedings of the 7th Doctoral Workshop on Mathematical and Engineering Methods in Computer Science, 2011, pp. 70–77.