Uncorrected proof for Journal of Automata, Languages and Combinatorics **23** (2018) 1–3
Final version created on April 3, 2018

THE OUTPUT SIZE PROBLEM FOR STRING-TO-TREE TRANSDUCERS

Martin Berglund[(C)] Frank Drewes[(B)] Brink van der Merwe[(A)]

[(A)] *Department of Computer Science, Stellenbosch University*
7600 Stellenbosch, South Africa
abvdm@cs.sun.ac.za

[(B)] *Department of Computing Science, Umeå University*
901 87, Umeå, Sweden
drewes@cs.umu.se

[(C)] *Department of Information Science and Center for AI Research (CSIR)*
Stellenbosch University, 7600 Stellenbosch, South Africa
pmberglund@sun.ac.za

ABSTRACT

The output size problem, for a string-to-tree transducer, is to determine the asymptotic behavior of the function describing the maximum size of output trees, with respect to the length of input strings. We show that the problem to determine, for a given regular expression, the worst-case matching time of a backtracking regular expression matcher, can be reduced to the output size problem. The latter can, in turn, be solved by determining the degree of ambiguity of a non-deterministic finite automaton.

Keywords: string-to-tree transducers, output size, backtracking regular expression matchers, NFA ambiguity

1. Introduction

The complexity of determining the asymptotic behavior of the maximum output size for trees produced by a given top-down tree transducer, as a function of the size of input trees, was initially studied in [5]. It was shown that the exponential output size problem is NL-complete for total top-down tree transducers, and DEXPTIME-complete for top-down tree transducers in general. Naturally, this problem asks whether the size of the output trees grows exponentially in the size of the input trees. We investigate the output size problem for string-to-tree transducers, and consider in particular the complexity of determining the degree of the polynomial, in cases where the maximum output size is polynomial in the size of input strings.

The motivation for this research is provided by the observation that the problem of determining the worst-case matching time of a backtracking regular expression matcher [2, 3] in terms of the length of the input string, can be reduced to an output

size problem, by constructing a transducer producing as output the computation tree of the matcher in question, for a given input string to the matcher. Thus in this case, the maximum output size provides the worst-case matching time behavior of a backtracking regular expression matcher, for a given regular expression.

Another motivation for this research, although not pursued in this paper, is provided by the fact that ET0L languages are precisely the yield of output languages of string-to-tree transducers ([6, 7]). Intuitively, the states of the transducer are the nonterminals of the ET0L system and the input alphabet of the transducer consists of labels for tables of an ET0L system. Also, each transduction step corresponds to a derivation step in the ET0L system which uses the table labeled by the input symbol consumed by the transducer. The problem of determining the asymptotic behavior of the maximum length of words in ET0L systems, as a function of the number of derivation steps used, can thus be reduced to the output size problem for string-to-tree transducers. A particular instance of growth of ET0L systems that has been well studied, is the growth function for D0L systems (see for example the section on L growth in [11, Chapter 5]), which corresponds to the output size problem for deterministic string-to-tree transducers with input strings from a unary alphabet.

In this paper, we determine the complexity of the output size problem for string-to-tree transducers by relating it to the problem of determining the degree of ambiguity of non-deterministic finite automata. In this way, we show that all output size problems we consider are NL-complete for total string-to-tree transducers and PSPACE-complete for string-to-tree transducers in general.

The outline of the paper is as follows. In the next section we introduce the required notation and definitions. This is followed by an outline of the regular expression matcher application. After this, we describe the complexity of deciding various ambiguity problems for non-deterministic finite automata (NFA), followed by a section describing how output size problems can be reduced to deciding ambiguity in NFA. We finally provide conclusions and describe possible future work.

This article is a revised and extended version of [12].

2. Definitions

In this section, we introduce the notation and some of the definitions required for the remainder of the paper.

For an alphabet Σ, we denote the set of all strings (or sequences) over Σ by Σ^*. In particular, Σ^* contains the empty string ε. We assume that $\varepsilon \notin \Sigma$ and let $\Sigma_\varepsilon = \Sigma \cup \{\varepsilon\}$. The length of a string w is denoted by $|w|$. We use $\mathbb{N}_+$ for the positive integers and $\mathbb{N} = \mathbb{N}_+ \cup \{0\}$. Although Σ is henceforth only used to denote finite alphabets, we use $\mathbb{N}_+^*$ to denote the set of strings over the infinite alphabet $\mathbb{N}_+$.

A *tree*, with labels in a finite set Δ, is a function $t\colon t_D \to \Delta$, where $t_D \subseteq \mathbb{N}_+^*$ is a non-empty, finite set of vertices (or nodes) such that

(I) t_D is prefix-closed, i. e., for all $v \in \mathbb{N}_+^*$ and $i \in \mathbb{N}_+$, $vi \in t_D$ implies $v \in t_D$, and

(II) t_D is closed to the left, i. e., for all $v \in \mathbb{N}_+^*$ and $i \in \mathbb{N}_+$, $v(i+1) \in t_D$ implies $vi \in t_D$.

The vertex ε is the root of the tree and vertex vi is the i-th child of v. We assume that Δ is a ranked alphabet, i. e., Δ is a union of (not necessarily disjoint) sets

$$\Delta^{(0)} \cup \Delta^{(1)} \cup \Delta^{(2)} \cup \cdots$$

(with only finitely many of the $\Delta^{(i)}$ being non-empty). When $f \in \Delta^{(k)}$, we say f has rank k, and we allow symbols in Δ to have more than one possible rank, which is convenient for our application to regular expressions. If we want to indicate explicitly that we consider f as a rank k symbol, we denote f by $f^{(k)}$. Also, all trees are ranked, thus if $v \in t_D$, then there exists a $k \in \mathbb{N}_+$, where k is one of the possible ranks of $t(v)$, such that $vk \in t_D$ but $v(k+1) \notin t_D$. A node v such that $v1 \notin t_D$, is a leaf.

Definition 1. The *size* of a tree $t\colon t_D \to \Delta$ is defined as $|t_D|$, and denoted by $|t|$.

We denote by

$$|t|_S = |\{\ v \in t_D \mid t(v) \in S\ \}|,$$

for $S \subseteq \Delta$, the number of occurrences of symbols from S in t. Moreover, t/v ($v \in t_D$), denotes the tree t', with

$$t'_D = \{\ w \in \mathbb{N}_+^* \mid vw \in t_D\ \},$$

where $t'(w) = t(vw)$ for all $w \in t'_D$. Given trees $t_1, \ldots, t_k$ and $\alpha \in \Delta^{(k)}$, we let

$$\alpha[t_1, \ldots, t_k]$$

denote the tree t with $t(\varepsilon) = \alpha$ and $t/i = t_i$ for all $i \in \{1, \ldots, k\}$. The tree $\alpha[\,]$ that consists only of a leaf labeled α may be abbreviated as α.

The yield of a tree $t = \alpha[t_1, \ldots, t_k]$, denoted by $\mathrm{yield}(t)$, is the concatenation of the labels of the leaves from left to right, i. e.,

$$\mathrm{yield}(t) = \begin{cases} \alpha & \text{if } k = 0 \\ \mathrm{yield}(t_1) \cdots \mathrm{yield}(t_k) & \text{if } k > 0. \end{cases}$$

Trees t with $t(v) \in \Delta^{(0)} \cup \Delta^{(1)}$, for all $v \in t_D$, may be written as $\alpha_1\alpha_2 \ldots \alpha_n$ (instead of $\alpha_1[\alpha_2[\ldots[\alpha_n]]]$), where $\alpha_i \in \Delta^{(1)}$ for $i < n$ and $\alpha_n \in \Delta^{(0)}$. Given a ranked alphabet Δ, the set of all ranked trees $t\colon t_D \to \Delta$ is denoted by T_Δ. Moreover, if Q is an alphabet disjoint from Δ, we let

$$T_\Delta(Q) := T_{\Delta \cup Q}$$

where the symbols from Q appear only at the leaves.

In our NFA definition, given next, the transition function δ is defined to allow for parallel transitions on the same symbol between a pair of states. Thus, it is of the form

$$\delta\colon Q \times \Sigma_\varepsilon \times Q \to \mathbb{N}_+,$$

where $\delta(p, \alpha, q) = i$ indicates that there are i transitions on α between p and q.

Definition 2. A *non-deterministic finite automaton* (NFA) is a tuple

$$A = (Q, \Sigma, I, \delta, F)$$

where

(i) Q is a finite set of *states*;
(ii) Σ is the *input alphabet*;
(iii) $I \subseteq Q$ is the set of *initial states*;
(iv) the partial function $\delta\colon Q \times \Sigma_\varepsilon \times Q \to \mathbb{N}_+$ is the *transition function*; and
(v) $F \subseteq Q$ is the set of *final states*.

Also, $|A|_Q := |Q|$ and $|A|_\delta := \sum_{q_1,q_2\in Q,\alpha\in\Sigma_\varepsilon} \delta(q_1, \alpha, q_2)$ are the *state* and *transition sizes* of A respectively.

Next we define (accepting) runs and the language accepted by an NFA.

Definition 3. For an NFA $A = (Q, \Sigma, I, \delta, F)$ and $w \in \Sigma^*$, a *run* on w is a string

$$r = s_0\alpha_1(j_1)s_1 \cdots s_{n-1}\alpha_n(j_n)s_n,$$

with $s_0 \in I$, $s_i \in Q$, $\alpha_i \in \Sigma_\varepsilon$ and $j_i \in \mathbb{N}_+$ such that

$$\delta(s_i, \alpha_{i+1}, s_{i+1}) \geq j_{i+1}$$

for $0 \leq i < n$, and $w = \alpha_1 \cdots \alpha_n$ (where ε is interpreted as the empty string rather than as a symbol). A run is *accepting* if $s_n \in F$, and $w \in \Sigma^*$ is *accepted by* A if there is an accepting run of A on w. The language accepted by A is

$$\mathcal{L}(A) = \{\, w \in \Sigma^* \mid w \text{ is accepted by } A \,\}.$$

We say that A has a path from $p \in Q$ to $q \in Q$ labeled $w \in \Sigma^*$, and denote this by $p \xrightarrow{w}_A q$, if $w \in \mathcal{L}(A')$ with $A' = (Q, \Sigma, \{p\}, \delta, \{q\})$. A state q is *useful* if $q_I \xrightarrow{w}_A q$ and $q \xrightarrow{v}_A q_F$ for some $w, v \in \Sigma^*$, $q_I \in I$, and $q_F \in F$.

By writing $p \xrightarrow{\alpha(j)} q$, where $1 \leq j \leq \delta(p, \alpha, q)$, we refer to the j-th-transition on α from p to q, but we also write $p \xrightarrow{\alpha} q$ if the specific choice of j is not important. Although parallel transitions do not influence the language accepted by an NFA, they do influence the number of accepting runs of a given input string, and thus play a role in our setting.

Remark 4. Instead of our definition of NFA, one could also use weighted automata over the semiring $\mathbb{N}$, thus interpreting $\delta(p, \alpha, q)$ as the weight of a single transition from p to q under α. We prefer the view above, to keep Definition 12, in Section 5, as close as possible to the corresponding definition from [1].

We now recall string-to-tree transducers, followed by the definition of its set of output trees, when applied to a given input string.

Definition 5. A *string-to-tree transducer* (or *transducer*, for short) is a tuple

$$td = (Q, \Gamma, \Delta, I, \delta),$$

where

- $\Gamma = \Gamma^{(1)}$ and Δ are the finite ranked input and output alphabets, respectively (with all input symbols having only rank 1),
- Q is a finite set of states disjoint with Δ,
- $I \subseteq Q$ is the set of initial states, and
- $\delta \subseteq (Q \times \{\$\} \times T_\Delta) \cup (Q \times \Gamma_\varepsilon^{(1)} \times T_\Delta(Q))$ is the transition relation.

When $(q, \alpha, t) \in \delta$, we also say that δ contains a *rule* $q \xrightarrow{\alpha} t$. Also,

$$|td|_\delta := \sum_{(q,\alpha,t)\in\delta} |t|$$

is the *transition size* of td.

For $w \in \Gamma^*$, the set of output trees, when applying td to w, is denoted by

$$td(w) \subseteq T_\Delta,$$

and defined as follows. We have that $t \in td(w)$ if w can be written as $\alpha_1 \cdots \alpha_n$, with $\alpha_i \in \Gamma_\varepsilon$ for $i \leq n$, such that there exists a sequence of trees $t_0, \ldots, t_n \in T_\Delta(Q)$ with $t_0 \in I$ and for every $i \in \{1, \ldots, n\}$, t_i is obtained from t_{i-1} by replacing every leaf v for which $t_{i-1}(v) \in Q$ with a tree t' such that $t_{i-1}(v) \xrightarrow{\alpha_i} t'$, and similarly, t is obtained from t_n by replacing every leaf v for which $t_n(v) \in Q$ with a tree $t' \in T_\Delta$ such that $t_n(v) \xrightarrow{\$} t'$.

We now define when transducers are total and deterministic. The presence of ε-input rules leads to non-standard definitions. According to the following definition, a transducer td is total if $td_q(w) \neq \emptyset$ for all $q \in Q$ and $w \in \Gamma^*$ (where td_q is td with its initial state replaced by q), and deterministic if, in each situation, at most one rule applies.

Definition 6. A transducer

$$td = (Q, \Gamma, \Delta, I, \delta)$$

is *total* if $q \in Q$, $a \in \Gamma$ implies $td_q(a) \neq \emptyset$ and $td_q(\varepsilon) \neq \emptyset$, where

$$td_q = (Q, \Gamma, \Delta, \{q\}, \delta).$$

Also, td is *deterministic* if for all $q \in Q$ and $\alpha \in \Gamma \cup \{\$\}$, there is at most one rule of the form $q \xrightarrow{\varepsilon} t$ or $q \xrightarrow{\alpha} t'$ in δ.

Next we give the output size definition from [5], and also an output size definition based on the length of the yield of output trees.

Definition 7. The *full output size* and the *yield output size* of a transducer td are given by functions $os^F_{td}, os^Y_{td} \colon \mathbb{N}_+ \to (\mathbb{N} \cup \{\infty\})$, respectively, such that

$$os^F_{td}(n) = \sup\{\ |t| \mid t \in td(s) \text{ and } |s| \leq n\ \} \text{ (with } \sup\ \emptyset = 0), \text{ and}$$
$$os^Y_{td}(n) = \sup\{\ |\text{yield}(t)| \mid t \in td(s) \text{ and } |s| \leq n\ \}.$$

The *exponential output size problem* is to decide if os^F_{td} has exponential rate of growth, and the *polynomial output size problem* is to determine the degree of the asymptotic polynomial growth of os^F_{td} and os^Y_{td} (if it is polynomial).

Note that since we allow ε-input transducer rules, it may happen that output trees of arbitrary size are produced for input trees of a given fixed size.

In the following example, transducers with exponential and with polynomial output size (of arbitrary degree), are given.

Example 8. First we define transducers td_k with $os^F_{td_k}$ (and $os^Y_{td_k}$) exponential. Let

$$td_k = (Q, \Gamma, \Delta_k, \{q_0\}, \delta_k)$$

with

$$\begin{aligned} Q &= \{q_0\}, \\ \Gamma &= \{f\}, \\ \Delta_k &= \{\diamond^{(0)}, g^{(k)}\}, \text{ for some fixed integer } k \geq 2, \text{ and} \\ \delta_k &= \{q_0 \xrightarrow{f} g[q_0, \ldots, q_0], q_0 \xrightarrow{\$} \diamond\}. \end{aligned}$$

Then $td_k(f^n)$ is a perfect k-ary tree of height n, and $|td_k(f^n)|$ is thus exponential, with base k, in n.

Next we give transducers $\overline{td}_k$, for $k \geq 1$, such that

$$os^F_{\overline{td}_k} \quad \text{and} \quad os^Y_{\overline{td}_k}$$

are polynomials of degrees k and $(k-1)$, respectively. In general, in the polynomial case, the degree of os^F_{td} is at most 1 larger than that of os^Y_{td}. We let

$$\overline{td}_k = (Q_k, \Gamma, \Delta, \{q_k\}, \delta_k)$$

with

$$\begin{aligned} Q_k &= \{q_1, \ldots, q_k\}, \\ \Gamma &= \{f\}, \\ \Delta &= \{\diamond^{(0)}, f^{(1)}, g^{(2)}\}, \text{ and} \\ \delta_k &= \{q_i \xrightarrow{f} g[q_{i-1}, q_i] \text{ for } 1 < i \leq k\} \cup \{q_1 \xrightarrow{f} f[q_1]\} \cup \{q_i \xrightarrow{\$} \diamond \text{ for } 0 \leq i \leq k\}. \end{aligned}$$

We have that

$$os^F_{td}(n) = |\overline{td}_k(f^n)| \in \Theta(n^k)$$

and

$$os^{Y}_{\overline{td}_k}(n) = |\text{yield}(\overline{td}_k(f^n))| \in \Theta(n^{k-1}),$$

which is obtained by induction, using

$$\overline{td}_k(f^n) = g[\overline{td}_{k-1}[f^{n-1}], \overline{td}_k[f^{n-1}]]$$

for $k > 1$, and thus

$$|\overline{td}_k(f^n)| = (n+1) + \sum_{i=0}^{n-1} |\overline{td}_{k-1}(f^i)|$$

and

$$|\text{yield}(\overline{td}_k(f^n))| = 1 + \sum_{i=0}^{n-1} |\text{yield}(\overline{td}_{k-1}(f^i))|,$$

with $|\overline{td}_1(f^i)| = (i+1)$ and $|\text{yield}(\overline{td}_1(f^i))| = 1$.

3. Regular Expression Matching Motivation

In this section, we provide more detail on the application of the output size problem to the worst-case matching time of backtracking regular expression matchers (regex matchers). We start by recalling the definition of a prioritized non-deterministic finite automaton (pNFA). In a pNFA, priorities are placed on ε-transitions from a given state, in contrast to NFA, and input is matched by doing an input directed depth first search on the pNFA, using the specified priorities.

Definition 9 [3]. A *prioritized non-deterministic finite automaton (pNFA)* is a tuple $A = (Q_1, Q_2, \Sigma, q_0, \delta_1, \delta_2, F)$, where if $Q := Q_1 \cup Q_2$, we have

(I) Q_1 and Q_2 are disjoint finite sets of *states*;
(II) Σ is the *input alphabet*;
(III) $q_0 \in Q$ is the *initial state*;
(IV) $\delta_1 \colon Q_1 \times \Sigma \to Q$ is the *deterministic*, but not necessarily total, *transition function*;
(V) $\delta_2 \colon Q_2 \to Q^*$ is the *non-deterministic prioritized transition function*; and
(VI) $F \subseteq Q_1$ is the set of *final states*.

Note that the transition function $\delta_2 \colon Q_2 \to Q^*$, in the definition above, places an ordering on the outgoing ε-transitions for each state $q \in Q_2$, i. e., the order they have in the sequence $\delta_2(q) \in Q^*$. Acceptance in pNFA is defined the same as in NFA. However, the prioritization makes it possible to assign to each accepted string a *unique accepting run*, namely the run among those ending in a final state that has the highest priority (not considering runs that use the same ε-transition twice in any consecutive sequence of ε-transitions).

In a regex matcher, a modified Thompson construction is used to convert a regular expression R to a pNFA A, instead of an NFA [2]. To simplify our discussion, we only consider pNFA without δ_2-loops, i. e., when ignoring the priorities of δ_2-transitions, an NFA without ε-loops is obtained. Regular expression matchers perform a depth-first search for accepting runs in the order of priority. They keep track of which ε-transitions were used since the last input symbol was consumed, and disable an ε-transition once it is used, until the next input symbol is consumed. This behaviour ensures that a matcher avoids infinite loops. A procedure, called *flattening*, which in effect performs a depth first search on all ε-transitions from each state in Q_2, and replaces a sequence of ε-transitions by a single ε-transition while keeping the order information encoded by δ_2-transitions intact, is described in [4, Section 5], and can be used to remove ε-cycles while only affecting the matching time up to a constant factor. A regex matcher attempts to match an input string w by doing a preorder traversal on $td_A(w)$, where td_A is a transducer (defined next – simplified from [2] by using ε-input rules) constructed from A. The idea is to use two states, a_q and f_q, for every state q of A, to implement a guess-and-verify strategy. The states a_q are used to "guess" the first accepting path of A for the given input, spawning sub-computations in states f_q that verify that that no prior paths in A with that input are accepting.

Definition 10. For a pNFA $A = (Q_1, Q_2, \Sigma, q_0, \delta_1, \delta_2, F)$, the *backtracking transducer*

$$td_A = (Q, \Gamma, \Delta, \{a_{q_0}, f_{q_0}\}, \delta)$$

is defined as follows:

$$Q = \{\ a_q, f_q \mid q \in Q_1 \cup Q_2\ \},$$
$$\Gamma = \Sigma \text{ (we assume } \$ \notin \Sigma), \text{ and}$$
$$\Delta = Q_1 \cup Q_2.$$

Furthermore, δ consists of the following transitions:

(I) For $q \in Q_1$ and $\alpha \in \Sigma$:
  (A) If $\delta_1(q, \alpha) = q'$, let $a_q \xrightarrow{\alpha} q[a_{q'}]$ and $f_q \xrightarrow{\alpha} q[f_{q'}]$.
  (B) If $\delta_1(q, \alpha)$ is undefined, $f_q \xrightarrow{\alpha} q$.

(II) For $q \in Q_2$, if $\delta_2(q) = q_1 \cdots q_n$, then $a_q \to q[f_{q_1}, \ldots, f_{q_i}, a_{q_{i+1}}]$ where $0 \leq i \leq n-1$, and $f_q \to q[f_{q_1}, \ldots, f_{q_n}]$.

(III) Finally, if $q \in F$, then $a_q \xrightarrow{\$} q$, whereas when $q \in Q_1 \setminus F$, then $f_q \xrightarrow{\$} q$.

It should be clear that there is a bijective correspondence between the root-to-leaf paths in $td_A(w)$ and the runs in A on w that are of priority higher than or equal to the priority of the accepting run of A if A accepts w, and the set of all runs of A on w otherwise. It thus follows that the output size problem can be applied to determine the worst-case matching time of regex matchers.

Example 11. A pNFA A, for the regular expression $R = a^*a^*$, is given in Figure 1.

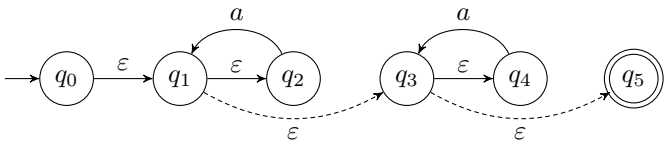

Figure 1: The prioritized non-deterministic finite automaton for the regular expression a^*a^*. For states with multiple outgoing ε-transitions (q_1 and q_3 here) the lower-priority one is indicated by the dashed arrow.

Since a regex matcher will try all possible ways of dividing the prefix a^n, in the input a^nb, between the two a^* subexpressions, the matcher will have quadratic (attempted) matching time on input a^nb. We have that $td_A = (Q, \Gamma, \Delta, \{a_{q_0}, f_{q_0}\}, \delta)$, where $\Delta = \{q_0, q_1, q_2, q_3, q_4, q_5\}$, $\Gamma = \{a, b\}$. The transition rules are:

$$
\begin{array}{llll}
a_{q_0} \to q_0[a_{q_1}], & f_{q_0} \to q_0[f_{q_1}]; & & \\
a_{q_1} \to q_1[a_{q_2}], & a_{q_1} \to q_1[f_{q_2}, a_{q_3}], & f_{q_1} \to q_1[f_{q_2}, f_{q_3}]; & \\
a_{q_2} \xrightarrow{a} q_2[a_{q_1}], & f_{q_2} \xrightarrow{a} q_2[f_{q_1}], & f_{q_2} \xrightarrow{b} q_2, & f_{q_2} \xrightarrow{\$} q_2; \\
a_{q_3} \to q_3[a_{q_4}], & a_{q_3} \to q_3[f_{q_4}, a_{q_5}], & f_{q_3} \to q_3[f_{q_4}, f_{q_5}]; & \\
a_{q_4} \xrightarrow{a} q_4[a_{q_3}], & f_{q_4} \xrightarrow{a} q_4[f_{q_3}], & f_{q_4} \xrightarrow{b} q_4, & f_{q_4} \xrightarrow{\$} q_4; \\
a_{q_5} \xrightarrow{\$} q_5, & f_{q_5} \xrightarrow{a} q_5, & f_{q_5} \xrightarrow{b} q_5. &
\end{array}
$$

An alternative interpretation of the transition rules of td_A is given in Figure 2, from which one can argue by induction that $|td_A(a^nb)|$ is quadratic in n.

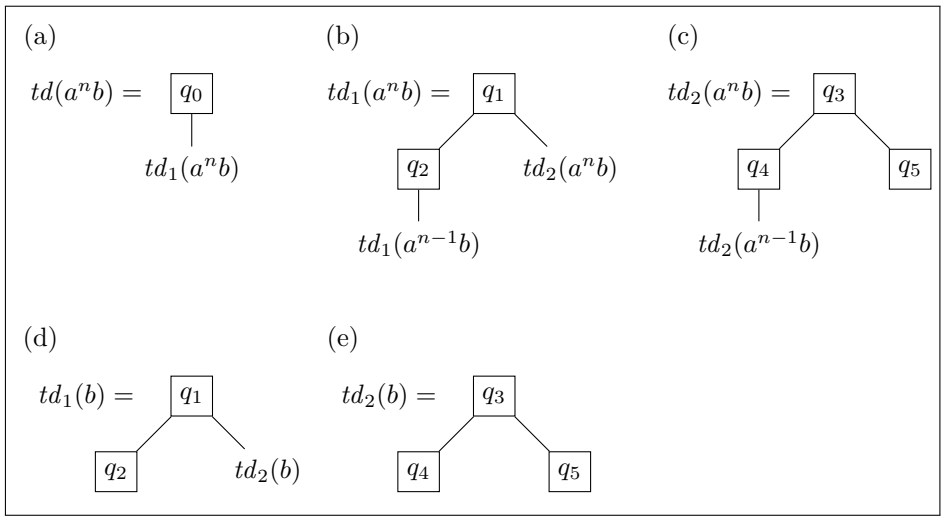

Figure 2: The transducer td_A, as presented in Example 11, defined in terms of transducers td_1 and td_2, where td_1 is td_A but with initial states $\{a_{q_1}, f_{q_1}\}$, and similarly, td_2 has $\{a_{q_3}, f_{q_3}\}$ as initial states.

To aid in the understanding of the above given transducer rules, $td_A(aab)$ is given in Figure 3.

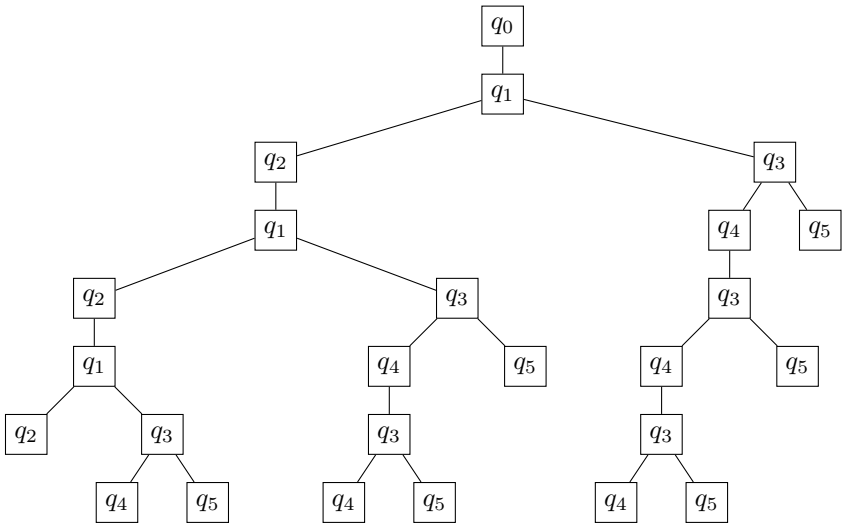

Figure 3: The output tree obtained when applying td_A, as defined in Example 11, to the string aab. Iterating the steps further, as would happen on a string with additional leading as, reveals quadratic growth.

4. NFA Ambiguity Testing

In this section, we recall and extend results, from [1], on ambiguity of NFA. This will be used in the next section to determine the degree of growth of the output size of transducers. We begin by introducing definitions related to NFA ambiguity.

Definition 12. The degree of ambiguity for $w \in \Sigma^*$, with respect to the NFA A, denoted by $d_A(w)$, is the number of accepting runs on w in A. Let

$$m_A(n) = \sup_{w \in \Sigma^*, |w| \leq n} d_A(w),$$

that is, the least upper bound of the ambiguity over all strings of length at most n, in the following.

- A has an *infinite degree of ambiguity* (IDA) if $m_A(n) \notin O(1)$ (that is, $m_A(n)$ is not bounded by any constant),
- A has an *exponential degree of ambiguity* (EDA) if $m_A(n) \in 2^{\Omega(n)}$ (that is, $m_A(n)$ grows at least exponentially),
- A has a *polynomial ambiguity of degree at least d* ($\text{IDA}_{\geq d}$) if $m_A(n) \in \Omega(n^d)$ (that is, $m_A(n)$ grows at least as quickly as some polynomial of degree d), and
- A has a *polynomial ambiguity of degree d* ($\text{IDA}_{=d}$) if $m_A(n) \in \Theta(n^d)$.

Note that, following [1], IDA actually means that A has *unbounded* ambiguity rather than $m_A(n) = \infty$ for some n, as one might believe. In fact, throughout the remainder of this paper we shall assume that $m_A(n) \neq \infty$ for all n, i. e., A does not contain any ε-cycles on useful states. We make this assumption because the case where it is not fulfilled is of little practical interest, and it is easy to check whether A contains such a cycle.

Moreover, although our NFA may contain parallel transitions for modeling reasons, we assume in the remainder of this section without loss of generality that there are no parallel transitions, i. e., $\delta(p, \alpha, q) \in \{0, 1\}$ for all $p, q \in Q$ and $\alpha \in \Sigma_\varepsilon$. This can easily be achieved without affecting $m_A(n)$ by replacing every transition by a transition on the same symbol followed by an ε-transition, with a fresh state in between.

Lemma 13. *An NFA A has*

- *IDA if and only if there exists two distinct useful states p and q and a non-empty string w such that $p \xrightarrow{w}_A p$, $p \xrightarrow{w}_A q$, and $q \xrightarrow{w}_A q$,*
- *EDA if and only if there exists useful states p, q and q' (with $q \neq q'$) and strings w and w' such that*

  $$p \xrightarrow{w}_A q,\ p \xrightarrow{w}_A q',\ q \xrightarrow{w'}_A p,\ \text{and } q' \xrightarrow{w'}_A p,$$

  note that this implies IDA (the string is $w'w$ and the two states are q and q'), and
- *IDA$_{\geq d}$ if and only if there exists useful states $p_1, q_1, \ldots, p_d, q_d$ and strings $v_1, \ldots, v_d, u_2, \ldots, u_d$, with the strings v_i non-empty, such that*

  $$p_i \neq q_i,\ p_i \xrightarrow{v_i}_A p_i,\ p_i \xrightarrow{v_i}_A q_i,\ \text{and } q_i \xrightarrow{v_i}_A q_i$$

  for all $i \in \{1, \ldots, d\}$, and $q_{i-1} \xrightarrow{u_i}_A p_i$ for $i \in \{2, \ldots, d\}$.

Proof. Lightly adjusted restatement from [1]. □

Remark 14. Note that, as it should, EDA implies IDA$_{\geq d}$ for all d, as one can take the states p, q, q' implied by EDA, let $p_1 = \cdots = p_d = q$ and $q_1 = \cdots = q_d = q'$, and $v_1 = \cdots = v_d = u_2 = \cdots = u_d = w'w$. Also, if any two states are the same in IDA$_{\geq d}$, for example if $p_i = p_{i+k}$ for $k > 0$, then we have EDA, since the loops on p_i and q_i (on the same string that can also be used to go from p_i to q_i), can be used to construct two distinct paths from p_i to $p_{i+k} = p_i$ on the same input string.

Note that this means that IDA$_{\geq d}$ implies EDA for $d \geq |Q|$. Hence, in the following we will generally assume that $d < |Q|$.

In [1] the properties above are used to prove the following lemma.

Lemma 15. *Let $A = (Q, \Sigma, I, \delta, F)$ be an NFA.*

(I) *It is decidable in time $O(|A|_\delta^2)$ whether A has EDA.*

(II) *If A does not have EDA, then it has IDA$_{=d}$ for some d, and this d can be computed in time $O(|A|_\delta^3)$.*

We now show that Lemma 13 can alternatively be used to decide these ambiguity properties in nondeterministic logarithmic space.

Lemma 16. *For an NFA A and a number $d \in \mathbb{N}$ as input, it is NL-complete to decide if A has IDA, EDA, IDA$_{\geq d}$, IDA$_{=d}$.*

Proof. Clearly all these decision problems are NL-hard, by Lemma 13 and the NL-hardness of reachability on a directed graph. Thus, it remains to be shown that it can be decided in non-deterministic logarithmic space if A has IDA, EDA, IDA$_{\geq d}$, or IDA$_{=d}$.

For states $p_1, q_1, \ldots, p_k, q_k$, write $(p_1, \ldots, p_k) \rightsquigarrow_A (q_1, \ldots, q_k)$ if there is a string w such that $p_i \xrightarrow{w}_A q_i$ for all i. Note that, for a constant k, the existence of w can be checked nondeterministically in logarithmic space. (For $k = 1$, this is just graph reachability.) From this fact, the EDA and IDA parts of the lemma obviously follow. For example, to confirm that A has IDA, nondeterministically select p and q, then check that p is reachable from an initial state, and that $(p, p, q) \rightsquigarrow_A (p, q, q)$. Finally check that final states are reachable from p and q (to establish their usefulness).

For checking IDA$_{\geq d}$ ($d < |Q|$), we use a loop over $i = 1, \ldots, d$. (Note that i can be stored in logarithmic space.) Initially, choose p (which will correspond to p_1 in Lemma 13) and check that it is reachable from some initial state. Now, for $i = 1, \ldots, d$, select some $q \neq p$ and p' (corresponding to q_i and p_{i+1}), and verify nondeterministically that $(p, p, q) \rightsquigarrow_A (p, q, q)$ and that $q \rightsquigarrow_A p'$. Then set $p \leftarrow p'$, choose new $q \neq p$ and p', and repeat. In the last step (for $i = d$), additionally make sure that p' is a final state in order to guarantee usefulness.

Finally, note that A has IDA$_{=d}$ if it has IDA$_{\geq d}$ but *not* IDA$_{\geq d+1}$, which can be decided by the previous paragraph since NL is closed under complement [9]. □

In the following section we use these results to determine the complexity of deciding the output size of non-total transducers, where an algorithm to turn non-total transducers into equivalent total transducers at the expense of obtaining an exponential blowup in state complexity, is combined with ambiguity algorithms in logarithmic space to achieve a polynomial space algorithm for deciding output size for transducers in general.

5. Linking Output Size to NFA Ambiguity

In this section, we show how the decision procedures for ambiguity of NFA (discussed in the previous section) can be used to determine the degree of os^F_{td} and os^Y_{td}, first when td is total, and finally in the general case when td is not necessarily total.

With every total deterministic transducer $td = (Q, \Gamma, \Delta, I, \delta)$, we associate NFA nfa^F_{td} and nfa^Y_{td}, for which the ambiguity of a given input string w is equal to $|td(w)|$ and $|\text{yield}(td(w))|$, respectively. If td is not total, the ambiguity values only provide upper bounds for $|td(w)|$ and $|\text{yield}(td(w))|$.

The NFA nfa^Y_{td} and nfa^F_{td} are constructed in very similar ways. Their input alphabet is Γ. The set of initial states I is the same as for td and the set of states is $Q \cup \{q_p, q_f\}$, where q_p and q_f are fresh states with q_f the only final state. The

transition functions δ_Y and δ_F of nfa^Y_{td} and nfa^F_{td} are built according to the following intuition: If td processes a symbol of rank 1 in a state q, producing the partial output t, then one "process" q_p is spawned for each occurrence of a symbol in $\Delta^{(0)}$ in the case of nfa^Y_{td}. In the case of nfa^F_{td}, the same is done for each occurrence of a symbol in Δ. Each of these will give rise to a single accepting computation without branching any further. Hence they contribute 1 to the ambiguity, in this way counting the output symbol of td that gave rise to the instance of q_p. In addition, each occurrence of a state in t gives rise to a corresponding sub-computation of nfa^Y_{td} and nfa^F_{td}, resp.

Formally, we define

$$\delta_F(q_p, \alpha, q_p) = \delta_Y(q_p, \alpha, q_p) = 1 \text{ for } \alpha \in \Gamma \text{ and}$$
$$\delta_F(q_p, \varepsilon, q_f) = \delta_Y(q_p, \varepsilon, q_f) = 1.$$

Moreover, if $F(t) = |t|_\Delta$ and $Y(t) = |t|_{\Delta^{(0)}}$ for $t \in T_\Delta(Q)$, then we let, for $\Xi \in \{F, Y\}$, $q, q' \in Q$, $\alpha \in \Gamma_\varepsilon$, and $t \in T_\Delta(Q)$,

$$\begin{aligned}
\delta_\Xi(q, \alpha, q') &= |t|_{\{q'\}} && \text{if } (q, \alpha, t) \in \delta;\\
\delta_\Xi(q, \alpha, q_p) &= \Xi(t) && \text{if } (q, \alpha, t) \in \delta;\\
\delta_\Xi(q, \varepsilon, q_f) &= \Xi(t) && \text{if } (q, \$, t) \in \delta.
\end{aligned}$$

Note that the resulting automaton is up to a constant no larger than the transducer under consideration.

Lemma 17. *Let δ, δ_F and δ_Y be the transition functions of td, nfa^F_{td} and nfa^Y_{td}, respectively. Then $|nfa^Y_{td}|_{\delta_Y} \leq |nfa^F_{td}|_{\delta_F} \leq |td|_\delta + |\Gamma| + 1$.*

Proof. By construction, at most one rule is added to the automaton for every node in the right hand side of a rule in td, with the exception of the additional transitions from q_p. □

With this construction in place we are prepared to establish the equivalence of the output size function of the transducer and the ambiguity of the constructed automaton.

Lemma 18. *Let td be a deterministic transducer. Then*

$$d_{nfa^F_{td}}(w) = os^F_{td}(w) \quad \textit{and} \quad d_{nfa^Y_{td}}(w) = os^Y_{td}(w)$$

for $w \in \Gamma^*$.

Proof. The proof is a straightforward induction on the length of the string w, verifying the following invariant for each step:

(I) The multiset of states in the current output tree matches the multiset of states (excluding q_p and q_f) which computation paths in the automaton have reached.

(II) Using that (I) holds: after any number of steps the number of output symbols (or output leaves in the case of Y) produced by the transducer so far is equal to the number of different computation paths currently in state q_p in the automaton.

Then simply note that after processing w, all paths currently in q_p take the ε-transition to q_f, thus counting in terms of ambiguity all symbols (or leaves) outputted by td in earlier steps, and to this is added the count of all symbols/leaves produced by td when applying rules of the form $q \xrightarrow{\$} t$ in the terminating step, by using $\delta_F(q, \varepsilon, q_f)$ (or $\delta_Y(q, \varepsilon, q_f)$ respectively). □

Example 19. In Figure 4, we show $nfa^F_{td_2}$ and $nfa^Y_{\overline{td_3}}$, with td_2 and $\overline{td_3}$ as in Example 8. Clearly, both $nfa^F_{td_2}$ and $nfa^Y_{\overline{td_3}}$ has IDA, and $nfa^F_{td_2}$ has EDA, whereas $nfa^Y_{\overline{td_3}}$ is polynomially ambiguous of degree 2. Thus td_2 has exponential full (and yield) output size, and the yield output size of $\overline{td_3}$ grows quadratically.

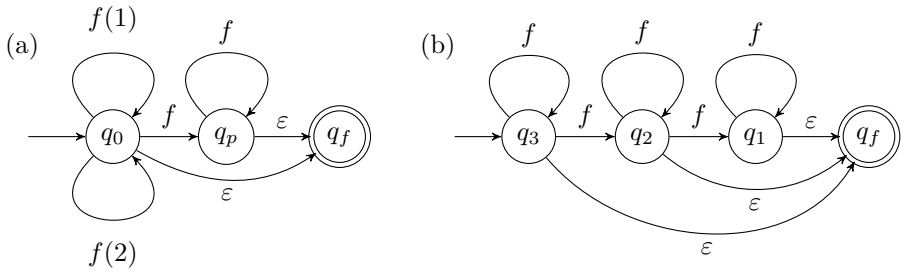

Figure 4: (a) $nfa^F_{td_2}$ and (b) $nfa^Y_{\overline{td_3}}$, with td_2 and $\overline{td_3}$ as in Example 8 (omitting the unreachable state q_p in the latter).

From [5, Lemma 3.2] it is known that, for every transducer td, one can construct a deterministic transducer td' such that, for some constant $a \in \mathbb{N}_+$, both

$$os^F_{td}(n/a) \leq os^F_{td'}(n) \leq os^F_{td}(n) \quad \text{and} \quad os^Y_{td}(n/a) \leq os^Y_{td'}(n) \leq os^Y_{td}(n)$$

for all $n \in \mathbb{N}$. The construction preserves totality, can be carried out in logarithmic space, and increases the number of transitions at most quadratically. Although this lemma is only stated and proved for full output size and transducers without ε-input rules, it is straightforward to extend the result to our more general setting. Thus, Lemma 18 can also be used in the case where td is nondeterministic. (Alternatively, the construction used in [5] can easily be incorporated into the way nfa^Y_{td} and nfa^F_{td} are built, thus avoiding the need to modify td.) Together with Lemma 15, we immediately obtain the following result on full and yield output size for transducers.

Theorem 20. *Let td be a total transducer. Then:*

(I) *It is decidable in time $O(|td|^4_\delta)$ if td has exponential full (and yield) output size.*

(II) *If the full (and yield) output size of td is not exponential (and not ∞, by our general assumption), then it is polynomial, and the degree of the polynomial growth of the full and yield output size of td can be computed in time $O(|td|^6_\delta)$.*

Remark 21. Notice since nfa^F_{td} (and nfa^Y_{td}) can be constructed in logarithmic space, the decision problems in Theorem 20 are in NL by applying Lemma 16. NL-hardness for any of the output size problems (for total transducers) in Theorem 20 is obtained as follows. Let td' be a total string-to-tree transducer with any output size property we want to show is NL-hard to decide. Take a directed graph G for which we want to decide if we can go from node p to q (i. e., an instance of the NL-hard graph reachability problem). Now let td be a transducer which first traverses edges in G starting at p, one edge for each rank 1 input symbol consumed, until q is reached. While traversing G, td simply deletes input symbols. Also, td produces $\$^{(0)}$ as output if the input string is consumed before q is reached. Once q is reached by td, it starts behaving like td'. Thus td either produces only $\$^{(0)}$ as output, or if q is reachable from p, will have the same output size behaviour as td'. Thus the decision problems in Theorem 20 are NL-complete.

We now turn to the case where td is not necessarily total. Given a transduction $\tau\colon \Gamma^* \to T_\Delta$ and a set $S \subseteq \Gamma^*$, let us denote the domain of τ by $dom(\tau)$, that is

$$dom(\tau) = \{\ w \in \Gamma^* \mid \tau(w) \neq \emptyset\ \},$$

and the domain restriction of τ to S by $\tau|_S$. We describe a relatively straightforward construction to turn a deterministic transducer td into a total deterministic transducer td^T, such that

$$\begin{aligned} dom(td) &\subseteq dom(td^T), \\ td^T|_{dom(td)} &= td, \text{ and} \\ os^F_{td}(n) &\leq os^F_{td^T}(n) \leq k \cdot os^F_{td}(n+k), \end{aligned}$$

for all $n \in \mathbb{N}$, where k is a constant determined by td, and similarly for os^Y. Thus, let

$$td = (Q, \Gamma, \Delta, I, \delta) \quad \text{and} \quad td^T = (Q_T, \Gamma, \Delta, I_T, \delta_T),$$

where Q_T, I_T and δ_T are defined below. First define $\bar{\delta}\colon 2^Q \times \Gamma_\varepsilon \to 2^Q$ by

$$\bar{\delta}(S, \alpha) = \bigcup_{(q,\alpha,t)\in\delta, q\in S} Q_t,$$

where Q_t denotes the set of states appearing in t, i. e., it is the smallest subset of Q such that $t \in T_\Delta(Q_t)$. Also, for $S \in 2^Q$, let

$$dom_S(td) = \bigcap_{q\in S} dom(td_q),$$

where $td_q = (Q, \Gamma, \Delta, \{q\}, \delta)$. Fix $a_0 \in \Delta^{(0)}$.[1] We let

$$\begin{aligned} Q_T &= \{\ (q,S) \mid q \in Q, S \in 2^Q\ \}, \\ I_T &= \{\ (q,\{q\}) \mid q \in I\ \}, \text{ and} \\ \delta^T &= \{\ ((q,S),\alpha,\delta_S(t)) \mid (q,\alpha,t) \in \delta, dom_S(td) \neq \emptyset\ \} \\ &\quad \cup \{\ ((q,S),\alpha,a_0) \mid dom_S(td) = \emptyset\ \}. \end{aligned}$$

where $\delta_S(t)$ denotes the tree obtained from t by replacing every occurrence of a state $q' \in Q$ by $(q', \bar{\delta}(S))$.

Theorem 22. *The transducer td^T obtained from a deterministic transducer td, described as above, is a total deterministic transducer with*

$$os^F_{td}(n) \leq os^F_{td^T}(n) \leq k \cdot os^F_{td}(n+k),$$

and

$$os^Y_{td}(n) \leq os^Y_{td^T}(n) \leq k \cdot os^Y_{td}(n+k),$$

for all $n \in \mathbb{N}$, where k is a constant determined by td. The construction can be performed using a worktape of polynomial size (i. e., not counting the exponentially large output tape, as usual).

Proof. We only consider

$$os^F_{td}(n) \leq os^F_{td^T}(n) \leq k \cdot os^F_{td}(n+k);$$

a similar argument works for $os^Y_{td}(n) \leq os^Y_{td^T}(n) \leq k \cdot os^Y_{td}(n+k)$. By induction on the length of input strings, it can be established that if $(q_1, S_1), \ldots, (q_k, S_k)$ are the states on the leaves of a tree obtained at an intermediate step when applying td^T, then $S_1 = \cdots = S_k = \{q_1, \ldots, q_k\}$. Note that the transduction steps on an input string w are identical when using td and td^T if we disregard the second components in the states (q, S) of td^T, except when an S is encountered such that $dom_S(td) = \emptyset$. If that happens, it shows that $td(w) = \emptyset$, whereas td^T immediately produces a tree in T_Δ when processing the next input symbol. From this follows that $os^F_{td}(n) \leq os^F_{td^T}(n)$. Let now $k' \in \mathbb{N}$ be such that $|w_S| \leq k'$ for all shortest strings w_S in $dom_S(td)$ with $dom_S(td) \neq \emptyset$. Then if $w = w'aw''$ and a state (q, S) with $dom_S(td) = \emptyset$ is reached after processing the prefix $w'a$ by td^T, we select a string $\bar{w}$, as short as possible, such that $w\bar{w} \in dom(td)$. We fix $k \in \mathbb{N}$ larger than k' and all right hand sides of transduction rules of td. Then $|td^T(w'a)| \leq k|td(w\bar{w})|$, and thus in general,

$$os^F_{td^T}(n) \leq k \cdot os^F_{td}(n+k).$$

The construction can be performed in polynomial space in a straightforward way by implementing a loop that outputs one transition rule at a time. Simply note that, for this, it suffices to keep a constant number variables holding states of td^T, and each such state can be represented by $O(|Q|)$ bits as it is essentially a subset of Q. □

[1] We may assume that $\Delta^{(0)} \neq \emptyset$ because otherwise td computes the empty transduction and all questions discussed here become trivial.

With this construction in hand we can determine output size for (non-total) transducers in polynomial space. Before that theorem however, we first restate for clarity a folklore fact about compositions of space-bounded Turing machines.

Lemma 23. *Let $f\colon \Sigma^* \to \Delta^*$ be a function that is computable in polynomial space, and let $g\colon \Delta^* \to \{0,1\}$ be (the characteristic function of) a decision problem in NL. Then the decision problem $g \circ f$ is in PSPACE.*

Proof. Let T_1 and T_2 be Turing machines computing f and g, respectively. By Savitch's theorem, it suffices to show how to compose T_1 and T_2 into a nondeterministic Turing machine running in polynomial space.

We, without loss of generality, assume that T_1 and T_2 do not step outside the useful part of their output- and input tape, respectively. Then construct T_3 as follows. First allocate room on the tape for two binary counters c_1 and c_2, and a single tape symbol v. Set $c_1 \leftarrow 1$ and $c_2 \leftarrow 1$.

Then simulate a run of T_1 to determine the output symbol at position c_2 (which, at the same time, is the input symbol of T_2 at that position). The simulation uses the input tape as usual, but each time T_1 would move the output write head, instead update the counter c_1 accordingly (i. e., $c_1 \leftarrow c_1 + 1$ on a step right, $c_1 \leftarrow c_1 - 1$ on a step left), and each time T_1 would write to the output tape, instead save the value in v if $c_1 = c_2$, and ignore the write otherwise. That is, the output symbol which would end up at position c_2 ends up being written to v.

Finally simulate a full run of T_2, except using v as the input symbol read in each step, and whenever T_2 would move the input read head instead; update c_2 accordingly, set $c_1 \leftarrow 1$, and perform a full simulation of T_1 as described above (up to the point where it updates v).

In this way, T_3 will produce the output T_2 would have produced with the input given by the output of T_1. The total space used is the space required for one run of T_1 (as it can be reused), one run of T_2, the two counters, the one symbol for v, and possibly some small space for bookkeeping information. As the output of a (terminating) Turing machine with a worktape of size $s(n)$ can be of size at most $k^{s(n)}$ for some constant k, a polynomial number of bits is sufficient to store c_1 and c_2. Hence, T_3 runs in polynomial space. □

Theorem 24. *Let td be a transducer. Then it is possible to decide in polynomial space if:*

(I) *td has exponential full (and yield) output size, and*

(II) *whether the degree of the polynomial equals d (in the case of polynomial output size).*

Proof. Apply Lemma 23 to Theorem 22 and Theorem 20. □

We complete this section by showing that the decision problems of Theorem 24 are in fact PSPACE-complete, by a straightforward reduction from the PSPACE-complete problem of deciding non-emptiness of intersection of a variable number of DFA.

Theorem 25. *The decision problems of Theorem 24 are PSPACE-complete.*

Proof. This can be shown using a reduction from the problem of deciding whether

$$\bigcap_{i=1}^{n} \mathcal{L}(A_i) \neq \emptyset,$$

where $A_1, \ldots, A_n$ are a variable number of DFA over a common alphabet Σ, a problem which is known to be PSPACE-complete [10].

Let td_G be a transducer over an input alphabet Δ disjoint with Σ, which exhibits the output size property we wish to demonstrate has a PSPACE-hard decision problem. Such a transducer is easy to construct – consider and modify Example 8 appropriately.

We want to combine td_G and (transducers corresponding to) $A_1, \ldots, A_n$ in such a way that the resulting transducer td retains the output size property of td_G if and only if

$$\bigcap_{i=1}^{n} \mathcal{L}(A_i) \neq \emptyset.$$

In a first step, extend each A_i so that it recognizes $\mathcal{L}(A_i)\Delta^*$. Similarly, extend td_G so that $td_G(uv) = td_G(v)$ for all $u \in \Sigma^*, v \in \Delta^*$. (Thus, td_G simply skips an initial prefix in Σ^* without producing output.)

Now, let $a^{(0)}$ be a rank 0 symbol in the output alphabet of td_G (which we may assume to exist). For each i, let td_{A_i} be a transducer such that $dom(td_{A_i}) = \mathcal{L}(A_i)$ and $td_{A_i}(w) = a^{(0)}$ if $w \in \mathcal{L}(A_i)$. Finally construct td by taking the disjoint union

$$td_G \cup td_{A_1} \cup \cdots \cup td_{A_n}$$

(defined in the obvious way) and adding a new state q_0, which is the only initial state of td. Let $f^{(n+1)}$ be an additional output symbol and add $(q_0, \varepsilon, f[q_G, q_{A_1}, \ldots, q_{A_n}])$ to the transition relation for td, where q_G and q_{A_i} are initial states for td_G and td_{A_i}.

Complete the proof by observing that

$$dom(td) = \emptyset \quad \text{if} \quad \bigcap_{i=1}^{n} \mathcal{L}(A_i) = \emptyset,$$

and thus td has $\mathrm{IDA}_{=0}$ in this case. Otherwise, let $u \in \Sigma^*$ be a shortest string in $\bigcap_{i=1}^{n} \mathcal{L}(A_i)$. Then

$$td(uv) = \{\ f[t, a, \ldots, a] \mid t \in td_G(t)\ \},$$

which means that td has the same output size property as td_G if and only if

$$\bigcap_{i=1}^{n} \mathcal{L}(A_i) \neq \emptyset,$$

as long as we are not considering $\mathrm{IDA}_{\geq 0}$ or $\mathrm{IDA}_{=0}$. For the latter cases, simply reduce the intersection emptiness problem (rather than the non-emptiness problem) in the same way as above, but using a td_G which has EDA. □

6. Conclusions and Future Work

In [2, 3] an investigation into the time complexity and semantics of backtracking regular expression matchers was initiated. Starting from a regular expression E, the question asked is how efficient (or inefficient) the corresponding matcher is. It was shown that, given E, one can construct a transducer td_E such that, for every input string w, $td_E(w)$ represents the computation tree of the matcher. Hence, to know the full output size of td_E is to know the running time of the matcher. Since not all string-to-tree transducers are necessarily obtained through regular expressions, the results in this paper can only be used to conclude that the worst-case matching time problem for regular expressions is in PSPACE. Determining if it is also PSPACE-hard, is left as future work. It may also be of interest to refine the results of Theorem 25 into lower bounds (stated in terms of, e. g., number of states and the structure of output trees) for algorithms when assuming the exponential time hypothesis, using analogous results from [8].

References

[1] C. Allauzen, M. Mohri, A. Rastogi, General algorithms for testing the ambiguity of finite automata. In: M. Ito, M. Toyama (eds.), *Proc. 12th Intl. Conf. on Developments in Language Theory (DLT 2008)*. LNCS 5257, 2008, 108–120.

[2] M. Berglund, F. Drewes, B. van der Merwe, Analyzing catastrophic backtracking behavior in practical regular expression matching. In: Z. Ésik, Z. Fülöp (eds.), *Proc. 14th Intl. Conf. on Automata and Formal Languages (AFL 2014)*. EPTCS 151, 2014, 109–123.

[3] M. Berglund, B. van der Merwe, On the semantics of regular expression parsing in the wild. In: F. Drewes (ed.), *Proc. 20th Intl. Conf. on Implementation and Application of Automata (CIAA 2015)*. LNCS 9223, 2015, 292–304.

[4] M. Berglund, B. van der Merwe, On the semantics of regular expression parsing in the wild. *Theoretical Computer Science* **679** (2017), 69–82.

[5] F. Drewes, The complexity of the exponential output size problem for top-down and bottom-up tree transducers. *Information and Computation* **169** (2001) 2, 264–283.

[6] F. Drewes, J. Engelfriet, Branching synchronization grammars with nested tables. *Journal of Computer and System Sciences* **68** (2004) 3, 611–656.

[7] J. Engelfriet, Surface tree languages and parallel derivation trees. *Theoretical Computer Science* **2** (1976) 1, 9–27.

[8] H. Fernau, A. Krebs, Problems on finite automata and the exponential time hypothesis. In: Y. Han, K. Salomaa (eds.), *Proc. 21st Intl. Conf. on Implementation and Application of Automata (CIAA 2016)*. LNCS 9705, 2016, 89–100.

[9] N. Immerman, Nondeterministic space is closed under complementation. *SIAM Journal on Computing* **17** (1988) 5, 935–938.

[10] K.-J. Lange, P. Rossmanith, The emptiness problem for intersections of regular languages. In: *Intl. Symposium on Mathematical Foundations of Computer Science*. 1992, 346–354.

[11] G. Rozenberg, A. Salomaa (eds.), *Handbook of Formal Languages, Vol. 1: Word, Language, Grammar*. Springer-Verlag New York, 1997.

[12] B. van der Merwe, N. Weideman, F. Drewes, The output size problem for string-to-tree transducers. In: A. Maletti (ed.), *Proc. 4th Intl. Workshop on Trends in Tree Automata and Tree Transducers (TTATT 2016)*. 2016, 43–50.

(Received: March 8, 2017; revised: November 19, 2017)