Largest Common Prefix of a Regular Tree Language

Markus Lohrey[a], Sebastian Maneth[b]

[a] *University of Siegen, Germany*
[b] *University of Bremen, Germany*

Abstract

A family of tree automata of size n is presented such that the size of the largest common prefix (lcp) tree of all accepted trees is exponential in n. Moreover, it is shown that this prefix tree is not compressible via DAGs (directed acyclic graphs) or tree straight-line programs. We also show that determining whether or not the lcp trees of two given tree automata are equal is coNP-complete; the result holds even for deterministic bottom-up tree automata accepting finite tree languages. These results are in sharp contrast to the case of context-free string grammars.

1. Introduction

For a given language L of finite strings one can define the *largest common prefix* of L as the longest string which is a prefix of every word in L. This definition can be extended to tree languages in a natural way. One motivation to compute the largest common prefix of a set of strings or trees is the so called earliest normal form, which has been studied for string transducers [2, 11] and tree transducers [4]. The existence of an earliest normal form has several important consequences. For instance, the transducer can in a simple further step be made canonical, which allows deciding equivalence and gives rise to Gold-style learning algorithms [12, 7]. Intuitively, an earliest transducer produces its output "as early as possible". In order to compute the earliest form of a given transducer, one has to consider all possible inputs (for a certain set of states), and has to determine if the corresponding outputs have a non-empty common prefix; if so, then the transducer is *not* earliest, because this common prefix is independent of the input and hence should have been produced before. The questions arise how large such common prefixes can possibly be, and whether or not they can be compressed.

In this paper we address these questions in a general setting where the trees of which the common prefix is computed are given by a finite tree automaton. We present a family of tree automata of size $\Theta(n)$ such that their largest common prefixes (lcps) are of size exponential in n and are essentially incompressible via common tree compression methods such as DAGs (directed acyclic graphs) or tree straight-line programs [6, 9]. Technically, the main ingredient in the construction of this family of tree automata is the well-known fact that

Email addresses: lohrey@eti.uni-siegen.de (Markus Lohrey), maneth@uni-bremen.de (Sebastian Maneth)

a deterministic string automaton needs exponentially many states in order to recognize strings where the n-th last symbol carries a specific label. Recently it has been shown that for a given context-free string grammar, a representation of the largest common prefix can be computed in polynomial time [10].

Whenever above we mention "tree automaton", we always mean "deterministic bottom-up tree automaton". These automata recognize exactly the regular tree languages. Let us now consider the case of *deterministic top-down tree automata*. It is known that these automata are strictly less expressive than deterministic bottom-up tree automata; in fact, they are so weak that they cannot even recognize finite tree languages such as $\{f(a, f(a, a)), f(f(a, a), a)\}$ (here, we use the standard term representation for trees; see Section 2). It turns out that the largest common prefix of the trees recognized by a deterministic top-down tree automaton can be computed by a simple (top-down) procedure. Moreover, the resulting lcps are compressible via DAGs, and the procedure can produce in linear time a DAG of the lcp.

We then address a second important problem for largest common prefixes given by tree automata, namely to determine whether or not the largest common prefixes of two given tree automata coincide. Note that when constructing an earliest canonical ("minimal") transducer, we need to determine whether two given states are equivalent; for this to hold, several lcps must be checked for equality. The following question arises: what is the precise complexity of checking equality of the lcps of two given tree automata? In this paper, we prove that this problem is **coNP**-complete using a reduction from the complement of 3-SAT.

An extended abstract of this paper appeared in [8].

2. Preliminaries

We assume that the reader is familiar with words and finite automata on words. A language $L \subseteq \{0,1\}^*$ is a *right-ideal* if $L = L\{0,1\}^*$. A set $S \subseteq \{0,1\}^*$ is *prefix-closed* if $uv \in S$ implies that $u \in S$ for all $u, v \in \{0,1\}^*$. Note that L is a right-ideal if and only if $\{0,1\}^* \setminus L$ is prefix-closed.

A DFA (deterministic finite automaton) over a finite alphabet Γ is a 5-tuple $A = (Q, \Gamma, q_0, F, \delta)$, where Q is the finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta : Q \times \Gamma \to Q$ is the transition mapping. The language $L(A)$ accepted by A is defined in the usual way. For an NFA (non-deterministic finite automaton) we have a set $I \subseteq Q$ of initial states and the transition function δ maps from $Q \times \Gamma$ to 2^Q (the powerset of Q).

We consider finite *binary trees* that are unlabeled, rooted, and ordered. The latter means that there is an order on the children of a node. Moreover, every node is either a leaf or has exactly two children. We will use two equivalent formalizations of such trees. We can view them as formal expressions over the set of function symbols $\{f, a\}$, where f gets two arguments and a is a constant-symbol (i.e., gets no arguments). The set of all such expressions is denoted by T_2 and is inductively defined by the following conditions: $a \in T_2$ and if $t_1, t_2 \in T_2$ then also $f(t_1, t_2) \in T_2$. Trees from T_2 are binary trees, where each leaf is labeled with a and every internal node is labeled with f. Obviously, the labeling bears no information, and trees from T_2 can be identified with unlabeled binary trees. For instance, the expression $f(f(a, a), a)$ represents the

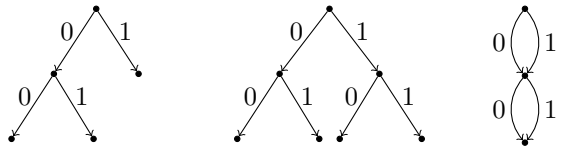

Figure 1: Trees $t_0 = f(f(a,a),a)$ (left), $t_1 = f(f(a,a),f(a,a))$ (midle) and the minimal DAG of t_1 (right).

binary tree t_0 from Figure 1. Alternatively, we can specify a binary tree by a path language. A *path language* P is a finite non-empty subset of $\{0,1\}^*$ such that

- P is prefix-closed and
- for every $w \in \{0,1\}^*$, $w0 \in P$ if and only if $w1 \in P$.

A binary tree $t \in T_2$ can be uniquely represented by a path language $P(t)$, and vice versa. Formally, we define $P(t)$ inductively as follows:

- $P(a) = \{\epsilon\}$
- $P(f(t_1,t_2)) = \{\epsilon\} \cup \{iw \mid i \in \{0,1\}, w \in P(t_i)\}$.

For instance, for the binary tree t_0 from Figure 1 we have $P(t_0) = \{\epsilon, 0, 1, 00, 01\}$. The nodes of t can be identified with the words in $P(t)$. The root of a tree corresponds to the empty word ϵ, $u0$ denotes the left child of u, and $u1$ denotes the right child of u. The leaves of t correspond to those words in $P(t)$ that are maximal with respect to the prefix relation. The *depth* of $t \in T_2$ can be defined as the maximal length of a word in $P(t)$. Note that the intersection of an arbitrary number of path languages is again a path language.

A *nondeterministic top-down tree automaton* (NTTA for short) is a 4-tuple $B = (Q, I, F, \delta)$, where Q is a finite set of states, $I \subseteq Q$ with $I \neq \emptyset$ is the set of initial states, $F \subseteq Q$ is the set of final states, and $\delta : Q \to 2^{Q^2}$ is the transition function (here and in the following we view elements of Q^2 as words of length two over the alphabet Q). A run of B on a tree t is a mapping $\rho : P(t) \to Q$ such that:

- If $v \in P(t)$ is a leaf of t, then $\rho(v) \in F$.
- If $v, v0, v1 \in P(t)$ with $\rho(v) = p$, $\rho(v_0) = p_0$ and $\rho(v_1) = p_1$ then $p_0p_1 \in \delta(p)$.

For $q \in Q$, we let $T(B,q)$ denote the set of all trees t for which there exists a run ρ of B such that $\rho(\varepsilon) = q$. Finally we define $T(B) = \bigcup_{q \in I} T(B,q)$ as the tree language accepted by B.

An NTTA $B = (Q, I, F, \delta)$ is called *productive* if $T(B,q) \neq \emptyset$ for every $q \in Q$. From a given NTTA B with $T(B) \neq \emptyset$ one can construct in polynomial time an equivalent productive NTTA B'. One first computes in polynomial time the set $P = \{p \in Q \mid T(B,p) \neq \emptyset\}$. Note that $F \subseteq P$. Then B' is obtained from B by removing all states from $Q \setminus P$. To do this, one also has to replace every set $\delta(q)$ $(q \in P)$ by $\delta(q) \cap P^2$.

A *deterministic top-down tree automaton* (DTTA for short) is a 4-tuple $B = (Q, q_0, F, \delta)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\delta : Q \to Q^2$ is the transition function. We can identify this 4-tuple with the NTTA $(Q, \{q_0\}, F, \delta')$ where $\delta'(q) = \{\delta(q)\}$. This allows us the transfer all definitions from NTTAs to DTTAs.

Finally a *deterministic bottom-up tree automaton* (DBTA for short) is an NTTA $B = (Q, I, F, \delta)$ such that $|F| = 1$ and for every $q_1q_2 \in Q^2$ there is at most one $q \in Q$ such that $q_1q_2 \in \delta(q)$. In other words, the sets $\delta(q)$ $(q \in Q)$ are pairwise disjoint. This allows defining a partially defined inverse δ^{-1} of δ by $\delta^{-1}(q_1q_2) = q$ if $q_1q_2 \in \delta(q)$. For every tree t there is at most one run of B on t and this run ρ can be constructed bottom-up by first setting $\rho(u) = q_f$ for every leaf u of t, where q_f is the unique state in F. Then, for all $v, v0, v1 \in P(t)$ such that $\rho(v0)$ and $\rho(v1)$ have been already defined, one sets $\rho(v) = \delta^{-1}(\rho(v0)\rho(v1))$.

It is well known that for every NTTA there exists an equivalent DBTA accepting the same tree language. On the other hand, there exist NTTAs which do not have an equivalent DTTA; see [3] for examples.

For a binary tree t with path language $P(t)$ and $u \in P(t)$ we define $P(t, u) = \{v \in \{0, 1\}^* \mid uv \in P(t)\}$. Then $P(t, u)$ is a path language too. The unique binary tree with path language $P(t, u)$ can be viewed as the subtree of t rooted in u. We can then define an equivalence relation $\equiv_t$ on $P(t)$ by $u \equiv_t u'$ if and only if $P(t, u) = P(t, u')$. Thus, the index of $\equiv_t$ is the number of pairwise non-isomorphic subtrees of t. The *minimal DAG* (*minimal directed acyclic graph*) for a tree $t \in T_2$ is obtained by keeping for every equivalence class C of $\equiv_t$ only one representative $u \in C$ to which all tree edges that point to nodes from C are redirected (if we keep for every equivalence class C of $\equiv_t$ at least one representative we obtain what is called a DAG for t). An example for a minimal DAG can be found in Figures 2 and 3. The size of the minimal DAG of t (measured in number of nodes) is exactly the number of pairwise non-isomorphic subtrees of t. There is a straightforward correspondence between minimal DAGs and minimal DFAs for the path language:

Lemma 2.1. *Let t be a tree. The following statements are equivalent:*

1. *The minimal DAG for the tree t has n nodes.*
2. *The minimal DFA for the path language $P(t)$ has $n + 1$ states.*

Proof. The proof of the lemma is straightforward: consider the minimal DAG D for the tree t, and let n be its size. It yields a DFA for $P(t)$ with $n + 1$ states by taking the root node as the initial state, all other nodes as final states and adding a failure state (note that a DFAs has a totally defined transition mapping according to our definition). Vice versa, a DFA A for $P(t)$ yields a DAG for t by removing the failure state of A (the resulting graph is clearly acyclic). □

Largest common prefix tree. Consider a non-empty tree language $L \subseteq T_2$. The *largest common prefix* $\mathrm{lcp}(L)$ of L is the unique binary tree t such that $P(t) = \bigcap_{t \in L} P(t)$. For instance, for $L = \{f(f(a, a), a), f(a, a)\}$ we obtain $\mathrm{lcp}(L) = f(a, a)$.

Lemma 2.2. *Assume that B is an NTTA with n states and such that $T(B) \neq \emptyset$. Then every word $w \in P(\mathrm{lcp}(T(B))) = \bigcap_{t \in T(B)} P(t)$ has length at most $n - 1$, i.e., the depth of $\mathrm{lcp}(T(B))$ is at most $n - 1$.*

Proof. It well-known that B must accept a tree t of depth at most $n-1$; see e.g. [3, Corollary 1.2.3]. Hence, $|w| \leq n-1$ for every word $w \in P(t)$. This implies the statement of the lemma. □

It is straightforward extend all the notions from this section to labelled binary trees. A Σ-labelled binary tree can be defined as a pair (P, λ) where $P \subseteq \{0,1\}^*$ is a path language and $\lambda : P \to \Sigma$ is the labelling function. Given a set L of Σ-labelled binary trees, one can define its lcp as the unique tree (P, λ) where P is the largest (with respect to inclusion) path language such that for all $(P', \lambda') \in L$ we have: $P \subseteq P'$ and $\lambda(u) = \lambda'(u)$ for all $u \in P$. All results in this paper also hold for Σ-labelled binary trees. Since the focus of this paper is on lower bounds, we decided to restrict our considerations to unlabelled trees.

3. From NTTAs to DFAs

In this and the next section we establish a correspondence between largest common prefix trees of regular tree languages and finite automata (on words).

Let $B = (Q, I, F, \delta)$ be a productive NTTA. Our goal is to come up with a DFA (working on strings) that recognizes the intersection of all path languages $P(t)$, where t ranges over all trees accepted by B. For this we extend $\delta : Q \to 2^{Q^2}$ to $\hat{\delta} : 2^Q \to 2^{Q^2}$ by setting $\hat{\delta}(Q') = \bigcup_{p \in Q'} \delta(p)$ for $Q' \subseteq Q$. For a state pair $p_0p_1 \in Q^2$ and $i \in \{0,1\}$ we define the projection $\pi_i(p_0p_1) = p_i$. For a set $S \subseteq Q^2$ and $i \in \{0,1\}$ we define $\pi_i(S) = \{\pi_i(pq) \mid pq \in S\}$.

We fix a fresh state q_f that does not belong to Q and define the DFA B^s (s for string) by

$$B^s = (2^Q \setminus \{\emptyset\} \uplus \{q_f\}, \{0,1\}, I, 2^Q \setminus \{\emptyset\}, \delta^s)$$

($\uplus$ denotes disjoint union) where for all $Q' \subseteq Q$ with $Q' \neq \emptyset$ and $i \in \{0,1\}$ we set

$$\delta^s(Q', i) = \begin{cases} \pi_i(\hat{\delta}(Q')) & \text{if } Q' \cap F = \emptyset \\ q_f & \text{if } Q' \cap F \neq \emptyset. \end{cases}$$

Moreover, $\delta^s(q_f, 0) = \delta^s(q_f, 1) = q_f$. The state q_f is called the failure state of B^s. Note that if $Q' \cap F = \emptyset$ for $Q' \neq \emptyset$, then the productivity of B implies that $\hat{\delta}(Q') \neq \emptyset$. In particular, we have $\pi_i(\hat{\delta}(Q')) \neq \emptyset$ if $Q' \neq \emptyset$, which implies that δ^s is well-defined.

Lemma 3.1. *Let B be a productive NTTA. Then*

$$L(B^s) = P(\mathrm{lcp}(T(B))) = \bigcap_{t \in T(B)} P(t).$$

Proof. Consider a word $w = a_1a_2 \cdots a_n$ with $a_1, \ldots, a_n \in \{0,1\}$. Let us first assume that $w \in L(B^s)$ and let $t \in T(B)$. We have to show that $w \in P(t)$. In order to get a contradiction, assume that $w \notin P(t)$. Let v be a longest prefix of w that belongs to $P(t)$. Since $\varepsilon \in P(t)$, v is well-defined. Clearly, v is a proper prefix of w and v is a leaf of t. Thus, we can write v as $v = a_1a_2 \cdots a_k$ for $k < n$. Fix a run ρ of B on t such that $\rho(\epsilon) \in I$. Let $q_i = \rho(a_1 \cdots a_i)$ for $0 \leq i \leq k$. Since v is a leaf of t we have $q_k \in F$. Since $w \in L(B^s)$ there exists a path

$$I = Q_0 \xrightarrow{a_1} Q_1 \xrightarrow{a_2} Q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} Q_n,$$

where, for $0 \leq i \leq n$, $Q_i \subseteq Q$ and $Q_i \neq \emptyset$, and for $0 \leq i \leq n-1$, $Q_i \cap F = \emptyset$ and $Q_{i+1} = \pi_{a_i}(\hat{\delta}(Q_i))$. The latter point implies by induction on i that $q_i \in Q_i$ for $0 \leq i \leq k$. Since $Q_k \cap F = \emptyset$, we must have $q_k \notin F$, which is a contradiction.

Now assume that $w \notin L(B^s)$. Hence, the unique run of B^s on w ends in the failure state q_f. Thus, there must exist a proper prefix $v = a_1 \cdots a_k$ of w such that $k < n$ and the run of B^s on w has the form

$$I = Q_0 \xrightarrow{a_1} Q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_k} Q_k \xrightarrow{a_{k+1}} q_f \xrightarrow{a_{k+2}} \cdots \xrightarrow{a_n} q_f.$$

where $Q_i \subseteq Q$ and $Q_i \neq \emptyset$ for $0 \leq i \leq k$, $Q_i \cap F = \emptyset$ and $Q_{i+1} = \pi_{a_i}(\hat{\delta}(Q_i))$ for $0 \leq i \leq k-1$, and $Q_k \cap F \neq \emptyset$. Let $q_k \in Q_k \cap F$.

We have to construct a tree $t \in T(B)$ such that $w \notin P(t)$. For this we choose states $q_i \in Q_i$ for $0 \leq i \leq k$. The state $q_k \in Q_k \cap F$ has already been chosen in the last paragraph. Assume that $q_{i+1} \in Q_{i+1}$ has been defined for some $0 \leq i \leq k-1$. To define q_i note that $q_{i+1} \in \pi_{a_i}(\hat{\delta}(Q_i))$. Hence, there exist states $p \in Q_i$ and $q'_{i+1} \in Q$ such that the following holds: if $a_i = 0$ then $q_{i+1}q'_{i+1} \in \delta(p)$ and if $a_i = 1$ then $q'_{i+1}q_{i+1} \in \delta(p)$. We set $q_i = p$. By the productivity of B there exist trees $t'_i \in T(B, q'_i)$ for $1 \leq i \leq k$. Moreover, since $q_k \in Q_k \cap F$, the one-node tree a belongs to $T(B, q_k)$. From the trees $t'_1, \ldots, t'_k, a$ we can now construct a tree $t \in T(B)$ such that $v = a_1 \cdots a_k$ is a leaf of t (and hence $w \notin P(t)$). For instance, if $v = 1^k$ then we take $t = f(t'_1, f(t'_2, f(t'_3, \cdots f(t'_k, a) \cdots)))$. For the general case, we define trees $t_0, t_1, \ldots, t_k$ inductively as follows:

- $t_k = a$,
- $t_i = f(t_{i+1}, t'_{i+1})$ if $0 \leq i \leq k-1$ and $a_{i+1} = 0$, and
- $t_i = f(t'_{i+1}, t_{i+1})$ if $0 \leq i \leq k-1$ and $a_{i+1} = 1$.

Finally, let $t = t_0$. Then t has the desired properties. □

Note that the size of the above DFA B^s is exponential in the size of B. In the case where we start with a DTTA, we can easily modify the above construction in order to construct in linear time a DFA (of linear size). Hence, let us redefine for a DTTA $B = (Q, q_0, F, \delta)$ the DFA $B^s = (Q \uplus \{q_f\}, \{0,1\}, q_0, Q, \delta^s)$ by setting for all $q \in Q$ and $i \in \{0,1\}$:

$$\delta^s(q,i) = \begin{cases} \pi_i(\delta(q)) & \text{if } q \notin F \\ q_f & \text{if } q \in F. \end{cases}$$

Moreover, $\delta^s(q_f, 0) = \delta^s(q_f, 1) = q_f$. The proof of the following lemma is similar to the proof for Lemma 3.1.

Lemma 3.2. *Let B be a DTTA with $T(B) \neq \emptyset$. Then*

$$L(B^s) = P(\mathrm{lcp}(T(B))) = \bigcap_{t \in T(B)} P(t).$$

4. From NFAs to NTTAs

We now consider NFAs that generate languages L over $\{0,1\}$ such that the complement of L is a finite path language. An NFA $A = (Q, \{0,1\}, I, F, \delta)$ is *well-behaved* if there are two different states $q_e, q_f \in Q$ such that

1. $F = \{q_f\}$ and $q_f \notin I$,
2. $\delta(q, a) \neq \emptyset$ for all $q \in Q$ and all $a \in \{0, 1\}$,
3. $q_f \notin \delta(q, a)$ for all $q \in Q \setminus \{q_e, q_f\}$ and all $a \in \{0, 1\}$,
4. $\delta(q_e, 0) = \delta(q_e, 1) = \delta(q_f, 0) = \delta(q_f, 1) = \{q_f\}$,
5. the NFA obtained from A by removing the state q_f is acyclic, and
6. all states are reachable from I.

In a well-behaved NFA A every path of length at least $|Q| - 1$ that starts in a state $q \neq q_f$ must visit q_e (this follows from points 2 and 5). Moreover, the complement $\{0, 1\}^* \setminus L(A)$ is a path language.

From a well-behaved NFA $A = (Q, \{0, 1\}, I, \{q_f\}, \delta)$ we construct the NTTA $A^t = (Q \setminus \{q_f\}, I, \{q_e\}, \delta^t)$ (t for tree) with

- $\delta^t(q) = \{q_1 q_2 \mid q_1 \in \delta(q, 0), q_2 \in \delta(q, 1)\}$ for $q \in Q \setminus \{q_e, q_f\}$, and
- $\delta^t(q_e) = \emptyset$.

Note that for every well-behaved NFA A, the NTTA A^t is productive.

Lemma 4.1. *Let A be a well-behaved NFA. Then $P(\mathrm{lcp}(T(A^t))) = \{0, 1\}^* \setminus L(A)$.*

Proof. Let $A = (Q, \{0, 1\}, I, \{q_f\}, \delta)$. We first assume that $w \in L(A)$ and show that $w \notin P(\mathrm{lcp}(T(A^t)))$. For this, we have to prove that there exists a tree $t \in T(A^t)$ such that $w \notin P(t)$. Since $w \in L(A)$ we can write $w = uv$ with $v \neq \epsilon$ such that in A there exists a u-labeled path from $q_0 \in I$ to q_e. Let us write this path as

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} q_n = q_e,$$

where $u = a_1 a_2 \cdots a_n$ and $a_1, \ldots, a_n \in \{0, 1\}$. Note that $q_0, \ldots, q_{n-1} \in Q \setminus \{q_e, q_f\}$. For $1 \leq i \leq n$ let us choose any state $q_i' \in \delta(q_{i-1}, \bar{a_i})$ (where $\bar{0} = 1$ and $\bar{1} = 0$). Such a state q_i' must exist since A is well-behaved. Moreover, choose for every $1 \leq i \leq n$ a tree $t_i \in T(A^t, q_i')$. Finally let t be the unique tree with

$$P(t) = \{u\} \cup \bigcup_{i=1}^{n} \{a_1 \cdots a_{i-1} \bar{a_i} u' \mid u' \in P(t_i)\}.$$

From the construction of A^t it follows that $t \in T(A^t)$. Moreover, since $w = uv$ with $v \neq \varepsilon$ we get $w \notin P(t)$. This concludes the first part of the proof.

Now assume that $w \notin P(\mathrm{lcp}(T(A^t)))$. We have to show that $w \in L(A)$. Since $w \notin P(\mathrm{lcp}(T(A^t)))$, there exists $t \in T(A^t)$ such that $w \notin P(t)$. We can factorize $w = uv$ with $v \neq \varepsilon$, where $u = a_1 \cdots a_n$ is the longest prefix of w with $u \in P(t)$. Hence, u leads in the tree t to a leaf. Since $t \in T(A^t)$, there exists a run ρ of A^t on t such that $\rho(\epsilon) \in I$. Let $q_i = \rho(a_1 \cdots a_n)$ for $0 \leq i \leq n$. Since u leads to a leaf of t we must have $q_n = q_e$. Then

$$q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \xrightarrow{a_3} \cdots \xrightarrow{a_n} q_n = q_e$$

is a u-labeled path in A from $q_0 \in I$ to q_e. Since $v \neq \varepsilon$ we get $w = uv \in L(A)$. $\square$

In general, the NTTA A^t is not a DBTA. For this, we need an additional assumption on A: We say that an NFA $A = (Q, \{0,1\}, I, F, \delta)$ is weakly injective if

- $|\delta(q,a)| = 1$ for all $q \in Q$ and $a \in \{0,1\}$, and
- for all $p, q \in Q \setminus \{q_f\}$ with $p \neq q$ we have $\delta(p,0) \neq \delta(q,0)$ or $\delta(p,1) \neq \delta(q,1)$.

In a weakly injective NFA we can view the transition function δ as a mapping $\delta : Q \times \{0,1\} \to Q$ (nondeterminism only comes from the fact that there can be several initial states).

Lemma 4.2. *If A is a well-behaved weakly injective NFA then A^t is a DBTA.*

Proof. The lemma follows immediately from the definition of a weakly injective NFA and the fact that A^t has the unique final state q_e. □

5. Incompressibility of Largest Common Prefix Trees

5.1. Incompressibility by DAGs

In this section we present our first main result, which shows that there is a family of DBTA such that the size of the minimal DAG of the corresponding largest common prefix tree is exponential in the automaton size. Before we go into the details of the construction, let us first briefly sketch the idea. Below, we construct a family of regular languages $L_n \subseteq \{0,1\}^*$ $(n \geq 1)$ such that (i) the complement of L_n is a path language, (ii) L_n can be accepted by a well-behaved weakly injective NFA A_n with $\Theta(n)$ states and (iii) every DFA for the complement of L_n has at least 2^n states. Then the family of DBTA A_n^t $(n \geq 1)$ obtained from Lemma 4.2 has the desired properties.

For $n \geq 1$ we define the language L_n by:

$$L_n = \{0,1\}^{2n+3}\{0,1\}^* \cup \bigcup_{i=0}^{n-1} \left(\{0,1\}^i 0 \{0,1\}^n 0 \{0,1\}^+\right).$$

Let us first establish that the complement

$$V_n = \{0,1\}^* \setminus L_n \tag{1}$$

is a path language. Since L_n is a right ideal, the complement V_n is prefix closed. Since all words of length at least $2n+3$ belong to L_n, the language V_n is finite. Finally, $w0 \in \{0,1\}^{2n+3}\{0,1\}^*$ iff $|w0| \geq 2n+3$ iff $|w1| \geq 2n+3$ iff $w1 \in \{0,1\}^{2n+3}\{0,1\}^*$ and $w0 \in \{0,1\}^i 0 \{0,1\}^n 0 \{0,1\}^+$ iff $w1 \in \{0,1\}^i 0 \{0,1\}^n 0 \{0,1\}^+$. Hence, $w0 \in L_n$ if and only if $w1 \in L_n$, and the same property must hold for the complement V_n of L_n. Thus, V_n is a path language. The corresponding tree has depth $2n+2$. For $n = 2$ this tree is shown in Figure 2 (edges to a left/right child should be labelled with 0/1). Its minimal DAG is shown in Figure 3.

Lemma 5.1. *The minimal DFA A for V_n has at least 2^n states.*

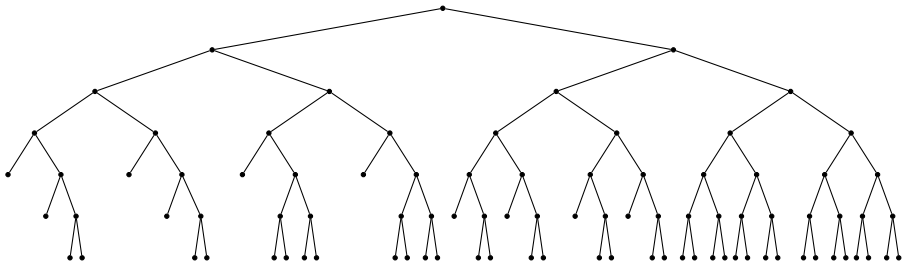

Figure 2: The tree with path language V_2; see (1).

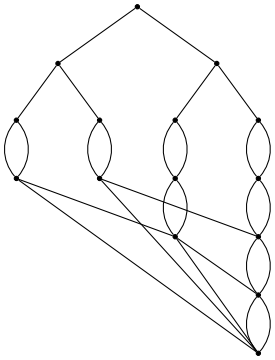

Figure 3: The minimal DAG for the tree from Figure 2.

Proof. Let $A = (Q, \{0,1\}, \delta, q_0, F)$. Consider the extension $\delta : Q \times \{0,1\}^* \to Q$ with $\delta(q, \epsilon) = q$ and $\delta(q, ua) = \delta(\delta(q, u), a)$ for $u \in \{0,1\}^*$, $a \in \{0,1\}$. We claim that $\delta(q_0, u) \neq \delta(q_0, v)$ for every $u, v \in \{0,1\}^n$ with $u \neq v$, which implies that A has at least 2^n states (and hence size at least 2^n). Assume by contradiction that $\delta(q_0, u) = \delta(q_0, v)$ for some $u, v \in \{0,1\}^n$ with $u \neq v$. We can write u and v as $u = x0y$ and $v = x1z$ (or vice versa) for some $x, y, z \in \{0,1\}^*$. Note that $0 \leq |x| \leq n-1$ and $|y| = |z|$. We define the words $u' = x0y1^{n-|y|}01 = u1^{n-|y|}01$ and $v' = x1z1^{n-|z|}01 = v1^{n-|y|}01$. Since $\delta(q_0, u) = \delta(q_0, v)$ we have $\delta(q_0, u') = \delta(q_0, v')$. It should be clear that $u' \in L_n = \{0,1\}^* \setminus V_n$. Hence, in order to get a contradiction, it suffices to show $v' \notin L_n$. First, note that $|v'| = 2n - |y| + 2 \leq 2n + 2$. This implies that if $v' \in L_n$, then it must belong to $\{0,1\}^i 0 \{0,1\}^n 0 \{0,1\}^+$ for some $0 \leq i \leq n-1$. But the word $v' = x1z1^{n-|z|}01$ contains no factor from $0\{0,1\}^n 0$ (note that $x1z$ has length n and hence cannot contain such a factor). □

Figure 4 shows a well-behaved weakly injective NFA A_n with $\Theta(n)$ states for the language L_n. Let $B_n := A_n^t$; it is a DBTA by Lemma 4.2. Moreover, B_n has $4n+5$ states and satisfies $P(\mathrm{lcp}(T(B_n))) = \{0,1\}^* \setminus L(A_n) = V_n$ by Lemma 4.1. From Lemma 2.1 and 5.1 it follows that the minimal DAG for $\mathrm{lcp}(T(A_n^t))$ has at least $2^n - 1$ nodes. We have shown:

Theorem 5.2. *For every n there is a DBTA B_n with $\Theta(n)$ states such that the minimal DAG for the tree $\mathrm{lcp}(T(B_n))$ has at least $2^n - 1$ nodes.*

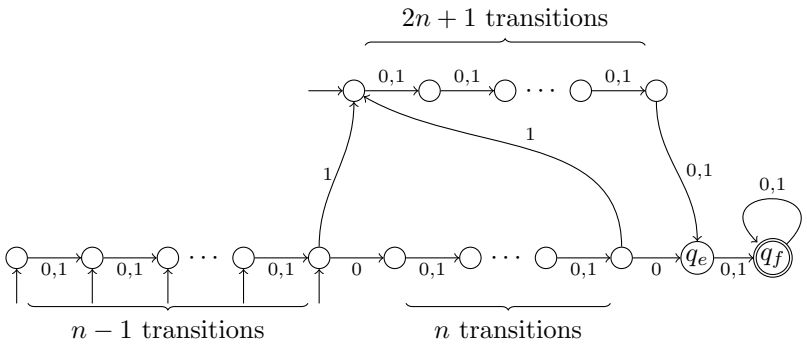

Figure 4: The well-behaved weakly injective NFA A_n recognizing the language L_n. Edges having no source node point to initial states.

Remark 5.3. The bound $2^n - 1$ in Theorem 5.2 is optimal up to constant factors in the exponent even for NTTAs: If B is an NTTA with n states then by Lemma 2.2, $\mathrm{lcp}(T(B))$ has depth at most $n-1$ and hence at most $2^n - 1$ nodes.

Remark 5.4. Recall that by Lemma 2.1, the minimal DFA for the path language $V_n = P(\mathrm{lcp}(T(B_n)))$ has at least 2^n states. We can also show that the every NFA for $P(\mathrm{lcp}(T(B_n)))$ has at least 2^n states. For this, we use the extended fooling set technique from [1]: Let L be a regular language and assume that there exists a finite set $S \subseteq \Sigma^* \times \Sigma^*$ such that:

- for all $(x, y) \in S$, $xy \in L$ holds, and
- for all $(x, y), (x', y') \in S$ with $(x, y) \neq (x', y') \in S$, $xy' \notin L$ or $x'y \notin L$ holds (in particular, we must have $x \neq x'$ and $y \neq y'$).

Then every NFA for L must have at least $|S|$ states.

We apply the extended fooling set technique to the finite language V_n. For this we define the set

$$S_n = \{(x1, \overline{x}1) \mid x \in \{0,1\}^n\}$$

where $\overline{x}$ results from x by replacing every 0 (resp., 1) by 1 (resp., 0). We obtain the following properties:

- for all $x \in \{0,1\}^n$, $x1\overline{x}1 \in V_n$;
- for all $x, y \in \{0,1\}^n$ with $x \neq y$, we have $x1\overline{y}1 \notin V_n$ or $y1\overline{x}1 \notin V_n$.

Hence, every NFA for V_n has at least $|S_n| = 2^n$ states.

5.2. Incompressibility by tree straight-line programs

So far we considered the compression of trees by DAGs. Let us now consider the more general formalism of tree straight-line programs (TSLPs) [6, 9].[1] In

[1] We define here monadic TSLPs in normal form [9], which makes no difference with respect to succinctness; see [9].

the following, it is useful to consider binary trees as expressions over the leaf symbol a and the binary symbol f as explained in Section 2.

A tree straight-line program is a so-called monadic linear context-free tree grammar that produces a single tree. Such a tree grammar has two types of variables (or nonterminals): variables of rank zero that produce binary trees and variables of rank one that produce binary trees in which a unique leaf is marked. Trees with marked leaves allow to define a concatenation operation similar to strings: a tree s with a marked leaf and a second tree t (which may or may not contain a marked leaf) can be concatenated to a tree $s[t]$ that is obtained by replacing in s the marked leaf by the tree t. In a tree straight-line program this kind of tree concatenation is used in the same way as string concatenation in ordinary (string) context-free grammars. In order to ensure that a tree straight-line program produces a single tree, one imposes the syntactic restriction that every variable is the left-hand side of a unique production. Moreover, one requires the grammar to be acyclic.

Formally, a TSLP is a 4-tuple $\mathcal{G} = (V_0, V_1, \rho, S)$ where V_0 (variables of rank zero) and V_1 (variables of rank one) are finite disjoint sets of variables, $S \in V_0$ is the start nonterminal, and ρ is a function that assigns to each variable A a formal expression (the right-hand side of A) such that one of the following conditions holds:

(a) $A \in V_0$ and $\rho(A) = a$,

(b) $A, B, C \in V_0$ and $\rho(A) = f(B, C)$,

(c) $A, C \in V_0$, $B \in V_1$ and $\rho(A) = B(C)$,

(d) $A, B, C \in V_1$ and $\rho(A) = B(C)$,

(e) $A \in V_1$, $B \in V_0$ and $\rho(A) = f(B, x)$,

(f) $A \in V_1$, $B \in V_0$ and $\rho(A) = f(x, B)$.

We require that the binary relation $E(\mathcal{G}) = \{(B, A) \mid B \text{ occurs in } \rho(A)\}$ is acyclic. We can therefore define a partial order $\leq_{\mathcal{G}}$ as the reflexive transitive closure of $E(\mathcal{G})$. The idea is that with the above rules, every variable $A \in V_0$ evaluates to a unique binary tree $[\![A]\!]_{\mathcal{G}}$, whereas every variable $A \in V_1$ evaluates to a unique binary tree $[\![A]\!]_{\mathcal{G}}$ with a marked leaf. This marked leaf is denoted by the special symbol x. For instance, $f(f(a, x), f(a, a))$ would be such a tree. We let $T_{2,x}$ denote the set of all such trees. For $s \in T_{2,x}$ and $t \in T_{2,x} \cup T_2$ we let $s[t]$ denote the result of replacing in s the unique occurrence of x by t. For instance, for $s = f(f(a, x), f(a, a))$ and $t = f(a, x)$ we have $s[t] = f(f(a, f(a, x)), f(a, a))$. Here are the formal inductive rules for the evaluation of variables. In all cases $t_B := [\![B]\!]_{\mathcal{G}}$ and $t_C := [\![C]\!]_{\mathcal{G}}$ are already defined by induction.

- if $A \in V_0$ and $\rho(A) = a$, then $[\![A]\!]_{\mathcal{G}} = a$,
- if $A, B, C \in V_0$ and $\rho(A) = f(B, C)$, then $[\![A]\!]_{\mathcal{G}} = f(t_B, t_C)$,
- if $A, C \in V_0$, $B \in V_1$ and $\rho(A) = B(C)$, then $[\![A]\!]_{\mathcal{G}} = t_B[t_C]$,
- if $A, B, C \in V_1$ and $\rho(A) = B(C)$, then $[\![A]\!]_{\mathcal{G}} = t_B[t_C]$,
- if $A \in V_1$, $B \in V_0$ and $\rho(A) = f(B, x)$, then $[\![A]\!]_{\mathcal{G}} = f(t_B, x)$,

- if $A \in V_1$, $B \in V_0$ and $\rho(A) = f(x, B)$, then $[\![A]\!]_{\mathcal{G}} = f(x, t_B)$.

Finally, we define $[\![\mathcal{G}]\!] = [\![S]\!]_{\mathcal{G}} \in T_2$.

Note that a DAG corresponds to a TSLP where only variables of the above types (a) and (b) are present. In contrast to DAGs, TSLPs can also compress deep narrow trees, such as caterpillar trees, for example.

Example 5.5. *Let* $\mathcal{G} = (\{A_0, A_2, A_3, A_5\}, \{A_1, A_4\}, \rho, A_0)$ *be the TSLP with*

- $\rho(A_0) = A_1(A_2)$,
- $\rho(A_1) = f(x, A_3)$,
- $\rho(A_2) = A_4(A_3)$,
- $\rho(A_3) = A_4(A_5)$,
- $\rho(A_4) = f(x, A_5)$,
- $\rho(A_5) = a$.

We have $[\![\mathcal{G}]\!] = f(f(f(a, a), a), f(a, a))$.

Lemma 5.6. *Let* $\mathcal{G} = (V_0, V_1, \rho, S)$ *be a TSLP with* $t = [\![\mathcal{G}]\!]$ *and let* d *be the depth of* t. *Then the minimal DAG for* t *has at most* $|V_0| \cdot d$ *nodes.*

Proof. We count the number of pairwise non-isomorphic subtrees of t. Consider a specific subtree $s \in T_2$ of t. By walking down from the start variable $S \in V_0$ we can determine the smallest variable A (with respect to $\leq_{\mathcal{G}}$) such that s is a subtree of $[\![A]\!]_{\mathcal{G}}$. Let us consider the cases (a)–(f) for the right-hand side $\rho(A)$.

If (a) or (b) holds, then we must have $s = [\![A]\!]_{\mathcal{G}}$. The cases (e) and (f) cannot occur (in both cases s would be a subtree of $[\![B]\!]_{\mathcal{G}}$). Similarly, (d) cannot occur since s would be a subtree of either $[\![B]\!]_{\mathcal{G}}$ or $[\![C]\!]_{\mathcal{G}}$. Finally in case (c), since s is neither a subtree of $[\![B]\!]_{\mathcal{G}}$ nor $[\![C]\!]_{\mathcal{G}}$, the subtree s must be rooted at one of the nodes on the path leading from the root of $[\![A]\!]_{\mathcal{G}}$ to the position of the symbol x in $[\![B]\!]_{\mathcal{G}}$ (excluding the position of x). There are at most d such nodes. It follows that $[\![\mathcal{G}]\!]$ contains at most $|V_0| \cdot d$ different subtrees. □

Theorem 5.7. *For every* n *there is an DBTA* B_n *with* $\Theta(n)$ *states such that the smallest TSLP for the tree* $\mathrm{lcp}(T(B_n))$ *has* $\Omega(2^n/n)$ *variables.*

Proof. We take the tree automata family from Theorem 5.2. Assume that $\mathcal{G} = (V_0, V_1, \rho, S)$ is a TSLP for the tree $\mathrm{lcp}(T(B_n))$ from Theorem 5.2. The minimal DAG for $\mathrm{lcp}(T(B_n))$ has at least $2^n - 1$ nodes. Recall that $P(\mathrm{lcp}(T(B_n))) = V_n = \{0, 1\}^* \setminus L_n$. Since L_n contains all word of length at least $2n+3$, the path language V_n contains only words of length at most $2n + 2$. Thus, the depth of the tree $\mathrm{lcp}(T(B_n))$ is at most $2n + 2$. With Lemma 5.6 it follows that the smallest TSLP for $\mathrm{lcp}(T(B_n))$ has at least $(2^n - 1)/(2n + 2)$ variables. □

The upper bound $\Omega(2^n/n)$ for NTTAs in Theorem 5.7 cannot be improved much: As remarked before, if an NTTA B has n states then the tree $\mathrm{lcp}(T(B))$ has at most 2^n nodes. By [6], $\mathrm{lcp}(T(B))$ has a TSLP with $\mathcal{O}(2^n/n)$ variables.

From a TSLP for a tree t with m variables one can easily construct a context-free grammar for $P(t)$ of size $O(m)$ (where the size of a context-free grammar is the total length of all right-hand sides of productions). We conjecture that the smallest context-free grammar for the language V_n has size $\Omega(2^n/n)$. This would yield an alternative proof for Theorem 5.7. One might try to apply the technique from [5] to prove the above conjecture.

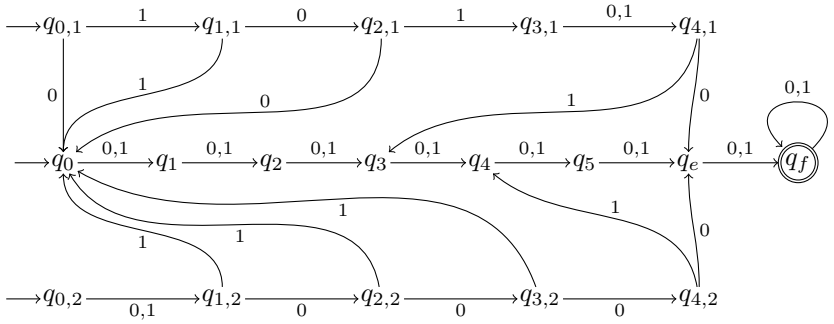

Figure 5: The construction from the proof of Theorem 6.3 for the 3-SAT formula $C = C_1 \wedge C_2$ with $C_1 = (\neg x_1 \vee x_2 \vee \neg x_3)$ and $C_2 = (x_2 \vee x_3 \vee x_4)$ (so $n = 4$ and $m = 2$).

6. Checking Equality of Largest Common Prefixes

We now deal with the problem of checking whether two tree languages yield the same lcp (or whether one lcp is contained in the other lcp). For DTTAs this is possible in polynomial time, whereas the problem becomes **coNP**-complete for DBTAs.

Theorem 6.1. *The problem of checking* $P(\mathrm{lcp}(T(B_1))) \subseteq P(\mathrm{lcp}(T(B_2)))$ *for two given DTTAs* B_1 *and* B_2 *can be solved in polynomial time.*

Proof. We compute the DFAs B_1^s and B_2^s from Section 3. Since B_1 and B_2 are DTTAs, these DFAs can be computed in polynomial time. By Lemma 3.2 we have $P(\mathrm{lcp}(T(B_1))) \subseteq P(\mathrm{lcp}(T(B_2)))$ if and only if $L(B_1^s) \subseteq L(B_2^s)$. The theorem follows because inclusion of DFAs can be checked in polynomial time. □

Theorem 6.2. *The problem of checking* $P(\mathrm{lcp}(T(B_1))) \subseteq P(\mathrm{lcp}(T(B_2)))$ *for two given NTTAs* B_1 *and* B_2 *belongs to* **coNP**.

Proof. We show that there exists a nondeterministic polynomial time machine that checks whether there exists $u \in P(\mathrm{lcp}(T(B_1)))$ with $u \notin P(\mathrm{lcp}(T(B_2)))$. W.l.o.g. we can assume that B_1 and B_2 are productive. Let m be the number of states of B_1. By Lemma 2.2 we know that $P(\mathrm{lcp}(T(B_1)))$ only contains words of length at most $m-1$. Hence, we can nondeterministically guess a word u of length at most $m-1$ and then verify whether $u \in P(\mathrm{lcp}(T(B_1)))$ and $u \notin P(\mathrm{lcp}(T(B_2)))$. For this we use the DFAs B_1^s and B_2^s from Lemma 3.1 and check whether $u \in L(B_1^s)$ and $u \notin L(B_2^s)$. For this, we do not have to construct the DFAs B_1^s and B_2^s explicitly (they have exponential size); it suffices to run B_1^s and B_2^s on the fly on the word u (recall that u has polynomial length). □

Theorem 6.3. *The problem of checking* $\mathrm{lcp}(T(B_1)) = \mathrm{lcp}(T(B_2))$ *for two given NTTAs* B_1 *and* B_2 *is* **coNP**-*complete. The* **coNP** *lower bound already holds for the case that* B_1 *and* B_2 *are DBTAs.*

Proof. Since **coNP** is closed under intersection, we obtain the upper bound from Theorem 6.2. Let us now show **coNP**-hardness for DBTAs by a reduction from

the complement of 3-SAT. Consider a 3-SAT formula $C = \bigwedge_{i=1}^{m} C_i$ where every C_i is a disjunction of three literals (possibly negated variables). Let $x_1, \ldots, x_n$ be the variables that occur in C. W.l.o.g. we can assume that $n \geq m$ (we can add dummy variables if necessary) and that there is no clause C_i and variable x_j such that x_j and $\neg x_j$ both belong to C_i. Given a bit string $w = a_1 a_2 \cdots a_n$ with $a_i \in \{0,1\}$ we write $w \models C_i$ (resp., $w \models C$) if C_i (resp., C) becomes true when every variable x_i gets the truth value a_i.

We first construct an (incomplete) acyclic DFA A_i for the language $\{w0 \mid w \in \{0,1\}^n, w \not\models C_i\}$. The states of A_i are $q_{0,i}, q_{1,i}, \ldots, q_{n,i}, q_{n+1,i}$, $q_{0,i}$ is the initial state, $q_{n+1,i}$ is the final state, and the transitions are defined as follows, where $1 \leq j \leq n$:

- $q_{j-1,i} \xrightarrow{0} q_{j,i}$ if x_j belongs to C_i,
- $q_{j-1,i} \xrightarrow{1} q_{j,i}$ if $\neg x_j$ belongs to C_i,
- $q_{j-1,i} \xrightarrow{0,1} q_{j,i}$ if neither x_j nor $\neg x_j$ belongs to C_i,
- $q_{n,i} \xrightarrow{0} q_{n+1,i}$.

By taking the disjoint union of the DFAs A_i, we obtain an NFA A with

$$\begin{aligned} L(A) &= \bigcup_{i=1}^{n} L(A_i) \\ &= \bigcup_{i=1}^{n} \{w0 \mid w \in \{0,1\}^n, w \not\models C_i\} \\ &= \{w0 \mid w \in \{0,1\}^n, w \not\models C\}. \end{aligned}$$

Hence, we have $L(A) = \{0,1\}^n 0$ if and only if C is not satisfiable. Note that the initial states of A are the states $q_{0,1}, \ldots, q_{0,m}$.

We finally construct a well-behaved NFA A_1 from A as follows (an example is shown in Figure 5):

- Merge the final states $q_{n+1,i}$ $(1 \leq i \leq m)$ into a single non-final state q_e.
- Add states $q_0, q_1, \ldots, q_{n+1}, q_f$, where q_0 is an initial state (hence, the initial states of A_1 are $q_0, q_{0,1}, \ldots, q_{0,m}$) and q_f is the unique final state of A_1.
- Add the transitions $q_j \xrightarrow{0,1} q_{j+1}$ for $0 \leq j \leq n$, $q_{n+1} \xrightarrow{0,1} q_e \xrightarrow{0,1} q_f \xrightarrow{0,1} q_f$.
- If some state $q_{j-1,i}$ $(1 \leq i \leq m,\ 1 \leq j \leq n)$ has no outgoing a-transition for $a \in \{0,1\}$ (this happens if $a = 0$ and $\neg x_j$ belongs to C_i or $a = 1$ and x_j belongs to C_i) then add the transition $q_{j-1,i} \xrightarrow{a} q_0$ to A_1.
- For every $1 \leq i \leq m$ we add a 1-transition from $q_{n,i}$ to one of the states $q_0, \ldots, q_{n+1}$ in such a way that no two such 1-transitions enter the same state. Since $m \leq n+2$, this is possible.

The automaton A_1 satisfies $L(A_1) = L(A)\{0,1\}\{0,1\}^* \cup \{0,1\}^{n+3}\{0,1\}^*$ and is is well-behaved and weakly injective. Hence, by Lemma 4.2, A_1^t is a DBTA.

It is straightforward to construct a well-behaved weakly injective NFA A_2 such that $L(A_2) = \{0,1\}^n 0\{0,1\}\{0,1\}^* \cup \{0,1\}^{n+3}\{0,1\}^*$ (one can make the

above construction with an unsatisfiable 3-SAT formula). We get the following equivalences:

$$\begin{aligned} C \text{ is unsatisfiable} &\Leftrightarrow L(A) = \{0,1\}^n 0 \\ &\Leftrightarrow L(A_1) = \{0,1\}^n 0\{0,1\}\{0,1\}^* \cup \{0,1\}^{n+3}\{0,1\}^* \\ &\Leftrightarrow L(A_1) = L(A_2) \\ &\Leftrightarrow \{0,1\}^* \setminus L(A_1) = \{0,1\}^* \setminus L(A_2) \\ &\Leftrightarrow P(\mathrm{lcp}(T(A_1^t))) = P(\mathrm{lcp}(T(A_2^t))) \\ &\Leftrightarrow \mathrm{lcp}(T(A_1^t)) = \mathrm{lcp}(T(A_2^t)). \end{aligned}$$

This concludes the proof of the theorem. □

7. Conclusion

We presented a family of tree automata of size n such that the size of the largest common prefix (lcp) tree of all accepted trees is exponential in n and basically incompressible with respect to DAGs (directed acyclic graphs) and tree straight-line programs. Moreover, we proved that the problem whether the largest common prefix trees of two regular tree languages (that are either specified by non-deterministic tree automata or deterministic bottom-up tree automata) is **coNP**-complete.

Our results are mainly negative; only for the very restricted class of deterministic top-down tree automata we obtain positive results with respect to compressibility and the complexity of the equivalence problem for lcp trees (which goes down to polynomial time). This leads to the question whether these positive results can be extended to some interesting class of tree automata that properly includes the class of deterministic top-down tree automata.

Another question is whether a good enough approximation (in a sense that has to be made precise) of the largest common prefix tree of a regular tree language allows some non-trivial compression with DAGs or tree straight-line programs.

Finally, from a theoretical side, one might also study larger classes of tree languages, e.g. context-free tree languages. It is not clear whether the equivalence problem for the lcp trees of two given context-free tree languages is decidable.

Acknowledgements. The first author was partially supported by the DFG research project LO748/10-2.

[1] J. Birget. Intersection and union of regular languages and state complexity. *Information Processing Letters*, 43(4):185–190, 1992.

[2] C. Choffrut. Minimizing subsequential transducers: a survey. *Theoretical Computer Science*, 292(1):131–143, 2003.

[3] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. Tree automata techniques and applications. Available at: `http://tata.gforge.inria.fr/`, 2007.

[4] J. Engelfriet, S. Maneth, and H. Seidl. Deciding equivalence of top-down XML transformations in polynomial time. *Journal of Computer and System Sciences*, 75(5):271–286, 2009.

[5] Y. Filmus. Lower bounds for context-free grammars. *Information Processing Letters*, 111(18): 895–898, 2011.

[6] M. Ganardi, D. Hucke, A. Jez, M. Lohrey, and E. Noeth. Constructing small tree grammars and small circuits for formulas. *Journal of Computer and System Sciences*, 86:136–158, 2017.

[7] A. Lemay, S. Maneth, and J. Niehren. A learning algorithm for top-down XML transformations. In *Proceedings of PODS 2010*, pages 285–296, ACM 2010.

[8] M. Lohrey and S. Maneth. Largest common prefix of a regular tree language. In *Proceedings of FCT 2019*, volume 11651 of *LNCS*, pages 95-108, Springer 2019.

[9] M. Lohrey, S. Maneth, and M. Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *Journal of Computer and System Sciences*, 78(5):1651–1669, 2012.

[10] M. Luttenberger, R. Palenta, and H. Seidl. Computing the longest common prefix of a context-free language in polynomial time. In *Proceedings of STACS 2018*, volume 96 of *LIPIcs*, pages 48:1–48:13. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2018

[11] M. Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234(1-2):177–201, 2000.

[12] J. Oncina, P. García, and E. Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):448–458, 1993.