

# Largest Common Prefix of a Regular Tree Language

Markus Lohrey<sup>1</sup> and Sebastian Maneth<sup>2</sup>

<sup>1</sup> University of Siegen, Germany [lohrey@eti.uni-siegen.de](mailto:lohrey@eti.uni-siegen.de)

<sup>2</sup> University of Bremen, Germany [maneth@uni-bremen.de](mailto:maneth@uni-bremen.de)

**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  $L$  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 [1, 8] and tree transducers [3]. 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 [9, 5]. 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 [4, 6]. 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 [7].

Whenever above we mention “tree automaton”, we always mean “nondeterministic (top-down or bottom-up) tree automaton”. Let us now consider the

case of *deterministic* tree automata. It is known that in the deterministic case, *top-down* automata are strictly less expressive than 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. In contrast, for deterministic bottom-up tree automata (which are equally expressive as nondeterministic tree automata), such a procedure is not possible. Surprisingly, for such automata similar results can be proven as for nondeterministic automata, e.g., a family of automata of size  $\Theta(n^2)$  can be defined, such that the size of their lcp is exponential in  $n$ . Technically, one ingredient of both families of automata (nondeterministic and deterministic ones) is the well-known fact that an automaton needs exponentially many states in order to recognize strings where the  $n$ -th last symbol carries a specific label.

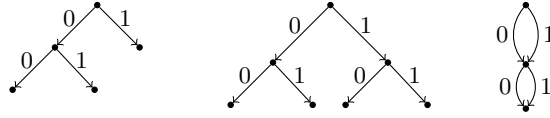
We then address a second important problem for largest common prefixes given by tree automata, namely to determine whether or not the largest common prefixes of two given tree automata coincide. Note that when constructing an earliest canonical (“minimal”) transducer, we need to determine whether two 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.

## 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  $\Gamma$  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  $\delta : Q \times \Gamma \rightarrow Q$  is the transition mapping. The language  $L(A)$  accepted by  $A$  is defined in the usual way. For an NFA (nondeterministic finite automaton) we have a set  $I \subseteq Q$  of initial states and the transition function  $\delta$  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



**Fig. 1.** Trees  $t_0 = f(f(a, a), a)$  (left),  $t_1 = f(f(a, a), f(a, a))$  (middle) and the minimal DAG of  $t_1$  (right).

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 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 root of a tree corresponds to the empty word  $\epsilon$ ,  $u0$  denotes the left child of  $u$ , and  $u1$  denotes the right child of  $u$ . The leaves of a tree  $t$  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  $\delta : Q \rightarrow 2^{Q^2}$  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  $\rho : P(t) \rightarrow Q$  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  $\rho(v) = p$ ,  $\rho(v0) = p_0$  and  $\rho(v1) = p_1$  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  $\rho$  of  $B$  such that  $\rho(\epsilon) = q$ . 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  $\delta : Q \rightarrow Q^2$  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  $q_1 q_2 \in Q^2$  there is at most one  $q \in Q$  such that  $q_1 q_2 \in \delta(q)$ . In other words, the sets  $\delta(q)$  ( $q \in Q$ ) are pairwise disjoint. This allows defining a partially defined inverse  $\delta^{-1}$  of  $\delta$  by  $\delta^{-1}(q_1 q_2) = q$  if  $q_1 q_2 \in \delta(q)$ . For every tree  $t$  there is at most one run of  $B$  on  $t$  and this run  $\rho$  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  $v, v_0, v_1 \in P(t)$  such that  $\rho(v_0)$  and  $\rho(v_1)$  have been already defined, one sets  $\rho(v) = \delta^{-1}(\rho(v_0)\rho(v_1))$ .

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 [2] for examples.

The *minimal DAG* for a tree  $t \in T_2$  is obtained by keeping for every subtree  $s$  of  $t$  exactly one isomorphic copy to which all tree edges that point to occurrences of  $s$  are redirected. The size of the minimal DAG of  $t$  (measured in number of nodes) is exactly the number of pairwise non-isomorphic subtrees of  $t$ .

**Lemma 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.*

The proof of the lemma is straightforward. Consider for instance the tree  $t_1 = f(f(a, a), f(a, a))$  from Figure 1. Its minimal DAG is shown in Figure 1 on the right. It yields a DFA for  $P(t_1)$  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).

**Largest common prefix tree.** Consider a non-empty tree language  $L \subseteq T_2$ . The *largest common prefix*  $\text{lcp}(L)$  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  $\text{lcp}(L) = f(a, a)$ .

**Lemma 2.** *Assume that  $B$  is an NTTA with  $n$  states and such that  $T(B) \neq \emptyset$ . Then every word  $w \in P(\text{lcp}(T(B))) = \bigcap_{t \in T(B)} P(t)$  has length at most  $n - 1$ , i.e., the depth of  $\text{lcp}(T(B))$  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. [2, Corollary 1.2.3]. Hence,  $|w| \leq n - 1$  for every word  $w \in P(t)$ . This implies the statement of the lemma.  $\square$

It is straightforward to extend all the notions from this section to labelled binary trees. A  $\Sigma$ -labelled binary tree can be defined as a pair  $(P, \lambda)$  where  $P \subseteq \{0, 1\}^*$  is a path language and  $\lambda : P \rightarrow \Sigma$  is the labelling function. Given a set  $L$  of  $\Sigma$ -labelled binary trees, one can define its lcp as the unique tree  $(P, \lambda)$  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  $\Sigma$ -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. We extend  $\delta : Q \rightarrow 2^{Q^2}$  to  $\hat{\delta} : 2^Q \rightarrow 2^{Q^2}$  by setting  $\hat{\delta}(Q') = \bigcup_{p \in Q'} \delta(p) \subseteq Q^2$  for  $Q' \subseteq Q$ . For a state pair  $p_0 p_1 \in Q^2$  and  $i \in \{0, 1\}$  we define the projection  $\pi_i(p_0 p_1) = p_i$ . 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  $q_f \notin Q$  and define a 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  $\delta^s$  is well-defined.

**Lemma 3.** *Let  $B$  be a productive NTTA. Then*

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

*Proof.* Consider a word  $w = a_1 a_2 \cdots a_n$  with  $a_1, \dots, a_n \in \{0, 1\}$ . 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  $v = a_1 a_2 \cdots a_k$  for  $k < n$ . Fix a run  $\rho$  of  $B$  on  $t$  such that  $\rho(\varepsilon) \in I$ . 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  $t'_1, \dots, t'_k, a$  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  $t = f(t'_1, f(t'_2, f(t'_3, \dots f(t'_k, a) \cdots)))$ . For the general case, we define trees  $t_0, t_1, \dots, t_k$  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.  $\square$

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 as the proof for Lemma 3.

**Lemma 4.** *Let  $B$  be a DTTA with  $T(B) \neq \emptyset$ . Then*

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

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

*Proof.* Let  $A = (Q, \{0,1\}, I, F, \delta)$ . We first assume that  $w \in L(A)$  and show that  $w \notin P(\text{lcp}(T(A^t)))$ . 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

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

where  $u = a_1 a_2 \dots a_n$  and  $a_1, \dots, a_n \in \{0,1\}$ . Note that  $q_0, \dots, q_{n-1} \in Q \setminus \{q_e, q_f\}$ . 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

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

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

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

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

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

## 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 tree automata such that the size of the minimal DAG of the corresponding largest common prefix tree is exponential in the automata size.

For  $n \geq 1$  we consider the following language  $L_n$ :

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

Let us first establish that the complement  $V_n = \{0, 1\}^* \setminus L_n$  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.

**Lemma 6.** *The minimal DFA  $A$  for  $V_n$  has at least  $2^n$  states.*

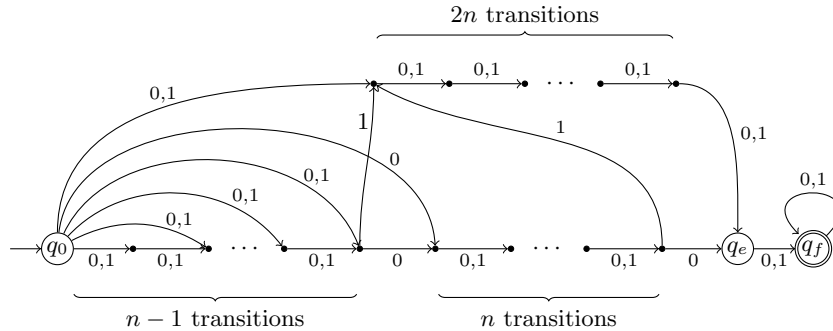
*Proof.* Let  $A = (Q, \{0, 1\}, \delta, q_0, F)$ . Consider the extension  $\delta : Q \times \{0, 1\}^* \rightarrow Q$  with  $\delta(q, \epsilon) = q$  and  $\delta(q, ua) = \delta(\delta(q, u), a)$  for  $u \in \{0, 1\}^*$ ,  $a \in \{0, 1\}$ . We claim that  $\delta(q_0, u) \neq \delta(q_0, v)$  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).  $\square$

Figure 2 shows a well-behaved NFA  $A_n$  with  $\Theta(n)$  states for the language  $L_n$ . The NTTA  $B_n := A_n^t$  has  $\Theta(n)$  states as well and satisfies  $P(\text{lcp}(T(B_n))) = \{0, 1\}^* \setminus L(A_n) = V_n$  by Lemma 5. From Lemma 1 and 6 it follows that the minimal DAG for  $\text{lcp}(T(A_n^t))$  has at least  $2^n - 1$  nodes. We have shown:

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

The bound  $2^n - 1$  in Theorem 1 is optimal up to constant factors in the exponent: If  $B$  is an NTTA with  $n$  states then by Lemma 2,  $\text{lcp}(T(B))$  has depth at most  $n - 1$  and hence at most  $2^n - 1$  nodes. A variation of the above construction yields a slightly weaker lower bound for DBTAs:





**Fig. 2.** The well-behaved NFA  $A_n$  recognizing the language  $L_n$

**Theorem 2.** For every  $n$  there is a DBTA  $B'_n$  with  $\Theta(n^2)$  states such that the minimal DAG for the tree  $\text{lcp}(T(B'_n))$  has at least  $2^n - 1$  nodes.

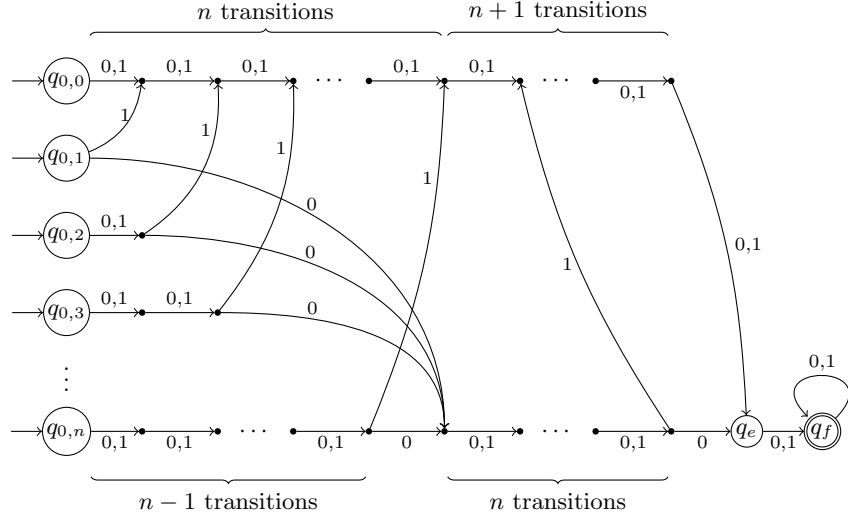
*Proof.* Consider the NFA  $A'_n = (Q, \{0, 1\}, I, \{q_f\}, \delta)$  from Figure 3. It is a well-behaved NFA that recognizes the language  $L_n$  as well. The transition function  $\delta$  can be viewed as a mapping  $\delta : Q \times \{0, 1\} \rightarrow Q$  (nondeterminism only comes from the fact that there are several initial states). Moreover,  $\delta$  has the property that for all states  $p, q \in Q \setminus \{q_f\}$ , if  $p \neq q$  then  $\delta(p, 0)\delta(p, 1) \neq \delta(q, 0)\delta(q, 1)$ . Together with the fact that  $q_e$  is the unique final state of the NTTA  $A'^t$ , this implies that  $A'^t$  is a DBTA.  $\square$

## 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) [4, 6].<sup>3</sup> As explained in Section 2 we consider binary trees as expressions over the leaf symbol  $a$  and the binary symbol  $f$ . A TSLP is a 4-tuple  $\mathcal{G} = (V_0, V_1, \rho, S)$  where  $V_0$  and  $V_1$  are finite disjoint sets of variables,  $S \in V_0$  is the start nonterminal, and  $\rho$  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  $\rho(A) = f(x, B)$ .

<sup>3</sup> We define here monadic TSLPs in normal form [6] which makes no difference with respect to succinctness; see [6].



**Fig. 3.** The well-behaved NFA  $A'_n$  recognizing  $L_n$  from the proof of Theorem 2.

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  $\llbracket A \rrbracket_{\mathcal{G}}$ , whereas every variable  $A \in V_1$  evaluates to a unique binary tree  $\llbracket A \rrbracket_{\mathcal{G}}$  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  $t_B := \llbracket B \rrbracket_{\mathcal{G}}$  and  $t_C := \llbracket C \rrbracket_{\mathcal{G}}$  are already defined by induction.

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

Finally, we define  $\llbracket \mathcal{G} \rrbracket = \llbracket S \rrbracket_{\mathcal{G}} \in T_2$ . Readers that are familiar with the notion of context-free tree grammars will notice that a TSLP is a context-free tree grammar that produces a unique tree. 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.

**Lemma 7.** *Let  $\mathcal{G} = (V_0, V_1, \rho, S)$  be a TSLP with  $t = \llbracket \mathcal{G} \rrbracket$  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  $\llbracket A \rrbracket_{\mathcal{G}}$ . Let us consider the cases (a)–(f) for the right-hand side  $\rho(A)$ .

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

**Theorem 3.** *For every  $n$  there is an NTTA  $B_n$  with  $\Theta(n)$  states such that the smallest TSLP for the tree  $\text{lcp}(T(B_n))$  has  $\Omega(2^n/n)$  variables. Moreover, for every  $n$  there is a DBTA  $B'_n$  with  $\Theta(n^2)$  states such that the smallest TSLP for the tree  $\text{lcp}(T(B'_n))$  has  $\Omega(2^n/n)$  variables.*

*Proof.* We take the tree automata families from Theorem 1 and 2, respectively. Assume that  $\mathcal{G} = (V_0, V_1, \rho, S)$  is a TSLP for the tree  $\text{lcp}(T(B_n))$  from Theorem 1. The minimal DAG for  $\text{lcp}(T(B_n))$  has at least  $2^n - 1$  nodes. Recall that  $P(\text{lcp}(T(B_n))) = V_n = \{0, 1\}^* \setminus L_n$ . 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  $\text{lcp}(T(B_n))$  is at most  $2n + 2$ . With Lemma 7 it follows that the smallest TSLP for  $\text{lcp}(T(B_n))$  has at least  $(2^n - 1)/(2n + 2)$  variables. For DBTAs one can argue analogously using Theorem 2.  $\square$

The upper bound  $\Omega(2^n/n)$  for NTTAs in Theorem 3 cannot be improved much: As remarked before, if an NTTA  $B$  has  $n$  states then the tree  $\text{lcp}(T(B))$  has at most  $2^n$  nodes. By [4],  $\text{lcp}(T(B))$  has a TSLP with  $\mathcal{O}(2^n/n)$  variables.

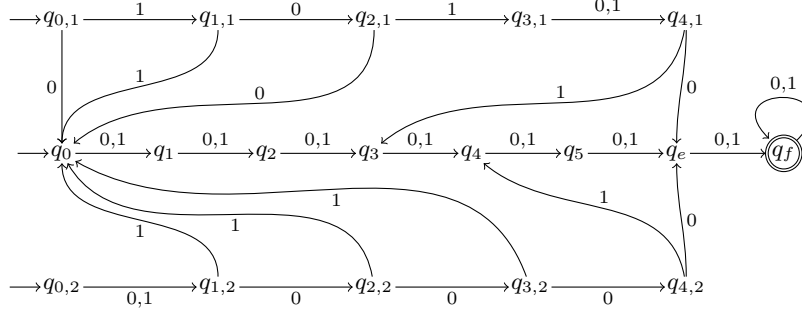
## 6 Checking Equality of Largest Common Prefixes

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 4.** *The problem of checking  $P(\text{lcp}(T(B_1))) \subseteq P(\text{lcp}(T(B_2)))$  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 4 we have  $P(\text{lcp}(T(B_1))) \subseteq P(\text{lcp}(T(B_2)))$  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.  $\square$

**Theorem 5.** *The problem of checking  $P(\text{lcp}(T(B_1))) \subseteq P(\text{lcp}(T(B_2)))$  for two given NTTAs  $B_1$  and  $B_2$  belongs to **coNP**.*



**Fig. 4.** The construction from the proof of Theorem 6 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  $m = 2$ ).

*Proof.* We show that there exists a nondeterministic polynomial time machine that checks whether there exists  $u \in P(\text{lcp}(T(B_1)))$  with  $u \notin P(\text{lcp}(T(B_2)))$ . 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 we know that  $P(\text{lcp}(T(B_1)))$  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  $u \in P(\text{lcp}(T(B_1)))$  and  $u \notin P(\text{lcp}(T(B_2)))$ . For this we use the DFAs  $B_1^s$  and  $B_2^s$  from Lemma 3 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).  $\square$

**Theorem 6.** *The problem of checking  $\text{lcp}(T(B_1)) = \text{lcp}(T(B_2))$  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 5. 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  $x_1, \dots, x_n$  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  $w = a_1 a_2 \dots a_n$  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  $q_{0,i}, q_{1,i}, \dots, q_{n,i}, q_{n+1,i}$ ,  $q_{0,i}$  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  $q_{0,1}, \dots, q_{0,m}$ .

We finally construct a well-behaved NFA  $A_1$  from  $A$  as follows (an example is shown in Figure 4):

- Merge the final states  $q_{n+1,i}$  ( $1 \leq i \leq m$ ) into a single non-final state  $q_e$ .
- Add states  $q_0, q_1, \dots, q_{n+1}, q_f$ , where  $q_0$  is an initial state (hence, the initial states of  $A_1$  are  $q_0, q_{0,1}, \dots, q_{0,m}$ ) 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  $q_0, \dots, q_{n+1}$  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 well-behaved. The transition function  $\delta$  of  $A_1$  can be viewed as a mapping  $\delta : Q \times \{0,1\} \rightarrow Q$  (nondeterminism only comes from the fact that there are several initial states). Moreover,  $\delta$  has the property that for all states  $p, q \in Q \setminus \{q_f\}$ , if  $p \neq q$  then  $\delta(p,0)\delta(p,1) \neq \delta(q,0)\delta(q,1)$ . Together with the fact that  $q_e$  is the unique final state of the NFTA  $A_1^t$ , this implies that  $A_1^t$  is a DBTA.

It is straightforward to construct a well-behaved NFA  $A_2$  such that  $L(A_2) = \{0,1\}^n 0 \{0,1\} \{0,1\}^* \cup \{0,1\}^{n+3} \{0,1\}^*$  and  $A_2^t$  is a DBTA (one can make the above construction with an unsatisfiable 3-SAT formula). We get the following equivalences:

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

This concludes the proof of the theorem. □

## References

1. C. Choffrut. Minimizing subsequential transducers: a survey. *Theoretical Computer Science*, 292(1):131–143, 2003.
2. 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.
3. 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.
4. M. Ganardi, D. Hucce, 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.
5. 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.
6. 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.
7. 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.
8. M. Mohri. Minimization algorithms for sequential transducers. *Theoretical Computer Science*, 234(1-2):177–201, 2000.
9. 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.