

# Deciding Twig-definability of Node Selecting Tree Automata

**Timos Antonopoulos · Dag Hovland ·  
Wim Martens · Frank Neven**

Published online: 17 April 2015  
© Springer Science+Business Media New York 2015

**Abstract** Node selecting tree automata (NSTAs) constitute a general formalism defining unary queries over trees. Basically, a node is selected by an NSTA when it is visited in a selecting state during an accepting run. We consider twig patterns as an abstraction of XPath. Since the queries definable by NSTAs form a strict superset of twig-definable queries, we study the complexity of the problem to decide whether the query by a given NSTA is twig-definable. In particular, we obtain that the latter problem is EXPTIME-complete. In addition, we show that it is also EXPTIME-complete

---

An extended abstract of this paper appeared in [2]

We acknowledge the financial support of the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599.

Supported by grant number MA 4938/2–1 from the Deutsche Forschungsgemeinschaft (Emmy Noether Nachwuchsgruppe).

---

T. Antonopoulos · F. Neven  
Hasselt University and Transnational University of Limburg, Hasselt, Belgium

T. Antonopoulos  
e-mail: timos.antonopoulos@gmail.com

F. Neven  
e-mail: frank.neven@uhasselt.be

D. Hovland (✉)  
University of Oslo, Oslo, Norway  
e-mail: hovland@ifi.uio.no

W. Martens  
Universität Bayreuth, Bayreuth, Germany  
e-mail: wim.martens@uni-bayreuth.de

to decide whether the query by a given NSTA is definable by a node selecting string automaton.

**Keywords** Automata · Twigs · Complexity · Definability

## 1 Introduction

As node selecting queries are fundamental in the context of tree-structured data like XML and web documents, many formalisms expressing such unary queries over trees have been investigated over time. Surprisingly many formalisms have been proposed which are expressively equivalent to the unary queries definable in monadic second-order logic (MSO) turning the latter into a yardstick for expressiveness over tree-structured data. We refer to these queries as the *regular unary queries*. Expressively equivalent formalisms are for instance based on attribute grammars [28, 29], automata [13, 16, 31], and logic [12, 19, 30]. Though expressive, well-understood, and robust, regular unary queries lack the simplicity and usability of less expressive languages like for instance XPath. Furthermore, a major advantage of XPath is without doubt the large body of research on efficient evaluation, optimization, and static analysis (see, e.g. [5] for a survey) and the availability of implementations. As such results for general unary regular queries are scarce, the goal of the present paper is to investigate the problem to decide whether a given regular unary query can in fact already be defined in an XPath-like formalism.

The proposed type of research has attracted a lot of attention in the area of logic and automata. There, a logic is said to have a decidable characterization if the following decision problem is decidable: “Given as input a finite automaton, decide if the recognized language can be defined using a formula of the logic”. Although quite a bit of research is available for logics over trees (cf., e.g., [6, 10, 37]), the most directly related result is by Place and Segoufin who showed that it is decidable whether a regular unranked tree language is definable in  $FO_2$  over the descendant and the following-sibling axes [33]. In terms of expressive power the latter logic corresponds to a fragment of the navigational core of XPath that contains modalities for going up to some ancestor, down to some descendant, left to some preceding sibling, and right to some following sibling. The devised decision problem leads to a high complexity with several nested exponentials. Although it is open whether this high complexity is unavoidable, in this paper, we do not consider  $FO_2$  over trees but restrict our attention to some of its fragments. Another related result is the one by Bojańczyk and Walukiewicz [11] showing that Boolean definability in the logic EX+EF is decidable in EXPTIME w.r.t. a given nondeterministic binary tree automaton. In short, the logic EX+EF is defined over binary trees, expresses the child and ancestor relation and is closed under the Boolean connectives. Specifically, we consider regular path queries and XPath with child, descendant and filter.<sup>1</sup> We refer to the latter as

---

<sup>1</sup>Filter is sometimes also called *predicate*, e.g., in the XPath specification by the World Wide Web Consortium.

twig queries. These twig queries are incomparable to EX+EF as they are defined over unranked trees and can define unary queries but are not closed under Boolean operations.

To represent unary regular queries, we employ the class of node selecting tree automata (NSTA) as defined in [16, 26] extended with wildcards. Basically, an NSTA is a non-deterministic unranked tree automaton with a distinguished set of *selecting* states. A node is then selected by an NSTA when it is visited in a selecting state during an accepting run. The output of the automaton consists of all selected nodes.

A regular path query selects a node based on regular properties of its ancestor-string, that is, the string formed by the labels on the path from the root to that node. We formalize the latter as NFA-definable queries. Specifically, an NFA can express a unary query by selecting every node which is visited in an accepting state on the path from the root to that node. We characterize the NFA-definable regular queries as those regular queries which are ancestor-based. The latter is a formalization of the idea that NFA-definable queries cannot distinguish between nodes with the same ancestor-string. Using this insight, we construct an NFA  $NFA(M)$  for a given NSTA  $M$ , such that  $M$  is equivalent to  $NFA(M)$  if and only if the query defined by  $M$  is NFA-definable. We then show that the latter equivalence test can be performed in exponential time. Altogether, we show that testing NFA-definability of NSTAs is EXPTIME-complete. We further discuss the relationship with ancestor-based types for XML schema languages as defined in [23] and address tractability.

Next, we turn to twig queries which are tree-patterns consisting of child and descendant edges. These correspond to the fragment of XPath restricted to child-axis, descendant-axis and filter. We show that NSTAs can be exponentially more succinct than twig queries. However, the large size of such twigs is due to a high degree of duplication which can be significantly reduced by folding them. We refer to the latter as DAG-twigs where DAG stands for a directed acyclic graph. In particular, we show that when an NSTA is twig-definable, there always exists an equivalent DAG-twig of at most linear size. To test twig-definability of NSTAs, one can simply guess a DAG-twig of linear size and test equivalence with the given NSTA. We show that the latter equivalence test can be done in EXPTIME through a reduction to emptiness of alternating tree-walking automata. The main result of this paper is that testing twig-definability of NSTAs is complete for EXPTIME.

*Related Work* Various properties of XPath have been investigated in the literature as for instance, its complexity, containment, and expressiveness. The complexity of XPath and efficient evaluation algorithms are investigated in, e.g., [9, 20, 21]. The containment and satisfiability problems for XPath have been deeply studied in the database literature, for example in [8, 25, 32, 36]. The expressiveness of various fragments and extensions of XPath have been investigated in, e.g., [4, 24, 37]. We refer to [5, 34] for surveys on these problems. To the best of our knowledge the above mentioned results of Place and Segoufin [33] and Bojańczyk and Walukiewicz [11] are the only research which studies decidability of XPath definability.

*Outline* In Section 2, we introduce the necessary definitions. In Section 3, we discuss regular path-definability of NSTAs. In Section 4, we discuss twig-definability of NSTAs. We conclude in Section 5.

## 2 Definitions

Here, we introduce the necessary definitions concerning trees, queries and automata. For a finite set  $S$ , we denote by  $|S|$  its number of elements.

### 2.1 Trees

Let  $\Delta$  always denote an infinite set of *labels*. Intuitively,  $\Delta$  is our abstraction of the set of XML-tags. We assume that we can test equality between elements from  $\Delta$  in constant time. We denote by  $\Delta^*$  the set of finite strings over  $\Delta$ . By  $\varepsilon$  we denote the empty string. We only consider rooted, ordered, finite, labelled, unranked trees which are directed from the root downwards. That is, we consider trees with a finite number of nodes and in which nodes can have arbitrarily many children. We view a tree  $t$  as a relational structure over a finite number of unary labelling relations  $\sigma(\cdot)$ , where each  $\sigma \in \Delta$ , and binary relations  $\text{child}(\cdot, \cdot)$  and  $\text{next-sibling}(\cdot, \cdot)$ . Here,  $\sigma(u)$  expresses that  $u$  is a node with label  $\sigma$ , and  $\text{child}(u, v)$  (respectively,  $\text{next-sibling}(u, v)$ ) expresses that  $v$  is a child (respectively, the next sibling) of  $u$ . When  $\text{next-sibling}(u, v)$  holds, we sometimes also say that  $v$  is (immediately) to the right of  $u$ .

We write  $\text{Nodes}^t$  for the set of nodes of  $t$ . The set of edges of a tree  $t$ , denoted by  $\text{Edges}^t$  is the set of pairs  $(u, v)$  such that  $\text{child}(u, v)$  holds in  $t$ . The root node of  $t$  is denoted by  $\text{root}(t)$ . We define the *size of  $t$* , denoted by  $|t|$ , to be the number of nodes of  $t$ . By  $\sigma(t_1, \dots, t_n)$  we denote a tree with root labelled  $\sigma$  and subtrees  $t_1, \dots, t_n$  attached below the root from left to right. We sometimes overload notation and denote by  $\sigma$  not only the  $\Delta$ -symbol but also the tree that has a  $\sigma$ -labelled root and no children below the root. By  $\mathcal{T}_\Delta$  we denote the set of all trees.

A *path* in tree  $t$  is a sequence of nodes  $v_0 \cdots v_n$  such that, for each  $i = 1, \dots, n$ , we have that  $(v_{i-1}, v_i) \in \text{Edges}^t$ . Paths therefore never run upwards, that is, turn towards to the root of  $t$ . We say that  $v_0 \cdots v_n$  is a path *from  $v_0$  to  $v_n$*  and that the *length* of the path is  $n$ . The *depth* of a node  $v \in \text{Nodes}^t$  is equal to the length of the (unique) path from  $\text{root}(t)$  to  $v$ . The *height* of a tree  $t$  is then defined as the maximum of the depths of all its nodes.

The label of each node  $v$  in  $t$  must be defined and unique, that is, for each node  $v \in \text{Nodes}^t$  there exists a unique  $\sigma \in \Delta$  such that  $\sigma(v)$  holds. We denote the label of  $v$  in  $t$  by  $\text{lab}^t(v)$ . For a node  $v$  in a tree  $t$ , the *ancestor-string* of  $v$ , denoted  $\text{ancstr}^t(v)$ , is the concatenation of the labels on all the nodes on the path from the root to  $v$ , including the two latter nodes. More specifically,  $\text{ancstr}^t(v)$  is the sequence  $\text{lab}^t(v_0) \cdots \text{lab}^t(v_n)$ , where  $v_0 \cdots v_n$  is the path from  $\text{root}(t)$  to  $v$ . For a tree  $t$  and a node  $v \in \text{Nodes}^t$ , the *subtree of  $t$  at  $v$* , denoted by  $\text{subtree}^t(v)$ , is the tree induced

by all the nodes  $u$  such that there is a (possibly empty) path from  $v$  to  $u$ . In particular, for any tree  $t$  and leaf node  $v$ ,  $\text{subtree}^t(v) = \text{lab}^t(v)$  and, for any other node  $u$ ,  $\text{subtree}^t(u) = \text{lab}^t(u)(\text{subtree}^t(u_1), \dots, \text{subtree}^t(u_n))$ , where  $u_1, \dots, u_n$  are the children of  $u$  from left to right.

Similarly, the *context of  $t$  at  $v$* , denoted by  $\text{context}^t(v)$ , is the tree induced by  $v$  and all the nodes that are *not* reachable by a path from  $v$  and which has a special marker at the position of  $v$ . In particular,  $\text{context}^t(v)$  is defined inductively as follows. Let  $\text{context}^t(\text{root}(t)) = \#$  for some  $\# \notin \Delta$ . If  $v$  is not the root of  $t$ , let  $u$  be the parent of  $v$  and let the children of  $u$  be  $v_1, \dots, v_n$ , from left to right. Assume that  $v = v_i$ . Then,  $\text{context}^t(v)$  is the tree obtained by replacing the unique  $\#$ -labelled node in  $\text{context}^t(u)$  with the tree

$$\text{lab}^t(u) (\text{subtree}^t(v_1), \dots, \text{subtree}^t(v_{i-1}), \#, \text{subtree}^t(v_{i+1}), \dots, \text{subtree}^t(v_n))$$

By  $t[v \leftarrow t']$  we denote the tree constructed from  $t$  by replacing  $\text{subtree}^t(v)$  at node  $v$  with  $t'$ . In other words, assuming w.l.o.g. that the sets of nodes in  $t$  and  $t'$  are disjoint,  $t[v \leftarrow t']$  is the tree obtained by replacing the  $\#$ -labelled node in  $\text{context}^t(v)$  with the tree  $t'$ .

### 2.2 Expressions and Automata

Throughout the paper,  $\Sigma \subseteq \Delta$  always denotes a finite alphabet. The set of *regular expressions* with symbols from a finite alphabet  $\Sigma$  is denoted by  $\mathcal{R}_\Sigma$ . We use standard regular expressions using the operators  $\cdot$  (concatenation),  $+$  (disjunction), and  $*$  (Kleene star). For a regular expression  $r$ ,  $L(r)$  is the language of the expression, and  $\text{Labels}(r)$  is the set of labels occurring in  $r$ . The *size* of a regular expression  $r$ , denoted by  $|r|$ , is defined as the length of its string representation.

The twig pattern queries we consider in this paper (see Section 4 for a formal definition) use only a finite set of labels, but the trees that match it can use arbitrary labels from an infinite set. This is to conform with the XPath query language, on which twig pattern queries are inspired. However, automata usually only use a finite alphabet of labels. To overcome this discrepancy we will use a wildcard symbol “ $\diamond$ ” that will give automata the same power. We assume that the single-symbol wildcard symbol  $\diamond$  is not in  $\Delta$  and we denote  $\Sigma \uplus \{\diamond\}$  by  $\Sigma_\diamond$ .

We define non-deterministic finite automata (NFAs) and their languages in the usual way, with the additional feature of a wildcard symbol that can match any  $\Delta$ -symbol not in  $\Sigma$ .

An  $\text{NFA}_w$  (*NFA with wildcards*) is a tuple  $A = (\Sigma, Q, q_I, \delta, F)$ , where  $Q$  is the finite set of states,  $q_I \in Q$  is the initial state,  $F \subseteq Q$  is the set of final states, and  $\delta : Q \times \Sigma_\diamond \rightarrow 2^Q$  is the transition function.

From the transition function  $\delta$ , we define the extended transition function  $\delta^* : (Q \times \Delta^*) \rightarrow 2^Q$  which can read entire  $\Delta$ -strings. In particular,  $\delta^*(q, \epsilon) = \{q\}$ ,  $\delta^*(q, a) = \delta(q, a)$  if  $a \in \Sigma$ ,  $\delta^*(q, a) = \delta(q, \diamond)$  if  $a \in \Delta - \Sigma$ , and  $\delta^*(q, a \cdot w) = \cup_{q' \in \delta^*(q, a)} \delta^*(q', w)$ , where  $a \in \Delta$  and  $w \in \Delta^*$ . A word  $w \in \Delta^*$  is accepted by  $A$  if

$\delta^*(q_I, w) \cap F \neq \emptyset$ . The set of words accepted by  $A$  is denoted by  $L(A)$ . The *size* of  $A$ , denoted by  $|A|$ , is defined as  $|Q| + \sum_{q \in Q, a \in \Sigma_\diamond} |\delta(q, a)|$ .

A *nondeterministic tree automaton with wildcards* or  $\text{NTA}_w$  is a tuple  $N = (\Sigma, Q, \delta, F)$  where  $Q$  is a finite set of states,  $F \subseteq Q$  is the set of final states, and the transition function  $\delta : Q \times \Sigma_\diamond \rightarrow \mathcal{R}_Q$  is a mapping from pairs of a state and a label to regular expressions over  $Q$ . Again, transitions labelled by  $\diamond$  can be followed by reading any symbol not in  $\Sigma$ .

A *run* of  $N$  on a tree  $t$  is an assignment of states to nodes  $\lambda : \text{Nodes}^t \rightarrow Q$  such that, for every  $v \in \text{Nodes}^t$  with  $n$  children  $v_1, \dots, v_n$  from left to right, the following holds: if  $\text{lab}^t(v) \in \Sigma$ , then

$$\lambda(v_1) \cdots \lambda(v_n) \in L(\delta(\lambda(v), \text{lab}^t(v))). \tag{2.1}$$

and if  $\text{lab}^t(v) \in \Delta - \Sigma$ , then

$$\lambda(v_1) \cdots \lambda(v_n) \in L(\delta(\lambda(v), \diamond)). \tag{2.2}$$

When  $v$  has no children, the criterion reduces to

$$\epsilon \in L(\delta(\lambda(v), \text{lab}^t(v))) \quad \text{or} \quad \epsilon \in L(\delta(\lambda(v), \diamond)).$$

A run is *accepting* if the root is labelled with a state in  $F$ . A tree  $t$  is *accepted* by  $N$  if there is an accepting run of  $N$  on  $t$ . The set of all trees accepted by  $N$  is denoted by  $L(N)$ . If  $L(N) = \mathcal{T}_\Delta$ , we call  $N$  *universal*. The *size* of  $N$  is defined as  $|Q| + \sum_{q \in Q, a \in \Sigma_\diamond} |\delta(q, a)|$ . We say that two  $\text{NTA}_w$ s are *equivalent* if they define the same language.

For any  $p \in Q$ , let  $N_p = (\Sigma, Q, \delta, \{p\})$ . We call  $p$  *universal in  $N$*  if  $N_p$  is universal.

We say that a state  $p$  is *reachable* from a state  $q$ , if  $p = q$ , or if there is an  $a \in \Delta$  and  $w_1 q' w_2 \in L(\delta(q, a))$  such that  $p$  is reachable from  $q'$ .

Notice that, in our automata definitions, we could have let a wildcard match any  $\Delta$ -symbol rather than any symbol in  $\Delta - \Sigma$ . While at first sight it may seem more natural to some to let the wildcard match any  $\Delta$ -symbol, we note that the variant we chose here is more powerful. Indeed, the former semantics can be simulated by the latter (by simply adding an extra transition for every  $\Sigma$ -symbol) but not vice versa. For example, the latter variant can test if a label is in  $\{a\} \cup (\Delta - \Sigma)$  with  $a \in \Sigma$ , whereas the former cannot. We therefore chose the more powerful variant.

Unless explicitly mentioned otherwise, we will assume that  $\text{NTA}_w$ s do not have *useless* states. That is, for each state  $q$ , there is at least one accepting run  $\lambda$  of the  $\text{NTA}_w$  on some tree  $t$  where  $\lambda(u) = q$  for some node  $u$  of  $t$ . We justify this assumption by the following lemma.

**Lemma 2.1** *Each  $\text{NTA}_w$  can be converted into an equivalent  $\text{NTA}_w$  without useless states in polynomial time.*

*Proof* An  $\text{NTA}_w B = (\Sigma, Q, \delta, F)$  can be converted into an equivalent  $\text{NTA}_w$  without useless states in polynomial time as follows:

- (1) Compute the set  $E$  of states  $q$  such that  $L(B_q) = \emptyset$ .
- (2) Replace every occurrence of a symbol in  $E$  in all regular expressions in the definition of  $\delta$  by  $\emptyset$  and remove all symbols of  $E$  from  $Q$  and  $F$ .
- (3) In the resulting  $\text{NTA}_w$ , compute the set  $U$  of states that are not reachable from a state in  $F$ .
- (4) Remove all states of  $U$  from all regular expressions in the definition of  $\delta$  and remove them from  $Q$ .

We note that this procedure closely follows the procedure that removes useless non-terminals from an extended context-free grammar which was presented in [1, 22] and where it is also explained that it can be performed in polynomial time. We note that the ordering between steps (2) and (3) is important. If this ordering is reversed, the algorithm does not remove all useless states from the  $\text{NTA}_w$  with  $F = \{q_0\}$  and transition function defined as  $\delta(q_0, a) = q_1q_2$ ,  $\delta(q_1, a) = q_1$ , and  $\delta(q_2, a) = \varepsilon$ .  $\square$

Since useless states can be removed efficiently, we also do not need to bother about removing useless states in the  $\text{NTA}_w$ s we construct in our algorithms.

The proof of the following theorem is a straightforward reduction to and from the finite alphabet case [35]. To represent the automata as used in [35], we use the same structure as for  $\text{NTA}_w$ s. We call the latter *NTAs without wildcards*, and their semantics is just as for  $\text{NTA}_w$ s, except that  $\diamond$  is given no special meaning. That is,  $\diamond$  is allowed in  $\Sigma$ , and the rule (2.2) is not used.

**Theorem 2.2** 1. *Deciding equivalence of  $\text{NTA}_w$ s is EXPTIME-complete.*  
 2. *Deciding universality of  $\text{NTA}_w$ s is EXPTIME-complete.*

*Proof* We prove the EXPTIME upper bound for equivalence and the EXPTIME lower bound for universality. Since an  $\text{NTA}_w$  is universal if and only if it is equivalent to some (fixed)  $\text{NTA}_w$  that accepts  $\mathcal{T}_\Delta$ , the whole theorem follows.

We first show that equivalence is in EXPTIME. Let  $N_1$  and  $N_2$  be two  $\text{NTA}_w$ s over alphabet  $\Sigma_1$  and  $\Sigma_2$ , respectively. In particular, let  $N_1 = (\Sigma_1, Q_1, \delta_1, F_1)$  and  $N_2 = (\Sigma_2, Q_2, \delta_2, F_2)$ . Notice that  $\Sigma_1$  might be different from  $\Sigma_2$ . We define two NTAs (without wildcards)  $M_1$  and  $M_2$  over the alphabet  $\Gamma = \Sigma_1 \cup \Sigma_2 \cup \{\diamond\}$  as follows. The automaton  $M_1 = (\Gamma, Q_1, \delta'_1, F_1)$  has the following transitions for each  $q \in Q_1$  and  $\sigma \in \Gamma$ :

1. if  $\sigma \in \Sigma_1 \cup \{\diamond\}$ , then  $\delta'_1(q, \sigma) = \delta_1(q, \sigma)$ , and
2. if  $\sigma \in \Sigma_2 - \Sigma_1$ , then  $\delta'_1(q, \sigma) = \delta_1(q, \diamond)$ .

The automaton  $M_2 = (\Gamma, Q_2, \delta'_2, F_2)$  and  $\delta'_2$  are defined similarly.

We want to show that  $M_1$  is equivalent to  $M_2$  (over  $\mathcal{T}_\Gamma$ ) if and only if  $N_1$  is equivalent to  $N_2$  (over  $\mathcal{T}_\Delta$ ). For any tree  $t \in \mathcal{T}_\Delta$ , let  $t_\diamond \in \mathcal{T}_\Gamma$  be the tree obtained from  $t$  by replacing the label of any node that is not in  $\Sigma_1 \cup \Sigma_2$ , with  $\diamond$ . Since the

transitions of  $N_i$  and  $M_i$  agree on all labels  $\sigma \notin \Sigma_1 \cup \Sigma_2$ , it holds that for any tree  $t \in \mathcal{T}_\Delta$  and run  $\lambda$  on  $t$ ,  $\lambda$  is an accepting run of  $N_i$  on  $t$  if and only if it is an accepting run of  $M_i$  on  $t_\diamond$ , for  $i \in \{1, 2\}$ . Let  $t$  be a tree in  $\mathcal{T}_\Gamma$ .

For the *only if* direction, suppose that  $t \in L(N_1)$ . Then there exists an accepting run  $\lambda$  of  $N_1$  on  $t$ . By construction of  $M_1$ ,  $\lambda$  is also an accepting run of  $M_1$  on  $t_\diamond$ , and hence  $t_\diamond \in L(M_1)$ . By assumption that  $M_1$  and  $M_2$  are equivalent, there is an accepting run  $\lambda'$  of  $M_2$  on  $t_\diamond$ . By construction of  $M_2$ ,  $\lambda'$  is also an accepting run of  $N_2$  on any tree  $t'$  which is such that  $t'_\diamond = t_\diamond$ . Hence,  $\lambda'$  is also an accepting run of  $N_2$  on  $t$ . The case is similar when  $t \in L(N_2)$ .

For the *if* direction, suppose that  $t \in L(M_1)$ . Then there is an accepting run  $\lambda$  of  $M_1$  on  $t$ . Then, by construction of  $M_1$ , the run  $\lambda$  is an accepting run of  $N_1$  on any tree  $t'$  which is such that  $t'_\diamond = t$ . By assumption that  $N_1$  and  $N_2$  are equivalent, it holds that  $t' \in L(N_2)$ , with  $\lambda'$  being the witness accepting run, and by construction of  $M_2$ ,  $\lambda'$  is an accepting run of  $M_2$  on  $t'_\diamond = t$ . The case is similar when  $t \in L(M_2)$ .

We continue by showing EXPTIME-hardness of universality. Let  $N$  be an NTA without wildcards over alphabet  $\Sigma$ . Then let  $M$  be an arbitrary but fixed  $\text{NTA}_w$  over the alphabet  $\Sigma$  which accepts any tree containing at least one symbol from  $\Delta - \Sigma$ . In other words,  $M$  accepts exactly the trees in  $\mathcal{T}_\Delta - \mathcal{T}_\Sigma$ . The  $\text{NTA}_w$   $M$  requires only two states that are not states of  $N$ . Then  $N$  is universal over  $\Sigma$  if and only if  $N \cup M$  is universal over  $\Delta$ . □

### 2.3 Queries

The focus of this paper is on unary queries. Basically, a unary query maps each tree to a subset of its nodes.

**Definition 2.3** (Unary Query) A unary query  $\mathcal{Q}$  is a mapping with domain  $\mathcal{T}_\Delta$  that is closed under isomorphism, and is such that for each  $t \in \mathcal{T}_\Delta$ ,  $\mathcal{Q}(t) \subseteq \text{Nodes}^t$ .

For two unary queries  $\mathcal{Q}, \mathcal{Q}'$ , and  $\odot \in \{\subseteq, \supseteq, =\}$ , we write  $\mathcal{Q} \odot \mathcal{Q}'$  if, for all  $t \in \mathcal{T}_\Delta$ , we have  $\mathcal{Q}(t) \odot \mathcal{Q}'(t)$ . In this paper we only consider unary queries and “query” will therefore mean “unary query”.

To facilitate proofs, in the following, we will sometimes reduce unary queries to Boolean ones. To this end, we will employ a standard technique (cf., e.g., [38]) which extends the set of labels by  $\Sigma \times \{0, 1\}$  and labels selected nodes by 1 and non-selected nodes by 0.

We will in the following, without loss of generality, assume that  $(\Delta \times \{0, 1\}) \subset \Delta$ . This assumption simplifies the text in our proofs.

**Definition 2.4** For a tree  $t \in \mathcal{T}_\Delta$ , we denote the set of nodes labelled by a symbol in  $\Sigma$ , as  $\text{Nodes}^t(\Sigma)$ . Then, let  $\text{bool}_\Sigma(\cdot, \cdot)$  be the mapping defined as follows. For each tree  $t \in \mathcal{T}_\Delta$  and  $v \in \text{Nodes}^t(\Sigma)$ , let  $\text{bool}_\Sigma(t, v) \in \mathcal{T}_\Delta$  be the tree with the same nodes and edges as  $t$ , but with the labelling function defined as follows:  $\text{lab}^{\text{bool}_\Sigma(t, v)}(v) = (\text{lab}^t(v), 1)$ , for  $v' \in \text{Nodes}^t(\Sigma) - \{v\}$ ,  $\text{lab}^{\text{bool}_\Sigma(t, v)}(v') =$



$(\text{lab}^t(v'), 0)$ , and for  $v' \notin \text{Nodes}^t(\Sigma)$ ,  $\text{lab}^{\text{bool}_\Sigma(t,v)}(v') = \text{lab}^t(v')$ . Finally, for a unary query  $\mathcal{Q}$ , let

$$\text{BoolQ}_\Sigma(\mathcal{Q}) = \bigcup_{\substack{t \in \mathcal{T}_\Delta \\ v \in \mathcal{Q}(t)}} \{\text{bool}_\Sigma(t, v)\}.$$

Then, the image of the function  $\text{bool}_\Sigma$  is:

$$\text{Image}(\text{bool}_\Sigma) = \bigcup_{\substack{t \in \mathcal{T}_\Delta \\ v \in \text{Nodes}^t(\Sigma)}} \{\text{bool}_\Sigma(t, v)\}.$$

**Lemma 2.5** *For any finite  $\Sigma \subseteq \Delta$ :*

1.  $\text{bool}_\Sigma$  is injective.
2.  $\text{BoolQ}_\Sigma$  is injective.

*Proof* (1) Let  $t_1, t_2 \in \mathcal{T}_\Delta$ ,  $v_1 \in \text{Nodes}^{t_1}$  and  $v_2 \in \text{Nodes}^{t_2}$ , and assume  $\text{bool}_\Sigma(t_1, v_1) = \text{bool}_\Sigma(t_2, v_2)$ . The latter implies immediately that  $t_1$  and  $t_2$  have the same nodes and edges, since  $\text{bool}_\Sigma$  preserves the structure of the tree. For each node  $v \in \text{Nodes}^{t_1}$ , since  $\text{lab}^{\text{bool}_\Sigma(t_1,v_1)}(v) = \text{lab}^{\text{bool}_\Sigma(t_2,v_2)}(v)$ , by definition also  $\text{lab}^{t_1}(v) = \text{lab}^{t_2}(v)$ . Finally, also  $v_1 = v_2$ , since otherwise  $\text{bool}_\Sigma(t_1, v_1)$  and  $\text{bool}_\Sigma(t_2, v_2)$  would have different labels at the nodes  $v_1$  and  $v_2$ .

- (2) To prove this we reformulate the definition of  $\text{BoolQ}_\Sigma$  to get

$$\text{BoolQ}_\Sigma(\mathcal{Q}) = \{\text{bool}_\Sigma(t, v) \mid (t, v) \in \mathcal{Q}\}$$

Assume unary queries  $\mathcal{Q}_1$  and  $\mathcal{Q}_2$  such that  $\text{BoolQ}_\Sigma(\mathcal{Q}_1) = \text{BoolQ}_\Sigma(\mathcal{Q}_2)$ . But since  $\text{bool}_\Sigma$  is injective this implies that  $\mathcal{Q}_1 = \mathcal{Q}_2$ . □

Lemma 2.5 implies that the mapping  $\text{bool}_\Sigma$  has an inverse over its image. We denote the inverse by  $\text{bool}_\Sigma^{-1}$ .

### 2.4 Selecting Tree Automata

The general formalism we use for expressing unary queries is that of selecting tree automata, which are defined as follows [16, 26].

**Definition 2.6** ( $\text{NSTA}_w$ ) *A non-deterministic selecting tree automaton (with wild-cards) or  $\text{NSTA}_w$   $M$ , is a pair  $(N, S)$ , where  $N$  is an  $\text{NTA}_w$  with state set  $Q$ , and  $S \subseteq Q$  is a set of selecting states. The query defined by  $M$  is denoted  $\mathcal{Q}_M$ . Formally,  $v \in \mathcal{Q}_M(t)$  if there is an accepting run  $\lambda$  such that  $\lambda(v) \in S$  and  $\text{lab}^t v \in \Sigma$ . Note that for all  $t \notin L(N)$ ,  $\mathcal{Q}_M(t) = \emptyset$ . The size of  $M$  is defined as the size of its underlying  $\text{NTA}_w$ .*

We refer to the class of queries defined by  $NSTA_w$ s as the (unary) regular queries. An  $NSTA_w$   $M$  is called *non-empty* if there is a  $t$  such that  $\mathcal{Q}_M(t) \neq \emptyset$ .

We say that two  $NSTA_w$ s are *equivalent* if they define the same query. The following theorem says that deciding equivalence of  $NSTA_w$ s is in EXPTIME. Although the results seems to belong to folklore, for the sake of completeness, we provide a proof below.

Specifically, Theorem 2.8 follows directly from the following lemma and Theorem 2.2.

**Lemma 2.7** *For any finite  $\Sigma \subseteq \Delta$  and any  $NSTA_w$ ,  $M = (N_M, S)$  over  $\Sigma$ , we can construct in polynomial time an  $NTA_w$   $N$  such that  $L(N) = \text{BoolQ}_\Sigma(\mathcal{Q}_M)$ , and such that  $|\mathcal{Q}_N| = 2 \cdot |\mathcal{Q}_{N_M}|$ , where  $\mathcal{Q}_N$  is the set of states of  $N$  and  $\mathcal{Q}_{N_M}$  is the set of states of  $N_M$ .*

*Proof* Let  $N_M = (\Sigma, Q, \delta, F)$ . We will construct two  $NTA_w$ s,  $N'$  and  $N''$ , such that  $L(N'') = \text{Image}(\text{bool}_\Sigma)$ , and  $L(N') \cap \text{Image}(\text{bool}_\Sigma) = \text{BoolQ}_\Sigma(\mathcal{Q}_M)$ . This implies that  $\text{BoolQ}_\Sigma(\mathcal{Q}_M) = L(N') \cap L(N'')$ . Since the intersection of  $NTA_w$ s can be done using a cross-product construction, the whole construction can be performed in polynomial time.

The  $NTA_w$   $N'' = (\Sigma \times \{0, 1\}, \{q_0, q_U\}, \delta'', \{q_0\})$ , where for each  $a \in \Sigma$ ,

$$\delta''(q_U, (a, 0)) = q_U^*, \delta''(q_0, (a, 0)) = q_U^*q_0q_U^*, \delta''(q_0, (a, 1)) = q_U^*,$$

and the wildcard transitions are  $\delta''(q_U, \diamond) = q_U^*$  and  $\delta''(q_0, \diamond) = q_U^*q_0q_U^*$ . To see that  $L(N'') = \text{Image}(\text{bool}_\Sigma)$  note first that  $N''_{q_U}$  recognizes all the trees where for each node, there is no  $a \in \Sigma$ , such that the label is  $(a, 1)$ . The state  $q_0$  can be assigned to a node if it has a label of the form  $(a, 1)$  for  $a \in \Sigma$ , or if it is not labelled  $(a, 1)$ , for  $a \in \Sigma$ , and  $q_0$  is assigned to exactly one of its children, and  $q_U$  is assigned to the other nodes. Hence,  $q_0$  is reached at a node if exactly one node in the tree has a label of the form  $(a, 1)$ , where  $a \in \Sigma$ .

The  $NTA_w$   $N' = (\Sigma \times \{0, 1\}, Q, \delta', F)$  has  $\delta'$  defined as follows: for each  $a \in \Sigma$  and  $q \in Q$ ,  $\delta'(q, (a, 0)) = \delta(q, a)$ , for each  $q \in S$ ,  $\delta'(q, (a, 1)) = \delta(q, a)$ , and for each  $q \in Q$ ,  $\delta'(q, \diamond) = \delta(q, \diamond)$ .

We first show that

$$L(N') \cap \text{Image}(\text{bool}_\Sigma) \subseteq \text{BoolQ}_\Sigma(\mathcal{Q}_M)$$

Assume  $t' \in L(N') \cap \text{Image}(\text{bool}_\Sigma)$ . Let  $(t, v) = \text{bool}_\Sigma^{-1}(t')$ . Hence  $\text{lab}^{t'}(v) = (\text{lab}^t(v), 1)$ , while for each  $v' \in \text{Nodes}^t(\Sigma) - \{v\}$ ,  $\text{lab}^{t'}(v') = (\text{lab}^t(v'), 0)$ . Let  $\lambda$  be the run of  $N'$  on  $t'$ . By construction,  $\lambda$  is also a run of  $(\Sigma, Q, \delta, F)$  on  $t$ , and  $\lambda(v) \in S$ . Hence  $v \in \mathcal{Q}_M(t)$ , and therefore  $\text{bool}_\Sigma(t, v) \in \text{BoolQ}_\Sigma(\mathcal{Q}_M)$ , that is,  $t' \in \text{BoolQ}_\Sigma(\mathcal{Q}_M)$ .

Secondly we show that

$$\text{BoolQ}_\Sigma(\mathcal{Q}_M) \subseteq L(N') \cap \text{Image}(\text{bool}_\Sigma)$$

First note that  $\text{BoolQ}_\Sigma(\mathcal{Q}_M) \subseteq \text{Image}(\text{bool}_\Sigma)$ , so we only need to show that

$$\text{BoolQ}_\Sigma(\mathcal{Q}_M) \subseteq L(N')$$

Assume  $t' \in \text{BoolQ}_\Sigma(\mathcal{Q}_M)$ . Let  $(t, v) = \text{bool}_\Sigma^{-1}(t')$ . Hence,  $v \in \mathcal{Q}_M(t)$ ,  $v \in \text{Nodes}^t(\Sigma)$ , and there is an accepting run  $\lambda$  of  $(\Sigma, Q, \delta, F)$  on  $t$  such that  $\lambda(v) \in S$ . But by construction  $\lambda$  is then also an accepting run of  $N'$  on  $t'$ .

Finally, since  $N'$  has  $|Q|$  states and  $N''$  has 2 states the cross product construction defining the intersection of their languages has  $2 \cdot |Q|$  states as required.  $\square$

**Theorem 2.8** *Deciding equivalence of NSTA<sub>w</sub>s is EXPTIME-complete.*

*Proof* For the lower bound, by Theorem 2.2 we can reduce the problem of NTA<sub>w</sub> equivalence to NSTA<sub>w</sub> equivalence. Given any NTA<sub>w</sub>  $N$ , we convert it to an NSTA<sub>w</sub>  $M$  with  $N$  as its underlying NTA<sub>w</sub>, that always selects the root of a tree in the language of  $N$ .

For the upper bound, given two NSTA<sub>w</sub>s  $M_1$  and  $M_2$ , we construct in polynomial time two NTA<sub>w</sub>s  $N_1$  and  $N_2$  as in Lemma 2.7, such that  $L(N_1) = \text{BoolQ}_\Sigma(\mathcal{Q}_{M_1})$  and  $L(N_2) = \text{BoolQ}_\Sigma(\mathcal{Q}_{M_2})$ . By Lemma 2.5  $\mathcal{Q}_{M_1} = \mathcal{Q}_{M_2}$  if and only if  $\text{BoolQ}_\Sigma(\mathcal{Q}_{M_1}) = \text{BoolQ}_\Sigma(\mathcal{Q}_{M_2})$ . By Theorem 2.2, we can check in EXPTIME whether  $N_1$  and  $N_2$  are equivalent.  $\square$

### 3 Regular Path Definability

In this section, we consider regular path definability. Here, we use NFA<sub>w</sub>s to define regular paths. More precisely, we investigate in Section 3.1 whether a query given by an NSTA<sub>w</sub> can already be defined by an NFA<sub>w</sub>. We further discuss in Section 3.2 the relationship with definability of single-type EDTDs. Finally, we address tractability in Section 3.3.

#### 3.1 NFA<sub>w</sub>-Definability

We first formally introduce queries defined by NFA<sub>w</sub>s.

**Definition 3.1** (NFA<sub>w</sub>-definable Query) The query defined by an NFA<sub>w</sub>  $A$  is denoted by  $\mathcal{Q}_A$  and is defined as follows. For any tree  $t \in \mathcal{T}_\Delta$ ,  $\mathcal{Q}_A(t) = \{v \in \text{Nodes}^t \mid \text{lab}^t(v) \in \Sigma, \text{ancstr}^t(v) \in L(A)\}$ . We say that a query  $\mathcal{Q}$  is NFA<sub>w</sub>-definable, if there is an NFA<sub>w</sub>  $A$  such that  $\mathcal{Q} = \mathcal{Q}_A$ .

As selection of a node only depends on the ancestor-string, NFA<sub>w</sub>-definable queries are ancestor-based as defined next:

**Definition 3.2** (Ancestor-based Query) A unary query  $\mathcal{Q}$  is ancestor-based if for each two trees  $t_1, t_2 \in \mathcal{T}_\Delta$ , and for any nodes  $v_1 \in \text{Nodes}^{t_1}$  and  $v_2 \in \text{Nodes}^{t_2}$ , if  $v_1 \in \mathcal{Q}(t_1)$  and  $\text{ancstr}^{t_1}(v_1) = \text{ancstr}^{t_2}(v_2)$ , then also  $v_2 \in \mathcal{Q}(t_2)$ .

It is easy to see that each  $NFA_w$ -definable unary query must be ancestor-based.

**Lemma 3.3** *If a unary query is  $NFA_w$ -definable, then it is also ancestor-based.*

*Proof* Let the unary query  $\mathcal{Q}$  be definable by an  $NFA_w$   $A$ , i.e.,  $\mathcal{Q} = \mathcal{Q}_A$ , and assume  $t_1, t_2 \in \mathcal{T}_\Delta$  such that  $v_1 \in \mathcal{Q}(t_1)$ ,  $v_2 \in \text{Nodes}^{t_2}$  and  $\text{ancstr}^{t_1}(v_1) = \text{ancstr}^{t_2}(v_2)$ . Since  $v_1 \in \mathcal{Q}_A(t_1)$ , by definition,  $\text{ancstr}^{t_1}(v_1) \in L(A)$ , and since  $\text{ancstr}^{t_1}(v_1) = \text{ancstr}^{t_2}(v_2)$ , also  $\text{ancstr}^{t_2}(v_2) \in L(A)$ , hence,  $v_2 \in \mathcal{Q}(t_2)$ .  $\square$

In general, the converse of Lemma 3.3 does not hold. For example, the query “select all nodes  $v$  such that  $\text{ancstr}^t(v)$  has an equal number of  $a$ ’s and  $b$ ’s” is ancestor-based but not  $NFA_w$ -definable. We will show in the remainder of this section, that ancestor-based *regular* queries do correspond precisely to the  $NFA_w$ -definable ones. The proof makes use of a specific construction on  $NSTA_w$ s. In particular, for a given  $NSTA_w$   $M$  we construct an automaton  $NFA(M)$  such that  $M$  is  $NFA_w$ -definable iff  $\mathcal{Q}_{NFA(M)} = \mathcal{Q}_M$ .

Basically, the automaton  $NFA(M)$  is constructed from  $M$  by turning it into an  $NFA_w$ . That is, a state at a node is only dependent on the state assigned to its parent (and no longer dependent on the states assigned to its siblings). Specifically, any state in  $\text{Labels}(\delta_M(q, a))$  can be assigned to a node whose parent is labelled  $a$  and is assigned state  $q$  where  $\delta_M$  is the transition function of  $M$ .<sup>2</sup> The formal construction is given next:

**Definition 3.4** For an  $NSTA_w$   $M = (N, S)$ , where  $N = (\Sigma, Q, \delta, F)$ , and for  $q_I \notin Q$ , define the  $NFA_w$

$$NFA(M) = (\Sigma, (Q \times \Sigma_\diamond) \cup \{q_I\}, q_I, \delta', F'),$$

where  $\Sigma_\diamond = \Sigma \uplus \{\diamond\}$  and

$$F' = \{(p, a) \mid p \in S, a \in \Sigma_\diamond \text{ and } \delta(p, a) \text{ is defined and not empty}\},$$

for each  $a \in \Sigma_\diamond$ , let  $\delta'(q_I, a) = \{(p, a) \mid p \in F\}$ , and for  $q \in Q$  and  $b \in \Sigma_\diamond$ , let

$$\delta'((q, a), b) = \{(p, b) \mid p \in \text{Labels}(\delta(q, a))\}.$$

The query defined by  $NFA(M)$  is always *complete*, that is, it always selects at least the nodes that are selected by  $M$ . Furthermore, if the query defined by  $M$  is ancestor-based, then we have that  $NFA(M)$  is *sound* as well, i.e., each node selected by  $NFA(M)$  is also selected by  $M$ .

To facilitate the proofs below, we introduce the notation  $\text{lab}^{t, \Sigma}(v)$ . If  $\text{lab}^t(v) \in \Sigma$ , then let  $\text{lab}^{t, \Sigma}(v) = \text{lab}^t(v)$ . Otherwise, let  $\text{lab}^{t, \Sigma}(v) = \diamond$ .

**Lemma 3.5** *Let  $M$  be an  $NSTA_w$ . Then the following holds:*

1.  $\mathcal{Q}_M \subseteq \mathcal{Q}_{NFA(M)}$ ; and,

<sup>2</sup>Recall that  $\text{Labels}(r)$  is the set of symbols occurring in regular expressions  $r$ .

2. if  $\mathcal{Q}_M$  is ancestor-based then  $\mathcal{Q}_{\text{NFA}(M)} \subseteq \mathcal{Q}_M$ .

*Proof* (1) Let  $M$  and  $\text{NFA}(M)$  be as in Definition 3.4. Assume  $v \in \mathcal{Q}_M(t)$  for some  $v$  and  $t$ . We will prove that also  $v \in \mathcal{Q}_{\text{NFA}(M)}(t)$ . Let  $v_1 \cdots v_n$  be the path from the root to  $v$  in  $t$ . (Hence  $v_1 = \text{root}(t)$  and  $v_n = v$ ). Since  $v \in \mathcal{Q}_M(t)$ , there is an accepting run  $\lambda$  of  $M$  on  $t$  which selects  $v$ , that is, such that  $\lambda(v) \in S$ . Let  $q_1, \dots, q_n$  be the states assigned to the nodes  $v_1, \dots, v_n$  by such an accepting run. Hence  $q_1 \in F$  and  $q_n \in S$ . By induction on  $i$ , where  $1 \leq i \leq n$ , we prove that  $(q_i, \text{lab}^{t, \Sigma}(v_i)) \in \delta'^*(q_I, \text{ancstr}^t(v_i))$ .

– For the base case  $i = 1$  it suffices to show that

$$(q_1, \text{lab}^{t, \Sigma}(v_1)) \in \delta'^*(q_I, \text{lab}^t(v_1))$$

But this follows from construction of  $\delta'$ , the extended transition function  $\delta'^*$ , and that  $q_1 \in F$ .

– For the induction case where  $i > 1$ , we can assume, by the induction hypothesis, that

$$(q_{i-1}, \text{lab}^{t, \Sigma}(v_{i-1})) \in \delta'^*(q_I, \text{ancstr}^t(v_{i-1}))$$

and it remains to prove that

$$(q_i, \text{lab}^{t, \Sigma}(v_i)) \in \delta'^*((q_{i-1}, \text{lab}^t(v_{i-1})), \text{lab}^t(v_i))$$

By construction of  $\delta'$  and  $\delta'^*$ , this holds if

$$q_i \in \text{Labels}(\delta(q_{i-1}, \text{lab}^{t, \Sigma}(v_{i-1})))$$

The latter holds, since  $\lambda$  is an accepting run.

Since  $q_n \in S$  and  $\lambda$  is an accepting run,  $(q_n, \text{lab}^{t, \Sigma}(v_n)) \in F'$ . This implies  $\text{ancstr}^t(v) \in L(\text{NFA}(M))$ , and hence  $v \in \mathcal{Q}_{\text{NFA}(M)}(t)$ .

(2) Let  $M = (N, S)$ , where  $N = (\Sigma, Q, \delta, F)$ , and

$$\text{NFA}(M) = (\Sigma, (Q \times \Sigma_\diamond) \cup \{q_I\}, q_I, \delta', F')$$

as in Definition 3.4. Assume that for some tree  $t$  and node  $v \in \text{Nodes}^t$ ,  $v \in \mathcal{Q}_{\text{NFA}(M)}(t)$ . We will show that there exists a tree  $t'$  and  $v' \in \text{Nodes}^{t'}$ , such that  $v' \in \mathcal{Q}_M(t')$  and  $\text{ancstr}^t(v) = \text{ancstr}^{t'}(v')$ . By our assumption that  $\mathcal{Q}_M$  is ancestor-based, it will follow that  $v \in \mathcal{Q}_M(t)$ .

Let  $v_1, \dots, v_n$  be the nodes on the path from the root to  $v$  (including the root and  $v$ ), and let  $w = \text{ancstr}^t(v)$ . Furthermore, let

$$q_I, (p_1, \text{lab}^{t, \Sigma}(v_1)), \dots, (p_n, \text{lab}^{t, \Sigma}(v_n))$$

be the states visited by  $\text{NFA}(M)$  (in order) when matching  $w$ . Specifically, this means  $(p_n, \text{lab}^{t, \Sigma}(v_n)) \in F'$ ,  $p_n \in S$ , and  $p_1 \in F$ .

Recall that we have defined  $N_p = (\Sigma, Q, \delta, \{p\})$  for  $p \in Q$  as the  $\text{NTA}_w N$  with single final state  $p$ . We prove (below) by induction on  $i$ ,  $0 \leq i < n$ , that there is a tree

$t'_{n-i} \in L(N_{p_{n-i}})$  and a  $v'_{n-i} \in \mathcal{Q}_{(N_{p_{n-i}}, S)}(t'_{n-i})$  such that  $\text{lab}^t(v_{n-i}) \cdots \text{lab}^t(v_n) = \text{ancstr}^{t'_{n-i}}(v'_{n-i})$ , and that there is a run of  $N_{p_{n-i}}$  on  $t'_{n-i}$  where the nodes on the path from the root to  $v'_{n-i}$  are assigned the states  $p_{n-i}, \dots, p_n$ , respectively.

- The base case  $i = 0$  is easy, since we know  $p_n \in S$  and  $(p_n, \text{lab}^{t, \Sigma}(v_n)) \in F'$ , and by construction the latter implies  $\delta(p_n, \text{lab}^{t, \Sigma}(v_n))$  is defined. Therefore, there is a tree  $t'_n$  with its root labelled with  $\text{lab}^t(v_n)$ , such that  $t'_n \in N_{p_n}$  as required.
- For the induction case, we can by the induction hypothesis assume the statement holds for  $i \geq 0$ , and we prove it for  $i + 1 < n$ . From the run of  $\text{NFA}(M)$  on  $w$  we must have that  $(p_{n-i}, \text{lab}^{t, \Sigma}(v_{n-i}))$  is in

$$\delta'((p_{n-i-1}, \text{lab}^{t, \Sigma}(v_{n-i-1})), \text{lab}^{t, \Sigma}(v_{n-i}))$$

By definition of the transition function  $\delta'$ , this implies

$$p_{n-i} \in \text{Labels}(\delta(p_{n-i-1}, \text{lab}^{t, \Sigma}(v_{n-i-1}))).$$

In particular, there is a string  $w_q = q_1 \cdots p_{n-i} \cdots q_r$ , such that

$$w_q \in L(\delta(p_{n-i-1}, \text{lab}^{t, \Sigma}(v_{n-i-1})))$$

Since there are no useless states, for each state  $q_j$  other than  $p_{n-i}$  in  $w_q$  there is a tree  $s_{q_j}$  such that  $s_{q_j} \in L(N_{q_j})$ , and by the inductive hypothesis, there is a tree  $t'_{n_i} \in L(N_{p_{n-i}})$  with the required properties. Then let

$$t'_{n-i-1} = \text{lab}^t(v_{n-i-1})(s_{q_1} \cdots t'_{n-i} \cdots s_{q_r}).$$

This tree satisfies the induction hypothesis statement.

Since  $\{p_1\} \subseteq F$ , it holds that  $L(N) \supseteq L(N_{p_1})$ , and therefore  $v'_1 \in \mathcal{Q}_M(t'_1)$ , as required. □

The following lemma relates  $\text{NFA}_w$ -definability and ancestor-based regular queries.

**Lemma 3.6** *For an  $\text{NSTA}_w$   $M$ , the following are equivalent*

1.  $\mathcal{Q}_M$  is  $\text{NFA}_w$ -definable;
2.  $\mathcal{Q}_M$  is ancestor-based; and,
3.  $\mathcal{Q}_M = \mathcal{Q}_{\text{NFA}(M)}$ .

*Proof* (1)  $\Rightarrow$  (2) holds by Lemma 3.3.

(2)  $\Rightarrow$  (3) holds by Lemma 3.5.

(3)  $\Rightarrow$  (1) holds by definition of  $\text{NFA}_w$ -definable query. □

We construct an  $NSTA_w$  defining the same query as an  $NFA_w$ . We remark that there is a difference of one step between how an  $NSTA_w$  and an  $NFA_w$  matches an ancestor-string: The path employed by an  $NFA_w$  while recognizing a word, is by one longer than the word (first the initial state, then one state for each letter), while the  $NSTA_w$  only labels each node with one state. We therefore need to introduce a one step increase in the  $\delta$ -function. To do this, the constructed  $NSTA_w$  employs  $(Q \times \Sigma_\diamond)$  for the states. We will prove the construction to be correct in Lemma 3.8.

**Definition 3.7** Given an  $NFA_w A = (\Sigma, Q, q_i, \delta, F)$  we define the  $NSTA_w$

$$NSTA(A) = ((\Sigma, Q', \delta', F'), S)$$

where  $Q' = (Q \times \Sigma_\diamond) \uplus \{q_U\}$ ,  $S = (F \times \Sigma_\diamond)$ ,  $F' = \{(q, a) | a \in \Sigma_\diamond, q \in \delta(q_I, a)\}$ , and  $\delta'$  is defined as follows: for each  $a \in \Sigma_\diamond$  put  $\delta'(q_U, a) = q_U^*$ , for each  $q \in F$  put  $\delta'((q, a), a) = r_q + q_U^*$ , and for each  $q \in Q - F$ , put  $\delta'((q, a), a) = r_q$ , where

$$r_q = q_U^* \cdot \left( \sum_{b \in \Sigma_\diamond, p \in \delta(q, b)} (p, b) \right) \cdot q_U^*.$$

**Lemma 3.8** For each  $NFA_w A$ ,  $\mathcal{Q}_{NSTA(A)} = \mathcal{Q}_A$ .

*Proof* Assume  $A = (\Sigma, Q, \delta, q_I, F)$ , and  $NSTA(A) = ((\Sigma, Q', \delta', F'), S)$  as in Definition 3.7. We first need an auxiliary result: Note that the value of the transition function  $\delta'(q, a)$  for a state  $q \in Q'$  and letter  $a \in \Sigma_\diamond$  is only defined if  $q = q_U$  or  $q = (q', a)$  for some  $q' \in Q$ . Hence, for any valid run  $\lambda$  of  $NSTA(A)$  on a tree  $t$ , and any node  $v \in Nodes^t$ , either  $\lambda(v) = q_U$ , or  $\lambda(v) = (q, lab^{t, \Sigma}(v))$  for some state  $q \in Q$ .

We now proceed to prove that  $\mathcal{Q}_{NSTA(A)} \subseteq \mathcal{Q}_A$ . Assume that  $t \in \mathcal{F}_\Sigma$  and  $v \in \mathcal{Q}_{NSTA(A)}(t)$ , and  $\lambda : Nodes^t \rightarrow Q'$  a corresponding run such that  $\lambda(v) \in S$ . Let  $v_1, \dots, v_n$  be the path starting in the root and ending in  $v$ . No node on this path can be mapped to  $q_U$ , since no state in  $S$  is reachable from  $q_U$ . By the result above, there are therefore  $q_1, \dots, q_n \in Q$  such that for  $i \in \{1, \dots, n\}$ ,  $\lambda(v_i) = (q_i, lab^{t, \Sigma}(v_i))$ . We prove by induction on  $i$ ,  $1 \leq i \leq n$ , that  $q_i \in \delta^*(q_I, ancstr^t(v_i))$ .

- For the base case  $i = 1$ , note first that  $q_I \in \delta^*(q_I, \epsilon)$ . Secondly, since  $\lambda$  is an accepting run, we must have that  $(q_1, lab^{t, \Sigma}(v_1))$  is a final state of the  $NFA_w$  in  $A$ , and by construction therefore  $q_1 \in \delta(q_I, lab^{t, \Sigma}(v_1))$ , i.e.,  $q_1 \in \delta^*(q_I, ancstr^t(v_1))$ .
- For the induction case where  $1 < i \leq n$ , assume  $q_{i-1} \in \delta^*(q_I, ancstr^t(v_{i-1}))$ . Since  $\lambda$  is an accepting run,

$$(q_i, lab^{t, \Sigma}(v_i)) \in Labels(\delta'((q_{i-1}, lab^{t, \Sigma}(v_{i-1})), lab^{t, \Sigma}(v_{i-1})))$$

By construction of  $\delta'$ , this implies that  $q_i \in \delta(q_{i-1}, lab^{t, \Sigma}(v_i))$ , so we have  $q_i \in \delta^*(q_I, ancstr^t(v_i))$ .

Since also  $\lambda(v) = (q_n, lab^{t, \Sigma}(v)) \in S$ , we get  $q_n \in F$ , hence  $A$  accepts  $ancstr^t(v)$ .

Secondly, we prove that  $\mathcal{Q}_{\text{NSTA}(A)} \supseteq \mathcal{Q}_A$ . Assume  $t \in \mathcal{T}_\Sigma$  and  $v \in \mathcal{Q}_A(t)$ , that is,  $\text{ancstr}^t(v) \in L(A)$ . Let  $v_1, \dots, v_n$  be the path starting at the root and ending in  $v$ , and let  $q_1, q_1, \dots, q_n$  be the states visited by a run that accepts  $\text{ancstr}^t(v)$  in  $A$ . Define  $\lambda : \text{Nodes}^t \rightarrow (Q \times \Sigma_\diamond) \cup \{q_U\}$  such that for  $v' \notin \{v_1, \dots, v_n\}$ ,  $\lambda(v') = q_U$ , while for  $1 \leq i \leq n$ , let  $\lambda(v_i) = (q_i, \text{lab}^{t,\Sigma}(v_i))$ . Since  $q_n \in F$ , we have that  $(q_n, \text{lab}^{t,\Sigma}(v)) \in S$ , and therefore we must only show that  $\lambda$  is an accepting run. But this is immediate from the construction of  $\delta'$ :  $q_U$  is a universal state, and for  $i, 1 \leq i < n$ , and  $k_1, k_2 \in \mathbb{N}$ , the word  $(q_U)^{k_1} \cdot (q_{i+1}, \text{lab}^{t,\Sigma}(v_{i+1})) \cdot (q_U)^{k_2}$  is in  $L(\delta'((q_i, \text{lab}^{t,\Sigma}(v_i)), \text{lab}^{t,\Sigma}(v_i)))$ , while for any  $k_3 \in \mathbb{N}$ , the word  $(q_U)^{k_3}$  is in  $L(\delta'(q_n, \text{lab}^{t,\Sigma}(v_n)), \text{lab}^{t,\Sigma}(v_n))$ . Finally,  $(q_1, \text{lab}^{t,\Sigma}(v_1))$  is in  $F'$  by definition of  $\text{NSTA}(A)$ , since  $q_1 \in \delta(q_1, \text{lab}^{t,\Sigma}(v_1))$ .  $\square$

We are now ready for the main result of this section:

**Theorem 3.9** *Deciding whether for an  $\text{NSTA}_w M$ ,  $\mathcal{Q}_M$  is  $\text{NFA}_w$ -definable, is complete for EXPTIME.*

*Proof* The lower bound follows from a reduction from the universality problem for  $\text{NTA}_w$ s (cf. Theorem 2.2). The reduction takes as input an  $\text{NTA}_w N = (\Sigma, Q, \delta, F)$  and constructs an  $\text{NSTA}_w M = (N', S)$  as follows. Let  $q_{\text{sel}} \notin Q$  and let  $a$  be some symbol in  $\Sigma$ . Then let  $N' = (\Sigma, Q \cup \{q_{\text{sel}}\}, \delta \cup \{(q_{\text{sel}}, a) \mapsto (\sum_{p \in F} p)^*\}, \{q_{\text{sel}}\})$  and  $S = \{q_{\text{sel}}\}$ . We show that  $\mathcal{Q}_M$  is  $\text{NFA}_w$ -definable iff  $L(N) = \mathcal{T}_\Delta$ . The query  $\mathcal{Q}_M$  selects the root node in all trees  $t = a(t_1, \dots, t_n)$  where for all  $1 \leq i \leq n$ ,  $t_i \in L(N)$ . If  $L(N) = \mathcal{T}_\Delta$ , then  $\mathcal{Q}_M$  is obviously  $\text{NFA}_w$ -definable, namely by any  $\text{NFA}_w$  selecting exactly the first letter in words in  $a\Delta^*$ . On the other hand, suppose  $\mathcal{Q}_M$  is  $\text{NFA}_w$ -definable. By Lemma 3.6, the query is ancestor-based and hence for every tree  $t'$  with a root labelled by  $a$ , the root of  $t'$  is in  $\mathcal{Q}_M(t')$ . Hence  $L(N)$  must be exactly  $\mathcal{T}_\Delta$ .

It remains to show the upper bound. By Lemmas 3.6 and 3.8, it suffices to test  $\mathcal{Q}_M = \mathcal{Q}_{\text{NSTA}(\text{NFA}(M))}$ . The construction of  $\text{NSTA}(\text{NFA}(M))$  can be done in polynomial time and with polynomial increase in size. By Theorem 2.8 the test can be done in exponential time.  $\square$

### 3.2 Single-Type EDTDs

In [23], it was shown that deciding whether an NTA is equivalent to a single-type extended DTD is complete for EXPTIME. As single-type EDTDs have ancestor-based types, which are superficially similar to ancestor-based queries as defined here, one might wonder what the relationship with the main result of the present section is. Of course, single-type EDTDs do not define queries or process trees which can have labels from an infinite set, but can easily be adapted to do so. Indeed, we can equip them with a wildcard type as our automata and just designate a set of types as output types. Then, the single-type EDTD selects those nodes which are assigned a selecting type. We now informally argue that NFAs are a subset of single-type EDTDs w.r.t. the



classes of unary queries they define. Indeed, a given NFA can be converted into an equivalent DFA, which can then be directly used to specify an equivalent single-type EDTD through its characterization as a DFA-based DTD [17, 23].

On the other hand, consider the query which selects the root when it has at least two children. The latter is definable by a single-type EDTD but is not NFA-definable as the query is not ancestor-based. To summarize, queries defined by single-type EDTDs can take the branching structure of the tree into account as the formalism is grammar-based, but at the same time type-assignment, and therefore selection, has to be deterministic whereas NFA-definable queries allow for nondeterministic selection but their expressiveness is restricted to single branches. In conclusion, Theorem 3.9 does not seem to imply or follow directly from the corresponding result on single-type EDTDs in [23].

### 3.3 Tractability

The EXPTIME-hardness in Theorem 3.9 is solely due to the expressiveness of the  $\text{NSTA}_w$ s. Indeed, when  $M$  as constructed in the proof is indeed equivalent to an  $\text{NFA}_w$ , that  $\text{NFA}_w$  is very simple: it just selects the root of the input tree. This means that, even for extremely simple subclasses of XPath (say, linear XPath), deciding definability of  $\text{NSTA}_w$ s within that class remains hard for EXPTIME. To obtain a tractability result we therefore need to restrict the class of regular unary queries. In this regard, Lemma 3.6 and Lemma 3.5 provide already sufficient criteria for tractability. Indeed, any subclass  $\mathcal{M}$  of the regular unary queries (or any representation  $\mathcal{M}$  of the regular unary queries) for which deciding  $\mathcal{Q}_{\text{NFA}(M)} \subseteq \mathcal{Q}_M$  is in PTIME for every  $M \in \mathcal{M}$ , renders the  $\text{NFA}_w$ -definability problem tractable. The latter is for instance the case for the single-type EDTDs as discussed in the previous section.

## 4 Twig-Definability of $\text{NSTA}_w$ s

In this section, we address twig-definability of  $\text{NSTA}_w$ s. We start by introducing the necessary definitions for twigs including the concept of characteristic tree in Section 4.1. In Section 4.2, we consider succinctness. In particular, we show that twigs can be exponentially more succinct than  $\text{NSTA}_w$ s and vice versa. This means that we cannot simply guess a small (i.e., polynomially bounded) twig equivalent to a given  $\text{NSTA}_w$ . Fortunately, the exponentially large twigs contain redundancy which can be represented succinctly by folding them into directed acyclic graphs (DAGs). We show in Section 4.3, that when an  $\text{NSTA}_w$  is equivalent to a twig its DAG-representation is at most of linear size. We further show in Section 4.4 that equivalence of  $\text{NSTA}_w$ s and folded twigs can be tested in exponential time through a reduction to emptiness of alternating tree-walking automata. In Section 4.5, we then obtain our main result, stating that testing twig-definability of  $\text{NSTA}_w$ s is EXPTIME-complete.

### 4.1 Basics

We start by defining twigs:

**Definition 4.1** (Twig Pattern) A *twig pattern*, or simply *twig* is a tuple  $T = (\Sigma, t, o, \text{Anc})$ , where  $\Sigma$  is a finite subset of  $\Delta$ ,  $t$  is a labelled tree over  $\Sigma$ ,  $\text{Anc} \subseteq \text{Edges}^t$  is the set of *ancestor edges*, and  $o \in \text{Nodes}^t$  is a designated output node.

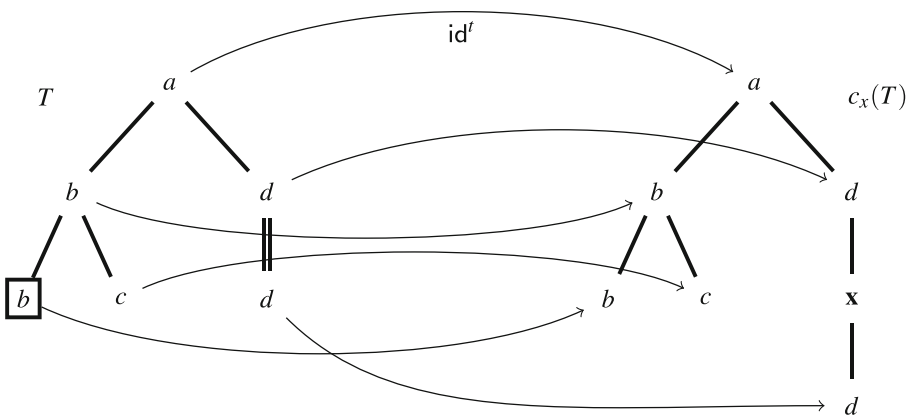
We say that  $T$  is a twig over the alphabet  $\Sigma$ . The semantics of twigs, however, is always defined over  $\Delta$ -trees. Formally, an *embedding* of  $T$  on a tree  $s$  is a total mapping  $m$  from  $\text{Nodes}^t$  to  $\text{Nodes}^s$  such that

- the root of  $t$  is mapped to the root of  $s$ ,
- $\text{lab}^t(v) = \text{lab}^s(m(v))$ , for all  $v \in \text{Nodes}^t$ , and
- for every two nodes  $v_1, v_2 \in \text{Nodes}^t$ 
  - if  $(v_1, v_2) \in \text{Edges}^t - \text{Anc}$ , then  $(m(v_1), m(v_2)) \in \text{Edges}^s$ ;
  - if  $(v_1, v_2) \in \text{Anc}$ , then  $m(v_1)$  is an ancestor of  $m(v_2)$ .

The *language* defined by  $T$  is denoted  $L(T)$  and consists of all  $\Delta$ -trees  $s$  for which there is an embedding of  $T$  into  $s$ . The *query* defined by  $T$ , denoted by  $\mathcal{Q}_T$ , is the function that maps a tree  $s$  to the set of nodes  $v \in \text{Nodes}^s$  for which there is an embedding  $m$  of  $T$  into  $s$  for which  $m(o) = v$ . In Fig. 1 we give an example of a twig and an embedding.

**Definition 4.2** (Subtwig) For a twig  $T = (\Sigma, t, o, \text{Anc})$  and a node  $v \in \text{Nodes}^t$ , let  $T[v]$  be the *subtwig* rooted at  $v$ , that is,

$$T[v] = (\Sigma, \text{subtree}^t(v), o, \text{Anc} \cap \{\text{Edges}^{\text{subtree}^t(v)}\})$$



**Fig. 1** A twig  $T = (\Sigma, t, o, \text{Anc})$  on the left, its characteristic tree  $c_x(T)$  on the right and the canonical embedding  $\text{id}^t$ . Edges of the twig in  $\text{Anc}$  are depicted as double edges and  $o$  is depicted with a rectangle

We will use the following basic property of subtrees.

**Lemma 4.3** For a twig  $T = (\Sigma, t, o, \text{Anc})$ , a tree  $s \in L(T)$ , and an embedding  $m$  of  $T$  into  $s$ ,  $\text{subtree}^s(m(v)) \in L(T[v])$  for every  $v \in \text{Nodes}^t$ .

*Proof* The embedding  $m$  is easily modified (by restricting the domain) to an embedding of  $T[v]$  into  $\text{subtree}^s(m(v))$ .  $\square$

**Definition 4.4** Two twigs  $T$  and  $T'$  are *language-equivalent* if  $L(T) = L(T')$ . They are *query-equivalent* if  $\mathcal{Q}_T = \mathcal{Q}_{T'}$ .

**Definition 4.5** (Twig-Definable Query) A unary query  $\mathcal{Q}$  is called *twig-definable* if there is a twig  $T$  such that  $\mathcal{Q} = \mathcal{Q}_T$ . A tree language  $L$  is called *twig-definable* if there is a twig  $T$  such that  $L = L(T)$ .

Next, we define a tree  $c_x(T)$  which is characteristic for a twig  $T$ . Basically, the tree is obtained by replacing each ancestor-edge with a sequence of two child edges where the new node is labeled with  $x \notin \Sigma$ . This tree is a member of the language defined by the twig. In addition, for a twig  $T'$ , when  $c_x(T) \in L(T')$  then  $L(T) \subseteq L(T')$ . The notion of characteristic trees is similar to the notion of canonical models defined by Miklau and Suciu [25] where every ancestor edge is replaced by a sequence of wildcards. Our notion is simpler because our twigs are less complex than the tree patterns used by Miklau and Suciu. Concretely, the tree patterns of Miklau and Suciu had nodes labeled by wildcards, which our twigs have not.

**Definition 4.6** (The Characteristic Tree) For a twig  $T = (\Sigma, t, o, \text{Anc})$  and  $x$  a label not in  $\Sigma$ , the *characteristic tree*  $c_x(T)$  of  $T$  is obtained from  $t$  by replacing all edges  $e = (v_1, v_2)$  in  $\text{Anc}$  with a path  $v_1, v_e, v_2$  of length 2, with labels  $\text{lab}^t(v_1)$ ,  $x$  and  $\text{lab}^t(v_2)$ , respectively. Here, for every edge  $e$ ,  $v_e$  is a new node occurring nowhere else in  $c_x(T)$ .

So, if  $T = (\Sigma, t, o, \text{Anc})$ , every node  $v \in \text{Nodes}^t$ , corresponds to a unique node of the tree  $c_x(T)$ . And every node in  $c_x(T)$  not labelled  $x$  corresponds to a unique node in  $t$ . Hence, the identity over  $\text{Nodes}^t$  is a bijection between  $\text{Nodes}^t$  and the nodes in  $c_x(T)$  not labelled  $x$ . We denote by  $\text{id}^t$  this identity function on  $\text{Nodes}^t$ . Lemma 4.7 shows that  $\text{id}^t$  is an embedding and we will refer to it as the *canonical embedding* from  $T$  into its characteristic tree. Figure 1 illustrates  $c_x(T)$  and  $\text{id}^t$ .

**Lemma 4.7** For any twig  $T = (\Sigma, t, o, \text{Anc})$  and  $x \notin \Sigma$ , the identity  $\text{id}^t$  is:

- an embedding of  $T$  into  $c_x(T)$
- a bijective function between  $\text{Nodes}^t$  and the set of all nodes in  $c_x(T)$  not labelled  $x$ .

*Proof* We first show that  $\text{id}^t$  is an embedding of  $T$  in  $c_x(T)$ . This is shown by an induction on the height of  $t$ . The base case where  $t$  has one node  $v$  is immediate, since then also  $c_x(T)$  has one node, with the same label  $\text{lab}^t(v)$ . For the induction case, assume that the statement holds for all trees of height no more than  $K$ , for some  $K \in \mathbb{N}$ . Assume then that  $t$  is of height  $K + 1$ . The root  $r$  of  $t$  is mapped to the root  $r$  of  $c_x(T)$ . For each child  $v$  of the root, where  $(r, v) \notin \text{Anc}$ ,  $(r, v)$  is also an edge in  $c_x(T)$ , and is such that  $\text{subtree}^{c_x(T)}(v) = c_x(T[v])$ . If  $(r, v) \in \text{Anc}$ ,  $r$  has in  $c_x(T)$  a child labelled  $x$  which has  $v$  as a single child, again such that  $\text{subtree}^{c_x(T)}(v) = c_x(T[v])$ . By the induction hypothesis the statement holds for the subtrees  $c_x(T[v])$  for each child  $v$  of  $r$  in  $t$  and therefore  $\text{id}^t$  is an embedding of  $T$  on  $c_x(T)$ .

The identity is of course a bijection. We only need to assert that the nodes labelled  $x$  in  $c_x(T)$  are exactly those not in  $t$ . The latter follows from the construction in Definition 4.6. Since the identity is its own inverse we will not introduce any new notation for the inverse of  $\text{id}^t$ . □

We will often use the following observation in our proofs.

*Remark 4.8* For every  $v \in \text{Nodes}^t$ ,  $\text{subtree}^{c_x(T)}(v) = c_x(T[v])$ .

The next lemma states that all embeddings and the inverses of all canonical embeddings preserve the ancestor relation. Its proof is immediate from the definitions.

**Lemma 4.9** For a twig  $T = (\Sigma, t, o, \text{Anc})$  and two nodes  $u, v \in \text{Nodes}^t$ :

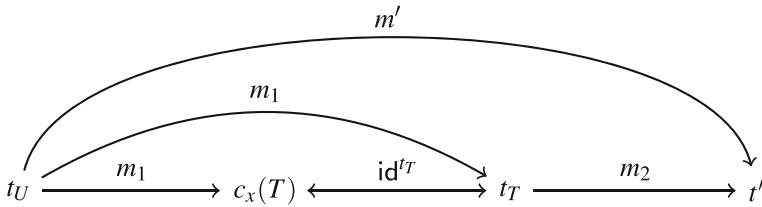
- if  $u$  is an ancestor of  $v$  in  $t$  then for any tree  $s \in L(T)$ , and embedding  $m$  of  $T$  on  $s$ ,  $m(u)$  is an ancestor of  $m(v)$  in  $s$ .
- if  $u$  is an ancestor of  $v$  in  $c_x(T)$ , then  $u$  is also an ancestor of  $v$  in  $t$ .

The proof of the following lemma is similar to the proof of Proposition 3 of [25].

**Lemma 4.10** For any two twigs  $T, U$  over an alphabet  $\Sigma$  with  $x \notin \Sigma$ ,  $c_x(T) \in L(U)$  implies  $L(T) \subseteq L(U)$ .

*Proof* Let  $T = (\Sigma, t_T, o_T, \text{Anc}_T)$  and  $U = (\Sigma, t_U, o_U, \text{Anc}_U)$ , and  $m_1$  the embedding of  $U$  on  $c_x(T)$ . To prove that  $L(T) \subseteq L(U)$ , we assume a tree  $t' \in L(T)$ , and prove  $t' \in L(U)$ . Let  $m_2$  be an embedding of  $T$  on  $t'$ . Since the image of  $m_1$  consists of nodes in  $c_x(T)$  not labelled  $x$ , which by Lemma 4.7 is exactly the nodes of  $t_T$ , we can compose  $m_1$  with  $m_2$ . We will show that this composed mapping,  $m' = m_2 \circ m_1$ , is an embedding of  $U$  on  $t'$ . The composition of  $m'$  is illustrated in Fig. 2.

It remains to show that the conditions for an embedding given in Definition 4.1 hold for  $m'$ . Recall first that for  $v \in \text{Nodes}^{t_U}$ ,  $m_1(v)$  is not labelled  $x$ , hence  $m_1(v)$  is in  $\text{Nodes}^{t_T}$  and has the same label as  $v$ . The two first properties of Definition 4.1,



**Fig. 2** The relation between the mappings and trees used in the proof of Lemma 4.10

namely mapping of the root and preservation of labels, therefore carry over easily from the same properties of  $m_1$  and  $m_2$ . For the last property, let  $(v_1, v_2) \in \text{Edges}^{t_U}$ . If  $(v_1, v_2) \notin \text{Anc}_U$ , then, since  $m_1$  is an embedding onto  $c_x(T)$ ,  $(m_1(v_1), m_1(v_2)) \in \text{Edges}^{c_x(T)}$ . From the construction of a characteristic tree, Def. 4.6, we therefore get  $(m_1(v_1), m_1(v_2)) \in \text{Edges}^{t_T} - \text{Anc}_{t_T}$ . Finally, from  $m_2$  being an embedding of  $T$  on  $t'$  we therefore get  $(m'(v_1), m'(v_2)) \in \text{Edges}^{t'}$ . On the other hand, if  $(v_1, v_2) \in \text{Anc}_U$ , then, since  $m_1$  is an embedding onto  $c_x(T)$ ,  $m_1(v_1)$  is an ancestor of  $m_1(v_2)$  in  $c_x(T)$ . By Lemma 4.9  $m_1(v_1)$  is then also an ancestor of  $m_1(v_2)$  in  $t_T$ . Further, from  $m_2$  being an embedding and applying Lemma 4.9 once more,  $m'(v_1)$  is an ancestor of  $m'(v_2)$  in  $t'$ . Hence,  $m'$  is an embedding of  $U$  onto  $t'$  and  $t' \in L(U)$ .  $\square$

We conclude this section with showing that the characteristic tree of a (proper) sub-twig is not in the language of the whole twig.

**Lemma 4.11** For a twig  $T = (\Sigma, t, o, \text{Anc})$  and a non-root node  $v \in \text{Nodes}^t$ ,

$$c_x(T[v]) \notin L(T)$$

*Proof* Let  $v_1, \dots, v_n$  be the  $n$  nodes on the longest path in  $t$ . Since  $v_1$  is the root of  $t$ , and hence  $v_1 \notin \text{Nodes}^{\text{subtree}^t(v)}$ , all paths in  $\text{subtree}^t(v)$  have less than  $n$  nodes. Hence all paths in  $c_x(T[v])$  have less than  $n$  nodes with label different from  $x$ . Any embedding of  $T$  on  $c_x(T[v])$  must map the nodes  $v_1, \dots, v_n$  to distinct nodes, each an ancestor of the next, on the same path, none labelled  $x$ . As no such path exists in  $c_x(T[v])$ , there cannot be any embedding of  $T$  on  $c_x(T[v])$ .  $\square$

A useful corollary is that the language of the subtwig cannot be completely included in the language of the whole twig:

**Corollary 4.12** For a non-root node  $v$  of a twig  $T$ ,  $L(T[v]) \not\subseteq L(T)$ .

### 4.2 Succinctness and Minimality

Next, we discuss succinctness and minimality of twigs. The *size* of a twig  $T$ , denoted by  $|T|$ , is defined as the number of nodes in its underlying tree. We distinguish two kinds of minimality.

**Definition 4.13** (Minimal twig) A twig is *language-minimal* (resp., *query-minimal*) if there is no language-equivalent (resp., query-equivalent) twig of strictly smaller size.

The following lemma summarizes basic facts on minimality used in this paper.

- Lemma 4.14** 1. *If a twig is language-minimal, then it is also query-minimal.*  
 2. *There are query-minimal twigs which are not language-minimal.*  
 3. *If a twig  $T = (\Sigma, t, o, \text{Anc})$  is query-minimal, then for all  $v \in \text{Nodes}^t$  where  $o \notin \text{Nodes}^{\text{subtree}^t(v)}$ , the twig  $T[v]$  is language-minimal.*  
 4. *If  $T = (\Sigma, t, o, \text{Anc})$  is a language-minimal twig, then for all nodes  $v \in \text{Nodes}^t$ ,  $T[v]$  is also language-minimal.*

*Proof* 1. Assume a twig  $T$  is language-minimal. For any twig  $T'$  such that  $\mathcal{Q}_T = \mathcal{Q}_{T'}$  it must be the case that  $L(T) = L(T')$ , so  $T'$  cannot be smaller than  $T$ .

2. Let  $T = (\Sigma, t, o, \text{Anc})$ , where  $t$  consists of three nodes: one root node having two child nodes. Furthermore, all labels are  $\ell$ . Let  $o$  be one of the leaves, and let the edge from the root to  $o$  be in  $\text{Anc}$ . Now,  $L(T)$  is all trees where the root and at least one child of the root is labelled  $\ell$ . Further,  $\mathcal{Q}_T$  selects any non-root node labelled  $\ell$  in a tree in  $L(T)$ .  $T$  is not language-minimal, since the node  $o$  and the edge in  $\text{Anc}$  could be removed from the tree without affecting the recognized language. But  $T$  is query-minimal. If it was not, there should be a twig  $T' = (\Sigma, t', o', \text{Anc}')$  with two nodes, one the child of the other, and with the leaf being the node  $o'$ . In both cases where the edge does or does not belong in  $\text{Anc}'$ , the twig is not query equivalent to  $T$ .
3. By contradiction: If  $T[v]$  is not language-minimal, then  $T$  is not query-minimal, since the smaller twig that is language-equivalent to  $T[v]$ , can replace  $T[v]$  also in  $T$ .
4. By an immediate contradiction: if  $T[v]$  is not language-minimal, then the smaller twig that is language-equivalent to  $T[v]$ , can be used at position  $v$  in  $T$  to give a smaller twig recognizing  $L(T)$ .

In particular, let  $v \in \text{Nodes}^t$  and suppose  $T' = (\Sigma, t', o', \text{Anc}')$  is a language-minimal twig and is such that  $L(T' = L(T[v]))$ . Suppose for contradiction that  $T'$  has less nodes than  $T[v]$ . We want to show that for any tree  $t_1 \in L(T)$ , there is an embedding  $m'$  from  $T[v \leftarrow T'] = (\Sigma, t'', o'', \text{Anc}'')$  to  $t_1$ . Let  $m_1$  be the embedding from  $T$  to  $t_1$  and let  $m_2$  be the embedding from  $T'$  to  $\text{subtree}^{t_1}(m_1(v))$ . Notice that  $\text{context}^t(v) = \text{context}^{t''}(v)$  and  $\text{subtree}^{t''}(v) = t'$ . Define  $m'$  as

$$m'(v') = \begin{cases} m_1(v') & \text{if } v' \in \text{Nodes}^{\text{context}^{t''}(v)} \\ m_2(v') & \text{if } v' \in \text{Nodes}^{\text{subtree}^{t''}(v)} \end{cases} .$$

Then,  $m'$  preserves the labelling of the nodes and the child and ancestor edges, both over the disjoint domains  $\text{Edges}^{\text{context}^{t''}(v)}$  and  $\text{Edges}^{\text{subtree}^{t''}(v)}$ , and the edge not in  $\text{Edges}^{\text{context}^{t''}(v)} \cup \text{Edges}^{\text{subtree}^{t''}(v)}$ . Therefore  $m'$  is an embedding from  $T[v \leftarrow T']$  to  $t_1$ , which is a contradiction to  $T$  being language-minimal. □

Twig-minimality plays an important role in the technical machinery developed in the next section. The following lemma specifies two sufficient criteria for a twig not to be language minimal.

**Lemma 4.15** *For a twig  $T = (\Sigma, t, o, Anc)$ ,  $x$  a label not in  $\Sigma$ , and two edges  $(v, v')$  and  $(v, v'')$  in  $t$  with  $v' \neq v''$ , then  $T$  is not language-minimal when either of the following conditions holds:*

- $(v, v'') \notin Anc$  and  $c_x(T[v'']) \in L(T[v'])$ ; or,
- $(v, v') \in Anc$  and  $\exists u \in \text{Nodes}^{\text{subtree}^t(v'')} : c_x(T[u]) \in L(T[v'])$ .

Moreover,  $\text{subtree}^t(v')$  can be removed from the twig without affecting the recognized language.

*Proof* We assume one of the two statements holds, and show that  $\text{subtree}^t(v')$  can be removed from  $t$  without affecting the language recognized by the twig. This leads to a twig  $T' = (\Sigma, t', o', Anc')$  with at least one less node. Note first that  $L(T) \subseteq L(T')$ , since any embedding of  $T$  is easily modified into an embedding of  $T'$ , by restricting the domain of the embedding. It remains to show that  $L(T') \subseteq L(T)$ . Let  $t_1 \in L(T')$  and  $m_1$  an embedding of  $T'$  on  $t_1$ . We show  $t_1 \in L(T)$  by a case distinction:

In the first case we have  $(v, v'') \notin Anc$  and  $c_x(T[v'']) \in L(T[v'])$ . Since the subtrees at  $v'$  and  $v''$  have no common nodes we have that  $T'[v''] = T[v'']$ , and therefore  $c_x(T'[v'']) = c_x(T[v''])$ . Since  $c_x(T[v'']) \in L(T[v'])$ , we have that  $c_x(T'[v'']) \in L(T[v'])$ . By applying Lemma 4.10 we therefore get  $L(T'[v'']) \subseteq L(T[v'])$ . By Lemma 4.3,

$$\text{subtree}^{t_1}(m_1(v'')) \in L(T'[v'']) \text{ and } \text{subtree}^{t_1}(m_1(v')) \in L(T[v'])$$

Let  $m_2$  be an embedding of  $T[v']$  on  $\text{subtree}^{t_1}(m_1(v'))$ . Notice that  $m_2(v') = m_1(v'')$ , as the root of  $T[v']$  must be mapped to the root of  $\text{subtree}^{t_1}(m_1(v''))$ , which is  $m_1(v'')$ .

Define the mapping  $m' : \text{Nodes}^t \rightarrow \text{Nodes}^{t_1}$  as follows. For all  $u \in \text{Nodes}^t$ ,

$$m'(u) = \begin{cases} m_1(u) & \text{if } u \in \text{Nodes}^{\text{context}^t(v')} \\ m_2(u) & \text{if } u \in \text{Nodes}^{\text{subtree}^t(v')} \end{cases}$$

We want to show that  $m'$  is an embedding of  $T$  on  $t_1$ . By definition  $m'$  preserves the labelling of the nodes, and over the disjoint domains  $\text{Edges}^{\text{context}^t(v')}$  and  $\text{Edges}^{\text{subtree}^t(v')}$ , the mapping  $m'$  preserves child and ancestor edges. For the edge  $(v, v') \notin \text{Edges}^{\text{context}^t(v')} \cup \text{Edges}^{\text{subtree}^t(v')}$ , it holds that

$$(m'(v), m'(v')) = (m_1(v), m_2(v')) = (m_1(v), m_1(v''))$$

is an edge in  $t_1$ , since  $(v, v'') \notin Anc$ . Therefore,  $m'$  is an embedding of  $T$  on  $t_1$ , irrespective of whether  $(v, v') \in Anc$  or  $(v, v') \notin Anc$  in  $t$ .

In the other case,  $(v, v') \in Anc$  and  $u \in \text{Nodes}^{\text{subtree}^t(v')}$ , such that  $c_x(T[u]) \in L(T[v'])$ . Since the subtrees at  $u$  and  $v'$  are non-overlapping, we have  $c_x(T[u]) = c_x(T'[u])$ , hence  $c_x(T'[u]) \in L(T[v'])$ . By Lemma 4.10,  $L(T'[u]) \subseteq L(T[v'])$ ,

and by Lemma 4.3  $\text{subtree}^{t_1}(m_1(u)) \in L(T'[u])$ . Therefore,  $\text{subtree}^{t_1}(m_1(u)) \in L(T[v'])$ . Let  $m_2$  be an embedding of  $T[v']$  on  $\text{subtree}^{t_1}(m_1(u))$ . Notice that  $m_2(v') = m_1(u)$ , since the root of  $T[v']$  must be mapped by  $m_2$  to the root of  $\text{subtree}^{t_1}(m_1(u))$ .

Define the mapping  $m' : \text{Nodes}^t \rightarrow \text{Nodes}^{t_1}$  as follows. For all  $z \in \text{Nodes}^t$ ,

$$m'(z) = \begin{cases} m_1(z) & \text{if } z \in \text{Nodes}^{\text{context}^t(v')} \\ m_2(z) & \text{if } z \in \text{Nodes}^{\text{subtree}^t(v')} \end{cases}$$

We want to show that  $m'$  is an embedding of  $T$  on  $t_1$ . Firstly, notice that by definition,  $m'$  preserves the labelling of the nodes. Furthermore, over the disjoint domains  $\text{Edges}^{\text{context}^t(v')}$  and  $\text{Edges}^{\text{subtree}^t(v')}$ ,  $m'$  preserves child and ancestor edges. For the edge  $(v, v') \notin \text{Edges}^{\text{context}^t(v')} \cup \text{Edges}^{\text{subtree}^t(v')}$ , notice that  $m'(v) = m_1(v)$  and  $m'(v') = m_2(v')$  by the definition of  $m'$ , and therefore  $m'(v)$  is an ancestor of  $m'(v')$ , since  $m_2(v') = m_1(u)$ , and  $m_1(v)$  is an ancestor of  $m_1(u)$ . □

Using Lemma 4.15, we can now show that for a minimal twig, the canonical embedding is the only embedding on the characteristic tree.

**Lemma 4.16** *For a language-minimal twig  $T = (\Sigma, t, o, \text{Anc})$ , there is exactly one embedding of  $T$  into  $c_x(T)$ .*

*Proof* Let  $m$  be an embedding from  $T$  to  $c_x(T)$  which is different from the canonical embedding  $\text{id}^t$ . Then, let  $v_1$  be a node in  $t$  of minimum height (closest to the root), such that  $v_1 \neq m(v_1)$ , and let  $v$  be the parent of  $v_1$  in  $t$ . Note that no node of  $t$  can be mapped by any embedding to a node labelled with  $x$  in  $c_x(T)$ , and therefore let  $v_2 \in \text{Nodes}^t$  be such that  $v_2 = m(v_1)$ . Notice also that by Definition 4.1  $m$  must map the root of  $t$  to the root of  $c_x(T)$ , and therefore  $v_1$  is not the root.

We want to show that  $v_1$  cannot be mapped to  $m(v_1)$ . From the embedding  $m$  and Remark 4.8, we have

$$\text{subtree}^{c_x(T)}(m(v_1)) = \text{subtree}^{c_x(T)}(v_2) = c_x(T[v_2]),$$

and by our assumption that  $m$  is an embedding from  $T$  to  $c_x(T)$ , by Lemma 4.3 we have that  $\text{subtree}^{c_x(T)}(m(v_1)) \in L(T[v_1])$ , and hence  $c_x(T[v_2]) \in L(T[v_1])$ . Therefore, by Lemma 4.10,  $L(T[v_2]) \subseteq L(T[v_1])$ .

We consider two cases, one where  $v_2$  is a child of  $v$ , and one where it is simply a descendant of  $v$ . Consider the first case. By our assumption that  $T$  is language-minimal, the conditions of Lemma 4.15 cannot be satisfied, and hence, if  $(v, v_1) \in \text{Anc}$  then by the second condition of the lemma,  $c_x(T[v_2]) \notin L(T[v_1])$ . Similarly, if  $(v, v_2) \notin \text{Anc}$ , by the first condition of the lemma, we have again that  $c_x(T[v_2]) \notin L(T[v_1])$ . Therefore,  $(v, v_2) \in \text{Anc}$  and  $(v, v_1) \notin \text{Anc}$ . But,  $m(v_1) = v_2$ , and therefore, since  $(v, v_2) \in \text{Anc}$ ,  $m(v_1)$  cannot be a child of  $v$  in  $c_x(T)$ , as there is an additional node between  $v$  and  $v_2$  labelled  $x$ . This contradicts the



fact that  $(v, v_1) \notin \text{Anc}$ , and hence  $m$  violates the conditions for being an embedding.

Consider then the second case where  $v_2$  is a descendant of  $v$  in  $t$ , and suppose firstly that  $v_2$  is not a descendant of  $v_1$ . Then, let  $v'_2$  be the ancestor of  $v_2$  that is a sibling of  $v_1$  in  $t$ , but different from  $v_1$ . The twig  $T$  is language-minimal and therefore the conditions of Lemma 4.15 are not satisfied. Therefore, by the second condition of the lemma, if  $(v, v_1) \in \text{Anc}$  there exists no node  $u \in \text{Nodes}^{\text{subtree}'(v'_2)}$  such that  $c_x(T[u]) \in L(T[v_1])$ . In particular, this contradicts  $c_x(T[v_2]) \in L(T[v_1])$ , and therefore the edge  $(v, v_1)$  must not be in  $\text{Anc}$ . But then  $m$  is not a valid embedding, since  $m(v_1) = v_2$  is not a child of  $m(v) = v$  in  $c_x(T)$ .

Finally, consider the case where  $v_2$  is a descendant of  $v_1$  in addition to being a descendant of  $v$ . But by Lemma 4.11, the node  $v_2$  cannot be a descendant of  $v_1$ , since  $c_x(T[v_2]) \in L(T[v_1])$ , which is a contradiction.  $\square$

We conclude our discussion on minimality with the following lemma. By construction it always holds that  $c_x(T) \in L(T)$  for  $x \notin \Sigma$ . Assume  $T$  is minimal and let  $u$  be one of its nodes. When we replace the subtree rooted at node  $u$  in  $c_x(T)$  by a new tree  $t'$  resulting in the tree  $s = c_x(T)[u \leftarrow t']$ , then the lemma says that when  $s$  still happens to be in the language defined by  $T$  then  $T[u]$ , the twig rooted at  $u$ , can always be mapped somewhere in  $t'$ .

**Lemma 4.17** *Let  $x$  be a label not in  $\Sigma$ . For a language-minimal twig  $T = (\Sigma, t, o, \text{Anc})$  over  $\Sigma$ , a node  $u \in \text{Nodes}^t$ , and a tree  $t' \in \mathcal{T}_\Delta$ , if  $c_x(T)[u \leftarrow t'] \in L(T)$ , then there is a node  $u' \in \text{Nodes}^{t'}$  such that  $\text{subtree}^{t'}(u') \in L(T[u])$ .*

*Proof* Let  $t'' = c_x(T)[u \leftarrow t']$  and suppose that  $t'' \in L(T)$ . Then, there is an embedding  $m$  of  $T$  on  $t''$ . Note first that  $u \notin \text{Nodes}^{t''}$ , since it is replaced by  $\text{root}(t')$ . Further, by Definition 4.1, for  $v' \in \text{Nodes}^{\text{context}^{t''}(u)}$ ,  $m(v')$  is either in  $\text{Nodes}^{t'}$  or in  $\text{Nodes}^{\text{context}^{t''}(u)} - \{u\}$ . The latter case is equal to  $m(v') \in \text{Nodes}^{\text{context}^{t'}(u)} - \{u\}$ , since  $\text{context}^{t''}u = \text{context}^{c_x(T)}u$ , and  $x \notin \Sigma$ . We will show that  $m(u)$  can be the  $u'$  in the lemma. That is, we will show that  $\text{subtree}^{t'}(m(u)) \in L(T[u])$ . It suffices to prove that  $m(u) \in \text{Nodes}^{t'}$ . We show a slightly stronger statement by induction on the path from the root to  $u$  in  $t$ . For each node  $v$  on the path, either

1.  $m(v) \in \text{Nodes}^{t'}$ ; or,
2.  $m(v)$  is on the path from  $v$  to  $u$  in  $t$ .

The base case is the root  $\text{root}(t)$ . If  $u = \text{root}(t)$ , then  $m(\text{root}(t)) = \text{root}(t')$  and the first part of the induction hypothesis holds. Otherwise, by Definitions 4.1 and 4.6  $m(\text{root}(t)) = \text{root}(t)$ , and the second condition of the induction hypothesis holds.

For the induction case, assume  $v \in \text{Nodes}^t$  is the lowest node in the path in  $t$  from the root to  $u$  for which the induction hypothesis has been shown to hold. If the first statement holds, then it also holds for the children of  $v$ , and we are done. Otherwise,  $m(v)$  is on the path from  $v$  to  $u$  in  $t$ . Let  $v_1$  be the child of  $v$  on the path from  $v$  to  $u$  in  $t$ . If  $m(v_1) \in \text{Nodes}^{t'}$  or if  $v_1 = m(v_1)$  we are done, since then the first or the second statement, respectively, holds. Otherwise, that is, if  $m(v_1) \notin \text{Nodes}^{t'}$  and

$v_1 \neq m(v_1)$ , we treat separately the cases where the edge between  $v$  and  $v_1$  is or is not in Anc.

Suppose first that the edge  $(v, v_1)$  is not in Anc. This means, by definition of embeddings, that  $m(v_1)$  is a child of  $m(v)$  in  $t''$ . By the construction of  $c_x(T)$ ,  $m(v_1)$  is therefore also a child of  $m(v)$  in  $t$ . We now treat separately the two cases depending on whether  $m(v) = v$ .

If  $m(v) = v$ , then  $m(v_1) \neq v_1$  is a sibling of  $v_1$  in  $t$ , while  $v_1$ , and therefore not  $m(v_1)$ , is an ancestor of  $u$ , by definition of the canonical embedding. But then  $\text{subtree}^{t''}(m(v_1)) = \text{subtree}^{c_x(T)}(m(v_1))$ , and the latter is equal to  $c_x(T[m(v_1)])$  by Remark 4.8. By Lemma 4.3, and the embedding  $m$ ,  $\text{subtree}^{t''}(m(v_1)) \in L(T[v_1])$  and therefore  $c_x(T[m(v_1)]) \in L(T[v_1])$ . Furthermore,  $v_1$  and  $m(v_1)$  are siblings and  $(v, v_1) \notin \text{Anc}$  and  $(v, m(v_1)) \notin \text{Anc}$ . By Lemma 4.15, this contradicts our assumption that  $T$  is language-minimal.

Since  $m(v) = v$  leads to a contradiction, we have  $m(v) \neq v$ . From the induction hypothesis we then get that  $m(v)$  is a descendant of  $v$  in  $t$ . Since  $v_1$  and  $m(v)$  both are on the path from  $v$  to  $u$  we get that  $m(v_1)$  is either on the path from  $v_1$  to  $u$ , or is a sibling of a node on the path from  $v_1$  to  $u$ . In the last case, notice that  $\text{subtree}^{t''}(m(v_1)) = \text{subtree}^{c_x(T)}(m(v_1))$ , which in turn is equal to  $c_x(T[m(v_1)])$ , by Remark 4.8. By Lemma 4.3 and the embedding  $m$ ,  $\text{subtree}^{t''}(m(v_1)) \in L(T[v_1])$ . It follows that  $c_x(T[m(v_1)]) \in L(T[v_1])$ . But  $T[v_1]$  is a language-minimal twig, and  $m(v_1)$  is a descendant of  $v_1$ , and by Lemma 4.11 this is a contradiction. Therefore  $m(v_1)$  is on the path from  $v_1$  to  $u$ , and the induction hypothesis holds for  $v_1$ .

Consider now the case where  $(v, v_1) \in \text{Anc}$ . We remind the reader that  $v_1 \neq m(v_1)$ ,  $m(v_1) \notin \text{Nodes}^{t'}$  and the second condition of the inductive hypothesis holds for  $v$ . That is,  $m(v) \notin \text{Nodes}^{t'}$ . Note that  $m(v_1)$  is a descendant of  $m(v)$  also in  $t$ . Let  $v_3$  be the child of  $m(v)$  and ancestor of (or equal to)  $m(v_1)$  in  $t$ .

We prove by contradiction that the second statement of the induction hypothesis holds. That is, we assume that  $m(v_1)$  is not on the path from  $v_1$  to  $u$  in  $t$ , and show that this leads to contradiction. From the latter assumption, we get that  $\text{subtree}^{t''}(m(v_1))$  does not contain  $t'$ . Therefore, since  $\text{context}^{t''}(u) = \text{context}^{c_x(T)}(u)$ , we get

$$\text{subtree}^{t''}(m(v_1)) = \text{subtree}^{c_x(T)}(m(v_1)).$$

By Remark 4.8  $\text{subtree}^{c_x(T)}(m(v_1)) = c_x(T[m(v_1)])$ . Combining the two latter equalities we get

$$\text{subtree}^{t''}(m(v_1)) = c_x(T[m(v_1)]). \tag{4.1}$$

By applying Lemma 4.3 to  $T, m, t''$  and  $v_1$  it holds that  $\text{subtree}^{t''}(m(v_1)) \in L(T[v_1])$ . Combining the latter with (4.1) we get

$$c_x(T[m(v_1)]) \in L(T[v_1]). \tag{4.2}$$

Now,  $m(v)$  is on the path from  $v$  to  $u$  by the inductive hypothesis, and we consider the two cases where  $m(v) \neq v$  and  $m(v) = v$ . In the first case,  $m(v)$  is a descendant of  $v$ , and therefore  $v_3$  and  $m(v_1)$  are descendants of  $v_1$ . By Lemma 4.11, it holds that  $c_x(T[m(v_1)]) \notin L(T[v_1])$ , which contradicts (4.2). In the case where  $m(v) = v$ ,  $v_3$  is either equal to  $v_1$  or a sibling of  $v_1$ . If  $v_3 = v_1$ ,  $m(v_1)$  is a descendant of  $v_1$

and a similar argument using Lemma 4.11 as above, leads to a contradiction. If  $v_3$  is a sibling of  $v_1$ , we have from Lemma 4.15, the fact that  $(v, v_1) \in \text{Anc}$ , and our assumption that  $T$  is language-minimal, that for any descendant of  $v_3$ , and therefore in particular  $m(v_1)$  as well,  $c_x(T[m(v_1)]) \notin L(T[v_1])$ , which also contradicts (4.2).

We have shown that assuming that  $m(v_1)$  is not on the path from  $v_1$  to  $u$  in  $t$  leads to a contradiction. Therefore the induction hypothesis holds for  $v_1$  also in the case where  $(v, v_1) \in \text{Anc}$ . □

Next, we discuss succinctness of twigs and NSTAs.

- Theorem 4.18** 1. *There is a family of NSTAs  $M_n$  (for  $n \in \mathbb{N}$ ) of size  $\mathcal{O}(n)$  such that the smallest query-equivalent twig is of size  $\Omega(2^n)$ .*  
 2. *For every twig  $T$  of size  $n$ , there exists an equivalent NSTA of size  $\mathcal{O}(2^n)$ .*  
 3. *There is a family of twigs  $T_n$  (for  $n \in \mathbb{N}$ ) of size  $\mathcal{O}(n)$  such that the smallest query-equivalent NSTA is of size  $\Omega(2^n)$ .*

*Proof* (1) First, we define a few more notions regarding subtrees that will be referred to in what follows. If  $S$  is a subset of  $\text{Nodes}^t$ , we say that  $S$  is connected if, for every two nodes  $v_1, v_2 \in S$ , there is a node  $v$  and paths from  $v$  to  $v_1$  and to  $v_2$  using only nodes in  $S$ . Notice that  $v$  may be equal to  $v_1$  or  $v_2$ . For a tree  $t$  and a connected subset  $S$  of  $\text{Nodes}^t$ , the *subgraph  $t'$  of  $t$  induced by  $S$* , is the tree with  $\text{Nodes}^{t'} = S$  and  $\text{Edges}^{t'} = (S \times S) \cap \text{Edges}^t$ .

Fix the alphabet  $\Sigma = \{a, b\}$ . For each  $n \in \mathbb{N}$ , we define the NSTA  $M_n = ((\Sigma, Q_n, \delta_n, F_n), F_n)$ ,  $Q_n = \{q_u, q_0, q_{1,a}, q_{1,b}, \dots, q_{n,a}, q_{n,b}\}$ ,  $F_n = \{q_0\}$ , and  $\delta_n$  is defined as follows. For  $1 \leq i < n$ , and  $\sigma \in \Sigma$

$$\begin{aligned} \delta_n(q_{i,\sigma}, \sigma) &= (q_u^* \cdot q_{i+1,a} \cdot q_u^* \cdot q_{i+1,b} \cdot q_u^*) + (q_u^* \cdot q_{i+1,b} \cdot q_u^* \cdot q_{i+1,a} \cdot q_u^*), \\ \delta_n(q_{n,\sigma}, \sigma) &= q_u^*, \\ \delta_n(q_0, a) &= (q_u^* \cdot q_{1,a} \cdot q_u^* \cdot q_{1,b} \cdot q_u^*) + (q_u^* \cdot q_{1,b} \cdot q_u^* \cdot q_{1,a} \cdot q_u^*), \\ \delta_n(q_u, \sigma) &= q_u^*, \\ \delta_n(q_u, \diamond) &= q_u^*. \end{aligned}$$

For each  $n \in \mathbb{N}$ , let  $S'_n$  be the set of complete binary trees of height  $n$ , where the root is labelled with  $a$  and each non-leaf node has exactly two children, one labelled with  $a$  and one with  $b$ . Then  $\mathcal{Q}_{M_n}$  contains exactly the pairs  $(s, \text{root}(s))$  where  $s$  has an induced subgraph  $s'$  such that  $s' \in S'_n$  and  $\text{root}(s) = \text{root}(s')$ .

Assume, for the purpose of proving a contradiction, that the twig  $T'_n$  has less than  $2^n - 1$  nodes and is such that  $\mathcal{Q}_{T'_n} = \mathcal{Q}_{M_n}$ . Let  $s' \in S'_n$  and  $m$  be the embedding of  $T'_n$  on  $s'$ . By the pigeon-hole principle, there must be a node  $v \in \text{Nodes}^{s'}$  which is not in the image of  $m$ . Let  $s_{err}$  be the same as  $s'$  except that we change the label of  $v$ .  $m$  is now obviously also an embedding of  $T'_n$  on  $s_{err}$ , but  $s_{err} \notin \mathcal{Q}_{M_n}$ . This contradicts our assumption that  $\mathcal{Q}_{T'_n} = \mathcal{Q}_{M_n}$ .

- (2) We prove by induction on the size  $n$  of the twig  $T = (\Sigma, t, o, \text{Anc})$ , that there is an equivalent NSTA  $M$  of size  $\mathcal{O}(2^n)$  and an NTA  $N$  of size  $\mathcal{O}(2^n)$  such that  $L(N) = L(T)$ .

As the base case, suppose that  $T = (\Sigma, t, o, \text{Anc})$  is such that  $t$  has only one node. Then the NSTA  $M$  requires two states, one of them final, to check that the root of the tree is labelled with the same label as the root of  $t$ . The set of selecting states  $S$  contains just the final state. The NTA  $N$  is defined exactly the same, but does not have the set of selecting states.

For the induction case, suppose that for every  $m < n$ , the statement holds for every twig of size  $m$ . Let  $T = (\Sigma, t, o, \text{Anc})$  be of size  $n$ , and in particular, let  $t = a(t_1, \dots, t_k)$ , where the size of  $t_i$  is  $m_i$ , for all  $1 \leq i \leq k$ . Hence,  $n = 1 + \sum_{i=1}^k m_i$ . We make a first case distinction, depending on whether  $o = \text{root}(t)$  or not.

If  $o = \text{root}(t)$  then, by the induction hypothesis, for  $1 \leq i \leq k$ , there exist NTAs  $N_i$ , of size  $\mathcal{O}(2^{m_i})$ , such that  $L(N_i) = L(T_i)$ , where  $T_i = T[v_i]$  and  $v_i$  is the root of  $t_i$  in  $t$ . Denote  $T_i = (\Sigma, t_i, o, \text{Anc}_i)$ . Then consider the twigs  $T'_i = (\Sigma, a(t_i), o, \text{Anc}'_i)$ , where  $\text{Anc}'_i$  contains all edges from  $\text{Anc}_i$ . Furthermore, if the edge between the root of  $a(t_1, \dots, t_n)$  and  $t_i$  is in  $\text{Anc}$ , we also add the edge between the root of  $a(t_i)$  and the root of  $t_i$  in  $\text{Anc}'_i$ . We now have that, for each  $i = 1, \dots, k$ , there exists an NTA  $N'_i$  which has at most two more states than  $N_i$ : one state for testing whether the root is labelled  $a$  and one state for simulating the ancestor edge between  $\text{root}(a(t_i))$  and  $\text{root}(t_i)$  in  $\text{Anc}'_i$ , if present. Altogether, the size of  $N'_i$  is  $\mathcal{O}(2^{m_i} + 2) = \mathcal{O}(2^{m_i})$  such that  $L(N'_i) = L(T'_i)$ .

Observe that the product  $N = N_1 \times \dots \times N_k$  for the intersection of these NTAs accepts precisely the language  $L(T)$ . Furthermore, the size of  $N$  is  $\mathcal{O}(2^{m_1} \dots 2^{m_k}) = \mathcal{O}(2^{m_1 + \dots + m_k}) = \mathcal{O}(2^n)$ . Finally, the NSTA  $M$  is obtained from  $N$  by defining the set of selecting states  $S$  to be the accepting states from  $N$ .

The construction for the case where  $o$  is not the root node of  $t$  is analogous. The difference is that we need to deal with the  $t_j$  that contains the node  $o$ . For this  $t_j$ , we take the NSTA  $M_j = (N_j, S)$  from the induction hypothesis and adapt it for the twig  $T'_j$ , similarly as we did for the  $N'_i$ . Finally, the NSTA  $M$  is then  $((N_1 \times \dots \times N_k), S')$ , where  $S' = \{(q_1, \dots, q_k) \mid q_j \in S\}$ .

- (3) For  $n \in \mathbb{N}$ , let  $\Sigma_n = \{a, a_1, \dots, a_n\}$  and let  $T_n = (\Sigma, t_n, \text{root}(t_n), \text{Anc})$  be the twig where  $t_n = a(a_1, \dots, a_n)$  and  $\text{Anc} = \text{Edges}^{t_n}$ . For each  $n$ ,  $T_n$  contains  $n + 1$  nodes, and the trees in  $L(T_n)$  are exactly the trees  $s$  such that  $\text{lab}^s \text{root}(s) = a$  and there are nodes  $v_1, \dots, v_n \in \text{Nodes}^s - \{\text{root}(s)\}$  such that  $\text{lab}^s v_1 = a_1, \dots, \text{lab}^s v_n = a_n$ .

For each (non-empty, strict) subset  $S$  of  $\{a_1, \dots, a_n\}$ , fix an arbitrary tree  $t_S$  such that  $t_S$  is labelled with exactly the labels from  $S$ . That is, for every  $a_i$  in  $S$ ,  $t_S$  has a node  $v_i$  with  $\text{lab}^{t_S} v_i = a_i$  and such that  $t_S$  has no nodes  $v$  with  $\text{lab}^{t_S} v \notin S$ . Furthermore, for every such subset  $S$ , denote by  $\bar{S}$  the set  $\{a_1, \dots, a_n\} - S$ . Notice that, for every  $S$ , the tree  $a(t_S, t_{\bar{S}}) \in L(T_n)$ .

Suppose then for contradiction, that there exists an NSTA  $M$  with fewer than  $2^n - 2$  states, accepting the language  $L(T_n)$ . Consider the  $2^n - 2$  strict (or proper) and non-empty subsets of  $\Sigma_n - \{a\}$ , and for each such subset  $S_i$ , consider the tree  $a(t_{S_i}, t_{\bar{S}_i})$ . The NSTA  $M$  has an accepting run on each of these trees, and each of these runs assigns some state to the node  $\text{root}(t_{S_i})$ . By the pigeon hole principle, there must

therefore be two different non-empty, strict subsets  $S_1$  and  $S_2$  of  $\Sigma_n - \{a\}$ , such that on the trees

$$a(t_{S_1}, t_{\overline{S_1}}) \text{ and } a(t_{S_2}, t_{\overline{S_2}}),$$

$M$  has accepting runs  $\lambda_1$  and  $\lambda_2$  that assign the same state  $q$  to the root of  $t_{S_1}$  and the root of  $t_{S_2}$ . We can assume w.l.o.g. that  $S_2 \not\subseteq S_1$ . (If  $S_2 \subseteq S_1$  then we can switch  $S_1$  and  $S_2$ .) Notice that  $M$  also has an accepting run  $\lambda$  on the tree  $t = a(t_{S_1}, t_{\overline{S_2}})$ . Indeed, this accepting run  $\lambda$  is the same as  $\lambda_1$  on the subtree  $t_{S_1}$ , it is the same as  $\lambda_2$  on subtree  $t_{\overline{S_2}}$  and on the root of  $t$ . However, since  $S_2 \not\subseteq S_1$ , there exists an  $a_i \in S_2 - S_1$ . As the tree  $t = a(t_{S_1}, t_{\overline{S_2}})$  does not contain the label  $a_i$ , it is not in  $L(T_n)$ . This means that  $M$  does not accept  $L(T_n)$  and is a contradiction.  $\square$

### 4.3 DAG-Twigs

Theorem 4.18 (1) excludes the possibility to simply guess an equivalent twig of small size for a given NSTA. Fortunately, as we will show in this section, when an NSTA is equivalent to a twig the latter has a small representation as a directed acyclic graph (DAG).

Below, we use DAGs to represent the trees in twigs. As usual, a DAG  $G$  is a directed graph  $G = (V, E)$ , where  $V$  is the set of vertices and  $E \subseteq V \times V$  is the set of directed edges, and is such that there is no directed cycle in the graph. Note that we do not consider multi-edges. A DAG  $G$  over the alphabet  $\Sigma$  has an associated labelling function  $\text{lab}^G : V \rightarrow \Sigma$ . We assume that all DAGs have exactly one vertex with no incoming edges (called the root and denoted by  $\text{root}(G)$ ) and that they are connected. In what follows, we also refer to the vertices of the DAG as nodes.

For any node  $v \in V$ , let  $\text{clean}^G(v)$  be the DAG obtained from  $G$  by removing every node that is not reachable from  $v$ . We next recursively define the *unfolding* of  $G$  into a tree  $\text{unfold}(G)$ . When  $|V| = 1$ ,  $\text{unfold}(G)$  is a single node with the same label as  $\text{root}(G)$ . When  $|V| > 1$ , let  $U = \{u \in V \mid (\text{root}(G), u) \in E\}$  and let  $u_1 <_U \dots <_U u_m$  be an arbitrary ordering of the nodes in  $U$ . Then, for each  $1 \leq k \leq m$ , let  $G_k = \text{clean}^{G - \{\text{root}(G)\}}(u_k)$ . The tree  $\text{unfold}(G)$  is then defined as

$$\text{lab}^G(\text{root}(G))(\text{unfold}(G_1), \dots, \text{unfold}(G_m)).$$

We denote by  $\text{fold}^G$  the canonical mapping from  $\text{Nodes}^{\text{unfold}(G)}$  to  $V$ . We say that a tree  $t$  is *represented* by a DAG  $G$ , if  $G$  can be unfolded into  $t$ .

**Definition 4.19** (Dag-Twig) A *DAG-twig* is a tuple  $D = (\Sigma, G, o, \text{Anc})$ , where  $G = (V, E)$  is a DAG over  $\Sigma$ , the node  $o \in V$  is such that there is exactly one path from the root to  $o$  in  $G$ , and  $\text{Anc} \subseteq E$ . The *query* defined by  $D$ , denoted by  $\mathcal{Q}_D$ , is the query  $\mathcal{Q}_T$  where  $T$  is the twig  $(\Sigma, \text{unfold}(G), o_G, \text{Anc}_G)$  for which

- $\text{fold}^G(o_G) = o$ ; and,
- $\text{Anc}_G = \{(v, u) \mid (\text{fold}^G(v), \text{fold}^G(u)) \in \text{Anc}\}$ .

We say that the DAG-twig  $D$  *represents* the twig  $T$ .

Notice that, as there is only one path from the root to  $o$  there can only be a unique node  $o_G$  for which  $\text{fold}^G(o_G) = o$ . Furthermore, due to the possibly many ways in which a DAG can be unfolded, there are multiple twigs that are represented by a DAG. However, since all these twigs define the same query, we feel that it is justified to refer to  $\mathcal{Q}_D$  as *the* query defined by  $D$ .

The next theorem says that if an  $\text{NSTA}_w$  is twig-definable, there exists an equivalent DAG-twig of at most linear size.

**Theorem 4.20** *For an  $\text{NSTA}_w$   $M = (N, S)$  over an alphabet  $\Sigma$ , if  $\mathcal{Q}_M$  is twig-definable, then there exists an equivalent DAG-Twig  $D$  over alphabet  $\Sigma$ , with at most  $2 \cdot |Q_N|$  nodes, where  $Q_N$  is the set of states of  $N$ .*

Before we start proving Theorem 4.20, we introduce a definition and some lemmas. Recall that in Definition 2.4 we defined the mappings  $\text{bool}_\Sigma$  and  $\text{BoolQ}_\Sigma$  and how they reduce unary queries to Boolean ones.

We now expand on this notation to define a similar twig:

**Definition 4.21** For a twig  $T = (\Sigma, t, o, \text{Anc})$ , the twig  $\text{BoolT}(T)$  is defined as  $(\Sigma \times \{0, 1\}, \text{bool}_\Sigma(t, o), o, \text{Anc})$ .

In the next lemma, we formulate the similarity between  $\text{BoolT}$  and  $\text{BoolQ}_\Sigma$  more exactly.

**Lemma 4.22** *For any twig  $T = (\Sigma, u, o, \text{Anc})$ ,*

$$L(\text{BoolT}(T)) \cap \text{Image}(\text{bool}_\Sigma) = \text{BoolQ}_\Sigma(\mathcal{Q}_T)$$

*Proof* We first show that  $\text{BoolQ}_\Sigma(\mathcal{Q}_T) \subseteq L(\text{BoolT}(T)) \cap \text{Image}(\text{bool}_\Sigma)$ . Since we have that  $\text{BoolQ}_\Sigma(\mathcal{Q}_T) \subseteq \text{Image}(\text{bool}_\Sigma)$  by Definition 2.4, we only need to show that  $\text{BoolQ}_\Sigma(\mathcal{Q}_T) \subseteq L(\text{BoolT}(T))$ . Let  $t$  be a tree in  $\text{BoolQ}_\Sigma(\mathcal{Q}_T)$ . Hence, we have  $t \in \text{Image}(\text{bool}_\Sigma)$  and, by Lemma 2.5, we can let  $(t', v) = \text{bool}_\Sigma^{-1}(t)$ . By Definition 2.4 this implies that  $v \in \mathcal{Q}_T(t')$ . From Definition 4.1 we then get  $t' \in L(T)$ , and that there is an embedding  $m$  of  $T$  on  $t'$  is such that  $m(o) = v$ . We show that  $m$  is also an embedding of  $\text{BoolT}(T)$  on  $t$ . Recall that, by Definition 2.4,  $t$  and  $t'$  have the same nodes and edges, as do  $u$  and  $\text{bool}_\Sigma(u, o)$ .  $\text{Anc}$  is also the same in  $T$  and  $\text{BoolT}(T)$ , so it only remains to show that for all  $v \in \text{Nodes}^u$ ,  $\text{lab}^t(m(v)) = \text{lab}^{\text{bool}_\Sigma(u, o)}(v)$ . Let  $v \in \text{Nodes}^u$ . Since  $m$  is an embedding of  $T$  on  $t'$ ,  $\text{lab}^{t'}(m(v)) = \text{lab}^u(v)$ . If  $\text{lab}^{\text{bool}_\Sigma(u, o)}(v) \notin \Sigma$ , then  $\text{lab}^{\text{bool}_\Sigma(u, o)}(v) = \text{lab}^u(v)$ , and  $\text{lab}^{t'}(m(v)) \notin \Sigma$ , so  $\text{lab}^t(m(v)) = \text{lab}^{t'}(m(v))$  and hence  $\text{lab}^t(m(v)) = \text{lab}^{\text{bool}_\Sigma(u, o)}(v)$  holds. For  $v = o$ , note that  $\text{lab}^t(m(o)) = (\text{lab}^{t'}(m(o)), 1)$  and  $\text{lab}^{\text{bool}_\Sigma(u, o)}(o) = (\text{lab}^u(o), 1)$ , so  $\text{lab}^t(m(o)) = \text{lab}^{\text{bool}_\Sigma(u, o)}(o)$  holds. Otherwise, if  $v \neq o$ , and  $\text{lab}^{\text{bool}_\Sigma(u, o)}(v) \in \Sigma$ ,  $\text{lab}^t(m(v)) = (\text{lab}^{t'}(m(v)), 0)$  and  $\text{lab}^{\text{bool}_\Sigma(u, o)}(v) = (\text{lab}^u(v), 0)$ , hence,  $\text{lab}^t(m(v)) = \text{lab}^{\text{bool}_\Sigma(u, o)}(v)$  also in this case.

To show that  $L(\text{BoolT}(T)) \cap \text{Image}(\text{bool}_\Sigma) \subseteq \text{BoolQ}_\Sigma(\mathcal{Q}_T)$ , assume that  $t' \in L(\text{BoolT}(T)) \cap \text{Image}(\text{bool}_\Sigma)$ , let  $m$  be an embedding of  $\text{BoolT}(T)$  on  $t'$  and let  $(t, v) = \text{bool}_\Sigma^{-1}(t')$ . By using similar arguments as in the other case above, we prove that  $m$  is also an embedding of  $T$  on  $t$ , and  $v = m(o)$ . Hence  $v \in \mathcal{Q}_T(t)$ , therefore  $\text{bool}_\Sigma(t, v) \in \text{BoolQ}_\Sigma(\mathcal{Q}_T)$ , that is,  $t' \in \text{BoolQ}_\Sigma(\mathcal{Q}_T)$ .  $\square$

We can now prove that query-minimality is transformed by  $\text{BoolT}$  into language-minimality.

**Lemma 4.23** *If a twig  $T$  is query-minimal, then  $\text{BoolT}(T)$  is language-minimal.*

*Proof* Let  $T = (\Sigma, t, o, \text{Anc})$  be a query-minimal twig, and suppose for contradiction that  $\text{BoolT}(T)$  is not language-minimal. Then there exists a twig  $T'$  that is smaller than  $\text{BoolT}(T)$  and is such that  $L(T') = L(\text{BoolT}(T))$ . We will show that there exists a twig  $T''$  over alphabet  $\Sigma$  which is of same size as  $T'$  and such that  $L(T') = L(\text{BoolT}(T''))$ . By Lemma 4.22  $\text{BoolQ}_\Sigma(\mathcal{Q}_{T''}) = \text{BoolQ}_\Sigma(\mathcal{Q}_T)$ . By applying Lemma 2.5 we then get  $\mathcal{Q}_{T''} = \mathcal{Q}_T$ . Furthermore, by Definition 2.4  $T''$  and  $T'$  then have the same size, that is, smaller than  $T$ . Hence  $T$  is not query-minimal which is a contradiction. It remains to show the existence of such a twig  $T''$ .

Let  $T' = (\Sigma, t', o', \text{Anc}')$ . We first show that  $t' \in \text{Image}(\text{bool}_\Sigma)$ . That is, we must show that there is a node  $v \in \text{Nodes}^{t'}$  such that  $\text{lab}^{t'} v \in \Sigma \times \{1\}$  and that for each  $v' \in \text{Nodes}^{t'} - \{v\}$ ,  $\text{lab}^{t'} v' \in (\Sigma \times \{0\}) \cup (\Delta - \Sigma)$ . This is shown applying  $L(T') = L(\text{BoolT}(T))$  and Definition 4.1. Note first that there are trees in  $L(T')$  with node labels only from  $(\Sigma \times \{0, 1\}) \cup (\Delta - \Sigma)$ , so the node labels in  $t'$  must also be from this set. Furthermore, there is exactly one node  $v \in \text{Nodes}^{t'}$  such that  $\text{lab}^{t'} v \in \Sigma \times \{1\}$ , since  $L(\text{BoolT}(T))$  only contains trees with exactly one such node.

We have now shown that  $t' \in \text{Image}(\text{bool}_\Sigma)$ , so we can let  $(t'', v) = \text{bool}_\Sigma^{-1}(t')$ . By Definition 4.21  $\text{BoolT}((\Sigma, t'', v, \text{Anc}'')) = (\Sigma, t', v, \text{Anc}')$ . The size of  $T'$  and  $(\Sigma, t', v, \text{Anc}')$  is the same, and by Definition 4.21,  $L(T') = L((\Sigma, t', v, \text{Anc}'))$ , so we can let  $T'' = (\Sigma, t'', v, \text{Anc}')$ .  $\square$

The remainder of this section is devoted to the proof of Theorem 4.20.

*Proof* Let  $D$  be the smallest DAG-twig representing a query-minimal twig equivalent to  $M$ . Notice first that the DAG in  $D$  can only have labels from  $\Sigma$ . Otherwise,  $D$  would require a label which  $M$  cannot test for, and hence  $M$  and  $D$  could not be equivalent. This means that we can safely replace the alphabet of  $D$  with  $\Sigma$ . Let  $T = (\Sigma, t, o, \text{Anc})$  be the unfolding of  $D$ . Towards a contradiction, assume that the size of  $D$ , that is, its number of nodes, is larger than  $2 \cdot |Q_N|$  where  $N = (\Sigma, Q_N, \delta_N, F_N)$ . We will identify two nodes in  $D$  which can be merged leading to a strictly smaller DAG-twig which unfolds to a twig of the same size as  $T$  and is equivalent to  $T$ . In other words, the merged DAG-twig will be equivalent to a query-minimal twig as required to contradict our assumption.

Let  $x \notin \Sigma$ . Recall that  $\text{fold}^D$  is the canonical mapping from the nodes of  $T$  to the nodes of  $D$  witnessing that  $T$  is represented by  $D$ . Next, we view  $M = (N, S)$  and  $T$  from the perspective of the languages they define. Specifically, let  $N_b$  be the NTA accepting  $\text{bool}_\Sigma(\mathcal{Q}_M)$  as is given by Lemma 2.7. Note that  $N_b$  has at most  $2|Q_N|$  states. Furthermore, let  $T_b = \text{BoolT}(T)$  as defined in Definition 4.21. Now, by Lemma 4.23,  $T_b$  is language-minimal. The following lemma relates  $N_b$  and  $T_b$ :

- Lemma 4.24** 1.  $L(N_b) \subseteq L(T_b)$ ; and,  
 2.  $c_x(T_b) \in L(N_b)$ .

*Proof* From Lemma 2.7, we have that  $L(N_b) = \text{BoolQ}_\Sigma(\mathcal{Q}_M)$ . By assumption, the latter is equal to  $\text{BoolQ}_\Sigma(\mathcal{Q}_T)$  which is equal to  $L(T_b) \cap \text{Image}(\text{bool}_\Sigma)$  by Lemma 4.22.

(1): From the above it follows that  $L(N_b) \subseteq L(T_b)$ .

(2): Notice that  $c_x(T_b) \in L(T_b)$  and also  $c_x(T_b) \in \text{Image}(\text{bool}_\Sigma)$ . Therefore,  $c_x(T_b) \in L(N_b)$ . □

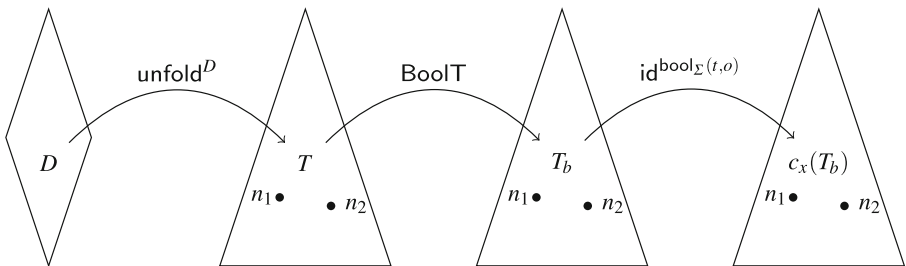
Let  $\rho$  be a run of  $N_b$  on  $c_x(T_b)$ . As  $N_b$  has at most  $2|Q_N|$  states and  $D$  has more than  $2|Q_N|$  nodes, by the pigeonhole principle, there are two nodes  $n_1, n_2$  in  $c_x(T_b)$ , not labelled by  $x$ , with  $\rho(n_1) = \rho(n_2)$  and corresponding to two different nodes in  $D$ . This means,  $\text{fold}^D(n_1) \neq \text{fold}^D(n_2)$ . Recall that the nodes in  $c_x(T_b)$  not labelled  $x$  are exactly the nodes in  $T_b$  and that  $\text{fold}^D(\cdot)$  maps nodes from  $T$  to  $D$ . Since  $T_b$  and  $T$  contain the same set of nodes the composition of these two functions is well-defined (Fig. 3). Since  $\rho(n_1) = \rho(n_2)$ , and since  $L(N_b) \subseteq L(T_b)$  by Lemma 4.24, it follows that

$$c_x(T_b) \left[ n_1 \leftarrow \text{subtree}^{c_x(T_b)}(n_2) \right] \in L(T_b), \tag{†}$$

$$c_x(T_b) \left[ n'_2 \leftarrow \text{subtree}^{c_x(T_b)}(n_1) \right] \in L(T_b). \tag{‡}$$

Using (†) and (‡), we can show the following lemma:

- Lemma 4.25** 1.  $L(T[n_1]) = L(T[n_2])$ ; and,  
 2. neither  $n_1$  nor  $n_2$  is an ancestor of or equal to the output node  $o$ .



**Fig. 3** The DAG  $D$ , the unfolding of  $D$  into  $T$ ,  $T_b = \text{BoolT}(T)$ , and the characteristic tree of  $T_b$  used in the proof of Theorem 4.20



Before we prove the lemma, let us first explain how it leads to the desired contradiction. From Lemma 4.25(1), it follows that in  $D$  the nodes  $\text{fold}^D(n_1)$  and  $\text{fold}^D(n_2)$  can be merged to give a smaller (by at least one node) DAG-twig defining the same query as defined by  $D$ . Let  $m_1 = \text{fold}^D(n_1)$  and  $m_2 = \text{fold}^D(n_2)$ . By assumption,  $m_1 \neq m_2$ . Furthermore, by Corollary 4.12 and Lemma 4.25(1), neither of these nodes can be an ancestor of the other. By merging  $m_1$  and  $m_2$ , we mean replacing  $m_1$  with  $m_2$  in all edges of the form  $(y, m_1)$  (for any  $y$ ), removing all edges of the form  $(m_1, z)$  (for any  $z$ ), removing  $m_1$  from the set of nodes, and finally removing all nodes and edges which are now not reachable from the root. Note that this merging is not well-defined when  $m_1$  and  $m_2$  are siblings, because it introduces multi-edges. However, when  $m_1$  and  $m_2$  are siblings then so are  $n_1$  and  $n_2$ . But as both  $T[n_1]$  and  $T[n_2]$  can be embedded on the same subtree of any tree in the language defined by the query, this would mean that  $T$  is not query-minimal. Therefore,  $m_1$  and  $m_2$  can not be siblings. Call the thus obtained DAG-Twig  $D'$ . Note that by Lemma 4.25(2) there is only one path from the root to the output node  $o$ . As  $D'$  is equivalent to  $D$ , it defines the same query as  $T$ , but it still needs to be argued that  $D'$  represents a query-minimal twig. That is, the unfolding of  $D'$  leads to a twig with the same number of nodes as  $T$ . From Lemma 4.25(2) and Lemma 4.14(3), it follows that both  $T[n_1]$  and  $T[n_2]$  are language minimal which means that they have the same number of nodes. So, the unfolding of  $D$  has the same number of nodes as  $T$  and is therefore query-minimal. This leads to the desired contradiction and ends the proof of Theorem 4.20.

We now prove Lemma 4.25.

*Proof* Let

$$s_1 = \text{subtree}^{c_x(T_b)}(n_2) \text{ and } s_2 = \text{subtree}^{c_x(T_b)}(n_1).$$

Then,  $s_1 = c_x(T_b[n_2])$  and  $s_2 = c_x(T_b[n_1])$ , by Remark 4.8. To show that  $L(T[n_1]) = L(T[n_2])$ , we first apply Lemma 4.17 to (†) and (‡), to obtain nodes  $v_1 \in \text{Nodes}^{s_1}$  and  $v_2 \in \text{Nodes}^{s_2}$  such that

$$\begin{aligned} \text{subtree}^{s_1}(v_1) &\in L(T_b[n_1]), \\ \text{subtree}^{s_2}(v_2) &\in L(T_b[n_2]). \end{aligned} \tag{*}$$

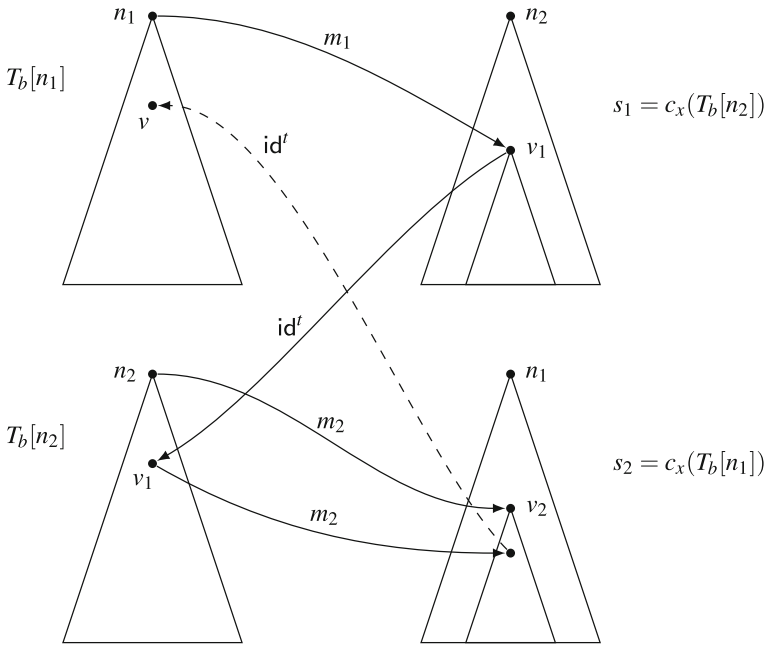
Note that,  $v_1$  and  $v_2$  are not labelled with  $x$  as the corresponding embeddings map  $n_1$  and  $n_2$  to them. We provide a graphical illustration of the employed trees and associated nodes in Fig. 4.

If  $v_1$  and  $v_2$  are the roots in the trees  $s_1$  and  $s_2$ , respectively, or in other words  $v_1 = n_2$  and  $v_2 = n_1$ , then

$$\text{subtree}^{s_2}(v_2) = \text{subtree}^{s_2}(n_1) = s_2 = c_x(T_b[n_1])$$

Similarly,  $\text{subtree}^{s_1}(v_1) = c_x(T_b[n_2])$ . Therefore, by (\*),

$$c_x(T_b[n_2]) \in L(T_b[n_1]) \text{ and } c_x(T_b[n_1]) \in L(T_b[n_2]),$$



**Fig. 4** Illustration of the trees and associated nodes used in the proof of Lemma 4.25

and by Lemma 4.25,  $L(T_b[n_1]) = L(T_b[n_2])$  which implies  $L(T[n_1]) = L(T[n_2])$ .

Suppose then that at least one of  $v_1$  and  $v_2$  is not the root, and w.l.o.g. let this be the case for  $v_2$ . Then we will argue towards a contradiction. Let the mapping  $m_1$  be the embedding showing that  $\text{subtree}^{s_1}(v_1) \in L(T_b[n_1])$  and  $m_2$  the embedding showing  $\text{subtree}^{s_2}(v_2) \in L(T_b[n_2])$ . Consider then the composition of mappings  $m = m_2 \circ m_1$ . The mapping  $m$  is an embedding from  $T_b[n_1]$  to  $\text{subtree}^{s_2}(m_2(v_1))$ . Since  $v_1$  is equal to or a descendant of  $n_2$ , so is  $m_2(v_1)$  equal to or a descendant of  $m_2(n_2)$ , and the latter is equal to  $v_2$ , because  $m_2$  is the embedding witnessing that  $\text{subtree}^{s_2}(v_2) \in L(T_b[n_2])$ . As we remarked above,  $s_2 = c_x(T_b[n_1])$ . So,  $\text{subtree}^{s_2}(m_2(v_1)) = c_x(T_b[v])$  where  $v = m_2(v_1)$ , which is a strict descendant of  $n_1$ , by our assumption that  $v_2$  is a strict descendant of  $n_1$ . This implies that the mapping  $m$  is a witness to  $c_x(T_b[v]) \in L(T_b[n_1])$ , for  $v$  a strict descendant of  $n_1$ , and by Lemma 4.11, this leads to a contradiction.

To show that neither  $n_1$  nor  $n_2$  is an ancestor of or equal to  $o$ , suppose for contradiction that at least one of them is. If exactly one of them is an ancestor of or equal to  $o$ , say  $n_1$ , then  $T_b[n_1]$  contains a node labelled with  $(a, 1)$  for some  $a \in \Sigma$ , but  $T_b[n_2]$  does not contain such a node, by definition of the mapping  $\text{bool}_\Sigma$ . Therefore  $L(T_b[n_1]) \neq L(T_b[n_2])$ , which is a contradiction. If both  $n_1$  and  $n_2$  are ancestors of or equal to  $o$ , then, either  $n_1$  is an ancestor of  $n_2$ , or  $n_2$  is an ancestor of  $n_1$ . If  $n_1$  is an ancestor of  $n_2$ , and  $L(T_b[n_1]) = L(T_b[n_2])$ , we have

a contradiction by Corollary 4.12. The case is similar when  $n_2$  is an ancestor of  $n_1$ .

Hence,  $L(T_b[n_1]) = L(T_b[n_2])$  and neither  $n_1$  nor  $n_2$  is an ancestor of or equal to  $o$ . Then  $L(T[n_1]) = L(T[n_2])$ , as needed.  $\square$

#### 4.4 Testing Equivalence of DAG-Twigs and NSTA<sub>w</sub>s

Now that we have a small model property of DAG-twigs compared to NSTA<sub>w</sub>s (Theorem 4.20), we can simply decide twig-definability of an NSTA<sub>w</sub> by guessing the DAG-twig and testing equivalence. Here, we argue that equivalence of such an NSTA<sub>w</sub>  $M$  and a DAG-twig  $D$  can be decided in exponential time. In particular, we will reduce the latter problem to emptiness of alternating tree-walking automata operating on  $\text{BoolQ}_\Sigma(\mathcal{Q}_M)$  and  $\text{BoolQ}_\Sigma(\mathcal{Q}_D)$ .

Let  $D$  be a DAG-twig representing the twig  $T$ . The goal of this Section is to describe a procedure that, given  $D$ , constructs an alternating tree-walking automaton accepting  $L(T)$ , the tree language associated with  $T$ .

Although DAG-twigs operate directly on unranked trees, we will intermediately work with binary trees encoding these unranked trees. Following [27], for an (unranked) tree  $t$ , let  $\text{enc}(t)$  be its binary encoding, obtained as follows: The nodes of  $\text{enc}(t)$  are the nodes of  $t$  plus a set of leaf nodes marked #. Further, the root node of  $\text{enc}(t)$  is the root node of  $t$  and for any node, its left child in  $\text{enc}(t)$  is its first child in  $t$  (or # if its a leaf), and its right child in  $\text{enc}(t)$  is its next sibling in  $t$  (or # if it has none). In Fig. 5, we depicted an example of an unranked tree and its binary encoding.

We start by recalling the definition of these alternating tree walking automata, which operate on binary trees:

**Definition 4.26** (Alternating Tree-Walking Automata) Let  $\text{PosBool}(P)$  be the set of positive Boolean formulas over propositions  $P$  (i.e., formulas without negation),

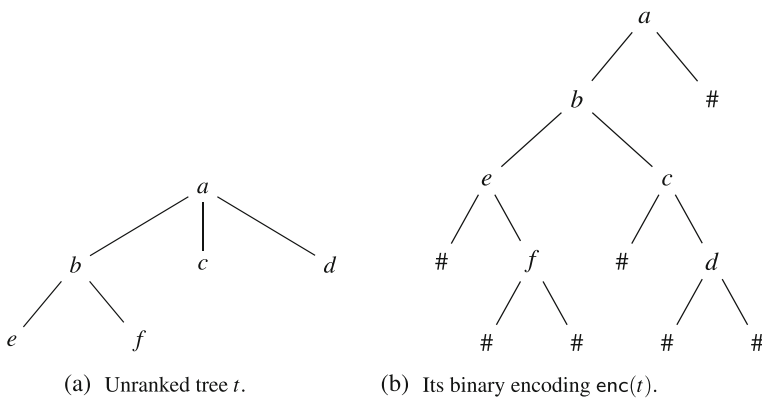


Fig. 5 An unranked tree and its binary encoding

but including true and false. An *alternating tree walking automaton with wildcards* (ATWA with wildcards) over binary trees is defined as a tuple  $W = (Q, \Sigma, \delta, q_0)$ , where

- $Q$  is a finite set of states,
- $\Sigma$  is a finite set of alphabet symbols,
- $\delta$  is a set of transition rules of the form  $(q, \sigma) \rightarrow \theta$ , where  $q \in Q, \sigma \in \Sigma_\diamond$ , and  $\theta$  is a formula from

$$\text{PosBool}(\{\swarrow, \searrow, -\} \times Q),$$

and

- $q_0$  is the initial state.

Recall that  $\Sigma_\diamond = \Sigma \uplus \{\diamond\}$ , where  $\diamond$  is the wildcard symbol.

The transition relation  $\delta$  should be such that for each pair  $(q, \sigma) \in Q \times \Sigma_\diamond$  there is at most one rule in  $\delta$  with  $(q, \sigma)$  as its left hand side. (If there would be two rules with the same left hand side, we can merge them into one rule by taking the disjunction of the right hand sides.) If  $(q, \sigma) \rightarrow \theta \in \delta$ , we also write  $\text{rhs}_W(q, \sigma) = \theta$ . Elements in  $\{\swarrow, \searrow, -\}$  denote directions in the tree. For a node  $u$  of  $t$ ,  $u \cdot \searrow$  (respectively,  $u \cdot \swarrow$ ) denotes the right child of  $u$  (respectively, left child of  $u$ ) if  $\text{lab}(u) \neq \#$  and is undefined otherwise. Further,  $u \cdot -$  is  $u$  itself (i.e.,  $-$  is used for stay transitions).

Given a binary tree  $t$ , a *run tree* of  $W$  on  $t$  is an unranked tree  $R$  in which each node is labelled by an element of  $\text{Nodes}^t \times Q$  such that the following holds. We say that an element  $a \in \Delta$  matches  $\sigma \in \Sigma_\diamond$  if either  $a = \sigma$  or  $a \notin \Sigma$  and  $\sigma = \diamond$ .

- The label of the root of  $R$  is  $(\text{root}(t), q_0)$  and
- for every node  $x$  of  $R$  with label  $(v, q_v)$ , if  $(q_v, \sigma) \rightarrow \theta \in \delta$  and  $\text{lab}^t(v)$  matches  $\sigma$ , then there is a set  $S \subseteq \{\swarrow, \searrow, -\} \times Q$  such that
  - for every  $(i, q') \in S$ ,  $v \cdot i$  is defined and there is a child  $y$  of  $x$  in  $R$  labelled  $(v \cdot i, q')$ ;
  - all children of  $x$  are labelled with  $(v \cdot i, q')$  such that  $(i, q') \in S$ ; and
  - the truth assignment that assigns true to all elements of  $S$ , and false to all other elements of  $\{\swarrow, \searrow, -\} \times Q$ , satisfies  $\theta$ .

A run tree  $R$  is *accepting* if, for every leaf node of  $R$  labelled  $(u, q)$ , there is a rule  $\text{rhs}_W(q, \sigma) = \text{true}$  such that  $\text{lab}^t(u)$  matches  $\sigma$ . A binary tree  $t$  is *accepted* by an ATWA  $W$  if there exists an accepting run tree of  $W$  on  $t$ .

By  $L(W)$  we denote the set of trees accepted by  $W$ .

We now show that, given a DAG-twig, we can efficiently construct an equivalent tree walking automaton. We note that it is well known that there is a connection between various XPath fragments and (two-way) alternating walking automata. Benedikt, Fan and Geerts [3] have shown that it is possible to construct, in linear time, a two-way alternating word automaton, accepting string encodings of trees defined by an XPath query. This construction, however, only works

when the considered trees have a fixed depth. Further, ten Cate and Lutz [36] have shown that it is possible to construct, in quadratic time, a two-way alternating tree automaton equivalent to a given XPath query. Our next lemma therefore uses rather standard techniques but, to be complete, we provide the full construction.

**Lemma 4.27** *Let  $D$  be a DAG-twig representing the twig  $T$ . An alternating tree walking automaton  $W$  with  $L(W) = \{enc(t) | t \in L(T)\}$  can be constructed in time  $O(|D|)$ .*

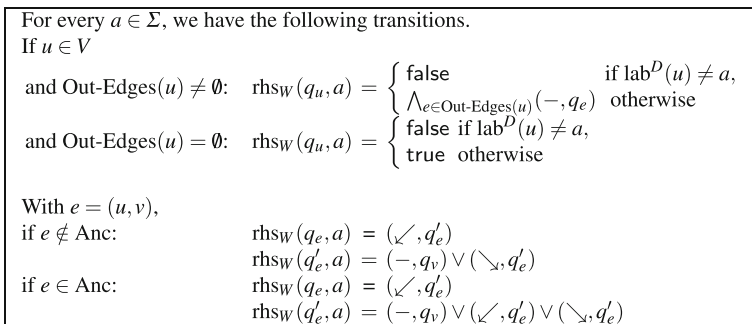
*Proof* Let  $D = (\Sigma, G, o, Anc)$  be a DAG-twig and let  $T$  be the corresponding unfolded twig. We construct  $W = (Q, \Sigma, \delta, q_0)$ , such that  $L(W) = \{enc(t) | t \in L(T)\}$ , as follows. The set  $Q$ , contains, for each node  $u$  of  $D$ , a state  $q_u$  and, for each edge  $e$  of  $G$ , the states  $q_e$  and  $q'_e$ . A state  $q_u$  or  $q_e$  can be seen as a pointer to a node  $u$  or edge  $e$  in the DAG-twig  $D$  for which the automaton guesses that the subgraph below  $u$  or  $e$ , respectively, can be matched at its current position in the tree  $T$ . The starting state  $q_0$  is the state corresponding to the root of  $D$ .

Finally, the transition relation  $\delta$  is given in Fig. 6. (Recall that  $rhs_W(q, a) = \theta$  if and only if  $(q, a) \rightarrow \theta$  is the unique rule in  $\delta$  with  $(q, a)$  as its left hand side.)

For  $G = (V, E)$  and node  $u \in V$  let  $Out-Edges(u)$  denote the set of outgoing edges of  $u$ , i.e., edges of the form  $(u, v)$  for some node  $v$  of  $V$ . Then, for each node  $u$  and each edge  $e = (u, v)$  the transitions given in Fig. 6 apply.

As an example let us look at the transitions from a state corresponding to the ancestor-edge ( $e \in Anc$ ). At first there is one step to the first child of the current node. From then on, the additional state  $q'_e$  is used to search for a proper sibling or descendant node.

It now follows by induction that, for a tree  $t$  and  $v \in V$ , there is an accepting run tree of  $W_{q_v} = (Q, \Sigma, \delta, q_v)$  on  $enc(t)$  if and only if, for each node  $v'$  of  $T$  that corresponds to  $v$ , we have  $t \in L(T[v'])$ .



**Fig. 6** Transitions of the ATWA from the Proof of Lemma 4.27

Finally, it is easy to see that the size of the resulting automaton is linear in the size of  $D$  and that it can be computed in linear time.  $\square$

The construction for proving Lemma 4.27 can be easily changed such that  $W$  accepts encodings of  $L(\text{BoolT}(T))$  instead of  $L(T)$ .

**Corollary 4.28** *Let  $D$  be a DAG-twig that represents the twig  $T$ . An alternating tree walking automaton  $W$  with  $L(W) = \{\text{enc}(t) \mid t \in L(\text{BoolT}(T))\}$  can be constructed in time  $O(|D|)$ .*

*Proof* The proof is almost the same as the one for Lemma 4.27. The only difference is that  $W$  needs to make a case distinction between the output node  $o$  from  $D$  and all other ones. The transition for the output node  $o$  then needs to test whether the label of the current node is  $(\text{lab}^D(o), 1)$  and all other transitions for nodes  $u$  of  $D$  test whether the label of the current node is  $(\text{lab}^D(u), 0)$ .  $\square$

We now reduce equivalence between an  $\text{NSTA}_w$  and a DAG-twig to the emptiness problem for ATWAs.

**Theorem 4.29** *Given a DAG-twig  $D$  and an  $\text{NSTA}_w$   $M$ , both over the same alphabet  $\Sigma$ , we can construct an ATWA  $W$  in polynomial time such that  $L(W) = \emptyset$  if and only if  $\text{BoolQ}_\Sigma(\mathcal{Q}_M) = \text{BoolQ}_\Sigma(\mathcal{Q}_D)$ .*

*Proof* Let  $D_{0,1} := \text{BoolT}(D)$ . Let  $N_{0,1}$  be the NTA with  $L(N_{0,1}) = \text{BoolQ}_\Sigma(\mathcal{Q}_M)$ , as obtained in Lemma 2.7. We construct an ATWA  $W$  that accepts a tree  $t$  if and only if  $t$  is in the symmetric difference of  $L(N_{0,1})$  and  $L(D_{0,1})$ . We assume w.l.o.g. that  $D_{0,1}$  and  $N_{0,1}$  have disjoint state sets.

When reading a tree  $t$ , the ATWA  $W$  starts with a stay transition at the root and guesses whether either

- $D_{0,1}$  would accept  $t$  and  $N_{0,1}$  would reject  $t$ ; or
- $D_{0,1}$  would reject  $t$  and  $N_{0,1}$  would accept  $t$ .

The ATWA  $W$  can do this in one transition:

$$(\text{root}(t), q_0) \rightarrow \left( \left( -, q_0^{D_{0,1}} \right) \wedge \left( -, \overline{q_0^{N_{0,1}}} \right) \right) \vee \left( \left( -, \overline{q_0^{D_{0,1}}} \right) \wedge \left( -, q_0^{N_{0,1}} \right) \right)$$

Here,  $q_0^{D_{0,1}}$  and  $q_0^{N_{0,1}}$  are the initial states of  $D_{0,1}$  and  $N_{0,1}$ , respectively. The remainder of the run of  $W$  starting with  $q_0^{D_{0,1}}$  (resp.,  $q_0^{N_{0,1}}$ ) therefore leads to acceptance if and only if  $D_{0,1}$  (resp.,  $N_{0,1}$ ) accepts  $t$ . Analogously, the states  $\overline{q_0^{D_{0,1}}}$  and  $\overline{q_0^{N_{0,1}}}$  are the states for the complement languages of  $D_{0,1}$  and  $N_{0,1}$ . The remainder of the run of  $W$  starting with  $\overline{q_0^{D_{0,1}}}$  (resp.,  $\overline{q_0^{N_{0,1}}}$ ) accepts if and only if  $D_{0,1}$  (resp.,  $N_{0,1}$ )

does *not* accept  $t$ . Notice that, since ATWAs can be complemented in polynomial time (analogously to [14], chapter 7),  $W$  can be constructed in polynomial time as well. □

**Theorem 4.30** *Testing equivalence between a DAG-twig  $D$  and an  $NSTA_w$   $M$  is EXPTIME-complete.*

*Proof* The lower bound follows from a reduction from the language universality problem for  $NTA_w$ s. The latter is EXPTIME-hard by Theorem 2.2. The reduction is similar to that in the proof of Theorem 3.9. It takes as input an  $NTA_w$   $N = (\Sigma, Q, \delta, F)$ . Let the  $NSTA_w$   $M$  and  $a \in \Sigma$  be as in the proof of Theorem 3.9. Recall that  $\mathcal{Q}_M$  selects the root node in all trees  $t = a(t_1, \dots, t_n)$  where for all  $1 \leq i \leq n, t_i \in L(N)$ . Let the DAG-twig  $D$  have only one node labelled  $a$ . Hence,  $\mathcal{Q}_M = \mathcal{Q}_D$  if and only if  $L(N) = \mathcal{T}_\Delta$ .

The upper bound follows from Theorem 4.29, since testing language emptiness for alternating tree walking automata with wildcards is the same as language emptiness for alternating tree walking automata without wildcards, and the latter problem is known to be in EXPTIME. (see, e.g., [7, 14]). □

### 4.5 Main Result

We are now ready to state and prove the main result of this section.

**Theorem 4.31** *Deciding whether for an  $NSTA_w$   $M$ ,  $\mathcal{Q}_M$  is twig-definable, is complete for EXPTIME.*

*Proof* For the lower bound, similarly to Theorem 3.9, we will reduce the problem of universality of  $NTA_w$ s to the problem considered here. Let  $N = (\Sigma, Q, \delta, F)$  be an  $NTA_w$ . We construct an  $NSTA_w$   $M$  such that  $\mathcal{Q}_M$  is twig-definable if and only if  $L(N) = \mathcal{T}_\Delta$ . Let  $q_{sel} \notin Q$  and define the  $NSTA_w$   $M = ((\Sigma, Q \uplus \{q_{sel}\}, \delta \uplus \{(q_{sel}, a) \mapsto (\sum_{p \in F} P)^*\}), \{q_{sel}\})$ . Then for any tree  $t' = a(t_1, \dots, t_n)$ , we have that  $\text{root}(t') \in \mathcal{Q}_M(t')$  if and only if, for each  $1 \leq i \leq n, t_i \in L(N)$ . In particular, we have that  $\mathcal{Q}_M$  selects the root of the tree  $t_{\text{small}} = a$ , consisting of just one node. However, by definition of twig queries, the only twigs that are able to select the root of  $t_{\text{small}}$  are of the form  $(\Sigma', t, o, \text{Anc})$  with  $t = a, o = \text{root}(t), \text{Anc} = \emptyset$ , and  $\Sigma' \supseteq \{a\}$ . Let  $T = (\{a\}, t, o, \text{Anc})$  be such a twig. Then,  $\mathcal{Q}_M$  is twig-definable if and only if  $\mathcal{Q}_M = \mathcal{Q}_T$ . However,  $\mathcal{Q}_M = \mathcal{Q}_T$  if and only if  $L(N) = \mathcal{T}_\Delta$ .

The upper bound is given by the following exponential-time algorithm. From Theorem 4.20, we know that if there exists a DAG-twig equivalent to  $M = ((\Sigma, Q, \delta, F), S)$ , there is one with alphabet  $\Sigma$ , and that has at most  $2 \cdot |Q|$  nodes. Therefore, we can enumerate every possible DAG-twig  $D$  with alphabet  $\Sigma$  and at most  $2 \cdot |Q|$  nodes and test whether  $D$  and  $M$  are equivalent. Theorem 4.30 states that we can test in exponential time whether a given DAG-twig  $D$  and a given  $NSTA_w$   $M$  are equivalent. Since the maximal size of each DAG-twig  $D$  is linear in our

input, this means that our total algorithm has an exponential-time test for each of the exponentially many DAG-twigs, which takes exponential time altogether.  $\square$

## 5 Conclusion

In this paper we have shown that deciding twig-definability of NSTAs is complete for EXPTIME.

There are many possible directions for future work. First of all, it would be interesting to identify meaningful subclasses of NSTAs for which deciding twig-definability is tractable. On the other hand, one could wonder how twig-queries can be extended while remaining within EXPTIME for testing twig-definability. When an NSTA is not equivalent to a twig, one could look at maximal sub- or minimal super-approximations, as, for instance, done in [17] for single-type EDTDs. Of course, other languages than XPath can be considered, like for instance, the Region Algebra [15], caterpillar expressions [18], or even tree-walking automata [7].

Another interesting question is the complexity of twig-definability in the finite alphabet case. The  $\text{NSTA}_{w,s}$  we consider in this paper have a wildcard symbol. However, what happens if we consider node-selecting tree automata without wildcard? In other words, assume that we have a finite alphabet  $\Sigma$  and we should decide whether a given NSTA (without wildcard transitions) can be rewritten to a twig that returns the same result on every  $\Sigma$ -tree. It seems that this problem requires a different technique than the one we developed here. Lemma 4.10, for instance, does not apply anymore for the simple reason that the construction of the characteristic tree  $c_x(T)$  requires a new alphabet symbol  $x$ .

## References

1. Albert, J., Giammerresi, D., Wood, D.: Normal form algorithms for extended context free grammars. *Theor. Comput. Sci.* **267**(1–2), 35–47 (2001)
2. Antonopoulos, T., Hovland, D., Martens, W., Neven, F.: Deciding twig-definability of node selecting tree automata. In: Deutsch, A. (ed.) *ICDT*, pp. 61–73. ACM (2012)
3. Benedikt, M., Fan, W., Geerts, F.: XPath satisfiability in the presence of DTDs. *J. ACM* **55**(2) (2008)
4. Benedikt, M., Fan, W., Kuper, G.M.: Structural properties of XPath fragments. *Theor. Comput. Sci.* **336**(1), 3–31 (2005)
5. Benedikt, M., Koch, C.: XPath leashed. *ACM Comput. Surv.* **41**(1) (2008)
6. Benedikt, M., Segoufin, L.: Regular tree languages definable in FO and in  $\text{FO}_{\text{mod}}$ . *ACM Trans. Comput. Log.* **11**(1) (2009)
7. Bojańczyk, M.: Tree-walking automata. In: *International Conference on Language and Automata Theory and Applications (LATA)*, pp. 1–2 (2008)
8. Bojańczyk, M., Muscholl, A., Schwentick, T., Segoufin, L.: Two-variable logic on data trees and XML reasoning. *J. ACM* **56**(3) (2009)
9. Bojańczyk, M., Parys, P.: XPath evaluation in linear time. *J. ACM*. To appear
10. Bojańczyk, M., Segoufin, L.: Tree languages defined in first-order logic with one quantifier alternation. *Logical Methods Comput. Sci.* **6**(4) (2010)



11. Bojańczyk, M., Walukiewicz, I.: Characterizing EF and EX tree logics. *Theor. Comput. Sci.* **358**(2–3), 255–272 (2006)
12. Calvanese, D., Giacomo, G.D., Lenzerini, M., Vardi, M.Y.: Node selection query languages for trees. In: AAAI Conference on Artificial Intelligence (AAAI) (2010)
13. Carme, J., Niehren, J., Tommasi, M.: Querying unranked trees with stepwise tree automata. In: International Conference on Rewriting Techniques and Applications (RTA), pp. 105–118 (2004)
14. Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., Löding, C., Tison, S., Tommasi, M.: Tree Automata Techniques and Applications. Available on <http://tata.gforge.inria.fr/> (2007)
15. Consens, M.P., Milo, T.: Algebras for querying text regions. Expressive power and optimization. *J. Comput. Syst. Sci.* **57**(3), 272–288 (1998)
16. Frick, M., Grohe, M., Koch, C.: Query evaluation on compressed trees (extended abstract). In: IEEE Symposium on Logic in Computer Science (LICS), pp. 188 (2003)
17. Gelade, W., Idziaszek, T., Martens, W., Neven, F.: Simplifying XML Schema: single-type approximations of regular tree languages. In: ACM Symposium on Principles of Database Systems (PODS), pp. 251–260 (2010)
18. Goris, E., Marx, M.: Looping caterpillars. In: Annual IEEE Symposium on Logic in Computer Science (LICS), pp. 51–60 (2005)
19. Gottlob, G., Koch, C.: Monadic datalog and the expressive power of languages for web information extraction. *J. ACM* **51**(1), 74–113 (2004)
20. Gottlob, G., Koch, C., Pichler, R.: Efficient algorithms for processing XPath queries. *ACM Trans. Database Syst.* **30**(2), 444–491 (2005)
21. Gottlob, G., Koch, C., Pichler, R., Segoufin, L.: The complexity of xpath query evaluation and XML typing. *J. ACM* **52**(2), 284–335 (2005)
22. Martens, W., Neven, F., Schwentick, T.: Complexity of decision problems for XML schemas and chain regular expressions. *SIAM J. Comput.* **39**(4), 1486–1530 (2009)
23. Martens, W., Neven, F., Schwentick, T., Bex, G.J.: Expressiveness and complexity of XML Schema. *ACM Trans. Database Syst.* **31**(3), 770–813 (2006)
24. Marx, M.: Conditional XPath. *ACM Trans. Database Syst.* **30**(4), 929–959 (2005)
25. Miklau, G., Suciu, D.: Containment and equivalence for a fragment of XPath. *J. ACM* **51**(1), 2–45 (2004)
26. Neven, F.: Design and Analysis of Query Languages for Structured Documents. PhD thesis, Limburgs Universitair Centrum (1999)
27. Neven, F.: Automata theory for XML researchers. *SIGMOD Record* **31**(3), 39–46 (2002)
28. Neven, F.: Attribute grammars for unranked trees as a query language for structured documents. *J. Comput. Syst. Sci.* **70**(2), 221–257 (2005)
29. Neven, F., den Bussche, J.V.: Expressiveness of structured document query languages based on attribute grammars. *J. ACM* **49**(1), 56–100 (2002)
30. Neven, F., Schwentick, T.: Expressive and efficient pattern languages for tree-structured data. In: ACM Symposium on Principles of Database Systems (PODS), pp. 145–156 (2000)
31. Neven, F., Schwentick, T.: Query automata over finite trees. *Theor. Comput. Sci.* **275**(1-2), 633–674 (2002)
32. Neven, F., Schwentick, T.: On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Logical Methods Comput. Sci.* **2**(3) (2006)
33. Place, T., Segoufin, L.: Deciding definability in  $FO_2(<)$  (or XPath) on trees. In: Annual IEEE Symposium on Logic in Computer Science (LICS), pp. 253–262 (2010)
34. Schwentick, T.: XPath query containment. *SIGMOD Record* **33**(1), 101–109 (2004)
35. Seidl, H.: Deciding equivalence of finite tree automata. *SIAM J. Comput.* **19**(3), 424–437 (1990)
36. ten Cate, B., Lutz, C.: The complexity of query containment in expressive fragments of XPath 2.0. *J. ACM* **56**(6) (2009)
37. ten Cate, B., Segoufin, L.: Transitive closure logic, nested tree walking automata, and XPath. *J. ACM* **57**(3) (2010)
38. Thomas, W.: Languages, automata, and logic. In: Rozenberg, G., Salomaa, A. (eds.) *Handbook of Formal Languages*, volume 3, chapter 7, Springer (1997)