Deciding Twig-definability of Node Selecting Tree Automata[*]

Timos Antonopoulos
Hasselt University and
Transnational University of Limburg
timos.antonopoulos@uhasselt.be

Dag Hovland
University of Oslo
hovland@ifi.uio.no

Wim Martens[†]
Institut für Informatik
Universität Bayreuth
wim.martens@uni-bayreuth.de

Frank Neven
Hasselt University and
Transnational University of Limburg
frank.neven@uhasselt.be

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

Categories and Subject Descriptors

F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages; H.2.1 [**Database Management**]: Logical Design

General Terms

Algorithms, Design, Theory

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 [25, 26], automata [11, 14, 28], and logic [10, 17, 27]. 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. [3] 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., [4, 8, 34]), 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 [30]. 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

[*] 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).

ICDT 2012, March 26–30, 2012, Berlin, Germany.

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 [9] 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.[1] We refer to the latter as 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 [14, 23] 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 $\mathsf{NFA}(M)$ for a given NSTA M, such that M is equivalent to $\mathsf{NFA}(M)$ 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 [20] 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., [18, 19, 7]. The containment and satisfiability problems for XPath have been deeply studied in the database literature, for example in [22, 29, 6, 33]. The expressiveness of various fragments and extensions of XPath have been investigated in, e.g., [2, 21, 34]. We refer to [3, 31] for surveys on these problems. To the best of our knowledge the above mentioned results of Place and Segoufin [30] and Bojańczyk and Walukiewicz [9] 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 Δ always denote an infinite set of *labels.* Intuitively, Δ is our abstraction of the set of XML-tags. We assume that we can test equality between elements from Δ in constant time. We denote by Δ^* the set of finite strings over Δ. By ε 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 child$(\cdot,\cdot)$ and next-sibling$(\cdot,\cdot)$. Here, $\sigma(u)$ expresses that u is a node with label σ, and child(u, v) (respectively, next-sibling(u, v)) expresses that v is a child (respectively, the next sibling) of u. When next-sibling(u, v) holds, we sometimes also say that v is (immediately) to the right of u. We write Nodes^t for the set of nodes of t. The set of edges of a tree t, denoted by Edges^t is the set of pairs (u, v) such that child(u, v) holds in t. The root node of t is denoted by $\mathsf{root}(t)$. We define the *size of* t, denoted by $|t|$, to be the number of nodes of t. We denote a tree with root labelled σ and subtrees $t_1, \ldots, t_n$ as $\sigma(t_1, \ldots, t_n)$. 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 $i = 1, \ldots, n$, we have that $(v_{i-1}, v_i) \in \mathsf{Edges}^t$. 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 $v \in \mathsf{Nodes}^t$ is equal to the length of the (unique) path from $\mathsf{root}(t)$ 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 $v \in \mathsf{Nodes}^t$ there exists a unique $\sigma \in \Delta$ such that $\sigma(v)$ holds. We denote the label of v by $\mathsf{lab}^t(v)$. For a node v in a tree t, the *ancestor-string* of v, denoted $\mathsf{ancstr}^t(v)$, 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, $\mathsf{ancstr}^t(v)$ is the sequence $\mathsf{lab}^t(v_0) \cdots \mathsf{lab}^t(v_n)$, where $v_0 \cdots v_n$ is the path from $\mathsf{root}(t)$ to v. For a tree t and a node $v \in \mathsf{Nodes}^t$, the *subtree of* t *at* v, denoted by $\mathsf{subtree}^t(v)$, is the tree induced by all the nodes u such that there is a (possibly empty)

[1] *Filter* is sometimes also called *predicate*, e.g., in the XPath specification by the World Wide Web Consortium.

path from v to u. In particular, for any tree t and leaf node v, $\mathsf{subtree}^t(v) = \mathsf{lab}^t(v)$ and, for any other node u, $\mathsf{subtree}^t(u) = \mathsf{lab}^t(u)(\mathsf{subtree}^t(u_1), \ldots, \mathsf{subtree}^t(u_n))$, where $u_1, \ldots u_n$ are the children of u from left to right.

Similarly, the *context of t at v*, denoted by $\mathsf{context}^t(v)$, 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, $\mathsf{context}^t(v)$ is defined inductively as follows. Let $\mathsf{context}^t(\mathsf{root}(t)) = \#$ 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 $v_1, \ldots, v_n$, from left to right. Assume that $v = v_i$. Then, $\mathsf{context}^t(v)$ is the tree obtained by replacing the unique $\#$-labelled node in $\mathsf{context}^t(u)$ with the tree $\mathsf{lab}^t(u)\big(\mathsf{subtree}^t(v_1), \ldots, \mathsf{subtree}^t(v_{(i-1)}), \#, \mathsf{subtree}^t(v_{(i+1)}), \ldots, \mathsf{subtree}^t(v_n)\big)$.

By $t[v \leftarrow t']$ we denote the tree constructed from t by replacing the subtree $\mathsf{subtree}^t(v)$ 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 $\mathsf{context}^t(v)$ 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 Σ 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 $\mathsf{Labels}(r)$ 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.

Since a twig pattern query (as defined in Section 4) uses only a finite set of labels, but matches trees with an infinite set of labels, we will use a wildcard symbol "$\diamond$" to give automata the same power. We assume that the single-symbol wildcard symbol $\diamond$ is not in Δ 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 Δ-symbol not in Σ. An *NFA (with wildcards)* is a tuple $A = (\Delta, \Sigma, Q, q_I, \delta, F)$, 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 $\delta : Q \times \Sigma_\diamond \to 2^Q$ is the transition function. From the transition function δ, we define the extended transition function $\delta^* : (Q \times \Delta^*) \to 2^Q$ which can read entire Δ-strings. In particular, $\delta^*(q, \varepsilon) = \{q\}$, $\delta^*(q, a) = \delta(q, a)$ 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 *NTA (with wildcards)* is a tuple $N = (\Delta, \Sigma, Q, \delta, F)$ where Q is a finite set of states, $F \subseteq Q$ is the set of final states, and the transition function $\delta : Q \times \Sigma_\diamond \to \mathcal{R}_Q$ 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 Σ.

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

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

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

$$\lambda(v_1) \cdots \lambda(v_n) \in L(\delta(\lambda(v), \diamond)).$$

When v has no children, the criterion reduces to

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

A run is *accepting* if the root is labelled with a state in F. A tree is *accepted* if there is an accepting run. The set of all accepted trees 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 NTAs are *equivalent* if they define the same language.

Notice that we could have let a wildcard match any Δ-symbol rather than any symbol in $\Delta - \Sigma$. As the former can be simulated by the latter (by simply adding an extra transition for every Σ-symbol) but not vice-versa, we decided to use the more powerful notion.

For any $p \in Q$, let $N_p = (\Delta, \Sigma, Q, \delta, \{p\})$. 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'.

Unless explicitly mentioned otherwise, we will assume that NTAs do not have *useless* states. That is, for each state, there is at least one accepting run of the NTA on some tree where some node is labelled by that state. We justify this remark by the following lemma.

Lemma 2.1. *Each NTA with wildcards can be converted into an equivalent NTA with wildcards without useless states in polynomial time.*

Since useless states can be removed efficiently, we also do not need to bother about removing useless states in the NTAs we construct in our algorithms.

The proof of the following theorem is a straightforward reduction to and from the finite alphabet case [32].

Theorem 2.2. 1. *Deciding equivalence of NTAs with wildcards is EXPTIME-complete.*

2. *Deciding universality of NTAs with wildcards is EXPTIME-complete.*

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 $t \in \mathcal{T}_\Delta$, $\mathcal{Q}(t) \subseteq \mathsf{Nodes}^t$.

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., [35]) which extends the set of labels to $\Delta \times \{0, 1\}$ and labels selected nodes by 1 and non-selected nodes by 0. For a tree $t \in \mathcal{T}_\Delta$, we denote the set of nodes labelled by a symbol in Σ, as $\mathsf{Nodes}^t(\Sigma)$, and $\Delta_{0,1}$ is the alphabet $(\Delta - \Sigma) \uplus \Sigma \times \{0, 1\}$. Then, let $\mathsf{Bool}(\cdot, \cdot)$ be the mapping defined as follows. For each tree $t \in \mathcal{T}_\Delta$ and $v \in \mathsf{Nodes}^t(\Sigma)$, let $\mathsf{Bool}(t, v) \in \mathcal{T}_{\Delta_{0,1}}$ be the tree with the same nodes as t, but with the labelling function defined as follows: $\mathsf{lab}^{\mathsf{Bool}(t,v)}(v) = (\mathsf{lab}^t(v), 1)$, for $v' \in \mathsf{Nodes}^t(\Sigma) - \{v\}$, $\mathsf{lab}^{\mathsf{Bool}(t,v)}(v') = (\mathsf{lab}^t(v'), 0)$, and for $v' \notin \mathsf{Nodes}^t(\Sigma)$, $\mathsf{lab}^{\mathsf{Bool}(t,v)}(v') = \mathsf{lab}^t(v')$. Then, let

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

Finally, for a unary query $\mathcal{Q}$, let

$$\mathsf{Bool}(\mathcal{Q}) = \bigcup_{\substack{t \in \mathcal{T}_\Delta \\ v \in \mathcal{Q}(t)}} \{\mathsf{Bool}(t,v)\}.$$

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 [14, 23].

Definition 2.4 (NSTA). *A non-deterministic selecting tree automaton (with wildcards) or NSTA (with wildcards) M, is a pair (N, S), where N is an NTA (with wildcards) 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 λ such that $\lambda(v) \in S$ and $\mathsf{lab}^t(v) \in \Sigma$. 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 NTA.*

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

We say that two NSTAs are *equivalent* if they define the same query. The following theorem says that deciding equivalence of NSTAs is in EXPTIME. Specifically, Theorem 2.6 follows directly from the following lemma and Theorem 2.2.

Lemma 2.5. *For any NSTA with wildcards, $M = (N_M, S)$, we can construct in polynomial time an NTA with wildcards N such that $L(N) = \mathsf{Bool}(\mathcal{Q}_M)$, and such that $|Q_N| = 2 \cdot |Q_{N_M}|$, where Q_N is the set of states of N and Q_{N_M} is the set of states of N_M.*

Theorem 2.6. *Deciding equivalence of NSTAs with wildcards is EXPTIME-complete.*

In the remainder, whenever we say NFA, NTA, or NSTA we always refer to our definition with wildcards.

3. REGULAR PATH DEFINABILITY

In this section, we consider regular path definability. Here, we use NFAs to define regular paths. More precisely, we investigate in Section 3.1 whether a query given by an NSTA can already be defined by an NFA. 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-definability

We first formally introduce queries defined by NFAs.

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

As selection of a node only depends on the ancestor-string, NFA-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 $v_1 \in \mathsf{Nodes}^{t_1}$ and $v_2 \in \mathsf{Nodes}^{t_2}$, if $v_1 \in \mathcal{Q}(t_1)$ and $\mathsf{ancstr}^{t_1}(v_1) = \mathsf{ancstr}^{t_2}(v_2)$, then also $v_2 \in \mathcal{Q}(t_2)$.

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

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

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

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

Basically, the automaton $\mathsf{NFA}(M)$ is constructed from M by turning it into an NFA. 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). Specfically, any state in $\mathsf{Labels}(\delta_M(q,a))$ can be assigned to a node whose parent is labelled a and is assigned state q where δ_M is the transition function of M.[2] The formal construction is given next:

Definition 3.4. For an NSTA with wildcards $M = (N, S)$, where $N = (\Delta, \Sigma, Q, \delta, F)$, and for $q_I \notin Q$, define the NFA

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

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

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

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

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

The query defined by $\mathsf{NFA}(M)$ is always *complete*, i.e., 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 $\mathsf{NFA}(M)$ is *sound* as well, i.e., each node selected by $\mathsf{NFA}(M)$ is also selected by M. To facilitate the proofs below, we introduce the notation $\mathsf{lab}^{t,\Sigma}(v)$. If $\mathsf{lab}^t(v) \in \Sigma$, then let $\mathsf{lab}^{t,\Sigma}(v) = \mathsf{lab}^t(v)$. Otherwise, let $\mathsf{lab}^{t,\Sigma}(v) = \diamond$.

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

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

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

[2] Recall that $\mathsf{Labels}(r)$ is the set of symbols occurring in regular expressions r.

Proof. (1) Let M and $\mathsf{NFA}(M)$ be as in Definition 3.4. Assume $v \in \mathcal{Q}_M(t)$ for some v and t. We will prove that also $v \in \mathcal{Q}_{\mathsf{NFA}(M)}(t)$. Let $v_1 \cdots v_n$ be the path from the root to v in t, and let $q_1, \ldots, q_n$ be the states assigned to these nodes by an accepting run λ of M on t. Hence $q_1 \in F$ and $q_n \in S$. By induction on i, where $1 \leq i \leq n$, one can prove that $(q_i, \mathsf{lab}^{t,\Sigma}(v_i)) \in \delta'^*(q_I, \mathsf{ancstr}^t(v_i))$. Since $q_n \in S$ and λ is an accepting run, $(q_n, \mathsf{lab}^{t,\Sigma}(v_n)) \in F'$. This implies that $\mathsf{ancstr}^t(v) \in L(\mathsf{NFA}(M))$, and hence $v \in \mathcal{Q}_{\mathsf{NFA}(M)}(t)$.

(2) Let $M = (N, S)$, where $N = (\Delta, \Sigma, Q, \delta, F)$, and $\mathsf{NFA}(M) = (\Delta, \Sigma, (Q \times \Sigma_\diamond) \cup \{q_I\}, q_I, \delta', F')$ as in Definition 3.4. Assume that for some tree t and node $v \in \mathsf{Nodes}^t$, $v \in \mathcal{Q}_{\mathsf{NFA}(M)}(t)$. We will show that there exists a tree t' and $v' \in \mathsf{Nodes}^{t'}$, such that $v' \in \mathcal{Q}_M(t')$ and $\mathsf{ancstr}^t(v) = \mathsf{ancstr}^{t'}(v')$. By our assumption that $\mathcal{Q}_M$ is ancestor-based, it will follow that $v \in \mathcal{Q}_M(t)$.

Let $v_1, \ldots, v_n$ be the nodes on the path from the root to v (including the root and v), and let $w = \mathsf{ancstr}^t(v)$. Furthermore, let $q_I, (p_1, \mathsf{lab}^{t,\Sigma}(v_1)), \ldots, (p_n, \mathsf{lab}^{t,\Sigma}(v_n))$ be the states visited by $\mathsf{NFA}(M)$ (in order) when matching w. Specifically, this means $(p_n, \mathsf{lab}^{t,\Sigma}(v_n)) \in F'$, $p_n \in S$, and $p_1 \in F$.

Recall that we have defined $N_p = (\Delta, \Sigma, Q, \delta, \{p\})$ for $p \in Q$ as the NTA 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 $\mathsf{lab}^t(v_{n-i}) \cdots \mathsf{lab}^t(v_n) = \mathsf{ancstr}^{t'_{n-i}}(v'_{n-i})$, 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 $p_{n-i}, \cdots, p_n$, respectively.

- The base case $i = 0$ is easy, since we know $p_n \in S$ and $(p_n, \mathsf{lab}^{t,\Sigma}(v_n)) \in F'$, and by construction the latter implies $\delta(p_n, \mathsf{lab}^{t,\Sigma}(v_n))$ is defined. Therefore, there is a tree t'_n with its root labelled with $\mathsf{lab}^t(v_n)$, 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 $\mathsf{NFA}(M)$ on w we must have that $(p_{n-i}, \mathsf{lab}^{t,\Sigma}(v_{n-i}))$ is in
$$\delta'((p_{n-i-1}, \mathsf{lab}^{t,\Sigma}(v_{n-i-1})), \mathsf{lab}^{t,\Sigma}(v_{n-i}))$$
By definition of the transition function δ', this implies
$$p_{n-i} \in \mathsf{Labels}(\delta(p_{n-i-1}, \mathsf{lab}^{t,\Sigma}(v_{n-i-1}))).$$
In particular, there is a string $w_q = q_1 \cdots p_{n-i} \cdots q_r$, such that $w_q \in L(\delta(p_{n-i-1}, \mathsf{lab}^{t,\Sigma}(v_{n-i-1})))$. 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
$$t'_{n-i-1} = \mathsf{lab}^t(v_{n-i-1})(s_{q_1} \cdots t'_{n-i} \cdots s_{q_r}).$$
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 NFA-definability and ancestor-based regular queries.

Lemma 3.6. *For an NSTA M, the following are equivalent*

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

Proof. (1) ⇒ (2) holds by Lemma 3.3.
(2) ⇒ (3) holds by Lemma 3.5.
(3) ⇒ (1) holds by definition of NFA-definable query. □

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

Theorem 3.7. *Deciding whether for an NSTA M, $\mathcal{Q}_M$ is NFA-definable, is complete for EXPTIME.*

Proof sketch. The lower bound follows from a reduction from the universality problem for NTAs (cf. Theorem 2.2). The reduction takes as input an NTA $N = (\Delta, \Sigma, Q, \delta, F)$ and constructs an NSTA $M = (N', S)$ as follows. Let $q_{\text{sel}} \notin Q$ and let a be some symbol in Σ. Then let $N' = (\Delta, \Sigma, Q \cup \{q_{\text{sel}}\}, \delta \cup \{(q_{\text{sel}}, a) \mapsto (\sum_{p \in F} p)^*\}, \{q_{\text{sel}}\})$ and $S = \{q_{\text{sel}}\}$. We show that $\mathcal{Q}_M$ is NFA-definable iff $L(N) = \mathcal{T}_\Delta$. The query $\mathcal{Q}_M$ selects the root node in all trees $t = a(t_1, \ldots, t_n)$ 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 NFA-definable, namely by any NFA selecting exactly the first letter in words in $a\Delta^*$. On the other hand, suppose $\mathcal{Q}_M$ is NFA-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 $\mathcal{Q}_M$. Hence $L(N)$ must be exactly $\mathcal{T}_\Delta$.

It remains to show the upper bound. By Lemma 3.6, it suffices to test $\mathcal{Q}_M = \mathcal{Q}_{\mathsf{NFA}(M)}$. To this end, we note that it is possible to construct an NSTA $\mathsf{NSTA}(M)$ equivalent to $\mathsf{NFA}(M)$, that is, $\mathcal{Q}_{\mathsf{NSTA}(M)} = \mathcal{Q}_{\mathsf{NFA}(M)}$, in time polynomial in the size of M and apply Theorem 2.6. □

3.2 Single-type EDTDs

In [20], 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 [15, 20].

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 3.7 does not seem to imply or follow directly from the corresponding result on single-type EDTDs in [20].

3.3 Tractability

The EXPTIME-hardness in Theorem 3.7 is solely due to the expressiveness of NSTAs. Indeed, when M as constructed in the proof is indeed equivalent to an NFA, that NFA 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 NSTAs 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 $\mathcal{Q}_{\mathsf{NFA}(M)} \subseteq \mathcal{Q}_M$ is in PTIME for every $M \in \mathcal{M}$, renders the NFA-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 NSTAs

In this section, we address twig-definability of NSTAs. 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 and NSTAs can be exponentially more succinct than one another. This means that we cannot simply guess a small, i.e. polynomially bounded, twig equivalent to a given NSTA. 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 NSTA is equivalent to a twig its DAG-representation is at most of linear size. We further show in Section 4.4 that equivalence of NSTAs 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 that testing twig-definability of NSTAs is EXPTIME-complete.

4.1 Basics

We start by defining twigs:

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

An *embedding* of T on a tree s is a total mapping m from Nodes^t to Nodes^s such that

- the root of t is mapped to the root of s,
- $\mathsf{lab}^t(v) = \mathsf{lab}^s(m(v))$, for all $v \in \mathsf{Nodes}^t$, and
- for every two nodes $v_1, v_2 \in \mathsf{Nodes}^t$
  - if $(v_1, v_2) \in \mathsf{Edges}^t - \text{Anc}$, $(m(v_1), m(v_2)) \in \mathsf{Edges}^s$;
  - 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 Δ-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 on the set of nodes $v \in \mathsf{Nodes}^s$ for which there is an embedding m of T into s for which $m(o) = v$. In Figure 1, we give an example of a twig and an embedding.

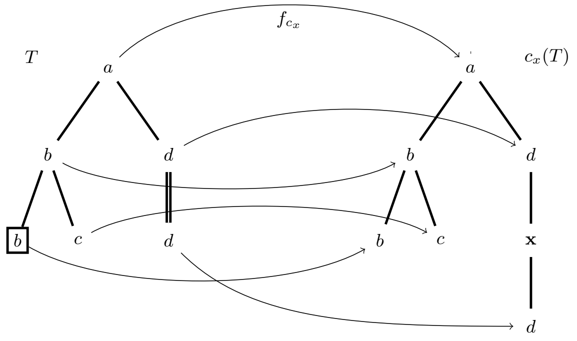


Figure 1: A twig $T = (t, o, \mathbf{Anc})$ on the left, its characteristic tree $c_x(T)$ on the right and the canonical embedding f_{c_x}. Edges of the twig in Anc are depicted as double edges and o is depicted with a rectangle.

For a node $v \in \mathsf{Nodes}^t$, let $T[v]$ be the subpattern rooted at v, that is, $T[v] = (\mathsf{subtree}^t(v), o, \text{Anc} \cap \{\mathsf{Edges}^{\mathsf{subtree}^t(v)}\})$. We will use the following basic property of subpatterns.

Lemma 4.2. *For a twig $T = (t, o, Anc)$, a tree $s \in L(T)$, and an embedding m of T into s, $\mathsf{subtree}^s(m(v)) \in L(T[v])$ for every $v \in \mathsf{Nodes}^t$.*

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

Definition 4.3. *Two twigs T and T' are* language-equivalent *iff $L(T) = L(T')$. They are* query-equivalent *iff $\mathcal{Q}_T = \mathcal{Q}_{T'}$.*

Definition 4.4 (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 model defined by Miklau and Suciu [22] where every ancestor edge is replaced by a sequence of wildcards.

Definition 4.5 (The Characteristic Tree). For a twig $T = (t, o, \text{Anc})$ over the alphabet Σ, and x a label not in Σ, *the characteristic tree* $c_x(T)$ of T is obtained from t, by replacing all edges (v_1, v_2) in Anc with a path v_1, v_3, v_2 of length 2, with labels $\mathsf{lab}^t(v_1)$, x and $\mathsf{lab}^t(v_2)$, respectively. Here, for every edge, v_3 is a new node.

So, every node $v \in \mathsf{Nodes}^t$, corresponds to a unique node of the tree $c_x(T)$. We denote by f_{c_x} the function from T to $c_x(T)$ which maps every node $v \in \mathsf{Nodes}^t$ to itself. Lemma 4.6 shows that f_{c_x} is an embedding and we will refer to it as the *canonical embedding* from T into its characteristic tree. Furthermore, as f_{c_x} is bijective over the original nodes we can use its inverse $f_{c_x}^{-1}$. Figure 1 illustrates $c_x(T)$ and f_{c_x}.

Lemma 4.6. *For any twig $T = (t, o, Anc)$ over the alphabet Σ and $x \notin \Sigma$, there is an embedding m of T into $c_x(T)$ such that*

- *for all nodes $v \in \mathsf{Nodes}^t$, $m(v) = f_{c_x}(v)$, and*
- *f_{c_x} is surjective over the set of all nodes in $c_x(T)$ not labelled by x.*

The proof of the following lemma is similar to the proof of Proposition 3 of [22].

Lemma 4.7. *For any two twigs T, U over an alphabet Σ with $x \notin \Sigma$, $c_x(T) \in L(U)$ implies $L(T) \subseteq L(U)$.*

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 4.8 (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 4.9.
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 $T = (t, o, Anc)$ is query-minimal, then for all $v \in \mathsf{Nodes}^t$ where $o \notin \mathsf{Nodes}^{\mathsf{subtree}^t(v)}$, the twig $T[v]$ is language-minimal.*
4. *If $T = (t, o, Anc)$ is a language-minimal twig, then for all nodes $v \in \mathsf{Nodes}^t$, $T[v]$ is also 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 4.10. *For a twig $T = (t, o, Anc)$, x not a label in t, 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 $\exists u \in \mathsf{Nodes}^{\mathsf{subtree}^t(v'')} : c_x(T[u]) \in L(T[v'])$.*

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

Using Lemma 4.10, we can show the following.

Lemma 4.11. *For a language-minimal twig $T = (t, o, Anc)$, there is exactly one embedding of T into $c_x(T)$.*

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 $f_{c_x}(u)$ in $c_x(T)$ by a new tree t' resulting in the tree $s = c_x(T)[f_{c_x}(u) \leftarrow t']$, 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 4.12. *Let x be a label not in Σ. For a language-minimal twig $T = (t, o, Anc)$ over Σ, a node $u \in \mathsf{Nodes}^t$, and a tree $t' \in \mathcal{T}_\Delta$, if $c_x(T)[f_{c_x}(u) \leftarrow t'] \in L(T)$, then there is a node $u' \in \mathsf{Nodes}^{t'}$ such that $\mathsf{subtree}^{t'}(u') \in L(T[u])$.*

Next, we discuss succinctness of twigs and NSTAs.

Theorem 4.13.
1. *There is a family of NSTAs M_n (for $n \in \mathbb{N}$) of size $\mathcal{O}(n)$ such that the smallest 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 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 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 Nodes^t, the *subgraph t' of t induced by S*, is the tree with $\mathsf{Nodes}^{t'} = S$ and $\mathsf{Edges}^{t'} = (S \times S) \cap \mathsf{Edges}^t$.

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

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

Then $\mathcal{Q}_{M_n}$ contains exactly the pairs $(s, \mathsf{root}(s))$ that have an induced subgraph s', containing the root $\mathsf{root}(s)$ of s, and such that s' is the complete binary tree 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. Notice that for each $n \in \mathbb{N}$, the smallest tree s such that $(s, \mathsf{root}(s)) \in \mathcal{Q}_{M_n}$ has $2^n - 1$ nodes.

We now argue that the minimal twig must have at least $2^n - 1$ nodes. If the query T_n selects the root of a tree t, then t has a complete binary tree of height n as a subtree at its top. Towards a contradiction, assume that there exists a twig $T_n' = (t', o', \mathrm{Anc}')$ query-equivalent to M_n that contains fewer than $2^n - 1$ nodes. We know that T_n' selects the root of t'. However, if t' has fewer than $2^n - 1$ nodes, this means that it does not contain a complete binary tree of height n and contradicts that T_n' is query-equivalent to M_n.

(2) It can be proved by induction on the size n of the twig $T = (t, o, \mathrm{Anc})$, 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 $L(N) = L(T)$.

(3) For $n \in \mathbb{N}$, let $\Sigma_n = \{a, a_1, \ldots, a_n\}$ and let $T_n = (t_n, o, \mathrm{Anc})$ be the twig where $t_n = a(a_1, \ldots, a_n)$, $o = \mathsf{root}(t)$ and $\mathrm{Anc} = \mathsf{Edges}^t$. For each n, T_n contains $n+1$ nodes, and the trees in $L(T_n)$ are exactly the trees s such that there are nodes $v, v_1, \ldots, v_n \in \mathsf{Nodes}^s$ with $v = \mathsf{root}(s)$, all $v_1, \ldots, v_n$ are different from $\mathsf{root}(s)$, and $\mathsf{lab}^s(v) = a$, $\mathsf{lab}^s(v_1) = a_1, \ldots,$ $\mathsf{lab}^s(v_n) = a_n$.

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 $\mathsf{lab}^{t_S}(v_i) = a_i$ and such that t_S has no nodes v with $\mathsf{lab}^{t_S}(v) \notin S$. Furthermore, for every such subset S, denote by $\overline{S}$ the set $\{a_1,\dots,a_n\} - S$. Notice that, for every S, the tree $a(t_S, t_{\overline{S}}) \in L(T_n)$.

Suppose then for contradiction, that there exists an NSTA M with fewer than $2^n - 2$ states, accepting the language $L(T_n)$. Then, by the pigeon hole principle, there must be two different, non-empty, strict subsets S_1 and S_2 of $\Sigma - \{a\}$, 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 λ_1 and λ_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 λ on the tree $t = a(t_{S_1}, t_{\overline{S_2}})$. Indeed, this accepting run λ is the same as λ_1 on the subtree t_{S_1}, it is the same as λ_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 $\mathcal{L}(T_n)$. This means that M does not accept $\mathcal{L}(T_n)$ and is a contradiction. $\square$

4.3 DAG-Twigs

Theorem 4.13(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 Σ has an associated labelling function $\mathsf{lab}^G : V \to \Sigma$. We assume that all DAGs have exactly one vertex with no incoming edges (called the root and denoted by $\mathsf{root}(G)$) 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 $\mathsf{clean}^G(v)$ 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 $\mathsf{unfold}(G)$. When $|V| = 1$, $\mathsf{unfold}(G)$ is a single node with the same label as $\mathsf{root}(G)$. When $|V| > 1$, let $U = \{u \in V \mid (\mathsf{root}(G), u) \in E\}$ and let $u_1 <_U \cdots <_U u_m$ be an arbitrary ordering of the nodes in U. Then, for each $1 \leq k \leq m$, let $G_k = \mathsf{clean}^{G - \{\mathsf{root}(G)\}}(u_k)$. The tree $\mathsf{unfold}(G)$ is then defined as

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

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

Definition 4.14 (Dag-Twig). *A DAG-twig is a tuple $D = (G, o, \mathrm{Anc})$, where $G = (V, E)$ is a DAG over Σ, the node $o \in V$ is such that there is exactly one path from the root to o in G, and $\mathrm{Anc} \subseteq E$. The query defined by D, denoted by $\mathcal{Q}_D$, is the query $\mathcal{Q}_T$ where T is the twig $(\mathsf{unfold}(G), o_G, \mathrm{Anc}_G)$ for which*

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

We say that the DAG 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 $\mathsf{fold}^G(o_G) = o$. 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 NSTA is twig-definable, there exists an equivalent twig of at most linear size.

Theorem 4.15. *For an NSTA $M = (N, S)$, if $\mathcal{Q}_M$ is twig-definable, then there exists an equivalent DAG-Twig D with at most $2 \cdot |Q_N|$ nodes, where Q_N is the set of states of N.*

Before we start proving Theorem 4.15, we introduce a definition and some lemmas.

Definition 4.16. *For a twig $T = (t, o, Anc)$, the twig $\mathsf{Bool}(T)$ is defined as $(\mathsf{Bool}(t, o), o, Anc)$.*

We can prove the following:

Lemma 4.17. *If a twig T is query-minimal, then $\mathsf{Bool}(T)$ is language-minimal.*

Lemma 4.18. *For a twig $T = (t, o, Anc)$ and a non-root node $v \in \mathsf{Nodes}^t$, $c_x(T[v]) \notin L(T)$.*

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 $v_1 \notin \mathsf{Nodes}^{\mathsf{subtree}^t(v)}$, all paths in $\mathsf{subtree}^t(v)$ 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 helpful corollary is the following:

Corollary 4.19. *For a non-root node v of a twig T, $L(T[v]) \not\subseteq L(T)$.*

The remainder of this section is devoted to the proof of Theorem 4.15.

Proof. Let D be the smallest DAG-twig representing a query-minimal twig equivalent to M. Let $T = (t, o, \mathrm{Anc})$ 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 $N = (\Delta, \Sigma, Q_N, \delta_N, F_N)$. 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 $x \notin \Delta$. Recall that 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 over the alphabet $\Delta_{0,1}$. Specifically, let N_b be the NTA accepting $\mathsf{Bool}(\mathcal{Q}_M)$ as is given by Lemma 2.5. Note that N_b has at most $2|Q_N|$ states. Furthermore, let $T_b = \mathsf{Bool}(T)$ as defined in Definition 4.16. Now, by Lemma 4.17, T_b is language-minimal. The following lemma relates N_b and T_b:

LEMMA 4.20. *1. $L(N_b) \subseteq L(T_b)$; and,*

2. $c_x(T_b) \in L(N_b)$.

Let ρ 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, $\mathsf{fold}^D(f_{c_x}^{-1}(n_1')) \neq \mathsf{fold}^D(f_{c_x}^{-1}(n_2'))$ for f_{c_x} the canonical embedding of T_b on $c_x(T_b)$.[3] Now, take two nodes n_1 and n_2 in T (or, equivalently, T_b) with $f_{c_x}(n_1) = n_1'$ and $f_{c_x}(n_2) = n_2'$. Since $\rho(n_1') = \rho(n_2')$, and since $L(N_b) \subseteq L(T_b)$ by Lemma 4.20, it follows that

$$c_x(T_b)[n_1' \leftarrow \mathsf{subtree}^{c_x(T_b)}(n_2')] \in L(T_b), \qquad (\dagger)$$

$$c_x(T_b)[n_2' \leftarrow \mathsf{subtree}^{c_x(T_b)}(n_1')] \in L(T_b). \qquad (\ddagger)$$

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

LEMMA 4.21. *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.

Before we prove the lemma, let us first explain how it leads to the desired contradiction. From Lemma 4.21(1), it follows that in D the nodes $\mathsf{fold}^D(n_1)$ and $\mathsf{fold}^D(n_2)$ can be *merged* to give a smaller (by at least one node) DAG-twig defining the same query as defined by D. Let $m_1 = \mathsf{fold}^D(n_1)$ and $m_2 = \mathsf{fold}^D(n_2)$. By assumption, $m_1 \neq m_2$. Furthermore, by Corollary 4.19 and Lemma 4.21(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. [4] Call the thus obtained DAG-Twig D'. Note that by Lemma 4.21(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 4.21(2) and Lemma 4.9(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 4.15. □

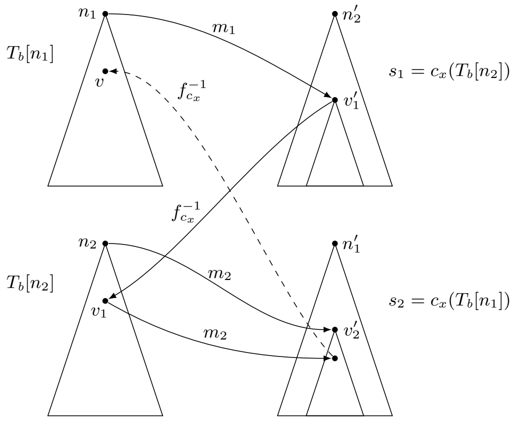


Figure 3: Illustration of the trees and associated nodes used in the proof of Lemma 4.21.

REMARK 4.22. *Note that for any $v \in \mathsf{Nodes}^t$,*

$$\mathsf{subtree}^{c_x(T)}(f_{c_x}(v)) = c_x(T[v]).$$

We now prove Lemma 4.21.

PROOF. Let

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

Then, $s_1 = c_x(T_b[n_2])$ and $s_2 = c_x(T_b[n_1])$, by Remark 4.22. To show that $L(T[n_1]) = L(T[n_2])$, we first apply Lemma 4.12 to (†) and (‡), to obtain nodes $v_1' \in \mathsf{Nodes}^{s_1}$ and $v_2' \in \mathsf{Nodes}^{s_2}$ such that

$$\mathsf{subtree}^{s_1}(v_1') \in L(T_b[n_1]),$$

$$\mathsf{subtree}^{s_2}(v_2') \in L(T_b[n_2]). \qquad (\star)$$

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 Figure 3.

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 $\mathsf{subtree}^{s_2}(v_2') = \mathsf{subtree}^{s_2}(n_1') = s_2 = c_x(T_b[n_1])$. Similarly, $\mathsf{subtree}^{s_1}(v_1') = c_x(T_b[n_2])$. Therefore, by ($\star$), $c_x(T_b[n_2]) \in L(T_b[n_1])$ and $c_x(T_b[n_1]) \in L(T_b[n_2])$, and by Lemma 4.7, $L(T_b[n_1]) = L(T_b[n_2])$ 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 $v_1 = f_{c_x}^{-1}(v_1')$ and $v_2 = f_{c_x}^{-1}(v_2')$, and let the mapping m_1 be the embedding showing that $\mathsf{subtree}^{s_1}(v_1') \in L(T_b[n_1])$ and m_2 the embedding showing $\mathsf{subtree}^{s_2}(v_2') \in L(T_b[n_2])$. Consider then the composition of mappings $m = m_2 \circ f_{c_x}^{-1} \circ m_1$. The mapping m is an embedding from $T_b[n_1]$ to $\mathsf{subtree}^{s_2}(m_2(v_1))$. 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 $\mathsf{subtree}^{s_2}(v_2') \in L(T_b[n_2])$. As we remarked above, $s_2 = c_x(T_b[n_1])$. So, $\mathsf{subtree}^{s_2}(m_2(v_1)) = c_x(T_b[v])$ where $v = f_{c_x}^{-1}(m_2(v_1))$, 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 4.18, this leads to a contradiction.

[3] Note that $f_{c_x}^{-1}$ maps nodes from $c_x(T_b)$ to T_b and $\mathsf{fold}^D()$ maps nodes from T to D, but since T_b and T contain the same set of nodes the composition of these two functions is well-defined.

[4] 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.

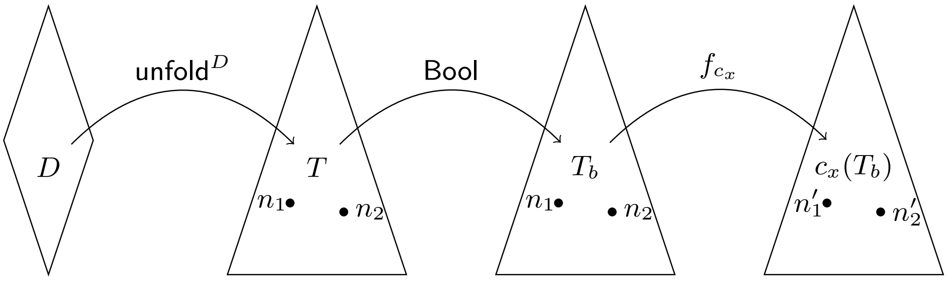


Figure 2: The DAG D, the unfolding of D into T, $T_b = \mathsf{Bool}(T)$, and the characteristic tree of T_b used in the proof of Theorem 4.15.

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 Bool. 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 4.19. 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. □

4.4 Testing equivalence of DAG-Twigs and NSTAs

Now we have a small model property of DAG-twigs compared to NSTAs (Theorem 4.15), we can simply decide twig-definability of an NSTA by guessing the DAG-twig and testing equivalence. Here, we argue that equivalence of such an NSTA 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 $\mathsf{Bool}(\mathcal{Q}_M)$ and $\mathsf{Bool}(\mathcal{Q}_D)$.

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 [24], for an (unranked) tree t, let $\mathsf{enc}(t)$ be its binary encoding, obtained as follows: The nodes of $\mathsf{enc}(t)$ are the nodes of t plus a set of leaf nodes marked #. Further, the root node of $\mathsf{enc}(t)$ is the root node of t and for any node, its left child in $\mathsf{enc}(t)$ is its first child in t (or # if its a leaf), and its right child in $\mathsf{enc}(t)$ is its next sibling in t (or # if it has none). In Figure 4, 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 4.23 (Alt. Tree-Walking Automata). Let $\mathrm{PosBool}(P)$ be the set of positive Boolean formulas over propositions P (i.e., formulas without negation), but including true and false. An *alternating tree walking automaton with wildcards* (ATWA with wildcards) over binary trees is defined as a tuple $W = (Q, \Delta, \Sigma, \delta, q_0)$, where

- Q is a finite set of states,
- Σ is a finite set of alphabet symbols,
- δ is a set of transition rules of the form $(q, \sigma) \to \theta$, where $q \in Q, \sigma \in \Sigma_\diamond$, and θ is a formula from
$$\mathrm{PosBool}(\{\swarrow, \searrow -\} \times Q),$$
and
- q_0 is the initial state.

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

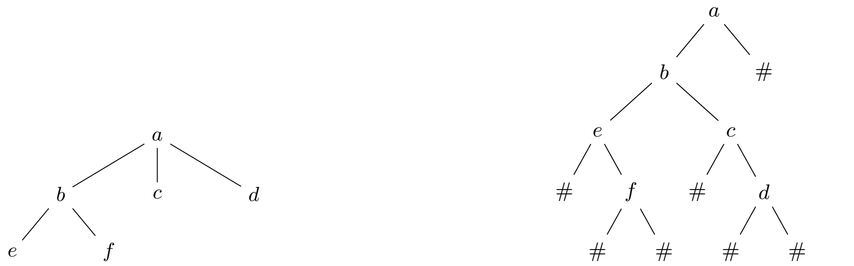
The transition relation δ should be such that for each pair $(q, \sigma) \in Q \times \Sigma_\diamond$ there is at most one rule in δ with (q, σ) as its left hand side. (If there would be two rules with the same left hand side, we can merge them into one rule by taking the disjunction of the right hand sides.) If $(q, \sigma) \to \theta \in \delta$, we also write $\mathrm{rhs}_W(q, \sigma) = \theta$. 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 $\mathrm{lab}(u) \neq \#$ 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 $\mathsf{Nodes}^t \times Q$ 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 $(\mathsf{root}(t), q_0)$ and
- for every node x of R with label (v, q_v), if $(q_v, \sigma) \to \theta \in \delta$ and $\mathrm{lab}^t(v)$ matches σ, 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 θ.

A run tree R is *accepting* if, for every leaf node of R labelled (u, q), there is a rule $\mathrm{rhs}_W(q, \sigma) = \mathsf{true}$ such that $\mathrm{lab}^t(u)$ matches σ. 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



(a) Unranked tree t.

(b) Its binary encoding $\mathsf{enc}(t)$.

Figure 4: An unranked tree and its binary encoding.

automata. Benedikt, Fan and Geerts [1] 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 [33] have shown that it is possible to construct, in quadratic time, a two-way alternating tree automaton equivalent to a given XPath query.

Lemma 4.24. *Let D be a DAG-twig representing the twig T. An alternating tree walking automaton W with $L(W) = \{\mathit{enc}(t) \mid t \in L(T)\}$ can be constructed in time $O(|D|)$.*

The construction for proving Lemma 4.24 can be easily changed such that W accepts encodings of $L(\mathsf{Bool}(T))$ instead of $L(T)$.

Corollary 4.25. *Let D be a DAG-twig that represents the twig T. An alternating tree walking automaton W with $L(W) = \{\mathit{enc}(t) \mid t \in L(\mathit{Bool}(T))\}$ can be constructed in time $O(|D|)$.*

We now reduce equivalence between an NSTA and a DAG-twig to the emptiness problem for ATWAs.

Theorem 4.26. *Given a DAG-twig D and an NSTA M, we can construct an ATWA W in polynomial time such that $L(W) = \emptyset$ if and only if $\mathit{Bool}(\mathcal{Q}_M) = \mathit{Bool}(\mathcal{Q}_D)$.*

Proof. Let $D_{0,1} := \mathsf{Bool}(D)$. Let $N_{0,1}$ be the NTA with $L(N_{0,1}) = \mathsf{Bool}(\mathcal{Q}_M)$, as obtained in Lemma 2.5. 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:

$$(\mathsf{root}(t), q_0) \to$$
$$\big((-, q_0^{D_{0,1}}) \wedge (-, \overline{q_0^{N_{0,1}}})\big) \vee \big((-, \overline{q_0^{D_{0,1}}}) \wedge (-, q_0^{N_{0,1}})\big)$$

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 [12], chapter 7), W can be constructed in polynomial time as well. □

Theorem 4.27. *Testing equivalence between a DAG-twig D and an NSTA M is EXPTIME-complete.*

Proof. The lower bound is immediate by a reduction from the language universality problem for NTAs. The upper bound is immediate from Theorem 4.26. The lower bound follows from the fact that testing language emptiness for alternating tree walking automata with wildcards is the same as language emptiness for alternating tree walking automata without wildcards. The latter problem is known to be in EXPTIME. (see, e.g., [5, 12]). □

4.5 Main Result

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

Theorem 4.28. *Deciding whether for an NSTA M, $\mathcal{Q}_M$ is twig-definable, is complete for EXPTIME.*

Proof. For the lower bound, similarly to Theorem 3.7, we will reduce the problem of universality of NTAs with wildcards to the problem considered here. Let $N = (\Delta, \Sigma, Q, \delta, F)$ be an NTA. We construct an NSTA M such that $\mathcal{Q}_M$ is twig-definable if and only if $L(N) = \mathcal{T}_\Delta$. Let $q_{\text{sel}} \notin Q$ and define the NSTA $M = ((\Delta, \Sigma, Q \uplus \{q_{\text{sel}}\}, \delta \uplus \{(q_{\text{sel}}, a) \mapsto (\sum_{p \in F} p)^*\}), \{q_{\text{sel}}\})$. Then for any tree $t' = a(t_1, \ldots, t_n)$, we have that $\mathsf{root}(t') \in \mathcal{Q}_M(t')$ 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 $t_{\text{small}} = \sigma$, consisting of just one node. However, by definition of twig queries, there is only one twig T that is able to select the root of t_{small}, namely the twig $T = (t, o, \text{Anc})$ with $t = \sigma$, $o = \mathsf{root}(t)$, and $\text{Anc} = \emptyset$. This means that $\mathcal{Q}_M$ 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 4.15, we know that if there exists a DAG-twig equivalent to $M = ((\Delta, \Sigma, Q, \delta, F), S)$, there is one that has at most $2 \cdot |Q|$ nodes. Therefore, we can enumerate every possible DAG-twig D with at most $2 \cdot |Q|$ nodes and test whether D and M are equivalent. Theorem 4.27 states that we can test in exponential time whether a given DAG-twig D and a given NSTA 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. □

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 [15] for single-type EDTDs. Of course, other languages than XPath can be considered, like for instance, the Region Algebra [13], caterpillar expressions [16], or even tree-walking automata [5].

6. REFERENCES

[1] M. Benedikt, W. Fan, and F. Geerts. XPath satisfiability in the presence of DTDs. *J. ACM*, 55(2), 2008.

[2] M. Benedikt, W. Fan, and G. M. Kuper. Structural properties of XPath fragments. *Theor. Comput. Sci.*, 336(1):3–31, 2005.

[3] M. Benedikt and C. Koch. XPath leashed. *ACM Comput. Surv.*, 41(1), 2008.

[4] M. Benedikt and L. Segoufin. Regular tree languages definable in FO and in FO_{mod}. *ACM Trans. Comput. Log.*, 11(1), 2009.

[5] M. Bojańczyk. Tree-walking automata. In *Int. Conf. on Language and Automata Theory and Applications (LATA)*, pages 1–2, 2008.

[6] M. Bojańczyk, A. Muscholl, T. Schwentick, and L. Segoufin. Two-variable logic on data trees and XML reasoning. *J. ACM*, 56(3), 2009.

[7] M. Bojańczyk and P. Parys. XPath evaluation in linear time. *J. ACM*, To appear.

[8] M. Bojańczyk and L. Segoufin. Tree languages defined in first-order logic with one quantifier alternation. *Logical Methods in Computer Science*, 6(4), 2010.

[9] M. Bojańczyk and I. Walukiewicz. Characterizing EF and EX tree logics. *Theor. Comput. Sci.*, 358(2–3):255–272, 2006.

[10] D. Calvanese, G. D. Giacomo, M. Lenzerini, and M. Y. Vardi. Node selection query languages for trees. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2010.

[11] J. Carme, J. Niehren, and M. Tommasi. Querying unranked trees with stepwise tree automata. In *International Conference on Rewriting Techniques and Applications (RTA)*, pages 105–118, 2004.

[12] H. Comon, M. Dauchet, R. Gilleron, F. Jacquemard, D. Lugiez, C. Löding, S. Tison, and M. Tommasi. *Tree Automata Techniques and Applications*. Available on http://tata.gforge.inria.fr/, 2007.

[13] M. P. Consens and T. Milo. Algebras for querying text regions: Expressive power and optimization. *J. Comput. Syst. Sci.*, 57(3):272–288, 1998.

[14] M. Frick, M. Grohe, and C. Koch. Query evaluation on compressed trees (extended abstract). In *IEEE Symposium on Logic in Computer Science (LICS)*, pages 188–, 2003.

[15] W. Gelade, T. Idziaszek, W. Martens, and F. Neven. Simplifying XML Schema: single-type approximations of regular tree languages. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 251–260, 2010.

[16] E. Goris and M. Marx. Looping caterpillars. In *Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 51–60, 2005.

[17] G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, 2004.

[18] G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. *ACM Trans. Database Syst.*, 30(2):444–491, 2005.

[19] G. Gottlob, C. Koch, R. Pichler, and L. Segoufin. The complexity of xpath query evaluation and XML typing. *J. ACM*, 52(2):284–335, 2005.

[20] W. Martens, F. Neven, T. Schwentick, and G. J. Bex. Expressiveness and complexity of XML Schema. *ACM Trans. Database Syst.*, 31(3):770–813, 2006.

[21] M. Marx. Conditional XPath. *ACM Trans. Database Syst.*, 30(4):929–959, 2005.

[22] G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *J. ACM*, 51(1):2–45, 2004.

[23] F. Neven. *Design and Analysis of Query Languages for Structured Documents.* PhD thesis, Limburgs Universitair Centrum, 1999.

[24] F. Neven. Automata theory for XML researchers. *SIGMOD Record*, 31(3):39–46, 2002.

[25] F. Neven. Attribute grammars for unranked trees as a query language for structured documents. *J. Comput. Syst. Sci.*, 70(2):221–257, 2005.

[26] F. Neven and J. V. den Bussche. Expressiveness of structured document query languages based on attribute grammars. *J. ACM*, 49(1):56–100, 2002.

[27] F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. In *ACM Symposium on Principles of Database Systems (PODS)*, pages 145–156, 2000.

[28] F. Neven and T. Schwentick. Query automata over finite trees. *Theor. Comput. Sci.*, 275(1-2):633–674, 2002.

[29] F. Neven and T. Schwentick. On the complexity of XPath containment in the presence of disjunction, DTDs, and variables. *Logical Methods in Computer Science*, 2(3), 2006.

[30] T. Place and L. Segoufin. Deciding definability in $\mathrm{FO}_2(<)$ (or XPath) on trees. In *Annual IEEE Symposium on Logic in Computer Science (LICS)*, pages 253–262, 2010.

[31] T. Schwentick. XPath query containment. *SIGMOD Record*, 33(1):101–109, 2004.

[32] H. Seidl. Deciding equivalence of finite tree automata. *SIAM J. Comput.*, 19(3):424–437, 1990.

[33] B. ten Cate and C. Lutz. The complexity of query containment in expressive fragments of XPath 2.0. *J. ACM*, 56(6), 2009.

[34] B. ten Cate and L. Segoufin. Transitive closure logic, nested tree walking automata, and XPath. *J. ACM*, 57(3), 2010.

[35] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 7. Springer, 1997.