Journal of Applied Logic 8 (2010) 210–232

Contents lists available at ScienceDirect

Journal of Applied Logic

www.elsevier.com/locate/jal

Reasoning about XML with temporal logics and automata

Leonid Libkin [a,*], Cristina Sirangelo [b]

[a] University of Edinburgh, United Kingdom
[b] LSV, ENS-Cachan, CNRS, INRIA, France

ARTICLE INFO

Article history:
Available online 22 September 2009

Keywords:
Query automata
Static analysis
Temporal logics
XML

ABSTRACT

We show that problems arising in static analysis of XML specifications and transformations can be dealt with using techniques similar to those developed for static analysis of programs. Many properties of interest in the XML context are related to navigation, and can be formulated in temporal logics for trees. We choose a logic that admits a simple single-exponential translation into unranked tree automata, in the spirit of the classical LTL-to-Büchi automata translation. Automata arising from this translation have a number of additional properties; in particular, they are convenient for reasoning about unary node-selecting queries, which are important in the XML context. We give two applications of such reasoning: one deals with a classical XML problem of reasoning about navigation in the presence of schemas, and the other relates to verifying security properties of XML views.

1. Introduction

Static analysis of XML specifications and transformations has been the focus of many recent papers. Typical problems include consistency of type declarations and constraints [1,5,25], or of schema specifications and navigational properties [7,18] or containment of XPath expressions [9,11,17,27,37]. They found application in query optimization, access control, data exchange, and reasoning about security properties of views, among others.

There is an analogy, at least at the level to tools and techniques, between many of XML static analysis problems and those arising in software verification. Specifications of program behavior are often expressed in temporal logic formalisms, while the programs themselves are abstracted as labeled transitions systems or Kripke structures, and thus can be viewed as automata. To reason about programs, logical specifications are turned into automata, and then verification problems are reduced to pure automata questions such as 'is there an accepting run of a given automaton?'.

When we turn to XML, we see both the ingredients. First, many XML specifications – for example, various schema formalisms – are automata-based. For example, DTDs are extended context free grammars, and extended DTDs (which add a notion of specialization to DTDs) have precisely the power of tree automata. Many other formalisms have automata-theoretic flavor, see [26] for a survey. Furthermore, there is a close connection between XML navigation (e.g., the language XPath), which is a key component of query languages, and temporal logics [6,28,27,23,17,18]. Thus, it is very natural to adapt automata-based techniques developed by the verification community (cf. [12]) to XML static analysis problems involving schemas and navigation.

This idea has been explored in the past, mainly by adapting existing verification tools, and reshaping the problem at hand so that those tools would be applicable to it. For example, [27] shows how to reason about XPath and XML schemas by encoding them in PDL (propositional dynamic logic). The problem is, given an input DTD d and Xpath expressions e_1 and

* Corresponding author.
E-mail addresses: libkin@inf.ed.ac.uk (L. Libkin), cristina.sirangelo@lsv.ens-cachan.fr (C. Sirangelo).

1570-8683/$ – see front matter © 2009 Elsevier B.V. All rights reserved.
doi:10.1016/j.jal.2009.09.005

e_2, to check whether containment $e_1(T) \subseteq e_2(T)$ holds for all trees T conforming to d. While the approach of [27] achieves a provably optimal EXPTIME bound, it does so by a rather complicated algorithm. For example, it uses, as a black box for one of its steps, a translation from PDL into a certain type of tree automata [43], for which no efficient implementations exist. Another example of such reasoning [17,18], for the same containment problem, goes via a better implementable μ-calculus, and achieves a similar $2^{O(n)}$ bound.

We propose an alternative approach: instead of using verification techniques as-is in the XML context, we adapt them to get better static analysis algorithms. The present paper can be viewed as a proof-of-concept paper: we demonstrate one logic-to-automata translation targeted to XML applications, which closely resembles the standard Vardi–Wolper's translation [42] of LTL (linear temporal logic) into nondeterministic Büchi automata, and show that it is easily applicable in some typical XML reasoning tasks.

Typically, temporal logic formulae are translated into either nondeterministic or alternating automata; for LTL, both are possible [42,40]. We believe that both should be explored in the XML context. For this paper, we concentrate on the former. A recent paper [11] developed an alternating-automata based approach along the lines of [40]. It handled a more expressive navigation language, but did not work out connections with XML schemas, as we do here.

Our goal is to find a clean direct translation from a logical formalism suitable for expressing many XML reasoning tasks, into an automata model. Towards that end, we use a simple LTL-like logic for trees, which we call $\mathrm{TL}^{\mathrm{tree}}$, rather than a W3C-designed language (but we shall show that such languages can be easily translated into $\mathrm{TL}^{\mathrm{tree}}$). This logic was defined in [36], and it was recently used in the work on XPath extensions [28], and as a key ingredient for an expressively-complete logic for reasoning about procedural programs [3,4]. It should be noted that we do not propose to use $\mathrm{TL}^{\mathrm{tree}}$ for writing specifications; rather, we view it as a convenient intermediate step as constantly changing standards and languages such as XPath. Those languages can be easily encoded in $\mathrm{TL}^{\mathrm{tree}}$, and the translation from $\mathrm{TL}^{\mathrm{tree}}$ into automata follows the lines of well established translations from temporal logics into automata.

The translation that we exhibit produces a bit more than automata rejecting or accepting trees; instead it will produce *query automata* [32,30,16] which can also select nodes from trees in their successful runs. The ability to produce such automata is not surprising at all (since in the Vardi–Wolper construction states are sets of formulae and successful runs tell us which formulae hold in which positions). Nonetheless, it is a useful feature for XML reasoning, since many XML data processing tasks are about *node-selecting queries* [20,32,38,31]. Furthermore, additional properties of query automata arising in the translation make operations such as complementation and testing containment very easy.

1.1. Main contributions

We present a single-exponential translation from the logic $\mathrm{TL}^{\mathrm{tree}}$ into unranked tree automata (in fact, into query automata with additional properties). We then show how to translate various flavors of XPath into $\mathrm{TL}^{\mathrm{tree}}$ (most of the time, linearly, but even in more complex cases still guaranteeing the overall single-exponential bound on the translation XPath $\to \mathrm{TL}^{\mathrm{tree}} \to$ automata). Furthermore we give two applications of the translations to XML reasoning, involving satisfiability/containment of XPath in the presence of schema information, and reasoning about XML views.

1.2. Organization

In Section 2 we give examples of XML reasoning where the logic/automata connection would be useful. Section 3 describes unranked trees and automata for unranked trees. In Section 4 we present the logic $\mathrm{TL}^{\mathrm{tree}}$ and various XPath formalisms, and in Section 5 we give an easy translation of XPath into $\mathrm{TL}^{\mathrm{tree}}$. In Section 6 we give a translation from $\mathrm{TL}^{\mathrm{tree}}$ to query automata. Section 7 applies this translation in complex reasoning tasks involving schemas and navigation in XML documents and to reasoning about XML views.

An extended abstract of this work has appeared in LPAR 2008 conference proceedings [24].

2. Motivating examples

We now consider two examples of XML static analysis problems that will later be handled by restating these problems with the help of $\mathrm{TL}^{\mathrm{tree}}$ and the automata translation. While formal definitions will be given later, for the reader not fluent in XML the following abstractions will be sufficient. First, XML documents themselves are labeled unranked trees (that is, different nodes can have a different number of children). XML schemas describe how documents are structured; we abstract them for now by means of tree automata. The most common of such formalisms is referred to as DTDs (document type definitions). And finally XPath is a navigational language; an XPath expression for now can be thought of as selecting a set of nodes in a tree.

2.1. Reasoning about schemas and navigation

A common static analysis problem in the XML context, arising in query optimization and consistency checking, is the interaction of navigational properties (expressed, for example, in XPath) with schemas (often given as DTDs). Examples of such problems are XPath containment [37] or XPath/DTD consistency [7].

The *containment problem* of XPath expressions under a DTD, is the problem of checking whether for all trees satisfying a DTD d, the set of nodes selected by an expression e_1 is contained in the set selected by e_2 (written as $d \models e_1 \subseteq e_2$). The *XPath/DTD consistency problem* is to check whether there exists a tree satisfying a given DTD where the set of nodes selected by a given Xpath expression is non-empty.

Known results about the complexity of such problems are typically stated in terms of completeness for various intractable complexity classes. They imply unavoidability of exponential-time algorithms, and they do not necessarily lead to reasonable algorithms that can be used in practice.

To illustrate this, consider the containment problem of XPath expressions under a DTD. For example, consider two expressions $e_1 = r//b$ and $e_2 = r/a/b$ (we assume that r refers to the root of a tree). The first expression selects all nodes (descendants of the root, which is expressed by $r//$) labeled b. The second expression selects b-labeled grandchildren of the root whose parent is labeled a. In general we have $e_2 \subseteq e_1$ but $e_1 \subseteq e_2$ does *not* hold. However, if we have a DTD d with the rules $r \to a^*$; $a \to b^*$; $b \to \varepsilon$ saying that all children of r must be labeled a and all children of a-labeled nodes must be labeled b, then we have the reverse inclusion as well, i.e., $d \models e_1 \subseteq e_2$.

To verify containment, one could use automata-based algorithms that translate XPath directly into automata (which could depend heavily on a particular syntactic class [33]). Alternatively, one could attempt a translation via an existing logic. This is the approach of [27,17] which translate e_1, e_2, and d into formulae of expressive logics such as PDL (in [27]) or μ-calculus (in [17,18]). Then one uses techniques of [43,41] to check if there exists a finite tree T satisfying d and a node s in T witnessing $e_1(s) \wedge \neg e_2(s)$, i.e., a counterexample to the containment.

While this is very much in the spirit of the traditional logic/automata connection used so extensively in static analysis of programs, there are some problems with this approach as currently used. The logics used were chosen because of their ability to encode DTDs, but this makes the constructions apply several algorithms as black-boxes. For example, the PDL approach of [27] combines three different constructions: one is a translation into PDL with converse on binary trees; another one is an algorithm of [43] translating PDL into a rather complex automata model; and a third one is a product construction with an extra automaton that restricts the produced automaton to finite trees. Another limitation of the above approaches to verify containment is that we do not get a concise description of the set of all possible counterexamples, rather a yes or no answer. Finally, the high expressiveness of logics comes at a cost. The running time of algorithms that go via μ-calculus or PDL is $2^{O(\|e_1\|+\|e_2\|+\|d\|)}$, where $\|\cdot\|$ denotes the size [27,18]. In several applications, we would rather avoid the $2^{O(\|d\|)}$ factor, since many DTDs are computer-generated from database schemas and could be very large, while XPath expressions tend to be small.

The translation we propose is a direct and simple construction (following the lines of Vardi–Wolper's translation), and does not rely on complicated algorithms such as the PDL-to-automata translation (which are unlikely to be implementable). It produces a concise description of *all* possible counterexamples. Finally, it exhibits an exponential blowup in the size of e_1 and e_2, but remains polynomial in the size of the schema.

To illustrate it, we revisit the example with the DTD d given by $r \to a^*$; $a \to b^*$; $b \to \varepsilon$ and expressions $e_1 = r//b$ and $e_2 = r/a/b$. We shall translate XPath expressions into temporal logic formulae:

$$\psi_{e_1} = b$$
$$\psi_{e_2} = b \wedge \mathbf{X}^-_{\mathrm{ch}}\big(a \wedge \mathbf{X}^-_{\mathrm{ch}} r\big).$$

The connective $\mathbf{X}^-_{\mathrm{ch}}$ means that a formula is true in the parent of a given node; that is, $b \wedge \mathbf{X}^-_{\mathrm{ch}}(a \wedge \mathbf{X}^-_{\mathrm{ch}} r)$ is true in a node if it is labeled by b, its parent by a, and its grandparent by r.

We then define $\varphi = \psi_{e_1} \wedge \neg\psi_{e_2}$ which describes *counterexamples* to containment. We modify it so that all negations apply only to labels:

$$\varphi = b \wedge \mathbf{X}^-_{\mathrm{ch}}\big(\neg a \vee \mathbf{X}^-_{\mathrm{ch}} \neg r\big).$$

We then follow a standard approach by translating this formula into an automaton whose states are sets of subformulae of φ, i.e., subsets of

$$\neg a, \quad \neg r, \quad b, \quad \mathbf{X}^-_{\mathrm{ch}}\neg r, \quad \neg a \vee \mathbf{X}^-_{\mathrm{ch}}\neg r, \quad \mathbf{X}^-_{\mathrm{ch}}\big(\neg a \vee \mathbf{X}^-_{\mathrm{ch}}\neg r\big), \quad \varphi.$$

The resulting automaton would accept some trees of course, because in general the containment $e_1 \subseteq e_2$ does not hold. But we then take the *product* of this automaton with the automaton that captures the DTD. The resulting automaton accepts counterexamples to containment *under the DTD*. In our example, this automaton would be empty.

Informally (and this will be made precise when we describe the translation), the product automaton is of size exponential in e_1 and e_2 (as in [27,18]) and linear in the size of d (unlike in [27,18]). Since testing for emptiness can be done in polynomial time, the construction removes an exponential factor $2^{O(\|d\|)}$.

The example shown above outlines the main elements of the construction we present here. The temporal formulae come from the logic $\mathrm{TL}^{\mathrm{tree}}$. We present two main technical devices in this paper:

1. A single-exponential translation from $\mathrm{TL}^{\mathrm{tree}}$ formulae to automata; and
2. A single-exponential, but often linear, translation from XPath to $\mathrm{TL}^{\mathrm{tree}}$.

While this *might* appear to lead to a double-exponential translation overall, we also show that the XPath-to-$\mathrm{TL}^{\mathrm{tree}}$ is done in a way that does not increase the number of subformulae. And since the states of the automaton are sets of subformulae, the overall translation is single-exponential.

Our approach can deal with more general problems than XPath containment under DTDs. Generalizations of this problem will be discussed in Section 7.1. In particular the ability to manipulate $\mathrm{TL}^{\mathrm{tree}}$ formulae also gives us algorithms for complex containment/equivalence conditions; the ability to take products lets us incorporate schema information in a way that avoids an exponential blowup in the size of the schema.

2.2. Reasoning about views and query answers

Often the user sees not a whole XML document, but just a portion of it, V (called a *view*), generated by a query. Such a query typically specifies a set of nodes selected from a source document, and thus can be represented by a query automaton $\mathcal{QA}_{\mathcal{V}}$: i.e., an extension of a tree automaton that can select nodes in trees; a formal definition will be given shortly.

If we only have access to V, we do not know the source document that produced it, as there could be many trees T satisfying $V = \mathcal{QA}_{\mathcal{V}}(T)$. We may know, however, that every such source has to satisfy some schema requirements, presented by a tree automaton $\mathcal{A}$. Moreover we may know that there is some particular information about the source that is considered a "secret", and therefore should not be available.

A common problem is to check whether V may reveal this "secret" information about the source. If $\mathfrak{Q}$ is a Boolean (yes/no) query, one defines the *certain answer* to $\mathfrak{Q}$ over V to be true iff $\mathfrak{Q}$ is true in every possible T that generates V:

$$\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_{\mathcal{V}}}(\mathfrak{Q}; V) = \bigwedge \bigl\{\mathfrak{Q}(T) \bigm| V = \mathcal{QA}_{\mathcal{V}}(T), T \text{ is accepted by } \mathcal{A}\bigr\}.$$

Now if by looking at V, we can conclude that $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_{\mathcal{V}}}(\mathfrak{Q}; V)$ is true, then V reveals that $\mathfrak{Q}$ is true in an unknown source. If $\mathfrak{Q}$ is a containment statement $e_1 \subseteq e_2$, such an inclusion could be information that needs to be kept secret (e.g., it may relate two different groups of people). For more on this type of applications, see [15,14].

For example, assume that the source document (that is unknown to us) conforms to the DTD d with the rules $r \to a^*, c^*$; $a \to b^*, d^*$; and $c^* \to f^*, b^*$ (plus $b, d, f \to \varepsilon$). Suppose that the view simply selects all the b-labeled nodes, and together with each node it selects the full path from the root to the node.

Next, assume that the 'secret' query $\mathfrak{Q}$ is the containment $r//b \subseteq r//a/b$. It says that every b node must be a child of an a node. If a and b talk about groups of people, products, etc., information of this kind may be important to hide. If we simply look at the DTD and the query, we cannot derive this fact, as some b's appear on the $r.c.b$ path. It appears that we need access to the document itself.

However, if we have access just to the view and not the source document, we may positively answer the query in some cases. For instance, if in the view every b node is a child of an a node, then we can derive, from the view and the schema information, that the same is true in the source, even if we do not know it.

In general, assume that the Boolean query $\mathfrak{Q}$ is definable by an automaton $\mathcal{A}_{\mathfrak{Q}}$. Our approach to computing certain answers is as follows. We attempt to convert automata $\mathcal{A}_{\mathfrak{Q}}$, $\mathcal{A}$, and the query automaton $\mathcal{QA}_{\mathcal{V}}$ into a new automaton $\mathcal{A}^*$ so that $\mathcal{A}^*$ accepts V iff $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_{\mathcal{V}}}(\mathfrak{Q}; V)$ is false. Then acceptance by $\mathcal{A}^*$ gives us some assurances that the secret is not revealed. Furthermore, since views are often given by XPath expressions, and e_1 and e_2 are often XPath expressions too, an efficient algorithm for constructing $\mathcal{A}^*$ would give us a verification algorithm exponential in (typically short) XPath expressions defining e_1, e_2, and $\mathcal{V}$, and polynomial in a (potentially large) expression defining the schema.

In fact, we shall present a polynomial-time construction for $\mathcal{A}^*$ for the case of views which are similar to the one we used in the example. Namely, such views are subtree- (or upward-closed): together with each node they select the whole path from the root to that node. Such queries have arisen in a number of applications in the XML context, see [2,8]. For them, using the previous efficient translations from logical formulae into query automata, we get efficient algorithms for verifying properties of views.

3. Unranked trees and automata

3.1. Unranked trees

XML documents are normally abstracted as labeled unranked trees (we disregard data values as well as references such as ID and IDREF that lead to more general graph structures). We now recall the classical definitions, see [31,23,38].

Nodes in unranked trees are elements of $\mathbb{N}^*$, i.e. strings of natural numbers. We write $s \cdot s'$ for the concatenation of strings, and ε for the empty string. The basic binary relations on $\mathbb{N}^*$ are:

- the *child relation*: $s \prec_{\mathrm{ch}} s'$ if $s' = s \cdot i$, for some $i \in \mathbb{N}$, and
- the *next-sibling* relation: $s' \prec_{\mathrm{ns}} s''$ if $s' = s \cdot i$ and $s'' = s \cdot (i + 1)$ for some $s \in \mathbb{N}^*$ and $i \in \mathbb{N}$.

The *descendant* relation $\prec^*_{\mathrm{ch}}$ and the younger sibling relation $\prec^*_{\mathrm{ns}}$ are the reflexive-transitive closures of $\prec_{\mathrm{ch}}$ and $\prec_{\mathrm{ns}}$.

An *unranked tree domain* D is a finite prefix-closed subset of $\mathbb{N}^*$ such that $s \cdot i \in D$ implies $s \cdot j \in D$ for all $j < i$. If Σ is a finite alphabet, a Σ-labeled *unranked tree* is a pair $T = (D, \lambda)$, where D is a tree domain and λ is a labeling function $\lambda : D \to \Sigma$.

An unranked tree $T = (D, \lambda)$ can be viewed as a structure $\langle D, \prec^*_{\text{ch}}, \prec^*_{\text{ns}}, (P_a)_{a \in \Sigma} \rangle$, where P_a's are labeling predicates: $P_a = \{s \in D \mid \lambda(s) = a\}$. Thus, when we talk about *first-order logic* (FO), or *monadic second-order logic* (MSO), we interpret them on these representations of unranked trees. Recall that MSO extends FO with quantification over sets.

In what follows we will often refer to unranked trees simply as trees.

3.2. Unranked tree automata and XML schemas

A *nondeterministic unranked tree automaton* (cf. [31,38]) over Σ-labeled trees is a triple $\mathcal{A} = (Q, F, \delta)$ where Q is a finite set of states, $F \subseteq Q$ is the set of final states, and δ is a mapping $Q \times \Sigma \to 2^{Q^*}$ such that each $\delta(q, a)$ is a regular language over Q. We assume that each $\delta(q, a)$ is given as an NFA (nondeterministic finite automaton). A *run* of $\mathcal{A}$ on a tree $T = (D, \lambda)$ is a function $\rho_{\mathcal{A}} : D \to Q$ such that if $s \in D$ is a node with n children, and $\lambda(s) = a$, then the string $\rho_{\mathcal{A}}(s \cdot 0) \cdots \rho_{\mathcal{A}}(s \cdot (n-1))$ is in $\delta(\rho_{\mathcal{A}}(s), a)$. Thus, if s is a leaf labeled a, then $\rho_{\mathcal{A}}(s) = q$ implies that $\varepsilon \in \delta(q, a)$. A run is *accepting* if $\rho_{\mathcal{A}}(\varepsilon) \in F$, and a tree is *accepted* by $\mathcal{A}$ if an accepting run exists. Sets of trees accepted by automata $\mathcal{A}$ are called regular and denoted by $L(\mathcal{A})$.

There are multiple notions of *schemas* for XML documents. What is common for such notions is that their structural aspects are subsumed by unranked tree automata, see [26] for several examples. More, translations from various schema formalisms into automata are usually very effective [26], and thus automata are naturally viewed as an abstraction of schemas in the XML literature. Among such schema formalisms, DTDs (i.e., extended context-free grammars) are most commonly used. So when we speak of XML schemas, we shall assume that they are given by unranked tree automata.

As an example, we show a simple translation from DTDs into unranked tree automata. Suppose we have a DTD d over an alphabet $\Sigma = \{a_0, a_1, \ldots, a_n\}$ given by the set of rules $a_i \to e_i$, where each e_i is a regular expression over Σ. We assume that a_0 is the root. The states of the automaton $\mathcal{A}_d$ are $Q = \{q_{a_0}, \ldots, q_{a_n}\}$, the final state is q_{a_0}, and the transition function is

$$\delta(q_{a_i}, a_i) = q_{e_i} \quad \text{for all } i$$
$$\delta(q_{a_i}, a_j) = \emptyset \quad \text{for all } i \neq j.$$

Here q_{e_i} is obtained from e_i by replacing each a_l by q_{a_l}.

For instance, if we have a DTD d with the rules $\mathit{root} \to \mathit{book}^+$, $\mathit{book} \to \mathit{title}, \mathit{author}^*$ and $\mathit{title}, \mathit{author} \to \varepsilon$, then the automaton $\mathcal{A}_d$ would have states $q_{\mathit{root}}, q_{\mathit{book}}, q_{\mathit{author}}, q_{\mathit{title}}$, and the transitions

$$\delta(q_{\mathit{root}}, \mathit{root}) = q^+_{\mathit{book}}$$
$$\delta(q_{\mathit{book}}, \mathit{book}) = q_{\mathit{title}}, q^*_{\mathit{author}}$$
$$\delta(q_{\mathit{title}}, \mathit{title}) = \varepsilon$$
$$\delta(q_{\mathit{author}}, \mathit{author}) = \varepsilon;$$

with all other values of δ being $\emptyset$.

In what follows, the size $\|d\|$ of a DTD d refers to the size of the automaton $\mathcal{A}_d$ (under any standard encoding of tree automata).

3.3. Query automata

It is well known that automata capture the expressiveness of MSO (monadic second order logic) sentences over finite and infinite strings and trees [39]. The model of query automata [32] captures the expressiveness of MSO formulae $\varphi(x)$ with one free first-order variable – that is, MSO-definable unary queries. We present here a nondeterministic version, as in [30,16].

A *query automaton* (QA) for Σ-labeled unranked trees is a tuple $\mathcal{QA} = (Q, F, Q_s, \delta)$, where (Q, F, δ) is a nondeterministic unranked tree automaton, and $Q_s \subseteq Q$ is the set of *selecting states*. The runs of $\mathcal{QA}$ on a tree T are defined as the runs of (Q, F, δ) on T. Each run ρ of $\mathcal{QA}$ on a tree $T = (D, \lambda)$ defines the set $S_\rho(T) = \{s \in D \mid \rho(s) \in Q_s\}$ of nodes assigned a selecting state. The unary query defined by $\mathcal{QA}$ is then, under the *existential semantics*,

$$\mathcal{QA}^\exists(T) = \bigcup \{S_\rho(T) \mid \rho \text{ is an accepting run of } \mathcal{QA} \text{ on } T\}.$$

Alternatively, one can define $\mathcal{QA}^\forall(T)$ under the *universal semantics* as $\bigcap \{S_\rho(T) \mid \rho$ is an accepting run of $\mathcal{QA}$ on $T\}$. Both semantics capture the class of unary MSO queries [30].

For example, the automaton $\mathcal{A}_d$ for the DTD d from the previous section can be turned into a query automaton $\mathcal{QA}_{d,\mathit{book}}$ that selects *book* nodes from documents that conform to D simply by declaring q_{book} as the selecting state.

These semantics $\mathcal{QA}^{\exists}(T)$ and $\mathcal{QA}^{\forall}(T)$ are not very convenient for reasoning tasks, as many runs need to be taken into account – different nodes may be selected in different runs. Also, it makes operations on query automata hard computationally: for example, a natural notion of complement for an existential-semantics QA will be expressed as a universal semantics QA, requiring an exponential time algorithm to convert it back into an existential QA.

To remedy this, we define a notion of *single-run query automata* as QAs (Q, F, Q_s, δ) satisfying two conditions:

1. For every tree T, and accepting runs ρ_1 and ρ_2, we have $S_{\rho_1}(T) = S_{\rho_2}(T)$; and
2. The automaton (Q, F, δ) accepts every tree.

For such QAs, we can unambiguously define the set of selected nodes as $\mathcal{QA}(T) = S_\rho(T)$, where ρ is an arbitrarily chosen accepting run.

While the conditions are fairly strong, they do not restrict the power of QAs:

Fact 1. *(See [16,34,35].) For every query automaton $\mathcal{QA}$, there exists an equivalent single-run query automaton, that is, a single-run query automaton $\mathcal{QA}'$ such that $\mathcal{QA}^{\exists}(T) = \mathcal{QA}'(T)$ for every tree T.*

The construction in [16] needs a slight modification to produce such QA; also it needs to be extended to unranked trees which is straightforward. This was also noticed in [35]. One can also get this result by slightly adapting the construction of [34].

For example, to make the query automaton $\mathcal{QA}_{d,book}$ (that selects *book* nodes from documents conforming to d) single-run, we use the following trick: in the beginning the QA guesses whether the tree conforms to DTD or not. If it does, it attempts to run $\mathcal{QA}_{d,book}$, with q_{book} as the selecting state (in a way that it will not accept if the tree does not conform to d). If the guess is that the tree does not conform to d, it runs an automaton accepting the complement of d with no selecting states. This will satisfy the definition of single-run.

We now make a few remarks about closure properties and decision problems for single-run QAs. It is known [31] that non-emptiness problem for existential-semantics QAs is solvable in polynomial time; hence the same is true for single-run QAs. Single-run QAs are easily closed under intersection: the usual product construction works. Moreover, if one takes a product $\mathcal{A} \times \mathcal{QA}$ of a tree automaton and a single-run QA (where selecting states are pairs containing a selecting state of $\mathcal{QA}$), the result is a QA satisfying (1) above, and the non-emptiness problem for it is solvable in polynomial time too.

We define the *complement* of a single-run QA as $\overline{\mathcal{QA}} = (Q, F, Q - Q_s, \delta)$, where $\mathcal{QA} = (Q, F, Q_s, \delta)$. It follows immediately from the definition that for every tree T with domain D, we have $\overline{\mathcal{QA}}(T) = D - \mathcal{QA}(T)$, if $\mathcal{QA}$ is single-run. This implies that the *containment problem* $\mathcal{QA}_1 \subseteq \mathcal{QA}_2$ (i.e., checking whether $\mathcal{QA}_1(T) \subseteq \mathcal{QA}_2(T)$ for all T) for single-run QAs is solvable in polynomial time, since it is equivalent to checking emptiness of $\mathcal{QA}_1 \times \overline{\mathcal{QA}_2}$.

4. Logics on trees: TL$^{\text{tree}}$ and XPath

4.1. TL$^{\text{tree}}$

We shall use a *tree temporal logic* [28,36], denoted here by TL$^{\text{tree}}$ [23]. It can be viewed as a natural extension of LTL with the past operators to unranked trees [21,41], with *next*, *previous*, *until*, and *since* operators for both child and next-sibling relations. The syntax of TL$^{\text{tree}}$ is defined by:

$$\varphi, \varphi' := \top \mid \bot \mid a \mid \varphi \vee \varphi' \mid \neg\varphi \mid \mathbf{X}_*\varphi \mid \mathbf{X}_*^-\varphi \mid \varphi\mathbf{U}_*\varphi' \mid \varphi\mathbf{S}_*\varphi',$$

where $\top$ and $\bot$ are true and false, a ranges over Σ, and $*$ is either 'ch' (child) or 'ns' (next sibling). The semantics is defined with respect to a tree $T = (D, \lambda)$ and a node $s \in D$:

- $(T, s) \models \top$; $(T, s) \not\models \bot$;
- $(T, s) \models a$ iff $\lambda(s) = a$;
- $(T, s) \models \varphi \vee \varphi'$ iff $(T, s) \models \varphi$ or $(T, s) \models \varphi'$;
- $(T, s) \models \neg\varphi$ iff $(T, s) \not\models \varphi$;
- $(T, s) \models \mathbf{X}_{\text{ch}}\varphi$ if there exists a node $s' \in D$ such that $s \prec_{\text{ch}} s'$ and $(T, s') \models \varphi$;
- $(T, s) \models \mathbf{X}_{\text{ch}}^-\varphi$ if there exists a node $s' \in D$ such that $s' \prec_{\text{ch}} s$ and $(T, s') \models \varphi$;
- $(T, s) \models \varphi\mathbf{U}_{\text{ch}}\varphi'$ if there is a node s' such that $s \prec_{\text{ch}}^* s'$, $(T, s') \models \varphi'$, and for all $s'' \neq s'$ satisfying $s \prec_{\text{ch}}^* s'' \prec_{\text{ch}}^* s'$ we have $(T, s'') \models \varphi$;
- $(T, s) \models \varphi\mathbf{S}_{\text{ch}}\varphi'$ if there is a node s' such that $s' \prec_{\text{ch}}^* s$, $(T, s') \models \varphi'$, and for all $s'' \neq s'$ satisfying $s' \prec_{\text{ch}}^* s'' \prec_{\text{ch}}^* s$ we have $(T, s'') \models \varphi$.

The semantics of $\mathbf{X}_{\text{ns}}$, $\mathbf{X}_{\text{ns}}^-$, $\mathbf{U}_{\text{ns}}$, and $\mathbf{S}_{\text{ns}}$ is analogous by replacing the child relation with the next-sibling relation.

We shall also use the standard abbreviations: $\mathbf{F}_{\text{ch}}\varphi$ is $\top\mathbf{U}_{\text{ch}}\varphi$ (there is a descendant where φ is true), $\mathbf{F}_{\text{ch}}^-\varphi$ is $\top\mathbf{S}_{\text{ch}}\varphi$ (there is an ancestor where φ) is true; and likewise for $\mathbf{F}_{\text{ns}}$ and $\mathbf{F}_{\text{ns}}^-$. We also use *root* as a shorthand for $\neg\mathbf{X}_{\text{ch}}^-\top$.

A $\mathrm{TL}^{\mathrm{tree}}$ formula φ defines a unary query $T \mapsto \{s \mid (T,s) \models \varphi\}$. It is known that $\mathrm{TL}^{\mathrm{tree}}$ is expressively complete for FO: the class of such unary queries is precisely the class of queries defined by FO formulae with one free variable [28,36].

4.2. XPath

We present a first-order complete extension of XPath, called *conditional XPath*, or CXPath [28]. We introduce very minor modifications to the syntax (e.g., we use an existential quantifier **E** instead of the usual XPath node test brackets [ ]) to make the syntax resemble that of temporal logics. CXPath has *node formulae* α and *path formulae* β given by:

$$\alpha, \alpha' := a \mid \neg\alpha \mid \alpha \vee \alpha' \mid \mathbf{E}\beta$$
$$\beta, \beta' := ?\alpha \mid \mathtt{step} \mid \mathtt{step}^* \mid (\mathtt{step}/?\alpha)^* \mid \beta/\beta' \mid \beta \vee \beta'$$

where a ranges over Σ and $\mathtt{step}$ is one of the following: $\prec_{\mathrm{ch}}$, $\prec_{\mathrm{ch}}^{-}$, $\prec_{\mathrm{ns}}$, or $\prec_{\mathrm{ns}}^{-}$. The language without the $(\mathtt{step}/?\alpha)^*$ is known as “core XPath”.

Intuitively $\mathbf{E}\beta$ states the existence of a path starting in a given node and satisfying β, the path formula $?\alpha$ tests if the node formula α is true in the initial node of a path, and / is the composition of paths.

Given a tree $T = (D, \lambda)$, the semantics of a node formula is a set of nodes $[\![\alpha]\!]_T \subseteq D$, and the semantics of a path formula is a binary relation $[\![\beta]\!]_T \subseteq D \times D$ given by the following rules. We use R^* to denote the reflexive-transitive closure of relation R, and $\pi_1(R)$ to denote its first projection.

$$\begin{array}{ll}
[\![a]\!]_T = \{s \in D \mid \lambda(s) = a\} & [\![?\alpha]\!]_T = \{(s,s) \mid s \in [\![\alpha]\!]_T\} \\
[\![\neg\alpha]\!]_T = D - [\![\alpha]\!]_T & [\![\mathtt{step}]\!]_T = \{(s,s') \mid s, s' \in D \text{ and } (s,s') \in \mathtt{step}\} \\
[\![\alpha \vee \alpha']\!]_T = [\![\alpha]\!]_T \cup [\![\alpha']\!]_T & [\![\beta \vee \beta']\!]_T = [\![\beta]\!]_T \cup [\![\beta']\!]_T \\
[\![\mathbf{E}\beta]\!]_T = \pi_1([\![\beta]\!]_T) & [\![\mathtt{step}^*]\!]_T = [\![\mathtt{step}]\!]_T^* \\
 & [\![\beta/\beta']\!]_T = [\![\beta]\!]_T \circ [\![\beta']\!]_T \\
 & [\![(\mathtt{step}/?\alpha)^*]\!]_T = [\![(\mathtt{step}/?\alpha)]\!]_T^*.
\end{array}$$

CXPath defines two kinds of unary queries: those given by node formulae, and those given by path formulae β, selecting $[\![\beta]\!]_T^{root} = \{s \in D \mid (\varepsilon, s) \in [\![\beta]\!]_T\}$. Both classes capture precisely unary FO queries on trees [28].

5. XPath and $\mathrm{TL}^{\mathrm{tree}}$

XPath expressions can be translated into $\mathrm{TL}^{\mathrm{tree}}$. For example, consider an expression in the “traditional” XPath syntax: $e = /a//b[//c]$. It says: start at the root, find children labeled a, their descendants labeled b, and select those which have a c-descendant. It can be viewed as both a path formula and a node formula of XPath. An equivalent path formula is

$$\beta = \prec_{\mathrm{ch}} /?a/ \prec_{\mathrm{ch}}^* /?\big(b \wedge \mathbf{E}(\prec_{\mathrm{ch}}^* /?c)\big).$$

The set $[\![\beta]\!]_T^{root} = \{s \mid (\varepsilon, s) \in [\![\beta]\!]_T\}$ is precisely the set of nodes selected by e in T. Alternatively we can view it as a node formula

$$\alpha = b \wedge \mathbf{E}(\prec_{\mathrm{ch}}^* /?c) \wedge \mathbf{E}\big((\prec_{\mathrm{ch}}^{-})^*/?\big(a \wedge \mathbf{E}(\prec_{\mathrm{ch}}^{-} /root)\big)\big).$$

Here *root* is an abbreviation for a formula that tests for the root node. Then $[\![\alpha]\!]_T$ generates the set of nodes selected by e. It is known [29] that for every path formula β, one can construct in linear time a node formula α so that $[\![\beta]\!]_T^{root} = [\![\alpha]\!]_T$. Thus, from now on we deal with node XPath formulae.

The above formulae can be translated into an equivalent $\mathrm{TL}^{\mathrm{tree}}$ expression

$$b \wedge \mathbf{F}_{\mathrm{ch}} c \wedge \mathbf{F}_{\mathrm{ch}}^{-}\big(a \wedge \mathbf{X}_{\mathrm{ch}}^{-} root\big).$$

This formula selects b-labeled nodes with c-labeled descendants, and an a-ancestor which is a child of the root – this is of course equivalent to the original expression.

In what follows, the size $\|\psi\|$ of a formula ψ (both a $\mathrm{TL}^{\mathrm{tree}}$ and a CXPath formula) refers to the number of nodes in the parse tree of ψ.

Since both $\mathrm{TL}^{\mathrm{tree}}$ and CXPath are first-order expressively-complete [28], each core or conditional XPath expression is equivalent to a formula of $\mathrm{TL}^{\mathrm{tree}}$; however, no direct translation has previously been produced. We now give such a direct translation that, for each CXPath formula α, produces an equivalent $\mathrm{TL}^{\mathrm{tree}}$ formula φ_α. The crucial property of this translation is that, even if φ_α can be exponential in the size of α, the size of its Fischer–Ladner closure (the set of all subformulae and their negations) is at most linear in the size of the original formula α. This, together with the translation from $\mathrm{TL}^{\mathrm{tree}}$ to QAs, will guarantee single-exponential bounds on QAs equivalent to XPath formulae. In the translation, CXPath formulae of the form $(\mathtt{step}/?\alpha)^*$ will be referred to as *conditional axes*.

Theorem 1. *Each node formula α of core or conditional XPath can be effectively translated into an equivalent formula φ_α of* $\mathrm{TL}^{\mathrm{tree}}$ *such that the number of subformulae of φ_α is at most linear in the size of α. Moreover, if α does not use disjunctions of path formulae nor child conditional axes, then the size of φ_α is at most linear in the size of α.*

Proof. Given two $\mathrm{TL}^{\mathrm{tree}}$ formulae φ and φ' and a CXPath path formula β, we say that $\varphi' \equiv \mathbf{X}_\beta\varphi$ if for each tree T and each node s of T, one has that $(T,s) \models \varphi'$ iff there exists a node s' in T, with $(s,s') \in [\![\beta]\!]_T$, such that $(T,s') \models \varphi$.

Each CXPath node formula α is translated into a $\mathrm{TL}^{\mathrm{tree}}$ formula which we denote by φ_α; while each CXPath path formula is translated into a mapping x_β from $\mathrm{TL}^{\mathrm{tree}}$ formulae to $\mathrm{TL}^{\mathrm{tree}}$ formulae. The intended semantics of the translation is as follows:

1. If α is a node formula, then φ_α is an equivalent $\mathrm{TL}^{\mathrm{tree}}$ formula, that is for each tree T and each node s in T, we have that $(T,s) \models \varphi_\alpha$ iff $s \in [\![\alpha]\!]_T$.
2. If β is a path formula, then x_β is a mapping such that, for each $\mathrm{TL}^{\mathrm{tree}}$ formula φ, one has $x_\beta(\varphi) \equiv \mathbf{X}_\beta\varphi$.

The syntactic translation rules are the following:

α	φ_α
a	a
$\neg\alpha'$	$\neg\varphi_{\alpha'}$
$\alpha' \vee \alpha''$	$\varphi_{\alpha'} \vee \varphi_{\alpha''}$
$\mathbf{E}\beta$	$x_\beta(\top)$

β	$x_\beta(\varphi)$
$?\alpha$	$\varphi_\alpha \wedge \varphi$
$\prec_{\mathrm{ch}}$	$\mathbf{X}_{\mathrm{ch}}\varphi$
$\prec^*_{\mathrm{ch}}$	$\top\mathbf{U}_{\mathrm{ch}}\varphi$
$(\prec_{\mathrm{ch}}/?\alpha)^*$	$\varphi \vee \mathbf{X}_{\mathrm{ch}}(\varphi_\alpha\mathbf{U}_{\mathrm{ch}}(\varphi \wedge \varphi_\alpha))$
$(\prec^-_{\mathrm{ch}}/?\alpha)^*$	$(\mathbf{X}^-_{\mathrm{ch}}\varphi_\alpha)\mathbf{S}_{\mathrm{ch}}\varphi$
$(\prec_{\mathrm{ns}}/?\alpha)^*$	$(\mathbf{X}_{\mathrm{ns}}\varphi_\alpha)\mathbf{U}_{\mathrm{ns}}\varphi$
$(\prec^-_{\mathrm{ns}}/?\alpha)^*$	$(\mathbf{X}^-_{\mathrm{ns}}\varphi_\alpha)\mathbf{S}_{\mathrm{ns}}\varphi$
β'/β''	$x_{\beta'} \circ x_{\beta''}(\varphi)$
$\beta' \vee \beta''$	$x_{\beta'}(\varphi) \vee x_{\beta''}(\varphi)$

In the cases $\beta = \prec^-_{\mathrm{ch}}$, $\beta = \prec_{\mathrm{ns}}$, $\beta = \prec^-_{\mathrm{ns}}$, translation rules are obtained from the case $\beta = \prec_{\mathrm{ch}}$ by replacing $\mathbf{X}_{\mathrm{ch}}$ with $\mathbf{X}^-_{\mathrm{ch}}$, $\mathbf{X}_{\mathrm{ns}}$ and $\mathbf{X}^-_{\mathrm{ns}}$, respectively. In the cases $\beta = \prec^{-*}_{\mathrm{ch}}$, $\beta = \prec^*_{\mathrm{ns}}$, $\beta = \prec^{-*}_{\mathrm{ns}}$, translation rules are obtained from the case $\beta = \prec^*_{\mathrm{ch}}$ by replacing $\mathbf{U}_{\mathrm{ch}}$ with $\mathbf{S}_{\mathrm{ch}}$, $\mathbf{U}_{\mathrm{ns}}$ and $\mathbf{S}_{\mathrm{ns}}$, respectively.

We now show by induction that φ_α and x_β have the intended semantics stated in (1) and (2) above. In the base case that $\alpha = a$, clearly φ_α is equivalent to α. Moreover in the base case that $\beta = \prec_{\mathrm{ch}}$ (resp. $\beta = \prec^*_{\mathrm{ch}}$), by the translation rules, $x_\beta(\varphi) = \mathbf{X}_{\mathrm{ch}}\varphi$ (resp. $x_\beta(\varphi) = \top\mathbf{U}_{\mathrm{ch}}\varphi$), therefore for each tree T and each node s of T, we have that $(T,s) \models x_\beta(\varphi)$ iff there exists s' such that $s \prec_{\mathrm{ch}} s'$ (resp. $s \prec^*_{\mathrm{ch}} s'$) and $(T,s') \models \varphi$. In other words, $(T,s) \models x_\beta(\varphi)$ iff there exists s' such that $(s,s') \in [\![\beta]\!]_T$ and $(T,s') \models \varphi$. By definition of $\mathbf{X}_\beta$, it follows that $x_\beta(\varphi) \equiv \mathbf{X}_\beta\varphi$. The proofs for the other steps ($\prec^-_{\mathrm{ch}}$, $\prec_{\mathrm{ns}}$, $\prec^-_{\mathrm{ns}}$, and their transitive closures) follow the same lines.

We now deal with the general cases, by using structural induction on the CXPath formula.

In the case that α is $\neg\alpha'$ or $\alpha' \vee \alpha''$ or $\mathbf{E}\beta$, we assume that $\varphi_{\alpha'}$, $\varphi_{\alpha''}$ and x_β have the intended semantics stated in (1) and (2) above. Then $\varphi_{\alpha'}$ and $\varphi_{\alpha''}$ are equivalent to α' and α'' respectively; similarly $x_\beta(\top) \equiv \mathbf{X}_\beta\top$. As a consequence, by definition of the semantics of $\mathrm{TL}^{\mathrm{tree}}$ and CXPath formulae, $\neg\varphi_{\alpha'}$ is equivalent to $\neg\alpha'$, and $\varphi_{\alpha'} \vee \varphi_{\alpha''}$ is equivalent to $\alpha' \vee \alpha''$. Also, by definition of $\mathbf{X}_\beta$, for each tree T and each node s of T, one has that $(T,s) \models x_\beta(\top)$ iff there exists a node s' in T, with $(s,s') \in [\![\beta]\!]_T$. Then $x_\beta(\top)$ is equivalent to $\mathbf{E}\beta$.

In the case that $\beta = ?\alpha$, by the induction hypothesis, φ_α is equivalent to α. Then for each tree T and each node s of T, one has that $(T,s) \models x_\beta(\varphi)$ iff $s \in [\![\alpha]\!]_T$ (or equivalently $(s,s) \in [\![?\alpha]\!]_T$) and $(T,s) \models \varphi$. It follows that $x_\beta(\varphi) \equiv \mathbf{X}_\beta\varphi$.

In the case that $\beta = (\prec_{\mathrm{ch}}/?\alpha)^*$, we have $x_\beta(\varphi) = \varphi \vee \mathbf{X}_{\mathrm{ch}}(\varphi_\alpha\mathbf{U}_{\mathrm{ch}}(\varphi \wedge \varphi_\alpha))$ and, by the induction hypothesis, φ_α is equivalent to α. Now notice that $(s,s') \in [\![\beta]\!]_T$ iff

(a) either $s' = s$ or
(b) there exists s_0 with $s \prec_{\mathrm{ch}} s_0$ such that $s_0 \prec^*_{\mathrm{ch}} s'$ and for all s'' with $s_0 \prec^*_{\mathrm{ch}} s'' \prec^*_{\mathrm{ch}} s'$, one has $s'' \in [\![\alpha]\!]_T$ (or equivalently $(T,s'') \models \varphi_\alpha$).

On the other hand, it follows from the definition of $x_\beta(\varphi)$ that $(T,s) \models x_\beta(\varphi)$ iff

(i) either $(T,s) \models \varphi$ or
(ii) there exists s_0 with $s \prec_{\mathrm{ch}} s_0$ and s' with $s_0 \prec^*_{\mathrm{ch}} s'$ such that $(T,s') \models \varphi$ and for all s'' satisfying $s_0 \prec^*_{\mathrm{ch}} s'' \prec^*_{\mathrm{ch}} s'$, we have $(T,s'') \models \varphi_\alpha$.

By comparing (i) and (ii) with (a) and (b) above, one derives that $(T,s) \models x_\beta(\varphi)$ iff there exists s' with $(s,s') \in [\![\beta]\!]_T$ such that $(T,s') \models \varphi$; that is $x_\beta(\varphi) \equiv \mathbf{X}_\beta \varphi$.

In the case that $\beta = (\prec_{\text{ch}}^- /?\alpha)^*$, we have $x_\beta(\varphi) = (\mathbf{X}_{\text{ch}}^- \varphi_\alpha)\mathbf{S}_{\text{ch}}\varphi$ and, by the induction hypothesis, φ_α is equivalent to α. First notice that $(s,s') \in [\![\beta]\!]_T$ iff $s' \prec_{\text{ch}}^* s$ and for all $s'' \neq s'$ with $s' \prec_{\text{ch}}^* s'' \prec_{\text{ch}}^* s$, one has $(T,s'') \models \mathbf{X}_{\text{ch}}^- \varphi_\alpha$. By definition of x_β and the semantics of $\mathbf{S}_{\text{ch}}$ it follows that $(T,s) \models x_\beta(\varphi)$ iff there exists s' with $(s,s') \in [\![\beta]\!]_T$, satisfying $(T,s') \models \varphi$; that is $x_\beta(\varphi) \equiv \mathbf{X}_\beta \varphi$.

The proofs for $\beta = (\prec_{\text{ns}} /?\alpha)^*$ and $\beta = (\prec_{\text{ns}}^- /?\alpha)^*$ follow the same lines as the case $\beta = (\prec_{\text{ch}}^- /?\alpha)^*$.

In the case that $\beta = \beta'/\beta''$, by the induction hypothesis, for each TL$^{\text{tree}}$ formula ψ, we have $x_{\beta'}(\psi) \equiv \mathbf{X}_{\beta'}\psi$ and $x_{\beta''}(\psi) \equiv \mathbf{X}_{\beta''}\psi$. Therefore for each tree T and each node s of T, we have that $(T,s) \models x_\beta(\varphi)$ iff there exists a node s' with $(s,s') \in [\![\beta']\!]_T$, such that $(T,s') \models x_{\beta''}(\varphi)$. On the other hand $(T,s') \models x_{\beta''}(\varphi)$ iff there exists a node s'' with $(s',s'') \in [\![\beta'']\!]_T$ such that $(T,s'') \models \varphi$. In other words, $(T,s) \models x_\beta(\varphi)$ iff there exists a node s'' with $(s,s'') \in [\![\beta'/\beta'']\!]_T$ such that $(T,s'') \models \varphi$. It follows that $x_\beta(\varphi) \equiv \mathbf{X}_\beta\varphi$.

In the case that $\beta = \beta' \vee \beta''$ – under the hypothesis that $x_{\beta'}(\varphi) \equiv \mathbf{X}_{\beta'}\varphi$ and $x_{\beta''}(\varphi) \equiv \mathbf{X}_{\beta''}\varphi$ – for a tree T and a node s of T, we have that $(T,s) \models x_\beta(\varphi)$ iff $(T,s) \models x_{\beta'}(\varphi)$ or $(T,s) \models x_{\beta''}(\varphi)$. This implies that $(T,s) \models x_\beta(\varphi)$ iff there exists a node s' with $(s,s') \in [\![\beta']\!]_T$ or $(s,s') \in [\![\beta'']\!]_T$ (that is $(s,s') \in [\![\beta']\!]_T \cup [\![\beta'']\!]_T$), such that $(T,s') \models \varphi$. Given that $[\![\beta' \vee \beta'']\!]_T = [\![\beta']\!]_T \cup [\![\beta'']\!]_T$, we have that $x_\beta(\varphi) \equiv \mathbf{X}_\beta\varphi$.

We now analyze the size of the produced TL$^{\text{tree}}$ formulae w.r.t. the size of the corresponding CXPath formulae. In what follows, for each TL$^{\text{tree}}$ formula φ, we let $\text{sf}(\varphi)$ stand for the set of its subformulae.

Claim 1.

1. *For each CXPath node formula α,*
$$\left|\text{sf}(\varphi_\alpha)\right| \leqslant 4\|\alpha\|.$$
Moreover if no subformula of α is a path formula of the form $\beta' \vee \beta''$ or $(\prec_{\text{ch}} /?\alpha')^*$, then $\|\varphi_\alpha\| \leqslant 2\|\alpha\|$.
2. *For each CXPath path formula β and* TL$^{\text{tree}}$ *formula φ,*
$$\text{sf}\big(x_\beta(\varphi)\big) = \text{sf}(\varphi) \cup C_\beta$$
for some set C_β of subformulae, with $|C_\beta| \leqslant 4\|\beta\|$. Moreover if no subformula of β is a path formula of the form $\beta' \vee \beta''$ or $(\prec_{\text{ch}} /?\alpha')^*$, for each TL$^{\text{tree}}$ *formula φ, we have that $\|x_\beta(\varphi)\| \leqslant \|\varphi\| + 2\|\beta\|$.*

Proof. We prove the claim by induction on the structure of the formulae α and β. In the base case that $\alpha = a$, $a \in \Sigma$, we have $|\text{sf}(\varphi_\alpha)| = \|\varphi_\alpha\| = 1$ while $\|\alpha\| = 1$, thus $|\text{sf}(\varphi_\alpha)| \leqslant 4\|\alpha\|$ and $\|\varphi_\alpha\| \leqslant 2\|\alpha\|$ hold.

In the case that $\beta = \texttt{step}$, we have $\|x_\beta(\varphi)\| = 1 + \|\varphi\|$ and $\text{sf}(x_\beta(\varphi)) = \text{sf}(\varphi) \cup \{\mathbf{X}_{\text{ch}}\varphi\}$ thus $|C_\beta| = 1$. On the other hand $\|\beta\| = 1$, therefore $|C_\beta| \leqslant 4\|\beta\|$ and $\|x_\beta(\varphi)\| \leqslant \|\varphi\| + 2\|\beta\|$ hold.

In the case that $\beta = \prec_{\text{ch}}^*$, we have $\|x_\beta(\varphi)\| = \|\varphi\| + 2$ and $\text{sf}(x_\beta(\varphi)) = \text{sf}(\varphi) \cup \{\top, \top\mathbf{U}_{\text{ch}}\varphi\}$, thus $|C_\beta| = 2$. On the other hand $\|\beta\| = 1$, therefore $|C_\beta| \leqslant 4\|\beta\|$ and $\|x_\beta(\varphi)\| \leqslant \|\varphi\| + 2\|\beta\|$ hold.

In the general case:

- If $\alpha = \neg\alpha'$ (or $\alpha = \alpha' \vee \alpha''$), the subformulae of φ_α are $\text{sf}(\varphi_\alpha) = \{\neg\varphi'_\alpha\} \cup \text{sf}(\varphi_{\alpha'})$ (or $\text{sf}(\varphi_\alpha) = \{\varphi_{\alpha'} \vee \varphi_{\alpha''}\} \cup \text{sf}(\varphi_{\alpha'}) \cup \text{sf}(\varphi_{\alpha''})$, resp.). By the induction hypothesis, $|\text{sf}(\varphi_{\alpha'})| \leqslant 4\|\alpha'\|$, therefore $|\text{sf}(\varphi_\alpha)| \leqslant 1 + 4\|\alpha'\| \leqslant 4 + 4\|\alpha'\| = 4\|\alpha\|$ (resp., $|\text{sf}(\varphi_\alpha)| \leqslant 1 + 4\|\alpha'\| + 4\|\alpha''\| \leqslant 4\|\alpha\|$).
Moreover if α does not contain disjunction between path formulae, nor conditional child axes, so does α'. Therefore, by the induction hypothesis on α', we have that $\|\varphi_\alpha\| = 1 + \|\varphi'_\alpha\| \leqslant 1 + 2\|\alpha'\| \leqslant 2\|\alpha\|$.
Similarly, in the case $\alpha = \alpha' \vee \alpha''$, the size $\|\varphi_\alpha\| = 1 + \|\varphi_{\alpha'}\| + \|\varphi_{\alpha''}\| \leqslant 1 + 2\|\alpha'\| + 2\|\alpha''\| \leqslant 2\|\alpha\|$.
- If $\alpha = E\beta$, then $\text{sf}(\varphi_\alpha) = \text{sf}(x_\beta(\top))$. By the induction hypothesis $|\text{sf}(x_\beta(\top))| \leqslant 1 + 4\|\beta\| \leqslant 4\|\mathbf{E}\beta\|$. Thus $|\text{sf}(\varphi_\alpha)| \leqslant 4\|\alpha\|$. If α, and therefore β, has no subformulae of the form $\beta' \vee \beta''$ or of the form $(\prec_{\text{ch}} /?\alpha')^*$, the induction hypothesis applies to β, implying $\|\varphi_\alpha\| = \|x_\beta(\top)\| \leqslant 1 + 2\|\beta\| \leqslant 2\|\alpha\|$.
- If $\beta = ?\alpha$, then $\text{sf}(x_\beta(\varphi)) = \{\varphi_\alpha \wedge \varphi\} \cup \text{sf}(\varphi_\alpha) \cup \text{sf}(\varphi)$ and, by the induction hypothesis, $|\{\varphi_\alpha \wedge \varphi\} \cup \text{sf}(\varphi_\alpha)| \leqslant 1 + |\text{sf}(\varphi_\alpha)| \leqslant 1 + 4\|\alpha\| \leqslant 4\|\beta\|$.
Moreover, if β contains no disjunctions between path formulae nor conditional child axes, the induction hypothesis on α implies that $\|x_\beta(\varphi)\| = 1 + \|\varphi_\alpha\| + \|\varphi\| \leqslant 1 + 2\|\alpha\| + \|\varphi\| \leqslant 2\|\beta\| + \|\varphi\|$.
- If $\beta = (\prec_{\text{ch}} /?\alpha)^*$, then $\text{sf}(x_\beta(\varphi)) = \{\varphi \vee \mathbf{X}_{\text{ch}}(\varphi_\alpha \mathbf{U}_{\text{ch}}(\varphi \wedge \varphi_\alpha)), \varphi \wedge \varphi_\alpha, \varphi_\alpha \mathbf{U}_{\text{ch}}(\varphi \wedge \varphi_\alpha), \mathbf{X}_{\text{ch}}(\varphi_\alpha \mathbf{U}_{\text{ch}}(\varphi \wedge \varphi_\alpha))\} \cup \text{sf}(\varphi_\alpha) \cup \text{sf}(\varphi)$. By the induction hypothesis $|\text{sf}(\varphi_\alpha)| \leqslant 4\|\alpha\|$ thus $|\{\varphi \vee \mathbf{X}_{\text{ch}}(\varphi_\alpha \mathbf{U}_{\text{ch}}(\varphi \wedge \varphi_\alpha)), \varphi \wedge \varphi_\alpha, \varphi_\alpha \mathbf{U}_{\text{ch}}(\varphi \wedge \varphi_\alpha), \mathbf{X}_{\text{ch}}(\varphi_\alpha \mathbf{U}_{\text{ch}}(\varphi \wedge \varphi_\alpha)\} \cup \text{sf}(\varphi_\alpha)| \leqslant 4 + 4\|\alpha\| = 4\|\beta\|$.
- If $\beta = (\prec_{\text{ch}}^- /?\alpha)^*$, then $\text{sf}(x_\beta(\varphi)) = \{(\mathbf{X}_{\text{ch}}^-\varphi_\alpha)\mathbf{S}_{\text{ch}}\varphi, \mathbf{X}_{\text{ch}}^-\varphi_\alpha\} \cup \text{sf}(\varphi_\alpha) \cup \text{sf}(\varphi)$. By the induction hypothesis $|\text{sf}(\varphi_\alpha)| \leqslant 4\|\alpha\|$ thus $|\{(\mathbf{X}_{\text{ch}}^-\varphi_\alpha)\mathbf{S}_{\text{ch}}\varphi, \mathbf{X}_{\text{ch}}^-\varphi_\alpha\} \cup \text{sf}(\varphi_\alpha)| \leqslant 2 + 4\|\alpha\| < 4\|\beta\|$.
We proceed similarly when β is based on the other `step` formulae ($\prec_{\text{ns}}$ and $\prec_{\text{ns}}^-$).
- If $\beta = \beta'/\beta''$, then $\text{sf}(x_\beta(\varphi)) = \text{sf}(x_{\beta'}(x_{\beta''}(\varphi)))$. By the induction hypothesis on β', $\text{sf}(x_\beta(\varphi)) = \text{sf}(x_{\beta''}(\varphi)) \cup C_{\beta'}$. Now we can apply the induction hypothesis on β'' to derive $\text{sf}(x_\beta(\varphi)) = \text{sf}(\varphi) \cup C_{\beta''} \cup C_{\beta'}$, where $|C_{\beta''} \cup C_{\beta'}| \leqslant 4\|\beta'\| + 4\|\beta''\| < 4\|\beta\|$.

Moreover when β contains no disjunction between path formulae and no conditional child axes, so do β' and β''. It follows from the induction hypothesis that $\|x_\beta(\varphi)\| = \|x_{\beta'}(x_{\beta''}(\varphi))\| \leqslant \|x_{\beta''}(\varphi)\| + 2\|\beta'\| \leqslant \|\varphi\| + 2\|\beta''\| + 2\|\beta'\| < \|\varphi\| + 2\|\beta\|$.

- If $\beta = \beta' \vee \beta''$, then $\mathrm{sf}(x_\beta(\varphi)) = \{x_{\beta'}(\varphi) \vee x_{\beta''}(\varphi)\} \cup \mathrm{sf}(x_{\beta'}(\varphi)) \cup \mathrm{sf}(x_{\beta''}(\varphi))$. By the induction hypothesis, we have that $\mathrm{sf}(x_{\beta'}(\varphi)) \cup \mathrm{sf}(x_{\beta''}(\varphi)) = \mathrm{sf}(\varphi) \cup C_{\beta'} \cup C_{\beta''}$ with $|C_{\beta'}| \leqslant 4\|\beta'\|$ and $|C_{\beta''}| \leqslant 4\|\beta''\|$. Therefore $\mathrm{sf}(x_\beta(\varphi)) = \mathrm{sf}(\varphi) \cup C_\beta$ where $|C_\beta| = |\{x_{\beta'}(\varphi) \vee x_{\beta''}(\varphi)\} \cup C_{\beta'} \cup C_{\beta''}| \leqslant 1 + 4\|\beta'\| + 4\|\beta''\| < 4(1 + \|\beta'\| + \|\beta''\|) = 4\|\beta\|$.

This proves the claim by induction. □

The proof of the theorem follows directly from item (1) of Claim 1. □

Notice that Core Xpath formulae do not contain conditional axes at all, therefore we derive the following corollary:

Corollary 1. *Node formulae α of Core Xpath that do not use any disjunctions of path formulae can be effectively translated into equivalent* $\mathrm{TL}^{\mathrm{tree}}$ *formulae of size at most linear in the size of α.*

Observe also that translation rules for $(\prec_{\mathrm{ch}}/?\alpha)^*$ and $\beta' \vee \beta''$ are the ones determining a size exponential blowup in the translation from CXPath to $\mathrm{TL}^{\mathrm{tree}}$; in fact the $\mathrm{TL}^{\mathrm{tree}}$ translation of these formulae contains duplicates of subformulae φ or φ_α. On the other hand, an alternative version of CXPath can be defined, where conditional axes have the form $\beta = (?\alpha/\mathtt{step})^*$. The translation rules for this form of conditional axes are simpler: for $\mathtt{step} = \prec_{\mathrm{ch}}$ we have $x_\beta(\varphi) = \varphi_\alpha \mathbf{U}_{\mathrm{ch}} \varphi$, and for the other steps ($\prec_{\mathrm{ch}}^-$, $\prec_{\mathrm{ns}}$, $\prec_{\mathrm{ns}}^-$), the operator $\mathbf{U}_{\mathrm{ch}}$ is replaced by $\mathbf{S}_{\mathrm{ch}}$, $\mathbf{U}_{\mathrm{ns}}$ and $\mathbf{S}_{\mathrm{ns}}$, respectively. Therefore, for this alternative version of CXPath, no translation rule other than the one for $\beta' \vee \beta''$ determines an exponential blowup in size. This is stated in the following corollary:

Corollary 2. *Node formulae α of CXPath with conditional axes of the form $(?\alpha/\mathtt{step})^*$ that do not use any disjunctions of path formulae can be effectively translated into equivalent* $\mathrm{TL}^{\mathrm{tree}}$ *formulae of size at most linear in the size of α.*

6. Tree logic into query automata: A translation

Our goal is to translate $\mathrm{TL}^{\mathrm{tree}}$ into single-run QAs. We do a direct translation into unranked QAs, as opposed to coding of unranked trees into binary (which is a common technique). Such coding is problematic for two reasons. First, simple navigation over unranked trees may look unnatural when coded into binary, resulting in more complex formulae (child, for example, becomes 'left successor followed by zero or more right successors'). Second, coding into binary trees makes reasoning about views much harder. The property of being "upward-closed" (i.e. of being a subtree view), which is essential for decidability of certain answers, is not even preserved by the translation. Thus, we do a direct translation into unranked QAs, and then apply it to XML specifications.

Since values of transitions $\delta(q,a)$ in unranked QAs are not sets of states but rather NFAs representing regular languages over states, we measure the size of $\mathcal{QA} = (Q, F, Q_s, \delta)$ not as the number $|Q|$ of states, but rather as

$$\|\mathcal{QA}\| = |Q| + \sum_{q \in Q, a \in \Sigma} \|\delta(q,a)\|,$$

where, if $\delta(q,a)$ is an NFA with states S and transition relation σ, the size $\|\delta(q,a)\| = |S| + |\sigma|$.

We first sketch the automaton construction and then show formally that every $\mathrm{TL}^{\mathrm{tree}}$ formula can be translated in exponential time into an equivalent query automaton. First, as is common with translations into nondeterministic automata [42], we need to work with a version of $\mathrm{TL}^{\mathrm{tree}}$ in which all negations are pushed to propositions. To deal with until and since operators, we shall introduce four operators $\mathbf{R}_*$ and $\mathbf{I}_*$ for $*$ being 'ch' or 'ns' so that $\neg(\alpha \mathbf{U}_* \beta) \leftrightarrow \neg\alpha \mathbf{R}_* \neg\beta$ and $\neg(\alpha \mathbf{S}_* \beta) \leftrightarrow \neg\alpha \mathbf{I}_* \neg\beta$; this part is completely standard. However, trees do not have a linear structure and we cannot just push negation inside the $\mathbf{X}$ operators: for example, $\neg\mathbf{X}_{\mathrm{ch}}\varphi$ is not $\mathbf{X}_{\mathrm{ch}}\neg\varphi$. Since our semantics of the next operators is existential (there is a successor node in which the formula is true), we need to add their universal analogs. For example, $\mathbf{X}_{\mathrm{ch}}^{\forall}\varphi$ is true in s if for every successor s' of s in the domain of the tree, φ is true in s'. Then of course we have $\neg\mathbf{X}_{\mathrm{ch}}\varphi \leftrightarrow \mathbf{X}_{\mathrm{ch}}^{\forall}\neg\varphi$. We add four such operators ($\mathbf{X}_{\mathrm{ch}}^{\forall}, \mathbf{X}_{\mathrm{ns}}^{\forall}, \mathbf{X}_{\mathrm{ch}}^{-\forall}, \mathbf{X}_{\mathrm{ns}}^{-\forall}$). Other axes have a linear structure, so one could alternatively add tests for the root, first, and last child of a node to deal with them. For example, $\neg\mathbf{X}_{\mathrm{ch}}^{-}\varphi \leftrightarrow \mathbf{X}_{\mathrm{ch}}^{-}\neg\varphi \vee \alpha_{root}$, where α_{root} is a test for the root. But for symmetry we prefer to deal with the four universal versions of the next/previous operators, since it is unavoidable for $\mathbf{X}_{\mathrm{ch}}$.

With these additions, we can push negations to propositions, so we assume negations only occur in subformulae $\neg a$ for $a \in \Sigma$. Given a formula φ in this version of $\mathrm{TL}^{\mathrm{tree}}$, we will denote as $\mathcal{QA}_\varphi$ the query automaton constructed for φ. The states of $\mathcal{QA}_\varphi$ will be maximally consistent subsets of the Fischer–Ladner closure of φ (in particular, for each state q and a subformula ψ, exactly one of ψ and $\neg\psi$ is in q).

The transitions have to ensure that all "horizontal" temporal connectives behave properly, and that "vertical" transitions are consistent. The alphabet of each automaton $\delta(q,a)$ is the set of states of $\mathcal{QA}_\varphi$; that is, letters of $\delta(q,a)$ are sets of formulae. Each $\delta(q,a)$ is a product of three automata. The first guarantees that eventualities $\alpha\mathbf{U}_{\mathrm{ns}}\beta$ and $\alpha\mathbf{S}_{\mathrm{ns}}\beta$ are fulfilled in the oldest and youngest siblings. For that, we impose conditions on the initial states $\delta(q,a)$'s that they need to read a letter (which is a state of $\mathcal{QA}_\varphi$) that may not contain $\alpha\mathbf{S}_{\mathrm{ns}}\beta$ without containing β, and on their final state guaranteeing that in the last letter we do not have a subformula $\alpha\mathbf{U}_{\mathrm{ns}}\beta$ without having β.

The second automaton enforces horizontal transitions, and it behaves very similarly to the standard LTL-to-Büchi construction; it only deals with next-sibling connectives. For example, if $\mathbf{X}_{\mathrm{ns}}\alpha$ is the current state of $\mathcal{QA}$ for a node $s\cdot i$, then the state for $s\cdot(i+1)$ contains α, and that if $\alpha\mathbf{U}_{\mathrm{ns}}\beta$ is in the state for $s\cdot i$ but β is not, then $\alpha\mathbf{U}_{\mathrm{ns}}\beta$ is propagated into the state for $s\cdot(i+1)$.

The third automaton enforces vertical transitions. We give a few sample rules. If q contains the negation of $\alpha\mathbf{S}_{\mathrm{ch}}\beta$, then the automaton rejects after seeing a state which contains $\alpha\mathbf{S}_{\mathrm{ch}}\beta$ but does not contain β (since in this case $\alpha\mathbf{S}_{\mathrm{ch}}\beta$ must propagate to the parent). If q contains $\alpha\mathbf{U}_{\mathrm{ch}}\beta$ and does not contain β, then the automaton only accepts if one of its input letters contains $\alpha\mathbf{U}_{\mathrm{ch}}\beta$. And if q contains $\mathbf{X}_{\mathrm{ch}}\alpha$, then it only accepts if one of its input letters contains α. In addition, we have to enforce eventualities $\alpha\mathbf{U}_{\mathrm{ch}}\beta$ by disallowing these automata to accept ε if q contains $\alpha\mathbf{U}_{\mathrm{ch}}\beta$ and does not contain β.

The final states of $\mathcal{QA}_\varphi$ at the root must enforce correctness of $\alpha\mathbf{S}_{\mathrm{ch}}\beta$ formulae: with each such formula, states from F must contain β as well. This completes the construction. When all automata $\delta(q,a)$ are properly coded, the $2^{O(n)}$ bound follows. We then show a standard lemma that in an accepting run, a node is assigned a state that contains a subformula α iff α is true in that node. This guarantees that for every tree, there is an accepting run. Since each state has either α or $\neg\alpha$ in it, it follows that the resulting QA is single-run.

We now show this formally:

Theorem 2. *Every* $\mathrm{TL}^{\mathrm{tree}}$ *formula* φ *of size n can be translated, in exponential time, into an equivalent single-run query automaton* $\mathcal{QA}_\varphi$ *of size* $2^{O(n)}$*, i.e. a query automaton such that* $\mathcal{QA}_\varphi(T)=\{s\mid(T,s)\models\varphi\}$ *for every tree* T.

Proof. We extend $\mathrm{TL}^{\mathrm{tree}}$ with eight operators, $\mathbf{R}_*$, $\mathbf{I}_*$, $\mathbf{X}_*^{\forall}$ and $\mathbf{X}_*^{-\forall}$, where $*$ is either ch or ns. The semantics of the new operators is defined so that:

- $\varphi\mathbf{R}_*\varphi'\leftrightarrow\neg(\neg\varphi\mathbf{U}_*\neg\varphi')$,
- $\varphi\mathbf{I}_*\varphi'\leftrightarrow\neg(\neg\varphi\mathbf{S}_*\neg\varphi')$,
- $\mathbf{X}_*^{\forall}\varphi\leftrightarrow\neg\mathbf{X}_*\neg\varphi$,
- $\mathbf{X}_*^{-\forall}\varphi\leftrightarrow\neg\mathbf{X}_*^{-}\neg\varphi$.

With these operators, we can assume that negation only occurs in subformulae $\neg a$ for $a\in\Sigma$. That is, we work with an equivalent $\mathrm{TL}^{\mathrm{tree}}$ syntax

$$\begin{aligned}\varphi,\varphi' := {} & \top\mid\bot\mid a\mid\neg a\mid\varphi\vee\varphi'\mid\varphi\wedge\varphi'\\ & \mid\mathbf{X}_*\varphi\mid\mathbf{X}_*^{-}\varphi\mid\mathbf{X}_*^{\forall}\varphi\mid\mathbf{X}_*^{-\forall}\varphi\mid\varphi\mathbf{U}_*\varphi'\mid\varphi\mathbf{S}_*\varphi'\mid\varphi\mathbf{R}_*\varphi'\mid\varphi\mathbf{I}_*\varphi'.\end{aligned}$$

A formula in $\mathrm{TL}^{\mathrm{tree}}$ as presented earlier can be rewritten, in linear time, into an equivalent formula in this syntax by propagating $\neg$ all the way to the atoms.

Next, as for LTL-to-automata translation, we define valid labelings of trees with $\mathrm{TL}^{\mathrm{tree}}$ formulae and their properties. We will then show how to construct a query automaton that enforces such a labeling for a given formula, and prove that it is the desired query automaton.

Valid labelings. Recall that the Fischer–Ladner closure of a $\mathrm{TL}^{\mathrm{tree}}$ formula φ is defined as the set of all subformulae of φ and their complements:

$$cl(\varphi)=\{\psi\mid\psi\text{ is a subformula of }\varphi\}\cup\{\neg\psi\mid\psi\text{ is a subformula of }\varphi\}$$

where the complement $\neg\psi$ stands for the formula obtained by pushing negation through the operators of ψ in the usual way. We identify $\neg\neg\psi$ with ψ.

Given an unranked Σ-labeled tree $T=(D,\lambda)$ and a $\mathrm{TL}^{\mathrm{tree}}$ formula φ over Σ, a *closure labeling* of T with φ is a mapping $\tau:D\to 2^{cl(\varphi)}$. A closure labeling τ of T is *valid* if each formula that labels a node is satisfied in that node. That is, for each node s of T and for each formula $\varphi\in\tau(s)$, we have that $(T,s)\models\varphi$. We next prove that a closure labeling τ that satisfies the following conditions on each node s, is valid:

1. (a) $\bot\notin\tau(s)$;
   (b) if $\varphi\vee\varphi'\in\tau(s)$, then either $\varphi\in\tau(s)$ or $\varphi'\in\tau(s)$;
   (c) if $\varphi\wedge\varphi'\in\tau(s)$, then both $\varphi\in\tau(s)$ and $\varphi'\in\tau(s)$;
2. for each $a\in\Sigma$, if $a\in\tau(s)$ then $\lambda(s)=a$, and if $\neg a\in\tau(s)$ then $\lambda(s)\neq a$;
3. (a) if $\mathbf{X}_{\mathrm{ch}}\varphi\in\tau(s)$, then for some i, the node $s\cdot i$ is in D and $\varphi\in\tau(s\cdot i)$;
   (b) if $\mathbf{X}_{\mathrm{ch}}^{\forall}\varphi\in\tau(s)$, then for all i such that $s\cdot i$ is in D, we have $\varphi\in\tau(s\cdot i)$;
   (c) if $\mathbf{X}_{\mathrm{ch}}^{-}\varphi\in\tau(s)$ then there exists $s'\in D$ such that $s'\prec_{\mathrm{ch}}s$ and $\varphi\in\tau(s')$;

(d) if $\mathbf{X}_{\mathrm{ch}}^{-\forall}\varphi \in \tau(s)$ and there exists $s' \in D$ such that $s' \prec_{\mathrm{ch}} s$, then $\varphi \in \tau(s')$;
(e) if $\varphi\mathbf{U}_{\mathrm{ch}}\varphi' \in \tau(s)$, then:
– either $\varphi' \in \tau(s)$ or
– $\varphi \in \tau(s)$ and, for some i, the node $s \cdot i$ is in D and $\varphi\mathbf{U}_{\mathrm{ch}}\varphi' \in \tau(s \cdot i)$;
(f) if $\varphi\mathbf{S}_{\mathrm{ch}}\varphi' \in \tau(s)$, then:
– either $\varphi' \in \tau(s)$ or
– $\varphi \in \tau(s)$ and $s = s' \cdot i$, for some $s' \in D$ and some i, and $\varphi\mathbf{S}_{\mathrm{ch}}\varphi' \in \tau(s')$;
(g) if $\varphi\mathbf{R}_{\mathrm{ch}}\varphi' \in \tau(s)$ then:
– $\varphi' \in \tau(s)$ and
– either $\varphi \in \tau(s)$ or $\varphi\mathbf{R}_{\mathrm{ch}}\varphi' \in \tau(s \cdot i)$, for each i such that $s \cdot i \in D$;
(h) if $\varphi\mathbf{I}_{\mathrm{ch}}\varphi' \in \tau(s)$, then:
– $\varphi' \in \tau(s)$ and
– if $s = s' \cdot i$ for some $s' \in D$ and some i, then either $\varphi \in \tau(s)$ or $\varphi\mathbf{I}_{\mathrm{ch}}\varphi' \in \tau(s')$;

4. (a) if $\mathbf{X}_{\mathrm{ns}}\varphi \in \tau(s)$, then there exists $s' \in D$ with $s \prec_{\mathrm{ns}} s'$ and $\varphi \in \tau(s')$;
(b) if $\mathbf{X}_{\mathrm{ns}}^{\forall}\varphi \in \tau(s)$ and there exists $s' \in D$ with $s \prec_{\mathrm{ns}} s'$, then $\varphi \in \tau(s')$;
(c) if $\mathbf{X}_{\mathrm{ns}}^{-}\varphi \in \tau(s)$, then there exists $s' \in D$ with $s' \prec_{\mathrm{ns}} s$ and $\varphi \in \tau(s')$;
(d) if $\mathbf{X}_{\mathrm{ns}}^{-\forall}\varphi \in \tau(s)$, and there exists $s' \in D$ with $s' \prec_{\mathrm{ns}} s$, then $\varphi \in \tau(s')$;
(e) if $\varphi\mathbf{U}_{\mathrm{ns}}\varphi' \in \tau(s)$, then:
– either $\varphi' \in \tau(s)$ or
– $\varphi \in \tau(s)$ and there exists $s' \in D$ with $s \prec_{\mathrm{ns}} s'$ and $\varphi\mathbf{U}_{\mathrm{ns}}\varphi' \in \tau(s')$;
(f) if $\varphi\mathbf{S}_{\mathrm{ns}}\varphi' \in \tau(s)$, then:
– either $\varphi' \in \tau(s)$ or
– $\varphi \in \tau(s)$ and there exists $s' \in D$ with $s' \prec_{\mathrm{ns}} s$ and $\varphi\mathbf{S}_{\mathrm{ns}}\varphi' \in \tau(s')$;
(g) if $\varphi\mathbf{R}_{\mathrm{ns}}\varphi' \in \tau(s)$ then:
– $\varphi' \in \tau(s)$ and
– if there exists $s' \in D$ with $s \prec_{\mathrm{ns}} s'$, either $\varphi \in \tau(s)$ or $\varphi\mathbf{R}_{\mathrm{ns}}\varphi' \in \tau(s')$;
(h) if $\varphi\mathbf{I}_{\mathrm{ns}}\varphi' \in \tau(s)$, then:
– $\varphi' \in \tau(s)$ and
– if there exists $s' \in D$ with $s' \prec_{\mathrm{ns}} s$, either $\varphi \in \tau(s)$ or $\varphi\mathbf{I}_{\mathrm{ns}}\varphi' \in \tau(s')$.

Lemma 1. *Given a* $\mathrm{TL}^{\mathrm{tree}}$ *formula* φ *over* Σ *and an unranked* Σ*-labeled tree* $T = (D, \lambda)$*, if* $\tau : D \to 2^{cl(\varphi)}$ *is a closure labeling of* T *satisfying conditions* (1)–(4)*, then* τ *is a valid labeling.*

Proof. We prove the statement by induction on the structure of formulae occurring in the labeling τ. The base case is that of atomic formulae: if $\top \in \tau(s)$ for some $s \in D$, then clearly $(T, s) \models \top$. Moreover, for each $a \in \Sigma$, if $a \in \tau(s)$ (resp., $\neg a \in \tau(s)$), then by rule (2), $(T, s) \models a$ (resp., $(T, s) \models \neg a$).

Now we consider non-atomic formulae, and we assume that for each of their subformulae φ, if $\varphi \in \tau(s)$, then $(T, s) \models \varphi$.

If $\varphi \vee \varphi' \in \tau(s)$ (or $\varphi \wedge \varphi' \in \tau(s)$), then by rule (1b) (resp., rule (1c)), either $\varphi \in \tau(s)$ or $\varphi' \in \tau(s)$ (resp., both $\varphi \in \tau(s)$ and $\varphi' \in \tau(s)$). By the induction hypothesis, this implies that either $(S, T) \models \varphi$ or $(S, T) \models \varphi'$ (resp., $(S, T) \models \varphi$ and $(S, T) \models \varphi'$). Therefore $(S, T) \models \varphi \vee \varphi'$ (resp., $(S, T) \models \varphi \wedge \varphi'$).

If $\mathbf{X}_{\mathrm{ch}}\varphi \in \tau(s)$ then, by rule (3a), $\varphi \in \tau(s \cdot i)$, for some i. By the induction hypothesis, $(T, s \cdot i) \models \varphi$ and then $(T, s) \models \mathbf{X}_{\mathrm{ch}}\varphi$. The same holds for $\mathbf{X}_{\mathrm{ns}}\varphi$ using rule (4a).

Similarly if $\mathbf{X}_{\mathrm{ch}}^{\forall}\varphi \in \tau(s)$ then, by rule (3b), $\varphi \in \tau(s \cdot i)$, for all i such that $s \cdot i \in D$. By the induction hypothesis, $(T, s \cdot i) \models \varphi$ and then $(T, s) \models \mathbf{X}_{\mathrm{ch}}^{\forall}\varphi$. The same holds for $\mathbf{X}_{\mathrm{ns}}^{\forall}\varphi$ using rule (4b).

If $\mathbf{X}_{\mathrm{ch}}^{-}\varphi \in \tau(s)$, then by rule (3c) and by the induction hypothesis, $s = s' \cdot i$, for some i and some $s' \in D$, and $(T, s') \models \varphi$, whence $(T, s) \models \mathbf{X}_{\mathrm{ch}}^{-}\varphi$. The same reasoning can be applied to the formula $\mathbf{X}_{\mathrm{ns}}^{-}\varphi$, using rule (4c).

If $\mathbf{X}_{\mathrm{ch}}^{-\forall}\varphi \in \tau(s)$, then by rule (3d) and by the induction hypothesis, we know that if there exist $s' \in D$ such that $s' \prec_{\mathrm{ch}} s$, then $(T, s') \models \varphi$, whence $(T, s) \models \mathbf{X}_{\mathrm{ch}}^{-\forall}\varphi$. The same reasoning can be applied to the formula $\mathbf{X}_{\mathrm{ns}}^{-\forall}\varphi$, using rule (4d).

If $\varphi\mathbf{U}_{\mathrm{ch}}\varphi' \in \tau(s)$, by successive application of rule (3e), there exists s' with $s \prec_{\mathrm{ch}}^{*} s'$ such that $\varphi' \in \tau(s')$ and, for each s'' with $s \prec_{\mathrm{ch}}^{*} s'' \prec_{\mathrm{ch}}^{*} s'$ and $s'' \neq s'$, we have that $\varphi \in \tau(s'')$. This implies, by the induction hypothesis, that $(T, s) \models \varphi\mathbf{U}_{\mathrm{ch}}\varphi'$. We proceed similarly for the formula $\varphi\mathbf{S}_{\mathrm{ch}}\varphi'$ (using rule (3f)), for the formula $\varphi\mathbf{U}_{\mathrm{ns}}\varphi'$ (using rule (4e)) and for the formula $\varphi\mathbf{S}_{\mathrm{ns}}\varphi'$ (using rule (4f)).

Now assume $\varphi\mathbf{R}_{\mathrm{ch}}\varphi' \in \tau(s)$. For each s' with $s \prec_{\mathrm{ch}}^{*} s'$, if $(T, s') \not\models \varphi'$, by the induction hypothesis, $\varphi' \notin \tau(s')$. Then, by successive application of rule (3g), there exists s'', with $s \prec_{\mathrm{ch}}^{*} s'' \prec_{\mathrm{ch}}^{*} s'$ and $s'' \neq s'$, such that $\varphi \in \tau(s'')$. Hence, by the induction hypothesis, $(T, s'') \models \varphi$. Since this holds for all descendant of s where φ' is not satisfied, we conclude that $(T, s) \models \varphi\mathbf{R}_{\mathrm{ch}}\varphi'$. Similarly we can prove the satisfaction of formulae of the form $\varphi\mathbf{R}_{\mathrm{ns}}\varphi'$ (by successive application of rule (4g)), formulae of the form $\varphi\mathbf{I}_{\mathrm{ch}}\varphi'$ (by rule (3h)), and formulae of the form $\varphi\mathbf{I}_{\mathrm{ns}}\varphi'$ (by rule (4h)). $\square$

A valid labeling τ of a tree T with a formula φ is *maximal* if, for each formula $\psi \in cl(\varphi)$ and for each node s of T, we have that $\psi \in \tau(s)$ if and only if $(T, s) \models \psi$.

Query automaton. For a given $\mathrm{TL}^{\mathrm{tree}}$ formula φ_0 over alphabet Σ, we now construct a query automaton $\mathcal{QA}_{\varphi_0} = (Q, F, Q_s, \delta)$ whose accepting runs on a Σ-labeled tree T compute a maximal closure labeling of T with φ_0. Intuitively the states of the automaton correspond to subsets of $cl(\varphi_0)$ and the accepting runs enforce conditions (1)–(4) to guarantee validity. Maximality is guaranteed by restricting the states of the automaton to maximally consistent subsets of $cl(\varphi_0)$.

In particular, the set of states $Q \subseteq 2^{cl(\varphi_0)}$ consists of all the subsets $q \subseteq cl(\varphi_0)$ such that:

- for each $\psi \in cl(\varphi_0)$, either $\psi \in q$ or $\neg\psi \in q$, but not both;
- $\bot \notin q$;
- if $\varphi \vee \varphi' \in q$, then either $\varphi \in q$ or $\varphi' \in q$;
- if $\varphi \wedge \varphi' \in q$, then both $\varphi \in q$ and $\varphi' \in q$.

The set of final states F consists of all $q_0 \in Q$ such that:

- q_0 does not contain formulae of the form $\mathbf{X}^-_{\mathrm{ch}}\varphi$; or $\mathbf{X}_{\mathrm{ns}}\varphi$ or $\mathbf{X}^-_{\mathrm{ns}}\varphi$;
- if q_0 contains a formula of the form $\varphi\mathbf{S}_{\mathrm{ch}}\varphi'$ or $\varphi\mathbf{I}_{\mathrm{ch}}\varphi'$ or $\varphi\mathbf{S}_{\mathrm{ns}}\varphi'$ or $\varphi\mathbf{I}_{\mathrm{ns}}\varphi'$ or $\varphi\mathbf{U}_{\mathrm{ns}}\varphi'$ or $\varphi\mathbf{R}_{\mathrm{ns}}\varphi'$, then q_0 also contains φ'.

For each formula $\varphi \in cl(\varphi_0)$, we let Q_φ be the set of states $q \in Q$ such that $\varphi \in q$. Then the selecting states are $Q_s = Q_{\varphi_0}$. The transition function $\delta : Q \times \Sigma \to 2^{Q^*}$ is defined as follows. For each $q \in Q$ and $a \in \Sigma$:

$$\delta(q,a) = \bigcap_{\psi \in q} L(\psi, q, a) \cap L^-(q) \cap L_{\mathrm{ns}}$$

where:

- $L(\psi, q, a)$ is the contribution of the formula $\psi \in q$ to $\delta(q,a)$, it enforces "future" vertical conditions and is defined as follows, depending on ψ:
  - if $\psi = a'$ with $a' \in \Sigma$ and $a' \neq a$, or if $\psi = \neg a$, then $L(\psi, q, a) = \emptyset$;
  - if $\psi = \mathbf{X}_{\mathrm{ch}}\varphi$ then $L(\psi, q, a) = Q^* Q_\varphi Q^*$;
  - if $\psi = \mathbf{X}^{\forall}_{\mathrm{ch}}\varphi$ then $L(\psi, q, a) = Q^*_\varphi$;
  - if $\psi = \varphi\mathbf{U}_{\mathrm{ch}}\varphi'$ then
  $$L(\psi,q,a) = \begin{cases} Q^* & \text{if } \varphi' \in q \\ \emptyset & \text{if } \varphi' \notin q \text{ and } \varphi \notin q \\ Q^* Q_\psi Q^* & \text{if } \varphi' \notin q \text{ and } \varphi \in q; \end{cases}$$
  - if $\psi = \varphi\mathbf{R}_{\mathrm{ch}}\varphi'$ then
  $$L(\psi,q,a) = \begin{cases} \emptyset & \text{if } \varphi' \notin q \\ Q^* & \text{if } \varphi' \in q \text{ and } \varphi \in q \\ Q^*_\psi & \text{if } \varphi' \in q \text{ and } \varphi \notin q; \end{cases}$$
  - for all other formulae, $L(\psi, q, a) = Q^*$.
- $L^-(q)$ enforces "past" vertical conditions and is defined as follows:
  $$L^-(q) = Q^-(q)^*$$
  where $Q^-(q)$ is the set of states q' satisfying all of the following conditions with q:
  - if q' contains $\mathbf{X}^-_{\mathrm{ch}}\varphi$ or $\mathbf{X}^{-\forall}_{\mathrm{ch}}\varphi$, then $\varphi \in q$;
  - if q' contains $\varphi\mathbf{S}_{\mathrm{ch}}\varphi'$, then either $\varphi' \in q'$, or $\varphi \in q'$ and $\varphi\mathbf{S}_{\mathrm{ch}}\varphi' \in q$;
  - if q' contains $\varphi\mathbf{I}_{\mathrm{ch}}\varphi'$, then $\varphi' \in q'$ and either $\varphi \in q'$ or $\varphi\mathbf{I}_{\mathrm{ch}}\varphi' \in q$.
- L_{ns} enforces horizontal conditions. It is a language over alphabet Q which simply enforces consistency of formula assignments. Since its alphabet symbols are already sets of formulae, this can be enforced by a DFA $\mathcal{A}_{\mathrm{ns}} = (Q_{\mathrm{ns}}, Q, q_0, Q_F, \delta_{\mathrm{ns}})$, having:
  - set of states $Q_{\mathrm{ns}} = \{q_0\} \cup Q$, where q_0 is a distinguished initial state, not occurring in Q;
  - set of final states Q_F containing precisely q_0 and all states $q_f \in Q$ such that:
    (a) q_f does not contain formulae of the form $\mathbf{X}_{\mathrm{ns}}\varphi$;
    (b) if $\varphi\mathbf{U}_{\mathrm{ns}}\varphi'$ is in q_f or $\varphi\mathbf{R}_{\mathrm{ns}}\varphi'$ is in q_f, then also φ' is in q_f;
  - transition function $\delta_{\mathrm{ns}} : Q_{\mathrm{ns}} \times Q \to Q_{\mathrm{ns}}$ defined as follows. For all states $q' \in Q_{\mathrm{ns}}$ and symbols $\tilde{q} \in Q$, we have that $q' = \delta_{\mathrm{ns}}(q_0, \tilde{q})$ iff $q' = \tilde{q}$ and both the following conditions hold:
    (a) q' does not contain formulae of the form $\mathbf{X}^-_{\mathrm{ns}}\varphi$;
    (b) if $\varphi\mathbf{S}_{\mathrm{ns}}\varphi' \in q'$ or $\varphi\mathbf{I}_{\mathrm{ns}}\varphi' \in q'$ then $\varphi' \in q'$.
    For all states q, q' in Q_{ns} and symbols $\tilde{q} \in Q$, with $q \neq q_0$, we have that $q' = \delta_{\mathrm{ns}}(q, \tilde{q})$ iff $q' = \tilde{q}$ and the following holds:

(a) if $\mathbf{X}_{\mathrm{ns}}\varphi \in q$ or $\mathbf{X}^{\forall}_{\mathrm{ns}}\varphi \in q$, then $\varphi \in q'$;
(b) if $\mathbf{X}^{-}_{\mathrm{ns}}\varphi \in q'$ or $\mathbf{X}^{-\forall}_{\mathrm{ns}}\varphi \in q'$, then $\varphi \in q$;
(c) if $\varphi\mathbf{U}_{\mathrm{ns}}\varphi' \in q$, then either $\varphi' \in q$ or $\varphi \in q$ and $\varphi\mathbf{U}_{\mathrm{ns}}\varphi' \in q'$;
(d) if $\varphi\mathbf{S}_{\mathrm{ns}}\varphi' \in q'$, then either $\varphi' \in q'$ or $\varphi \in q'$ and $\varphi\mathbf{S}_{\mathrm{ns}}\varphi' \in q$;
(e) if $\varphi\mathbf{R}_{\mathrm{ns}}\varphi' \in q$ then $\varphi' \in q$ and either $\varphi \in q$ or $\varphi\mathbf{R}_{\mathrm{ns}}\varphi' \in q'$;
(f) if $\varphi\mathbf{I}_{\mathrm{ns}}\varphi' \in q'$, then $\varphi' \in q'$ and either $\varphi \in q'$ or $\varphi\mathbf{I}_{\mathrm{ns}}\varphi' \in q$.

Lemma 2. *ρ is an accepting run of $\mathcal{QA}_{\varphi_0}$ on a tree T iff ρ is a maximal valid labeling of T with φ_0.*

Proof. Assume ρ is a maximal valid labeling of T with φ_0, then directly by maximality and by the semantics of $\mathrm{TL}^{\mathrm{tree}}$ formulae, ρ satisfies conditions (1)–(4). Maximality also implies that for each $s \in T$ the set of formulae $\rho(s)$ is a maximally consistent subset of $cl(\varphi_0)$. Together with conditions (1) for ρ, this implies that for each node $s \in T$, the set $\rho(s)$ is a state of $\mathcal{QA}_{\varphi_0}$.

Now we prove that ρ is an accepting run, that is: (1) $\rho(\varepsilon) \in F$ and (2) for each node s, with $n \geqslant 0$ children, $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1)) \in \delta(\rho(s), \lambda(s))$.

Conditions (3c), (3f), (3h), and (4), satisfied by ρ in $s = \varepsilon$, imply directly the properties defining F. Thus $\rho(\varepsilon) \in F$. We next prove that for all nodes s, with $n \geqslant 0$ children:

(A) the sequence of states $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1))$ belongs to $L(\psi, \rho(s), \lambda(s))$, for all $\psi \in \rho(s)$,
(B) $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1)) \in L^{-}(\rho(s))$ and
(C) $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1)) \in L_{\mathrm{ns}}$.

These will prove $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1)) \in \delta(\rho(s), \lambda(s))$.

(A) For each formula ψ of the form a' or $\neg a'$ (with $a' \in \Sigma$) in $\rho(s)$, by condition (2), $L(\psi, \rho(s), \lambda(s)) = Q^*$. Then $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1)) \in L(\psi, \rho(s), \lambda(s))$.
For each formula of the form $\mathbf{X}_{\mathrm{ch}}\varphi$ in $\rho(s)$, condition (3a) for ρ implies that, for some i, the label $\rho(s\cdot i)$ is in Q_φ. Hence $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1))$ belongs to $Q^*Q_\varphi Q^* = L(\mathbf{X}_{\mathrm{ch}}\varphi, \rho(s), \lambda(s))$.
For each formula of the form $\mathbf{X}^{\forall}_{\mathrm{ch}}\varphi$ in $\rho(s)$, condition (3b) for ρ implies that, for all i such that $s\cdot i \in D$, the label $\rho(s\cdot i)$ is in Q_φ. Hence $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1))$ belongs to $Q^*_\varphi = L(\mathbf{X}^{\forall}_{\mathrm{ch}}\varphi, \rho(s), \lambda(s))$.
For each formula of the form $\varphi\mathbf{U}_{\mathrm{ch}}\varphi'$ in $\rho(s)$, by condition (3e), there are two cases:
- if $\varphi' \in \rho(s)$, then $L(\varphi\mathbf{U}_{\mathrm{ch}}\varphi', \rho(s), \lambda(s)) = Q^*$, therefore it contains $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1))$;
- otherwise $\varphi \in \rho(s)$ and, for some i, $\rho(s\cdot i) \in Q_{\varphi\mathbf{U}_{\mathrm{ch}}\varphi'}$. Hence $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1))$ belongs to $Q^*Q_{\varphi\mathbf{U}_{\mathrm{ch}}\varphi'}Q^*$. This language coincides with $L(\varphi\mathbf{U}_{\mathrm{ch}}\varphi', \rho(s), \lambda(s))$ (since $\varphi' \notin \rho(s)$ and $\varphi \in \rho(s)$).

For each formula of the form $\varphi\mathbf{R}_{\mathrm{ch}}\varphi' \in \rho(s)$, condition (3g) satisfied by ρ implies that $L(\varphi\mathbf{R}_{\mathrm{ch}}\varphi', \rho(s), \lambda(s))$ is not empty. Moreover we have two cases:
- if $\varphi \in \rho(s)$, then $L(\varphi\mathbf{R}_{\mathrm{ch}}\varphi', \rho(s), \lambda(s)) = Q^*$, therefore it contains $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1))$;
- otherwise, for each i such that $s\cdot i \in D$, the state $\rho(s\cdot i)$ is in $Q_{\varphi\mathbf{R}_{\mathrm{ch}}\varphi'}$. Hence $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1)) \in Q^*_{\varphi\mathbf{R}_{\mathrm{ch}}\varphi'}$. On the other hand $Q^*_{\varphi\mathbf{R}_{\mathrm{ch}}\varphi'}$ coincides with $L(\varphi\mathbf{R}_{\mathrm{ch}}\varphi', \rho(s), \lambda(s))$, since in this case, $\varphi' \in \rho(s)$ and $\varphi \notin \rho(s)$.

For all other formulae ψ in $\rho(s)$, the language $L(\psi, \rho(s), \lambda(s))$ is Q^*, thus membership is trivial.
(B) Conditions (3c), (3d), (3f) and (3h) satisfied by ρ on all nodes $s\cdot i$, for $i = 0, \dots, n-1$, directly imply that $\rho(s\cdot i)$ belongs to $Q^{-}(\rho(s))$, for all i. Therefore $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1))$ belongs to $Q^{-}(\rho(s))^*$, that is to $L^{-}(\rho(s))$.
(C) If s is a leaf, its sequence of children is ε and $\varepsilon \in L_{\mathrm{ns}}$ (since in $\mathcal{A}_{\mathrm{ns}}$ the initial state is also final).
If s is not a leaf, condition (4) is satisfied in $s\cdot i$, for all i. This implies that $\mathcal{A}_{\mathrm{ns}}$ accepts $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1))$. Indeed, conditions (4c), (4f) and (4h) satisfied by ρ in the node $s\cdot 0$ imply, by definition of $\mathcal{A}_{\mathrm{ns}}$, that $\delta_{\mathrm{ns}}(q_0, \rho(s\cdot 0)) = \rho(s\cdot 0)$. Similarly $\delta_{\mathrm{ns}}(\rho(s\cdot i), \rho(s\cdot(i+1))) = \rho(s\cdot(i+1))$, for $i = 0, \dots, n-2$ by conditions (4a), (4b), (4e) and (4g) on node $s\cdot i$, and conditions (4c), (4e), (4f) and (4h) on node $s\cdot(i+1)$. Finally $\rho(s\cdot(n-1))$ is a final state for $\mathcal{A}_{\mathrm{ns}}$, by conditions (4a), (4e) and (4g) on node $s\cdot(n-1)$. This shows that there exists an accepting run of $\mathcal{A}_{\mathrm{ns}}$ on $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1))$.

Properties (A), (B), (C), proved above imply $\rho(s\cdot 0), \dots, \rho(s\cdot(n-1)) \in \delta(\rho(s), \lambda(s))$, for all nodes s, and prove that ρ is an accepting run. This completes the proof of one direction.

Conversely, assume that ρ is an accepting run of $\mathcal{QA}_{\varphi_0}$ on T. We first prove that ρ is closure labeling satisfying conditions (1)–(4) (stated in the beginning of the proof), and thus, by Lemma 1, a valid labeling. Maximality will follow from the fact that states are maximally consistent subsets of $cl(\varphi_0)$.

Conditions (1) are satisfied for each node s, by the definition of states of $\mathcal{QA}_{\varphi_0}$.

Condition (2) is satisfied since ρ is accepting and then, for each node s, the set $\delta(\rho(s), \lambda(s))$ is non-empty. Therefore, by definition of δ, for each $a \in \Sigma$, if a (or $\neg a$) is in $\rho(s)$, the language $L(a, q, \lambda(s))$ (resp., $L(\neg a, q, \lambda(s))$) must be non-empty. This directly implies condition (2) for ρ.

We now prove that ρ satisfies condition (3). For each node s of T with $n \geqslant 0$ children:

- If $\mathbf{X}_{\mathrm{ch}}\varphi \in \rho(s)$, by definition of $\delta(\rho(s), \lambda(s))$, and since ρ is accepting, the sequence $\rho(s\cdot 0), \ldots, \rho(s\cdot(n-1))$ belongs to the language $L(\mathbf{X}_{\mathrm{ch}}\varphi, \rho(s), \lambda(s)) = Q^* Q_\varphi Q^*$. Hence $\varphi \in \rho(s\cdot i)$, for some i. This proves condition (3a) for ρ in s.
- Similarly, if $\mathbf{X}^{\forall}_{\mathrm{ch}}\varphi \in \rho(s)$, by definition of $\delta(\rho(s), \lambda(s))$, and since ρ is accepting, the sequence $\rho(s\cdot 0), \ldots, \rho(s\cdot(n-1))$ belongs to the language $L(\mathbf{X}^{\forall}_{\mathrm{ch}}\varphi, \rho(s), \lambda(s)) = Q^*_\varphi$. Hence $\varphi \in \rho(s\cdot i)$, for each i such that $s\cdot i \in D$. This proves condition (3b) for ρ in s.
- If $\varphi\mathbf{U}_{\mathrm{ch}}\varphi' \in \rho(s)$, again by definition of δ, the sequence $\rho(s\cdot 0), \ldots, \rho(s\cdot(n-1))$ belongs to the language $L(\varphi\mathbf{U}_{\mathrm{ch}}\varphi', \rho(s))$, which is then not empty. Therefore, by definition of $L(\varphi\mathbf{U}_{\mathrm{ch}}\varphi', \rho(s), \lambda(s))$, one of the following holds:
  - either $\varphi' \in \rho(s)$ or
  - $\varphi \in \rho(s)$ and $\rho(s\cdot 0), \ldots, \rho(s\cdot(n-1)) \in Q^* Q_{\varphi\mathbf{U}_{\mathrm{ch}}\varphi'} Q^*$. Hence $\varphi\mathbf{U}_{\mathrm{ch}}\varphi' \in \rho(s\cdot i)$, for some i.

  This proves that ρ satisfies condition (3e) in s.
- Similarly, if $\varphi\mathbf{R}_{\mathrm{ch}}\varphi' \in \rho(s)$, then $\rho(s\cdot 0), \ldots, \rho(s\cdot(n-1))$ belongs to the language $L(\varphi\mathbf{R}_{\mathrm{ch}}\varphi', \rho(s), \lambda(s))$, which is therefore not empty. This implies that:
  - $\varphi' \in \rho(s)$ and
  - either $\varphi \in \rho(s)$ or $\rho(s\cdot 0), \ldots, \rho(s\cdot(n-1)) \in Q^*_{\varphi\mathbf{R}_{\mathrm{ch}}\varphi'}$. In this last case $\varphi\mathbf{R}_{\mathrm{ch}}\varphi' \in \rho(s\cdot i)$, for all i.

  Thus ρ satisfies condition (3g) in s.

On the whole, this shows that conditions (3a), (3b), (3e) and (3g) hold for all nodes. Moreover, since ρ is an accepting run, we also know that for each node s, the sequence of states $\rho(s\cdot 0), \ldots, \rho(s\cdot(n-1))$ belongs to the language $L^-(\rho(s))$. So for each i such that $s\cdot i \in D$, the state $\rho(s\cdot i)$ is in $Q^-(\rho(s))$. By definition of Q^-, this implies conditions (3c), (3d), (3f) and (3h) on $s\cdot i$. On the other hand, conditions (3c), (3d), (3f) and (3h) on the root ε are directly implied by the fact that $\rho(\varepsilon)$ is a final state of $\mathcal{QA}_{\varphi_0}$. This proves that ρ satisfies all conditions (3) on all nodes of T.

We now prove that ρ satisfies conditions (4). Conditions (4) in the root are again implied by the fact that $\rho(\varepsilon)$ is a final state of $\mathcal{QA}_{\varphi_0}$. For all non-root nodes, ρ satisfies (4), thanks to the constraints enforced by the accepting runs of $\mathcal{A}_{\mathrm{ns}}$. In particular, for each node of the form $s\cdot i$, let $s\cdot 0, \ldots, s\cdot(n-1)$ be the sequence of children of s. We know that $\rho(s\cdot 0), \ldots, \rho(s\cdot(n-1))$ is accepted by $\mathcal{A}_{\mathrm{ns}}$, therefore:

- $\delta_{\mathrm{ns}}(q_0, \rho(s\cdot 0)) = \rho(s\cdot 0)$. By definition of δ_{ns}, this implies that ρ satisfies conditions (4c), (4d), (4f) and (4h) on the node $s\cdot 0$.
- $\delta_{\mathrm{ns}}(\rho(s\cdot i), \rho(s\cdot i+1)) = \rho(s\cdot i+1)$, for all $i = 0, \ldots, n-2$. This implies, by definition of δ_{ns}, that condition (4a), (4b), (4e) and (4g) are satisfied on $s\cdot i$, and conditions (4c), (4d), (4f) and (4h) are satisfied on $s\cdot i+1$.
- $\rho(s\cdot(n-1))$ is a final state of $\mathcal{A}_{\mathrm{ns}}$. This directly implies that conditions (4a), (4b), (4e) and (4g) are satisfied by ρ on the node $s\cdot(n-1)$.

As a consequence, on the whole, all conditions (4a)–(4h) are satisfied in $s\cdot i$ for each i. Thus conditions (4) are satisfied also on all non-root nodes.

We conclude that ρ satisfies all conditions (1)–(4) on all nodes. Therefore, by Lemma 1, ρ is a valid labeling. Now observe that for each node s, the state $\rho(s)$ is a maximally consistent subsets of $cl(\varphi_0)$. Therefore, for each formula $\psi \in cl(\varphi_0)$, and for each node s, if $\psi \in \rho(s)$, then $(T, s) \models \psi$; on the contrary, if $\psi \notin \rho(s)$ then $\neg\psi \in \rho(s)$, implying $(T, s) \models \neg\psi$. This proves maximality of ρ and concludes the proof the lemma. □

Based on Lemma 2, it is straightforward to prove that $\mathcal{QA}_{\varphi_0}$ is the desired automaton:

Lemma 3. *$\mathcal{QA}_{\varphi_0}$ is a single-run query automaton and it computes φ_0. That is $\mathcal{QA}_{\varphi_0}(T) = \{s \in T \mid (T, s) \models \varphi_0\}$, for each Σ-labeled tree T.*

Proof. By Lemma 2, there exists exactly one accepting run ρ of QA_{φ_0} on each tree T, corresponding to the maximal valid labeling of T with φ_0. Thus the single run conditions trivially hold. Moreover, by maximality of the labeling ρ, for each node s we have that $\varphi_0 \in \rho(s)$ (that is $\rho(s) \in Q_{\varphi_0}$) iff $(T, s) \models \varphi_0$. Since the selecting states of $\mathcal{QA}_{\varphi_0}$ are Q_{φ_0}, this implies that the accepting run ρ selects precisely the nodes where φ_0 holds. That is $QA_{\varphi_0}(T) = \{s \in T \mid (T, s) \models \varphi_0\}$. □

Size of the query automaton and complexity of the construction. We now show that, if m is the size of $cl(\varphi_0)$, then $\mathcal{QA}_{\varphi_0}$ has size bounded by $2^{O(m)}$ and can be constructed from input φ_0 in time $2^{O(m)}$.

First notice that $|cl(\varphi_0)|$ is bounded by $|\varphi_0|$, since the number of subformulae of φ_0 is bounded by the number of nodes in its parse tree. Since different nodes of a parse tree can represent the same subformula, the number of distinct subformulae of φ_0 can be much smaller than its size. In particular we also have that $|\varphi_0|$ is bounded by $2^{O(m)}$.

Clearly the set of states of $\mathcal{QA}_{\varphi_0}$ has cardinality $|Q| \leqslant 2^m$. They can be computed from the input formula φ_0 by first computing $cl(\varphi_0)$ and then checking, on each subset of the closure, the local conditions defining the states of $\mathcal{QA}_{\varphi_0}$.

The closure can be computed in time polynomial in $|\varphi_0|$ (therefore in time $2^{O(m)}$) by identifying distinct subtrees in the parse trees of φ_0 and $\neg\varphi_0$. Moreover all subsets of the closure are computed in time $2^{O(m)}$.

Now conditions have to be checked on each subset of the closure in order to identify states, selecting states and final states. Checking all the required local conditions on a subset of the closure can be always done in polynomial time in m. This implies that states, selecting states and final states can be computed in time $2^{O(m)}$. Similarly one can compute states Q_φ for all $\varphi \in cl(\varphi_0)$ as well as states $Q^-(q)$, for all $q \in Q$, states $Q_{\rm ns}$ and Q_F, in time $2^{O(m)}$.

We now analyze the size and construction of the transition function. For each $q \in Q$ and $a \in \Sigma$, the automaton $\delta(q,a)$ can be obtained as the product of:

- $\mathcal{A}_{\rm ns}$ recognizing $L_{\rm ns}$;
- a DFA recognizing $L(\psi, q, a)$, for all $\psi \in q$;
- a DFA recognizing $L^-(q)$.

There exist DFAs with at most two states that recognize $L^-(q)$ and $L(\psi,q,a)$ (for all ψ, a and q). Computing an automaton for $L^-(q)$, as well as for $L(\psi,q,a)$, for a given q and $\psi \in q$, requires at most checking a local condition on the state q in time linear in $|q| \leqslant m$, and then producing a transition function from a constant number of states and alphabet Q. This can be clearly done in time $2^{O(m)}$. The construction of $\mathcal{A}_{\rm ns}$, requires the computation of a transition function of size $|Q|^3$. Each of its elements is computed in time polynomial in m, therefore the time bound for computing $\mathcal{A}_{\rm ns}$ is still $2^{O(m)}$.

The product of all the automata for $\delta(q,a)$ has number of states bounded by $2^{|q|} \times Q_{\rm ns}$ (where $Q_{\rm ns}$ is the number of states of $\mathcal{A}_{\rm ns}$). It can be computed in time polynomial in the number of states of $\delta(q,a)$ and in $|Q|$, that is still in time $2^{O(m)}$. Therefore the overall computation of $\mathcal{QA}_{\varphi_0}$ can be done in time $2^{O(m)}$.

The size $\|\mathcal{QA}_{\varphi_0}\|$ by definition is $|Q| + \sum_{q\in Q, a\in\Sigma} \|\delta(q,a)\|$. If $S_{q,a}$ denotes the number of states of $\delta(q,a)$, we have $\|\delta(q,a)\| \leqslant |S_{q,a}| + |S_{q,a}|^2 \times |Q|$. Since $|S_{q,a}|$ and $|Q|$ are bounded by $2^{O(m)}$, so are $\|\delta(q,a)\|$ and $\|\mathcal{QA}_{\varphi_0}\|$.

Because m is bounded by $|\varphi_0|$, this concludes the proof of the theorem. □

We now give an example of the automaton construction.

Example. Let $\varphi = \top\mathbf{U}_{\rm ch}a$ be a $\mathrm{TL}^{\rm tree}$ formula over alphabet $\Sigma = \{a, b\}$. We now construct a query automaton $\mathcal{QA}_\varphi = (Q, F, Q_s, \delta)$ equivalent to φ over Σ-labeled trees. The Fischer–Ladner closure of φ is

$$cl(\varphi) = \{\top, a, \top\mathbf{U}_{\rm ch}a, \bot, \neg a, \bot\mathbf{R}_{\rm ch}\neg a\}$$

and the set of states Q are all the maximally consistent subsets of $cl(\varphi)$ which do not contain $\bot$ (because no formula of $cl(\varphi)$ contains boolean connectives), that is:

$$Q = \{q_1, q_2, q_3, q_4\}$$

with

$$q_1 = \{\top, a, \top\mathbf{U}_{\rm ch}a\}$$

$$q_2 = \{\top, \neg a, \top\mathbf{U}_{\rm ch}a\}$$

$$q_3 = \{\top, a, \bot\mathbf{R}_{\rm ch}\neg a\}$$

$$q_4 = \{\top, \neg a, \bot\mathbf{R}_{\rm ch}\neg a\}.$$

All states are final ($F = Q$) because no state contains formulae with past or horizontal temporal connectives. The selecting states are the ones containing φ, that is $Q_s = \{q_1, q_2\}$.

We now compute the transition function δ. Note that, since $cl(\varphi)$ contains no formulae with horizontal temporal connectives, the language $L_{\rm ns}$ (enforcing horizontal conditions on states associated to siblings in a run of $\mathcal{QA}_\varphi$) is Q^*. Similarly, $cl(\varphi)$ contains no formulae with past temporal connectives. Therefore the language $L^-(q_i)$ (enforcing past conditions on states associated to parent-child pairs in a run of $\mathcal{QA}_\varphi$) is also Q^*, for all q_i. It follows that for each state $q_i \in Q$ and each symbol $s \in \Sigma$, the transition function $\delta(q_i, s)$ is a language obtained by intersecting the contribution ($L(\psi, q_i, s)$) of each formula $\psi \in q_i$. Then δ is as follows:

- $\delta(q_1, a) = Q^*$

because no formula in q_1 needs to enforce constraints on the states associated to the children of a node (that is $L(\psi, q_1, a) = Q^*$ for all $\psi \in q_1$);

- $\delta(q_2, b) = Q^*\{q_1, q_2\}Q^*$

because $\top \mathbf{U}_{\mathrm{ch}} a \in q_2$ but $a \notin q_1$; then δ enforces that the formula $\top \mathbf{U}_{\mathrm{ch}} a$ (contained only in q_1 and q_2) is propagated to at least one child (that is $L(\top \mathbf{U}_{\mathrm{ch}} a, q_2, b) = Q^*\{q_1, q_2\}Q^*$ and the contribution of all other formulae in q_2 is Q^*);

- $\delta(q_4, b) = \{q_3, q_4\}^*$

because $\bot \mathbf{R}_{\mathrm{ch}} \neg a \in q_4$ but $\bot \notin q_4$; therefore δ enforces that all children nodes are assigned a state containing $\bot \mathbf{R}_{\mathrm{ch}} \neg a$ (that is $L(\bot \mathbf{R}_{\mathrm{ch}} \neg a, q_4, b) = \{q_3, q_4\}^*$, while the contribution of all other formulae in q_4 is Q^*);

- $\delta(q_1, b) = \emptyset$

because the formula $a \in q_1$ cannot be satisfied in a node labeled b (that is $L(a, q_1, b) = \emptyset$);

- $\delta(q_2, a) = \emptyset$

because the formula $\neg a \in q_2$ cannot be satisfied in a node labeled a (that is $L(\neg a, q_2, a) = \emptyset$);

- $\delta(q_3, a) = \emptyset$

because q_3 contains the formula $\bot \mathbf{R}_{\mathrm{ch}} \neg a$ but not $\neg a$ (that is $L(\bot \mathbf{R}_{\mathrm{ch}} \neg a, q_3, a) = \emptyset$);

- $\delta(q_3, b) = \emptyset$

because $L(a, q_3, b) = \emptyset$;

- $\delta(q_4, a) = \emptyset$

because $L(\neg a, q_4, a) = \emptyset$.

Given δ, it is easy to verify that there is only one accepting run of $\mathcal{QA}_\varphi$ on a tree: this assigns state q_1 to all nodes labeled a, state q_2 to all nodes labeled b which have an a-labeled descendant, and state q_4 to all nodes labeled b having no a-labeled descendant. Thus nodes assigned either q_1 or q_2 (that is one of the selecting states) are precisely nodes where $\top \mathbf{U}_{\mathrm{ch}} a$ is satisfied. This shows that $\mathcal{QA}_\varphi$ is equivalent to φ.

7. Applications

7.1. Reasoning about document navigation

As mentioned in Section 2, typical XML static analysis tasks include consistency of schema and navigational properties (e.g., is a given XPath expression consistent with a given DTD?), or query optimization (e.g., is a given XPath expression e contained in a another expression e' for all trees that conform to a DTD d?). We now show two applications of our results for such analyses of XML specifications.

As a starter, let us see how XPath containment in the presence of DTDs can be checked. Suppose we want to check whether $d \models e_1 \subseteq e_2$. Translate e_1 and e_2 into $\mathrm{TL}^{\mathrm{tree}}$ formulae e_1' and e_2', and construct $\mathcal{QA}_{e_1' \wedge \neg e_2'}$ – a query automaton for the formula $e_1' \wedge \neg e_2'$ which witnesses counterexamples to the containment. Let $\mathcal{A}_d$ be an automaton recognizing trees that conform to d. Then $\mathcal{A}_d \times \mathcal{QA}_{e_1' \wedge \neg e_2'}$ finds trees that conform to d and witness a counterexample to $e_1 \subseteq e_2$. Hence, $d \models e_1 \subseteq e_2$ iff the language of $\mathcal{A}_d \times \mathcal{QA}_{e_1' \wedge \neg e_2'}$ is empty. The size of the product QA is $\|d\| \cdot 2^{O(\|e_1\| + \|e_2\|)}$, i.e., this is precisely the construction that was promised in Section 2. Moreover, the query automaton $\mathcal{A}_d \times \mathcal{QA}_{e_1' \wedge \neg e_2'}$ describes exactly the counterexamples to containment: it selects nodes s from trees T that conform to d such that s is selected by e_1 but not by e_2.

Since we know how to construct query automata for *arbitrary* $\mathrm{TL}^{\mathrm{tree}}$ formulae, this simple idea works not only for containment of two expressions but for more complex satisfiability and containment conditions. We give two examples below.

7.1.1. Satisfiability algorithms for sets of XPath expressions

The exponential-time complexity for satisfiability of XPath expressions in the presence of a schema is already known [27, 7]. We now show how we can verify satisfiability of multiple sets of XPath expressions, in a uniform way, using translation into query automata.

Given an arbitrary set $E = \{e_1, \dots, e_n\}$ of XPath (core or conditional) expressions and a subset $E' \subseteq E$, let $\mathcal{Q}(E')$ be a unary query defining the intersection of queries given by all the $e \in E'$. That is, $\mathcal{Q}(E')$ selects nodes that satisfy every expression $e \in E'$. We can capture *all* (exponentially many) such queries $\mathcal{Q}(E')$s by a single automaton, that is instantiated into different QAs by different selecting states.

Proposition 1. *One can construct, in time $2^{O(\|E\|)}$ (that is, $2^{O(\|e_1\|+\cdots+\|e_n\|)}$), an unranked tree automaton $\mathcal{A}(E) = (Q, F, \delta)$ and a relation $\sigma \subseteq E \times Q$ such that, for every $E' \subseteq E$,*

$$\mathcal{QA}_{E'} = \Big(Q, F, \bigcap\big\{\sigma(e) \mid e \in E'\big\}, \delta\Big)$$

is a single-run QA defining the unary query $\mathcal{Q}(E')$.

Proof. The construction of $\mathcal{QA}_{E'}$ simply takes the product of all the $\mathcal{QA}_{e'_i}$s, produced by Theorem 2, where e'_i is a $\mathrm{TL}^{\mathrm{tree}}$ translation of e_i, produced by Theorem 1. The relation σ relates tuples of states that include selecting states of $\mathcal{QA}_{e'_i}$ with $e_i \in E$. Then checking non-emptiness of $\mathcal{QA}_{E'}$, we see if all $e \in E'$ are simultaneously satisfiable. □

The containment problem for XPath expressions that we looked at before is a special case of the problem we consider. To check whether $d \models e_1 \subseteq e_2$, we construct $\mathcal{QA}_{\{e_1, \neg e_2\}}$ as in Proposition 1, and take the product of it with the automaton for d.

7.1.2. Verifying complex containment statements under DTDs

We can now extend the previous example and check not a single containment, as is usually done [37], but arbitrary Boolean combinations of XPath containment statements, without additional complexity. Assume that we are given a DTD d (or any other schema specification presented by an automaton), a set $\{e_1, \ldots, e_n\}$ of XPath expressions, and a Boolean combination $\mathcal{C}$ of inclusions $e_i \subseteq e_j$. We now want to check whether $d \models \mathcal{C}$, that is, whether $\mathcal{C}$ is true in every tree T that conforms to d. We shall refer to size of $\mathcal{C}$ as $\|\mathcal{C}\|$; the definition is extended in the natural way from the definition of $\|e\|$.

Theorem 3. *In the above setting, one can construct an unranked tree automaton of size $\|d\| \cdot 2^{O(\|\mathcal{C}\|)}$ whose language is empty iff $d \models \mathcal{C}$.*

This is achieved by replacing $e_i \subseteq e_j$ in $\mathcal{C}$ with the formula $\neg\mathbf{F}_{\mathrm{ch}}(e'_i \wedge \neg e'_j)$ and $e_i \not\subseteq e_j$ in $\mathcal{C}$ with the formula $\mathbf{F}_{\mathrm{ch}}(e'_i \wedge \neg e'_j)$, where e'_i, e'_j are $\mathrm{TL}^{\mathrm{tree}}$ translations of e_i and e_j produced by Theorem 1. Thus we can view $\mathcal{C}$ as a $\mathrm{TL}^{\mathrm{tree}}$ formula $\alpha_{\mathcal{C}}$. Now construct a QA for $\neg\alpha_{\mathcal{C}}$, by Theorem 2, and turn it into an automaton that checks whether the root gets selected. Now we take the product of this automaton with the automaton for d. The result accepts counterexamples to $\mathcal{C}$ under d, and the result follows. The construction of the automaton is polynomial-time in $\|d\|$ and single-exponential time in $\|\mathcal{C}\|$.

7.2. Reasoning about views

Recall the problem outlined in Section 2. We have a view definition given by a query automaton $\mathcal{QA}_{\mathcal{V}}$. For each source tree T, it selects a set of nodes $V = \mathcal{QA}_{\mathcal{V}}(T)$ which can also be viewed as a tree (we can assume, for example, that $\mathcal{QA}_{\mathcal{V}}$ always selects the root). Source trees are required to satisfy a schema constraint (e.g., a DTD). Since all schema formalisms for XML are various restrictions or reformulations of tree automata, we assume that the schema is given by an automaton $\mathcal{A}$.

If we only have access to V, we would like to be sure that secret information about an unknown source T is not revealed. This information, which we assume to be coded by a Boolean query $\mathfrak{Q}$, would be revealed by V if the answer to $\mathfrak{Q}$ were true in all source trees T that conform to the schema and generate V – that is, if $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_{\mathcal{V}}}(\mathfrak{Q}; V)$ were true. Thus, we would like to construct a new automaton $\mathcal{A}^*$ that accepts V iff $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_{\mathcal{V}}}(\mathfrak{Q}; V)$ is false, giving us some security assurances about the view.

In general, such an automaton construction is impossible: if $\mathcal{QA}_{\mathcal{V}}$ generates the yield of a tree, views essentially code context-free languages. Combining multiple CFLs with the help of DTDs, we get an undecidability result:

Proposition 2. *The problem of checking, for source and view schemas (automata) $\mathcal{A}_s$ and $\mathcal{A}_v$, a view definition $\mathcal{QA}_{\mathcal{V}}$, and a Boolean first-order query $\mathfrak{Q}$, whether there exists a view V accepted by the automaton $\mathcal{A}_v$ that satisfies $\underline{certain}^{\mathcal{A}_s}_{\mathcal{QA}_{\mathcal{V}}}(\mathfrak{Q}; V) = \mathrm{true}$, is undecidable.*

Proof. Consider an arbitrary context-free grammar G over alphabet Σ. It can of course be viewed as a DTD d_G, since DTDs are extended CFGs. Create a new DTD d with a new root r and productions $r \to g|ng$, where g is the root of d_G, and ng is a new symbol, as well as $ng \to \Sigma^*$. The view DTD is $r \to \Sigma^*$.

The view definition $\mathcal{Q}_{\mathcal{V}}$ selects the root and all the leaves labeled with symbols in Σ (i.e., skipping all the nodes labeled with nonterminals of G and ng). The query $\mathfrak{Q}$ is an existential FO query $\exists x P_{ng}(x)$ – i.e., it asks whether there exists a node labeled ng.

If we now have a view V (which is essentially a string $s_V \in \Sigma^*$, written left-to-right, at children of the root), then $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_\mathcal{V}}(\mathfrak{Q}; V)$ is true iff s_V is not in the language of G (otherwise there would be a source tree having g as the child of the root and thus no ng-labeled node). Hence decidability of $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_\mathcal{V}}(\mathfrak{Q}; V)$ being true would imply decidability of (non)universality of G – but the latter is of course undecidable.

Note that we can even assume wlog that we deal with ranked source trees, by putting the grammar in the Chomsky normal form. □

Schemas and queries required for this result are very simple, so to ensure the existence of the automaton $\mathcal{A}^*$, we need to put restrictions on the class of views. We assume that they are *upward-closed* as in [8]: if a node is selected, then so is the entire path to it from the root.

Note that the upward-closure $\mathcal{QA}_\uparrow$ of a query automaton $\mathcal{QA}$ can be obtained in linear time by adding a bit to the state indicating whether a selecting state has been seen and propagating it up. Thus, we shall assume without loss of generality that QAs defining views are *upward-closed*: if $s \in \mathcal{QA}(T)$ and s' is an ancestor of s, then $s' \in \mathcal{QA}(T)$.

The key observation that we need is that for an upward-closed QA, satisfying the single-run condition, its image is regular. Furthermore, it can be accepted by a small tree automaton:

Lemma 4. *Let $\mathcal{QA}$ be an upward-closed query automaton that satisfies condition* (1) *of the definition of single-run QAs. Then one can construct, in quadratic time, an unranked tree automaton $\mathcal{A}^*$ that accepts trees V for which there exists a tree T satisfying $V = \mathcal{QA}(T)$. Moreover, the number of states of $\mathcal{A}^*$ is at most the number of states of $\mathcal{QA}$.*

Proof. Since $\mathcal{QA}$ is upward-closed, for each tree $T = (D, \lambda)$, the answer $\mathcal{QA}(T)$ is a prefix-closed subset of D. In general, if D' is an arbitrary prefix-closed subset of D, the restriction of T (viewed as a logical structure) to domain D' is isomorphic to another tree structure which we denote by $T|_{D'}$, throughout the proof. Therefore if V is a Σ-labeled tree, the equality $\mathcal{QA}(T) = V$ will stand for $T|_{\mathcal{QA}(T)} = V$.

The automaton $\mathcal{A}^*$ is constructed so as to simulate accepting runs of $\mathcal{QA}$ which select V from some other tree "expanding" V.

Let $\mathcal{QA} = (Q, F, Q_s, \delta)$, then $\mathcal{A}^* = (Q_s, F_*, \delta_*)$ has set of states Q_s, coinciding with the selecting states of $\mathcal{QA}$, and set of final states $F_* = F \cap Q_s$.

We will now describe how the transition function δ_* can be constructed from $\mathcal{QA}$.

The transition function δ_* can be constructed in several steps described below.

- First compute the set of states $R \subseteq Q - Q_s$, which are reachable in runs of $\mathcal{QA}$ without going through a selecting state. More precisely, we will call a run ρ of a query automaton on a tree T *non-selecting* if $S_\rho(T) = \emptyset$.
  Then R is the set of all states $q \in Q - Q_s$ for which there exists a Σ-labeled tree T and a non-selecting run of $\mathcal{QA}$ on T, such that $\rho(\varepsilon) = q$.
  The set R can be computed in $O(\|\mathcal{QA}\|^2)$, via a standard reachability analysis algorithm. It is constructed incrementally. At the first step, R consists of all states $q \in Q - Q_s$ such that $\varepsilon \in \delta(q, a)$ for some $a \in \Sigma$. At step $i > 0$, we compute all states $q \in Q - Q_s$ such that $\delta(q, a) \cap R^* \neq \emptyset$, for some $a \in \Sigma$. These states are then added to R. The computation ends when no new states are produced.
  At the first step, computing R requires only a linear time check on the states of each NFA $\delta(q, a)$, for all $q \in Q$ and $a \in \Sigma$. Thus the cost of the first step is linear in $\|\mathcal{QA}\|$.
  At step i, the set R can be computed by checking non-emptiness of each NFA $\delta(q, a)$, for all $q \in Q$ and $a \in \Sigma$. In particular, first transitions over symbols outside R are removed from $\delta(q, a)$, and then non-emptiness is checked for the resulting NFA. Both the cost of removing transitions and the cost of checking non-emptiness are linear in the size $\|\delta(q, a)\|$.
  Thus the cost of each step is linear in $\|\mathcal{QA}\|$. Since there are at most $O(|Q|)$ steps, the overall cost of computing R is $O(\|\mathcal{QA}\|^2)$.
- Then, for each state $q \in Q_s$ and for each $a \in \Sigma$, the NFA $\delta_*(q, a)$ is constructed as follows. Observe that $\delta(q, a)$ is an automaton over alphabet Q; let $S_{q,a}$ its set of states and $\sigma_{q,a} \subseteq S \times Q \times S$ its transition function (we will denote them by S and σ when this does not arise confusion). We first construct an NFA $\mathcal{A}_\varepsilon(q, a)$ over alphabet Q_s with ε-transitions, having set of states S, initial and final states as in $\delta(q, a)$, and transition function $\sigma_\varepsilon \subseteq S \times (Q_s \cup \varepsilon) \times S$. Intuitively σ_ε is obtained from σ by discarding all transitions with symbols outside $Q_s \cup R$ and then replacing by ε all symbols in R. More formally σ_ε is defined as follows:
  – for all $q \in Q_s$, and for all states $s_1, s_2 \in S$, the transition (s_1, q, s_2) is in σ_ε if and only if $(s_1, q, s_2) \in \sigma$;
  – for all states $s_1, s_2 \in S$, the transition (s_1, ε, s_2) is in σ_ε if and only if $(s_1, q, s_2) \in \sigma$, for some $q \in R$.
  Finally $\delta_*(q, a)$ is obtained from $\mathcal{A}_\varepsilon(q, a)$ by removing ε-transitions.
  The cost of constructing $\delta_*(q, a)$ is dominated by the cost of removing ε-transitions, since the construction of $\mathcal{A}_\varepsilon(q, a)$ is linear in the size of σ. Removing ε-transitions can be done in time $O(|S|(|S| + |\sigma|))$ without increasing the number of states. For that, we first do a reachability analysis on the ε-transition portion alone, producing, for each state s, its ε-closure $L(s)$. The ε-closure of a state s is the set of all states of S reachable using only ε-transitions; it can be

computed in time $O(|\sigma|)$. Thus all ε-closures are computed in time $O(|S| \cdot |\sigma|)$. Now we replace each state $s \in S$ with $L(s)$ and add a new transition $(L(s), q, L(s'))$ iff there exists a state $s'' \in L(s)$ and a transition $(s'', q, s') \in \sigma_\varepsilon$, with $q \in Q_s$; this is done in time $O(|S|(|S| + |\sigma|))$.
Thus the overall cost of computing the transition function of $\mathcal{A}^*$ is $O(\sum_{q \in Q_s, a \in \Sigma} |S_{q,a}|(|S_{q,a}| + |\sigma_{q,a}|))$ which is $O(\|\mathcal{QA}\|^2)$.

Summing up, the overall cost of constructing $\mathcal{A}^*$ from $\mathcal{QA}$ is quadratic in the total size of $\mathcal{QA}$.
As a direct consequence of the construction of δ_* we have:

Claim 2. *For all $q \in Q_s$, all $q_1 q_2 \cdots q_k \in Q_s^*$, and all $a \in \Sigma$, the word $q_1 q_2 \cdots q_k$ belongs to $\delta_*(q, a)$ iff $R^* q_1 R^* q_2 \cdots q_k R^* \cap \delta(q, a) \neq \emptyset$.*

Proof. The word $q_1 q_2 \cdots q_k$ is accepted by $\delta_*(q, a)$ if and only if it is accepted by $\mathcal{A}_\varepsilon(q, a)$. This is the case if and only if in $\mathcal{A}_\varepsilon(q, a)$ there exist states $\overleftarrow{s_i}$ and $\overrightarrow{s_i}$, for $i = 1, \ldots, k$, such that: (1) the transition function of $\mathcal{A}_\varepsilon(q, a)$ contains transitions $(\overleftarrow{s_i}, q_i, \overrightarrow{s_i})$, for each $i = 1, \ldots, k$; (2) the state $\overleftarrow{s_{i+1}}$ is reachable in $\mathcal{A}_\varepsilon(q, a)$ from $\overrightarrow{s_i}$ via ε-transitions, for all $i = 1, \ldots, k-1$; (3) the state $\overleftarrow{s_1}$ is reachable from an initial state of $\mathcal{A}_\varepsilon(q, a)$ via ε-transitions; (4) some final state of $\mathcal{A}_\varepsilon(q, a)$ is reachable from $\overrightarrow{s_k}$ via ε-transitions.

Directly by definition of $\mathcal{A}_\varepsilon(q, a)$ this holds if and only if in $\delta(q, a)$ there exists a path from some initial state to some final state with labels in $R^* q_1 R^* q_2 \cdots q_k R^*$. $\square$

We are now ready to prove that A^* is the desired automaton:

Claim 3. *$\mathcal{A}^*$ accepts a tree V iff there exists a tree T such that $\mathcal{QA}(T) = V$.*

Proof. Assume $\mathcal{A}^*$ accepts V and let ρ be an accepting run. We next prove by induction the following statement:

(*) for each node s of V, if we let V_s the subtree of V rooted in s, then there exists a tree T_s and a run ρ_s of $\mathcal{QA}$ on T_s such that:
1. $\rho_s(\varepsilon) = \rho(s)$;
2. if we let D_s be $S_{\rho_s}(T_s)$, then D_s is prefix-closed and $T_s|_{D_s} = V_s$.

We prove (*) by induction on the depth of the node s. If s is a leaf with label a then, since ρ is an accepting run of $\mathcal{A}^*$, we have that $\varepsilon \in \delta_*(\rho(s), a)$. Therefore by Claim 2, there exists a word $r_1 \cdots r_n \in R^* \cap \delta(\rho(s), a)$. For each r_i we know, by definition of R, that there exists a tree T_i and a non-selecting run ρ_i of $\mathcal{QA}$ on T_i reaching state r_i in the root. We define T_s as the tree whose root, labeled a, has n children such that T_i is the subtree rooted at the i-th child. The run ρ_s of $\mathcal{QA}$ on T_s is defined as ρ_i on each T_i, for $i = 1, \ldots, n$, and as $\rho(s)$ on the root (ρ is a run of $\mathcal{QA}$ since $r_1 \cdots r_n$ are the states reached at the children of the root and $r_1 \cdots r_n \in \delta(\rho(s), a)$). Clearly the root of T_s is the only node selected by ρ_s, in fact each run ρ_i is non-selecting, while the state $\rho(s)$ in the root is in Q_s. As a consequence D_s is clearly prefix-closed. Moreover $T_s|_{D_s}$ is a tree of domain $\{\varepsilon\}$ with label a, hence it coincides with V_s, the subtree rooted in the leaf s of label a. This concludes the proof of the base case.

Now assume that s is a node with $m > 0$ children and label a. Since ρ is an accepting run of $\mathcal{A}^*$, we have $\rho(s \cdot 0) \cdots \rho(s \cdot (m-1)) \in \delta_*(\rho(s), a)$. Again by Claim 2, there exists a word $w_0 \rho(s \cdot 0) w_1 \rho(s \cdot 1) \cdots \rho(s \cdot (m-1)) w_m \in \delta(\rho(s), a)$, with $w_i \in R^*$.

By the induction hypothesis, there exist trees $T_{s \cdot i}$ and runs $\rho_{s \cdot i}$ satisfying conditions (1) and (2) above. Moreover, exactly as in the base case, for each word $w_i \in R^*$ we construct a forest of trees T_{w_i}, and non-selecting runs of $\mathcal{QA}$ on trees of T_{w_i} reaching the sequence of states w_i at the roots.

Then T_s is obtained by connecting the forest $T_{w_0} T_{s \cdot 0} T_{w_1} T_{s \cdot 2} \cdots T_{s \cdot (m-1)} T_{w_m}$ to a new common root with label a.

The run ρ_s of $\mathcal{QA}$ on T_s is defined as:

- $\rho_{s \cdot i}$ on the trees $T_{s \cdot i}$, for each $i = 0, \ldots, m-1$;
- the non-selecting runs defined on trees of T_{w_i}, for each $i = 0, \ldots, m$;
- $\rho(s)$ on the root.

This defines a run of $\mathcal{QA}$ since states $w_0 \rho(s \cdot 0) w_1 \rho(s \cdot 1) \cdots \rho(s \cdot (m-1)) w_m$ are reached at the children of the root and $w_0 \rho(s \cdot 0) w_1 \rho(s \cdot 1) \cdots \rho(s \cdot (m-1)) w_m \in \delta(\rho(s), a)$.

The set of nodes D_s selected by ρ_s on T_s is:

- empty, in trees T_{w_i} for each $i = 0, \ldots, m$ (since the run is non-selecting on these trees);
- $D_{s \cdot i}$ in each subtree $T_{s \cdot i}$, for $i = 0, \ldots, m-1$ (since this is the set of nodes selected by $\rho_{s \cdot i}$);
- the root ε (since $\rho(s) \in Q_s$).

By the induction hypothesis, the sets $D_{s\cdot i}$ are prefix closed, hence also D_s is prefix-closed. Finally, from the description of D_s it follows that $T_s|_{D_s}$ is a tree whose root has label a and m children: for each $i = 0, \dots, m-1$, the subtree rooted in child i is $T_{s\cdot i}|_{D_{s\cdot i}}$. By the induction hypothesis, $T_{s\cdot i}|_{D_{s\cdot i}} = V_{s\cdot i}$, for each $i = 0, \dots, m-1$; therefore $T_s|_{D_s}$ coincides with the subtree V_s rooted in s.

This proves (*). In particular (*) holds in the root of V, therefore there exists a tree T_ε and a run ρ_ε of $\mathcal{QA}$ on T_ε such that the set of selected nodes D_ε is prefix-closed, $T_\varepsilon|_{D_\varepsilon} = V$ and $\rho_\varepsilon(\varepsilon) = \rho(\varepsilon)$. This implies that ρ_ε is an accepting run of $\mathcal{QA}$ on T_ε. In fact, since ρ is an accepting run of $\mathcal{A}^*$ on V, the state $\rho(\varepsilon)$ is in F_*, but $F_* \subseteq F$, then $\rho(\varepsilon)$ is also final states of $\mathcal{QA}$. Moreover, since $\mathcal{QA}$ satisfies condition (1) of the definition of QAs, we have $D_\varepsilon = \mathcal{QA}(T_\varepsilon)$. Hence $T_\varepsilon|_{\mathcal{QA}(T_\varepsilon)} = V$. This shows $\mathcal{QA}(T_\varepsilon) = V$ and concludes the proof of one direction.

For the other direction, assume that there exists a tree T such that $\mathcal{QA}(T) = V$. Then, since V is non-empty, there exists an accepting run $\hat{\rho}$ of $\mathcal{QA}$ on T, and $\mathcal{QA}(T) = S_{\hat{\rho}}(T)$. Moreover, the equality $T|_{\mathcal{QA}(T)} = V$ implies that the restriction of T to nodes $\mathcal{QA}(T)$ is isomorphic to V (where T and V are viewed as structures). Let h be such isomorphism, viewed as a one-to-one mapping from nodes of V to nodes $\mathcal{QA}(T)$; then we can write $\mathcal{QA}(T) = \{h(s) \mid s \text{ is a node of } V\}$.

We define a run ρ of $\mathcal{A}^*$ on V such that $\rho(s) = \hat{\rho}(h(s))$, for each node s of V. We need to verify that ρ is indeed a run of $\mathcal{A}^*$, and that it is also accepting. In the following we prove that:

(a) ρ is a mapping from nodes of V to Q_s;
(b) for each node s in V with label a and $n \geqslant 0$ children,

$$\rho(s\cdot 0)\cdots\rho\big(s\cdot(n-1)\big) \in \delta_*\big(\rho(s), a\big);$$

(c) $\rho(\varepsilon) \in F_*$.

Item (a) holds because, for each node s in V, we know that $h(s)$ is in $\mathcal{QA}(T)$, that is, in $S_{\hat{\rho}}(T)$. Hence $\hat{\rho}(h(s)) \in Q_s$.

We now prove item (b). For each node $s \in V$, with label a and $n \geqslant 0$ children,

$$\rho(s\cdot 0)\cdots\rho\big(s\cdot(n-1)\big) = \hat{\rho}\big(h(s\cdot 0)\big)\cdots\hat{\rho}\big(h\big(s\cdot(n-1)\big)\big).$$

Since h is a isomorphism, $h(s)$ has label a and $h(s\cdot i) = h(s)\cdot j_i$, for some j_i and for each $i = 0, \dots, n-1$. Therefore

$$\rho(s\cdot 0)\cdots\rho\big(s\cdot(n-1)\big) = \hat{\rho}\big(h(s)\cdot j_0\big)\cdots\hat{\rho}\big(h(s)\cdot j_{n-1}\big).$$

Now let $m \geqslant j_{n-1}$ the number of children of $h(s)$ in T; since $\hat{\rho}$ is an accepting run of $\mathcal{QA}$ on T, we have that $\hat{\rho}(h(s)\cdot 0)\cdots\hat{\rho}(h(s)\cdot(m-1)) \in \delta(\hat{\rho}(h(s)), a)$.

We next prove that for each $j = 0, \dots, m-1$, with $j \notin \{j_i \mid i = 0, \dots, n-1\}$, the state $\hat{\rho}(h(s)\cdot j)$ is in R. Observe that since h is an isomorphism, a descendant $h(s)\cdot w$ of $h(s)$ (with $w \neq \varepsilon$) is in $\mathcal{QA}(T)$ iff $h(s)\cdot w$ is a descendant of $h(s\cdot i)$ for some i in $0, \dots, n-1$. Thus, for each $j = 0, \dots, m-1$, with $j \notin \{j_i \mid i = 0, \dots, n-1\}$, neither the node $h(s)\cdot j$ nor its descendants are in $\mathcal{QA}(T)$. Therefore, if T_j is the subtree of T rooted in $h(s)\cdot j$, the run $\hat{\rho}$ on T_j is non-selecting, hence, $\hat{\rho}(h(s)\cdot j) \in R$. This implies that the sequence of states $\hat{\rho}(h(s)\cdot 0)\cdots\hat{\rho}(h(s)\cdot(m-1))$ belongs also to $R^*\hat{\rho}(h(s)\cdot j_0)R^*\cdots\hat{\rho}(h(s)\cdot j_{n-1})R^*$, where $\hat{\rho}(h(s)\cdot j_i) \in Q_s$ for each $i = 0, \dots, n-1$. As a consequence, by Claim 2, the sequence of states $\hat{\rho}(h(s)\cdot j_0)\cdots\hat{\rho}(h(s)\cdot j_{n-1})$ belongs to $\delta_*(\hat{\rho}(h(s)), a)$; or in other words, $\rho(s\cdot 0)\cdots\rho(s\cdot(n-1)) \in \delta_*(\rho(s), a)$. This proves (b).

It remains to prove (c). Notice that $\rho(\varepsilon) = \hat{\rho}(h(\varepsilon))$ and $h(\varepsilon) = \varepsilon$ (since h is an isomorphism and $\varepsilon \in \mathcal{QA}(T)$). Therefore $\rho(\varepsilon)$ is equal to $\hat{\rho}(\varepsilon)$ which belongs to F, the final states of $\mathcal{QA}(T)$. Moreover $\rho(\varepsilon)$ is in Q_s, thus $\rho(\varepsilon) \in F_*$, the final states of $\mathcal{A}^*$.

This proves (c), shows that ρ is an accepting run of $\mathcal{A}^*$ on V and concludes the proof of the claim. □

We have already shown that $\mathcal{A}^*$ can be computed in time $O(\|\mathcal{QA}\|^2)$, moreover the number of states of $\mathcal{A}^*$ coincides with the selecting states of $\mathcal{QA}$, thus it is bounded by $|Q|$. This, together with Claim 3, concludes the proof of the lemma. □

To apply Lemma 4 to the problem of finding certain answers $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_\mathcal{V}}(\mathfrak{Q}; V)$, we now take the product of $\mathcal{QA}_\mathcal{V}$ with $\mathcal{A}$ and the automaton for $\neg\mathfrak{Q}$ (the selecting states in the product will be determined by $\mathcal{QA}_\mathcal{V}$), and obtain:

Theorem 4. *Let $\mathcal{QA}_\mathcal{V}$ be an upward-closed and single-run query automaton with states $Q_\mathcal{V}$, let $\mathcal{A}$ be an unranked tree automaton with states $Q_\mathcal{A}$ defining a schema, and let $\mathcal{A}_{\neg\mathfrak{Q}}$ be an automaton accepting trees for which $\mathfrak{Q}$ is false, and having states $Q_{\neg\mathfrak{Q}}$. Then one can construct, in polynomial time, an unranked tree automaton $\mathcal{A}^*$ such that*

1. *the number of states of $\mathcal{A}^*$ is at most $|Q_\mathcal{V}|\cdot|Q_\mathcal{A}|\cdot|Q_{\neg\mathfrak{Q}}|$;*
2. *$\mathcal{A}^*$ accepts $V \Leftrightarrow \underline{certain}^{\mathcal{A}}_{\mathcal{QA}_\mathcal{V}}(\mathfrak{Q}; V) = false$.*

Proof. Observe that $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_\mathcal{V}}(\mathfrak{Q}; V)$ is false if and only if there exists a tree T accepted by both $\mathcal{A}$ and $\mathcal{A}_{\neg\mathfrak{Q}}$, such that $\mathcal{QA}_\mathcal{V}(T) = V$. This can be restated by defining the product query automaton

$$\mathcal{QA} = \mathcal{QA}_\mathcal{V} \times \mathcal{A} \times \mathcal{A}_{\neg\mathfrak{Q}}$$

with selecting states consisting of all the triples whose first component is a selecting state of $\mathcal{QA}_{\mathcal{V}}$. Clearly for each tree T, the query answer $\mathcal{QA}(T)$ coincides with $\mathcal{QA}_{\mathcal{V}}(T)$ if T is accepted by $\mathcal{A}$ and $\mathcal{A}_{\neg\mathfrak{Q}}$, and is empty otherwise. This implies that also $\mathcal{QA}$ is upward-closed and:

$$\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_{\mathcal{V}}}(\mathfrak{Q}; V) \text{ is false} \quad \text{iff} \quad \text{there exists a tree } T \text{ such that } \mathcal{QA}(T) = V.$$

The query automaton $\mathcal{QA}$ is no longer single-run, because it does not accept all trees. But for each accepting run ρ of $\mathcal{QA}$ on a tree T, we have $S_\rho(T) = \mathcal{QA}_{\mathcal{V}}(T)$. Therefore $\mathcal{QA}$ satisfies condition 1 of the definition of QA.

Then, by Lemma 4, we can construct an automaton $\mathcal{A}^*$ that accepts a tree V iff $\underline{certain}^{\mathcal{A}}_{\mathcal{QA}_{\mathcal{V}}}(\mathfrak{Q}; V)$ is false. The construction of $\mathcal{A}^*$ requires first the computation of $\mathcal{QA}$ and then the construction of $\mathcal{A}^*$ form $\mathcal{QA}$.

Hence, the algorithm consists of two steps:

1. constructing the product QA $\mathcal{QA}_{\mathcal{V}} \times \mathcal{A} \times \mathcal{A}_{\neg\mathfrak{Q}}$, and
2. applying the algorithm of Lemma 4 to this QA to obtain an automaton $\mathcal{A}^*$.

The product query automaton $\mathcal{QA}$ can be computed in time polynomial in $\|\mathcal{QA}_{\mathcal{V}}\| \cdot \|\mathcal{A}\| \cdot \|\mathcal{A}_{\neg\mathfrak{Q}}\|$. Therefore, by Lemma 4, $\mathcal{A}^*$ can be computed in time polynomial in $\|\mathcal{QA}_{\mathcal{V}}\| \cdot \|\mathcal{A}\| \cdot \|\mathcal{A}_{\neg\mathfrak{Q}}\|$, and has number of states at most $|Q_{\mathcal{V}}| \cdot |Q_{\mathcal{A}}| \cdot |Q_{\neg\mathfrak{Q}}|$.

This concludes the proof of the theorem. □

Combining Theorem 4 with previous translations into single-run QAs and properties of the latter, we obtain algorithms for verifying properties of views given by XPath expressions. Revisiting our motivating example from Section 2, we make the following assumptions:

- the view definition is given by an XPath (conditional or core) expression e_V; the view V of a source tree T has all the nodes selected by e_V and their ancestors;
- the schema definition is given by a DTD d;
- the query $\mathfrak{Q}$ is an arbitrary Boolean combination of containment statements $e \subseteq e'$, where e, e' come from a set E of XPath expressions.

Then, for a given V, we want to check if $\underline{certain}^{d}_{e_V}(\mathfrak{Q}; V)$ is false: that is, the secret encoded by $\mathfrak{Q}$ cannot be revealed by V, since not all source trees T that conform to d and generate V satisfy $\mathfrak{Q}$. Then, using the algorithm from the proof of Theorem 4, we have the following:

Corollary 3. *In the above setting, one can construct in time polynomial in $\|d\|$ and exponential in $\|E\| + \|e_V\|$ an unranked tree automaton $\mathcal{A}^*$ with $\|d\| \cdot 2^{O(\|e_V\|+\|E\|)}$ states that accepts a view V iff $\underline{certain}^{d}_{e_V}(\mathfrak{Q}; V)$ is false.*

Note that again the exponent contains the size of typically small XPath expressions, and not the potentially large schema definition d.

8. Conclusion

There are several extensions we would like to consider. One concerns relative specifications often used in the XML context – these apply to subtrees. Results of [22,3] on model-checking of *now* and *within* operators on words and nested words indicate that an exponential blowup is unavoidable, but there could well be relevant practical cases that do not exhibit it. We would like to see how LTL-to-Büchi optimization techniques (e.g., in [13,19]) could be adapted in our setting, to produce automata of smaller size. We also would like to see if automata can be used for reasoning about views without imposing upward-closeness of [8], which does not account for some of the cases of secure XML views [15]. One could look beyond first-order at logics having the power of MSO or ambient logics with known translations into automata, and investigate their translations into QAs [10,20,16].

Acknowledgements

We thank Pablo Barceló and Floris Geerts for their comments. Part of this work was done while the second author was at the University of Edinburgh. The authors were supported by EPSRC grants E005039 and F028288, and the FET-Open Project FoX (grant agreement 233599); the first author was also supported by the EC grant MEXC-CT-2005-024502.

References

[1] S. Abiteboul, B. Cautis, T. Milo, Reasoning about XML update constraints, in: PODS'07, pp. 195–204.
[2] S. Abiteboul, L. Segoufin, V. Vianu, Representing and querying XML with incomplete information, ACM TODS 31 (2006) 208–254.
[3] R. Alur, M. Arenas, P. Barceló, K. Etessami, N. Immerman, L. Libkin, First-order and temporal logics for nested words, in: LICS'07, pp. 151–160.
[4] R. Alur, K. Etessami, P. Madhusudan, A temporal logic of nested calls and returns, in: TACAS'04, pp. 467–481.

[5] M. Arenas, W. Fan, L. Libkin, Consistency of XML specifications, in: Inconsistency Tolerance, Springer, 2005, pp. 15–41.
[6] P. Barceló, L. Libkin, Temporal logics over unranked trees, in: LICS'05, pp. 31–40.
[7] M. Benedikt, W. Fan, F. Geerts, XPath satisfiability in the presence of DTDs, in: PODS'05, pp. 25–36.
[8] M. Benedikt, I. Fundulaki, XML subtree queries: Specification and composition, in: DBPL'05, pp. 138–153.
[9] M. Bojanczyk, C. David, A. Muscholl, Th. Schwentick, L. Segoufin, Two-variable logic on data trees and XML reasoning, in: PODS'06, pp. 10–19.
[10] I. Boneva, J.-M. Talbot, S. Tison, Expressiveness of a spatial logic for trees, in: LICS 2005, pp. 280–289.
[11] D. Calvanese, G. De Giacomo, M. Lenzerini, M.Y. Vardi, Regular XPath: Constraints, query containment and view-based answering for XML documents, in: Logic in Databases, 2008.
[12] E. Clarke, O. Grumberg, D. Peled, Model Checking, MIT Press, 1999.
[13] M. Daniele, F. Giunchiglia, M.Y. Vardi, Improved automata generation for linear temporal logic, in: CAV'99, pp. 249–260.
[14] W. Fan, C.Y. Chan, M. Garofalakis, Secure XML querying with security views, in: SIGMOD'04, pp. 587–598.
[15] W. Fan, F. Geerts, X. Jia, A. Kementsietsidis, Rewriting regular XPath queries on XML views, in: ICDE'07, pp. 666–675.
[16] M. Frick, M. Grohe, C. Koch, Query evaluation on compressed trees, in: LICS'03, pp. 188–197.
[17] P. Genevés, N. Layaida, A system for the static analysis of XPath, ACM TOIS 24 (2006) 475–502.
[18] P. Genevés, N. Layaida, A. Schmitt, Efficient static analysis of XML paths and types, in: Proc. of ACM SIGPLAN Conf. on Programming Language Design and Implementation, 2007, pp. 342–351.
[19] R. Gerth, D. Peled, M. Vardi, P. Wolper, Simple on-the-fly automatic verification of linear temporal logic, in: PSTV 1995, pp. 3–18.
[20] G. Gottlob, C. Koch, Monadic datalog and the expressive power of languages for web information extraction, J. ACM 51 (2004) 74–113.
[21] O. Kupferman, A. Pnueli, Once and for all, in: LICS'95, pp. 25–35.
[22] F. Laroussinie, N. Markey, Ph. Schnoebelen, Temporal logic with forgettable past, in: LICS'02, pp. 383–392.
[23] L. Libkin, Logics for unranked trees: An overview, in: ICALP'05, pp. 35–50.
[24] L. Libkin, C. Sirangelo, Reasoning about XML with temporal logics and automata, in: LPAR'08, pp. 97–112.
[25] S. Maneth, T. Perst, H. Seidl, Exact XML type checking in polynomial time, in: ICDT 2007, pp. 254–268.
[26] W. Martens, F. Neven, Th. Schwentick, Simple off the shelf abstractions for XML schema, SIGMOD Record 36 (3) (2007) 15–22.
[27] M. Marx, XPath with conditional axis relations, in: EDBT 2004, pp. 477–494.
[28] M. Marx, Conditional XPath, ACM TODS 30 (2005) 929–959.
[29] M. Marx, M. de Rijke, Semantic characterizations of navigational XPath, SIGMOD Record 34 (2005) 41–46.
[30] F. Neven, Design and analysis of query languages for structured documents, PhD Thesis, U. Limburg, 1999.
[31] F. Neven, Automata, logic, and XML, in: CSL 2002, pp. 2–26.
[32] F. Neven, Th. Schwentick, Query automata over finite trees, TCS 275 (2002) 633–674.
[33] F. Neven, Th. Schwentick, On the complexity of XPath containment in the presence of disjunction, DTDs, and variables, LMCS 2 (3) (2006).
[34] F. Neven, J. Van den Bussche, Expressiveness of structured document query languages based on attribute grammars, J. ACM 49 (1) (2002) 56–100.
[35] J. Niehren, L. Planque, J.-M. Talbot, S. Tison, N-ary queries by tree automata, in: DBPL 2005, pp. 217–231.
[36] B.-H. Schlingloff, Expressive completeness of temporal logic of trees, Journal of Applied Non-Classical Logics 2 (1992) 157–180.
[37] Th. Schwentick, XPath query containment, SIGMOD Record 33 (2004) 101–109.
[38] Th. Schwentick, Automata for XML – a survey, JCSS 73 (2007) 289–315.
[39] W. Thomas, Languages, automata, and logic, in: Handbook of Formal Languages, vol. III, pp. 389–455.
[40] M.Y. Vardi, An automata-theoretic approach to linear temporal logic, in: Banff Higher Order Workshop, 1996.
[41] M.Y. Vardi, Reasoning about the past with two-way automata, in: ICALP'98, pp. 628–641.
[42] M.Y. Vardi, P. Wolper, Reasoning about infinite computations, Inf. & Comput. 115 (1994) 1–37.
[43] M.Y. Vardi, P. Wolper, Automata-theoretic techniques for modal logics of programs, JCSS 33 (1986) 183–221.