Node Selection Query Languages for Trees

Diego Calvanese
Research Centre for Knowledge and Data
Free University of Bozen-Bolzano, Italy

Giuseppe De Giacomo, Maurizio Lenzerini
Dip. di Informatica, Automatica e Gestionale
Sapienza Università di Roma, Italy

Moshe Y. Vardi
Dept. of Computer Science
Rice University, Houston, U.S.A.

Abstract

The study of node selection query languages for (finite) trees has been a major topic in the recent research on query languages for Web documents. On one hand, there has been an extensive study of XPath and its various extensions. On the other hand, query languages based on classical logics, such as first-order logic (FO) or Monadic Second-Order Logic (MSO), have been considered. Results in this area typically relate an XPath-based language to a classical logic. What has yet to emerge is an XPath-related language that is as expressive as MSO, and at the same time enjoys the computational properties of XPath, which are linear time query evaluation and exponential time query-containment test. In this paper we propose $\mu XPath$, which is the alternation-free fragment of XPath extended with fixpoint operators. Using two-way alternating automata, we show that this language does combine desired expressiveness and computational properties, placing it as an attractive candidate for the definite node-selection query language for trees.

Keywords: tree-structured data, XML databases, fixpoint logics, query evaluation, query containment, weak alternating tree automata

1 Introduction

XML[1] is the standard language for Web documents supporting semistructured data. From the conceptual point of view, an XML document can be seen as a finite node-labeled tree, and several formalisms have been proposed as query languages over XML documents considered as finite trees.

Broadly speaking, there are two main classes of such languages, those focusing on selecting a set of nodes based on structural properties of the tree [55, 72], and those where the mechanisms for the selection of the result also take into account node attributes and their associated values taken from a specified domain [8, 7, 11, 25, 57, 41]. We focus here on the former class of queries, which we call *node selection queries*. Many of such formalisms come from the tradition of modal logics, similarly to the most expressive languages of the Description Logics family [4], based on the correspondence between the tree edges and the accessibility relation used in the interpretation strcutures of modal logics. *XPath* [20] is a notable example of these formalisms, and, in this sense, it can also be seen as an expressive Description Logic over finite trees. Relevant extensions of *XPath* are inspired by the family of Propositional Dynamic Logic (PDL) [59]. For example, *RXPath* is the extension of *XPath* with binary relations specified through regular expression, used to formulate expressive navigational patterns over XML documents [17]. Here, the correspondence is between programs of PDL and paths in the tree.

A main line of research on node selection queries has been on identifying nice computational properties of *XPath*, and studying extensions of such language that still enjoy these properties. An important feature of *XPath* is the tractability of query evaluation in data complexity, i.e., with respect to the size of the input tree. In fact, queries in the navigational core *CoreXPath* can be evaluated in time that is linear in

[1] `http://www.w3.org/TR/REC-xml/`

both the size of the query and the size of the input tree [35, 9]. This property is enjoyed also by various extensions of *XPath*, including *RXPath* [50]. Another nice computational property of *XPath* is that checking query containment, which is the basic task for static analysis of queries, is in ExpTime [56, 63]. This property holds also for *RXPath* [69, 17], and other extensions of *XPath* [68].

Another line of research focused on expressive power. Marx has shown that *XPath* is expressively equivalent to FO^2, the 2-variable fragment of first-order logic, while *CXPath*, which is the extension of *XPath* with conditional axis relations, is expressively equivalent to full FO [50, 51]. Regular extensions of *XPath* are expressively equivalent to extensions of FO with transitive closure [67, 69]. Another classical logic is Monadic Second-Order Logic (MSO). This logic is more expressive than FO and its extensions by transitive closure [48, 67, 69]. In fact, it has been argued that MSO has the right expressiveness required for Web information extraction and hence can serve as a yardstick for evaluating and comparing wrappers [34]. Various logics are known to have the same expressive power as MSO, cf. [48], but so far no natural extension of *XPath* that is expressively equivalent to MSO and enjoys the nice computational properties of *XPath* has been identified.

A further line of research focuses on the relationship between query languages for finite trees and tree automata [49, 54, 64]. Various automata models have been proposed. Among the cleanest models is that of node-selecting tree automata, which are automata on finite trees, augmented with node selecting states [55, 30]. What has been missing in this line of inquiry is an automaton model that can be used *both* for testing query containment and for query evaluation [64].

Some progress on the automata-theoretic front was recently reported in [17], where a comprehensive automata-theoretic framework for both evaluating and reasoning about *RXPath* was developed. The framework is based on *two-way weak alternating tree automata*, denoted 2WATAs [44], but specialized for finite trees, and enables one to derive both a linear-time algorithm for query evaluation and an exponential-time algorithm for testing query containment.

The goal of this paper is to introduce a declarative query language, namely μ*XPath*[2], based on *XPath* enriched with *alternation-free* fixpoint operators, which preserves these nice computational properties. The significance of this extension is due to a further key result of this paper, which shows that on *finite* trees alternation-free fixpoint operators are sufficient to capture all of MSO, which is considered to be the benchmark query language on tree-structured data. Alternation freedom implies that the least and greatest fixpoint operators do not interact, and is known to yield computationally amenable logics [14, 44]. It is also known that unfettered interaction between least and greatest fixpoint operators results in formulas that are very difficult for people to comprehend, cf. [42].

Fixpoint operators have been studied in the μ-calculus, interpreted over arbitrary structures [42], which by the tree-model property of this logic, can be restricted to be interpreted over infinite trees. It is known that, to obtain the full expressive power of MSO on infinite trees, arbitrary alternations of fixpoints are required in the μ-calculus (see, e.g., [36]). Forms of μ-calculus have also been considered in Description Logics [24, 61, 43, 10], again interpreted over infinite trees. In this context, the present work can provide the foundations for a description logic tailored towards acyclic finite (a.k.a. well-founded) frame structures. In this sense, the present work overcomes [15], where an explicit well-foundedness construct was used to capture XML in description logics.

In a finite-tree setting, extending *XPath* with forms of fixpoint operators, has been studied earlier [2, 67, 48, 32, 31]. While for arbitrary fixpoints the resulting query language is equivalent to MSO and has an exponential-time containment test, it is not known to have a linear-time evaluation algorithm. In contrast, as μ*XPath* is alternation free it is closely related to a stratified version of Monadic Datalog proposed as a query language for finite trees in [34, 30], which enjoys linear-time evaluation. Note, however, that the complexity of containment of stratified Monadic Datalog is unknown.

We prove here that there is a very direct correspondence between μ*XPath* and 2WATAs. Specifically, there are effective translations from μ*XPath* queries to 2WATAs and from 2WATAs to μ*XPath*. We show that this yields the nice computational properties for μ*XPath*. We then prove the equivalence of 2WATAs to node-selecting tree automata (NSTA), shown to be expressively equivalent to MSO [30]. On the one hand, we have an exponential translation from 2WATAs to NSTAs. On the other hand, we have a linear translation from NSTAs to 2WATAs. This yields the expressive equivalence of μ*XPath* to MSO.

It is worth noting that the automata-theoretic approach of 2WATAs is based on techniques developed in the context of program logics [44, 71]. Here, however, we leverage the fact that we are dealing with *finite* trees, rather than *infinite* trees that are usually used in the program-logics context. Indeed, the automata-theoretic techniques used in reasoning about infinite trees are notoriously difficult [62, 66] and

[2]An earlier version of this paper has been published in the Proceedings of the 24th AAAI Conference on Artificial Intelligence (AAAI 2010) [18].

have resisted efficient implementation. The restriction to finite trees here enables one to obtain a much more feasible algorithmic approach. In particular, one can make use of symbolic techniques, at the base of modern model checking tools [14], for effectively querying and verifying XML documents. It is worth noting that while 2WATAs run over finite trees they are allowed to have infinite runs. This separates 2WATAs from the alternating finite-tree automata used elsewhere [23, 65].

The paper is organized as follows. In Section 2 we present syntax and semantics of $\mu XPath$, as well as examples of queries expressed in this language. In Sections 3 and 4 we show how to make use of two-way automata over finite trees as a formal tool for addressing query evalation and query containemnt in the context of $\mu XPath$. More specifically, in Section 3 we introduce the class of two-way weak alternating tree automata, and devise mutual translation between them and $\mu XPath$ queries, and in Section 4 we provide algorithms for deciding the acceptance and non-emptiness problems for 2WATAs. In Section 5 we exploit the correspondence between two-way weak alternating tree automata and $\mu XPath$ to illustrate the main characteristics of $\mu XPath$ as a query language over finite trees. Section 6 deals with the expressive power of $\mu XPath$, by establishing the relationship between two-way weak alternating tree automata and MSO. Finally, Section 7 concludes the paper.

2 The Query Language $\mu XPath$

In this paper we are concerned with query languages over tree-structured data, which is customary in the XML setting [50, 51]. More precisely, we consider databases as finite sibling-trees, which are tree structures whose nodes are linked to each other by two relations: the child relation, connecting each node with its children in the tree; and the immediate-right-sibling relation, connecting each node with its sibling immediately to the right in the tree. Such a relation models the order between the children of the node in an XML documents. Each node of the sibling tree is labeled by (possibly many) elements of a set Σ of atomic propositions that represent either XML tags or XML attribute-value pairs. Observe that in general sibling trees are more general than XML documents since they would allow the same node to be labeled by several tags.

Formally, a (finite) tree is a complete prefix-closed non-empty (finite) set of words over $\mathbb{N}$, i.e., the set of positive natural numbers. In other words, a *(finite) tree* is a (finite) set of words $\Delta \subseteq \mathbb{N}^*$, such that if $x{\cdot}i \in \Delta$, where $x \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then also $x \in \Delta$, and if $i > 1$, then also $x{\cdot}(i{-}1) \in \Delta$. The elements of Δ are called *nodes*, the empty word ε is the *root* of Δ, and for every $x \in \Delta$, the nodes $x{\cdot}i$, with $i \in \mathbb{N}$, are the *successors* of x. By convention we take $x{\cdot}0 = x$, and $x{\cdot}i{\cdot}{-1} = x$. The *branching degree* $d(x)$ of a node x denotes the number of successors of x. If the branching degree of all nodes of a tree is bounded by k, we say that the tree is *ranked* and has branching degree k. In particular, if the branching degree is 2, we say that the tree is *binary*. Instead, if the number of successors of the nodes is a priori unbounded, we say that the tree is *unranked*. In contrast, *ranked* trees have a bound on the number of successors of nodes; in particular, for *binary trees* the bound is 2. A *(finite) labeled tree* over an alphabet $\mathcal{L}$ of labels is a pair $T = (\Delta^T, \ell^T)$, where Δ^T is a (finite) tree and the labeling $\ell^T : \Delta^T \to \mathcal{L}$ is a mapping assigning to each node $x \in \Delta^T$ a label $\ell^T(x)$ in $\mathcal{L}$.

A *sibling tree* T is a finite labeled unranked tree each of whose nodes is labeled with a set of atomic propositions in an alphabet Σ, i.e., $\mathcal{L} = 2^\Sigma$. Given $A \in \Sigma$, we denote by A^T the set of nodes x of Δ^T such that $A \in \ell^T(x)$. It is customary to denote a sibling tree T by $(\Delta^T, \cdot^T)$. On sibling trees, two auxiliary binary relations between nodes, and their inverses are defined:

$$\begin{array}{lcl}
\mathsf{child}^T & = & \{(z, z{\cdot}i) \mid z, z{\cdot}i \in \Delta^T\} \\
(\mathsf{child}^-)^T & = & \{(z{\cdot}i, z) \mid z, z{\cdot}i \in \Delta^T\} \\
\mathsf{right}^T & = & \{(z{\cdot}i, z{\cdot}(i{+}1)) \mid z{\cdot}i, z{\cdot}(i{+}1) \in \Delta^T\} \\
(\mathsf{right}^-)^T & = & \{(z{\cdot}(i{+}1), z{\cdot}i) \mid z{\cdot}i, z{\cdot}(i{+}1) \in \Delta^T\}
\end{array}$$

The relations child and right are called *axes*.

One of the core languages used to query tree-structured data is *XPath*, whose definition we briefly recall here. An *XPath node expression* φ is defined by the following syntax, which is inspired by Propositional Dynamic Logic (PDL) [29, 3, 17]:

$$\begin{array}{lcl}
\varphi & \longrightarrow & A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2 \mid \langle P \rangle\varphi \mid [P]\varphi \\
P & \longrightarrow & \mathsf{child} \mid \mathsf{right} \mid \mathsf{child}^- \mid \mathsf{right}^-
\end{array}$$

where A denotes an atomic proposition belonging to an alphabet Σ, child and right denote the main atomic relations between nodes in a tree, usually called *axis* relations. The expressions child^- and right^-

denote their inverses, which in fact correspond to the other two standard *XPath* axes parent and left, respectively. Intuitively, a node expression is a formula specifying a property of nodes, where an atomic proposition A asserts that the node is labeled with A, negation, conjunction, and disjunction have the usual meaning, $\langle P\rangle\varphi$, where P is one of the axes, denotes that the node is connected via P with a node satisfying φ, and $[P]\varphi$ asserts that all nodes connected via P satisfy φ. We also adopt the usual abbreviations for booleans, i.e., true, false, and $\varphi_1 \to \varphi_2$.

The query language studied in this paper, called *μXPath* is an extension of *XPath* with a mechanism for defining sets of nodes by means of explicit fixpoint operators over systems of equations. *μXPath* is essentially the Alternation-Free μ-Calculus, where the syntax allows for the fixpoints to be defined over vectors of variables [28].

To define *μXPath* queries, we consider a set $\mathcal{X}$ of variables, disjoint from the alphabet Σ. An *equation* has the form

$$X \doteq \varphi$$

where $X \in \mathcal{X}$, and φ is an *XPath* node expression having as atomic propositions symbols from $\Sigma \cup \mathcal{X}$. We call the left-hand side of the equation its *head*, and the right-hand side its *body*. A set of equations can be considered as mutual fixpoint equations, which can have multiple solutions in general. We are actually interested in two particular solutions: the smallest one, i.e., the least fixpoint (lfp), and the greatest one, i.e., the greatest fixpoint (gfp), both of which are guaranteed to exist under a suitable syntactic monotonicity condition to be defined below. Given a set of equations

$$\{X_1 \doteq \varphi_1, \ldots, X_n \doteq \varphi_n\},$$

where we have one equation with X_i in the head, for $1 \leq i \leq n$, a *fixpoint block* has the form $\mathit{fp}\{X_1 \doteq \varphi_1, \ldots, X_n \doteq \varphi_n\}$, where

- *fp* is either lfp or gfp, denoting respectively the least fixpoint and the greatest fixpoint of the set of equations, and
- each variable X_i, for $1 \leq i \leq n$, appears positively in φ_i, for $1 \leq i \leq n$ (see [42]).

We say that the variables $X_1, \ldots, X_n$ are *defined* in the fixpoint block $\mathit{fp}\{X_1 \doteq \varphi_1, \ldots, X_n \doteq \varphi_n\}$.

A *μXPath query* has the form $X : \mathcal{F}$, where $X \in \mathcal{X}$ and $\mathcal{F}$ is a set of fixpoint blocks such that:

- X is a variable defined in $\mathcal{F}$;
- the sets of variables defined in different fixpoint blocks in $\mathcal{F}$ are mutually disjoint;
- for each fixpoint block $F \in \mathcal{F}$, each variable X defined in F appears only positively in the bodies of equations in F (*syntactic monotonicity*);
- there exists a partial order $\preceq$ on the fixpoint blocks in $\mathcal{F}$ such that, for each $F_i \in \mathcal{F}$, the bodies of equations in F_i contain only variables defined in fixpoint blocks $F_j \in \mathcal{F}$ with $F_j \preceq F_i$.

The meaning of a query q of the form $X : \mathcal{F}$ is based on the fact that, when evaluated over a tree T, $\mathcal{F}$ assigns to each variable defined in it a set of nodes of T, and that q returns as result the set assigned to X. We intuitively explain the mechanism behind the assignment of $\mathcal{F}$ to its variables. We choose partial order $\preceq$ on the fixpoint blocks in $\mathcal{F}$ respecting the conditions above, and we operate one block of equations at a time according to $\preceq$. For each fixpoint block, we compute the solution of the corresponding equations, obviously taking into account the type of fixpoint, and using the assignments for the variables already computed for previous blocks. We come back to the formal semantics below, and first give some examples of *μXPath* queries.

The following query computes the nodes reaching a `red` node on all child-paths (possibly of length 0), exploiting the encoding of transitive closure by means of a least fixpoint:

$$X : \{\mathsf{lfp}\{X \doteq \mathtt{red} \lor [\mathsf{child}]X\}\}.$$

As another example, to obtain the nodes all of whose descendants (including the node itself) are not simultaneously `red` and `blue`, we can write the query:

$$X : \{\mathsf{gfp}\{X \doteq (\mathtt{red} \to \neg\mathtt{blue}) \land [\mathsf{child}]X\}\}.$$

Notice that such nodes are those that do not have descendant that are simultaneously `red` and `blue`. The latter set of nodes is characterized by a least fixpoint, and therefore query q can also be considered as the negation of such least fixpoint.

$$
\begin{aligned}
&A_\rho^T = A^T,\\
&X_\rho^T = \begin{cases} \rho(X), & \text{if } X \text{ is defined in } F_i\\ \mathcal{E}, & \text{if } X \text{ is defined in some } F_j \preceq F_i \text{ and } X/\mathcal{E} \in (F_j)_\rho^T \end{cases}\\
&(\neg\varphi)_\rho^T = \Delta^T \setminus \varphi_\rho^T,\\
&(\varphi_1 \wedge \varphi_2)_\rho^T = (\varphi_1)_\rho^T \cap (\varphi_2)_\rho^T,\\
&(\varphi_1 \vee \varphi_2)_\rho^T = (\varphi_1)_\rho^T \cup (\varphi_2)_\rho^T,\\
&(\langle P\rangle\varphi)_\rho^T = \{z \mid \exists z'.(z,z') \in P^T \wedge z' \in \varphi_\rho^T\},\\
&([P]\varphi)_\rho^T = \{z \mid \forall z'.(z,z') \in P^T \rightarrow z' \in \varphi_\rho^T\},\\
&(\mathsf{lfp}\{X_1 \doteq \varphi_1, \ldots, X_n \doteq \varphi_n\})_\rho^T = \{X_1/\mathcal{E}_1^\mu, \ldots, X_n/\mathcal{E}_n^\mu\},\\
&(\mathsf{gfp}\{X_1 \doteq \varphi_1, \ldots, X_n \doteq \varphi_n\})_\rho^T = \{X_1/\mathcal{E}_1^\nu, \ldots, X_n/\mathcal{E}_n^\nu\},
\end{aligned}
$$

Figure 1: Semantics of the *μXPath* formulas in fixpoint block F_i

We now illustrate an example where both a least and a greatest fixpoint block are used in the same query. Indeed, to compute `red` nodes all of whose `red` descendants have only `blue` children and all of whose `blue` descendants have at least a `red` child, we can use the following query:

$$
\begin{aligned}
X_1 : \{\mathsf{gfp}\{X_0 \doteq\ & (\mathtt{red} \rightarrow [\mathsf{child}]\mathtt{blue}) \wedge\\
& (\mathtt{blue} \rightarrow \langle\mathsf{child}\rangle\mathtt{red}) \wedge [\mathsf{child}]X_0\}\},\\
\mathsf{lfp}\{X_1 \doteq\ & \mathtt{red} \wedge X_0\}
\end{aligned}
$$

Notice that in the above query, the only partial order coherent with the conditions of *μXPath* given above is the one where the greatest fixpoint block precedes the least fixpoint block.

Notice also that in the above query we could have used the greatest fixpoint in the second block instead of the least fixpoint. Indeed, it is easy to see that, whenever a set of equations in non-recursive, least and greatest fixpoints have the same meaning, since they both characterize the obvious single solution of the systems of equations.

Now, suppose that we want to denote the `red` nodes all of whose `red` descendants reach `blue` nodes on all child-paths, and all of whose `blue` descendants reach `red` nodes on at least one child-path. The resulting query is the following, where we have written the fixpoint blocks according to a partial order coherent with the conditions of *μXPath*:

$$
\begin{aligned}
X_3 : \{&\mathsf{lfp}\{X_0 \doteq \mathtt{blue} \vee [\mathsf{child}]X_0\},\\
&\mathsf{lfp}\{X_1 \doteq \mathtt{red} \vee \langle\mathsf{child}\rangle X_1\},\\
&\mathsf{gfp}\{X_2 \doteq (\mathtt{red} \rightarrow X_0) \wedge (\mathtt{blue} \rightarrow X_1) \wedge [\mathsf{child}]X_2\},\\
&\mathsf{lfp}\{X_3 \doteq \mathtt{red} \wedge X_2\}\}
\end{aligned}
$$

Finally, to denote the nodes having a `red` sibling that follows it in the sequence of right siblings, and such that all siblings along such sequence have a `blue` descendant, we can use the following query:

$$
\begin{aligned}
X_0 : \{\mathsf{lfp}\{&X_0 \doteq X_1 \wedge (\mathtt{red} \vee \langle\mathsf{right}\rangle X_0),\\
&X_1 \doteq \mathtt{blue} \vee \langle\mathsf{child}\rangle X_1\}\}
\end{aligned}
$$

The formal semantics of *μXPath* is defined by considering sibling trees as interpretation structures. To specify the semantics of equations, we introduce second order variable assignments. A *(second order) variable assignment* ρ on a tree $T = (\Delta^T, \cdot^T)$ is a mapping that assigns to variables of $\mathcal{X}$ sets of nodes in Δ^T. To specify the semantics of a *μXPath* query $X : \mathcal{F}$ relative to a sibling tree T and a variable assignment ρ, we consider a partial order $\preceq$ of the fixpoint blocks in $\mathcal{F}$, and proceed by induction on $\preceq$. Consider now the induction step dealing with the fixpoint block $F_i \in \mathcal{F}$. The role of this step is to provide the semantics of F_i in terms of a variable assignment $\{X_1/\mathcal{E}_1, \ldots, X_n/\mathcal{E}_n\}$, where $X_1, \ldots, X_n$ are all the variables defined in F_i and $\mathcal{E}_1, \ldots, \mathcal{E}_n$ are the sets of nodes of T associated to such variables by the assignment. The semantics of F_i, denoted ${F_i}_\rho^T$, is specified as shown in Figure 1, where:

- The semantics of A, $\neg\varphi$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\langle P\rangle\varphi$, and $[P]\varphi$ is the usual one.
- The semantics of a variable X depends on whether X is defined in F_i or not. In the former case, it is simply given by the variable assignment ρ; otherwise, it is determined by the variable assignment of block F_j in which X is defined. Observe that F_j precedes F_i in the partial order $\preceq$.

- The semantics of a least fixpoint block $\mathsf{lfp}\{X_1 \doteq \varphi_1, \ldots, X_n \doteq \varphi_n\}$ is the variable assignment $\{X_1/\mathcal{E}_1^\mu, \ldots, X_n/\mathcal{E}_n^\mu\}$, where $(\mathcal{E}_1^\mu, \ldots, \mathcal{E}_n^\mu)$ is the intersection of all solutions of the fixpoint block, where each solution is an n-tuple of sets of nodes of T, and the intersection is done component-wise. Formally:

$$(\mathcal{E}_1^\mu, \ldots, \mathcal{E}_n^\mu) \;=\; \bigcap\{(\mathcal{E}_1, \ldots, \mathcal{E}_n) \mid \mathcal{E}_1 = (\varphi_1)^T_{\rho[X_1/\mathcal{E}_1,\ldots,X_n/\mathcal{E}_n]}, \ldots, \mathcal{E}_n = (\varphi_n)^T_{\rho[X_1/\mathcal{E}_1,\ldots,X_n/\mathcal{E}_n]}\},$$

  where $\rho[X_1/\mathcal{E}_1, \ldots, X_n/\mathcal{E}_n]$ denotes the variable assignment identical to ρ, except that it assigns to X_i the value $\mathcal{E}_i$, for $1 \leq i \leq n$. Note that, due to syntactic monotonicity, $(\mathcal{E}_1^\mu, \ldots, \mathcal{E}_n^\mu)$ is itself a solution of the fixpoint block, and indeed the smallest one.

- The semantics of a greatest fixpoint block $\mathsf{gfp}\{X_1 \doteq \varphi_1, \ldots, X_n \doteq \varphi_n\}$ is the variable assignment $\{X_1/\mathcal{E}_1^\nu, \ldots, X_n/\mathcal{E}_n^\nu\}$, where $(\mathcal{E}_1^\nu, \ldots, \mathcal{E}_n^\nu)$ is the union of all solutions of the fixpoint block, i.e.:

$$(\mathcal{E}_1^\nu, \ldots, \mathcal{E}_n^\nu) \;=\; \bigcup\{(\mathcal{E}_1, \ldots, \mathcal{E}_n) \mid \mathcal{E}_1 = (\varphi_1)^T_{\rho[X_1/\mathcal{E}_1,\ldots,X_n/\mathcal{E}_n]}, \ldots, \mathcal{E}_n = (\varphi_n)^T_{\rho[X_1/\mathcal{E}_1,\ldots,X_n/\mathcal{E}_n]}\}$$

  Again note that, due to syntactic monotonicity, $(\mathcal{E}_1^\nu, \ldots, \mathcal{E}_n^\nu)$ is itself a solution of the fixpoint block, and in this case the largest one.

Finally, the *semantics* of a *μXPath* query $X : \mathcal{F}$ over a sibling tree T is the set $\mathcal{E} \subseteq \Delta^T$ of nodes of T that the fixpoint block $F \in \mathcal{F}$ defining X assigns to X in T. We denote such set $\mathcal{E}$ as $(X : \mathcal{F})^T$. Notice that, since all second-order variables appearing in $\mathcal{F}$ are assigned values in the fixpoint block in which they are defined, we can omit from $(X : \mathcal{F})^T_\rho$ the second order variables assignment ρ, and denote it as $(X : \mathcal{F})^T$.

We observe that, through the use of fixpoints, we can actually capture *RXPath* queries [50, 51], whose node expressions are formed by means of regular expressions over the *XPath* axes, namely:

$$P \longrightarrow \mathsf{child} \mid \mathsf{right} \mid \varphi? \mid P_1; P_2 \mid P_1 \cup P_2 \mid P^* \mid P^-$$

Indeed, node expression of the form $\langle P\rangle\phi$ and $[P]\phi$ with complex P can be considered as abbreviations [42]. First of all, we notice that in expressions of the form P^-, we can apply recursively the following equivalences to push the inverse operator $^-$ inside *RXPath* expressions, until it is applied to *XPath* axes only:

$$\begin{array}{rcl}
(\varphi?)^- &=& \varphi? \\
(P_1; P_2)^- &=& P_2^-; P_1^- \\
(P_1 \cup P_2)^- &=& P_1^- \cup P_2^- \\
(P^*)^- &=& (P^-)^*
\end{array}$$

Also, considering that $\varphi_1 \vee \varphi_2 \equiv \neg(\neg\varphi_1 \wedge \neg\varphi_2)$, and $[P]\varphi \equiv \neg\langle P\rangle\neg\varphi$, we can assume w.l.o.g., that *RXPath* queries are formed as follows:

$$\begin{array}{rcl}
\varphi &\longrightarrow& A \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \langle P\rangle\varphi \\
P &\longrightarrow& \mathsf{child} \mid \mathsf{right} \mid \mathsf{child}^- \mid \mathsf{right}^- \mid \varphi? \mid P_1; P_2 \mid P_1 \cup P_2 \mid P^*
\end{array}$$

Then, we can transform an arbitrary *RXPath* query φ into the *μXPath* query $X_\varphi : \mathcal{F}$, where $\mathcal{F}$ is a set of fixpoint blocks constructed by inductively decomposing φ. Formally, we let $\mathcal{F} = \tau(\varphi)$, where $\tau(\varphi)$ is defined by induction on φ as follows:

$$\begin{array}{rcl}
\tau(A) &=& \{\mathsf{lfp}\{X_A \doteq A\}\} \\
\tau(\neg\varphi') &=& \{\mathsf{lfp}\{X_{\neg\varphi'} \doteq \neg X_{\varphi'}\}\} \cup \tau(\varphi') \\
\tau(\varphi_1 \wedge \varphi_2) &=& \{\mathsf{lfp}\{X_{\varphi_1\wedge\varphi_2} \doteq X_{\varphi_1} \wedge X_{\varphi_2}\}\} \cup \tau(\varphi_1) \cup \tau(\varphi_2) \\
\tau(\langle P\rangle\varphi') &=& \{\mathsf{lfp}\,\tau_p(\langle P\rangle\varphi')\} \cup \tau_t(P) \cup \tau(\varphi')
\end{array}$$

where the function $\tau_p(\cdot)$, which is defined over formulas of the form $\langle P\rangle\varphi'$, returns a set of fixpoint equations, and the function $\tau_t(\cdot)$, which is defined over path expressions P, returns the set of fixpoint blocks corresponding to the node formulas appearing in the tests in P. Specifically, $\tau_p(\langle P\rangle\varphi')$ is defined by induction on the structure of the path expression P as follows:

$$\begin{array}{rcll}
\tau_p(\langle \mathit{axis}\rangle\varphi') &=& \{X_{\langle \mathit{axis}\rangle\varphi'} \doteq \langle \mathit{axis}\rangle X_{\varphi'}\}, & \text{for } \mathit{axis} \in \{\mathsf{child}, \mathsf{right}, \mathsf{child}^-, \mathsf{right}^-\} \\
\tau_p(\langle\varphi''?\rangle\varphi') &=& \{X_{\langle\varphi''?\rangle\varphi'} \doteq X_{\varphi''} \wedge X_{\varphi'}\} & \\
\tau_p(\langle P_1; P_2\rangle\varphi') &=& \{X_{\langle P_1;P_2\rangle\varphi'} \doteq X_{\langle P_1\rangle\langle P_2\rangle\varphi'}\} \cup \tau_p(\langle P_1\rangle\langle P_2\rangle\varphi') \cup \tau_p(\langle P_2\rangle\varphi') & \\
\tau_p(\langle P_1 \cup P_2\rangle\varphi') &=& \{X_{\langle P_1\cup P_2\rangle\varphi'} \doteq X_{\langle P_1\rangle\varphi'} \vee X_{\langle P_2\rangle\varphi'}\} \cup \tau_p(\langle P_1\rangle\varphi') \cup \tau_p(\langle P_2\rangle\varphi') & \\
\tau_p(\langle P^*\rangle\varphi') &=& \{X_{\langle P^*\rangle\varphi'} \doteq X_{\varphi'} \vee \langle P\rangle X_{\langle P^*\rangle\varphi'}\} \cup \tau_p(\langle P\rangle\varphi') &
\end{array}$$

Note that τ_p decomposes inductively only the path expression inside the first $\langle\cdot\rangle$ formula. Hence, the *μXPath* formula $\tau(\varphi)$ is linear in the size of the *RXPath* formula φ.

For example $\langle \mathsf{right}^* \rangle A$ can be expressed as

$$X : \{\mathsf{lfp}\{X \doteq A \vee \langle \mathsf{right} \rangle X\}\}.$$

Instead, $[\mathsf{right}^*]A$, which is equivalent to $\neg\langle \mathsf{right}^* \rangle \neg A$, can be expressed as

$$\begin{aligned} X : \{&\mathsf{lfp}\{X \doteq \neg X_1\}, \\ &\mathsf{lfp}\{X_1 \doteq \neg A \vee \langle \mathsf{right} \rangle X_1\}\}, \end{aligned} \tag{1}$$

which in turn is equivalent to

$$X : \{\mathsf{gfp}\{X \doteq A \wedge [\mathsf{child}]X\}\}.$$

Observe that the form of equation (1) resembles the encoding of the corresponding *RXPath* formula into stratified Monadic Datalog [34].

Considering the above encoding, we can actually extend *μXPath* by allowing as syntactic sugar the use of regular expressions over the axis relations, i.e., instead of axes only, we can allow path expressions of the form

$$P \;\longrightarrow\; \mathsf{child} \mid \mathsf{right} \mid \varphi? \mid P_1; P_2 \mid P_1 \cup P_2 \mid P^* \mid P^-.$$

Finally, we observe that sibling trees are unranked, but in fact this is not really a crucial feature. Indeed, we can move to *binary sibling trees* by considering an additional axis fchild, connecting each node to its first child only, interpreted as

$$\mathsf{fchild}^T \;=\; \{(z, z{\cdot}1) \mid z, z{\cdot}1 \in \Delta^T\}.$$

Using fchild, we can thus re-express the child axis as $\mathsf{fchild}; \mathsf{right}^*$. In the following, we will focus on *μXPath* queries that use only the fchild and right axis relations, and are evaluated over binary sibling trees.

3 2WATAs and their Relationship to *μXPath*

We consider now two-way automata over finite trees and use them as a formal tool to address the problems about *μXPath* in which we are interested in this paper. Specifically, after having introduced the class of two-way weak alternating tree automata (2WATAs), we establish a tight relationship between them and *μXPath* by devising mutual translations between the two formalisms.

3.1 Two-way Weak Alternating Tree Automata

We consider a variant of two-way alternating automata [65] (see also [54, 21]) that run, possibly infinitely, on finite labeled trees (Note that typically, infinite runs of automata are considered in the context of infinite input structures [36], whereas here we consider possibly infinite runs over finite structures.) Specifically, alternating tree automata generalize nondeterministic tree automata, and two-way tree automata generalize ordinary tree automata by being allowed to traverse the tree both upwards and downwards. Formally, let $\mathcal{B}^+(I)$ be the set of positive Boolean formulae over a set I, built inductively by applying $\wedge$ and $\vee$ starting from **true**, **false**, and elements of I. For a set $J \subseteq I$ and a formula $\varphi \in \mathcal{B}^+(I)$, we say that J *satisfies* φ if assigning **true** to the elements in J and **false** to those in $I \setminus J$, makes φ true. We make use of $[-1..k]$ to denote $\{-1, 0, 1, \ldots, k\}$, where k is a positive integer. A *two-way weak alternating tree automaton* (2WATA) running over labeled trees all of whose nodes have at most k successors, is a tuple $\mathbf{A} = (\mathcal{L}, S, s_0, \delta, \alpha)$, where $\mathcal{L}$ is the alphabet of tree labels, S is a finite set of states, $s_0 \in S$ is the initial state, $\delta : S \times \mathcal{L} \to \mathcal{B}^+([-1..k] \times S)$ is the transition function, and α is the acceptance condition discussed below.

The transition function maps a state $s \in S$ and an input label $a \in \mathcal{L}$ to a positive Boolean formula over $[-1..k] \times S$. Intuitively, if $\delta(s, a) = \varphi$, then each pair (c', s') appearing in φ corresponds to a new copy of the automaton going to the direction suggested by c' and starting in state s'. For example, if $k = 2$ and $\delta(s_1, a) = ((1, s_2) \wedge (1, s_3)) \vee ((-1, s_1) \wedge (0, s_3))$, when the automaton is in the state s_1 and is reading the node x labeled by a, it proceeds either by sending off two copies, in the states s_2 and s_3 respectively, to the first successor of x (i.e., $x{\cdot}1$), or by sending off one copy in the state s_1 to the predecessor of x (i.e., $x{\cdot}{-1}$) and one copy in the state s_3 to x itself (i.e., $x{\cdot}0$).

A run of a 2WATA is obtained by resolving all existential choices. The universal choices are left, which gives us a tree. Because we are considering two-way automata, runs can start at arbitrary tree nodes, and need not start at the root. Formally, a *run* of a 2WATA $\mathbf{A}$ over a labeled tree $T = (\Delta^T, \ell^T)$ from a node $x_0 \in \Delta^T$ is, in general, an infinite $\Delta^T \times S$-labeled tree $R = (\Delta^R, \ell^R)$ satisfying:

1. $\varepsilon \in \Delta^R$ and $\ell^R(\varepsilon) = (x_0, s_0)$.
2. Let $\ell^R(r) = (x, s)$ and $\delta(s, \ell^T(x)) = \varphi$. Then there is a (possibly empty) set $S = \{(c_1, s_1), \ldots, (c_n, s_n)\} \subseteq [-1..k] \times S$ such that S satisfies φ, and for each $i \in \{1, \ldots, n\}$, we have that $r \cdot i \in \Delta^R$, $x \cdot c_i \in \Delta^T$, and $\ell^R(r \cdot i) = (x \cdot c_i, s_i)$. In particular, this means that if φ is **true** then r need not have successors, and φ cannot be **false**.

Intuitively, a run R keeps track of all transitions that the 2WATA $\mathbf{A}$ performs on a labeled input tree T: a node r of R labeled by (x, s) describes a copy of $\mathbf{A}$ that is in the state s and is reading the node x of T. The successors of r in the run represent the transitions made by the multiple copies of $\mathbf{A}$ that are being sent off either upwards to the predecessor of x, downwards to one of the successors of x, or to x itself.

2WATAs are called "weak" due to the specific form of the acceptance condition, given in the form of a set $\alpha \subseteq S$ [44]. Specifically, there exists a partition of S into disjoint sets, S_i, such that for each set S_i, either $S_i \subseteq \alpha$, in which case S_i is an *accepting set*, or $S_i \cap \alpha = \emptyset$, in which case S_i is a *rejecting set*. In addition, there exists a partial order $\leq$ on the collection of the S_i's such that, for each $s \in S_i$ and $s' \in S_j$ for which s' occurs in $\delta(s, a)$, for some $a \in \mathcal{L}$, we have $S_j \leq S_i$. Thus, transitions from a state in S_i lead to states in either the same S_i or a lower one. It follows that every infinite path of a run of a 2WATA ultimately gets "trapped" within some S_i. The path is *accepting* if and only if S_i is an accepting set. A run (T_r, r) is *accepting* if all its infinite paths are accepting. A node x is *selected* by a 2WATA $\mathbf{A}$ from a labeled tree T if there exists an accepting run of $\mathbf{A}$ over T from x.

3.2 Binary Trees and Sibling Trees

As mentioned before, we assume that *μXPath* queries are expressed over binary sibling trees, where the left successor of a node corresponds to the fchild axis, and the right successor corresponds to the right axis. To ensure that generic binary trees (i.e., trees of branching degree 2) represent binary sibling trees, we make use of special propositions *ifc*, *irs*, *hfc*, *hrs*. The proposition *ifc* (resp., *irs*) is used to keep track of whether a node *is the first child* (resp., *is the right sibling*) of its predecessor, and *hfc* (resp., *hrs*) is used to keep track of whether a node *has a first child* (resp., *has a right sibling*). In particular, we consider binary trees whose nodes are labeled with subsets of $\Sigma \cup \{\mathit{ifc}, \mathit{irs}, \mathit{hfc}, \mathit{hrs}\}$. We call such a tree $T = (\Delta^T, \ell^T)$ a *well-formed binary tree* if it satisfies the following conditions:

- For each node x of T, if $\ell^T(x)$ contains *hfc*, then $x \cdot 1$ is meant to represent the fchild successor of x and hence $\ell^T(x \cdot 1)$ contains *ifc* but not *irs*. Similarly, if $\ell^T(x)$ contains *hrs*, then $x \cdot 2$ is meant to represent the right successor of x and hence $\ell^T(x \cdot 2)$ contains *irs* but not *ifc*.
- The label $\ell^T(\varepsilon)$ of the root of T contains neither *ifc*, nor *irs*, nor *hrs*. In this way, we restrict the root of T so as to represent the root of a sibling tree.

Notice that every (binary) sibling tree T trivially induces a well-formed binary tree $\pi_b(T)$ obtained by simply adding the labels *ifc*, *irs*, *hfc*, *hrs* in the appropriate nodes.

On the other hand, a well-formed binary tree $T = (\Delta^T, \ell^T)$ induces a sibling tree $\pi_s(T)$. To define $\pi_s(T) = (\Delta^{T_s}, \cdot^{T_s})$, we define, by induction on Δ^T, a mapping π_s from Δ^T to words over $\mathbb{N}$ as follows:

- $\pi_s(\varepsilon) = \varepsilon$;
- if $\mathit{hfc} \in \ell^T(\varepsilon)$, then $\pi_s(1) = 1$;
- if $\mathit{hfc} \in \ell^T(x)$ and $\pi_s(x) = z \cdot n$, with $z \in \mathbb{N}^*$ and $n \in \mathbb{N}$, then $\pi_s(x \cdot 1) = z \cdot n \cdot 1$;
- if $\mathit{hrs} \in \ell^T(x)$ and $\pi_s(x) = z \cdot n$, with $z \in \mathbb{N}^*$ and $n \in \mathbb{N}$, then $\pi_s(x \cdot 2) = z \cdot (n{+}1)$.

Then, we take Δ^{T_s} to be the range of π_s, and we define the interpretation function $\cdot^{T_s}$ as follows: for each $A \in \Sigma_a$, we define $A^{T_s} = \{\pi_s(x) \in \Delta^{T_s} \mid A \in \ell^T(x)\}$. Note that the mapping π_s ignores irrelevant parts of the binary tree, e.g., if the label of a node x does not contain *hfc*, even if x has a 1-successor, such a node is not included in the sibling tree.

$$
\begin{array}{lll}
\text{if } \psi \in CL(\varphi) & \text{then } nnf(\neg\psi) \in CL(\varphi), & \text{if } \psi \text{ is not of the form } \neg\psi' \\
\text{if } \neg\psi \in CL(\varphi) & \text{then } \psi \in CL(\varphi) & \\
\text{if } \psi_1 \wedge \psi_2 \in CL(\varphi) & \text{then } \psi_1,\ \psi_2 \in CL(\varphi) & \\
\text{if } \psi_1 \vee \psi_2 \in CL(\varphi) & \text{then } \psi_1,\ \psi_2 \in CL(\varphi) & \\
\text{if } \langle P \rangle \psi \in CL(\varphi) & \text{then } \psi \in CL(\varphi), & \text{for } P \in \{\mathsf{fchild}, \mathsf{right}, \mathsf{fchild}^-, \mathsf{right}^-\} \\
\text{if } [P] \psi \in CL(\varphi) & \text{then } \psi \in CL(\varphi), & \text{for } P \in \{\mathsf{fchild}, \mathsf{right}, \mathsf{fchild}^-, \mathsf{right}^-\}
\end{array}
$$

Figure 2: Closure of $\mu XPath$ expressions

3.3 From μ*XPath* to 2WATAs

We show now how to construct (i) from each $\mu XPath$ query φ (over binary sibling trees) a 2WATA $\mathbf{A}_\varphi$ whose number of states is linear in $|\varphi|$ and that selects from a tree T precisely the nodes in φ^T, and (ii) from each 2WATA $\mathbf{A}$ a $\mu XPath$ query $\varphi_\mathbf{A}$ of size linear in the number of states of $\mathbf{A}$ that, when evaluated over a tree T, returns precisely the nodes selected by $\mathbf{A}$ from T.

In order to translate $\mu XPath$ to 2WATAs, we need to make use of a notion of syntactic closure, similar to that of Fisher-Ladner closure of a formula of PDL [29]. The *syntactic closure* $CL(X : \mathcal{F})$ of a $\mu XPath$ query $X : \mathcal{F}$ is defined as $\{ifc, irs, hfc, hrs\} \cup CL(\mathcal{F})$, where $CL(\mathcal{F})$ is defined as follows: for each equation $X \doteq \varphi$ in some fixpoint block in $\mathcal{F}$, $\{X, nnf(\varphi)\} \subseteq CL(\mathcal{F})$, where $nnf(\psi)$ denotes the negation normal form of ψ, and then we close the set under sub-expressions (in negation normal form), by inductively applying the rules in Figure 2. It is easy to see that, for a $\mu XPath$ query q, the cardinality of $CL(q)$ is linear in the length of q.

Let $q = X_0 : \mathcal{F}$ be a $\mu XPath$ query. We show how to construct a 2WATA $\mathbf{A}_q$ that, when run over a well-formed binary tree T, selects exactly the nodes in q^T. The 2WATA $\mathbf{A}_q = (\mathcal{L}, S_q, s_q, \delta_q, \alpha_q)$ is defined as follows.

- The alphabet is $\mathcal{L} = 2^{\Sigma \cup \{ifc, irs, hfc, hrs\}}$. This corresponds to labeling each node of the tree with a truth assignment to the atomic propositions, including the special ones that encode information about the predecessor node and about whether the children are significant.
- The set of states is $S_q = CL(q)$. Intuitively, when the automaton is in a state $\psi \in CL(q)$ and visits a node x of the tree, it checks that the node expression ψ holds in x.
- The initial state is $s_q = X_0$.
- The transition function δ_q is defined as follows:
  1. For each $\lambda \in \mathcal{L}$, and each $\sigma \in \Sigma \cup \{ifc, irs, hfc, hrs\}$,
  $$
  \begin{array}{rcl}
  \delta_q(\sigma, \lambda) & = & \begin{cases} \mathbf{true}, & \text{if } \sigma \in \lambda \\ \mathbf{false}, & \text{if } \sigma \notin \lambda \end{cases} \\
  \delta_q(\neg\sigma, \lambda) & = & \begin{cases} \mathbf{true}, & \text{if } \sigma \notin \lambda \\ \mathbf{false}, & \text{if } \sigma \in \lambda \end{cases}
  \end{array}
  $$
  Such transitions check the truth value of atomic propositions, and of their negations in the current node of the tree, by simply checking whether the node label contains the proposition or not.
  2. For each $\lambda \in \mathcal{L}$ and each formula $\psi \in CL(q)$, the automaton inductively decomposes ψ and moves to appropriate states to check the sub-expressions as follows:
  $$
  \begin{array}{rcl}
  \delta_q(\psi_1 \wedge \psi_2, \lambda) & = & (0, \psi_1) \ \wedge \ (0, \psi_2) \\
  \delta_q(\psi_1 \vee \psi_2, \lambda) & = & (0, \psi_1) \ \vee \ (0, \psi_2) \\
  \\
  \delta_q(\langle \mathsf{fchild} \rangle \psi, \lambda) & = & (0, hfc) \wedge (1, \psi) \\
  \delta_q(\langle \mathsf{right} \rangle \psi, \lambda) & = & (0, hrs) \wedge (2, \psi) \\
  \delta_q(\langle \mathsf{fchild}^- \rangle \psi, \lambda) & = & (0, ifc) \wedge (-1, \psi) \\
  \delta_q(\langle \mathsf{right}^- \rangle \psi, \lambda) & = & (0, irs) \wedge (-1, \psi) \\
  \\
  \delta_q([\mathsf{fchild}] \psi, \lambda) & = & (0, \neg hfc) \vee (1, \psi) \\
  \delta_q([\mathsf{right}] \psi, \lambda) & = & (0, \neg hrs) \vee (2, \psi) \\
  \delta_q([\mathsf{fchild}^-] \psi, \lambda) & = & (0, \neg ifc) \vee (-1, \psi) \\
  \delta_q([\mathsf{right}^-] \psi, \lambda) & = & (0, \neg irs) \vee (-1, \psi)
  \end{array}
  $$

3. Let $X \doteq \varphi$ be an equation in one of the blocks of $\mathcal{F}$. Then, for each $\lambda \in \mathcal{L}$, we have $\delta_q(X, \lambda) = (0, \varphi)$.

- To define the weakness partition of $\mathbf{A}_q$, we partition the expressions in $CL(q)$ according to the partial order on the fixpoint blocks in $\mathcal{F}$. Namely, we have one element of the partition for each fixpoint block $F \in \mathcal{F}$. Such an element is formed by all expressions (including variables) in $CL(q)$ in which at least one variable defined in F occurs and no variable defined in a fixpoint block F' with $F \prec F'$ occurs. In addition, there is one element of the partition consisting of all expressions in which no variable occurs. Then the acceptance condition α_q is the union of all elements of the partition corresponding to a greatest fixpoint block. Observe that the partial order on the fixpoint blocks in $\mathcal{F}$ guarantees that the transitions of $\mathbf{A}_q$ satisfy the weakness condition. In particular, each element of the weakness partition is either contained in α_q or disjoint from α_q. This guarantees that an accepting run cannot get trapped in a state corresponding to a least fixpoint block, while it is allowed to stay forever in a state corresponding to a greatest fixpoint block.

Theorem 1 *Let q be a μXPath query. Then:*

1. *The number of states of the corresponding 2WATA $\mathbf{A}_q$ is linear in the size of q.*
2. *For every binary sibling tree T, a node x of T is in q^T iff $\mathbf{A}_q$ selects x from the well-formed binary tree $\pi_b(T)$ induced by T.*

Proof. Item 1 follows immediately from the fact that the size of $CL(q)$ is linear in the size of q. We turn to item 2. In the proof, we blur the distinction between T and $\pi_b(T)$, denoting it simply as T, since the two trees are identical, except for the additional labels in $\pi_b(T)$, which are considered by $\mathbf{A}_q$ but ignored by q.

Let $q = X : \mathcal{F}$. We show by simultaneous induction on the structure of $\mathcal{F}$ and on the nesting of fixpoint blocks, that for every expression $\psi \in CL(\mathcal{F})$ and for every node x of T, we have that $\mathbf{A}_q$, when started in state ψ, selects x from T if and only if $x \in \psi^T$.

- Indeed, when ψ is an atomic proposition, then the claim follows immediately by making use of the transitions in item 1 of the definition of δ.
- When $\psi = \psi_1 \wedge \psi_2$ or $\psi = \psi_1 \vee \psi_2$, the claim follows by inductive hypothesis, making use of the first two transitions in item 2.
- When $\psi = \langle \mathsf{fchild} \rangle \psi_1$, the 2WATA checks that x has a first child $y = x \cdot 1$, and moves to y checking that y is selected from T starting in state ψ_1. Then, by induction hypothesis, we have that $y \in \psi_1^T$, and the claim follows.

  The cases of $\psi = \langle \mathsf{right} \rangle \psi_1$, $\psi = \langle \mathsf{fchild}^- \rangle \psi_1$, and $\psi = \langle \mathsf{right}^- \rangle \psi_1$ are analogous.
- When $\psi = [\mathsf{fchild}] \psi_1$, the 2WATA checks that either x does not have a first child, or that the first child $y = x \cdot 1$ is selected from T starting in state ψ_1. Then, by induction hypothesis, we have that $y \in \psi_1^T$, and the claim follows.

  The cases of $\psi = [\mathsf{right}] \psi_1$, $\psi = [\mathsf{fchild}^-] \psi_1$, and $\psi = [\mathsf{right}^-] \psi_1$ are analogous.
- When $\psi = X_1$, let $X_1 \doteq \psi_1$ be the equation defining X_1. Then according to the transitions in item 3, the 2WATA checks that x is selected from T starting in state ψ_1. The definition of the 2WATA acceptance condition α_q guarantees that, if X_1 is defined in a least fixpoint block then an accepting run cannot get trapped in the element S_i of the weakness partition containing X_1; instead, if X_1 is defined in a greatest fixpoint block then an accepting run is allowed to stay forever in S_i.

  We consider only the least fixpoint case; the greatest fixpoint case is similar. If there is an accepting run, it will go through states in S_i (including X_1) only a finite number of times, and on each of its paths it will get to a node y in a state $\xi \in S_j$, where S_j strictly precedes S_i, i.e., with $S_j \leq S_i$ and $S_j \neq S_i$. By induction on the nesting of fixpoint blocks (corresponding to the elements of the state partition), we have that $\mathbf{A}_q$, when started in state ξ, selects y from T if and only if $y \in \xi^T$. Then, since the automaton state X_1 is not contained in α_q, the acceptance condition ensures that the transition in item 3 is applied only a finite number of times, and considering the least fixpoint semantics, by structural induction we get that $x \in X_1^T$.

For the other direction, we show that, if $x \in X_1^T$, then $\mathbf{A}_q$ has an accepting run $R = (\Delta^R, \ell^R)$ witnessing that x is selected from T starting in state X_1. We can define the run R by exploiting the equation $X_1 \doteq \psi_1$ to make the transition according to item 3, and following structural induction to decompose formulas, ensuring that, for all nodes $y \in \Delta^R$ with $\ell^R(y) = (x', \psi')$ we have that $x' \in \psi'^T$. In particular, we resolve the nondeterminism coming from disjunctions in the transition function of $\mathbf{A}_q$ (in turn coming from disjunctions in q) by choosing the disjunct that is satisfied in the node of T. Consider a node $y \in \Delta^R$, with $\ell^R(y) = (x', X_1)$. We say that y is an *escape node* if $x' \in X_1^T$ because $x' \in \xi^T$, where ξ is a subformula of ψ_1 in which X_1 does not appear. Since X_1 is defined by a least fixpoint, all the nodes $y \in \Delta^R$ with $\ell^R(y) = (x', X_1)$ eventually reach an escape node. Hence, the run (Δ^R, ℓ^R) does not loop on X_1, and hence does not violate the acceptance condition.

The claim then follows since the initial state of $\mathbf{A}_q$ is q. $\square$

We observe that, although the number of states of $\mathbf{A}_q$ is linear in the size of q, the alphabet of $\mathbf{A}_q$ is the powerset of that of q, and hence the transition function and the entire $\mathbf{A}_q$ is exponential in the size of q. However, as we will show later, this does not affect the complexity of query evaluation, query containment, and more in general reasoning over queries.

3.4 From 2WATAs to μ*XPath*

We show now how to convert 2WATAs into μ*XPath* queries while preserving the set of nodes selected from (well formed) binary trees.

Consider a 2WATA $\mathbf{A} = (\mathcal{L}, S, s_0, \delta, \alpha)$, where $\mathcal{L} = 2^{\Sigma \cup \{ifc, irs, hfc, hrs\}}$, and let $S = \cup_{i=1}^k S_i$ be the weakness partition of $\mathbf{A}$. We define a translation π as follows.

- For a positive Boolean formula $f \in \mathcal{B}^+([-1..2] \times S)$, we define a μ*XPath* node expression $\pi(f)$ inductively as follows:

$$
\begin{array}{rlrl}
\pi(\mathbf{false}) = \mathsf{false} & & \pi(\mathbf{true}) = \mathsf{true} \\
\pi((1,s)) = \langle \mathsf{fchild} \rangle s & & \pi((2,s)) = \langle \mathsf{right} \rangle s \\
\pi((0,s)) = s & & \pi((-1,s)) = (ifc \wedge \langle \mathsf{fchild}^- \rangle s) \vee (irs \wedge \langle \mathsf{right}^- \rangle s) \\
\pi(f_1 \wedge f_2) = \pi(f_1) \wedge \pi(f_2) & & \pi(f_1 \vee f_2) = \pi(f_1) \vee \pi(f_2)
\end{array}
$$

- For each state $s \in S$, we define a μ*XPath* equation $\pi(s)$ as follows:

$$
s \ \doteq \ \textstyle\bigvee_{\lambda \in \mathcal{L}} (\tilde{\lambda} \wedge \pi(\delta(s, \lambda))),
$$

  where $\tilde{\lambda} = (\bigwedge_{a \in \lambda} a) \wedge (\bigwedge_{a \in (\sigma \setminus \lambda)} \neg a)$.
- For each element S_i of the weakness partition, we define a μ*XPath* fixpoint block as follows:

$$
\pi(S_i) \ = \ \begin{cases} \mathsf{gfp}\{\pi(s) \mid s \in S_i\}, & \text{if } S_i \subseteq \alpha \\ \mathsf{lfp}\{\pi(s) \mid s \in S_i\}, & \text{if } S_i \cap \alpha = \emptyset \end{cases}
$$

- Finally, we define the μ*XPath* query $\pi(\mathbf{A})$ as:

$$
\pi(\mathbf{A}) \ = \ s_0 : \{\pi(S_1), \ldots, \pi(S_k)\}.
$$

Theorem 2 *Let* $\mathbf{A}$ *be a 2WATA. Then:*

1. *The length of the μXPath query $\pi(\mathbf{A})$ corresponding to $\mathbf{A}$ is linear in the size of $\mathbf{A}$.*
2. *For every binary sibling tree T, we have that $\mathbf{A}$ selects a node x from $\pi_b(T)$ iff x is in $(\pi(\mathbf{A}))^T$.*

Proof. Item 1 follows immediately from the above construction. We only observe that in defining the μ*XPath* equations $\pi(s)$ for a state $s \in S$, we have a disjunction over the label set $\mathcal{L}$, which is exponential in the number of atomic propositions in Σ. On the other hand, the transition function of the 2WATA itself needs to deal with the elements of $\mathcal{L}$, and hence is also exponential in the size of Σ.

We turn to item 2. We again ignore the distinction between T and $\pi_b(T)$. Consider a variation of the construction specified in Section 3.3, in which we replace the transitions for conjunction and disjunction, respectively with

$$
\begin{array}{rcl}
\delta_q(\psi_1 \wedge \psi_2, \lambda) & = & \delta_q(\psi_1) \wedge \delta_q(\psi_2) \\
\delta_q(\psi_1 \vee \psi_2, \lambda) & = & \delta_q(\psi_1) \vee \delta_q(\psi_2)
\end{array}
$$

and the transitions $\delta_q(X, \lambda) = (0, \varphi)$ for an equation $X \doteq \varphi$ with the transitions $\delta_q(X, \lambda) = \delta_q(\varphi)$. It is easy to check that the 2WATA obtained in this way is equivalent to the one specified in Section 3.3. On the other hand, by applying the modified construction to the *μXPath* query $\pi(\mathbf{A})$, we obtain a 2WATA $\mathbf{A}_{\pi(\mathbf{A})}$ that on well-formed binary trees is equivalent to $\mathbf{A}$. Indeed, for every transition of $\mathbf{A}$, the construction introduces in $\mathbf{A}_{\pi(\mathbf{A})}$ a corresponding transition. Notice that, for an atom of the form $(-1, s)$ appearing in the right-hand side of a transition of $\mathbf{A}$, we obtain in $\pi(\mathbf{A})$ a *μXPath* expression $\varphi = (\mathit{ifc} \land \langle \mathsf{fchild}^- \rangle s) \lor (\mathit{irs} \land \langle \mathsf{right}^- \rangle s)$. Then we have that $\delta_{\pi(\mathbf{A})}(\varphi, \lambda) = ((0, \mathit{ifc}) \land (-1, s)) \lor (0, \mathit{irs}) \land (-1, s)$. In both cases where λ contains *ifc* or *irs*, this expression results in $(-1, s)$, while in the root (where λ contains neither *ifc* nor *irs*) the expression results in **false**, thus yielding a transition equivalent to the one resulting from the atom $(-1, s)$ of $\mathbf{A}$. Hence, by Theorem 2, we get the claim. □

4 Acceptance and Non-Emptiness for 2WATAs

We provide now computationally optimal algorithms for deciding the acceptance and non-emptiness problems for 2WATAs.

4.1 The Acceptance Problem

Given a 2WATA $\mathbf{A} = (\mathcal{L}, S, s_0, \delta, \alpha)$, a labeled tree $T = (\Delta^T, \ell^T)$, and a node $x_0 \in \Delta^T$, we'd like to know whether x_0 is selected by $\mathbf{A}$ from T. This is called the *acceptance problem*. We follow here the approach of [44], and solve the acceptance problem by first taking a product $\mathbf{A} \times T_{x_0}$ of $\mathbf{A}$ and T from x_0. This product is an alternating automaton over a one letter alphabet $\mathcal{L}_0$, consisting of a single letter, say a. This product automaton simulates a run of $\mathbf{A}$ on T from x_0. The product automaton is $\mathbf{A} \times T_{x_0} = (\mathcal{L}_0, S \times \Delta^T, (s_0, x_0), \delta', \alpha \times \Delta^T)$, where δ' is defined as follows:

- $\delta'((s, x), a) = \Theta_x(\delta(s, \ell^T(x)))$, where Θ_x is the substitution that replaces a pair (c, t) in $\delta(s, \ell^T(x))$ by the pair $(t, x{\cdot}c)$ if $x{\cdot}c \in \Delta^T$, and by **false** otherwise.

Note that the size of $\mathbf{A} \times T_{x_0}$ is simply the product of the size of $\mathbf{A}$ and the size of T, and that the only elements of $\mathcal{L}$ that are used in the construction of $\mathbf{A} \times T_{x_0}$ are those that appear among the labels of T. Note also that $\mathbf{A} \times T_{x_0}$ can be viewed as a weak alternating word automaton running over the infinite word a^ω, as by taking the product with T we have eliminated all directions. In fact, one can simply view $\mathbf{A} \times T_{x_0}$ as a 2-player infinite game; see [36].

We can now state the relationship between $\mathbf{A} \times T_{x_0}$ and $\mathbf{A}$, which is essentially a restatement of Proposition 3.2 in [44].

Proposition 3 *Node x_0 is selected by $\mathbf{A}$ from T iff $\mathbf{A} \times T_{x_0}$ accepts a^ω.*

The advantage of Proposition 3 is that it reduces the acceptance problem to the question of whether $\mathbf{A} \times T_{x_0}$ accepts a^ω. This problem is referred to in [44] as the "one-letter nonemptiness problem". It is shown there that this problem can be solved in time that is linear in the size of $\mathbf{A} \times T_{x_0}$ by an algorithm that imposes an evaluation of and-or trees over a decomposition of the automaton state space into maximal strongly connected components, and then analyzes these strongly connected components in a bottom-up fashion. The result in [44] is actually stronger; the algorithm there computes in linear time the set of states from which the automaton accepts a^ω, that is, the states that yield acceptance if chosen as initial states. We therefore obtain the following result about the acceptance problem.

Theorem 4 *Given a 2WATA $\mathbf{A}$ and a labeled tree T, we can compute the set of nodes selected by $\mathbf{A}$ from T in time that is linear in the product of the sizes of $\mathbf{A}$ and T.*

Proof. We constructed above the product automaton $\mathbf{A} \times T_{x_0} = (\mathcal{L}_0, S \times \Delta^T, (s_0, x_0), \delta', \alpha \times \Delta^T)$. Note that the only place in this automaton where x_0 plays a role is in the initial state (s_0, x_0). That is, replacing the initial state by (s_0, x) for another node $x \in \Delta^T$ gives us the product automaton $\mathbf{A} \times T_x$. As pointed out above, the bottom-up algorithm of [44] actually computes the set of states from which the automaton accepts a^ω. Thus, x is selected by $\mathbf{A}$ from T iff the state (s_0, x) of the product automaton is accepting. That is, to compute the set of nodes of T selected by $\mathbf{A}$, we construct the product automaton, compute states from which the automaton accepts, and then select all nodes x such that the automaton accepts from (s_0, x). □

Thus, Theorem 4 provides us with a query-evaluation algorithms for 2WATA queries, which is linear both in the size of the tree and in the size of the automaton.

4.2 The Nonemptiness Problem

The *nonemptiness problem* for 2WATAs consists in determining, for a given 2WATA $\mathbf{A}$ whether it selects the root ε from some tree T. In this case we say that $\mathbf{A}$ *accepts* T. This problem is solved in [71] for 2WATAs (actually, for a more powerful automata model) over infinite trees, using rather sophisticated automata-theoretic techniques. Here we solve this problem over finite trees, which requires less sophisticated techniques, and, consequently, is much easier to implement.

In order to decide non-emptiness of 2WATAs, we resort to a conversion to standard one-way nondeterministic tree automata [22]. A one-way nondeterministic tree automaton (NTA) is a tuple $\mathbf{A} = (\mathcal{L}, S, s_0, \delta)$, analogous to a 2WATA, except that (i) the acceptance condition α is empty and has been dropped from the tuple, (ii) the directions -1 and 0 are not used in δ and, (iii) for each state $s \in S$ and letter $a \in \mathcal{L}$, the positive Boolean formula $\delta(s, a)$, when written in DNF, does not contain a disjunct with two distinct atoms (c, s_1) and (c, s_2) with the same direction c. In other words, each disjunct corresponds to sending at most one "subprocess" in each direction. We also allow an NTA to have a *set* of initial states, requiring that starting with *one* initial state must lead to acceptance.

While for 2WATAs we have separate input tree and run tree, for NTAs we can assume that the run of the automaton over an input tree $T = (\Delta^T, \ell^T)$ is an S-labeled tree $R = (\Delta^T, \ell^R)$, which has the same underlying tree as T, and thus is finite, but is labeled by states in S. Nonemptiness of NTAs is known to be decidable [26]. As shown there, the set Acc of states of an NTA that leads to acceptance can be computed by a simple fixpoint algorithm:

(1) Initially: $Acc = \emptyset$.

(2) At each iteration: $Acc := Acc \cup \{s \mid \alpha_{Acc} \models \delta(s, a) \text{ for some } a \in \mathcal{L}\}$, where α_X is the truth assignment that maps (c, s) to true precisely when $s \in X$,

It is known that such an algorithm can be implemented to run in linear time [27]. Thus, to check nonemptiness we compute Acc and check that it has nonempty intersection with the set of initial states.

It remains to describe the translation of 2WATAs to NTAs. Given a 2WATA $\mathbf{A} = (\mathcal{L}, S, s_0, \delta, \alpha)$ and an input tree $T = (\Delta^T, \ell^T)$ as above, let $\mathcal{T} = 2^{S \times [-1..k] \times S}$; that is, an element of $\mathcal{T}$ is a set of transitions of the form (s, i, s'). A *strategy for* $\mathbf{A}$ *on* T is a mapping $\tau : \Delta^T \to \mathcal{T}$. Thus, each label in a strategy is an edge-$[-1..k]$-labeled directed graph on S. For each label $\zeta \subseteq S \times [-1..k] \times S$, we define $state(\zeta) = \{u \mid (u, i, v) \in \zeta\}$, i.e., $state(\zeta)$ is the set of sources in the graph ζ. In addition, we require the following:

(1) for each node $x \in \Delta^T$ and each state $s \in state(\tau(x))$, the set $\{(c, s') \mid (s, c, s') \in \tau(x)\}$ satisfies $\delta(s, \ell^T(x))$ (thus, each label can be viewed as a strategy of satisfying the transition function), and

(2) for each node $x \in \Delta^T$, and each edge $(s, i, s') \in \tau(x)$, we have that $s' \in state(\tau(x \cdot i))$.

A *path* β in the strategy τ is a maximal sequence $(u_0, s_0), (u_1, s_1), \ldots$ of pairs from $\Delta^T \times S$ such that $u_0 = \varepsilon$ and, for all $i \geq 0$, there is some $c_i \in [-1..k]$ such that $(s_i, c_i, s_{i+1}) \in \tau(u_i)$ and $u_{i+1} = u_i \cdot c_i$. Thus, β is obtained by following transitions in the strategy. The path β is accepting if the path $s_0, s_1, \ldots$ is accepting. The strategy τ is *accepting* if all its paths are accepting.

Proposition 5 ([71]) *A 2WATA* $\mathbf{A}$ *accepts an input tree* T *iff* $\mathbf{A}$ *has an accepting strategy for* T.

We have thus succeeded in defining a notion of run for alternating automata that will have the same tree structure as the input tree. We are still facing the problem that paths in a strategy tree can go both up and down. We need to find a way to restrict attention to uni-directional paths. For this we need an additional concept.

Let $\mathcal{E}$ be the set of relations of the form $S \times \{0, 1\} \times S$. Thus, each element in $\mathcal{E}$ is an edge-$\{0, 1\}$-labeled directed graph on S. An *annotation* for $\mathbf{A}$ on T with respect to a strategy τ is a mapping $\eta : \Delta^T \to 2^{S \times \{0,1\} \times S}$. Edge labels need not be unique; that is, an annotation can contain both triples $(s, 0, s')$ and $(s, 1, s')$. We require η to satisfy some closure conditions for each node $x \in \Delta^T$. Intuitively, these conditions say that η contains all relevant information about finite paths in τ. Thus, an edge (s, c, s') describes a path from s to s', where $c = 1$ if this path goes through α. The conditions are:

(1) if $(s, c, s') \in \eta(x)$ and $(s', c', s'') \in \eta(x)$, then $(s, c'', s'') \in \eta(x)$ where $c'' = \max\{c, c'\}$,

(2) if $(s, 0, s') \in \tau(x)$ then $(s, c, s') \in \eta(x)$, where $c = 1$ if $s' \in \alpha$ and $c = 0$ otherwise,

(3) if $y = x{\cdot}i$ (for $i > 0$), $(s, i, s') \in \tau(x)$, $(s', c, s'') \in \eta(y)$, and $(s'', -1, s''') \in \tau(y)$, then $(s, c', s''') \in \eta(x)$, where $c' = 1$ if $s \in \alpha$, $c = 1$, or $s''' \in \alpha$, and $c' = 0$ otherwise.

(4) if $x = y{\cdot}i$ (for $i > 0$), $(s, -1, s') \in \tau(x)$, $(s', c, s'') \in \eta(y)$, and $(s'', i, s''') \in \tau(y)$, then $(s, c', s''') \in \eta(x)$, where $c' = 1$ if $s' \in \alpha$, $c = 1$, or $s''' \in \alpha$, and $c' = 0$ otherwise.

The annotation η is *accepting* if for every node $x \in \Delta^T$ and state $s \in S$, if $(s, c, s) \in \eta(x)$, then $c = 1$. In other words, η is accepting if *all* cycles visit accepting states.

Proposition 6 ([71]) *A 2WATA* $\mathbf{A}$ *accepts an input tree* T *iff* $\mathbf{A}$ *has a strategy* τ *on* T *and an accepting annotation* η *of* τ.

Consider now an *annotated tree* $(\Delta^T, \ell^T, \tau, \eta)$, where τ is a strategy tree for $\mathbf{A}$ on (Δ^T, ℓ^T) and η is an annotation of τ. We say that $(\Delta^T, \ell^T, \tau, \eta)$ is *accepting* if η is accepting.

Theorem 7 *Let* $\mathbf{A}$ *be a 2WATA. Then there is an NTA* $\mathbf{A}_n$ *such that* $\mathscr{L}(\mathbf{A}) = \mathscr{L}(\mathbf{A}_n)$*. The number of states of* $\mathbf{A}_n$ *is at most exponential in the number of states of* $\mathbf{A}$.

Proof. The proof follows by specializing the construction in [71] to 2WATAs on finite trees.

Let $\mathbf{A} = (\mathcal{L}, S, s_0, \delta, \alpha)$ and let the input tree be $T = (\Delta^T, \ell^T)$. The automaton $\mathbf{A}_n$ guesses mappings $\tau : \Delta^T \to \mathcal{T}$ and $\eta : \Delta^T \to \mathcal{E}$ and checks that τ is a strategy for $\mathbf{A}$ on T and η is an accepting annotation for $\mathbf{A}$ on T with respect to τ. The state space of $\mathbf{A}_n$ is $\mathcal{T} \times \mathcal{E}$; intuitively, before reading the label of a node x, $\mathbf{A}_n$ needs to be in state $(\tau(x), \eta(x))$. The transition function of $\mathbf{A}_n$ checks that *state*, τ, and η satisfies all the required conditions.

Formally, $\mathbf{A}_n = (\mathcal{L}, Q, Q_0, \rho)$, where

- $Q = \mathcal{T} \times \mathcal{E}$,
- We first define a function $\rho : \mathcal{T} \times \mathcal{E} \times \mathcal{L} \times [1..k] \to 2^{\mathcal{T} \times \mathcal{E}}$:

  We have that $(r', R') \in \rho(r, R, a, i)$ if

  1. if $(s, i, s') \in r$, then $s' \in \mathit{state}(r')$, if $(s, -1, s') \in r'$, then $s' \in \mathit{state}(r)$,
  2. if $(s, c, s) \in R$, then $c = 1$,
  3. for each $s \in \mathit{state}(r)$, the set $\{(c, s') \mid (s, c, s) \in r\}$ satisfies $\delta(s, a)$,
  4. if $(s, c, s') \in R'$ and $(s', c', s'') \in R'$, then $(s, c'', s'') \in R'$ where $c'' = \max\{c, c'\}$,
  5. if $(s, 0, s') \in r$, then $(s, c, s') \in R$, where $c = 1$ if $s' \in \alpha$ and $c = 0$ otherwise,
  6. if $(s, i, s') \in r$, $(s', c, s'') \in R'$, and $(s'', -1, s''') \in r'$, then $(s, c', s''') \in r$, where $c' = 1$ if $s \in \alpha$, $c = 1$, or $s''' \in \alpha$, and $c' = 0$ otherwise,
  7. if $(s, -1, s') \in r'$, $(s', c, s'') \in R$, and $(s'', i, s''') \in r$, then $(s, c', s''') \in R'$, where $c' = 1$ if either $s' \in \alpha$, $c = 1$, or $s''' \in \alpha$, and $c' = 0$ otherwise.

  Intuitively, the transition function ρ checks that all conditions on the strategy and annotation hold, except for the condition on the strategy at the root.

  We now define $\rho(r, R, a) = \bigvee_{1 \leq i \leq k} \bigwedge_{(r', R') \in \rho(r, R, a, i)} (i, r', R')$. If, however, we have that $(\emptyset, \emptyset) \in \rho(r, R, a, i)$ for all $1 \leq i \leq k$, then we define $\rho(r, R, a) = \mathbf{true}$.
- The set of initial states is $Q_0 = \{(r, R) \mid s_0 \in \mathit{state}(r)$ and there is no transition $(s, -1, s') \in r\}$.

It follows from the argument in [71] that $\mathbf{A}_n$ accepts a tree T iff $\mathbf{A}$ has a strategy tree on T and an accepting annotation of that strategy. $\square$

We saw earlier that nonemptiness of NTAs can be checked in linear time. From Proposition 6 we now get:

Theorem 8 *Given a 2WATA* $\mathbf{A}$ *with* n *states and an input alphabet with* m *elements, deciding nonemptiness of* $\mathbf{A}$ *can be done in time exponential in* n *and linear in* m.

The key feature of the state space of $\mathbf{A}_n$ is the fact that states are pairs consisting of subsets of $S \times \{0,1\} \times S$ and $S \times [-1..k] \times S$. Thus, a set of states of $\mathbf{A}_n$ can be described by a Boolean function on the domain $S^4 \times \{0,1\} \times [-1..k]$. Similarly, the transition function of $\mathbf{A}_n$ can also be described as a Boolean function. Such functions can be represented by binary decision diagrams (BDDs) [12], enabling a symbolic implementation of the fixpoint algorithm discussed above.

We note that the framework of [71] also converts a two-way alternating tree automaton (on infinite trees) to a nondeterministic tree automaton (on infinite trees). The state space of the latter, however, is considerably more complex than the one obtained here. In fact, the infinite-tree automata-theoretic approach so far has resisted attempts at practically efficient implementation [62, 66], due to the use of Safra's determinization construction [60] and parity games [40]. This makes it very difficult in practice to apply the symbolic approach in the infinite-tree setting.

5 Query Evaluation and Reasoning on μ*XPath*

We now exploit the correspondence between μ*XPath* and 2WATAs we establish the main characteristics of μ*XPath* as a query language over sibling trees. We recall that sibling trees can be encoded in (well-formed) binary trees in linear time, and hence we blur the distinction between the two.

5.1 Query Evaluation

We can evaluate μ*XPath* queries over sibling trees by exploiting the correspondence with 2WATAs, obtaining the following complexity characterization.

Theorem 9 *Given a (binary) sibling tree T and a μXPath query q, we can compute q^T in time that is linear in the number of nodes of T (data complexity) and in the size of q (query complexity).*

Proof. By Theorem 1, we can construct from q a 2WATA $\mathbf{A}_q$ whose number of states is linear in the size of q. On the other hand, the well-formed binary tree $\pi_b(T)$ induced by T can be built in linear time. By Theorem 4, we can evaluate $\mathbf{A}_q$ over $\pi_b(T)$ in linear time in the product of the sizes of $\mathbf{A}_q$ and $\pi_b(T)$ by constructing the product automaton $\mathbf{A}_q \times \pi_b(T)_x$ (where x is an arbitrary node of $\pi_b(T)$). Notice that, while the alphabet of $\mathbf{A}_q$ is the powerset of the alphabet of q, in $\mathbf{A}_q \times \pi_b(T)_x$ only the labels that actually appear in $\pi_b(T)$ are used, hence the claim follows. □

5.2 Query Satisfiability and Containment

We now turn our attention to query satisfiability and containment. A μ*XPath* query q is *satisfiable* if there is a sibling tree T_s and a node x in T_s that is returned when q is evaluated over T_s. A μ*XPath* query q_1 is *contained* in a μ*XPath* query q_2 if for every sibling tree T_s, the query q_1 selects a subset of the nodes of T_s selected by q_2. Checking satisfiability and containment of queries is crucial in several contexts, such as query optimization, query reformulation, knowledge-base verification, information integration, integrity checking, and cooperative answering [37, 13, 53, 19, 47, 46, 52] Obviously, query containment is also useful for checking equivalence of queries, i.e., verifying whether for all databases the answer to a query is the same as the answer to another query. For a summary of results on query containment in graph and tree-structured data, see [16, 6, 5].

Satisfiability of a μ*XPath* query q can be checked by checking the non-emptiness of a 2WATA $\mathbf{A}_q^{wf}$. Such automaton $\mathbf{A}_q^{wf}$ accepts a binary tree T (i.e., selects its root ε) if and only if *(i)* T is well-formed (and hence correspond to a binary sibling tree), and *(ii)* $\mathbf{A}_q$ selects a non-deterministically chosen node x from T. Formally, given $\mathbf{A}_q = (\mathcal{L}, S_q, s_q, \delta_q, \alpha_q)$, the 2WATA $\mathbf{A}_q^{wf} = (\mathcal{L}, S, s_{ini}, \delta, \alpha)$ is defined as follows:

- The set of states is $S = S_q \cup \{s_{ini}, s_{struc}, s_q^0\}$, where s_{ini} is the initial state, s_{struc} is used to check structural properties of well-formed trees, and s_q^0 is used to non-deterministically move to a node from which to check q.
- The transition function is constituted by all transitions in δ_q, plus the following transitions:
    1. For each $\lambda \in \mathcal{L}$, there is a transition

    $$\delta(s_{ini}, \lambda) = (0, s_{struc}) \wedge (0, s_q^0)$$

    Such transitions move both to state s_{struc}, from which structural properties of the tree are verified, and to state s_q^0 used to non-deterministically choose a node to be selected by $\mathbf{A}_q$.

2. For each $\lambda \in \mathcal{L}$, there is a transition

$$\begin{aligned}\delta(s_{struc}, \lambda) \;=\; & ((0, \neg \mathit{hfc}) \vee ((1, \mathit{ifc}) \wedge (1, \neg \mathit{irs}) \wedge (1, s_{struc}))) \wedge \\ & ((0, \neg \mathit{hrs}) \vee ((2, \mathit{irs}) \wedge (2, \neg \mathit{ifc}) \wedge (2, s_{struc})))\end{aligned}$$

   Such transitions check that, *(i)* for a node labeled with *hfc*, its left child is labeled with *ifc* but not with *irs*, and satisfies the same structural property; and *(ii)* for a node labeled with *hrs*, its right child is labeled with *irs* but not with *ifc*, and satisfies the same structural property.

3. For each $\lambda \in \mathcal{L}$, there is a transition

$$\delta(s_q^0, \lambda) \;=\; (0, s_q) \vee (1, s_q^0) \vee (2, s_q^0)$$

   Such transitions non-deterministically either verify that q holds at the current node by moving to the initial state s_q of $\mathbf{A}_q$, or move downwards in the tree to repeat the same checks at the children.

- The set of accepting states is $\alpha = \alpha_q \cup \{s_{struc}\}$. The states s_{ini} and s_{struc} form each a single element of the partition of states, where $\{s_{ini}\}$ precedes all other elements, and $\{s_{struc}\}$ follows them.

As for the size of $\mathbf{A}_q^{wf}$, by Theorem 1, and considering that the additional states and transitions in $\mathbf{A}_q^{wf}$ are of constant size, which does not depend on q, we get that the number of states of $\mathbf{A}_q^{wf}$ is linear in the size of q.

Proposition 10 *Let q be a μXPath query, and $\mathbf{A}_q^{wf}$ the corresponding 2WATA constructed as above. Then $\mathbf{A}_q^{wf}$ is nonempty if and only if q is satisfiable.*

Proof. "$\Rightarrow$" Let $\mathbf{A}_q^{wf}$ accept a binary tree T. Consider the subtree T' of T where every subtree rooted at a node in which neither *hfc* nor *hrs* holds is pruned away. By Transitions (1), and (2) in the definition of $\mathbf{A}_q^{wf}$, we have that T' is well-formed, and hence we can consider the sibling tree $T_s = \pi_s(T')$ induced by T'. Considering that by Transitions (3), there is node x that is selected by $\mathbf{A}_q$ from T', and that $T' = \pi_b(T_s)$, by Theorem 1, we have that q selects x from T_s, and hence is satisfiable.

"$\Leftarrow$" If q is satisfiable, then there exists a sibling tree T_s and a node x in T_s that is selected by q. By Theorem 1, $\mathbf{A}_q$ selects x from $\pi_b(T_s)$, and being $\pi_b(T_s)$ well-formed by construction, $\mathbf{A}_q^{wf}$ accepts $\pi_b(T_s)$. $\square$

From the above result, we obtain a characterization of the computational complexity for both query satisfiability and query containment.

Theorem 11 *Checking satisfiability of a μXPath query is* ExpTime*-complete.*

Proof. For the upper bound, by Theorem 10, checking satisfiability of a *μXPath* query q can be reduced to checking nonemptiness of the 2WATA $\mathbf{A}_q^{wf}$. $\mathbf{A}_q^{wf}$ has just three states more than $\mathbf{A}_q$, which in turn, by Theorem 1 has a number of states that is linear in the size of q and an alphabet whose size is exponential in the size of the alphabet of q. Finally, by Theorem 8 checking nonemptiness of $\mathbf{A}_q^{wf}$ can be done in time exponential in its number of states and linear in the size of its alphabet, from which the claim follows.

For the hardness, it suffices to observe that satisfiability of *RXPath* queries, which can be encoded in linear time into *μXPath* (see Section 2), is already ExpTime-hard [3]. $\square$

Theorem 12 *Checking containment between two μXPath queries is* ExpTime*-complete.*

Proof. To check query containment $(X_1 : \mathcal{F}_1) \subseteq (X_2 : \mathcal{F}_2)$, it suffices to check satisfiability of the *μXPath* query $X_0 : \mathcal{F}_1 \cup \mathcal{F}_2 \cup \{\mathsf{lfp}\{X_0 = X_1 \wedge \neg X_2\}\}$, where without loss of generality we have assumed that the variables defined in $\mathcal{F}_1$ and $\mathcal{F}_2$ are disjoint and different from X_0. Hence, by Theorem 11, we get the upper bound.

For the lower bound, it suffices to observe that query $(X_1 : \mathcal{F}_1)$ is unsatisfiable if and only if it is contained in the query $X_2 : \{\mathsf{lfp}\{X_0 = \mathbf{false}\}\}$. $\square$

5.3 Root Constraints

Following [50], we now introduce *root constraints*, which in our case are *μXPath* formulas intended to be true on the root of the document, and study the problem of reasoning in the presence of such constraints. Formally, the root constraint φ is *satisfied* in a sibling tree T_s if $\varepsilon \in \varphi^{T_s}$. A (finite) set Γ of root constraints is *satisfiable* if there exists a sibling tree T_s that satisfies all constraints in Γ. A set Γ of root constraints *logically implies* a root constraint φ, written $\Gamma \models \varphi$, if φ is satisfied in every sibling tree that satisfies all constraints in Γ.

Root constraints are indeed a quite powerful mechanism to describe structural properties of documents. For example, as shown in [51], *RXPath* (and hence *μXPath*) formulas allow one to express all first-order definable sets of nodes, and this allows for quite sophisticated conditions as root constraints. In fact, *μXPath* differently from *RXPath* [3], can express arbitrary MSO root constraints (see Section 6.2).

Also they allow for capturing XML *DTDs*[3] by encoding the right-hand side of DTD element definitions in a suitable path along the right axis. We illustrate the latter on a simple example (cf. also [15, 50] for a similar encoding).

Consider the following DTD element type definition (using grammar-like notation, with "," for concatenation and "|" for union), where A is the element type being defined, and C, D, E are element types:

$$A \;\longrightarrow\; B, (C^*|D), E$$

The constraint on the sequence of children of an A-node that is imposed on an XML document by such an element type definition, can be directly expressed through the following *RXPath* constraint:

$$[\mathbf{u}](A \to \langle \mathsf{fchild}; B?; ((\mathsf{right}; C?)^* \cup (\mathsf{right}; D?)); \mathsf{right}; E?\rangle[\mathsf{right}]\mathsf{false})$$

where $\mathbf{u}$ is an abbreviation for the path expression $(\mathsf{fchild} \cup \mathsf{right})^*$, and we have assumed to have one atomic proposition for each element type, and that such proposition are pairwise disjoint (in turn enforced through a suitable *RXPath* constraint). Similarly, by means of (*RXPath*) root constraints, one can express also *Specialized DTDs* [58] and the structural part of XML Schema Definitions[4] (cf. [50]).

XPath includes identifiers, which are special propositions that hold in a single node of the sibling tree. It is easy to see that the following root constraint N_A forces a proposition A to be an identifier:

$$\begin{array}{rcll} N_A & = & \langle \mathbf{u}\rangle A \wedge & (1)\\ & & [\mathbf{u}](((\langle \mathsf{fchild}; \mathbf{u}\rangle A \to [\mathsf{right}; \mathbf{u}]\neg A) \wedge & (2)\\ & & \quad ((\langle \mathsf{right}; \mathbf{u}\rangle A \to [\mathsf{fchild}; \mathbf{u}]\neg A) \wedge & (3)\\ & & \quad (A \to [(\mathsf{fchild} \cup \mathsf{right}); \mathbf{u}]\neg A)) & (4) \end{array}$$

In the above constraint, Line 1 expresses that there exists a node of the tree where A holds. Line 2 expresses that, if a node where A holds exists in the fchild subtree of a node n, then A never holds in the right subtree of n. Line 3 is analogous to Line 2, with fchild and right swapped. Finally, Line 4 expresses that, if A holds in a node n, then it holds neither in the fchild nor in the right subtree of n.

It is immediate to see that every set $\{X_1 : \mathcal{F}_1, \ldots X_k : \mathcal{F}_k\}$ of *μXPath* root constraint can be expressed as a *μXPath* query that selects only the root of the tree:

$$X_r : \{\mathsf{lfp}\{X_r \doteq ([\mathsf{fchild}^-]\mathsf{false}) \wedge ([\mathsf{right}^-]\mathsf{false}) \wedge X_1 \wedge \cdots \wedge X_k\}\} \cup \mathcal{F}_1 \cup \cdots \cup \mathcal{F}_k.$$

As a consequence, checking for satisfiability and logical implication of root constraints can be directly reduced to satisfiability and containment of *μXPath* queries. Considering that satisfiability and logical implication is already ExpTime-hard for *RXPath* root constraints [50], we get the following result.

Theorem 13 *Satisfiability and logical implication of μXPath root constraints are* ExpTime*-complete.*

We can also consider query satisfiability and query containment under root constraints, i.e., with respect to all sibling trees that satisfy the constraints. Indeed, a *μXPath* query $X_q : \mathcal{F}_q$ can be expressed as the root constraint:

$$X_r : \{\mathsf{lfp}\{X_r \doteq X_q \vee (\langle \mathsf{fchild}\rangle X_r) \vee (\langle \mathsf{right}\rangle X_r)\}\} \cup \mathcal{F}_q.$$

Hence, we immediately get the following result.

---
[3] `http://www.w3.org/TR/REC-xml/`
[4] `http://www.w3.org/TR/xmlschema-1`

Theorem 14 *Satisfiability and containment of $\mu XPath$ queries under $\mu XPath$ root constraints are* ExpTime*-complete.*

Proof. The upper bound follows from Theorem 13. The lower bound follows from Theorems 11 and 12, by considering an empty set of constraints. $\square$

5.4 View-based Query Processing

View-based query processing is another form of reasoning that has recently drawn a great deal of attention in the database community [38, 39]. In several contexts, such as data integration, query optimization, query answering with incomplete information, and data warehousing, the problem arises of processing queries posed over the schema of a virtual database, based on a set of materialized views, rather than on the raw data in the database [1, 45, 70]. For example, an information integration system exports a global virtual schema over which user queries are posed, and such queries are answered based on the data stored in a collection of data sources, whose content in turn is described in terms of views over the global schema. In such a setting, each data source corresponds to a materialized view, and the global schema exported to the user corresponds to the schema of the virtual database. Notice that typically, in data integration, the data in the sources are correct (i.e., sound) but incomplete with respect to their specification in terms of the global schema. This is due the fact that typically the global schema is not designed taking the sources into account, but rather the information needs of users. Hence it may not be possible to precisely describe the information content of the sources. In this paper we will concentrate on this case (*sound views*), cf. [45].

Consider now a sibling tree that is accessible only through a collection of views expressed as $\mu XPath$ queries, and suppose we need to answer a further $\mu XPath$ query over the tree only on the basis of our knowledge on the views. Specifically, the collection of views is represented by a finite set $\mathcal{V}$ of *view symbols*, each denoting a set of tree nodes. Each view symbol $V \in \mathcal{V}$ has an associated *view definition* q_V and a *view extension* $\mathcal{E}_V$. The view definition q_V is simply a $\mu XPath$ query. The view extension $\mathcal{E}_V$ is a set of *node references*, where each node reference is either an identifier, or an explicit path expression that is formed only by chaining fchild and right and that identifies the node by specifying how to reach it from the root. Observe that a node reference a is interpreted in a sibling tree T_s as a singleton set of nodes a^{T_s}. We use $(\mathcal{E}_V)^{T_s}$ to denote the set of nodes resulting from interpreting the node references in T_s. We say that a *sibling tree T_s satisfies a view V* if $(\mathcal{E}_V)^{T_s} \subseteq (q_V)^{T_s}$. In other words, in T_s all the nodes denoted by $(\mathcal{E}_V)^{T_s}$ must appear in $(q_V)^{T_s}$, but $(q_V)^{T_s}$ may contain nodes not in $(\mathcal{E}_V)^{T_s}$.

Given a set $\mathcal{V}$ of views, and a $\mu XPath$ query q, the set of *certain answers* to q with respect to $\mathcal{V}$ under root constraints Γ is the set $cert_{q,\mathcal{V},\Gamma}$ of node references a such that $a^{T_s} \in q^{T_s}$, for every sibling tree T_s satisfying each $V \in \mathcal{V}$ and each constraint in Γ. *View-based query answering* under root constraints consists in deciding whether a given node reference is a certain answer to q with respect to $\mathcal{V}$.

View-based query answering can also be reduced to satisfiability of root constraints. Given a view V, with extension $\mathcal{E}_V$ and definition $X_V : \mathcal{F}_V$, for each $a \in \mathcal{E}_V$:

- if a is an identifier, then we introduce the root constraint

$$X_a : \{\mathsf{lfp}\{X_a \doteq a \wedge X_V \vee (\langle \mathsf{fchild} \rangle X_a) \vee (\langle \mathsf{right} \rangle X_a)\}\} \cup \mathcal{F}_V.$$

- if a is an explicit path expressions $P_1; \cdots ; P_n$, then we introduce the root constraint

$$X_a : \{\mathsf{lfp}\{X_a \doteq \langle P_1 \rangle \cdots \langle P_n \rangle X_V\}\} \cup \mathcal{F}_V.$$

Let $\Gamma_\mathcal{V}$ be the set of $\mu XPath$ root constraints corresponding to the set of $\mu XPath$ views $\mathcal{V}$, $q = X_q : \mathcal{F}_q$ a $\mu XPath$ query, and Γ a finite set of root constraints. Then a node reference c belongs to $cert_{q,\mathcal{V},\Gamma}$ if and only if the following set of root constraints is unsatisfiable:

$$\Gamma \cup \Gamma_\mathcal{V} \cup \Gamma_{id} \cup \Gamma_{\neg q},$$

where Γ_{id} consists of one root constraint N_a imposing that a behaves as an identifier (see above), for each node a appearing in $\mathcal{V}$, and:

- if c is and identifier, then $\Gamma_{\neg q}$ is

$$X_c : \{\mathsf{lfp}\{X_c \doteq c \wedge \neg X_q \vee (\langle \mathsf{fchild} \rangle X_c) \vee (\langle \mathsf{right} \rangle X_c)\}\} \cup \mathcal{F}_q.$$

- if c is an explicit path expressions $P_1; \cdots ; P_n$, then $\Gamma_{\neg q}$ is

$$X_c : \{\mathsf{lfp}\{X_c \doteq \langle P_1\rangle \cdots \langle P_n\rangle \neg X_q\}\} \cup \mathcal{F}_q.$$

Hence we have linearly reduced view-based query answering under root constraints to unsatisfiability of *μXPath* root constraints, and the following result immediately follows.

Theorem 15 *View-based query answering under root constraints in μXPath is* ExpTime*-complete.*

We conclude this section by observing that reasoning over *RXPath* formulas can be reduced to checking satisfiability in *Propositional Dynamic Logics (PDLs)*, as shown in [50]. Specifically, one can resort to *Repeat-Converse-Deterministic PDL (repeat-CDPDL)*, a variant of PDL that allows for expressing the finiteness of trees and for which satisfiability is ExpTime-complete [71]. This upper bound, however, is established using sophisticated infinite-tree automata-theoretic techniques, which, we just point out have resisted practically efficient implementations. The main advantage of our approach here is that we use only automata on finite trees, which require a much "lighter" automata-theoretic machinery. Indeed, symbolic-reasoning-techniques, including BDDs and Boolean saisfiability solving have been used successfully for XML reasoning [31]. We leave further exploration of this aspect to future work.

6 Relationship among *μXPath*, 2WATAs, and MSO

In this section, we show that *μXPath* is expressively equivalent to Monadic Second-Order Logic (MSO). We have already shown that *μXPath* is equivalent to 2WATAs, hence it suffices to establish the relationship between 2WATAs and MSO. To do so we make use of nondeterministic node-selecting tree automata, which were introduced in [30], following earlier work on deterministic node-selecting tree automata in [55]. (For earlier work on MSO and Datalog, see [33, 34].) For technical convenience, we use here top-down, rather than bottom-up automata. It is also convenient here to assume that the top-down tree automata run on *full* binary trees, even though our binary trees are not full. Thus, we can assume that there is a special label $\perp$ such that a node that should not be present in the tree (e.g., left child of a node that does not contain *hfc* in its label) is labeled by $\perp$.

A *nondeterministic node-selecting top-down tree automaton* (NSTA) on binary trees is a tuple $\mathbf{A} = (\mathcal{L}, S, S_0, \delta, F, \sigma)$, where $\mathcal{L}$ is the alphabet of tree labels, S is a finite set of states, $S_0 \subseteq S$ is the initial state set, $\delta : S \times \mathcal{L} \to 2^{S^2}$ is the transition function, $F \subseteq S$ is a set of accepting states, and $\sigma \subseteq S$ is a set of selecting states. Given a tree $T = (\Delta^T, \ell^T)$, an *accepting run* of $\mathbf{A}$ on T is an S-labeled tree $R = (\Delta^T, \ell^R)$, with the same node set as T, where:

- $\ell^R(\varepsilon) \in S_0$.
- If $x \in \Delta^T$ is an interior node, then $\langle \ell^R(x \cdot 1), \ell^R(x \cdot 2)\rangle \in \delta(\ell^R(x), \ell^T(x))$.
- If $x \in \Delta^T$ is a leaf, then $\delta(\ell^R(x), \ell^T(x)) \cap F^2 \neq \emptyset$.

A node $x \in \Delta^T$ is *selected* by $\mathbf{A}$ from T if there is a run $R = (\Delta^T, \ell^R)$ of $\mathbf{A}$ on T such that $\ell^R(x) \in \sigma$. The notion of accepting run used here is standard, cf. [22]. It is the addition of selecting states that turns these automata from a model of tree recognition to a model of tree querying.

Theorem 16 [30] *(i) For each MSO query $\varphi(x)$, there is an NSTA $\mathbf{A}_\varphi$ such that a node x in a tree $T = (\Delta^T, \ell^T)$ satisfies $\varphi(x)$ iff x is selected from T by $\mathbf{A}_\varphi$. (ii) For each NSTA $\mathbf{A}$, there is an MSO query $\varphi_{\mathbf{A}}$ such that a node x in a tree $T = (\Delta^T, \ell^T)$ satisfies $\varphi_{\mathbf{A}}(x)$ iff x is selected by $\mathbf{A}$ from T.*

We now establish back and forth translations between 2WATAs and NSTAs, implying the equivalence of 2WATAs and MSO.

6.1 From 2WATAs to NSTAs

Theorem 17 *For each 2WATA $\mathbf{A}$, there is an NSTA $\mathbf{A}'$ such that a node x in a binary tree T is selected by $\mathbf{A}$ if and only if it is selected by $\mathbf{A}'$.*

Proof. In Section 4.2, we described a translation of 2WATAs to NTAs. Both the 2WATA and the NTA start their runs there from the root ε of the tree. Here we need the 2WATA to start its run from a node $x_0 \in \Delta^T$, on one hand, and we want the NSTA to select this node x_0. Note, however, that the fact that the 2WATA starts its run from ε played a very small role in the construction in Section 4.2.

Namely, we defined the set of initial states as: $Q_0 = \{(r, R) \mid s_0 \in \mathit{state}(r)$ and there is no transition $(s, -1, s') \in r\}$. The requirement that $s_0 \in \mathit{state}(r)$ corresponds to the 2WATA starting its run in ε.

More generally, however, we can say that the strategy τ is *anchored at a node* $x_0 \in \Delta^T$ if we have $s_0 \in \mathit{state}(\tau(x_0))$. In particular, the strategies studied in Section 4.2 are anchored at ε.

We can now relax the claims in Section 4.2:

Claim 1 [71]

1. *A node x_0 of T is selected by the 2WATA $\mathbf{A}$ iff $\mathbf{A}$ has an accepting strategy for T that is anchored at x_0.*

2. *A node x_0 of T is selected by the 2WATA $\mathbf{A}$ iff $\mathbf{A}$ has a strategy for T that is anchored at x_0 and an accepting annotation η of τ.*

To match this relaxation in the construction of the NSTA, we need to redefine the set of initial states as $Q_0 = \{(r, R) \mid$ there is no transition $(s, -1, s') \in r\}$, which means that the requirement that the 2WATA starts its run from the root is dropped, as the strategy guessed by the NSTA no longer needs to be anchored at ε. Instead, we want the strategy to be anchored at the node x_0 selected by $\mathbf{A}$. To that end, we define the set of selecting states as $\sigma = \{(r, R) \in \mathcal{T} \times \mathcal{E} \mid s_0 \in \mathit{state}(r)\}$. That is, if $\mathbf{A}$ starts its run at x_0, then the strategy needs to be anchored at x_0 and x_0 is selected by the NSTA.

Finally, while the NTA constructed in Section 4 accepts when the transition function yields the truth value **true**, the NSTA accepts by means of accepting states. We can simply add a special accepting state *accept* and transition to it whenever the transition function yields **true**. $\square$

We remark that the translation from 2WATA to NSTA is exponential. Together with the results in the previous sections, we get an exponential translation from *μXPath* to NSTAs. This explains why NSTAs are not useful for efficient query-evaluation algorithms, as noted in [64].

6.2 From NSTAs to 2WATAs

For the translation from NSTAs to 2WATAs, the idea is to take an accepting run of an NSTA, which starts from the root of the tree, and convert it to a run of a 2WATA, which starts from a selected node. The technique is related to the translation from tree automata to Datalog in [34]. The construction here uses the propositions *ifc*, *irs*, *hfc*, and *hrs* introduced earlier.

Theorem 18 *For each NSTA $\mathbf{A}$, there is a 2WATA $\mathbf{A}'$ such that a node x_0 in a tree T is selected by $\mathbf{A}$ if and only if it is selected by $\mathbf{A}'$.*

Proof. Let $\mathbf{A} = (\mathcal{L}, S, S_0, \delta, F, \sigma)$ be an NSTA. We construct an equivalent 2WATA $\mathbf{A}' = (\mathcal{L}, S', s'_0, \delta', \alpha')$ as follows (for $s \in S$ and $a \in \mathcal{L}$):

- $S' = \{s_0\} \cup S \times \{u, d, l, r\} \cup \Sigma$. (We add a new initial state, and we keep four copies, tagged with u, d, l, or r of each state in S. We also add the alphabet to the set of states.)

- $\alpha' = \emptyset$. (Infinite branches are not allowed in runs of $\mathbf{A}'$.)

- $\delta'(s_0, a) = \bigvee_{s \in \sigma}(((s, d), 0) \wedge ((s, u), 0))$. ($\mathbf{A}'$ guesses a selecting state of $\mathbf{A}$ and spawns two copies, tagged with d and u, respectively to go downwards and upwards.)

- If a does not contain *hfc* and does not contain *hrs* (that is, we are reading a leaf node), then $\delta'((s, d), a) = \mathbf{true}$ if $\delta(s, a) \cap F^2 \neq \emptyset$, and $\delta'((s, d), a) = \mathbf{false}$ if $\delta(s, a) \cap F^2 = \emptyset$. (In a leaf node, a transition from (s, d) either accepts or rejects, just like $\mathbf{A}$ from s.)

- If a contains *hfc* or *hrs* (that is, we are reading an interior node), then $\delta'((s, d), a) = \bigvee_{(t_1, t_2) \in \delta(s, a)}((t_1, d), 1) \wedge ((t_2, d), 2)$. (States tagged with d behave just like the corresponding states of $\mathbf{A}$.)

- If a contains neither *ifc* nor *irs* (that is, we are reading the root node), then $\delta'((s, u), a) = \mathbf{true}$ if $s \in S_0$, and $\delta'((s, u), a) = \mathbf{false}$, otherwise (that is, if an upword state reached the root, then we just need to check that the root has been reached with an initial state),

- If a contains *ifc* (it is a left child), then $\delta'((s, u), a) = \bigvee_{t \in S, a' \in \mathcal{L}, (s, t') \in \delta(t, a')}((t, u), -1) \wedge (a', -1) \wedge ((t', r), -1)$. (Guess a state and letter in the node above, and proceed to check them.)

- If a contains *irs* (it is a right child), then $\delta'((s,u),a) = \bigvee_{t\in S, a'\in\mathcal{L}, (t',s)\in\delta(t,a')}((t,u),-1)\wedge(a',-1)\wedge((t',l),-1)$. (Guess a state and letter in the node above, and proceed to check them.)
- $\delta'(a',a) = \textbf{true}$ if $a'=a$ and $\delta'(a',a)=\textbf{false}$ if $a'\neq a$. (Check that the guessed letter was correct.)
- $\delta'((s,l),a) = ((s,d),1)$. (Check left subtree.)
- $\delta'((s,r),a) = ((s,d),2)$. (Check right subtree.) □

Intuitively, $\mathbf{A}'$ tries to guess an accepting run of $\mathbf{A}$ that selects x_0. $\mathbf{A}'$ starts at x_0 in a selecting state of $\mathbf{A}$, guesses the subrun below x_0, and also goes up the tree to guess the rest of the run. Note that we need not worry about cycles in the run of $\mathbf{A}'$, as it only goes upward in the u mode, and once it leaves the u mode it never enters again the u mode.

We need to show that that a node x_0 in a tree T is selected by $\mathbf{A}$ if and only if it is selected by $\mathbf{A}'$. If a node x_0 of T is selected by $\mathbf{A}$, then $\mathbf{A}$ has an accepting run on T that reaches x_0 in some selecting state $s_a\in\sigma$. Then $\mathbf{A}'$ starts its run at x_0 in state s_a, and it proceeds to emulate precisely the accepting run of $\mathbf{A}$. More precisely, at x_0 $\mathbf{A}'$ branches conjunctively to both (s_a,d) and (s_a,u). From (s_a,d), $\mathbf{A}'$ continues downwards and emulate the run of $\mathbf{A}$. That is, if $\mathbf{A}$ reaches a node x below x_0 in state s, then $\mathbf{A}'$ reaches x in state (s,d). At the leaves, $\mathbf{A}$ transitions to accepting states, and $\mathbf{A}'$ transitions to **true**. From (s_a,u), $\mathbf{A}'$ first continues upwards. Let x be a node above x_0 such that (1) x_0 is at or below the left child of x, (2) x is labeled by the letter a, and (3) x is reached by $\mathbf{A}$ in state t, and $\mathbf{A}$ reaches the right child of x in state t'. Then $\mathbf{A}'$ reaches x with states (t,u), a, and (t',r). Then $\mathbf{A}'$ continues the upward emulation from (t,u), verifies that a is the letter at x, and also transitions to the right child in state (t',d), from which it continues with the downward emulation of $\mathbf{A}$. The upward emulation eventually reaches the root in an initial state.

On the other hand, for $\mathbf{A}'$ to select x_0 in T it must start at x_0 in some state $s_a\in\sigma$. While $\mathbf{A}'$ is a 2WATA and its run is a run tree, this run tree has a very specific structure. From a state (s,d), $\mathbf{A}'$ behaves just like an NTA. From a state (s,u), $\mathbf{A}'$ proceeds upward, trying to label every node on the path to the root with a single state, such that the root is labeled by an initial state, and from every node on that path $\mathbf{A}'$ can then go downward, again labeling each node by a single state. Thus, $\mathbf{A}'$ essentially guesses an accepting run tree of $\mathbf{A}$ that selects x_0. □

While the translation from 2WATAs to NSTAs was exponential, the translation from NSTAs to 2WATAs is linear. It follows from the proof of Theorem 18 that the automaton $\mathbf{A}'$ correspond to lfp-μ*XPath*, which consists of μ*XPath* queries with a single, least fixpoint block. This clarifies the relationship between μ*XPath* and Datalog-based languages studied in [34, 30]. In essence, μ*XPath* corresponds to stratified monadic Datalog, where rather than use explicit negation, we use alternation of least and greatest fixpoints, while lfp-μ*XPath* corresponds to monadic Datalog. The results of the last two sections provide an exponential translation from μ*XPath* to lfp-μ*XPath*. Note, however, that lfp-μ*XPath* does not have a computational advantage over μ*XPath*, for either query evaluation or query containment. In contrast, while stratified Datalog queries can be evaluated in polynomial time (in terms of data complexity), there is no good theory for containment of stratified monadic Datalog queries.

The above results provide us a characterization of the expressive power of μ*XPath*.

Theorem 19 *Over (binary) sibling trees, μXPath and MSO have the same expressive power.*

Proof. By Theorems 1 and 2, μ*XPath* is equivalent to WATAs, and by Theorems 16, 17, and 18, 2WATAs are equivalent to MSO. □

7 Conclusion

The results of this paper fill a gap in the theory of node-selection queries for trees. With a natural extension of *XPath* by fixpoint operators, we obtained μ*XPath*, which is expressively equivalent to MSO, has linear-time query evaluation and exponential-time query containment, as *RXPath*. 2WATAs, the automata-theoretic counterpart of μ*XPath*, fills another gap in the theory by providing an automaton model that can be used for both query evaluation and containment testing. Unlike much of the theory of automata on infinite trees, which so far has resisted implementation, the automata-theoretic machinery over finite trees should be much more amenable to practical implementations

Our automata-theoretic approach is based on techniques developed in the context of program logics [44, 71]. Here, however, we leverage the fact that we are dealing with finite trees, rather than the infinite trees used in the program-logics context. Indeed, the automata-theoretic techniques used in reasoning about infinite trees are notoriously difficult [62, 66] and have resisted efficient implementation. The restriction to finite trees here enables us to obtain a much more feasible algorithmic approach. In particular, as pointed out in [17], one can make use of symbolic techniques, at the base of modern model checking tools, for effectively querying and verifying XML documents. It is worth noting that while our automata run over finite trees they are allowed to have infinite runs. This separates 2WATAs from the alternating tree automata used in [23, 65]. The key technical results here are that acceptance of trees by 2WATAs can be decided in linear time, while nonemptiness of 2WATAs can be decided in exponential time.

References

[1] Serge Abiteboul and Oliver Duschka. Complexity of answering queries using materialized views. In *Proc. of the 17th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*, pages 254–265, 1998.

[2] L. Afanasiev, T. Grust, M. Marx, J. Rittinger, and J. Teubner. An inflationary fixed point operator in XQuery. In *Proc. of the 24th IEEE Int. Conf. on Data Engineering (ICDE)*, pages 1504–1506, 2008.

[3] Loredana Afanasiev, Patrick Blackburn, Ioanna Dimitriou, Bertrand Gaiffe, Evan Goris, Maarten Marx, and Maarten de Rijke. PDL for ordered trees. *J. of Applied Non-Classical Logics*, 15(2):115–135, 2005.

[4] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook: Theory, Implementation and Applications.* Cambridge University Press, 2003.

[5] Pablo Barceló Baeza. Querying graph databases. In *Proc. of the 32nd ACM SIGACT SIGMOD SIGAI Symp. on Principles of Database Systems (PODS)*, pages 175–188, 2013.

[6] Henrik Björklund, Wim Martens, and Thomas Schwentick. Conjunctive query containment over trees. *J. of Computer and System Sciences*, 77(3):450–472, 2011.

[7] Mikolaj Bojanczyk, Claire David, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data words. *ACM Trans. on Computational Logic*, 12(4):27, 2011.

[8] Mikolaj Bojanczyk, Anca Muscholl, Thomas Schwentick, and Luc Segoufin. Two-variable logic on data trees and XML reasoning. *J. of the ACM*, 56(3):13:1–13:48, 2009.

[9] Mikolaj Bojanczyk and Pawel Parys. XPath evaluation in linear time. In *Proc. of the 27th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*, pages 241–250, 2008.

[10] Piero Bonatti, Carsten Lutz, Aniello Murano, and Moshe Y. Vardi. The complexity of enriched μ-calculi. *Logical Methods in Computer Science*, 4(3:11):1–27, 2008.

[11] Patricia Bouyer. A logical characterization of data languages. *Information Processing Lett.*, 84(2):75–85, 2002.

[12] Randal E. Bryant. Graph-based algorithms for Boolean-function manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, 1986.

[13] Peter Buneman, Susan Davidson, Gerd Hillebrand, and Dan Suciu. A query language and optimization technique for unstructured data. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, pages 505–516, 1996.

[14] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[15] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Representing and reasoning on XML documents: A description logic approach. *J. of Logic and Computation*, 9(3):295–318, 1999.

[16] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. View-based query answering and query containment over semistructured data. In Giorgio Ghelli and Gösta Grahne, editors, *Revised Papers of the 8th International Workshop on Database Programming Languages (DBPL 2001)*, volume 2397 of *Lecture Notes in Computer Science*, pages 40–61. Springer, 2002.

[17] Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, and Moshe Y. Vardi. An automata-theoretic approach to Regular XPath. In *Proc. of the 12th Int. Symp. on Database Programming Languages (DBPL)*, volume 5708 of *Lecture Notes in Computer Science*, pages 18–35. Springer, 2009.

[18] Diego Calvanese, Giuseppe De Giacomo, and Moshe Y. Vardi. Node selection query languages for trees. In *Proc. of the 24th AAAI Conf. on Artificial Intelligence (AAAI)*, pages 279–284, 2010.

[19] S. Chaudhuri, S. Krishnamurthy, S. Potarnianos, and K. Shim. Optimizing queries with materialized views. In *Proc. of the 11th IEEE Int. Conf. on Data Engineering (ICDE)*, pages 190–200, 1995.

[20] James Clark and Steve DeRose. XML path language (XPath) version 1.0. W3C Recommendation, World Wide Web Consortium, November 1999. Available at `http://www.w3.org/TR/1999/REC-xpath-19991116`.

[21] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Loöding, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available at `http://www.grappa.univ-lille3.fr/tata/`, 2008.

[22] Hubert Comon, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Sophie Tison, and Marc Tommasi. Tree automata techniques and applications. Available at `http://www.grappa.univ-lille3.fr/tata/`, 2002.

[23] Stavros S. Cosmadakis, Haim Gaifman, Paris C. Kanellakis, and Moshe Y. Vardi. Decidable optimization problems for database logic programs. In *Proc. of the 20th ACM SIGACT Symp. on Theory of Computing (STOC)*, pages 477–490, 1988.

[24] Giuseppe De Giacomo and Maurizio Lenzerini. Concept language with number restrictions and fixpoints, and its relationship with μ-calculus. In *Proc. of the 11th Eur. Conf. on Artificial Intelligence (ECAI)*, pages 411–415, 1994.

[25] Stéphane Demri and Ranko Lazić. LTL with the Freeze quantifier and register automata. *ACM Trans. on Computational Logic*, 10(3):1–30, 2009.

[26] John E. Doner. Decidability of the weak second-order theory of two successors. *Notices Amer. Math. Soc.*, 12:819, 1965.

[27] W. F. Dowling and J. H. Gallier. Linear-time algorithms for testing the satisfiability of propositional horn formulae. *J. of Logic Programming*, 1(3):267–284, 1984.

[28] E. A. Emerson and C.-L. Lei. Efficient model checking in fragments of the mu-calculus. In *Proc. of the 1st IEEE Symp. on Logic in Computer Science (LICS)*, pages 267–278, 1986.

[29] Michael J. Fischer and Richard E. Ladner. Propositional dynamic logic of regular programs. *J. of Computer and System Sciences*, 18:194–211, 1979.

[30] Markus Frick, Martin Grohe, and Christoph Koch. Query evaluation on compressed trees (extended abstract). In *Proc. of the 18th IEEE Symp. on Logic in Computer Science (LICS)*, pages 188–197, 2003.

[31] Pierre Genevès and Nabil Layaïda. XML reasoning made practical. In *Proc. of the 26th IEEE Int. Conf. on Data Engineering (ICDE)*, pages 1169–1172, 2010.

[32] Pierre Genevès, Nabil Layaïda, and Alan Schmitt. Efficient static analysis of XML paths and types. In *Proc. of the ACM SIGPLAN 2007 Conf. on Programming Language Design and Implementation (PLDI 2007)*, pages 342–351, 2007.

[33] Georg Gottlob and Christoph Koch. Monadic Datalog and the expressive power of languages for web information extraction. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*, pages 17–28, 2002.

[34] Georg Gottlob and Christoph Koch. Monadic Datalog and the expressive power of languages for web information extraction. *J. of the ACM*, 51(1):74–113, 2004.

[35] Georg Gottlob, Christoph Koch, and Reihard Pichler. Efficient algorithms for processing XPath queries. *ACM Trans. on Database Systems*, 30(2):444–491, 2005.

[36] Erich Grädel, Wolfgang Thomas, and Thomas Wilke, editors. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, 2002. Outcome of a Dagstuhl seminar in February 2001.

[37] Ashish Gupta and Jeffrey D. Ullman. Generalizing conjunctive query containment for view maintenance and integrity constraint verification (abstract). In *Workshop on Deductive Databases (In conjunction with JICSLP)*, page 195, Washington D.C. (USA), 1992.

[38] Alon Y. Halevy. Theory of answering queries using views. *SIGMOD Record*, 29(4):40–47, 2000.

[39] Alon Y. Halevy. Answering queries using views: A survey. *Very Large Database J.*, 10(4):270–294, 2001.

[40] M. Jurdzinski. Small progress measures for solving parity games. In *Proc. of the 17th Symp. on Theoretical Aspects of Computer Science (STACS)*, volume 1770 of *Lecture Notes in Computer Science*, pages 290–301. Springer, 2000.

[41] Michael Kaminski and Tony Tan. Tree automata over infinite alphabets. In *Pillars of Computer Science, Essays Dedicated to Boris (Boaz) Trakhtenbrot on the Occasion of His 85th Birthday*, volume 4800 of *Lecture Notes in Computer Science*, pages 386–423. Springer, 2008.

[42] Dexter Kozen. Results on the propositional μ-calculus. *Theoretical Computer Science*, 27:333–354, 1983.

[43] Orna Kupferman, Ulrike Sattler, and Moshe Y. Vardi. The complexity of the graded mu-calculus. In *Proc. of the 18th Int. Conf. on Automated Deduction (CADE)*, 2002.

[44] Orna Kupferman, Moshe Y. Vardi, and Pierre Wolper. An automata-theoretic approach to branching-time model checking. *J. of the ACM*, 47(2):312–360, 2000.

[45] Maurizio Lenzerini. Data integration: A theoretical perspective. In *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*, pages 233–246, 2002.

[46] Alon Y. Levy and Marie-Christine Rousset. Verification of knowledge bases: a unifying logical view. In *Proc. of the 4th European Symposium on the Validation and Verification of Knowledge Based Systems*, Leuven, Belgium, 1997.

[47] Alon Y. Levy and Yehoshua Sagiv. Semantic query optimization in Datalog programs. In *Proc. of the 14th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*, pages 163–173, 1995.

[48] Leonid Libkin. Logics for unranked trees: An overview. *Logical Methods in Computer Science*, 2(3), 2006.

[49] Leonid Libkin and Cristina Sirangelo. Reasoning about XML with temporal logics and automata. In *Proc. of the 15th Int. Conf. on Logic for Programming, Artificial Intelligence, and Reasoning (LPAR)*, pages 97–112, 2008.

[50] Maarten Marx. XPath with conditional axis relations. In *Proc. of the 9th Int. Conf. on Extending Database Technology (EDBT)*, volume 2992 of *Lecture Notes in Computer Science*, pages 477–494. Springer, 2004.

[51] Maarten Marx. First order paths in ordered trees. In *Proc. of the 10th Int. Conf. on Database Theory (ICDT)*, volume 3363 of *Lecture Notes in Computer Science*, pages 114–128. Springer, 2005.

[52] Tova Milo and Dan Suciu. Index structures for path expressions. In *Proc. of the 7th Int. Conf. on Database Theory (ICDT)*, volume 1540 of *Lecture Notes in Computer Science*, pages 277–295. Springer, 1999.

[53] Amihai Motro. Panorama: A database system that annotates its answers to queries with their properties. *J. of Intelligent Information Systems*, 7(1), 1996.

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

[55] Frank Neven and Thomas Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1–2):633–674, 2002.

[56] Frank Neven and Thomas Schwentick. XPath containment in the presence of disjunction, DTDs, and variables. In *Proc. of the 9th Int. Conf. on Database Theory (ICDT)*, pages 315–329, 2003.

[57] Frank Neven, Thomas Schwentick, and Victor Vianu. Finite state machines for strings over infinite alphabets. *ACM Trans. on Computational Logic*, 5(3):403–435, 2004.

[58] Yannis Papakonstantinou and Victor Vianu. DTD inference for views of XML data. In *Proc. of the 19th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*, pages 35–46, 2000.

[59] Vaugham R. Pratt. A practical decision method for Propositional Dynamic Logic. In *Proc. of the 10th ACM Symp. on Theory of Computing (STOC)*, pages 326–337, 1978.

[60] Shmuel Safra. On the complexity of ω-automata. In *Proc. of the 29th Annual Symp. on the Foundations of Computer Science (FOCS)*, pages 319–327, 1988.

[61] Klaus Schild. Terminological cycles and the propositional μ-calculus. In *Proc. of the 4th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR)*, pages 509–520, 1994.

[62] C. Schulte Althoff, W. Thomas, and N. Wallmeier. Observations on determinization of Büchi automata. In *Proc. of the 10th Int. Conf. on the Implementation and Application of Automata*, 2005.

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

[64] Thomas Schwentick. Automata for XML – A survey. *J. of Computer and System Sciences*, 73(3):289–315, 2007.

[65] Giora Slutzki. Alternating tree automata. *Theoretical Computer Science*, 41:305–318, 1985.

[66] S. Tasiran, R. Hojati, and R. K. Brayton. Language containment using non-deterministic Omega-automata. In *Proc. of the 8th Advanced Research Working Conf. on Correct Hardware Design and Verification Methods (CHARME)*, volume 987 of *Lecture Notes in Computer Science*, pages 261–277. Springer, 1995.

[67] Balder ten Cate. The expressivity of XPath with transitive closure. In *Proc. of the 25th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*, pages 328–337, 2006.

[68] Balder ten Cate and Carsten Lutz. The complexity of query containment in expressive fragments of XPath 2.0. *J. of the ACM*, 56(6), 2009.

[69] Balder ten Cate and Luc Segoufin. XPath, transitive closure logic, and nested tree walking automata. In *Proc. of the 27th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*, pages 251–260, 2008.

[70] Jeffrey D. Ullman. Information integration using logical views. In *Proc. of the 6th Int. Conf. on Database Theory (ICDT)*, volume 1186 of *Lecture Notes in Computer Science*, pages 19–40. Springer, 1997.

[71] Moshe Y. Vardi. Reasoning about the past with two-way automata. In *Proc. of the 25th Int. Coll. on Automata, Languages and Programming (ICALP)*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer, 1998.

[72] Victor Vianu. A web odyssey: From Codd to XML. In *Proc. of the 20th ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS)*, 2001. Invited talk.