Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence (AAAI-10)

Node Selection Query Languages for Trees

Diego Calvanese KRDB Research Centre Free University of Bozen-Bolzano, Italy calvanese@inf.unibz.it

Giuseppe De Giacomo, Maurizio Lenzerini Dip. di Informatica e Sistemistica SAPIENZA Università di Roma, Italy *lastname*@dis.uniroma1.it

Moshe Y. Vardi Dept. of Computer Science Rice University, Houston, U.S.A. vardi@cs.rice.edu

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 expressive as MSO, and at the same time enjoys the computational properties of XPath, which are linear query evaluation and exponential query-containment test. In this paper we propose μ*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 as the definite query language for trees.

Introduction

XML has become the standard language for Web documents supporting semistructured data, and the last few years have witnessed an extensive interest in XML queries. 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. We focus here on queries that select sets of nodes, which we call *node selection queries*. Many of such formalisms come from the tradition of modal and dynamic logics, similarly to the most expressive languages of the Description Logics family (Baader et al. 2003), and therefore include the use of regular path expressions to navigate through XML documents. *XPath* (Clark and DeRose 1999) is a notable example of these formalisms, and, in this sense, it can also be seen as an expressive Description Logic over finite trees.

A main line of research has been on identifying nice computational properties of *XPath* and studying extensions of *XPath* that still enjoy these properties. An important feature of *XPath* is the tractability of query evaluation (in data complexity); queries in the navigational core *CoreXPath* can be evaluated in time that is linear in both the size of the query and the size of the input tree (Gottlob, Koch, and Pichler 2005; Bojanczyk and Parys 2008). This property is enjoyed also by various extensions of *XPath*. Specifically, it is shown in (Calvanese et al. 2009) that *RXPath*, which is the extension of *XPath* with regular expressions, also has this property. Another nice computational property of *XPath* is that containment testing is in EXPTIME(Neven and Schwentick 2003; Schwentick 2004). This property holds also for *RXPath* (Marx 2004; ten Cate and Segoufin 2008; Calvanese et al. 2009) and other extensions of XPath (ten Cate and Lutz 2009).

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 (Marx 2004; 2005). Regular extensions of *XPath* are expressively equivalent to extensions of FO with transitive closure (ten Cate 2006; ten Cate and Segoufin 2008). Another classical logic is monadic second-order logic MSO. This logic is more expressive than FO and its extensions by transitive closure (Libkin 2006; ten Cate 2006; ten Cate and Segoufin 2008). 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 (Gottlob and Koch 2004). Various logics are known to have the same expressive power as MSO, cf. (Libkin 2006), 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 third line of research focuses on the relationship between query languages for finite trees and tree automata (Libkin and Sirangelo 2008; Neven 2002; Schwentick 2007). Various automata models have been proposed. Among the cleanest models is that of node-selecting tree automata, which are standard automata on finite trees, augmented with node selecting states (Neven and Schwentick 2002; Frick, Grohe, and Koch 2003). 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 (Schwentick 2007).

Progress on the automata-theoretic front was recently reported in (Calvanese et al. 2009), where a comprehensive automata-theoretic framework for evaluating and reasoning about *RXPath* was developed. The framework is based on *two-way weak alternating tree automata*, denoted 2WATAs (Kupferman, Vardi, and Wolper 2000), but specialized for finite trees, and enables one to derive both a linear-time algorithm for query evaluation and an exponential-time algo-

Copyright © 2010, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

rithm for testing query containment.

In this paper we show that we can preserve these nice computational properties, and extend the automata-theoretic framework based on 2WATAs to *μXPath*, which is *XPath* enriched with *alternation-free* fixpoint operators. Alternation freedom implies that the least and greatest fixpoint operators interact, and is known to yield computationally amenable logics (Kupferman, Vardi, and Wolper 2000). 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. (Kozen 1983). 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.

Fixpoint operators have been studied in the μ-calculus, interpreted over arbitrary structures (Kozen 1983), 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., (Grädel, Thomas, and Wilke 2002)). Forms of μ-calculus have also been considered in Description Logics (De Giacomo and Lenzerini 1994; Kupferman, Sattler, and Vardi 2002; Bonatti et al. 2008), 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 (Calvanese, De Giacomo, and Lenzerini 1999), where an explicit well-foundedness construct was used to capture XML in description logics.

In a finite-tree setting, extending *XPath* with arbitrary fixpoint operators, has been studied earlier (ten Cate 2006; Libkin 2006; Genevès, Layaïda, and Schmitt 2007) (see also (Afanasiev et al. 2008)), but while 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, being *μXPath* alternation free, it is closely related to a stratified version of monadic Datalog proposed as a query language for finite trees in (Gottlob and Koch 2004), 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 *μXPath* and 2WATA. Specifically, there are linear translations from *μXPath* queries to 2WATA and from 2WATA to *μXPath*. This immediately yields the nice computational properties for *μXPath*. We then prove the equivalence of 2WATA to node-selecting tree automata (NSTA), shown to be expressively equivalent to MSO (Frick, Grohe, and Koch 2003). On the one hand, we have an exponential translation from 2WATA to NSTA. On the other hand, we have a linear translation from NSTA to 2WATA. This yields the expressive equivalence of *μXPath* to MSO.

μXPath

The query language *μXPath* is an extension of *RXPath* that is equipped with explicit fixpoint operators over systems of equations. To define *μXPath*, we start from *RXPath node expressions*, for which we adopt the Propositional Dynamic Logic (PDL) syntax (Fischer and Ladner 1979; Calvanese et al. 2009). An *RXPath* node expression φ is defined by the following syntax:

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

where A denotes an atomic proposition belonging to an alphabet Σ, and child and right denote the two main *XPath* axis relations, and P denotes a *path expression*, formed as a regular expression over the axis relations. We consider the other *XPath* axis relations parent and left as abbreviations for child^- and right^-, respectively. Also, we use the usual abbreviations, including true, false, and $\varphi_1 \rightarrow \varphi_2$.[1]

Using *RXPath* node expressions, we define *μXPath* queries as follows. We consider a set $\mathcal{X}$ of variables, disjoint from Σ. An *equation* has the form $X \doteq \varphi$ where $X \in \mathcal{X}$, and φ is an *RXPath* node expression having as atomic propositions symbols from $\Sigma \cup \mathcal{X}$, with the proviso that each variable $X' \in \mathcal{X}$ may occur only positively in φ (see (Kozen 1983)). 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 interested in 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 the proviso above, see e.g., (Vardi and Wolper 1984). 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 $i \in [1..n]$, a *fixpoint block* has the form $fp\{X_1 \doteq \varphi_1, \ldots, X_n \doteq \varphi_n\}$, where *fp* is either lfp or gfp, denoting respectively the least fixpoint and the greatest fixpoint of the set of equations. We say that the variables $X_1, \ldots, X_n$ are *defined* in the fixpoint block.

A *μXPath (node selection) 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;
- 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 $F_j \preceq F_i$.

We now give some examples. To denote nodes that on all child-paths (possibly of length 0) reach a `red` node:

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

To denote nodes all of whose descendants (including the node itself) are not simultaneously `red` and `blue`:

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

To denote `red` nodes all of whose `red` descendants have only `blue` children and all of whose `blue` descendants have at least a `red` child:

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

Notice that we could replace the least fixpoint with a greatest fixpoint operator, since the equations in that block are not recursive. To denote `red` nodes all of whose `red` descendants

[1] Notice that we do not consider identifiers in the language, i.e., atomic propositions that denote singletons, since in *RXPath* we can explicitly impose that a proposition denotes a singleton.

reach `blue` nodes on all child-paths and all of whose `blue` descendants reach `red` nodes on at least one child-path:

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

To denote those nodes that have a `red` right-sibling and such that all siblings along the path to it have a `blue` descendant:

$$X_0 : \{\mathsf{lfp}\{X_0 \doteq \mathtt{red} \vee \langle\mathsf{right}\rangle X_0 \wedge X_1,\\ X_1 \doteq \mathtt{blue} \vee \langle\mathsf{child}\rangle X_1\}\}$$

Once we introduce fixpoints, node expression of the form $\langle P\rangle\phi$ and $[P]\phi$ with complex P can simply be considered as abbreviations (Kozen 1983). For example $\langle\mathsf{right}^*\rangle A$ can be expressed as $X : \{\mathsf{lfp}\{X \doteq A \vee \langle\mathsf{right}\rangle X\}\}$. Hence, for simplicity, wlog, in the following we restrict path expressions in *μXPath* queries to be atomic or inverse of atomic:

$$P \longrightarrow \mathsf{child} \mid \mathsf{right} \mid P^-$$

Next we turn to the semantics of *μXPath*. Following (Marx 2004; 2005), we formalize XML documents as finite sibling trees. A (finite) tree is a complete prefix-closed non-empty (finite) set of words over $\mathbb{N}$ (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 $x \in \Delta$. 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 $x{\cdot}i{\cdot}{-1} = x$. If the number of successors of the nodes of a tree is a priori unbounded, we say that the tree is *unranked*. On the contrary, *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 $\ell^T : \Delta^T \rightarrow \mathcal{L}$ is a mapping assigning to each node $x \in \Delta^T$ a label $\ell^T(x)$ in $\mathcal{L}$.

A *sibling tree* is a pair $T_s = (\Delta^{T_s}, \cdot^{T_s})$, where Δ^{T_s} is an unranked tree and $\cdot^{T_s}$ is an interpretation function that assigns to each atomic symbol $A \in \Sigma$ a set A^{T_s} of nodes of Δ^{T_s}, and that interprets the axis relations and path expressions in the obvious way, namely:

$$\begin{aligned}\mathsf{child}^{T_s} &= \{(z, z{\cdot}i) \mid z, z{\cdot}i \in \Delta^{T_s}\}\\ \mathsf{right}^{T_s} &= \{(z{\cdot}i, z{\cdot}(i{+}1)) \mid z{\cdot}i, z{\cdot}(i{+}1) \in \Delta^{T_s}\}\\ (P^-)^{T_s} &= \{(z', z) \mid (z, z') \in P^{T_s}\}\end{aligned}$$

Since we have to deal also with variables in equations, in order to give the semantics of *μXPath* we need to introduce variable assignments. A *(variable) assignment* ρ on a tree $T_s = (\Delta^{T_s}, \cdot^{T_s})$ is a mapping that assigns to variables of $\mathcal{X}$ sets of nodes in Δ^{T_s}. Given an assignment ρ, we use the notation $\rho[X_1/\mathcal{E}_1, \ldots, X_n/\mathcal{E}_n]$ to denote the assignment identical to ρ except that it assigns to X_i the value $\mathcal{E}_i$, for $i \in [1..n]$. Given a sibling tree T_s and an assignment ρ, we interpret the fixpoint blocks in a *μXPath* query $X : \mathcal{F}$ by induction on the partial order $\preceq$ of fixpoint blocks in $\mathcal{F}$ for a fixpoint block $F_i \in \mathcal{F}$, as shown in Figure 1 (see also (Vardi and Wolper 1984)). The sets $\{X_1/\mathcal{E}_1^\mu, \ldots, X_n/\mathcal{E}_n^\mu\}$ and $\{X_1/\mathcal{E}_1^\nu, \ldots, X_n/\mathcal{E}_n^\nu\}$ are variable assignments that provide respectively the smallest and the greatest solution of the set of equations $\{X_1 \doteq \varphi_1, \ldots, X_n \doteq \varphi_n\}$. Note that the initial assignment plays no role in the interpretation of fixpoint blocks. The *evaluation* $(X : \mathcal{F})^{T_s}$ of a *μXPath* query $X : \mathcal{F}$

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

where $\{X_1/\mathcal{E}_1^\mu, \ldots, X_n/\mathcal{E}_n^\mu\}$ is the variable assignment for $X_1, \ldots, X_n$ defined as (using component-wise intersection) $\bigcap_{\{X_1/\mathcal{E}_1, \ldots, X_n/\mathcal{E}_n\}} \{\mathcal{E}_1 = (\varphi_1)_{\rho[X_1/\mathcal{E}_1, \ldots, X_n/\mathcal{E}_n]}^{T_s}, \ldots, \mathcal{E}_n = (\varphi_n)_{\rho[X_1/\mathcal{E}_1, \ldots, X_n/\mathcal{E}_n]}^{T_s}\}$, and $\{X_1/\mathcal{E}_1^\nu, \ldots, X_n/\mathcal{E}_n^\nu\}$ is the variable assignment for $X_1, \ldots, X_n$ defined as (using component-wise union) $\bigcup_{\{X_1/\mathcal{E}_1, \ldots, X_n/\mathcal{E}_n\}} \{\mathcal{E}_1 = (\varphi_1)_{\rho[X_1/\mathcal{E}_1, \ldots, X_n/\mathcal{E}_n]}^{T_s}, \ldots, \mathcal{E}_n = (\varphi_n)_{\rho[X_1/\mathcal{E}_1, \ldots, X_n/\mathcal{E}_n]}^{T_s}\}$.

Figure 1: Semantics of *μXPath*

over a sibling tree T_s is $\mathcal{E} \subseteq \Delta^{T_s}$ such that $X/\mathcal{E} \in F^{T_s}$, where $F \in \mathcal{F}$ is the fixpoint block defining X.

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

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

Using fchild, we can thus re-express the child axis as $\mathsf{fchild};\mathsf{right}^*$, see (Calvanese et al. 2009). In the following, we will focus on *μXPath* queries that use only the fchild and right axis relations, and are evaluated over binary (representations of sibling) trees.

Relationship between *μXPath* and 2WATAs

We establish now the relationship between *μXPath* and two-way tree automata. Specifically, we resort to two-way weak alternating automata over finite trees (2WATAs) (Calvanese et al. 2009), which have nice computational properties, and for which we can devise efficient translations from and to *μXPath*. We show how to construct (*i*) from each *μXPath* query φ (over binary 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 *μXPath* 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.

Two-way Weak Alternating Tree Automata

Two-way weak alternating automata over finite labeled trees were introduced in (Calvanese et al. 2009). Differently from ordinary two-way automata over finite trees (Slutzki 1985), such automata have possibly infinite runs on finite trees, and they are called "weak" due to the specific form of the acceptance condition, which is formulated in terms of the infinite paths in a run. Note that typically, infinite runs of automata are considered in the context of infinite input structures (Grädel, Thomas, and Wilke 2002). 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 $f \in \mathcal{B}^+(I)$, we say that J *satisfies* f if assigning **true** to the elements in J and **false** to those in $I \setminus J$, makes f true. For integers i, j, with $i \leq j$, let $[i..j] = \{i, \ldots, j\}$. A *two-way weak alternating tree automaton* (2WATA) running over finite labeled binary trees 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, $\delta : S \times \mathcal{L} \to \mathcal{B}^+([-1..2] \times S)$ is the transition function, and α is the accepting 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 $[-1..2] \times S$. Intuitively, if $\delta(s, a) = f$, then each pair (c', s') appearing in f corresponds to a new copy of the automaton going to the direction suggested by c' and starting in state s'. For example, $\delta(s_1, a) = ((1, s_2) \wedge (1, s_3)) \vee ((-1, s_1) \wedge (0, s_3))$, 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 a (not necessarily finite) $\Delta^T \times S$-labeled tree $R = (\Delta^R, \ell^R)$ satisfying:

- $\varepsilon \in \Delta^R$ and $\ell^R(\varepsilon) = (x_0, s_0)$.
- Let $\ell^R(r) = (x, s)$ and $\delta(s, \ell^T(x)) = f$. Then there is a (possibly empty) set $\{(c_1, s_1), \ldots, (c_n, s_n)\} \subseteq [-1..2] \times S$ satisfying f, and such that for each $i \in [1..n]$, we have that $r{\cdot}i \in \Delta^R$, $x{\cdot}c_i \in \Delta^T$, and $\ell^R(r{\cdot}i) = (x{\cdot}c_i, s_i)$.

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.

A 2WATA is called "weak" due to the specific form of the acceptance condition α. Specifically, $\alpha \subseteq S$, and 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*. We call the partition $S = \cup_i S_i$ the *weakness partition* of $\mathbf{A}$. In addition, there exists a partial order $\prec$ 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 $S_j \prec S_i$. 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.

The following theorems show the nice computational properties of 2WATAs: linear time evaluation and exponential time non-emptiness (satisfiability).

Theorem 1 (Kupferman, Vardi, and Wolper 2000; Calvanese et al. 2009) *Given a 2WATA* $\mathbf{A}$ *and a labeled tree* T*, we can compute in time that is linear in the product of the sizes of* $\mathbf{A}$ *and* T *the set of nodes selected by* $\mathbf{A}$ *from* T*.*

Theorem 2 (Calvanese et al. 2009) *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*.*

From *μXPath* to 2WATAs

We assume that sibling trees are represented by binary trees whose nodes are additionally labeled with the special propositions *ifc*, *irs*, *hfc*, *hrs*, according to whether a node is a first child, is a right sibling, has a first child, or has a right sibling (Calvanese et al. 2009). Such trees are called *well-formed binary trees*, and *μXPath* queries are expressed over such binary trees. We need to make use of a notion of syntactic closure, similar to that of Fisher-Ladner closure of a formula of PDL (Fischer and Ladner 1979). The *syntactic closure* $CL(X : \mathcal{F})$ of a *μXPath* 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). It is easy to see that, for a *μXPath* query q, the cardinality of $CL(q)$ is linear in the length of q.

Let $q = X_0 : \mathcal{F}$ be a *μXPath* query. We show how to construct a 2WATA $\mathbf{A}_q$ that, when run over a well-formed binary tree T, accepts exactly from 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 $\sigma \in \Sigma \cup \{ifc, irs, hfc, hrs\}$,
  $$\delta_q(\sigma, \lambda) = \begin{cases} \textbf{true}, & \text{if } \sigma \in \lambda \\ \textbf{false}, & \text{if } \sigma \notin \lambda \end{cases}$$
  $$\delta_q(\neg\sigma, \lambda) = \begin{cases} \textbf{true}, & \text{if } \sigma \notin \lambda \\ \textbf{false}, & \text{if } \sigma \in \lambda \end{cases}$$
  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 shown in Figure 2.
  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

$$
\begin{array}{rcl}
\delta_q(\psi_1 \wedge \psi_2, \lambda) & = & (0, \psi_1) \ \wedge \ (0, \psi_2) \\
\delta_q(\langle \mathsf{fchild} \rangle \psi, \lambda) & = & (0, hfc) \wedge (1, \psi) \\
\delta_q(\langle \mathsf{right} \rangle \psi, \lambda) & = & (0, hrs) \wedge (2, \psi) \\
\delta_q(\langle \mathsf{fchild}^- \rangle \psi, \lambda) & = & (0, ifc) \wedge (-1, \psi) \\
\delta_q(\langle \mathsf{right}^- \rangle \psi, \lambda) & = & (0, irs) \wedge (-1, \psi) \\
\delta_q(\psi_1 \vee \psi_2, \lambda) & = & (0, \psi_1) \ \vee \ (0, \psi_2) \\
\delta_q([\mathsf{fchild}] \psi, \lambda) & = & (0, \neg hfc) \vee (1, \psi) \\
\delta_q([\mathsf{right}] \psi, \lambda) & = & (0, \neg hrs) \vee (2, \psi) \\
\delta_q([\mathsf{fchild}^-] \psi, \lambda) & = & (0, \neg ifc) \vee (-1, \psi) \\
\delta_q([\mathsf{right}^-] \psi, \lambda) & = & (0, \neg irs) \vee (-1, \psi)
\end{array}
$$

Figure 2: 2WATA transitions to decompose a formula

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 $\mathcal{F}$, due to the alternation freedom of q, 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. As for the size of $\mathbf{A}_q$, considering the size of $CL(q)$, it is easy to verify that the number of states of $\mathbf{A}_q$ is linear in the size of q.

Theorem 3 *Let q be a μXPath query, $\mathbf{A}_q$ the corresponding 2WATA, and T a well-formed binary tree. Then a node x of T is in q^T iff $\mathbf{A}_q$ selects x from T.*

Since sibling trees can be encoded in well-formed binary trees in linear time, from Theorem 1 and Theorem 3, we get:

Theorem 4 *Given a sibling tree T_s and a μXPath query q, we can compute q^{T_s} in time that is linear in the number of nodes of T_s (data complexity) and in the size of q (query complexity).*

Finally, we consider reasoning on *μXPath* queries. Satisfiability of a *μXPath* query q can be checked by checking the non-emptiness of the 2WATA $\mathbf{A}_q$ intersected with a 2WATA that accepts only binary trees that are well-formed (Calvanese et al. 2009). 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 wlog 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 2, we get:

Theorem 5 *μXPath query satisfiability and containment are in* EXPTIME.

From 2WATAs to *μXPath*

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

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

- For $f \in \mathcal{B}^+([-1..2] \times S)$, we define $\pi(f)$ inductively:

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

- For each state $s \in S$, we define $\pi(s)$ as the equation

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

- For each element S_i of the weakness partition, we define

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

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

From the above construction we get that the length of $\pi(\mathbf{A})$ is linear in the size of $\mathbf{A}$.

Theorem 6 *Let $\mathbf{A}$ be a 2WATA, $\pi(\mathbf{A})$ the corresponding μXPath query, and T a well-formed binary tree. Then $\mathbf{A}$ selects a node x from T iff x is in $(\pi(\mathbf{A}))^T$.*

Relationship between 2WATAs and MSO

To establish the relationship between 2WATAs and MSO, we make use of nondeterministic node-selecting tree automata, which were introduced in (Frick, Grohe, and Koch 2003), following earlier work on deterministic node-selecting tree automata in (Neven and Schwentick 2002). 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. It is the addition of selecting states that turns these trees from a model of tree recognition to a model of tree querying.

Theorem 7 (Frick, Grohe, and Koch 2003) *(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 from* T *by* $\mathbf{A}$.

The next two theorems establish back and forth translations between 2WATAs and NSTAs.

Theorem 8 *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* $\mathbf{A}'$.

For space reasons we do not provide here the proof of the above theorem. However, we remark that the construction used to show the result exhibits an exponential blowup: if the 2WATA has n states, then we get an NSTA with 2^{n^2+1} states. Together with the results in the previous section we get an exponential translation from *μXPath* to NSTAs. This explains why NSTAs are not useful for efficient query-evaluation algorithms, as noted in (Schwentick 2007).

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 (Gottlob and Koch 2004). The construction here uses the propositions ifc, irs, hfc, and hrs introduced earlier.

Theorem 9 *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* $\mathbf{A}'$.

While the translation from 2WATAs to NSTAs was exponential, the translation from NSTAs to 2WATAs is linear. It follows from the proof of Theorem 9 that the automaton $\mathbf{A}'$ correspond to lfp-*μXPath*, which consists of *μXPath* queries with a single, least fixpoint block. This clarifies the relationship between *μXPath* and Datalog-based languages studied in (Gottlob and Koch 2004; Frick, Grohe, and Koch 2003). In essence, *μXPath* corresponds to stratified Datalog, where rather than use explicit negation, we use alternation of least and greatest fixpoints, while lfp-*μXPath* corresponds to Datalog. The results of the last two sections provide an exponential translation from *μXPath* to lfp-*μXPath*. (Note, however, that lfp-*μXPath* does not have a computational advantage over *μXPath*.)

References

Afanasiev, L.; Grust, T.; Marx, M.; Rittinger, J.; and Teubner, J. 2008. An inflationary fixed point operator in XQuery. In *Proc. of ICDE 2008*, 1504–1506.

Baader, F.; Calvanese, D.; McGuinness, D.; Nardi, D.; and Patel-Schneider, P. F., eds. 2003. *The Description Logic Handbook: Theory, Implementation and Applications*. Cambridge University Press.

Bojanczyk, M., and Parys, P. 2008. XPath evaluation in linear time. In *Proc. of PODS 2008*, 241–250.

Bonatti, P.; Lutz, C.; Murano, A.; and Vardi, M. Y. 2008. The complexity of enriched μ-calculi. *Logical Methods in Computer Science* 4(3:11):1–27.

Calvanese, D.; De Giacomo, G.; Lenzerini, M.; and Vardi, M. Y. 2009. An automata-theoretic approach to Regular XPath. In *Proc. of DBPL 2009*, volume 5708 of *LNCS*, 18–35. Springer.

Calvanese, D.; De Giacomo, G.; and Lenzerini, M. 1999. Representing and reasoning on XML documents: A description logic approach. *J. of Logic and Computation* 9(3):295–318.

Clark, J., and DeRose, S. 1999. XML Path Language (XPath) version 1.0. W3C Recommendation, World Wide Web Consortium.

De Giacomo, G., and Lenzerini, M. 1994. Concept language with number restrictions and fixpoints, and its relationship with μ-calculus. In *Proc. of ECAI'94*, 411–415.

Fischer, M. J., and Ladner, R. E. 1979. Propositional dynamic logic of regular programs. *J. of Computer and System Sciences* 18:194–211.

Frick, M.; Grohe, M.; and Koch, C. 2003. Query evaluation on compressed trees (extended abstract). In *Proc. of LICS 2003*, 188–197.

Genevès, P.; Layaïda, N.; and Schmitt, A. 2007. Efficient static analysis of XML paths and types. In *Proc. of the ACM SIGPLAN 2007 Conf. on Programming Language Design and Implementation (PLDI 2007)*, 342–351.

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

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

Grädel, E.; Thomas, W.; and Wilke, T., eds. 2002. *Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *LNCS*. Springer.

Kozen, D. 1983. Results on the propositional μ-calculus. *Theor. Comp. Sci.* 27:333–354.

Kupferman, O.; Sattler, U.; and Vardi, M. Y. 2002. The complexity of the graded mu-calculus. In *Proc. of CADE 2002*.

Kupferman, O.; Vardi, M. Y.; and Wolper, P. 2000. An automata-theoretic approach to branching-time model checking. *J. of the ACM* 47(2):312–360.

Libkin, L., and Sirangelo, C. 2008. Reasoning about XML with temporal logics and automata. In *Proc. of LPAR 2008*, 97–112.

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

Marx, M. 2004. XPath with conditional axis relations. In *Proc. of EDBT 2004*, volume 2992 of *LNCS*, 477–494. Springer.

Marx, M. 2005. First order paths in ordered trees. In *Proc. of ICDT 2005*, volume 3363 of *LNCS*, 114–128. Springer.

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

Neven, F., and Schwentick, T. 2003. XPath containment in the presence of disjunction, DTDs, and variables. In *Proc. of ICDT 2003*, 315–329.

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

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

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

Slutzki, G. 1985. Alternating tree automata. *Theor. Comp. Sci.* 41:305–318.

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

ten Cate, B., and Segoufin, L. 2008. XPath, transitive closure logic, and nested tree walking automata. In *Proc. of PODS 2008*, 251–260.

ten Cate, B. 2006. The expressivity of XPath with transitive closure. In *Proc. of PODS 2006*, 328–337.

Vardi, M. Y., and Wolper, P. 1984. Automata-theoretic techniques for modal logics of programs. In *Proc. of STOC'84*, 446–455.