Conditional XPath

MAARTEN MARX
Informatics Institute, Universiteit van Amsterdam
The Netherlands

No Institute Given

Abstract. XPath 1.0 is a variable free language designed to specify paths between nodes in XML documents. Such paths can alternatively be specified in first-order logic. The logical abstraction of XPath 1.0, usually called Navigational or Core XPath, is not powerful enough to express every first-order definable path. In this paper we show that there exists a natural expansion of Core XPath in which every first-order definable path in XML document trees is expressible. This expansion is called Conditional XPath. It contains additional axis relations of the form `(child::n[F])+`, denoting the transitive closure of the path expressed by `child::n[F]`. The difference with XPath's `descendant::n[F]` is that the path `(child::n[F])+` is conditional on the fact that all nodes in between should be labeled by `n` and should make the predicate `F` true. This result can be viewed as the XPath analogue of the expressive completeness of the relational algebra with respect to first-order logic.

1 Introduction

XPath 1.0 [46] is a variable free language used for navigation between nodes in XML documents. XPath is an important part of other XML technologies such as XSLT [48] and XQuery [47]. The topic of this paper is the expressivity of XPath measured in terms of first-order logic.

We use the abstraction to the logical core of XPath 1.0 (called *Core XPath)* developed in [15, 18]. Core XPath is interpreted on XML document tree models. The central expression in XPath is the location path $\mathsf{axis} :: \mathit{node\_label}[\mathsf{filter}]$ which when evaluated at node n yields an answer set consisting of nodes n' such that axis relates n and n', the tag of n' is *node_label*, and the expression filter evaluates to true at n'. Provided the relation axis and the predicate filter are first-order definable, this is a first-order definition of the meaning of location paths. All Core XPath axis relations are first-order definable from the primitives *descendant* and *following_sibling*. It follows from [17] that

Fact 1. Every Core XPath expression is equivalent to a first-order formula in two free variables in the signature with two binary *descendant* and *following_sibling* relation symbols and unary predicates.

Both XPath expressions and first order formulas in two free variables define sets of paths in trees. For instance, the following two expressions define the same set

of paths

$$\texttt{descendant} :: \texttt{p}[\texttt{not child} :: *]/\texttt{following\_sibling} :: \texttt{q} \tag{1}$$

$$\exists z(x\,\texttt{descendant}\,z \wedge P(z) \wedge \neg\exists w(z\,\texttt{child}\,w) \wedge z\,\texttt{following\_sibling}\,y \wedge Q(y)). \tag{2}$$

Here $z\,\texttt{child}\,w$ abbreviates the formula $z\,\texttt{descendant}\,w \wedge \neg\exists v(z\,\texttt{descendant}\,v \wedge v\,\texttt{descendant}\,y)$.

The converse of Fact 1 is a desirable property for a query language. If true, Core XPath could be called expressively complete for first order definable paths: if a path is first order definable, there is an equivalent Core XPath expression which defines it. Unfortunately, the converse of Fact 1 is false, a result which goes back to [23]. This paper shows that a rather simple addition to Core XPath is sufficient to obtain the converse. To introduce this addition and to see why the converse of Fact 1 fails consider the following expression:

$$\textsf{self} :: *[(\texttt{child} :: *[\texttt{F}_1])^*/\texttt{child} :: *[\texttt{F}_2]]. \tag{3}$$

Here $(\texttt{child} :: *[\texttt{F}_1])^*$ is not a Core XPath expression. It denotes the reflexive and transitive closure of the relation expressed by $\texttt{child} :: *[\texttt{F}_1]$. Thus the following are equivalent

$$(\texttt{child} :: *[\texttt{F}_1])^* \tag{4}$$

$$x = y \vee (x\,\texttt{descendant}\,y \wedge \texttt{F}_1(y) \wedge \forall z(x\,\texttt{descendant}\,z \wedge z\,\texttt{descendant}\,y \rightarrow \texttt{F}_1(z))). \tag{5}$$

Expression (3) defines the set of all paths (x, x) such that x has a descendant y at which $\texttt{F}_2$ is true and for all z strictly in between x and y, $\texttt{F}_1$ is true. Thus the main filter expression in (3) corresponds to a first order formula $\phi(x)$ which is written using the three variables $\{x, y, z\}$. It is not hard but tedious to show that three variables are necessary to express this filter expression. But every Core XPath filter expression is equivalent to a first order formula in two variables [30]. Hence (3) is not Core XPath expressible. Note that the main filter expression in (3) expresses the formula $\mathit{Until}(\texttt{F}_2, \texttt{F}_1)$ from temporal logic. The relation between XPath and temporal logic is further discussed in Section 7.1.

So each expansion of Core XPath for which the converse of Fact 1 holds must be able to express $(\texttt{child} :: *[\texttt{F}_1])^*$. We call $(\texttt{child} :: *[\texttt{F}_1])^*$ a *conditional axis* relation. It differs from the XPath axis $\texttt{descendant\_or\_self} = \texttt{child}^*$ in that it is conditional on the filter expression $\texttt{F}_1$. Conditional XPath is the expansion of Core XPath with conditional axis relations in all four directions descendant, ancestor, following-sibling, and preceding-sibling.

Our main result is that the converse of Fact 1 holds for Conditional XPath. That means that we can call Conditional XPath *expressively complete* for first order definable paths. This is an XPath analogue of the first order expressive completeness of the relational algebra [9]. Expressive completeness is a desirable property provided that the formalism retains the positive properties of Core

XPath. One of the nice features of XPath is that queries are drawable as pictures of trees. Conditional XPath retains this feature because the conditional axis relations are drawable using ellipsis. Another important property of Core XPath is its linear time query evaluation complexity [17, 18]. This property is also inherited by Conditional XPath (Fact 2).

The proofs in this paper are based on Shelah's composition method [43], Ehrenfeucht–Fraïssé games [10], the close resemblance of XPath and Tarski's relation algebras [42], and the encoding of XML document trees into ordered binary trees. Furthermore we establish a syntactic normal form for Conditional XPath expressions inspired by separation results in temporal logic [13].

This paper is organized as follows. The next section handles the preliminaries: it defines Core and Conditional XPath, and first-order logic of trees. Section 3 contains all results, which are proved in the two subsequent sections. Section 6 discusses computational complexity of the defined languages and the complexity of the translation from first-order logic. Related work and conclusions end the paper.

2 Preliminaries

2.1 Trees and first order logic

The semantics of XPath expressions is given with respect to *finite node labeled, sibling ordered trees*. If no confusion arises, we just call them trees. A tree domain N is a finite set of finite sequences of natural numbers closed under taking initial segments, and for any sequence s, if $s \cdot k \in N$, then either $k = 0$ or $s \cdot k - 1 \in N$. A node labeled, sibling ordered tree consists of a tree domain and a function labeling each node with a set of primitive symbols from some alphabet. We denote them by $\mathfrak{M}$.

Node labeled, sibling ordered trees can be queried in a first order language in an appropriate signature. The signature consists of

- two binary relation symbols, $R_{\Downarrow}$ and $R_{\Rightarrow}$, denoting the descendant and the following sibling relation, respectively;
- unary predicate symbols corresponding to the node labels;
- equality.

For $n, n' \in N$, $nR_{\Downarrow}n'$ holds iff n is a proper prefix of n'; $nR_{\Rightarrow}n'$ holds iff $n = s \cdot k$, $n' = s \cdot l$ and $k < l$; $P(n)$ holds iff n is labeled by the symbol P.

Definition 1. *We define the language* FO^{tree} *as follows:*

- *the atomic formulas are* $x = y$, $P_i(x)$, $xR_{\Downarrow}y$, *and* $xR_{\Rightarrow}y$, *for any variables* x, y, *for any unary predicate symbol* P_i*;*
- *if* ϕ, ψ *are formulas, then* $\neg\phi, \phi \wedge \psi$, *and* $\exists x\phi$ *are formulas.*

FO^{tree} is interpreted on node labeled, sibling ordered trees as indicated above with variables ranging over the set of nodes. We use the standard notation $\mathfrak{M} \models \phi(a_1, \ldots, a_k)$ to denote that $\phi(x_1, \ldots, x_k)$ is true at $\mathfrak{M}$ under any assignment g which maps x_i to a_i.

Definition 2. *Let $\phi(x_1, \ldots, x_k)$ be a FO^{tree} formula.*

1. *For $\mathfrak{M}$ a tree with domain N, the* denotation *of $\phi(x_1, \ldots, x_k)$ in $\mathfrak{M}$ (abbreviated by $[\![\phi(x_1, \ldots, x_k)]\!]_{\mathfrak{M}}$) is the set $\{(n_1, \ldots, n_k) \in N^k \mid \mathfrak{M} \models \phi(n_1, \ldots, n_k)\}$.*
2. *The* meaning *of $\phi(x_1, \ldots, x_k)$ (abbreviated by $[\![\phi(x_1, \ldots, x_k)]\!]_{\mathsf{tree}}$) is the class of all pairs $(\mathfrak{M}, (n_1, \ldots, n_k))$ for $\mathfrak{M}$ a tree such that $\mathfrak{M} \models \phi(n_1, \ldots, n_k)$.*

The meaning of a sentence ϕ is then simply a subclass of the class of all trees.

The main subject of this paper is the equivalence relation between formulas (of different languages) on trees.

Definition 3. *Let ϕ and ψ be formulas, possibly of different languages. We say that ϕ and ψ are* equivalent *if $[\![\phi]\!]_{\mathsf{tree}} = [\![\psi]\!]_{\mathsf{tree}}$.*

Thus "equivalence" means "equivalence on the class of all trees". Hence $xR_{\Downarrow}y$ and $xR_{\Downarrow}y \wedge \neg yR_{\Downarrow}x$ are equivalent.

Remark 1. Note that the nodes in our trees may contain multiple labels. Thus the labels do not correspond directly to XML tags, of which each node has exactly one. Instead our labels model both XML tag names as well as attribute-value pairs. We did not include attribute-functions directly in the signature because of the following: for each attribute a, each binary relation θ on the range of a, and each constant c in the range of a, the formula $a(x)\theta c$ can be seen as a unary predicate stating that x has the property $\lambda x.a(x)\theta c$. Thus formulas of the form $a(x)\theta c$ can be encoded inside FO^{tree}, and we need not consider them separately.

2.2 Navigational XPath

Core XPath [15, 18] is a stripped down version of XPath 1.0 which focuses on the logical core of the XPath 1.0 language. It contains just what is needed to define paths in, and select nodes from, the *skeleton* of an XML tree. The skeleton is what is left of an XML document after removing all PCDATA elements. Thus the skeleton is a sibling ordered tree whose nodes are labeled by tags and attribute-value pairs. We view the set of tags and attribute-value pairs as just one set of primitive symbols.

It is instructive to view Core XPath as a two sorted language. Location paths like `descendant :: para` are of the sort `path` and denote paths in the tree. The node tests like `para` and the predicates in filter expressions (those between square brackets) are of the sort `node` and denote sets of nodes in an XML tree. The two sorts differ syntactically in the operations which are allowed on them. Complex path expressions can be formed by the regular expression operators / (concatenation), $\cup$ (union) and the Kleene $*$ (reflexive and transitive closure). Complex node expressions are formed by the Boolean operations.

The use of the Kleene $*$ is heavily restricted in XPath 1.0, and also in Core XPath. This restriction is lifted slightly in Conditional XPath. The restriction is such that all expressions are first order definable. It is instructive to see how these two languages live inside a larger system, which we might dub *Regular XPath*.

We define Regular XPath in the style set by two closely related languages known as *Kleene algebras with tests* and *Propositional Dynamic Logic* [24, 20]. Consider the following grammar. First we define the basic path expressions called step:

$$\mathsf{step} \ ::= \ \mathsf{child} \mid \mathsf{parent} \mid \mathsf{right} \mid \mathsf{left}.$$

child and parent are just the XPath axis with the same name. right and left denote *immediate sibling* to the right and to the left, respectively. Given a tree, these expressions define *paths* or equivalently a set of pairs of nodes. For instance, the meaning of child is the set of all edges in the tree. From the steps, we define the complex path expressions:

$$\mathsf{p\_wff} \ ::= \ \mathsf{step} \mid \mathsf{p\_wff}/\mathsf{p\_wff} \mid \mathsf{p\_wff} \cup \mathsf{p\_wff} \mid \mathsf{p\_wff}^* \mid ?\mathsf{n\_wff}.$$

p_wff abbreviates "path-wff". Wff (pronounced as "wif") is short for well formed formula. We use $\mathsf{p\_wff}^+$ as an abbreviation of $\mathsf{p\_wff}/\mathsf{p\_wff}^*$. The meaning of the regular expression operators is as expected. For instance, child^* is the same as `descendant_or_self`, the expression $\mathsf{right}/\mathsf{right}^*$ denotes `following-sibling` and $\mathsf{parent}^*/\mathsf{right}^+/\mathsf{child}^*$ denotes the XPath axis `following`.

The non regular expression operator ? is called a *test*. It takes a node sort expression and returns a path sort expression. It behaves like a test in a program: it either fails, or succeeds but then remains in the same state.

The node expressions are defined as follows: for p_i any atomic label,

$$\mathsf{n\_wff} \ ::= \ p_i \mid \top \mid \langle \mathsf{p\_wff} \rangle \mid \neg \mathsf{n\_wff} \mid \mathsf{n\_wff} \vee \mathsf{n\_wff} \mid \mathsf{n\_wff} \wedge \mathsf{n\_wff}.$$

The meaning of a node expression is a set of nodes. Thus the meaning of p_i is the set of all nodes labeled by p_i. The boolean operations have their standard interpretation. Thus $?\top$ denotes the self axis. The operator $\langle \cdot \rangle$ takes a path expression and returns a node expression. The meaning of $\langle \mathsf{p\_wff} \rangle$ is the same as that of p_wff written inside square brackets in XPath 1.0: it is the set of nodes which *start a* p_wff relation.

One of the difficulties in learning XPath 1.0 is that a p_wff expression inside a filter expression does not mean the same as p_wff outside square brackets. We solved this by putting the hooks around it. It is reminiscent of the diamond formulas in Propositional Dynamic Logic. Observe that the language of Regular XPath is now nicely balanced: both sorts have an operator which takes an expression of the other sort and turns it into something of it own sort.

The relation between the language just defined and XPath 1.0 is exemplified in Table 1 in which some expressions in our notation are given also as equivalent XPath expressions.

$\mathsf{child} :: \mathtt{p_i}$	$\mathsf{child}/?p_i$
$\mathsf{child} :: \mathtt{p_i}[\mathtt{descendant} :: *]$	$\mathsf{child}/?p_i/?\langle\mathsf{child}^+\rangle$
$/\mathtt{descendant} :: \mathtt{p_i}$	$?\neg\langle\mathsf{parent}\rangle/\mathsf{child}^+/?p_i$
$\mathsf{child} :: *$	child
$\mathtt{self} :: \mathtt{p_i}[\mathsf{child}]$	$?(p_i \wedge \langle\mathsf{child}\rangle)$
$\mathtt{preceding} :: \mathtt{p_i}$	$\mathsf{parent}^*/\mathsf{left}^+/\mathsf{child}^*/?p_i$
$\mathtt{following} :: \mathtt{p_i}$	$\mathsf{parent}^*/\mathsf{right}^+/\mathsf{child}^*/?p_i.$

Table 1. Equivalent XPath 1.0 and Regular XPath expressions.

$$
\begin{aligned}
[\![p_i]\!]_{\mathfrak{M}} &= \{n \text{ in } \mathfrak{M} \mid n \text{ is labeled with } p_i\}\\
[\![\langle R\rangle]\!]_{\mathfrak{M}} &= \{n \text{ in } \mathfrak{M} \mid \exists n', (n,n') \in [\![R]\!]_{\mathfrak{M}}\}\\
[\![\top]\!]_{\mathfrak{M}} &= \{n \text{ in } \mathfrak{M}\}\\
[\![\neg A]\!]_{\mathfrak{M}} &= \{n \text{ in } \mathfrak{M} \mid n \notin [\![A]\!]_{\mathfrak{M}}\}\\
[\![A \wedge B]\!]_{\mathfrak{M}} &= [\![A]\!]_{\mathfrak{M}} \cap [\![B]\!]_{\mathfrak{M}}.\\
[\![A \vee B]\!]_{\mathfrak{M}} &= [\![A]\!]_{\mathfrak{M}} \cup [\![B]\!]_{\mathfrak{M}}.\\
[\![\mathsf{child}]\!]_{\mathfrak{M}} &= \{(n,n') \mid n' = n \cdot k, \text{ for } k \text{ a natural number}\}\\
[\![\mathsf{parent}]\!]_{\mathfrak{M}} &= [\![\mathsf{child}]\!]^{-1}_{\mathfrak{M}}\\
[\![\mathsf{right}]\!]_{\mathfrak{M}} &= \{(n,n') \mid n = s \cdot k \text{ and } n' = s \cdot k + 1, \text{ for } s \cdot k \text{ a sequence}\}\\
[\![\mathsf{left}]\!]_{\mathfrak{M}} &= [\![\mathsf{right}]\!]^{-1}_{\mathfrak{M}}\\
[\![?A]\!]_{\mathfrak{M}} &= \{(n,n) \mid n \in [\![A]\!]_{\mathfrak{M}}\}\\
[\![R \cup S]\!]_{\mathfrak{M}} &= [\![R]\!]_{\mathfrak{M}} \cup [\![S]\!]_{\mathfrak{M}}\\
[\![R/S]\!]_{\mathfrak{M}} &= [\![R]\!]_{\mathfrak{M}} \circ [\![S]\!]_{\mathfrak{M}} \;\; (= \{(n,n') \mid \exists n''((n,n'') \in [\![R]\!]_{\mathfrak{M}} \wedge (n'',n') \in [\![S]\!]_{\mathfrak{M}})\})\\
[\![R^+]\!]_{\mathfrak{M}} &= [\![R]\!]^+_{\mathfrak{M}} \;\; (= [\![R]\!]_{\mathfrak{M}} \cup ([\![R]\!]_{\mathfrak{M}} \circ [\![R]\!]_{\mathfrak{M}}) \cup ([\![R]\!]_{\mathfrak{M}} \circ [\![R]\!]_{\mathfrak{M}} \circ [\![R]\!]_{\mathfrak{M}}) \cup \dots)\\
[\![R^*]\!]_{\mathfrak{M}} &= \{(n,n) \mid n \text{ in } \mathfrak{M}\} \cup [\![R^+]\!]_{\mathfrak{M}}.
\end{aligned}
$$

Table 2. The semantics of XPath.

Throughout the text, we use variables R, S, T to denote arbitrary path wffs and A, B, C to denote node wffs. We often abbreviate $\mathsf{p\_wff}/?\mathsf{n\_wff}$ as $\mathsf{p\_wff}?\mathsf{n\_wff}$ (as in $\mathsf{child}?A$ instead of $\mathsf{child}/?A$). The semantics of XPath expressions is given with respect to node labeled sibling ordered trees $\mathfrak{M}$. Given a tree $\mathfrak{M}$ and an expression R, the denotation or meaning of R in $\mathfrak{M}$ is written as $[\![R]\!]_{\mathfrak{M}}$. As promised, path wffs denote sets of pairs, and node wffs sets of nodes. Table 2 contains the definition of $[\![\,\cdot\,]\!]_{\mathfrak{M}}$. The equivalence with the W3C syntax and semantics (cf., e.g., [4, 17, 49]) should be clear.

Definition 4. *Let R be a Regular XPath expression. Then $[\![R]\!]_{\mathsf{tree}}$ denotes the class of all pairs $\mathfrak{M}, (n,n')$ such that $(n,n') \in [\![R]\!]_{\mathfrak{M}}$.*

2.3 Conditional and Core XPath

In Regular XPath one can define relations which are not first order definable, like $(\mathsf{child}/\mathsf{child})^*$, the relation between nodes which are an even number of child steps removed from each other. In Conditional and Core XPath, the use of the Kleene $*$ is so much restricted that all relations are still first order definable.

The use of the Kleene star is very much restricted in Core XPath. In fact, so much that it is not even needed as a connective, and the allowed transitive closure paths are the four primitives `descendant, ancestor, following-sibling` and `preceding-sibling` (above we saw how the `following` and `preceding` axis can be defined from these). In Conditional XPath we allow a bit more.

Definition 5.

1. *Core XPath is the language defined just like Regular XPath but with the following restricted set of path wffs:*

$$\mathsf{p\_wff} \ ::= \ \mathsf{step} \mid \mathsf{step}^* \mid ?\mathsf{n\_wff} \mid \mathsf{p\_wff}/\mathsf{p\_wff} \mid \mathsf{p\_wff} \cup \mathsf{p\_wff}.$$

2. *Conditional XPath is the language defined just like Regular XPath but with the following restricted set of path wffs:*

$$\mathsf{p\_wff} \ ::= \ \mathsf{step} \mid (\mathsf{step}/?\mathsf{n\_wff})^* \mid ?\mathsf{n\_wff} \mid \mathsf{p\_wff}/\mathsf{p\_wff} \mid \mathsf{p\_wff} \cup \mathsf{p\_wff}.$$

The restrictions put on the use of the Kleene star make that the meaning of the expressions can still be defined in a first order manner. For example,

$$\begin{aligned} [\![\mathsf{child}^*]\!]_{\mathfrak{M}} &= [\![x = y \vee xR_{\Downarrow}y]\!]_{\mathfrak{M}} \\ [\![(\mathsf{child}/?p_i)^*]\!]_{\mathfrak{M}} &= [\![x = y \vee (xR_{\Downarrow}y \wedge P_i(y) \wedge \forall z(xR_{\Downarrow}z \wedge zR_{\Downarrow}y \rightarrow P_i(z)))]\!]_{\mathfrak{M}}. \end{aligned}$$

Thus each Core and Conditional XPath expression is equivalent to an FO^{tree} formula. More precisely, each node wff is equivalent to an FO^{tree} formula in one free variable, and each path wff to an FO^{tree} formula in two free variables.

But not only there is a bound on the number of free variables in the corresponding first order formula, we can also bound the *total* number of variables used. This follows from known results in modal logic and Tarski's relation algebras [5, 42]. For completeness, we provide the proof.

Proposition 1. *Every Conditional XPath path wff is equivalent to a* FO^{tree} *formula* $\phi(x, y)$ *in* three *variables.*

Proof. The proof of this proposition uses the standard translation from modal and temporal logic [5]. We present the translation in the style of [45]. We define two translation functions $(\cdot)^t$ and $(\cdot)^{pt}$ for node wffs and path wffs, respectively. By replacing each occurrence of the form $(\mathsf{step}/?\mathsf{n\_wff})^*$ by the equivalent $?\top \cup$

$(\mathsf{step}/?\mathsf{n\_wff})^+$, we may assume that the input formula contains no Kleene stars.

$$
\begin{array}{ll}
p_i^t & = P_i(x) \\
(\cdot)^t & \text{commutes with the booleans} \\
\langle R/?A \rangle^t & = \exists y(R^{pt} \wedge \exists x(x = y \wedge A^t)).
\end{array}
$$

$$
\begin{array}{llll}
\mathsf{child}^{pt} & = xR_{\Downarrow}y \wedge \neg\exists z(xR_{\Downarrow}z \wedge zR_{\Downarrow}y) & (\mathsf{child}^+)^{pt} & = xR_{\Downarrow}y \\
\mathsf{parent}^{pt} & = yR_{\Downarrow}x \wedge \neg\exists z(yR_{\Downarrow}z \wedge zR_{\Downarrow}x) & (\mathsf{parent}^+)^{pt} & = yR_{\Downarrow}x \\
\mathsf{right}^{pt} & = xR_{\Rightarrow}y \wedge \neg\exists z(xR_{\Rightarrow}z \wedge zR_{\Rightarrow}y) & (\mathsf{right}^+)^{pt} & = xR_{\Rightarrow}y \\
\mathsf{left}^{pt} & = yR_{\Rightarrow}x \wedge \neg\exists z(yR_{\Rightarrow}z \wedge zR_{\Rightarrow}x) & (\mathsf{left}^+)^{pt} & = yR_{\Rightarrow}x
\end{array}
$$

$$
\begin{array}{ll}
((\mathsf{step}?A)^+)^{pt} & = (\mathsf{step}^+)^{pt} \wedge \exists x(x = y \wedge A^t) \wedge \\
& \quad \forall z(\exists y(z = y \wedge (\mathsf{step}^+)^{pt}) \wedge \exists x(z = x \wedge (\mathsf{step}^+)^{pt}) \rightarrow \exists x(z = x \wedge A^t) \\
(?A)^{pt} & = x = y \wedge A^t \\
(R \cup S)^{pt} & = R^{pt} \vee S^{pt} \\
(R/S)^{pt} & = \exists z(\exists y(z = y \wedge R^{pt}) \wedge \exists x(z = x \wedge S^{pt})).
\end{array}
$$

A tedious induction shows that the translation is correct.

The translation gets a little opaque with all the substitutions. One can check that $((\mathsf{child}?p_i)^+)^{pt}$ is equivalent to the expected $xR_{\Downarrow}y \wedge P_i(y) \wedge \forall z(xR_{\Downarrow}z \wedge zR_{\Downarrow}y \rightarrow P_i(z))$.

Remark 2. Although we borrowed the name Core XPath from [17], our language is slightly more expressive, due to the availability of the left and right axis relations. This is easily shown using a translation based on the equivalences in Table 1 (cf. also [28]). Arguably, the left and right axis relations must be available in an XPath dialect which calls itself *navigational*. For instance we need them to express XPath's `child::A[n]` for `n` a natural number without using negation.

3 Expressive completeness of Conditional XPath

This section lists our main results.

Definition 6. *Let L be a language for which $[\![R]\!]_{\mathsf{tree}}$ is defined for every $R \in L$. We call L* expressively complete for first order definable paths *if for every $\phi(x,y) \in FO^{\mathsf{tree}}$, there exists a formula $R \in L$ such that $[\![\phi(x,y)]\!]_{\mathsf{tree}} = [\![R]\!]_{\mathsf{tree}}$.*

Theorem 1. *Conditional XPath is expressively complete for first order definable paths. In particular, for every FO^{tree} formula $\phi(x,y)$, there exists an equivalent Conditional XPath path expression.*

Corollary 1. *(i) For every FO^{tree} formula $\phi(x)$, there exists an equivalent Conditional XPath node expression.*
(ii) For every FO^{tree} sentence ϕ, there exists a node expression A such that for every tree $\mathfrak{M}$, $\mathfrak{M} \models \phi$ iff $\mathsf{root} \in [\![A]\!]_{\mathfrak{M}}$.

Proof. For (i), consider the formula $\phi(x) \wedge x = y$. By Theorem 1, there is an equivalent path expression R. Then the equivalent node expression is $\langle R \rangle$. For (ii), consider the formula $\phi \wedge \neg \exists y \; y R_{\Downarrow} x$. There is an equivalent node expression A by (i). A is such that it can only be true at the root.

To show Theorem 1 we proceed in two steps. In Section 4 we show

Theorem 2. *On the class of finite node labeled sibling ordered trees every FO^{tree} formula $\phi(x, y)$ is equivalent to an FO^{tree} formula $\phi'(x, y)$ in the same signature which is written using just three variables in total.*

This theorem yields a sufficient criterion for being complete for paths. Call a language L closed under complementation of path expressions if for every path expression $R \in L$ there exists a path expression $S \in L$ which defines the complement of R. That is, on every model $\mathfrak{M}$, $[\![S]\!]_{\mathfrak{M}} = \{(n, n') \mid (n, n') \notin [\![R]\!]_{\mathfrak{M}}\}$.

Corollary 2. *Any expansion of Core XPath which is closed under complementation of path expressions is expressively complete for first order definable paths.*

Proof. Let $\phi(x, y)$ be an FO^{tree} formula. By Theorem 2, we may assume it is written using three variables. Thus $\phi(x, y)$ is a formula in a signature with at most binary relations symbols which contains at most three free and bound (possibly reused) variables and at most two free variables. Each such formula is equivalent to an expression R in Tarski's relation algebras [42]. These are algebras of the form $(A, \cup, \overline{(\cdot)}, \circ, (\cdot)^{-1}, id)$ with A a set of binary relations, and the operators having the standard set theoretic meaning. R is an expression in the signature $R_{\Downarrow}, R_{\Rightarrow}$ and binary atoms under the identity (that is, of the form $P \cap id$). These atoms are closed under the operation $(\cdot)^{-1}$. Thus we may assume that R does not contain $(\cdot)^{-1}$, as $(\cdot)^{-1}$ can be pushed onto the atoms. Now let L be an expansion as in the statement of the corollary. Then L can express $\cup$ and $\circ$ by the Core XPath operators $\cup$ and /, respectively. The constant id is definable in Core XPath as $?\top$. And complementation is expressible by assumption. Thus R is expressible in L.

The second step in the proof of Theorem 1 is to show that Conditional XPath is closed under complementation of path expressions. This will be done in Section 5.

Thus adding the conditional path expressions cures the expressive incompleteness of Core XPath. One may wonder if Core XPath itself has an elegant characterization too. Indeed it has, both for answer sets and for the paths it can define [30]. For instance, the answer sets definable by Core XPath queries are exactly those definable by FO^{tree} formulas $\phi(x)$ written using just two variables which may additionally contain binary predicates corresponding to the child and right relations.

4 Binary formulas on ordered trees need only 3 variables

In this section we prove Theorem 2. We first reduce the problem to ordered binary trees, and then prove the theorem by showing that the encoding on unranked ordered trees into ordered binary trees can be done using three variables.

An ordered binary tree domain is a finite set of finite sequences of 0's and 1's closed under taking initial segments. Note that nodes may have a 1 successor without having a 0 successor. This is more convenient in our encodings. Labeled trees are defined as before, again nodes may have multiple labels.

Let FO^{bin} be the first order language in the signature with two binary relations $<_{\mathsf{left}}$ and $<_{\mathsf{right}}$ and unary predicates corresponding to the labels Σ. $x <_{\mathsf{left}} y$ is read "y is a left descendant of x" and similarly for $x <_{\mathsf{right}} y$. Let $x < y$ abbreviate $x <_{\mathsf{left}} y \vee x <_{\mathsf{right}} y$, and $x \leq y$ abbreviate $x < y \vee x = y$. FO_3^{bin} denotes the restriction of FO^{bin} to formulas in at most three variables. The atomic FO^{bin} formulas are interpreted on binary trees $\mathfrak{M}$ as follows: for nodes n, n',

$$\begin{array}{ll} \mathfrak{M} \models n <_{\mathsf{left}} n' & \iff n' = n \cdot 0 \cdot s \text{ for some (possibly empty) sequence } s \\ \mathfrak{M} \models n <_{\mathsf{right}} n' & \iff n' = n \cdot 1 \cdot s \text{ for some (possibly empty) sequence } s \\ \mathfrak{M} \models \sigma(n) & \iff n \text{ is labeled by } \sigma. \end{array}$$

For x, y terms, let $lca(x, y)$ denote the least common ancestor of x and y. On ordered binary trees the formula $z = lca(x, y)$ is expressible using no extra variables as the disjunction of the formulas

$$x = y = z \tag{6}$$
$$x < y \wedge x = z \tag{7}$$
$$y < x \wedge y = z \tag{8}$$
$$z <_{\mathsf{left}} x \wedge z <_{\mathsf{right}} y \tag{9}$$
$$z <_{\mathsf{left}} y \wedge z <_{\mathsf{right}} x. \tag{10}$$

Tarski showed that three variables are not sufficient if we consider binary trees *without* order [42], Section 3.6.

For $\bar{x}$ a finite set of variables, let $O(\bar{x})$ denote a tree order type of $\bar{x}$, that is, a maximal consistent set of formulas in the language with $<_{\mathsf{left}}$, $<_{\mathsf{right}}$, $=$ and *lca* terms generated from $\bar{x}$. Because *lca* generates finitely many new elements on a finite set, each $O(\bar{x})$ can equivalently be written as a conjunction of atoms in $\bar{x}$ and additional free variables without *lca* terms. We denote this conjunction by $T(\bar{x})$ with $\bar{x}$ all the free variables occurring in the conjunction.

A conjunction of binary atoms in variables $\bar{x}$ can be seen as specifying a directed multigraph over the set of nodes $\bar{x}$ in which the edges are labeled by the atoms. A formula $\exists \bar{x} C(\bar{x}, \bar{y})$, is called a *conjunctive tree query* if $C(\bar{x}, \bar{y})$ is a conjunction whose graph is a tree. Each tree order $T(\bar{x})$ can equivalently be written as a conjunctive tree query. If we disregard the equality edges, then the shape of this tree query exactly corresponds to the way the denotation of the variables are located in any tree which satisfies $T(\bar{x})$. An example is given in Figure 1. Note that the tree indeed describes an order type: for every two variables u, v in the tree, for each of the relations $R \in \{<_{\mathsf{left}}, <_{\mathsf{right}}, =\}$, either uRv or $\neg uRv$ follows from the tree description. In the sequel we will equate $T(\bar{x})$ with its description as a conjunctive tree query.

Our main technical result is the following composition lemma, similar to Lemma 4.1 in [41]. For A, B trees and $\bar{a}, \bar{b}$ sequences of elements from A and B,

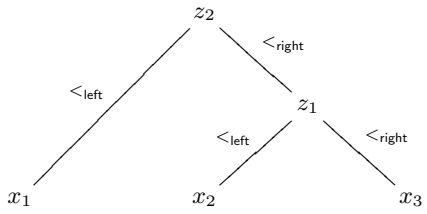

Fig. 1. A tree query.

respectively, we use the notation $A, \bar{a} \equiv_n^{FO^{\mathsf{bin}}} B, \bar{b}$ to indicate that $A, \bar{a}$ and $B, \bar{b}$ satisfy exactly the same FO^{bin} formulas up to quantifier depth n.

Lemma 1. *Let A, B be finite ordered binary trees and $\bar{a}$ a sequence of elements from A which is closed under least common ancestors. Let $\bar{b}$ be a sequence of the same length from B. Let $T_A(\bar{a})$ and $T_B(\bar{b})$ be the order types of $\bar{a}$ in A, and $\bar{b}$ in B, respectively. Then the following are equivalent:*

1. *$A, \bar{a} \equiv_n^{FO^{\mathsf{bin}}} B, \bar{b}$;*
2. *for each $R \in \{<_{\mathsf{left}}, <_{\mathsf{right}}, =\}$, for each a_i, a_j in $\bar{a}$, $a_i R a_j$ in $T_A(\bar{a})$ iff $b_i R b_j$ in $T_B(\bar{b})$, and for each edge $a_i R a_j$ in $T_A(\bar{a})$, $A, a_i, a_j \equiv_n^{FO^{\mathsf{bin}}} B, b_i, b_j$.*

The lemma has the following corollary:

Corollary 3. *Let $\psi(\bar{x}) \in FO^{\mathsf{bin}}$ be a formula of quantifier depth n such that $\psi(\bar{x})$ implies $T(\bar{x})$ for some order type T. Then $\psi(\bar{x})$ is equivalent to a disjunction of conjunctive tree queries whose edges are labeled by binary FO^{bin} formulas of quantifier depth n.*

Proof. Let $\psi(\bar{x}) \in FO^{\mathsf{bin}}$ be a formula of quantifier depth n such that $\psi(\bar{x})$ implies $T(\bar{x})$ for some order type T. Let C_ψ and $\overline{C_\psi}$ be the classes of all finite ordered binary trees $(A, \bar{a})$ in the signature of ψ such that $A \models T(\bar{a}) \wedge \psi(\bar{a})$, and $A \models T(\bar{a}) \wedge \neg\psi(\bar{a})$, respectively. First observe that for any $(A, \bar{a}) \in C_\psi$, for any $(B, \bar{b}) \in \overline{C_\psi}$, there exists a formula $\delta_{A\bar{a}B\bar{b}}(x_i, x_j)$ of quantifier rank n in two free variables x_i, x_j such that $x_i R x_j$ is an edge in $T(\bar{x})$ and

$$A \models \delta_{A\bar{a}B\bar{b}}(a_i, a_j) \text{ and } B \not\models \delta_{A\bar{a}B\bar{b}}(b_i, b_j).$$

For, suppose to the contrary. Then for each edge $x_i R x_j$ in $T(\bar{x})$, $A, a_i, a_j \equiv_n^{FO^{\mathsf{bin}}} B, b_i, b_j$. Moreover $A \models T(\bar{a})$ and $B \models T(\bar{b})$ by definition. Thus by Lemma 1, $A, \bar{a} \equiv_n^{FO^{\mathsf{bin}}} B, \bar{b}$. But then as $A \models \psi(\bar{a})$, also $B \models \psi(\bar{b})$, a contradiction.
Now define

$$\delta(\bar{x}) = \bigvee_{(A,\bar{a}) \in C_\psi} \bigwedge_{(B,\bar{b}) \in \overline{C_\psi}} \delta_{A\bar{a}B\bar{b}}(x_i, x_j).$$

As the $\delta_{A\bar{a}B\bar{b}}(x_i, x_j)$ are of quantifier rank at most n and in a finite signature, $\delta(\bar{x})$ is finite modulo logical equivalence. We claim that $\delta(\bar{x}) \wedge T(\bar{x})$ is equivalent to $\psi(\bar{x})$. For suppose $A \models \psi(\bar{a})$, for $A, \bar{a}$ arbitrary. Then $A \models T(\bar{a})$ by assumption and $(A, \bar{a})$ in C_ψ. But then $A \models \bigwedge_{B \in \overline{C_\psi}} \delta_{A\bar{a}B\bar{b}}(\bar{a})$, and a fortiori also $A \models \delta(\bar{a})$. Now suppose that $B \models \delta(\bar{b}) \wedge T(\bar{b})$, for $B, \bar{b}$ arbitrary. Then for some $A, \bar{a}$, $B \models \bigwedge_{(B,\bar{b}) \in \overline{C_\psi}} \delta_{A\bar{a}B\bar{b}}(\bar{b})$. In particular $B \models \delta_{A\bar{a}B\bar{b}}(\bar{b})$. Now suppose to the contrary that that $B \not\models \psi(\bar{b})$. Because we assumed that $B \models T(\bar{b})$ $(B, \bar{b})$ is in $\overline{C_\psi}$. But by the definition of $\delta_{A\bar{a}B\bar{b}}$ this contradicts the fact that $B \models \delta_{A\bar{a}B\bar{b}}(\bar{b})$.
Finally we bring $\delta(\bar{x})$ in disjunctive normal form, and distribute the disjuncts over $T(\bar{x})$. The resulting formula is a disjunction of conjunctive tree queries.

We prove Lemma 1 using n round Ehrenfeucht–Fraïssé games. We recall the basic terminology. For details, see [10]. Let A, B be finite trees and $\bar{a}$ and $\bar{b}$ be sequences of nodes of A and B, respectively, of the same length. The n round Ehrenfeucht–Fraïssé game $G_n(A, \bar{a}, B, \bar{b})$ is played by two players, often called the spoiler and the duplicator, respectively. It is convenient to assume that the spoiler is male and the duplicator is female. In each of the n rounds, the spoiler chooses a node in one of the two structures and the duplicator chooses a node in the other structure. Let the node that is chosen from A in round m be a_m, and the node chosen from B be b_m. Then duplicator wins the game $G_n(A, \bar{a}, B, \bar{b})$ if the mapping which maps each a_m to b_m and each a_i in $\bar{a}$ to b_i in $\bar{b}$ is a partial isomorphism. These games are used to characterize first order logical equivalence of two structures up to some quantifier depth. In particular for A and B ordered binary trees, the duplicator has a winning strategy in the game $G_n(A, \bar{a}, B, \bar{b})$ if and only $A, \bar{a} \equiv_n^{FO^{\mathsf{bin}}} B, \bar{b}$.

Proof of Lemma 1. The direction from (1) to (2) is trivial. Now assume (2). First observe that the mapping which sends a_i to b_i is a partial isomorphism for every n. Thus the statement holds for $n = 0$. For higher n, we use the games. By assumption duplicator has winning strategies for all the "small games" $G_n(A, a_i, a_j B, b_i, b_j)$ whenever $a_i R a_j$ is an edge in $T_A(\bar{a})$. We patch them together to form a winning strategy in the game $G_n(A, \bar{a}, B, \bar{b})$. The result follows by the equivalence of the latter with $A, \bar{a} \equiv_n^{FO^{\mathsf{bin}}} B, \bar{b}$.

Let $a_i <_{\mathsf{left}} a_j \in T(\bar{a})$. To every small subgame $G_n(A, a_i, a_j, B, b_i, b_j)$ we associate the following two corresponding regions in A and B:

$$\{a \in A \mid a_i <_{\mathsf{left}} a \wedge a_j \not< a\} \text{ and } \{b \in A \mid b_i <_{\mathsf{left}} b \wedge b_j \not< b\}.$$

We create similar regions whenever $a_i <_{\mathsf{right}} a_j$.

It follows from the assumption that duplicator has a winning strategy in each of the games $G_n(A, a_i, B, b_i)$ for a_i and b_i either both the root or a_i and b_i both leaves of $T(\bar{a})$ and $T(\bar{b})$, respectively. Associate to the games on leaves a_i and b_i the regions $\{a \in A \mid a_i < a\}$ and $\{b \in A \mid b_i < b\}$. Associate to the game for the roots a_i, b_i, the regions $\{a \in A \mid a_i \not< a\}$ and $\{b \in A \mid b_i \not< b\}$.

Note that these regions partition both structures. Also note that when spoiler chooses an element from a region associated to a small game, duplicators winning

strategy forces her to play in the corresponding region in the other structure. Duplicators strategy in $G_n(A, \bar{a}, B, \bar{b})$ is the following: if spoiler chooses an element x she answers with an element y according to the small subgame associated to the region from which x is picked. Since the regions partition the structures and each region is associated to one game, this is a well defined strategy. Now we show that it is winning for duplicator. Let $\bar{a}'$ and $\bar{b}'$ be the final configuration after n rounds. We must show that this is a partial isomorphism. The restriction of $\bar{a}'$ and $\bar{b}'$ to any of the regions is a partial isomorphism because duplicator played according to her winning strategy associated to that region. Let a_k and a_l be in different regions, and suppose $a_k <_{\mathsf{left}} a_l$. By choice of the regions, there must be a_i, a_j in $\bar{a}$ such that $a_k <_{\mathsf{left}} a_j \leq a_i \leq a_l$ and $b_k <_{\mathsf{left}} b_j$ and $b_i \leq b_l$. We have already shown that for elements in $\bar{a}$, $a_j \leq a_i$ iff $b_j \leq b_i$. Hence $b_j \leq b_i$ holds and by transitivity, $b_k <_{\mathsf{left}} b_l$. The same argument applies to $<_{\mathsf{right}}$ and also starting from B. QED

Lemma 2. *Every FO^{bin} formula $\phi(x, y)$ is equivalent to a FO_3^{bin} formula.*

Proof. Let $\phi(x, y) \in FO^{\mathsf{bin}}$. We prove the lemma by induction on the quantifier depth of ϕ. If the depth is 0, there is nothing to prove. So assume that ϕ is of depth $n + 1$. $\phi(x, y)$ is equivalent to a boolean combination of atoms in x, y and formulas of the form $\exists z \psi(x, y, z)$, with ψ of quantifier depth n. We show that each of the latter can be written using just three variables. As there are finitely many order types generated by $\{x, y, z\}$, $\exists z \psi(x, y, z)$ is equivalent to a disjunction of formulas of the form

$$\exists z \exists \bar{u}(T(\bar{u}, x, y, z) \wedge \psi(x, y, z)). \tag{11}$$

By Corollary 3, $T(\bar{u}, x, y, z) \wedge \psi(x, y, z)$ is equivalent to a disjunction of conjunctive tree queries Q whose edges are labeled by binary FO^{bin} formulas of quantifier depth n. Thus (11) is equivalent to a disjunction of formulas of the form $\exists z \exists \bar{u}(Q)$. This formula itself is a conjunctive tree query in two free variables in which the atoms are binary FO^{bin} formulas. But such formulas are equivalent to formulas in three variables in total over the atoms [4]. All edges in Q are labeled by binary FO^{bin} formulas of quantifier depth n. So by the induction hypothesis, we may assume that they are labeled by FO_3^{bin} formulas. Thus the complete formula can be written in three variables. This proves the lemma.

Now we describe how to reduce the problem from unranked to binary trees. We use the encoding of unranked ordered trees to ordered binary trees as described in [32]. Let $T = (N, \Sigma)$ be a sibling ordered node labeled tree. Let $f : N \longrightarrow \{0, 1\}^*$ be defined as follows

$$\begin{aligned} f(\langle\rangle) &= \langle\rangle \\ f(s \cdot 0) &= f(s) \cdot 0 \\ f(s \cdot k + 1) &= f(s \cdot k) \cdot 1. \end{aligned}$$

The encoding of T, denoted by $\mathsf{enc}(T)$, has tree domain $\{f(n) \mid n \in N\}$ and for each n, $f(n)$ has the same labels as n.

Lemma 3. *1. For each $\phi(x,y) \in FO^{\mathsf{tree}}$, there exists a $\phi^f(x,y) \in FO^{\mathsf{bin}}$ such that for all trees T and nodes a, b, $T \models \phi(a,b)$ iff $\mathsf{enc}(T) \models \phi^f(a,b)$.*
2. For each $\phi(x,y) \in FO_3^{\mathsf{bin}}$ there exists a $\phi^b(x,y) \in FO_3^{\mathsf{tree}}$ such that for all trees T and nodes a, b, $\mathsf{enc}(T) \models \phi(a,b)$ iff $T \models \phi^b(a,b)$.

Proof. We define translations $(\cdot)^f$ and $(\cdot)^b$ by induction on the structure of the formulas:

$$\begin{array}{ll}(\sigma(x))^f & = \sigma(x)\\ (xR_{\Downarrow}y)^f & = x <_{\mathsf{left}} y\\ (xR_{\Rightarrow}y)^f & = x <_{\mathsf{right}} y \wedge \neg\exists z(x <_{\mathsf{right}} z \wedge z <_{\mathsf{left}} y)\\ (\cdot)^f \text{ commutes with the logical operators}\end{array}$$

$$\begin{array}{ll}(\sigma(x))^b & = \sigma(x)\\ (x <_{\mathsf{left}} y)^b & = xR_{\Downarrow}y\\ (x <_{\mathsf{right}} y)^b & = xR_{\Rightarrow}y \vee \exists z(xR_{\Rightarrow}z \wedge zR_{\Downarrow}y)\\ (\cdot)^f \text{ commutes with the logical operators}\end{array}$$

The variable z used in the translation needs to be different from x and y. Note that it is bound by the quantifier, so the translation backward uses at most three variables. By induction the translations can be shown correct.

Proof of Theorem 2. By Lemmas 3 and 2. QED

5 Closure under complementation

In this section we prove the remaining part of the proof of Theorem 1, that is

Theorem 3. *Conditional XPath is closed under complementation of path expressions.*

Recall that in order to prove this theorem, we must find, given an arbitrary Conditional XPath path wff R, a Conditional XPath path wff R' which is equivalent to the complement of R. The proof is divided into a number of lemmas. We give a brief outline. First we establish a syntactic normal form for path wffs. Every path wff is shown equivalent to a union of *separated basic compositions*. These are path wffs without $\cup$ whose syntactic form guarantees that they are subrelations of one of

$$?\top, \quad \mathsf{child}^+, \quad \mathsf{parent}^+, \quad \mathsf{parent}^*/\mathsf{right}^+/\mathsf{child}^*, \quad \mathsf{parent}^*/\mathsf{left}^+/\mathsf{child}^*. \qquad (12)$$

These five relations correspond to XPath's axis relations

$$\texttt{self}, \ \texttt{descendant}, \ \texttt{ancestor}, \ \texttt{following}, \ \texttt{preceding}.$$

As is well known, given a tree and a node n, the answer sets of these relations evaluated at n form a partition of the tree [46].

In subsection 5.2 we use this normal form to show that path wffs are closed under intersections. Because the relations in (12) partition a tree, the problem

reduces to showing that separated basic compositions of the same form are closed under intersections. Hence, showing closure under complementation reduces to showing that separated basic compositions are closed under complementation.

In subsection 5.4 we further reduce the problem: it is sufficient to find, for step either child or right, path wffs equivalent to $\mathsf{step}^+ \cap \overline{R}$, where R is a separated basic composition which is a subrelation of step^+. By this we have reduced the problem of reasoning on trees to reasoning on strings. For instance, to define when $\mathsf{child}^+ \cap \overline{\mathsf{child}^+/?A/(\mathsf{child}?B)^+}$ holds between nodes n and n', we only need to reason about the nodes in between n and n'. This is done in the last subsection 5.5.

5.1 Preparing the input

In the first two lemmas, R is brought into a shape which is easier to handle. We need a bit of terminology. An *atom* is a path wff of the form $\mathsf{step}?A$, or $(\mathsf{step}?B)^+?A$. A *test* is a path wff of the form $?A$. A *basic composition* is a test followed by a non empty sequence of atoms separated by /'s.

Lemma 4. *Every path wff is equivalent to a union of basic compositions and a test.*

Proof. The lemma is shown by distributing unions over /, combining tests using the equivalences $?A/?B \equiv ?(A \wedge B)$ and $?A \cup ?B \equiv ?(A \vee B)$, and replacing $(\mathsf{step}/?A)^*$ by $?\top \cup (\mathsf{step}/?A)^+$. As basic compositions must have a leading test, one may need to add a dummy test using $R \equiv ?\top/R$.

In the following example, we first use distribution and then combine tests.

$$?C_1/((\mathsf{child}?B)^+?A_1 \ \cup \ ?C_2/\mathsf{parent}?A_2)/?C_3/\mathsf{right}?A_4 \quad (13)$$

$$?C_1/(\mathsf{child}?B)^+?A_1/?C_3/\mathsf{right}?A_4 \ \bigcup \ ?C_1/?C_2/\mathsf{parent}?A_2/?C_3/\mathsf{right}?A_4 \quad (14)$$

$$?C_1/(\mathsf{child}?B)^+?(A_1 \wedge C_3)/\mathsf{right}?A_4 \ \bigcup \ ?(C_1 \wedge C_2)/\mathsf{parent}?(A_2 \wedge C_3)/\mathsf{right}?A_4 \quad (15)$$

We need a bit more terminology. We call an atom *down* if it is of the form $\mathsf{child}?A$, or $(\mathsf{child}?B)^+?A$. Analogously, we define atoms being *up, right*, and *left*. A path wff *has form* T if it is a test. It has form D, U, R, L if it is a basic composition of down, up, right or left atoms, respectively. We say that a basic composition is *separated* if it has one of the following forms:

$$D, \quad U, \quad U^*/R/D^*, \quad U^*/L/D^*. \tag{16}$$

Here we use $U^*/R/D^*$ as an abbreviation for the forms U/R, R, R/D, $U/R/D$, and similarly for $U^*/L/D^*$.

The formula (15) is a union of path wffs of the form D/R and U/R. Thus the second disjunct is separated, but the first is not. But the first is easily separable,

namely

$$
\begin{array}{ll}
?C_1 \,/\, (\mathsf{child}?B)^+?(A_1 \wedge C_3) \,/\, \mathsf{right}?A_4 & \equiv \\
?C_1 \,/\, (\mathsf{child}?B)^* \,/\, \mathsf{child}?(A_4 \wedge \langle \mathsf{left}?(B \wedge A_1 \wedge C_3)\rangle) & \equiv \\
?C_1 \,/\, \mathsf{child}?(A_4 \wedge \langle \mathsf{left}?(B \wedge A_1 \wedge C_3)\rangle) \bigcup & \\
\quad ?C_1 \,/\, (\mathsf{child}?B)^+?\top \,/\, \mathsf{child}?(A_4 \wedge \langle \mathsf{left}?(B \wedge A_1 \wedge C_3)\rangle) &
\end{array}
$$

Note that the result is a union of formulas of the form D. The occurrence of the left axis inside the test is irrelevant for the form. The form is only determined by the axis *not* occurring in the tests.

Lemma 5. *Every path wff is equivalent to a union of tests and separated basic compositions.*

Proof. By Lemma 4 it is enough to describe a procedure that separates basic compositions. Table 3 shows how every composition of *two* atoms can be separated. The result follows by an induction on the number of /'s in basic compositions using the equivalences in Table 3 and distribution of / over $\cup$. The proofs for the equivalences in the table are semantic arguments based on elaborate case distinctions. They are similar to the example given below.

A having form	is separated as a union of forms
D/U	$D, T,$ or U
D/R	D
D/L	D
U/D	$D, T, U, U^*/R/D^*,$ or $U^*/L/D^*$
U/R	U/R
U/L	U/L
R/D	R/D
R/U	U
R/L	$T, L,$ or R
L/D	L/D
L/U	U
L/R	$T, L,$ or R

Table 3. Syntactical separation for compositions of two atoms.

Above we saw an example of the form D/R which reduced to a union of compositions of form D. A representative example of all other cases is

$$(\mathsf{right}?A_1)^+?B_1 \,/\, (\mathsf{left}?A_2)^+?B_2.$$

Suppose, for nodes a, b in some tree, $a\,(\mathsf{right}?A_1)^+?B_1/(\mathsf{left}?A_2)^+?B_2\,b$ holds. Then there is a node c such that $a\,(\mathsf{right}?A_1)^+?B_1\,c$ and $c\,(\mathsf{left}?A_2)^+?B_2\,b$. It follows that $a\,\mathsf{right}^+\,c$ and $b\,\mathsf{right}^+\,c$. There are three cases, depending on the

relation between a and b, depicted in Figure 2. The corresponding path wffs for each case are given in Table 4. Here E is an abbreviation of the node wff

$$\langle(\mathsf{right}?(A_1 \wedge A_2))^*/\mathsf{right}?(B_1 \wedge A_1)\rangle.$$

Thus the equivalent wff is a union of wffs of the form T, L and R.

case	equivalent path wff
$a\ \mathsf{right}^+\ b$	$(\mathsf{right}?A_1)^+?(A_2 \wedge B_2 \wedge E)$
$a = b$	$?(A_2 \wedge B_2 \wedge E)$
$b\ \mathsf{right}^+\ a$	$?(A_2 \wedge E)/(\mathsf{left}?A_2)^+?B_2.$

Table 4. Separating $(\mathsf{right}?A_1)^+?B_1/(\mathsf{left}?A_2)^+?B_2$.

The following lemma is useful to reduce the number of cases. For R a path wff, define the *converse* of R, denoted by R^{-1}, with meaning

$$[\![R^{-1}]\!]_{\mathfrak{M}} = \{(n', n) \mid (n, n') \in [\![R]\!]_{\mathfrak{M}}\}.$$

Lemma 6. *(i) The path wffs of Conditional XPath are closed under taking converses.*
(ii) The converse of a basic composition consisting of up (left) atoms is equivalent to a union of basic compositions consisting of down (right) atoms.

Proof. (i) Let R be a path wff. By replacing $(\mathsf{step}/?A)^*$ by $?\top \cup (\mathsf{step}/?A)^+$ we may assume that R contains no Kleene stars. R^{-1} is defined inductively using the following equivalences:

$$\begin{array}{llll} \mathsf{child}^{-1} & = \mathsf{parent} & (?A)^{-1} & = ?A \\ \mathsf{parent}^{-1} & = \mathsf{child} & (R/S)^{-1} & = S^{-1}/R^{-1} \\ \mathsf{right}^{-1} & = \mathsf{left} & (R \cup S)^{-1} & = R^{-1} \cup S^{-1} \\ \mathsf{left}^{-1} & = \mathsf{right} & ((\mathsf{step}/?A)^+)^{-1} & = ?A/(\mathsf{step}^{-1}/?A)^*/\mathsf{step}^{-1}. \end{array}$$

(ii) follows from the above equivalences since

$$\begin{aligned} ((\mathsf{step}/?A)^+)^{-1} &\equiv ?A/(\mathsf{step}^{-1}/?A)^*/\mathsf{step}^{-1} \\ &\equiv ?A/\mathsf{step}^{-1}?\top \ \bigcup\ ?A/(\mathsf{step}^{-1}/?A)^+?\top/\mathsf{step}^{-1}?\top. \end{aligned}$$

5.2 Closure under intersection

We prove that Conditional XPath path wffs are closed under intersection and that certain positive existential formulas are expressible. We first define the latter. An *in-between down query* is a formula of the form $\exists \bar{z} B(\bar{z}, \bar{w}, x, y)$ with

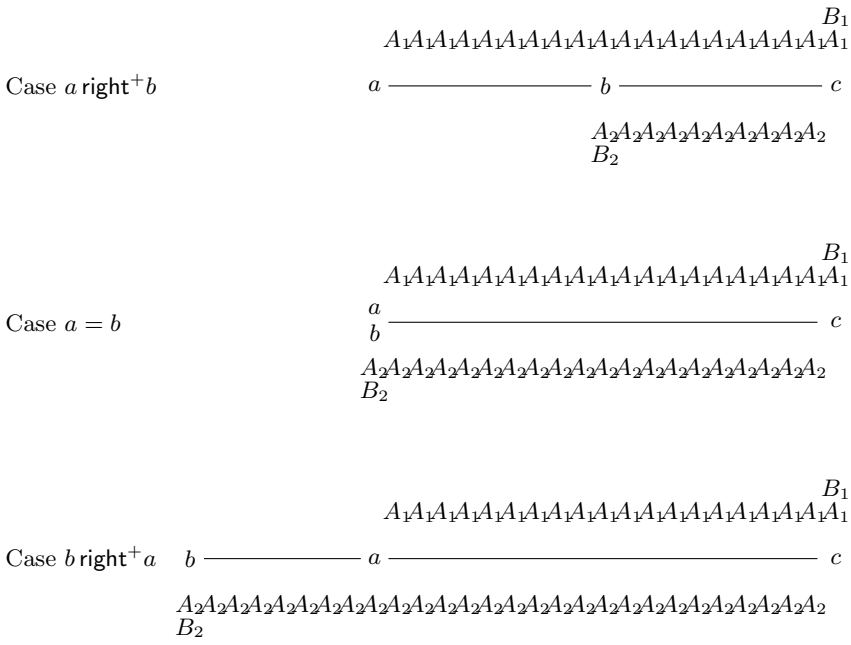

Fig. 2. Three cases for the sibling relation between a and b.

$B(\bar{z}, \bar{w}, x, y)$ a formula generated from down atoms of the form uRv and a test $x?Ax$ using disjunction and conjunction. Moreover $B(\bar{z}, \bar{w}, x, y)$ implies $x\mathtt{child}^+y$ and for all other free variables u in B, $x\mathtt{child}^+u$ and $u\mathtt{child}^+y$. In-between right queries are defined similarly by substituting down and $\mathtt{child}$ by right.

An example of an in-between down query is $\exists z(x\mathsf{child}/?Az \wedge z(\mathsf{child}?B)^+/?Cy)$. The meaning of the atoms is defined by $\mathfrak{M} \models aRb$ iff $(a, b) \in [\![R]\!]_{\mathfrak{M}}$.

Lemma 7. *Every in-between down (right) query in free variables x, y is equivalent to a union of Conditional XPath path wffs of the down (right) form.*

Proof. We prove the lemma for down queries. The argument for right is identical. Let $Q(x, y)$ be such a query. Then it is equivalent to a disjunction of queries of the form $\exists z_1, \ldots, z_k B$, with B a conjunction. Because all variables z_i are between x and y and $x\,\mathsf{child}^+\,y$ holds, these variables can be totally ordered. Let $TO(x, y, z_1, \ldots, z_k)$ be a conjunction of formulas $u\mathsf{child}^+v$ and $u = v$ which specifies such a total order. Then the formula $\exists z_1, \ldots, z_k B$ is equivalent to the finite disjunction of all formulas $\exists z_1, \ldots, z_k(TO(x, y, z_1, \ldots, z_k) \wedge B)$. Without loss we may assume that all variables are different (otherwise substitute them away). Rewrite the query using the equivalences in Table 5 into the form

$$\exists v_1, \ldots, v_k(x?Ax \wedge x\ Atom_0\ v_1 \wedge\ v_1\ Atom_1\ v_2 \wedge \ldots \wedge v_n\ Atom_n\ y).$$

which is equivalent to $?A/Atom_0/Atom_1/\ldots/Atom_n$. The rewriting goes as follows: first use the equivalences of the first group to obtain a conjunction consisting only of atoms $u\ Atom_0\ v$ such that for no w, $u\mathtt{child}^+w\mathtt{child}^+v$ holds. Then apply the rules from the second and the third group to obtain one atom for each such pair u, v.

$u\mathsf{child}^+v \wedge u\mathsf{child}^+z \wedge z\mathsf{child}^+v$	$\equiv u\mathsf{child}^+z \wedge z\mathsf{child}^+v$
$u\mathsf{child}?Av \wedge u\mathsf{child}^+z \wedge z\mathsf{child}^+v$	$\equiv \bot$
$u(\mathsf{child}?A)^+?Bv \wedge u\mathsf{child}^+z \wedge z\mathsf{child}^+v$	$\equiv u(\mathsf{child}?A)^+z \wedge z(\mathsf{child}?A)^+?Bv$
$u\mathsf{child}?Av \wedge u\mathsf{child}?Bv$	$\equiv u\mathsf{child}?(A \wedge B)v$
$u\mathsf{child}?Av \wedge u(\mathsf{child}?B)^+?Cv$	$\equiv u\mathsf{child}?(A \wedge B \wedge C)v$
$u(\mathsf{child}?A_1)^+?B_1v \wedge u(\mathsf{child}?A_2)^+?B_2v$	$\equiv u(\mathsf{child}?(A_1 \wedge A_2)^+?(B_1 \wedge B_2)v$
$u\mathsf{child}?Av \wedge v\mathsf{child}^+u$	$\equiv \bot$
$u(\mathsf{child}?A)^+?Bv \wedge v\mathsf{child}^+u$	$\equiv \bot$.

Table 5. Equivalences for rewriting conjunctive queries.

Lemma 8. *Conditional XPath path wffs are closed under intersection.*

Proof. Let R, S be Conditional XPath path wffs. We must find a path wff T such that for each tree $\mathfrak{M}$, $[\![R]\!]_{\mathfrak{M}} \cap [\![S]\!]_{\mathfrak{M}} = [\![T]\!]_{\mathfrak{M}}$. By Lemma 5, $R \equiv \bigcup_i R_i$ and $S \equiv \bigcup_j S_j$, and the R_i and S_j are separated basic compositions or tests. Thus $R \cap S \equiv \bigcup_{ij}(R_i \cap S_j)$. Because the five forms partition trees, $R_i \cap S_j \equiv \bot$ whenever they are of a different form. Thus we need only to define $R \cap S$ for R, S of the same form. Suppose they are both of the form U/L. Concretely, let R_1 and S_1 be compositions of up atoms, and R_2 and S_2 compositions of left atoms. Because the models are trees, we have

$$R_1/R_2 \ \cap \ S_1/S_2 \ \equiv \ (R_1 \cap S_1) \ / \ (R_2 \cap S_2).$$

Similar results hold for all other forms. Thus we need only define $R \cap S$ for R, S either both basic compositions of the same form D, U, L, R or both tests. The intersection of two tests $?A$ and $?B$ is simply $?A/?B$. By Lemma 6, the cases U and L reduce to the cases D and R, respectively. We give the argument for U. The one for L is the same. Let U_i and D_i denote basic compositions of the U and the D form, respectively. Then

$$\begin{array}{ll} (U_1 \cap U_2) & \equiv \\ ((U_1 \cap U_2)^{-1})^{-1} & \equiv \\ (U_1^{-1} \cap U_2^{-1})^{-1} & \equiv \\ ((D_1 \cup \ldots \cup D_n) \cap (D_{n+1} \cup \ldots \cup D_{n+k}))^{-1} & \equiv \\ (D_1 \cap D_{n+1})^{-1} \cup \ldots \cup (D_n \cap D_{n+k})^{-1}. & \end{array}$$

Now let $R = ?A/R_1/\ldots/R_n$ and $S = ?B/S_1/\ldots/S_m$ with the R_i, S_i either all down or all right atoms. R and S are equivalent to the following in-between queries

$$R \equiv \exists v_1 \ldots v_n (x?Ax \wedge xR_1v_1 \wedge \ldots \wedge v_nR_ny) \tag{17}$$

$$S \equiv \exists w_1 \ldots w_m (x?Bx \wedge xS_1w_1 \wedge \ldots \wedge w_mS_my). \tag{18}$$

Thus $R \cap S$ is equivalent to the in-between query

$$\exists v_1 \ldots v_n \exists w_1 \ldots w_m (x?(A \wedge B)x \wedge xR_1v_1 \wedge xS_1w_1 \wedge \ldots \wedge v_nR_ny \wedge w_mS_my). \tag{19}$$

(19) is equivalent to a Conditional XPath path wff by Lemma 7.

5.3 Proving Theorem 3

Lemmas 5 and 8 reduce the task of proving Theorem 3 to showing

Lemma 9. *The complement of each separated basic composition is equivalent to a Conditional XPath path wff.*

The proof of Lemma 9 consists of an easy and a hard part, separated in Lemma 10 and 11, which are shown below.

PROOF OF THEOREM 3. Let R be a path wff. Then by Lemma 5 $R \equiv \bigcup_i R_i$, with the R_i tests and separated basic compositions. Thus $\overline{R} \equiv \bigcap_i \overline{R_i}$. By Lemma 8, the path wffs are closed under intersection. The complement of a test $?A$ is equivalent to $?\neg A \cup \mathsf{not\_equal}$ with the latter abbreviating

$$\mathsf{child}^+ \cup \mathsf{parent}^+ \cup \mathsf{parent}^*/\mathsf{left}^+/\mathsf{child}^* \cup \mathsf{parent}^*/\mathsf{right}^+/\mathsf{child}^*.$$

By Lemma 9 each complement of a separated basic composition is equivalent to a path wff. Hence the theorem. QED

5.4 From trees to strings

We rewrote the path wffs into separated basic compositions because it helps to reduce the reasoning to "lines" or "strings". For example, consider a path wff S of the form U. By the syntactic form of S it holds that whenever aSb holds in a tree, then a is below b. Thus if in a tree $a\,\overline{S}\,b$ holds, we can break into two cases:

- a is not below b;
- a is below b, but not $a\,S\,b$.

As an example let $S = \mathsf{parent}/?A$. Then the second case corresponds to a is below b, but b is not the parent of a or A is false at b.

The first case is easy to express (using the partition again). The second is harder but, as S is a composition of up atoms, *we only need to reason about the elements in between and including a and b*. That is, we need to reason about a line segment. But not all separated path wffs are of this simple form, consisting of one direction. The next lemma however states that complements of these can be defined using complements of the uni-directed forms.

Lemma 10. *Let D and R denote basic compositions of the forms D and R respectively. If the formulas*

$$(\mathsf{child}^+ \cap \overline{D}) \text{ and } (\mathsf{right}^+ \cap \overline{R}) \tag{20}$$

are equivalent to path wffs, then the complement of each separated basic composition is equivalent to a path wff.

Proof. First we show the lemma assuming that formulas of the form

$$(\mathsf{child}^+ \cap \overline{D}),\ (\mathsf{parent}^+ \cap \overline{U}),\ (\mathsf{right}^+ \cap \overline{R}),\ \text{and}\ (\mathsf{left}^+ \cap \overline{L})$$

are equivalent to path wffs. As an example, consider a basic composition of the form U/R. The other forms are handled using the same argument. Then

$$\overline{U/R} \equiv (\overline{\mathsf{parent}^+/\mathsf{right}^+} \cap \overline{U/R}) \ \cup\ (\mathsf{parent}^+/\mathsf{right}^+ \cap \overline{U/R}). \tag{21}$$

By the syntactic form of U and R, each relation of the form U/R is contained in the relation $\mathsf{parent}^+/\mathsf{right}^+$. Thus the first disjunct is equivalent to $\overline{\mathsf{parent}^+/\mathsf{right}^+}$, which is equivalent to

$$\mathsf{child}^* \cup \mathsf{parent}^*/\mathsf{left}^+/\mathsf{child}^* \cup \mathsf{parent}^+ \cup \mathsf{right}^+/\mathsf{child}^* \cup \mathsf{parent}^+/\mathsf{right}^+/\mathsf{child}^+.$$

For the second disjunct, we use the following equation:

$$\mathsf{parent}^+/\mathsf{right}^+ \cap\ \overline{U/R} \equiv (\mathsf{parent}^+ \cap \overline{U})/\mathsf{right}^+\ \cup\ \mathsf{parent}^+/(\mathsf{right}^+ \cap \overline{R}). \quad (22)$$

Rewriting (22) into first order logic makes it easier to see the equivalence:

$$\exists z(x\mathsf{parent}^+ z \wedge z\mathsf{right}^+ y) \wedge \forall z(x\overline{U}z \vee z\overline{R}y)$$
$$\equiv$$
$$\exists z(x\mathsf{parent}^+ z \wedge z\mathsf{right}^+ y \wedge x\overline{U}z)\ \vee\ \exists z(x\mathsf{parent}^+ z \wedge z\mathsf{right}^+ y \wedge z\overline{R}y).$$

The up to down direction is a validity for all relations. The other direction is not, but it holds because the models are trees. To see it, suppose that the lower side is true and the upper side fails in some model. Then $xUa \wedge aRy$ holds, for some node a. By the syntactic form of U and R, we obtain that $x\,\mathsf{parent}^+\,a$ and $a\,\mathsf{right}^+\,y$. In a tree, given nodes x and y, there is exactly one element a such that $x\,\mathsf{parent}^+\,a$ and $a\,\mathsf{right}^+\,y$ both hold. But that means that —given the lower side— either $x\,\overline{U}\,a$ or $a\,\overline{R}\,y$ holds, a contradiction.

We now use the fact that Conditional XPath is closed under conversion (denoted by R^{-1}) and intersection to reduce the number of cases to two. Consider $\mathsf{parent}^+ \cap \overline{U}$. The converse of a formula of the form U is equivalent to a union of formulas of the form D by Lemma 6(ii). Thus we have the following equivalences:

$$\begin{array}{ll}
\mathsf{parent}^+ \cap \overline{U} & \equiv \\
((\mathsf{parent}^+ \cap \overline{U})^{-1})^{-1} & \equiv \\
((\mathsf{parent}^+)^{-1} \cap \overline{U^{-1}})^{-1} & \equiv \\
(\mathsf{child}^+ \cap \overline{(D_1 \cup \ldots \cup D_n)})^{-1} & \equiv \\
(\mathsf{child}^+ \cap \overline{(D_1)})^{-1} \cap \ldots \cap (\mathsf{child}^+ \cap \overline{(D_n)})^{-1}. &
\end{array}$$

We can similarly relate the L and R forms. By Lemma 6 path wffs are closed under $(\cdot)^{-1}$ and by Lemma 8 under $\cap$. Thus $\mathsf{parent}^+ \cap \overline{U}$ and $\mathsf{left}^+ \cap \overline{L}$ are equivalent to path wffs if $\mathsf{child}^+ \cap \overline{D}$ and $\mathsf{right}^+ \cap \overline{R}$ are.

All the preparation has been done, we can start the real work. We just have to define the relations in (20) as Conditional XPath path wffs.

5.5 The real work

To reduce the number of cases, we use a notion well known from temporal logic. For A, B node wffs, define the path wff $\mathsf{until}(A, B)$ with the semantics

$$x\,\mathsf{until}(A, B)\,y \iff xR_{\Downarrow}y \wedge A(y) \wedge \forall z(x\,R_{\Downarrow}\,z\,R_{\Downarrow}\,y \rightarrow B(z)).$$

Please note that $\mathsf{until}(A, B)$ is a path wff, and denotes a set of pairs, unlike its use in temporal logic. Temporal logic is a *one-sorted* formalism, containing only node wffs. The until formula from temporal logic, denoting a set of points is of course expressed in our formalism as $\langle\mathsf{until}(A, B)\rangle$.

Both down atoms are expressible as an until formula: $\mathsf{child}?A \equiv \mathsf{until}(A, \neg\top)$ and $(\mathsf{child}?B)^+?A \equiv \mathsf{until}(A \wedge B, B)$. In order to increase readability we use $<$ and $\leq$ instead of child^+ and child^*, respectively.

We can similarly define $\mathsf{until}_\Rightarrow$ path wffs using the $R_\Downarrow$ relation, and use it to define the right atoms. But introducing yet another notation is not needed because we prove all results using only the fact that $R_\Downarrow$ is a partial order which is linear toward the past. $R_\Downarrow$ is a total order, thus has the same properties.

Thus it is sufficient to show how to define $\mathsf{child}^+ \cap \overline{?C/R}$, for R a non empty sequence of until formulas separated by $/$, and C an arbitrary test. We call such formulas *until wffs*.

Lemma 11. *Let D and R denote basic compositions of the forms D and R respectively. Each of the formulas $(\mathsf{child}^+ \cap \overline{D})$ and $(\mathsf{right}^+ \cap \overline{R})$ is equivalent to a Conditional XPath path wff.*

Proof. As explained in the introduction to this subsection it is sufficient to show the lemma for formulas of the form $\mathsf{child}^+ \cap \overline{R}$ for R an untill wff.

We define complementation by a case distinction. The first case is when there is only one atom:

$$< \cap \overline{?C/\mathsf{until}(A, B)} \;\equiv\; ?\neg C/< \;\cup\; ?C/</?\neg A \;\cup\; ?C/</?\neg B/</?A. \quad (23)$$

For the case with more atoms we make a further case distinction. Let $R = S/\mathsf{until}(A, B)$, with S an until wff. Then

$$< \cap \overline{R} \equiv (\overline{S/<} \cap < \cap \overline{R}) \cup (S/< \cap < \cap \overline{R}). \quad (24)$$

As $[\![\overline{S/<}]\!]_{\mathfrak{M}} \subseteq [\![\overline{S/\mathsf{until}(A, B)}]\!]_{\mathfrak{M}}$, for each tree $\mathfrak{M}$, the first disjunct is simply equivalent to $< \cap \overline{S/<}$. Lemma 14 shows how to define $< \cap \overline{S/<}$.

Now we explain how to define $S/< \cap < \cap \overline{R}$, the second disjunct in (24). For S an until wff, define $\mathsf{max}(S, x, z, y)$ as the ternary relation

$$x < z < y \wedge xSz \wedge \neg\exists w(z < w < y \wedge xSw).$$

$\mathsf{max}(S, x, z, y)$ expresses that (x, z) is the largest proper subinterval in (x, y) which is in S. We claim that $x(S/< \cap < \cap \overline{R})y$ is equivalent to $\exists z(\mathsf{max}(S, x, z, y) \wedge z\,(< \cap \overline{\mathsf{until}(A, B)})\,y)$. From left to right is obvious. For the other direction, Suppose $\exists z(\mathsf{max}(S, x, z, y) \wedge z\,(< \cap \overline{\mathsf{until}(A, B)})\,y)$. Then $x(S/< \cap <)y$. Let z be the *last* between x and y such that xSz and $z\,(< \cap \overline{\mathsf{until}(A, B)})\,y$. Now suppose to the contrary that there is a z such that xSz and $z\mathsf{until}(A, B)y$. But then $z \leq z' < y$ and thus $z'\mathsf{until}(A, B)y$, a contradiction.

To summarize, for $R = S/\mathsf{until}(A, B)$, the expression $< \cap \overline{R}$ is equivalent to the union of $< \cap \overline{S/<}$ and a formula expressing $\exists z(\mathsf{max}(S, x, z, y) \wedge z\,(< \cap \overline{\mathsf{until}(A, B)})\,y)$. Lemma 14 below defines $< \cap \overline{S/<}$ as a path wff. Lemma 13 below defines $\mathsf{max}(S, x, z, y)$ as an in-between down query. (23) defines $z\,(< \cap \overline{\mathsf{until}(A, B)})\,y$ as a disjunction of down atoms. But then $\exists z(\mathsf{max}(S, x, z, y) \wedge z\,(< \cap \overline{\mathsf{until}(A, B)})\,y)$ is equivalent to an in-between down query in x, y, and thus by Lemma 7 equivalent to a path wff.

In defining both $< \cap \overline{S/<}$ and the max predicate we use a crucial lemma, which we prove first.

For R a path wff, let $\mathsf{range}(R)$ be the node wff which is true at a point x iff there exists a point y such that yRx holds. These node wffs are definable in Conditional XPath, using conversion:

$$\mathsf{range}(R) \equiv \langle R^{-1} \rangle.$$

The statement in the next lemma is graphically represented in Figure 3.

Lemma 12. *Let R be an until wff. For all points x, y, a, b, such that $x < a \leq y \leq b$, if xRa and xRb and $\mathsf{range}(R)y$, then also xRy.*

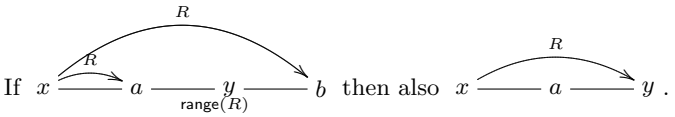

Fig. 3. Lemma 12 in a picture.

Proof. By induction on the number of /'s in R. First let $R = ?C/\mathsf{until}(A, B)$. Then Ay, because $\mathsf{range}(R)y$. As $x < y \leq b$ and xRb, C is true at x and B at all points in between x and b, hence a fortiori between x and y. Thus $x\ ?C/\mathsf{until}(A, B)\ y$ holds.

For the inductive step, let S be an until wff and let $R = S/\mathsf{until}(A, B)$. We obtain a', b', y' such that

- $y' < y$ and $y'\mathsf{until}(A, B)y$ and $\mathsf{range}(S)y'$,
- $x < a' < a$ and xSa' and $a'\mathsf{until}(A, B)a$, and
- $x < b' < b$ and xSb' and $b'\mathsf{until}(A, B)b$.

The points a' and b' are both between x and b. Thus either $b' < a'$ or $a' \leq b'$. If $b' < a'$, then from $a' < y \leq b$, $A(y)$ and $b'\mathsf{until}(A, B)b$ we obtain $a'\mathsf{until}(A, B)y$. This together with xSa' yields $xS/\mathsf{until}(A, B)y$. Now consider $a' \leq b'$. As $<$ denotes the descendant relation, and is linear toward the past, there are three cases for the position of y' relative to a' and b': (1) $y' < a'$, (2) $a' \leq y' \leq b'$, and (3) $b' < y'$.

If $y' < a'$, then from $y'\mathsf{until}(A, B)y$, we obtain $a'\mathsf{until}(A, B)y$, so with xSa', we get $x\ S/\mathsf{until}(A, B)\ y$.

If $a' \leq y' \leq b'$, by inductive hypothesis we get xSy', so with $y'\mathsf{until}(A, B)y$ we obtain $x\ S/\mathsf{until}(A, B)\ y$.

Let $b' < y'$. If $y = b$, we are done. Otherwise $b' < y < b$. By $y'\mathsf{until}(A, B)y$, A holds at y. As $b'\mathsf{until}(A, B)b$ (and $b' < y < b$), we thus have $b'\mathsf{until}(A, B)y$. Together with xSb' this yields $x\ S/\mathsf{until}(A, B)\ y$.

Recall that an in-between down query is a positive existential formula generated from down atoms. Each formula $u\,\mathsf{until}(A,B)\,v$ is equivalent to the formula $u\mathsf{child}/?Av \vee \exists z(u(\mathsf{child}?B)^+/?Bz \wedge z\mathsf{child}?Av)$. Thus a positive existential formula consisting of until atoms is equivalent to an in-between down query.

Lemma 13. *For R an until wff, $\mathsf{max}(R,x,z,y)$ is definable as an in-between down query.*

Proof. We define $\mathsf{max}(R,x,z,y)$ by induction on the number of /'s in R. If $R = ?C/\mathsf{until}(A,B)$, then

$$\begin{aligned}\mathsf{max}(R,x,z,y) \equiv\ & x < z < y & \wedge\\ & x\,?C/\mathsf{until}(A,B)\,z & \wedge\\ & (z\,?\neg B\,z \vee z(< \cap \overline{\mathsf{until}(A,B)/<})y).\end{aligned}$$

The third conjunct forbids that xRw holds for any w between z and y. The expression $< \cap \overline{\mathsf{until}(A,B)/<}$ is defined as a path wff as follows:

$$\begin{aligned}< \cap \overline{\mathsf{until}(A,B)/<} \equiv\ & \mathsf{until}(\top,\bot) \ \cup\\ & \mathsf{until}(\top,\neg A) \ \cup\\ & \mathsf{until}(\neg B \wedge \neg A, \neg A)/\leq.\end{aligned} \tag{25}$$

To explain (25), suppose x and y stand in this relation. Then the first disjunct states that y is a direct successor of x; the second disjunct that there are no A nodes in between x and y and the third that before any A node between x and y there is already a node which is not B.

For the inductive case, let $R = S/\mathsf{until}(A,B)$. Suppose $\mathsf{max}(S,x,w,y)$ holds. To define $\mathsf{max}(R,x,z,y)$ one must consider three cases: $w < z$, $w = z$ and $z < w$. The first case is simple:

$$\exists w(\mathsf{max}(S,x,w,y) \wedge \mathsf{max}(\mathsf{until}(A,B),w,z,y)).$$

The second case is also straightforward:

$$xRz \wedge \mathsf{max}(S,x,z,y) \wedge z(< \cap \overline{\mathsf{until}(A,B)/<})y.$$

In the last case, the largest subinterval in $S/\mathsf{until}(A,B)$ is smaller than the largest subinterval in S. Thus $\mathsf{max}(S,x,w,y)$ and $w(< \cap \overline{\mathsf{until}(A,B)/<})y$ must hold. To make (x,z) the largest subinterval of (x,y) in R we should force xRz and for all z' such that $z < z' \leq w$, $x\overline{R}z'$. For this, we use Lemma 12. The final formula for the third case is then

$$\begin{aligned}\exists w(x < z < w < y \wedge \mathsf{max}(S,x,w,y) \wedge w(< \cap \overline{\mathsf{until}(A,B)/<})y\\ \wedge xRz \wedge z\mathsf{until}(\neg\mathsf{range}(R), \neg\mathsf{range}(R))w).\end{aligned}$$

Thus $\mathsf{max}(S/\mathsf{until}(A,B),x,z,y)$ is defined as the disjunction of these three cases. By the inductive hypothesis, $\mathsf{max}(S,x,w,y)$ is equivalent to an in-between query. $\mathsf{max}(\mathsf{until}(A,B),w,z,y)$ is already defined as an in-between query and (25) defines $z(< \cap \overline{\mathsf{until}(A,B)/<})y$. Hence each of the cases is equivalent to an in-between query.

Lemma 14. *For R an until wff, $< \cap \overline{R/<}$ is definable as a Conditional XPath path wff.*

Proof. We prove by induction on the number of /'s in R that $< \cap \overline{R/<}$ is expressible as an in-between down query in two free variables. Then the result follows by Lemma 7. The base case is (25). Thus let $R = S/\mathsf{until}(A, B)$. Then $< \cap \overline{R/<}$ is equivalent to

$$(< \cap S/< \cap \overline{R/<}) \cup (< \cap \overline{S/<} \cap \overline{R/<}).$$

The second disjunct is equivalent to $< \cap \overline{S/<}$ because for each tree $\mathfrak{M}$, $[\![R/<]\!]_{\mathfrak{M}} = [\![S/\mathsf{until}(A, B)/<]\!]_{\mathfrak{M}} \subseteq [\![S/</<]\!]_{\mathfrak{M}} \subseteq [\![S/<]\!]_{\mathfrak{M}}$. As S is shorter than R, this is definable by the inductive hypothesis.

We define $x(< \cap S/< \cap \overline{S/\mathsf{until}(A, B)/<})y$ as an in-between down query. The formula uses two existentially quantified variables z, z', which are ordered like $x < z \leq z' < y$. The interval (x, z) is the smallest subinterval of (x, y) which is in S. This is expressed by $xSz \wedge x(< \cap \overline{S/<})z$. The interval (x, z') is the largest S subinterval, which we express using the max predicate. Now we must ensure that $x(\overline{S/\mathsf{until}(A, B)/<})y$ holds. So we must say that $z'(\overline{\mathsf{until}(A, B)/<})y$, and that starting at z there is no $\mathsf{until}(A, B)$ interval. Moreover, there should not be a z'' in between z and z' such that xRz''. This is done by saying that either $z = z'$ or $z\mathsf{until}(\neg\mathsf{range}(R), \neg\mathsf{range}(R))z'$. So the final formula becomes

$$\begin{aligned}
\exists z z'(\; & x < z \leq z' < y && \wedge \\
& xSz \wedge x(< \cap \overline{S/<})z && \wedge \\
& \mathsf{max}(S, x, z', y) && \wedge \\
& z'(< \cap \overline{\mathsf{until}(A, B)/<})y \wedge \\
& (z = z' \vee z\mathsf{until}(\neg\mathsf{range}(R), \neg\mathsf{range}(R))z') \\
). &
\end{aligned}$$

By Lemma 12 this definition is correct. By the inductive hypothesis, Lemma 13, and (25) all parts of the formula are definable as in-between down queries. Thus also the formula itself.

This concludes the proof of Lemma 11.

6 Complexity

We briefly discuss the complexity of the query evaluation problem and the succinctness of Conditional XPath compared to first order logic.

Query evaluation for Core XPath is hard for PTIME (combined complexity) and can be done in time $O(|D| \cdot |Q|)$, with $|D|$ the size of the data and $|Q|$ the size of the query [17, 18]. Using model checking results for Propositional Dynamic Logic [2] this can be extended to Regular (thus also to Conditional) XPath [28]:

Fact 2. Given nodes n, n', a tree $\mathfrak{M}$ and a Regular XPath path wff R. The problem whether $(n, n') \in [\![R]\!]_{\mathfrak{M}}$ is decidable in time $O(|\mathfrak{M}| \cdot |R|)$.

Because Conditional XPath is expressively complete for first order definable paths, it is interesting to compare this to results for first order logic. Query evaluation for first order queries is PSPACE complete and can be done in time $O(|D|^n \cdot |Q|)$, where n is the number of variables in Q [8, 21, 44]. So we can explain PTIME–completeness of Conditional XPath by the fact that it is a subset of FO^{tree} with at most three variables. The difference between Conditional XPath and first order logic becomes even sharper when we look at parametrized complexity. [12] showed that there is no query evaluation algorithm for FO^{tree} on the class of unranked trees whose running time is bounded by $f(|Q|) \cdot p(|D|)$, for an elementary function f and a polynomial p.

Even though not immediately visible, the transformations described in this paper can be turned into a translation from first order formulas into Conditional XPath queries. A natural question is how large the query will be. In the worst case it will be huge, because of the following: Satisfiability for Conditional XPath over finite trees is complete for exponential time [28]. Satisfiability of first order sentences with $<$ over finite words is in no elementary space-bounded complexity class [38].

Corollary 4. *No translation from FO^{tree} into Conditional XPath can be elementary space-bounded.*

7 Related work and conclusions

The results make us conclude that both Core and Conditional XPath are very natural languages for talking about ordered trees. Their simplicity and visual attractiveness make them suitable candidates for a user-friendly alternative to first order logic. The expressive completeness result for paths is very important, as arguably the relations in Conditional XPath are still "drawable". With drawable we mean that one can make an intuitive picture which exactly captures the meaning of the query. The conditional axis relation $(\mathsf{step}?A)^+$ is also drawable using ellipsis. Of course one should not draw the filter expressions, but just indicate them with formulas attached to nodes in the drawings.

The query evaluation problems for both Core and Conditional XPath are still solvable in time $O(|Q| \cdot |D|)$, with $|Q|$ and $|D|$ the sizes of the query and the data, respectively. Also the query containment problem is equally complex for both languages [28]. It would be exciting to see how existing XPath algorithms can be adjusted in order to deal efficiently with conditional path expressions.

In this context it is interesting to note a repetition in history. The natural class of models in computational linguistics is the class of finite ordered trees, i.e., parse trees. In the beginning of the field of model theoretic syntax Monadic Second Order Logic was invariably used to reason about these structures [39]. Later, formalisms based on modal logic were proposed as alternatives. Arguments for the alternatives were both based on computational complexity (which

is lower both for model checking and theorem proving) and on "naturalness" of expressing properties (in this case of grammars). In fact the here discussed languages have their roots in the nineties: [6], [36] and [25] define more or less isomorphic variants of the filter expressions of Core, Conditional and Regular XPath, respectively.

Even though Core XPath has a first order definition, one may argue that a node selecting language over ordered trees should have the expressive power of monadic second order (MSO) logic. This is done by [33] and they present a language which can express each MSO definable set of nodes. A sublanguage is shown complete for first order definable sets of nodes in [41]. Monadic datalog can also express each MSO definable set of nodes [16]. These languages however do not have the "look-and-feel" of XPath 1.0, and are more of theoretical interest. It is tempting to believe that Regular XPath is a candidate, but it cannot even define each regular class of trees [26].

A number of papers describe characterizations of positive fragments of Core XPath: [4] and [19] characterize positive Core XPath as positive existential first order formulas; [34] shows the equivalence of XPath fragments and fragments without backwards axis relations.

7.1 Relation with temporal logic

The observation that languages for XML like XPath, DTD's and XML Schema can be viewed as propositional temporal, modal or description logics has been made by several authors. For instance, [31] and [17] embed XPath into CTL. The group round Calvanese, de Giacomo and Lenzerini published a number of papers relating DTD's and XPath to description logic, thereby obtaining powerful complexity results cf, e.g., [7]. [1] reduce certain XML constraint inference problems to propositional dynamic logic. [6] define a modal language which is *exactly* as expressive as the node wffs of Core XPath. This language was proposed as a means to specify constraints on *parse trees*, indeed finite node labeled sibling ordered trees. The connection with XPath is established in [30].

Let us explain the connection between temporal logic and XPath from the viewpoint of XPath. The simplest model of time is a linear flow of time points. For now, we choose a flow which is like the natural numbers with the $<$ ordering. Then such a flow is a tree of rank one, and we can interpret our XPath expressions on it. Temporal logic is concerned with defining sets of time points, so we need only consider the node wffs. First observe that the horizontal steps are definable in a trivial way. For instance, $\langle \mathsf{right} \rangle \equiv \bot$, etc. So on this class of models only the vertical steps are meaningful in the language. The syntax of temporal logic can now be defined in the same spirit as XPath's node wffs:

$$\begin{aligned} \mathsf{t\_wff} ::= {} & p_i \mid \neg \mathsf{t\_wff} \mid \mathsf{t\_wff} \wedge \mathsf{t\_wff} \mid \\ & \langle \mathsf{child} \rangle \mathsf{t\_wff} \mid \langle \mathsf{child}^+ \rangle \mathsf{t\_wff} \mid \langle \mathsf{parent} \rangle \mathsf{t\_wff} \mid \langle \mathsf{parent}^+ \rangle \mathsf{t\_wff}. \end{aligned}$$

The meaning of $\langle \mathsf{child} \rangle \mathsf{t\_wff}$ is exactly the meaning of $\langle \mathsf{child}/?\mathsf{t\_wff} \rangle$, it is just another notation. This formalism is equally expressive as vertical Core XPath

[30]. The operators $\langle R\rangle \mathsf{t\_wff}$ are interpreted as follows:

$\langle \mathsf{child}\rangle \mathsf{t\_wff}$	Tomorrow t_wff will be true
$\langle \mathsf{child}^+\rangle \mathsf{t\_wff}$	Sometimes in the future, t_wff will be true
$\neg\langle \mathsf{child}^+\rangle\neg \mathsf{t\_wff}$	Always in the future, t_wff will be true
$\langle \mathsf{parent}\rangle \mathsf{t\_wff}$	Yesterday t_wff was true
$\vdots$	

Temporal logic having only unary temporal operators (those having only one argument, in contrast to the until operator which has two arguments) is usually called Priorean temporal logic, after Arthur Prior, who first formulated these kind of languages. We recall the intuitive explanation from [13] why this language cannot define every set of time points definable in first order logic. If it could the language would also have the so called separation property for temporal formulas. A temporal formula is separated if it is a Boolean combination of pure past, pure future and pure present formulas. These are *semantic* notions defined as follows: a formula A is pure future if its truth at a time point t depends only on that part of the model which is in the future of that point t. More precisely, if A is true at t in some model $\mathfrak{M}$, then A remains true at t if we change the labels at any time point *before or equal* to t, but keep the same labels in the future of t. The other two notions are defined similarly. Now consider the formula

$$\langle \mathsf{child}^+\rangle(p \wedge \neg\langle \mathsf{parent}^+\rangle\neg q), \tag{26}$$

It is instructive to try to separate this formula into a past, present and future part. It won't work. What is needed is a connective expressing "until p is true, q is true". This can be defined if we allow the conditional paths of the form $(\mathsf{child}?A)^*$ inside the temporal modalities:

$$\mathsf{until}(p, q) \ ::= \ \langle(\mathsf{child}?q)^*\rangle\langle \mathsf{child}\rangle p.$$

Note that $\mathsf{until}(p, q)$ is a pure future formula. With until we can separate (26) as

$$\neg\langle \mathsf{parent}^+\rangle\neg q \wedge q \wedge \mathsf{until}(p, q).$$

The study of expressive completeness with respect to first order logic of modal languages originated with Kamp's thesis [23]. He showed that Priorean tense logic is not sufficient for expressing each first order definable set of time points. He added the since and until connectives and subsequently showed that the resulting language is expressively complete with respect to Dedekind complete linear structures. Expressive completeness here means with respect to formulas in *one* free variable. [14] made this result accessible by introducing the notion of *separation* and generalized it. The monograph [13] is a good entry point to this field. A model–theoretic version of the separation property is Shelah's Composition Method [43].

Rather surprisingly, it took almost thirty years before someone asked which fragment of first order logic then corresponds to Priorean tense logic, when

interpreted on linear flows of time. This was answered by [11]: exactly the first order formulas in one free variable which are written in at most *two* variables in total. [30] extended this result to ordered trees and the Core XPath language.

[22] showed that first order logic has the three variable property when interpreted on linear structures. They also classify how many variables are needed for ranked trees. Unranked trees do not have a finite variable property.

[40] generalized Kamp's Theorem to ranked and unranked trees. Besides since and until, he has a connective $\mathcal{X}_k$ counting the number of children making a wff true:

$$t \in [\![\mathcal{X}_k A]\!]_{\mathfrak{M}} \iff \exists t_1 \dots t_k \bigwedge_{1 \leq i \neq j \leq k} (tR_{\downarrow} t_i \wedge t_i \neq t_j \wedge t_i \in [\![A]\!]_{\mathfrak{M}}).$$

Note that $\mathcal{X}_k$ is expressible over ordered trees. For k ranked trees (trees in which each node has at most k children), the language with since and until plus all $\mathcal{X}_i$, for $2 \leq i \leq k$ is expressively complete. For unranked trees, $\mathcal{X}_i$ is needed for every natural number i. Unfortunately this result does not imply separation. Consider the node wff $\langle \mathsf{parent} \rangle \langle \mathsf{child} \rangle A$. On linear structures, this is equivalent to $\langle \mathsf{parent} \rangle \top \wedge A$, on ordered trees to $\langle \mathsf{parent} \rangle \top \wedge (A \vee \langle \mathsf{left}^+ \rangle A \vee \langle \mathsf{right}^+ \rangle A)$, which are both separated formulas. But the signature of unordered trees is too weak to separate this formula. Expressive completeness with respect to first order formulas in one free variable for a language closely related to Conditional XPath's node wffs was announced in [35]. Unfortunately, the proof does not use separation, it is over 50 pages long and very hard to follow.

Here we just mentioned work about *first order* expressive completeness of variable free languages. For second order expressivity results of temporal logics on trees cf., [37, 3].

Acknowledgments

Maarten Marx was supported by the Netherlands Organization for Scientific Research (NWO), under project number 612.000.106. The history of the results in this paper is as follows. [28] announced the expressive completeness of Conditional XPath with respect to first order definable sets of nodes, which was proved in [27]. The step from expressing sets of nodes to sets of paths was made in [29].

I would like to thank the reviewers for their extensive and extremely helpful comments. Special thanks are due to Balder ten Cate and Petrucio Viana. I also thank Loredana Afanasiev, David Gabelaia, Evan Goris, Jan Hidders, Sanjay Modgil, Maarten de Rijke, Thomas Schwentick, Moshe Vardi and Yde Venema for helpful discussions.

References

1. N. Alechina, S. Demri, and M. de Rijke. A modal perspective on path constraints. *Journal of Logic and Computation*, 13:1–18, 2003.

2. N. Alechina and N. Immerman. Reachability logic: An efficient fragment of transitive closure logic. *Logic Journal of the IGPL*, 8(3):325–337, 2000.
3. P. Barcelo and L. Libkin. Temporal logics over unranked trees. In *Proceedings of the IEEE Symposium on Logic in Computer Science (LICS'05)*, 2005.
4. M. Benedikt, W. Fan, and G. Kuper. Structural properties of XPath fragments. In *Proceedings ICDT 2003*, 2003.
5. P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press, 2001.
6. P. Blackburn, W. Meyer-Viol, and M. de Rijke. A proof system for finite trees. In H. Kleine Büning, editor, *Computer Science Logic*, volume 1092 of *LNCS*, pages 86–105. Springer, 1996.
7. Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Representing and reasoning on XML documents: A description logic approach. *Journal of Logic and Computation*, 9(3):295–318, 1999.
8. A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proceedings of 9th ACM Symposium on Theory of Computing*, pages 77–90, 1977.
9. E. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Database Systems*, pages 33–64. Prentice-Hall, 1972.
10. H.-D. Ebbinghaus and J. Flum. *Finite Model Theory*. Perspectives in Mathematical Logic. Springer, 1995.
11. K. Etessami, M. Vardi, and Th. Wilke. First-order logic with two variables and unary temporal logic. In *Proc. LICS'97*, pages 228–235, 1997.
12. M. Frick and M. Grohe. The complexity of first-order and monadic second-order logic revisited. In *17th IEEE Symposium on Logic in Computer Science (LICS'02)*, pages 215–224, 2002.
13. D.M. Gabbay, I. Hodkinson, and M. Reynolds. *Temporal Logic*. Oxford Science Publications, 1994. Volume 1: Mathematical Foundations and Computational Aspects.
14. D.M. Gabbay, A. Pnueli, S. Shelah, and J. Stavi. On the temporal analysis of fairness. In *Proc. 7th ACM Symposium on Principles of Programming Languages*, pages 163–173, 1980.
15. G. Gottlob and C. Koch. Monadic queries over tree-structured data. In *Proc. LICS*, Copenhagen, 2002.
16. G. Gottlob and C. Koch. Monadic datalog and the expressive power of languages for web information extraction. *J. ACM*, 51(1):74–113, 2004.
17. G. Gottlob, C. Koch, and R. Pichler. Efficient algorithms for processing XPath queries. In *VLDB'02*, 2002.
18. G. Gottlob, C. Koch, and R. Pichler. The complexity of XPath query evaluation. In *PODS'03*, pages 179–190, 2003.
19. G. Gottlob, C. Koch, and K. Schulz. Conjunctive queries over trees. In *Proceedings of PODS*, pages 189–200, 2004.
20. D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. MIT Press, 2000.
21. N. Immerman. Upper and lower bounds for first order expressibility. *J. Comput. Syst. Sci.*, 25:76–98, 1982.
22. N. Immerman and D. Kozen. Definability with bounded number of bound variables. In *Proceedings of the Symposium of Logic in Computer Science*, pages 236–244, Washington, 1987. Computer Society Press.
23. J.A.W. Kamp. *Tense Logic and the Theory of Linear Order*. PhD thesis, University of California, Los Angeles, 1968.

24. D. Kozen. Kleene algebra with tests. *ACM Transactions on Programming Languages and Systems*, 19(3):427–443, 1997.
25. M. Kracht. Inessential features. In Christian Retore, editor, *Logical Aspects of Computational Linguistics*, number 1328 in LNAI, pages 43–62. Springer, 1997.
26. M. Kracht. *Tools and Techniques in Modal Logic*. Number 142 in Studies in Logic. Elsevier, Amsterdam, 1999.
27. M. Marx. Conditional XPath, the first order complete XPath dialect. In *Proc. PODS'04*, pages 13–22, 2004.
28. M. Marx. XPath with conditional axis relations. In *Proc. EDBT'04*, volume 2992 of *LNCS*, pages 477–494, 2004.
29. M. Marx. First order paths in ordered trees. In *Proc. ICDT 2005*, volume 3363 of *LNCS*, pages 114–128, 2005.
30. M. Marx and M. de Rijke. Semantic characterizations of XPath. In *TDM'04 workshop on XML Databases and Information Retrieval.*, Twente, The Netherlands, June 21, 2004.
31. G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Proc. PODS'02*, pages 65–76, 2002.
32. F. Neven. *Design and Analysis of Query Languages for Structured Documents*. PhD thesis, Limburgs Universitair Centrum, Belgium, 1999.
33. F. Neven and T. Schwentick. Expressive and efficient pattern languages for tree-structured data. In *Proc. PODS*, pages 145–156. ACM, 2000.
34. D. Olteanu, H. Meuss, T. Furche, and F. Bry. Xpath: Looking forward. In *EDBT Workshops*, volume 2490 of *Lecture Notes in Computer Science*, pages 109–127. Springer, 2002.
35. A. Palm. *Transforming tree constraints into formal grammars*. PhD thesis, Universität Passau, 1997.
36. A. Palm. Propositional tense logic for trees. In *Sixth Meeting on Mathematics of Language*. University of Central Florida, Orlando, Florida, 1999.
37. A. Rabinovich. Expressive power of temporal logics. In *CONCUR 2002, Proceedings*, volume 2421 of *LNAI*, pages 57–75, 2002.
38. K. Reinhardt. The complexity of translating logic to finite automata. In E. Grädel et al., editor, *Automata, Logics, and Infinite Games*, volume 2500 of *LNCS*, pages 231–238. Springer, 2002.
39. J. Rogers. *A Descriptive Approach to Language Theoretic Complexity*. CSLI Press, 1998.
40. B-H. Schlingloff. Expressive completeness of temporal logic of trees. *Journal of Applied Non–Classical Logics*, 2(2):157–180, 1992.
41. Th. Schwentick. On diving in trees. In *Proc. 25th Symposium on Mathematical Foundations of Computer Science*, pages 660–669, 2000.
42. A. Tarski and S. Givant. *A Formalization of Set Theory without Variables*, volume 41. AMS Colloquium publications, Providence, Rhode Island, 1987.
43. Wolfgang Thomas. Ehrenfeucht games, the composition method, and the monadic theory of ordinal words. In *Structures in Logic and Computer Science, A Selection of Essays in Honor of Andrzej Ehrenfeucht*, pages 118–143. Springer-Verlag, 1997.
44. M. Vardi. On the complexity of bounded–variable queries. In *Proceedings of the ACM 14th Symposium on Principles of Database Systems*, pages 266–276, 1995.
45. M. Vardi. Why is modal logic so robustly decidable? In *DIMACS Series in Discrete Mathematics and Theoretical Computer Science 31*, pages 149–184. American Math. Society, 1997.
46. W3C XPath 1.0. XML path language (XPath): Version 1.0. `http://www.w3.org/TR/xpath.html`.

47. W3C XQuery. XQuery 1.0: A query language for XML. http://www.w3.org/TR//xquery/.
48. W3C XSLT. XSL transformations language (XSLT): Version 2.0. http://www.w3.org/TR/xslt20/.
49. P. Wadler. Two semantics for XPath. Technical report, Bell Labs, 2000.