From XQuery to Relational Logics

Predicate logic has long been seen as a good foundation for querying relational data. This is embodied in the correspondence between relational calculus and first-order logic, and can also be seen in mappings from fragments of the standard relational query language SQL to extensions of first-order logic (e.g. with counting). A key question is what is the analog to this correspondence for querying tree-structured data, as seen, for example, in XML documents. We formalize this as the question of the appropriate logical query language for defining transformations on tree-structured data. The predominant practitioner paradigm for defining such transformations is *top-down tree building*. This is embodied by the XQuery query language, which builds the output tree in parallel starting at the root, based on variable bindings and nodeset queries in the XPath language. The goal of this paper is to compare the expressiveness of top-down tree-building languages based on a benchmark of predicate logic. We start by giving a "formalized XQuery" XQ that can serve as a representative of the top-down approach. We show that all queries in XQ with only atomic equality are equivalent to "first-order interpretations", an analog to first-order logic (FO) in the setting of transformations of tree-structured data. We then consider fragments of atomic XQ. We identify a fragment that maps *efficiently* into first-order, a fragment that maps into existential first-order logic, and a fragment that maps into the "navigationally two-variable" fragment of first-order logic – an analog of two-variable logic in the setting where data values are unbounded. When XQ is considered with deep equality, we find that queries can be translated into FO with counting ($FO(\mathbf{Cnt})$). Translations from XQ to logical languages on relations have a number of consequences. We use them to derive complexity bounds for XQ fragments, and to bound the Boolean expressiveness of XQ fragments.

Categories and Subject Descriptors: H.2.3 [**Languages**]: Query languages

1. INTRODUCTION

The formal foundation for relational query languages is well-established. Much of relational querying can be done in the relational calculus, which is equivalent in expressiveness to first-order logic. What about tree-structured data, as exemplified by XML data trees? For Boolean and nodeset queries on pure node-labeled trees, the querying model is fairly well understood. The predominant practitioner language is XPath, and over a fixed label alphabet the core of XPath corresponds in expressiveness to the two-variable fragment of first-order logic over trees [Marx 2005]. The expressiveness of XPath with data value joins over an unbounded set of data values is less well-understood, although some connections with logic are known [Anonymous 2008a]. However, when we turn to queries that produce trees from trees, the logical counterpart becomes less clear, since logics by themselves do not define mappings from structures to structures.

The predominant paradigm for XML querying in practice is *top-down non-recursive tree building*, as exemplified in the standard language XQuery. The key feature of XQuery and its precursors (e.g. Quilt [Chamberlin et al. 2000]) is that the output tree is built top-down in parallel, with threads of control at every leaf of the partially-constructed output. Rather than using a general structural recursion mechanism, these languages use explicit nesting of subqueries to build the output up to some fixed depth. In our paper we formalize the top-down paradigm by the query language XQ, an abstraction of XQuery that captures the main tree-formation constructs. We wish to compare the expressiveness of XQ and its frag-

ments to some external benchmark. For relational query languages, a benchmark is first-order logic; the results of Codd show that the relational calculus and relational algebra are both equivalent in expressiveness to first-order logic (see, e.g. [Abiteboul et al. 1995]), and thus form a natural foundation for query languages. In this work we will choose to use logics as a benchmark. We rely on the well-known notion of *first-order interpretation* [Ebbinghaus and Flum 1999], which associates a collection of first-order formulas with a mapping from structures to structures. Informally, a tree-to-tree transformation T is given by a first-order interpretation if there are a set of relational calculus expressions that produce, from a relational coding of an input data tree $\mathcal{T}$, the relational coding of $T(\mathcal{T})$. The use of logics, rather than relational algebra, will simplify e the proofs; it will also make it possible to apply prior results from logic in getting expressiveness results for sublanguages. However, using the classical results cited above, we can see that queries given by first-order interpretations can be implemented relationally using relational algebra or relational calculus.

Contributions: We start by showing that XQ queries that cannot make "deep equality comparisons" (that is, isomorphism tests of subtrees) can be transformed into equivalent FO interpretations: this is an analog for tree query languages of the simulations for complex object languages in relational algebra [Paredaens and Van Gucht 1988]. We then trace how the mapping between XQ and first-order logic filters down to fragments of XQ. Although the transformation for general XQ requires exponential time, we prove that it can be done in polynomial time when restricting to XQ queries in a natural class (the "composition-free" XQ queries [Koch 2006]). We then examine queries that do not use node equality comparisons; for queries in this fragment that use only downward navigation, as well as those that are composition-free, one can map to a restricted fragment of FO, one that is "almost" within two-variable logic. We show that the positive fragment of XQ maps to the existential fragment of first-order logic. We establish lower bounds on the complexity of both of these transformations.

The above concerns the equivalence of XQ with atomic equality and interpretations given by first-order logic/relational calculus. We also study the variant of XQ where queries can compare trees for structural equality (we refer to this as "deep equality", in analogy with the corresponding construct in complex value languages). Not every such query can be transformed into a corresponding first-order interpretation. We hence compare the expressiveness of deep equality queries with interpretations given using first-order logic with counting $FO(\mathbf{Cnt})$, which corresponds to a restricted form of relational calculus with aggregation. Queries given by these interpretations can still be implemented relationally in a fragment of SQL. We give a transformation producing from each XQ query with deep equality a corresponding $FO(\mathbf{Cnt})$ interpretation, and extend the complexity bounds on this transformation from the first-order case.

Our results are applied to the *complexity* of XQ. Using our translations, we give new upper bounds on the data complexity of fragments of XQ, as well as simpler proofs of prior bounds.

In a companion paper [Anonymous 2008b], we examine the question of mapping back from relational logics to XQ, considering whether the correspondence between XQ queries and logics is tight.

Organization: Section 2 gives background on the XQuery fragments we con-

sider and the notion of logical interpretation that we use to capture query language expressiveness. Section 3 gives a translation from XQ queries using only atomic equality into first-order interpretations. The translation algorithm in this section is also the basis for the analyses that come later in the paper. Sections 4 and 5 prove the correctness of the algorithm. Section 6 looks at properties of this translation, including the complexity of translation and the behavior of the translation on fragments of the language. Section 7 gives translations between full XQ and interpretations in first-order logic with a count operator. Section 8 gives conclusions, while Section 9 discusses related work.

2. BACKGROUND AND NOTATIONS

2.1 Data and Query Model

An *unranked ordered tree* is a tree in which nodes may have a variable number of children, with an order among them. A *data tree* is a two-sorted relational structure with sorts Node, Lab and signature

$$\sigma_{nav} = (\mathsf{Haslabel}, \mathsf{child}, \mathsf{next\text{-}sibling}, \mathsf{descendant}, \mathsf{following\text{-}sibling}).$$

This represents a unranked ordered tree whose nodes are the elements of sort Node and whose labels are the elements of sort Lab. That is, the interpretation of the sorts will vary with the data tree, and hence we do not assume the labeling alphabet to be fixed. The binary relation Haslabel represents a partial function $\mathsf{lab}() : \mathsf{Node} \rightarrow \mathsf{Lab}$ that takes a node to its label; $\mathsf{Haslabel}(x, z)$ holds iff node x has label z, that is, $\mathsf{lab}(x) = z$. The binary relation child is the parent-child relation among nodes, next-sibling is the binary immediate right-sibling relation among nodes (i.e., $\mathsf{next\text{-}sibling}(x, y)$ iff y is the right-sibling of x), descendant denotes the descendant relation, following-sibling is the transitive closure of the next-sibling relation.

The binary relation $=_{atomic}$ is derivable from the above, where for nodes x and y we have $x =_{atomic} y \leftrightarrow \mathsf{lab}(x) = \mathsf{lab}(y)$. From the basic navigation relations in σ_{nav} we can derive their inverses: parent the inverse of child, ancestor the inverse of descendant, previous-sibling the inverse of next-sibling, and preceding-sibling the inverse of following-sibling. For a data tree $\mathcal{T}$, we let the binary relation $<^{\mathcal{T}}_{doc}$ on nodes be the *document-order* on $\mathcal{T}$: the depth-first left-to-right traversal order through $\mathcal{T}$.

A *data forest* is a relational structure of the same signature σ_{nav}, but with the underlying node structure being a forest; i.e. we allow multiple root nodes, and we do not have a way of ordering nodes that lie beneath different roots. Given a node b in a data forest, the *subtree of* b refers to the σ_{nav}-substructure of the data forest whose domain consists of b and its decendants and the labels associated with those nodes by the relation Haslabel.

An *indexed forest* $\mathcal{F}_{\#}$ is a pair consisting of a data forest and a sequence of its nodes. It will be convenient for us to consider indexed forests as a two-sorted relational structure. We will use a two-sorted signature

$$\sigma_{nav,ind} = (\sigma_{nav}, \mathsf{IsNode}, \mathsf{IsInd}, \mathsf{ItemOf}, <_{\mathrm{ind}}).$$

with a sort Lab for labels and a "combined node/index sort" NI. IsNode is a unary predicate on the combined sort, identifying the nodes of $\mathcal{F}_{\#}$. The unary predicate IsInd, represents the complement of IsNode within the combined sort, the indices of $\mathcal{F}_{\#}$. $\mathsf{ItemOf}(x', x)$ is a binary relation on the combined sort holding iff index x'

maps to node x. The binary relation $<_{\text{ind}}$ fixes a total order on the index elements. (Thus the indexes denote a sequence of nodes, as claimed.) The other predicates are interpreted as before.

All of the trees, forests, and indexed forests considered in this paper will be finite.

EXAMPLE 2.1. *The data tree represented in XML as* $\langle A\rangle\langle B/\rangle\langle C/\rangle\langle/A\rangle$ *is represented as a* σ_{nav} *structure in which we*

—*interpret the nodes by a set of three elements* $\mathsf{Node} = \{n_1, n_2, n_3\}$,
—*interpret the set of labels by* $\mathsf{Lab} = \{A, B, C\}$,
—*map both* child *and* descendant *to* $\{(n_1, n_2), (n_1, n_3)\}$,
—*map both* next-sibling *and* following-sibling *to* $\{(n_2, n_3)\}$, *and*
—*interpret* Haslabel *as* $\{(n_1, A), (n_2, B), (n_3, C)\}$.

Now consider the indexed forest consisting of the data tree above supplemented with a sequence consisting of the two non-root nodes n_2 *and* n_3, *in document order. Our relational representation of this indexed forest consists of*

—*the set of indices* $\{i_1, i_2\}$
—*a set* n_1, n_2, n_3, i_1, i_2, *representing the nodes and indices*
—*the* ItemOf *relation* $\{(i_1, n_2), (i_2, n_3)\}$
—*the* $<_{\text{ind}}$ *relation with* (i_1, i_2)

and the remaining sets and relations are as above.

In the examples we give throughout this paper, we will use XML documents without attributes or PCDATA as our data trees. In these examples, the tag of an element in the XML document will correspond to the label of a node in the data tree. However, we note that data trees can model more complex XML documents. Since we do not assume that the labels come from a fixed set, the labeling could represent the *value* of an attribute node (say, with the attribute name coded into the value) or the *text content* of a PCDATA node. Our atomic equality $=_{atomic}$ would thus allow us to check for equality of attribute values, or to check textual equivalence of PCDATA elements.

There is some arbitrariness in our choice of predicates: we could have made do with a single sort, and could have had predicates for nodes, indices, and labels. The rationale for our particular choices should become clearer later, but we give a hint as to the reasons now. In the signature for both data trees and indexed forests, we separate out labels as a distinct sort, because variables will never need to vary over both labels and non-labels; by having labels as a sort rather than a predicate, we get a concise notation for restricting a variable or constant to range over labels. In contrast, in the signature for indexed forests we do *not* have nodes and indices as separate sorts because we will occasionally need variables that can range over both of these. The use of a combined sort will also be helpful in modeling XQuery's tree-formation constructors, which build new nodes out of indices.

Equality relativized to nodes will be denoted by $=_{node}$. Two data forests $\mathcal{F}, \mathcal{F}'$ are said to be *isomorphic* if the set of labels occurring on some node is the same in $\mathcal{F}$ and $\mathcal{F}'$ and there is a bijection between the nodes of $\mathcal{F}$ and the nodes of $\mathcal{F}'$ that preserves the child and sibling relations as well as the labeling function. Isomorphism between indexed forests $\mathcal{F}_{\#}$ and $\mathcal{F}_{\#}{}'$ is defined similarly, requiring the

labels of $\mathcal{F}_{\#}$ and $\mathcal{F}_{\#}{}'$ to be the same, and requiring a pair of bijections between the nodes and the indices that together preserve each predicate. Isomorphism between data forests will be denoted by $=_{deep}$ (also referred to as "deep equality"). Given two nodes in (possibly distinct) data forests, we will say they are $=_{deep}$ if their subtrees within the respective forests are isomorphic. Note that for a mapping between data forests to be an isomorphism, the labels of associated nodes must be *the same*; i.e. not just the same up to re-labeling. At the XML level, our deep equality allows us to model equalities between the string value associated with interior nodes (e.g. elements) of an XML document.

We will also consider a weaker form of isomorphism in which sibling order is ignored. For data forests $\mathcal{F}$ and $\mathcal{F}'$ we say that $\mathcal{F} =_{deep,un} \mathcal{F}'$ if there is a mapping from $\mathsf{Dom}(\mathcal{F})$ to $\mathsf{Dom}(\mathcal{F}')$ that preserves all the relations of $\sigma_{nav} - \{\mathsf{next\text{-}sibling}, \mathsf{following\text{-}sibling}\}$.

By a *data forest query* we mean any function taking as input a data forest expanded by an assignment of nodes to a given set of variables and returning a data forest, and by an *indexed forest query* a function taking an indexed forest and a variable assignment and returning an indexed forest. A *Boolean query* is any function taking as input either a data forest or indexed forest, expanded by a variable assignment, and returning a value in $\{true, false\}$. We also talk about a *query with parameters* $v_1 \dots v_n$ when we want to specify the variables. Two data forest queries Q, Q' with parameters $v_1 \dots v_n$ are said to be *equivalent* if $\forall(\mathcal{F}, b_1 \dots b_n)\ Q(\mathcal{F}, \vec{b}) =_{deep} Q'(\mathcal{F}, \vec{b})$, and *equivalent modulo order* if for all $\mathcal{F}$ and $\vec{b}$, $Q(\mathcal{F}, \vec{b}) =_{deep,un} Q'(\mathcal{F}, \vec{b})$. The notion of indexed forest queries being equivalent is defined analogously.

2.2 The XQuery Fragment XQ

Our query language XQ has the abstract syntax given in Figure 1.

$$
\begin{array}{rcl}
\mathit{query} & ::= & () \mid \mathit{query}\ \mathit{query} \mid \mathit{var} \mid \mathit{var}/\mathit{axis} :: \nu \\
& \mid & \langle a \rangle \mathit{query} \langle /a \rangle \mid \langle \mathsf{lab}(\mathit{var}) \rangle\ \mathit{query}\ \langle /\mathsf{lab}(\mathit{var}) \rangle \\
& \mid & \textbf{for}\ \mathit{var}\ \textbf{in}\ \mathit{query}\ \textbf{return}\ \mathit{query} \\
& \mid & \textbf{if}\ \mathit{cond}\ \textbf{then}\ \mathit{query} \\
\mathit{cond} & ::= & \mathit{var} =_{node} \mathit{var} \mid \mathit{var} =_{atomic} \mathit{var} \mid \mathit{var} =_{deep} \mathit{var} \mid \mathit{query}
\end{array}
$$

Fig. 1. Syntax of XQ.

In this grammar, a denotes a label, and *axis* denotes the tree-structure relations child, parent, descendant, ancestor, previous-sibling, preceding-sibling, next-sibling, and following-sibling. Above *var* refers to one in a set of variables \$$x$, \$$x_1$, \$$x_2$, ..., \$$y$, \$$z$, ..., and ν to a *label test*, which is either a label or "*". Thus, our XQ fragment extends that of [Koch 2006] by a node equality primitive, which returns true on a pair of nodes iff they are the same node, and the power to construct new nodes taking labels from arbitrary input nodes. Note that our data model does not assume that the set of labels for documents is from a fixed set. Our fragment extends that used in the expressiveness results of [Koch 2006] by both the presence of node equality and the inclusion of more tree structure relations ("axes"; those of XQuery, and additional ones) whereas [Koch 2006] only considers "child" and

$$
\begin{aligned}
[\![()]\!]_n(\mathcal{F},\vec{e}) &:= (\mathcal{F},[\,]) \\
[\![\langle a\rangle\alpha\langle/a\rangle]\!]_n(\mathcal{F},\vec{e}) &:= \mathit{construct}(a,[\![\alpha]\!]_n(\mathcal{F},\vec{e})) \\
[\![\langle \mathsf{lab}(\$x_i)\rangle\alpha\langle/\mathsf{lab}(\$x_i)\rangle]\!]_n(\mathcal{F},e_1,\ldots,e_n) &:= \mathit{construct}(\mathsf{lab}(e_i),[\![\alpha]\!]_n(\mathcal{F},\vec{e})) \\
[\![\alpha\ \beta]\!]_n(\mathcal{F},\vec{e}) &:= [\![\alpha]\!]_n(\mathcal{F},\vec{e}) \uplus [\![\beta]\!]_n(\mathcal{F},\vec{e}) \\
[\![\textbf{for } \$v_{n+1} \textbf{ in } \alpha \textbf{ return } \beta]\!]_n(\mathcal{F},\vec{e}) &:= \text{let } (\mathcal{F}',\vec{l}) = [\![\alpha]\!]_n(\mathcal{F},\vec{e}) \text{ in } \biguplus_{1\le i\le|\vec{l}|} [\![\beta]\!]_{n+1}(\mathcal{F}',\vec{e}\cdot\vec{l}_i) \\
[\![\$x_i]\!]_n(\mathcal{F},e_1,\ldots,e_n) &:= (\mathcal{F},[e_i]) \\
[\![\$x_i/\chi :: \nu]\!]_n(\mathcal{F},e_1,\ldots,e_n) &:= (\mathcal{F},\text{list of nodes } v \text{ such that } \chi^{\mathcal{F}}(e_i,v) \text{ and} \\
&\quad\ \mathsf{lab}(v)=\nu \text{ in order } <_{doc}^{tree(e_i)}) \\
[\![\textbf{if } \phi \textbf{ then } \alpha]\!]_n(\mathcal{F},\vec{e}) &:= \text{if } \pi_2([\![\phi]\!]_n(\mathcal{F},\vec{e}))\neq[\,] \text{ then } [\![\alpha]\!]_n(\mathcal{F},\vec{e}) \text{ else } (\mathcal{F},[\,]) \\
[\![\$x_i=\$x_j]\!]_n(\mathcal{F},e_1,\ldots,e_n) &:= \text{if } e_i=e_j \text{ then } \mathit{construct}(\text{“}\mathit{yes}\text{”},(\mathcal{F},[\,])) \\
&\quad\ \text{else } (\mathcal{F},[\,])
\end{aligned}
$$

Fig. 2. Semantics of XQ.

"descendant". A more detailed comparison with the models used in related work can be found in Section 9.

Our language aims to capture the major tree-structure forming constructs of the practitioner language XQuery.

Semantics. The notion of a variable being *free* in an XQ query is given using the obvious inductive definition. For example, in **for** $\$v_{n+1}$ **in** α **return** β, the free variables are those of α and those of β, minus $\$v_{n+1}$. In giving the semantics of XQ expressions with n free variables, we will choose an ordering of the free variables as $\$v_1 \ldots \$v_n$. We will assume that every XQ query has at least one free variable in it; since data must be accessed initially via a variable in our fragment, this assumption does not limit expressiveness.

We define the semantics of an XQ expression α with at most n free variables using a function $[\![\alpha]\!]_n$ – given in Figure 2 – that takes a data forest $\mathcal{F}$ and a n-tuple of nodes from the forest as input and returns an indexed forest.

In Figure 2, in some cases, the index component is described as a list or integer-indexed sequence, but it can clearly be converted to the relational representation of indexes required by the signature $\sigma_{nav,ind}$ (e.g. by taking the indexes to be an initial segment of the integers). We also make use of some functions that construct indexed forests.

—The operator $\mathit{construct}(a,(\mathcal{F},[w_1\ldots w_n]))$, denotes construction of a new tree. The argument a is a label, while $\mathcal{F}$ is a data forest and $[w_1\ldots w_n]$ is a list of nodes in $\mathcal{F}$. When applied, *construct* returns an indexed forest $(\mathcal{F}\cup T',[\mathit{root}(T')])$, where T' is a tree with domain a new set of nodes, whose root $\mathit{root}(T')$ is labeled with a, and with the subtree rooted at the i^{th} (in sibling order) child of $\mathit{root}(T')$ being an isomorphic copy of the subtree rooted by w_i in $\mathcal{F}$. The function *construct* is assumed to return a tree with a distinct set of nodes each time it is called.

—The symbol $\uplus$ in Figure 2, takes two indexed forests $(\mathcal{F}_1,\vec{l_1}),(\mathcal{F}_2,\vec{l_2})$ where the $\mathcal{F}_i$ are data forests and the $\vec{l_i}$ are lists of nodes in $\mathcal{F}_i$. It returns an indexed forest $(\mathcal{F}_1\cup\mathcal{F}_2,\vec{l})$ where $\vec{l}$ is the concatenation of $\vec{l_1}$ and $\vec{l_2}$.

—$\biguplus$ is defined similarly to take the union of a sequence of structures; the list component returned by $\biguplus$ is the concatenation of the lists of its arguments. In the return clause, $\vec{l}_i$ is the i-th element of list $\vec{l}$.

—In the semantics of $x_i/\chi :: \nu$, we use $tree(e_i)$ to denote the (maximal) tree within the input forest that contains the node e_i, hence $<_{doc}^{tree(e_i)}$ is the document-order on the tree containing e_i. For the "forward axes", child, descendant, next-sibling, and following-sibling, $\chi^{\mathcal{F}}$ is simply the interpretation of the axis relation of the same name in the data forest. For the other axes, $\chi^{\mathcal{F}}$ denotes the corresponding derived relation: for $\chi =$ parent, it is the inverse of child in $\mathcal{F}_{\#}$, etc. The equality lab(v)='*' is true on all nodes v. The semantic function for $=$ is overloaded for each of the three kinds of equality.

—In the semantics for conditionals, π_2 refers to projection on to the second component of a pair (in this case, the list component of the indexed forest returned by the semantic function).

Output-node and Boolean semantics. The general semantics of XQ takes as input a forest and a variable assignment and produces an indexed forest. An alternative semantics is to consider only the "reachable output" of a query; that is, the data-forest-to-data-forest query defined by an XQ expression is obtained from the prior semantics by returning the forest consisting of all subtrees rooted at the returned nodes in the nodelist component. We call this the *output-node semantics* of an expression, and for an expression Q we write Q_{out} for the query returning this forest. When comparing the expressiveness of sublanguages of XQ, it is natural to use the output-node semantics. However the full semantics will be needed inductively within proofs.

Finally, we can consider an XQ query Q to define a Boolean query Q_{Bool} on data forests: this is the query that returns true on forest $\mathcal{F}$ for a given assignment to the variables iff the list output by Q when applied to $\mathcal{F}$ with that assignment is non-empty. It is easy to show that alternative definitions of Boolean queries (e.g. by looking at all the queries generated by the "condition" non-terminal in the grammar) yield the same set of queries.

Example 2.2. Consider the evaluation of query Q

$$\textbf{for } \$w \textbf{ in } \langle a\rangle\langle b/\rangle\langle/a\rangle \textbf{ return } (\$v_0/\mathsf{descendant} :: *\ \ \$w/\mathsf{child} :: *)$$

on an input tree $\mathcal{T}$ consisting of two nodes n_1 and n_2, with n_2 a child of n_1 and the free variable $\$v_0$ pointing to n_1.

Under the **full semantics of** XQ, Q returns a forest and a sequence of nodes. The forest component contains the input tree $\mathcal{T}$ and two new trees, $\mathcal{T}'$ consisting of node n_5 labeled b and $\mathcal{T}''$ consisting of nodes n_3 and n_4, where n_3 is a root node labeled a and n_4 is a child of n_3 and is labeled b (i.e., the single-node tree n_4 is a copy of $\mathcal{T}'$). The sequence is $< n_2, n_4 >$. Under the **output-node semantics**, Q outputs a forest consisting of a tree consisting only of n_2 and a tree containing only the node n_4. Under the **boolean semantics** Q returns **true**. □

Dialects of XQ**.** We will consider the following sublanguages of XQ:

—By $AtomXQ$ we denote XQ where we restrict general deep equality comparisons between variables, and instead allow only conditions $var =_{deep} \langle a/\rangle$ for labels a.

—By $PosXQ$, we refer to the subset where deep equality is not permitted at all.
—$AtomXQ^-$ denotes the sublanguage of $AtomXQ$ where comparisons $var =_{node} var$ are forbidden, and $PosXQ^-$ denotes the corresponding sublanguage of $PosXQ$.

We will also consider fragments obtained from these sublanguages by excluding the next-sibling and following-sibling axes. Moreover, for purely technical reasons, we will also introduce the language $AtomXQ^+$, which *extends* $AtomXQ$ by a true composition construct, in Section 3.

Derived Operators. In our definition of the syntax of XQ, we have been economical with operators introduced. For example, we use only simple axis expressions as primitives in a "step". In fact, one can show that one could have included all of Core XPath [Gottlob et al. 2005] as a sublanguage without affecting the expressiveness; indeed, this follows from the results of this paper along with [Anonymous 2008b]. For the moment, we note that the following useful constructs can be derived:

$$\begin{aligned}
(\textbf{let } \$x \ := \ \alpha_0)\ \beta \ &:= \ \textbf{for } \$x \textbf{ in } \alpha_0 \textbf{ return } \beta \\
\phi \textbf{ or } \psi \ &:= \ \phi\ \psi \qquad \phi \textbf{ and } \psi := \textbf{if } \phi \textbf{ then } \psi \\
\textbf{true} \ &:= \ \langle a/\rangle \qquad \textbf{not } \phi := (\textbf{let } \$v \ := \ \langle a\rangle\phi\langle/a\rangle)\ \ \$v =_{deep} \langle a/\rangle \\
\textbf{some } \$x \textbf{ in } \alpha \textbf{ satisfies } \phi \ &:= \ \textbf{for } \$x \textbf{ in } \alpha \textbf{ return } \phi
\end{aligned}$$

where $\langle a/\rangle$ is a shortcut for $\langle a\rangle()\langle/a\rangle$. In using the above as a definition of **let**, we must restrict α_0 to be either of form $\langle a\rangle\phi\langle/a\rangle$ or $\langle \mathsf{lab}(var)\rangle\phi\langle/\mathsf{lab}(var)\rangle$, if we want this notation to be consistent with our requirement that variables bind always to single nodes. Conditions **every** $\$x$ **in** α **satisfies** ϕ are defined using **not** and **some** in the obvious way.

Let us now explain in what sense these definitions capture the usual logical relations. Recall that an XQ expression Q defines a Boolean query Q_{Bool} holding iff Q is nonempty. Using this definition we can see that the Boolean query corresponding to ϕ **and** ψ evaluates to true for a given binding of the free variables iff the Boolean queries corresponding to ϕ and to ψ both evaluate to true. Similarly, the other derived operators give Boolean queries with the semantics corresponding to their usual meaning in logic. In particular, the Boolean query that is defined will be independent of which particular label "a" is used in the definition of **not**. Let us note also that the restricted form of deep equality of $AtomXQ$ is sufficient to define negation, but this power is missing from $PosXQ$ (thus the name, suggesting "positive" XQ).

Our largest syntactic divergence from XQuery is that we assume if-expressions of the form "**if** ϕ **then** α" rather than "**if** ϕ **then** α **else** β": it is easy to see that the more general **if** can be simulated in any of our languages except for $PosXQ$, using:

$$\textbf{for } \$v \textbf{ in } \langle a\rangle\phi\langle/a\rangle \textbf{ return } \ (\textbf{if } \$v =_{deep} \ \langle a/\rangle \textbf{ then } \beta)$$

As with the simulation of **not**, the particular label that we use (above, a) is fixed but arbitrary above. Which particular label is used will not impact the sequence component of the answer, but will affect the forest component in a superficial way; if an a is used as the label in the simulation, the output forest will include the "temporary" tree rooted with a.

Faithfulness to XQuery standard. We discussed one syntactic deviation from the XQuery standard just above. Another syntactic distinction is that we have included the "immediate next and immediate previous" sibling axes previous-sibling and next-sibling, which are not part of XQuery. There is no reason to exclude them for our purposes, since all of our results hold in their presence.

In the semantics, we have made some simplifications over the standard, as given in [World Wide Web Consortium 2002]. Since our data model does not distinguish values of various types, hence the output types of our query language are simpler than XQuery's. To simplify further, we have avoided dealing with multiple output types in XQ entirely, and have assumed that all queries output a list of nodes, each of which is associated with a tree in the output forest. These assumptions are made to simplify syntax and semantics, but the expressiveness results would not change if the natural distinction among queries with output types node, sequence of node, and Boolean was introduced (e.g. with conditions *cond* producing a Boolean). In our semantics we also assume, in order to simplify the semantic function, that XQ variables always bind to single nodes rather than lists; our fragment assures this. We do not model the document() function of XQuery, which is used to bind variables to a document or a query result. Instead, we assume that there exist one or more initial free variables that are each bound to a node of the input forest.

XQuery has several modes of equality, including value equality, general equality, and node equality. We do not make a formal claim that these can be simulated in our framework, which would require us to formalize a relationship of the XQuery data model to ours. But we give an informal claim of the relationships below, which the reader can check against the semantics in Subsection 4.5 of [World Wide Web Consortium 2002]. XQuery's value equality, when restricting the compared items to expressions returning a single node (as our fragment guarantees), corresponds to the use of our $=_{deep}$. Our $=_{atomic}$ corresponds to a restricted use of value equality, used on some atomic value associated with a node: the most obvious way to use $=_{atomic}$ is to simulate value equality applied to the label of a node, but one can also simulate value equality on an attribute or PCDATA node. XQuery's general equality generalizes value equality to deal with sequences; in our language equality is restricted to variables, which bind to a single node, so there is no need for this distinction. XQuery's node equality corresponds to the use of our $=_{node}$.

Of course, our fragment, like those of many other research studies (and as in [Koch 2006]) deals with only a subset of the language. It deals with a simplified data model, in which (for example) no distinction is made among nodes of various kinds (attribute, element, comment etc.), and it also ignores the various primitive value-manipulation functions of XQuery (e.g. concatenation), many of which work only on nodes of a particular kind. The excluded features can have a dramatic impact on the expressiveness of the query language. We do not downplay their importance, but we believe that they are orthogonal to the issue of the expressiveness of the navigational and tree construction operators, which is the focus of this work.

2.3 Tree Structures and First-Order Interpretations

We now show how query languages on data forests can be defined via relational logics.

A common way to capture transformations using a logic is via the notion of *interpretation* in model theory (see e.g. [Gurevich and Shelah 1989]). One describes

the domain of the output by a formula to be evaluated on the input structure, and also describes each relation in the output structure via formulas that are to be evaluated on the input structure. For example, consider the transformation that takes a directed graph relation (V, E) and returns the "reversed digraph". This would be described by the interpretation consisting of two formulas $\phi_V(x) = true$, $\phi_E(x, y) = E(y, x)$. This states that the new graph has as its vertices all of the vertices of the input, and as its edges the reverse of the edges of the input. We can produce an output that is bigger than the input by allowing formulas with multiple variables in the interpretation. For example, consider the transformation that takes a directed graph (V, E) and returns the complete digraph on $|V|^2$ vertices. This would be described by the interpretation consisting of two formulas $\phi_V(x_1, x_2) = true$, $\phi_E(x_1, x_2; y_1, y_2) = true$. This states that the new graph has as its vertices all of the *pairs* of vertices of the input, and its edge relation is obtained by connecting every two pairs.

We need to use a variation of the standard definition of interpretation in our setting. The standard definition is on one-sorted structures, while we use the natural extension to many-sorted structures. We also allow the formulas in our interpretation to have additional "parameter variables", which correspond to the free variables in a query, and which we will consistently denote by $v_1 \dots v_n$ throughout this article.

In addition to extending the usual definition, we restrict our interpretations in several ways. The standard definition allows even equality in the new structure to be interpreted by a formula. In our interpretations, we require equality to be interpreted by equality on tuples. We also demand that the labels of the output must come only from the input labels or a fixed set of constants, and we do not produce "unused labels". The interpretations we produce from *XQ* queries will satisfy these restrictions.

Indexed forest interpretations. Interpretations can be defined over any logic; indeed over any query language. To interpret *AtomXQ* formulas, we will use formulas of *first-order logic*, built up using existential and universal quantifiers and boolean connectives. We use the usual notation and semantics for *FO* (e.g. [Ebbinghaus and Flum 1999]). We will deal with many-sorted first-order logic, in which each variable has a sort and every place in a relation or function symbol is associated with a sort. Given a first-order formula $\psi(\vec{v}; \vec{x})$, where $\vec{v}$ are a distinguished set of n free *parameter variables* and $\vec{x}$ are further free variables, over some signature σ, a structure $\mathcal{A}$, and elements $b_1 \dots b_n$ of $\mathcal{A}$, then $\psi(\mathcal{A}, \vec{b})$ denotes the relation $\{\vec{x} \mid \mathcal{A} \vDash \psi[b_1, \dots b_n, \vec{x}]\}$.

For a signature σ and number n, a $\sigma(v_1 \dots v_n)$ *structure* is a pair consisting of a σ-structure and an n-tuple of elements from the structure. We say that a logical formula is "over $\sigma(v_1 \dots v_n)$" if it uses symbols from σ and distinguished variables $v_1 \dots v_n$.

An *indexed forest interpretation over variables* $v_1 \dots v_n$ is given by a finite set of constants $\mathcal{C} = \mathcal{C}_{\mathsf{NI}} \cup \mathcal{C}_{\mathsf{Lab}}$ where $\mathcal{C}_{\mathsf{NI}}$ are constants of combined sort and $\mathcal{C}_{\mathsf{Lab}}$ are constants of label sort, integers $k, k' \geq 1$ (k is the *node arity* and k' is the *index arity*), a new unary node predicate $\mathsf{InputNode}$ (Informally, $\mathsf{InputNode}(x)$ says that "x is a node that is not one of the new constant symbols".), and formulas over signature $(\sigma_{nav,ind}, \mathsf{InputNode}, \mathcal{C})(v_1 \dots v_n)$

—$\phi_{\mathsf{Node}}(\vec{v}; x_1 \dots x_k)$,

—for each binary relation $R \in \{\mathsf{child}, \mathsf{descendant}, \mathsf{next\text{-}sibling}, \mathsf{following\text{-}sibling}\}$ a formula of the form $\phi_R(\vec{v}; x_1 \dots x_k; y_1 \dots y_k)$,

—$\phi_{\mathsf{Haslabel}}(\vec{v}; x_1 \dots x_k; z)$, where z is of sort Lab,

—two formulas $\phi_{\mathsf{Ind}}(\vec{v}; x'_1 \dots x'_{k'})$ and $\phi_{<_{\mathrm{ind}}}(\vec{v}; x'_1 \dots x'_{k'}; y'_1 \dots y'_{k'})$ describing the set of indexes and the ordering on indices in the new indexed forest, with again the partition being given with the formula.

—a formula $\phi_{\mathsf{ItemOf}}(\vec{v}; x'_1 \dots x'_{k'}; x_1 \dots x_k)$, where again the partition of the variables is part of the input. All of the variables will be of combined sort. This formula indicates which node an index is associated with.

Interpretations will define mappings from $\sigma_{nav,ind}(v_1 \dots v_n)$ structures to $\sigma_{nav,ind}$-structures. To explain the semantics of interpretations, we first have to say what it means to extend an input $\sigma_{nav,ind}(v_1 \dots v_n)$ structure using the constants in $\mathcal{C}$. Given indexed forest $\mathcal{F}_\#$ and an additional finite set of constants $\mathcal{C}$, each of which may be either of combined sort or label sort, we can form a corresponding $\sigma_{nav,ind} \cup \{\mathsf{InputNode}\}$ structure, denoted $\mathcal{F}_\# + \mathcal{C}$, by:

—interpreting the predicate IsNode by the nodes of $\mathcal{F}_\#$, unioned with a subset of the constants $\mathcal{C}$ of combined sort,

—interpreting predicate IsInd by the indices of $\mathcal{F}_\#$, unioned with all constants of $\mathcal{C}$ not in the interpretation of IsNode,

—interpreting the combined sort by the union of IsNode and IsInd, and interpreting the label sort by the set of labels found in any node of the underlying forest of $\mathcal{F}_\#$ unioned with the constants,

—interpreting $\mathsf{InputNode}$ by the set of nodes in $\mathcal{F}_\#$, and

—letting the other predicates ($\mathsf{ItemOf}, <_{\mathrm{ind}}, \dots$) hold as before, with their domains excluding all elements of $\mathcal{C}$.

Note that predicates of $\mathcal{F}_\# + \mathcal{C}$ are defined arbitrarily on the constants. The truth value of a formula on $\mathcal{F}_\# + \mathcal{C}$ is only well-defined if it is independent of such choices. The presence of the predicate $\mathsf{InputNode}$ will be a convenience in performing composition; note that this predicate is definable in $\mathcal{F}_\# + \mathcal{C}$ as the conjunction of inequalities with the constants.

We are now ready to give the semantics of an interpretation. Given constants $\mathcal{C} = \mathcal{C}_{\mathsf{NI}} \cup \mathcal{C}_{\mathsf{Lab}}$, we get a function taking a $\sigma_{nav,ind}(v_1 \dots v_n)$ structure $(\mathcal{F}_\#, b_1 \dots b_n)$ as input and returning a $\sigma_{nav,ind}$ structure. The $\sigma_{nav,ind}$ structure of the output is defined by evaluating the formulas in the interpretation over $\mathcal{F}_\# \cup \mathcal{C}$. So the output structure would be:

$\big(\mathsf{IsNode} = \{\vec{x} \in (\mathsf{NI}^{\mathcal{F}_\#} \cup \mathcal{C}_{\mathsf{NI}})^k : \mathcal{F}_\# \cup \mathcal{C} \models \phi_{\mathsf{Node}}(\vec{b}; \vec{x})\}$,

$\mathsf{IsInd} = \{\vec{x}' \in (\mathsf{NI}^{\mathcal{F}_\#} \cup \mathcal{C}_{\mathsf{NI}})^{k'} : \mathcal{F}_\# \cup \mathcal{C} \models \phi_{\mathsf{Ind}}(\vec{b}; \vec{x}')\}$,

$\mathsf{child} = \{(\vec{x}, \vec{y}) \in (\mathsf{NI}^{\mathcal{F}_\#} \cup \mathcal{C}_{\mathsf{NI}})^{2k} : \mathcal{F}_\# \cup \mathcal{C} \models \phi_{\mathsf{child}}(\vec{b}; \vec{x}; \vec{y})\}$,

$\mathsf{descendant} = \{(\vec{x}, \vec{y}) \in (\mathsf{NI}^{\mathcal{F}_\#} \cup \mathcal{C}_{\mathsf{NI}})^{2k} : \mathcal{F}_\# \cup \mathcal{C} \models \phi_{\mathsf{descendant}}(\vec{b}; \vec{x}; \vec{y})\}$,

$\mathsf{next\text{-}sibling} = \{(\vec{x}, \vec{y}) \in (\mathsf{NI}^{\mathcal{F}_\#} \cup \mathcal{C}_{\mathsf{NI}})^{2k} : \mathcal{F}_\# \cup \mathcal{C} \models \phi_{\mathsf{next\text{-}sibling}}(\vec{b}; \vec{x}; \vec{y})\}$,

$\mathsf{following\text{-}sibling} = \{(\vec{x}, \vec{y}) \in (\mathsf{NI}^{\mathcal{F}_\#} \cup \mathcal{C}_{\mathsf{NI}})^{2k} : \mathcal{F}_\# \cup \mathcal{C} \models \phi_{\mathsf{following\text{-}sibling}}(\vec{b}; \vec{x}; \vec{y})\}$,

$\mathsf{Haslabel} = \{(\vec{x}, z) \in (\mathsf{NI}^{\mathcal{F}_\#} \cup \mathcal{C}_{\mathsf{NI}})^k \times (\mathsf{Lab}(\mathcal{F}_\#) \cup \mathcal{C}_{\mathsf{Lab}}) : \mathcal{F}_\# \cup \mathcal{C} \models \phi_{\mathsf{Haslabel}}(\vec{b}; \vec{x}; z)\}$,

$$\mathsf{IsLab} = \{z \in \mathsf{Lab}^{\mathcal{F}_\#} \cup \mathcal{C}_{\mathsf{Lab}} : \mathcal{F}_\# \cup \mathcal{C} \models \exists \vec{x}\ \phi_{\mathsf{Node}}(\vec{b}; \vec{x}) \wedge \phi_{\mathsf{Haslabel}}(\vec{c}; \vec{x}; z)\},$$
$$\mathsf{ItemOf} = \{(\vec{m}, \vec{m}') \in (\mathsf{NI}^{\mathcal{F}_\#} \cup \mathcal{C}_{\mathsf{NI}})^{k'+k} : \mathcal{F}_\# \cup \mathcal{C} \models \phi_{\mathsf{ItemOf}}(\vec{b}; \vec{m}; \vec{m}')\},$$
$$<_{\mathrm{ind}} = \{(\vec{m}, \vec{m}') \in (\mathsf{NI}^{\mathcal{F}_\#} \cup \mathcal{C}_{\mathsf{NI}})^{2k'} : \mathcal{F}_\# \cup \mathcal{C} \models \phi_{<_{\mathrm{ind}}}(\vec{b}; \vec{m}; \vec{m}')\},)$$

That is, the set of k-tuples $\phi_{\mathsf{Node}}(\mathcal{F}_\# \cup \mathcal{C})$ is identified as the domain of the Node sort of the output structure. The child relation is interpreted by the set of pairs of k-tuples of nodes satisfying ϕ_{child}, and similarly for the other axes. The labeling relation is given by the labeling formula, and the labels are derived from the nodes and the labeling relation (i.e. they are the labels that occur on some node). The rest of the structure (indices, indexing relation, index ordering) is given by formulas in the same way.

Abbreviations and Extensions. We need to make a few extensions to the notion of interpretation above.

—**Abbreviating conventions.** To write out all the formulas of an interpretation is tedious, and we will require some conventions that make the output more succinct. The first convention is that once we have the formula ϕ_{Node} giving the nodes of the output forest in an interpretation, we will assume that any variables that interpret nodes in other formulas must satisfy ϕ_{Node}. Thus a formula $\phi_{\mathsf{child}}(\vec{v}; \vec{x}; \vec{y})$ is an abbreviation for the formula $\phi_{\mathsf{Node}}(\vec{v}; \vec{x}) \wedge \phi_{\mathsf{Node}}(\vec{v}; \vec{y}) \wedge \phi_{\mathsf{child}}(\vec{v}; \vec{x}; \vec{y})$. Likewise, $\phi_{\mathsf{ItemOf}}(\vec{v}; \vec{x}'; \vec{x})$ is implicitly conjoined with $\phi_{\mathsf{Ind}}(\vec{v}; \vec{x}')$ and $\phi_{\mathsf{Node}}(\vec{v}; \vec{x})$, while $\phi_{<_{\mathrm{ind}}}(\vec{v}; \vec{x}'; \vec{y}')$ is implicitly conjoined with $\phi_{\mathsf{Ind}}(\vec{v}; \vec{x}')$ and $\phi_{\mathsf{Ind}}(\vec{v}; \vec{y}')$.

—**Embedding Formula.** We will also include in an interpretation an additional *embedding formula* $\phi_{\mathsf{emb}}(\vec{v}; x; x_1 \ldots x_k)$. This predicate will give an isomorphism of the input into the output structure. That is, we will consider only interpretations such that for every input indexed forest, vector of nodes $\vec{b}$, and node a, there is a unique tuple $a_1 \ldots a_k$, where each a_i is a node or index of the input or a constant, such that $\phi_{\mathsf{Node}}(\vec{b}; a_1 \ldots a_k) \wedge \phi_{\mathsf{emb}}(\vec{b}; a; a_1 \ldots a_k)$ holds, and the corresponding function from the input forest to the output forest is a data forest isomorphism.

Thus, the result of applying an indexed forest interpretation to an input indexed forest is not only an output structure, but an isomorphism of the input into a substructure of the output. We need this additional structure to form interpretations compositionally. The embedding is required to deal with the fact that first-order logic and pure relational calculus do not have the ability to "create fresh nodes" as needed in a naive simulation of element construction (see the function *construct* used in the semantics of Figure 2). In our approach, we will deal with this as follows: when we have two subqueries Q_1 and Q_2 that we are translating, we will not be able to inductively assume that the new nodes returned by Q_1 are disjoint from those of Q_2. But we will record in our interpretation which are the newly-constructed nodes and which are in the copy of the input tree for each Q_i, using the formula ϕ_{emb}. Using this information, whenever we have a query Q that combines those two subqueries (e.g. $Q = Q_1 Q_2$), we will be able to identify the old elements in Q_1 and in Q_2 in the interpretation for Q while making the new nodes in Q_1 and Q_2 disjoint. Intuitively, we can think of an alternative semantics for XQ queries (but equivalent up to isomorphism) in which *construct* does not necessarily return different nodes each time, but where $(O_1, l_1) \uplus (O_2, l_2)$ is replaced with an operation that takes as an additional argu-

ment an embedding of the input forest into O_1 and O_2, and returns a data forest in which the images of the input are identified and the additional nodes are made disjoint (and similarly for $\biguplus$).

—**Restricting variables to range over nodes.** The use of one "combined sort" for nodes and indices will allow us to more easily compose interpretations, since when building up new queries in interpretations we will sometimes want to turn the indices of one structure into the nodes of another (as in the new-tree formation construction of XQ). However, it has a distinct notational inconvenience, in that restricting a variable or constant to range over nodes or indices will now require a formula, rather than being implicit in the sort. To preserve some of the advantage in brevity of the single-sort notation, we will say that a variable x or constant is "enforced to range over nodes" (resp. over indices) to mean that whenever x occurs in a formula, this formula is taken to be conjoined with $\mathsf{IsNode}(x)$ (resp. $\mathsf{IsInd}(x)$).

We will, by abuse of notation, use the term indexed forest interpretation to mean an interpretation including the additional embedding formula above. For such an interpretation we use $[I]$ to refer both to the semantic function given by the interpretation and to the function that returns the output indexed forest alone, without the embedding. Which of these we mean will be clear from the context.

We say that I defines an indexed forest query if for every $\mathcal{F}_\#$ and every $\vec{c}$, the structure $[I](\mathcal{F}_\#, \vec{c})$ is an indexed forest, with the nodes of $[I](\mathcal{F}_\#, \vec{c})$ a superset of the image of the nodes of $\mathcal{F}_\#$ under $\phi_{\mathsf{emb}}(\vec{c}; \mathcal{F}_\# \cup \mathcal{C})$, and $\phi_{\mathsf{emb}}(\vec{c}/\vec{v})$ defines a σ_{nav}-preserving embedding of $\mathcal{F}_\#$ into $[I](\mathcal{F}_\#, \vec{c})$.

EXAMPLE 2.3. *Consider the query Q from Example 2.2 once more. Q can be described by an indexed forest interpretation with constants $c_a, c_b, c_{\mathsf{emb}}, d$ of combined sort, node arity $k = 2$ and index arity $k' = 1$. The free variables in the formulas below will all be enforced to range over nodes. The forest structure is then given as follows:*

$$
\begin{aligned}
\phi_{\mathsf{Node}}(v_0; x_1, x_2) &:= (x_1 = c_a \wedge x_2 = c_a) \vee (x_1 = c_b \wedge x_2 = c_b) \vee \\
&\quad (x_1 = c_{b'} \wedge x_2 = c_{b'}) \vee \\
&\quad (x_1 = c_{\mathsf{emb}} \wedge \mathsf{InputNode}(x_2) \ \wedge \\
&\quad (\mathsf{descendant}(v_0, x_2) \vee v_0 = x_2)) \\
\phi_{\mathsf{child}}(x_1, x_2; y_1, y_2) &:= (x_1 = c_a \wedge x_2 = c_a \wedge y_1 = c_b \wedge y_2 = c_b) \vee \\
&\quad (x_1 = c_{\mathsf{emb}} \wedge y_1 = c_{\mathsf{emb}} \wedge \ \mathsf{child}(x_2, y_2)) \\
\phi_{\mathsf{descendant}}(x_1, x_2; y_1, y_2) &:= (x_1 = c_a \wedge x_2 = c_a \wedge y_1 = c_b \wedge y_2 = c_b) \vee \\
&\quad (x_1 = c_{\mathsf{emb}} \wedge y_1 = c_{\mathsf{emb}} \wedge \ \mathsf{descendant}(x_2, y_2)) \\
\phi_{\mathsf{next\text{-}sibling}}(x_1, x_2; y_1, y_2) &:= \mathsf{next\text{-}sibling}(x_2, y_2) \\
\phi_{\mathsf{following\text{-}sibling}}(x_1, x_2; y_1, y_2) &:= \mathsf{following\text{-}sibling}(x_2, y_2) \\
\phi_{\mathsf{Haslabel}}(x_1, x_2; z) &:= (x_2 = c_a \wedge z = a) \\
&\quad \vee \ (x_2 = c_{b'} \wedge z = b) \vee (x_2 = c_b \wedge z = b) \vee (x_1 = c_{\mathsf{emb}} \wedge \mathsf{Haslabel}(x_2, z))
\end{aligned}
$$

The nodes in the new tree are represented by the pairs (c_a, c_a), (c_b, c_b), and $(c_{b'}, c_{b'})$, and a node b in the input tree is represented by the node (c_{emb}, b) of the interpretation.

In the example, we have listed the variable v_0 as a free variable only for formulas where it is used. We have made use of several of the conventions mentioned above. The domain of the child relation is, again by convention, implicitly restricted to the pairs (x_1, x_2) and (y_1, y_2) that satisfy ϕ_{Node}.

We now give the index structure:

$$\begin{aligned}
\phi_{\mathsf{Ind}}(v_0, x_1) &:= (x_1 = d) \vee \mathsf{descendant}(v_0, x_1) \\
\phi_{\mathsf{ItemOf}}(v_0; x_1; x_1', x_2') &:= (x_1 \neq d \wedge x_1' = c_{\mathsf{emb}} \wedge x_1' = x_1 \wedge x_2' \neq v_0) \\
&\vee \ (x_1 = d \wedge x_1' = c_b \wedge x_2' = c_b) \\
\phi_{<_{\mathrm{ind}}}(x_1; x_1') &:= (x_1' = d \wedge x_1 \neq d) \\
&\vee \ (x_1 \neq d \wedge x_1' \neq d \wedge \mathsf{descendant}(x_1', x_2'))
\end{aligned}$$

The formulas above say that there are indices for every descendant of the input node, along with one additional index. For an index of the form b, where b is a descendant of the input node, the node it indexes is (c_{emb}, b): recall from above that this is the image of b in the output the document. The additional index points to the unique child node in the newly-created tree, represented by the tuple (c_b, c_b). The ordering puts the index of the new node last, and orders the nodes from the input document according to the descendant relation.

Finally, the embedding formula is $\phi_{\mathsf{emb}}(x; x_1, x_2) := (x_1 = c_{\mathsf{emb}} \wedge x_2 = x)$. That is, the embedding of the input into the output is just the function that pads an input node with the constant c_{emb}. □

EXAMPLE 2.4. *Consider the query $Q_\flat$ that flattens an arbitrary data tree $\mathcal{T}$ producing a depth-one tree with the same root and leaves as $\mathcal{T}$, where the leaves are arranged in document order. Below, we focus on the forest structure produced by $Q_\flat$ (omitting the index structure), which can be described by the following interpretation with $k = 1$:*

$$\begin{aligned}
\phi_{\mathsf{Node}}(x) &:= (\neg\exists w\ \mathsf{child}(w, x)) \vee (\neg\exists w\ \mathsf{child}(x, w)) \\
\phi_{\mathsf{child}}(x, y) &:= (\neg\exists w\ \mathsf{child}(w, x)) \wedge (\neg\exists w\ \mathsf{child}(y, w)) \\
\phi_{\mathsf{descendant}}(x, y) &:= \phi_{\mathsf{child}}(x, y) \\
\phi_{\mathsf{next\text{-}sibling}}(x, y) &:= \mathsf{following}(x, y) \wedge \neg\exists w(\phi_{\mathsf{Node}}(w) \wedge \\
&\qquad \mathsf{following}(x, w) \wedge \mathsf{following}(w, y)) \\
\phi_{\mathsf{following\text{-}sibling}}(x, y) &:= \mathsf{following}(x, y) \\
\phi_{\mathsf{Haslabel}}(x; z) &:= \mathsf{Haslabel}(x, z)
\end{aligned}$$

where the formula $\mathsf{following}(x, y)$ *is defined as* $\exists w_1\ \exists w_2\ (\mathsf{descendant}(w_1, x) \vee w_1 = x) \wedge (\mathsf{descendant}(w_2, y) \vee w_2 = y) \wedge \mathsf{following\text{-}sibling}(w_1, w_2)$.

The first formula says that the nodes of the output structure consist of the root and the leaves of the input structure. The second and third say that the child (and descendant) relation in the new structure connects leaves of the input with the root of the input, while the fourth and fifth say that the sibling relation on the output is the document-order relation on the input. The final formula says that the labeling of the output is inherited from the input. □

$FO(\mathbf{Cnt})$ and $FO(\mathbf{Cnt})$ interpretations. We now extend the discussion above

from first-order logic to first-order logic with an aggregate operator that can count tuples.

For a many-sorted vocabulary σ, the logic $FO(\mathbf{Cnt})(\sigma)$ (or $FO(\mathbf{Cnt})$, when σ is understood) is defined over variables having either one of the sorts of σ or the distinguished *number sort* $\mathcal{N}$. Atomic formulas include all atomic formulas of σ, while there are no atomic predicates or functions on variables of sort $\mathcal{N}$ (other than equality). Formulas are closed under Boolean operations and quantifiers $\exists x, \forall x, \exists i, \forall i$ for x of domain sort and i of number sort. We also have the counting quantifiers $\exists^{=i}\vec{x}, \forall^{=i}\vec{x}$ where i is of $\mathcal{N}$ sort and $\vec{x}$ is a tuple of variables. In a formula of the form $\exists^{=i}\vec{x}\ \phi(\vec{x}, \vec{y}, \vec{j})$, $\vec{x}$ becomes bound but i is still free.

The semantics of $FO(\mathbf{Cnt})$ formulas is given with respect to a σ-structure $\mathcal{A}$ and an environment mapping variables of any of the sorts of σ to elements of the corresponding domain in $\mathcal{A}$ and variables of sort $\mathcal{N}$ to non-negative integers. A formula $\exists^{=i}\vec{x}\ \phi(\vec{x}, \vec{y}, \vec{j})$ holds in structure $\mathcal{A}$ in an environment assigning $\vec{y}$ to $\vec{c}$, i to integer i_0 and $\vec{j}$ to integers $\vec{j}_0$ iff $|\{\vec{d} \in \mathcal{A} \mid (\mathcal{A}, \vec{c}\,, \vec{d}, \vec{j}_0) \models \phi\}|$ is exactly i_0.

One can show that quantifiers $\exists^{>i}$ and $\exists^{<i}$ are expressible in $FO(\mathbf{Cnt})$, on models (like the ones we consider) which have a total order definable on them. Within such models every definable set has cardinality equality to that of an initial segment of a product of the ordering, and inequality in cardinality is equivalent to containment of the corresponding initial segments. Further, the results of [Barrington et al. 1990] show that the power of $FO(\mathbf{Cnt})$ is not increased if arithmetic is permitted on variables of integer sort. However, note that FO extended by unary counting quantifiers $\exists^{=i}x$ is known to be strictly weaker than $FO(\mathbf{Cnt})$ [Schweikardt 2005]. The following is also known:

THEOREM 2.5 [BARRINGTON ET AL. 1990]. *Suppose σ contains a binary relation $<$, and $Lin(\sigma)$ is the class of finite σ-structures in which $<$ is interpreted as a linear-ordering on the domain. Then for any query Q over $Lin(\sigma)$ invariant under $(\sigma - \{<\})$-isomorphism, Q is in* TC^0 *iff Q is expressible in $FO(\mathbf{Cnt})(\sigma)$. That is, $FO(\mathbf{Cnt})$ captures* TC^0 *over ordered structures.*

We refer the reader to [Schweikardt 2005] for more on $FO(\mathbf{Cnt})$ and to [Johnson 1990] for a reference on complexity classes including the classes AC^0 and TC^0.

For variables $v_1 \ldots v_n$, a $\sigma_{nav,ind}(v_1 \ldots v_n)$ $FO(\mathbf{Cnt})$ *interpretation* is defined as with FO.

By AC^0 we refer to the class of languages recognizable by LOGSPACE-uniform families of circuits using and- and or-gates of unbounded fan-in of polynomial size and constant depth. By TC^0 we refer to the same class except that in addition so-called majority-gates are permitted, which compute "true" iff more than half the inputs are true. For details on the standard complexity classes as well as circuit complexity and the notion of uniformity the reader can consult [Johnson 1990].

The evaluation complexity of FO and $FO(\mathbf{Cnt})$ interpretations can be read off from classical results on the logics. Given the standard encoding of a σ_{nav} structure as a string [Immerman 1999],

PROPOSITION 2.6 CF. [IMMERMAN 1999]. *Every FO (resp., $FO(\mathbf{Cnt})$) interpretation over σ_{nav} can be evaluated in* AC^0 *(resp.,* TC^0*) data complexity and* PSPACE *combined complexity.*

2.4 Informal Summary of Main Results

Now that we have defined the top-down tree building languages and the relational logics we deal with, we are ready to state the contributions of this paper more precisely.

We start by looking at translations from XQuery fragments to first-order interpretations: We will show that every $AtomXQ$ query maps into a first-order interpretation. This follows from a more general result, which extends $AtomXQ$ into a language $AtomXQ^+$ that has an explicit composition operator. While $AtomXQ$ queries take as input a forest and output an indexed forest, $AtomXQ^+$ queries take indexed forests to indexed forests, and hence composing them makes sense. We will then show that the larger language $AtomXQ^+$ can be captured using an "indexed forest interpretation" – one that captures both the forest and the sequence component of the query output. This more general result is stated in Section 3. Its proof begins in Section 4, where we show that the basic queries in $AtomXQ^+$, other than the composition operator, can be captured by interpretations. The case of the composition operator is handled in Section 5, which shows that interpretations are closed under composition.

In Section 6, we will consider properties of the translation.

—We will show that $PosXQ$ queries map into "existential first-order interpretations" – an analog of conjunctive queries.

—We will show that the translation to relational logics requires exponential time in general (strictly speaking, we show that it cannot be done in polynomial time), but we identify a fragment of $AtomXQ$ (the "composition-free queries") for which it is polynomial time.

We then give analogous results for full XQ with deep equality (Section 7).

—We show that every XQ query maps into an $FO(\mathbf{Cnt})$ interpretation.

—Again the translation to $FO(\mathbf{Cnt})$ requires exponential time in general, but it is polynomial time on a subfragment.

A rough picture of the relationship between top-down query languages and logics can be found in Figure 3 in Section 8.

3. FROM XQ WITH ATOMIC EQUALITY TO FO: FORMALIZATION

We want to compare the expressiveness of XQ with interpretations. We notice a mismatch immediately, since XQ queries map forests to index-forests, while our interpretations map indexed forests to indexed forests. However an XQ expression can be associated with a transformation on indexed forests in the obvious way: the transformation will be the one that ignores the index component of the input indexed forest and applies the full semantics of Q to the remaining forest. Formally, we say that an indexed forest interpretation $I(v_1 \dots v_n)$ is *equivalent* to an XQ query Q with free variables $\$v_1 \dots \$v_n$ if $[I](\mathcal{F}_\#, b_1 \dots b_n)$ is isomorphic to $[\![Q]\!]_n(\mathcal{F}, b_1 \dots b_n)$ for every indexed forest $\mathcal{F}_\# = (\mathcal{F}, \vec{l})$ and for arbitrary nodes $b_1 \dots b_n \in \mathcal{F}$. (Thus, the choice of $\vec{l}$ has no influence on the result of $[I](\mathcal{F}_\#, b_1 \dots b_n)$.)

Our main result is the following.

THEOREM 3.1. *There is an* EXPTIME *function that maps every AtomXQ query Q with free variables $\$v_1 \ldots \$v_n$ to an FO indexed forest interpretation I such that Q is equivalent to $I(v_1 \ldots v_n)$.*

The proof of this result is rather involved, and will take up the next few sections.

3.1 Extending XQ to an indexed forest query language

As we saw, indexed forest interpretations are more general than XQ queries, since they can make use of the index structure of their input. We will now make up the gap somewhat, considering a query language XQ^+ on indexed forests that is richer than XQ, which will include queries that can make use of the index component of their input. All of the queries Q in XQ^+ will have the property that they are in a sense *monotone on the forest component of their input*: for every indexed forest $\mathcal{F}_\# = (\mathcal{F}, \vec{l})$, $[\![Q]\!]_n(\mathcal{F}_\#, \cdot) = (\mathcal{F}', \vec{l'})$ where $\mathcal{F}'$ extends $\mathcal{F}$ by the addition of new trees. The language XQ^+ extends XQ (considered as an indexed forest query language) by the following two additional rules:

$$\mathit{query} \;:=\; \mathit{query} \circ \mathit{query} \;\mid\; \textbf{union}\ \$v_{n+1}\ \textbf{over}\ \mathit{query}$$

The operator $\circ$ takes two indexed forest queries and returns the query representing their standard functional composition. That is, if α and β both have free variables contained in $\$v_1 \ldots \$v_n$, we define the semantics of the two operations as:

$$\begin{aligned}
[\![\beta \circ \alpha]\!]_n(\mathcal{F}_\#, e_1 \ldots e_n) \;&:=\; \text{let } \mathcal{F}_\#{}' = [\![\alpha]\!]_n(\mathcal{F}_\#, e_1 \ldots e_n) \\
&\qquad \text{in } [\![\beta]\!]_n(\mathcal{F}_\#{}', e_1 \ldots e_n) \\
[\![\textbf{union}\ \$v_{n+1}\ \textbf{over}\ \beta]\!]_n((\mathcal{F}, \vec{l}), \vec{e}) \;&:=\; \biguplus_{1 \le i \le |\vec{l}|} [\![\beta]\!]_{n+1}((\mathcal{F}, ()), \vec{e} \cdot \vec{l}_i)
\end{aligned}$$

One can show by induction that a) all queries in this language are monotone on their forest component, and b) all queries are well-defined. In the induction step for the composition operator, we know inductively that the forest component of $[\![\alpha]\!]_n((\mathcal{F}, \vec{l}), e_1 \ldots e_n)$ contains the input forest $\mathcal{F}$, and hence contains $e_1 \ldots e_n$ as well, hence the composition is well-defined. Monotonicity is easily shown to be preserved by the **union** operator.

Let $AtomXQ^+$ be the language XQ^+ above restricted so that deep equality $=_{deep}$ is used in tests only with constants, as in $AtomXQ$. Consider the following redefinition of **for** as an indexed forest query in $AtomXQ^+$:

$$\textbf{for}\ \$v_{n+1}\ \textbf{in}\ \alpha\ \textbf{return}\ \beta \;:=\; (\textbf{union}\ \$v_{n+1}\ \textbf{over}\ \beta) \;\circ\; \alpha.$$

That is, **for** is the composition of **union** and $\circ$. It is immediate that this definition is compatible with the definition of **for** as a query from forests to indexed forests. Thus, we can consider **for** to be a derived operator in $AtomXQ^+$. Thus in translating to interpretations, we can concentrate on translating the other $AtomXQ$ constructs, **union**, and the composition operator $\circ$.

We can now state our modified result, which clearly implies Theorem 3.1.

THEOREM 3.2. *There is an* EXPTIME *function that, given any $AtomXQ^+$ query Q with free variables $\$v_1 \dots \$v_n$, returns an indexed forest interpretation I over variables $v_1 \dots v_n$ that is equivalent as a query. That is, for every indexed forest $\mathcal{F}_\#$ and all bindings of $v_1 \dots v_n$ to nodes $b_1 \dots b_n$ in $\mathcal{F}_\#$, $[\![Q]\!]_n(\mathcal{F}_\#, b_1 \dots b_n)$ is isomorphic to $[I](\mathcal{F}_\#, b_1 \dots b_n)$ where the isomorphism restricted to $\mathcal{F}_\#$ agrees with ϕ_{emb}.*

4. PROOF OF THEOREM 3.2: THE BASIC CASES

We will now show how, from an $AtomXQ^+$ query α, we can construct the formulas ϕ^α_R that make up an indexed forest interpretation equivalent to α. We translate expressions to indexed forest interpretations inductively.

We begin by giving the construction for all cases except $\circ$ – we deal with this afterwards, since it is more complex. Note that the new tree formation and equality test queries produce a list component which is a singleton, hence the corresponding ordered structure in the output will be *degenerate*: there is only one index, so the formula ϕ_{Ind} holds only of a single constant, and this constant is mapped to a single tuple by ϕ_{ItemOf}.

4.1 New tree formation

Suppose α is the expression $\langle a \rangle \beta \langle / a \rangle$.

We inductively suppose we have an interpretation I^β for β. For example, for each navigational predicate R, the interpretation is given by formulas $\phi^\beta_R(\vec{v}; \vec{x}; \vec{y})$ with $\vec{x}, \vec{y}$ having arity k_β. We will assume $k_\beta > 1$ (the case $k_\beta = 1$ is similar but simpler). We will let k'_β be the arity associated with indices of β.

The arity for nodes of our interpretation will be $k = k_\beta + k'_\beta$. The set of constants used will be those of the interpretation for β, plus new constants c_a and c_{emb} of the combined sort. c_a will be used to represent the new root node labeled a and c_{emb} will be used to pad a representation of the input in the output. The interpretation will also include the label constant a.

The formulas giving the tree structure are as follows:

$$\phi^\alpha_{\mathsf{Node}}(\vec{v}; x_1 \dots x_k) := \Big(\bigwedge_i x_i = c_a\Big) \vee \Big(\Big(\bigwedge_{i \le k'_\beta} x_i = c_{\mathsf{emb}}\Big) \wedge \phi^\beta_{\mathsf{Node}}(\vec{v}; x_{k'_\beta+1} \dots x_k)\Big)$$
$$\vee \Big(\phi^\beta_{\mathsf{Ind}}(\vec{v}; x_1 \dots x_{k'_\beta}) \wedge \phi^\beta_{\mathsf{Node}}(\vec{v}; x_{k'_\beta+1} \dots x_k) \wedge$$
$$\exists r_1 \dots r_{k_\beta}\ \phi^\beta_{\mathsf{ItemOf}}(\vec{v}; x_1 \dots x_{k'_\beta}; \vec{r}) \wedge$$
$$\Big(\phi^\beta_{\mathsf{descendant}}(\vec{v}; \vec{r}; x_{k'_\beta+1} \dots x_k) \vee \bigwedge_{i \le k_\beta} r_i = x_{k'_\beta+i}\Big)\Big)$$

The above formula says that the nodes in the output are the union of

—a unique node marked by $c_a \dots c_a$,

—nodes produced by β, padded with a constant c_{emb} on each of the first k'_β variables to indicate that they are copied from the output of β, and

—a new subtree for each index from the index structure of the output of β. The nodes in the subtree for a given index $x_1 \dots x_{k'_\beta}$ consist of the node $b_{\vec{x}}$ indexed by $\vec{x}$ in the output of β, along with the descendants of $b_{\vec{x}}$.

$$\phi^{\alpha}_{\mathsf{child}}(\vec{v}; x_1 \dots x_k; y_1 \dots y_k) := \Big(\bigwedge_{1 \le i \le k} x_i = c_a \wedge \phi^{\beta}_{\mathsf{ItemOf}}(\vec{v}; y_1 \dots y_{k'_\beta}; y_{k'_\beta+1} \dots y_k) \Big)$$
$$\vee \Big(\Big(\bigwedge_{1 \le i \le k'_\beta} x_i = y_i = c_{\mathsf{emb}} \Big) \wedge \phi^{\beta}_{\mathsf{child}}(\vec{v}; x_{k'_\beta+1} \dots x_k; y_{k'_\beta+1} \dots y_k) \Big)$$
$$\vee \Big(\Big(\bigwedge_{1 \le i \le k'_\beta} x_i = y_i \Big) \wedge \phi^{\beta}_{\mathsf{child}}(\vec{v}; x_{k'_\beta+1} \dots x_k; y_{k'_\beta+1} \dots y_k) \Big)$$

The reader is reminded of our convention that this formula is implicitly conjoined with $\phi^{\alpha}_{\mathsf{Node}}(\vec{v}; x_1 \dots x_k)$ and $\phi^{\alpha}_{\mathsf{Node}}(\vec{v}; y_1 \dots y_k)$.

The above states that the nodes inherited from β have the same child relation as in β and the new node corresponding to c_a has as children all the copies of nodes indexed by β. The remaining new nodes are each associated with a pair consisting of a tuple representing an index and a tuple representing a node in β; two such pairs are in a parent/child relation only if the index portions are the same and the node portions are in a parent/child relation according to β.

The other axes are similar. Note that in an interpretation we are required to provide formulas describing the transitive axes (e.g., descendant) on the output, but that is easy to do in this case. In the case of descendant, we replace child by descendant in the formulas above, and in the second disjunct we require that the new node corresponding to c_a is above all the copied nodes.

The index structure consists of a single constant of combined sort, which indexes the tuple which is c_a on every component. The label structure is given as follows:

$$\phi^{\alpha}_{\mathsf{Haslabel}}(\vec{v}; x_1 \dots x_k; z) := (x_1 = c_a \wedge z = a) \vee \big(x_1 \neq c_a \wedge \phi^{\beta}_{\mathsf{Haslabel}}(\vec{v}; x_{k'_\beta+1} \dots x_k; z)\big)$$

The formula $\phi^{\alpha}_{\mathsf{emb}}(\vec{v}; x; x_1 \dots x_k)$ is

$$x_1 = c_{\mathsf{emb}} \wedge \mathsf{InputNode}(x) \wedge \phi^{\beta}_{\mathsf{emb}}(\vec{v}; x; x_{k'_\beta+1} \dots x_k).$$

Note that we could have replaced $x_1 = c_{\mathsf{emb}}$ in the above by $\bigwedge_{i \le k'_\beta} x_i = c_{\mathsf{emb}}$, since all nodes in the interpretation that have c_{emb} on the first component have c_{emb} on the first k'_β components.

Correctness: To see that this construction works, let $\mathcal{F}_{\#}$ be an indexed forest, $b_1 \dots b_n$ nodes of $\mathcal{F}_{\#}$, and ${\mathcal{F}_{\#}}^{\beta} = (\mathcal{F}^{\beta}, l_1 \dots l_s)$ the indexed forest resulting by applying β to $\mathcal{F}_{\#}$ with $b_1 \dots b_n$ interpreting $\$v_1 \dots \$v_n$, i.e., ${\mathcal{F}_{\#}}^{\beta} = [\![\beta]\!]_n(\mathcal{F}_{\#}, b_1 \dots b_n)$.

By induction, we can assume I^{β} correctly interprets β, and hence in particular there is a valid indexed forest ${\mathcal{F}_{\#}}^{\beta, I} = [I^{\beta}](\mathcal{F}_{\#}, b_1 \dots b_n)$: that is, the child and sibling relations of ${\mathcal{F}_{\#}}^{\beta, I}$ do form an ordered tree, the set of indices are linearly ordered by the $<_{\mathrm{ind}}$ relation, etc. Also by induction, we have a bijection H_β taking nodes of ${\mathcal{F}_{\#}}^{\beta}$ to nodes of ${\mathcal{F}_{\#}}^{\beta, I}$, where H_β extends $\phi^{\beta}_{\mathsf{emb}}$, and a bijection J_β taking the indices of ${\mathcal{F}_{\#}}^{\beta}$ to indices of ${\mathcal{F}_{\#}}^{\beta, I}$.

Let ${\mathcal{F}_{\#}}^{\alpha} = [\![\alpha]\!]_n(\mathcal{F}_{\#}, b_1 \dots b_n)$ and let ${\mathcal{F}_{\#}}^{\alpha, I} = [I^{\alpha}](\mathcal{F}_{\#}, b_1 \dots b_n)$ be the result of applying the interpretation I^{α} formed above to $\mathcal{F}_{\#}$. Note that the definition of I^{α} guarantees that the index component of ${\mathcal{F}_{\#}}^{\alpha}$ is a singleton, as is the case for ${\mathcal{F}_{\#}}^{\beta}$. We will give an isomorphism from the nodes of ${\mathcal{F}_{\#}}^{\alpha}$ to the nodes of ${\mathcal{F}_{\#}}^{\alpha, I}$. From

the definition of the constructor $\langle a\rangle\beta\langle/a\rangle$, we see that the nodes of ${\mathcal{F}_{\#}}^{\alpha}$ are either "inherited nodes" (elements of $\mathcal{F}^{\beta}$), "copied nodes" (copies of the ${\mathcal{F}_{\#}}^{\beta}$ nodes that are descendants of some list item l_i, with a distinct copy for each i), or the new root node labeled by a. Consider the mapping H_α on the nodes of ${\mathcal{F}_{\#}}^{\alpha}$ that takes:

—the new root of ${\mathcal{F}_{\#}}^{\alpha}$ to $(c_a)^k$, where for a constant c and integer i, c^i denotes the i-tuple all of whose elements are c.

—each node b of $\mathcal{F}^{\beta}$ to $c^{k'_\beta}_{\mathsf{emb}} \cdot H_\beta(b)$, where $\cdot$ denotes concatenation of tuples.

—a node b that is a copy of a node under l_i to $J_\beta(i) \cdot H_\beta(b)$

We claim that this mapping gives a bijection from the nodes of indexed forest ${\mathcal{F}_{\#}}^{\alpha}$ to the nodes of ${\mathcal{F}_{\#}}^{\alpha,I}$ preserving the forest structure (which, in particular implies that ${\mathcal{F}_{\#}}^{\alpha,I}$ is an indexed forest). We first argue that the formula $\phi^{\alpha}_{\mathsf{Node}}$ enforces that the nodes of $\mathcal{F}^{\alpha,I}$ are exactly the image of H_α. The first two disjuncts in $\phi^{\alpha}_{\mathsf{Node}}$ correspond to the first two cases of H_β above. The third disjunct in $\phi^{\alpha}_{\mathsf{Node}}$ states of a node in $\mathcal{F}^{\alpha,I}$ that it consists of an index output by β – say $J_\beta(i)$ for some i — concatenated with a node that is in the descendant-or-self relation to l_i. Hence the third disjunct of $\phi^{\alpha}_{\mathsf{Node}}$ corresponds to the third case within the definition of H_β above.

By induction we know that for each node b in $\mathcal{F}_{\#}$ there is a unique tuple $\vec{t}$ such that $\phi^{\beta}_{\mathsf{emb}}(b_1 \ldots b_n; b; \vec{t})$ holds. Letting F^{β}_{emb} be the function mapping a node to the corresponding tuple $\vec{t}$, we see that $\phi^{\alpha}_{\mathsf{emb}}$ also defines a function, taking a node b in the input to $c^{k'_\beta}_{\mathsf{emb}} \cdot F^{\beta}_{\mathsf{emb}}(b)$. Since by induction H_β extends the function corresponding to $\phi^{\beta}_{\mathsf{emb}}$, we see that H_α does indeed extend the function corresponding to $\phi^{\alpha}_{\mathsf{emb}}$.

To see that H_α preserves the child relation, note that the first disjunct of $\phi^{\alpha}_{\mathsf{child}}$, along with the correctness of H_β guarantee that H_α preserves the child relation on inherited nodes. The second disjunct of $\phi^{\alpha}_{\mathsf{child}}$ guarantees that the parent/child relation is preserved when the parent is the new root node, and the third disjunct of $\phi^{\alpha}_{\mathsf{child}}$ guarantees that the parent/child relation is preserved when both nodes are copied nodes.

The correctness of other axes is argued similarly.

Note that the definition of I^α guarantees that the index component of ${\mathcal{F}_{\#}}^{\alpha}$ is a singleton, and that this element indexes the tuple given by the constant function c_a. This constant tuple is the image under the isomorphism H_β of the root of ${\mathcal{F}_{\#}}^{\alpha}$, and hence must be the sole root of ${\mathcal{F}_{\#}}^{\alpha}$. From this we see that the mapping J_α taking the sole index of ${\mathcal{F}_{\#}}^{\alpha}$ to the sole index of ${\mathcal{F}_{\#}}^{\alpha,I}$ is an isomorphism of the index structure, and that in conjunction with H_α it preserves the ItemOf relation.

Alternative tree formation constructor: Suppose α is the expression

$$\langle \mathsf{lab}(\$v_i)\rangle\beta\langle/\mathsf{lab}(\$v_i)\rangle.$$

The interpretation is formed similarly to above, except that in the labeling formula, $z = a$ is replaced by $\mathsf{Haslabel}(v_i, z)$.

4.2 Sequential Composition

Suppose α is the expression $\beta\ \gamma$.

By padding β and γ we can assume they have the same free variables $v_1 \ldots v_n$. As before, we inductively suppose we have interpretations for β and γ. In particular,

for each axis relation R we have formulas ϕ_R^β and ϕ_R^γ.

First, suppose the the node arities are the same (i.e. $k_\gamma = k_\beta = k$), and similarly assume the interpretations have the same index arity k'. Let c_γ, c_β be new constant symbols of combined sort. Then the interpretation for α will have arity $k+1$ for nodes and $k'+1$ for indices. The tree structure will be given as follows:

$$\begin{aligned}\phi^\alpha_{\mathsf{Node}}(\vec{v}; x_1 \dots x_{k+1}) := \big(x_1 = c_\beta \wedge \phi^\beta_{\mathsf{Node}}(\vec{v}; x_2 \dots x_{k+1}) \wedge \mathsf{newtuple}^\beta(\vec{v}; x_2 \dots x_{k+1})\big) \\ \vee\ (x_1 = c_\gamma \wedge \phi^\gamma_{\mathsf{Node}}(\vec{v}; x_2 \dots x_{k+1}) \wedge \mathsf{newtuple}^\gamma(\vec{v}; x_2 \dots x_{k+1})) \\ \vee\ \Big(\exists w\ \mathsf{InputNode}(w) \wedge \bigwedge_{i \le k+1} x_i = w\Big)\end{aligned}$$

Recall that the formula $\mathsf{InputNode}(w)$ states that w is a node but not one of the constants. We let $\mathsf{newtuple}^\beta(\vec{v}; \vec{x})$ state that $\vec{x}$ represents a "new node" in the output of $\beta(\vec{v})$ – one that is not in the embedded image of the input forest; this can be expressed as $\phi^\beta_{\mathsf{Node}}(\vec{v}; \vec{x}) \wedge \neg\exists y\ \phi^\beta_{\mathsf{emb}}(\vec{v}; y; \vec{x})$. We will return to this formula later on in the paper.

This definition enforces that the nodes returned by α are the "new" nodes of β and γ (distinguished from each other by having a distinct constant on the first component) plus a copy of the nodes from the input tree. This is one of the places where we require the embedding of the input into the output to be available within the interpretation.

$$\begin{aligned}\phi^\alpha_{\mathsf{Haslabel}}(\vec{v}; x_1 \dots x_{k+1}; z) := (x_1 = c_\beta \wedge \phi^\beta_{\mathsf{Haslabel}}(\vec{v}; x_2 \dots x_{k+1}; z)) \vee \\ (x_1 = c_\gamma \wedge \phi^\gamma_{\mathsf{Haslabel}}(\vec{v}; x_2 \dots x_{k+1}; z)) \vee (\mathsf{InputNode}(x_1) \wedge \mathsf{Haslabel}(x_1, z))\end{aligned}$$

For any axis relation R we set:

$$\begin{aligned}\phi^\alpha_R(\vec{v}; x_1 \dots x_{k+1}; y_1 \dots y_{k+1}) := (\mathsf{InputNode}(x_1) \wedge \mathsf{InputNode}(y_1) \wedge R(x_1, y_1)) \\ \vee\ (x_1 = c_\beta \wedge y_1 = c_\beta \wedge \phi^\beta_R(\vec{v}; x_2 \dots x_{k+1}; y_2 \dots y_{k+1})) \\ \vee\ (x_1 = c_\gamma \wedge y_1 = c_\gamma \wedge \phi^\gamma_R(\vec{v}; x_2 \dots x_{k+1}; y_2 \dots y_{k+1}))\end{aligned}$$

For new constants c'_β, c'_γ, we set

$$\begin{aligned}\phi^\alpha_{\mathsf{Ind}}(\vec{v}; x'_1 \dots x'_{k'+1}) := (x'_1 = c'_\beta \wedge \phi^\beta_{\mathsf{Ind}}(\vec{v}; x'_2 \dots x'_{k'+1})) \vee \\ (x'_1 = c'_\gamma \wedge \phi^\gamma_{\mathsf{Ind}}(\vec{v}; x'_2 \dots x'_{k'+1}))\end{aligned}$$

$$\phi^{\alpha}_{\mathsf{ItemOf}}(\vec{v}; x'_1 \dots x'_{k'+1}; x_1 \dots x_{k+1}) :=$$
$$(x'_1 = c'_\beta \wedge x_1 = c_\beta \wedge \mathsf{newtuple}^{\beta}(\vec{v}; x_2 \dots x_{k+1}) \wedge \phi^{\beta}_{\mathsf{ItemOf}}(\vec{v}; x'_2 \dots x'_{k'+1}; x_2 \dots x_{k+1})) \vee$$
$$(x'_1 = c'_\gamma \wedge x_1 = c_\gamma \wedge \mathsf{newtuple}^{\gamma}(\vec{v}; x_2 \dots x_{k+1}) \wedge \phi^{\gamma}_{\mathsf{ItemOf}}(\vec{v}; x'_2 \dots x'_{k'+1}; x_2 \dots x_{k+1})) \vee$$
$$\big(x'_1 = c'_\beta \wedge \exists w_2 \dots w_{k+1}\ \phi^{\beta}_{\mathsf{ItemOf}}(\vec{v}; x'_2 \dots x'_{k'+1}; w_2 \dots w_{k+1}) \wedge$$
$$\phi^{\beta}_{\mathsf{emb}}(\vec{v}; x_2; w_2 \dots w_{k+1}) \wedge \bigwedge_{3 \le i \le k+1} x_i = x_2\big) \vee$$
$$\big(x'_1 = c'_\gamma \wedge \exists w_2 \dots w_{k+1}\ \phi^{\gamma}_{\mathsf{ItemOf}}(\vec{v}; x'_2 \dots x'_{k'+1}; w_2 \dots w_{k+1}) \wedge$$
$$\phi^{\gamma}_{\mathsf{emb}}(\vec{v}; x_2; w_2 \dots w_{k+1}) \wedge \bigwedge_{3 \le i \le k+1} x_i = x_2\big)$$

We see above that the node associated with an index is a bit subtle. If the index is associated with a new node of β – one that is not an input element — we just use the tuple output from the interpretation of β , marked with c_β. If the index is associated with an "old node" (i.e., in the input forest) of β, we find the corresponding input element using the embedding predicate of β, and then use α's representation of that input element, which is a constant function. Analogously for an index associated with a node of γ.

$$\phi^{\alpha}_{<_{\text{ind}}}(\vec{v}; x_1 \dots x_{k'+1}; x'_1 \dots x'_{k'+1}) :=$$
$$(x_1 = c'_\beta \wedge x'_1 = c'_\gamma) \vee (x_1 = c'_\beta \wedge x'_1 = c'_\beta \wedge \phi^{\beta}_{<_{\text{ind}}}(\vec{v}; x_1 \dots x_{k'}; x'_1 \dots x'_{k'})) \vee$$
$$(x_1 = c'_\gamma \wedge x'_1 = c'_\gamma \wedge \phi^{\gamma}_{<_{\text{ind}}}(\vec{v}; x_1 \dots x_{k'}; x'_1 \dots x'_{k'}))$$

$<_{\text{ind}}$ puts all the $k+1$ tuples beginning with c'_β (i.e. those that represent indices of β) before those beginning with c'_γ (which represent indices of γ). It then uses the ordering on β (resp. γ) for those $k'+1$ tuples beginning with c'_β, and similarly for c'_γ.

The formula $\phi^{\alpha}_{\mathsf{emb}}(\vec{v}; x; x_1 \dots x_{k+1})$ is $\bigwedge_{1 \le i \le k+1} x_i = x$.

Correctness: Given an input indexed forest $\mathcal{F}_\#$ and an assignment of nodes $b_1 \dots b_n$ to the variables, let ${\mathcal{F}_\#}^{\beta} = [\![\beta]\!]_n(\mathcal{F}_\#, \vec{b})$, ${\mathcal{F}_\#}^{\beta,I} = [I^\beta](\mathcal{F}_\#, \vec{b})$, ${\mathcal{F}_\#}^{\gamma} = [\![\gamma]\!]_n(\mathcal{F}_\#, \vec{b})$, and ${\mathcal{F}_\#}^{\gamma,I} = [I^\gamma](\mathcal{F}_\#, \vec{b})$. Let ${\mathcal{F}_\#}^{\alpha} = [\![\alpha]\!]_n(\mathcal{F}_\#, \vec{b})$ and ${\mathcal{F}_\#}^{\alpha,I} = [I^\alpha](\mathcal{F}_\#, \vec{b})$. We can assume by induction we have bijections H_β from the nodes of ${\mathcal{F}_\#}^{\beta}$ to the nodes of ${\mathcal{F}_\#}^{\beta,I}$ and J_β from the indexes of ${\mathcal{F}_\#}^{\beta}$ to the indexes of ${\mathcal{F}_\#}^{\beta,I}$, with H_β extending the function corresponding to $\phi^{\beta}_{\mathsf{emb}}$. Similarly we have H_γ extending $\phi^{\gamma}_{\mathsf{emb}}$ and J_γ taking ${\mathcal{F}_\#}^{\gamma}$ to ${\mathcal{F}_\#}^{\gamma,I}$. We consider the mapping H_α that takes a node b of ${\mathcal{F}_\#}^{\alpha}$ to:

—$c_\beta \cdot H_\beta(b)$ if b is in ${\mathcal{F}_\#}^{\beta} - \mathcal{F}_\#$
—$c_\gamma \cdot H_\gamma(b)$ if b is in ${\mathcal{F}_\#}^{\gamma} - \mathcal{F}_\#$
—b^{k+1} if b is in $\mathcal{F}_\#$.

Notice that the third case above guarantees that H_α extends $\phi^{\alpha}_{\mathsf{emb}}$. Let n_β be the number of indices in ${\mathcal{F}_\#}^{\beta}$, and similarly let n_γ be the number of indices in ${\mathcal{F}_\#}^{\gamma}$. We then let J_α map integer i to the image under J_β of the i^{th} index of ${\mathcal{F}_\#}^{\beta}$, if $i \le n_\beta$ and map $n_\beta + j$ to the image under J_γ of the j^{th} index of ${\mathcal{F}_\#}^{\gamma}$, for $j \le n_\gamma$. One can verify that H_α, J_α form an isomorphism of ${\mathcal{F}_\#}^{\alpha}$ to ${\mathcal{F}_\#}^{\alpha,I}$.

Remaining cases: If $k_\gamma \neq k_\beta$, say $k_\gamma - k_\beta = m$ where $m > 0$, we proceed similarly but with the tuple of elements beginning with c_β being padded out with c_β on the additional components.

4.3 Cases of navigation

Suppose α is the expression $\$v_i$. The node arity for the interpretation corresponding to α will be 1. We will have $\phi^\alpha_{\mathsf{Node}}(\vec{v}; x_1) := \mathsf{IsNode}(x_1)$ and $\phi^\alpha_R(\vec{v}; x_1; y_1) := R(x_1, y_1)$ for every axis relation R. The labeling structure will state that the label of node x_1 is just the label x_1 has as the input. That is, in this case the query just returns the tree and label structure copied from the input. The index structure consists of a single element (we use a new constant c for this element, although we could also have used v_i), and the indexing relations are given by $\phi^\alpha_{\mathsf{Ind}}(\vec{v}; x_1) := x_1 = c$ and $\phi^\alpha_{\mathsf{ItemOf}}(\vec{v}; x_1; x'_1) := x'_1 = v_i$. The embedding formula $\phi^\alpha_{\mathsf{emb}}$ is the identity.

Suppose α is $\$v_i/\mathsf{descendant} :: \nu$. The tree and label structure are the same as in the case $\alpha = \$v_i$ above. The index structure is

$$\begin{aligned} \phi^\alpha_{\mathsf{Ind}}(\vec{v}; x_1) &:= \mathsf{descendant}(v_i, x_1) \wedge \mathsf{Haslabel}(x_1, \nu) \\ \phi^\alpha_{\mathsf{ItemOf}}(\vec{v}; x_1; x'_1) &:= x'_1 = x_1 \end{aligned}$$

That is, we have an index for every descendant of v_i, and it is associated with the copy of the corresponding element from the input. The ordering on indices $\phi^\alpha_{<_{\mathrm{ind}}}$ is given by the document ordering on elements of the input, and the embedding formula is as with the case immediately above.

The general case of $\$v_i/axis :: \nu$ is done as above, with the only change being the obvious revision of $\phi^\alpha_{\mathsf{Ind}}(\vec{v}; x_1)$.

Correctness: Fix an input forest and an interpretation b_i of variable $\$v_i$. Let H_α be the identity on nodes. Since all of the data forest structure on the output is the same as that of the input, H_α preserves all the data forest structure. Because the document ordering is a linear-ordering, the output tree is always an indexed forest. Let the function J_α map the j^{th} index of the input to the j^{th} element in the axis relation to the interpretation of b_i. Then the definitions imply that H_α, J_α preserve the indexing relations.

4.4 Conditional

Suppose α is of the form **if** β **then** γ, and suppose γ is given by an interpretation with node arity k. Then the interpretation for α also has arity k and is given as $\phi^\alpha_{\mathsf{Node}}(\vec{v}; \vec{x}) := \mathsf{oldtuple}^\gamma(\vec{v}; \vec{x}) \vee (\exists \vec{x}'\ \phi^\beta_{\mathsf{Ind}}(\vec{v}; \vec{x}') \wedge \mathsf{newtuple}^\gamma(\vec{v}; \vec{x}))$.

The formula $\mathsf{oldtuple}^\gamma(\vec{v}; \vec{x})$ states that $\vec{x}$ is a representation in the output of the interpretation for $\beta(\vec{v})$ of a node from the input forest. This can be expressed as $\exists y\ \phi^\gamma_{\mathsf{emb}}(\vec{v}; y; \vec{x})$. The formula $\mathsf{newtuple}^\gamma$ is the negation of $\mathsf{oldtuple}^\gamma$.

The formula $\phi^\alpha_{\mathsf{Node}}$ states that we always return the copy of the input in γ, but return the new nodes of γ only when β returns a non-empty list component.

The formula ϕ^α_R is the same as ϕ^γ_R for all axis predicates R. Since this formula is (again, by our convention) conjoined with $\phi^\alpha_{\mathsf{Node}}$, it will automatically apply only to the "old nodes" (copies of the input) in the case that β returns an empty sequence component. The other formulas similarly state that if β returns a nonempty sequence component, the label and index predicates are the same as those of γ, while if β returns an empty sequence, the labels and labeling predicates are just those of the input, while the index structure is empty. These can easily be expressed in

FO. For example, we have $\phi^{\alpha}_{\mathsf{Ind}}(\vec{v};\vec{x}) = \exists \vec{x}'\ \phi^{\beta}_{\mathsf{Ind}}(\vec{v};\vec{x}') \wedge \phi^{\gamma}_{\mathsf{Ind}}(\vec{v};\vec{x})$. The embedding predicate is that of γ.

Correctness: Given input indexed forest $\mathcal{F}_{\#}$ and $b_1 \ldots b_n$ nodes of $\mathcal{F}_{\#}$, let $\mathcal{F}_{\#_\beta}$ be $\beta(\mathcal{F}_{\#}, \vec{b})$ and $\mathcal{F}_{\#_\beta}'$ be the result of applying the interpretation for β to $(\mathcal{F}_{\#}, \vec{b})$. Similarly, let $\mathcal{F}_{\#_\gamma}$ be $\gamma(\mathcal{F}_{\#}, \vec{b})$, $\mathcal{F}_{\#_\gamma}'$ be the result of applying the interpretation for γ to $(\mathcal{F}_{\#}, \vec{b})$, and let H_γ, J_γ be mappings from the nodes and indices of $\mathcal{F}_{\#_\gamma}$ to the nodes and indices of $\mathcal{F}_{\#_\gamma}'$. If the sequence component of $\mathcal{F}_{\#_\beta}$ is nonempty, we set H_α equal to H_γ and similarly set $J_\alpha = J_\gamma$. By correctness of the interpretation for β, we have that $\exists \vec{x}'\ \phi^{\beta}_{\mathsf{Ind}}(\vec{b};\vec{x}')$ holds in $\mathcal{F}_{\#}$, and thus the interpretation above reduces to F_γ. If the sequence component of $\mathcal{F}_{\#_\beta}$ has an empty sequence component, we set H_α to be the identity mapping and J_α to be the empty mapping. One can verify that the above interpretation reduces to the identity in this case.

4.5 Equality Tests

If condition β is of the form $\$v_1 =_{node} \$v_2$, then let c_Y be a new constant of combined sort (denoting "yes"), and "*yes*" be a label constant. The interpretation for β will have arity 1, with:

$$\begin{aligned}
\phi^{\beta}_{\mathsf{Node}}(\vec{v};x_1) &:= \mathsf{InputNode}(x_1) \vee (v_1 = v_2 \wedge x_1 = c_Y) \\
\phi^{\beta}_{\mathsf{Haslabel}}(\vec{v};x_1;z) &:= (\mathsf{InputNode}(x_1) \wedge \mathsf{Haslabel}(x_1,z)) \vee (x_1 = c_Y \wedge z = \text{“}yes\text{”}) \\
\phi^{\beta}_{\mathsf{ItemOf}}(\vec{v};x_1';x_1) &:= v_1 = v_2 \wedge x_1 = x_1'
\end{aligned}$$

The tree structure is simple, representing the disjoint union of the input and additionally (if $v_1 = v_2$) a tree with at most one node. The embedding is the identity.

If condition β is of the form $\$v_1 =_{deep} \langle a/\rangle$, then let c_Y be a new constant of combined sort, and "yes" a label constant. The interpretation for β will have arity 1, with:

$$\begin{aligned}
\phi^{\beta}_{\mathsf{Node}}(v_1;x_1) &:= \mathsf{InputNode}(x_1) \vee \\
&\qquad (\mathsf{Haslabel}(v_1,a) \wedge \neg(\exists y\ \mathsf{child}(v_1,y)) \wedge x_1 = c_Y) \\
\phi^{\beta}_{\mathsf{Haslabel}}(v_1;x_1;z) &:= (\mathsf{InputNode}(x_1) \wedge \mathsf{Haslabel}(x_1,z) \vee (x_1 = c_Y \wedge v = \text{“}yes\text{”})
\end{aligned}$$

The other formulas are similar.

If condition β is of the form $\$v_1 =_{atomic} \langle a/\rangle$, then let c_Y and "yes" be as for $=_{deep}$ above, and let $\phi^{\beta}_{\mathsf{Node}}(\vec{v};x_1) := \mathsf{InputNode}(x_1) \vee (\mathsf{Haslabel}(v_1,a) \wedge x_1 = N_Y)$. The other formulae are as above.

If condition β is of the form $\$v_1 =_{atomic} \$v_2$, then we do as above, except

$$\phi^{\beta}_{\mathsf{Node}}(\vec{v};x_1) := \mathsf{InputNode}(x_1) \vee (\exists z\ \mathsf{Haslabel}(v_1,z) \wedge \mathsf{Haslabel}(v_2,z) \wedge x_1 = c_Y).$$

Correctness is easy to see: note that the output indexed forest differs by at most one node from the input.

4.6 Other Cases

If expression α is of the form (), then let $\phi^{\alpha}_{\mathsf{Node}}(x_1) := \mathsf{InputNode}(x_1)$. The formula for ϕ_{Ind} is **false**. The rest of the forest structure is identical to the input.

Note that according to our semantics from Subsection 2.2, () does not return an empty output forest. It returns an indexed forest in which the underlying forest is the input forest, and the sequence component is empty. The interpretation has the same property.

4.7 Example

EXAMPLE 4.1. We illustrate part of the construction of the previous proof with a concrete example. Consider the XQ expression $\delta =$ **if** α **then** γ with $\alpha = \$v_1 =_{atomic} \langle a/\rangle$, $\gamma = \langle b\rangle\beta\langle/b\rangle$, and $\beta = ()$. We apply δ to an indexed forest structure consisting of a single a-labeled node N_0, with $\$v_1$ assigned to this node.

Then, the interpretation $(\phi^{\beta}_{\mathsf{Node}}, \dots)$ has one node N_0 labeled a, and an empty set of indices. $\phi^{\beta}_{\mathsf{emb}}$ maps N_0 to itself. $\phi^{\gamma}_{\mathsf{Node}} = \{(c_{\mathsf{emb}}, N_0), (c_b, c_b)\}$ with labels a, b, respectively. $\phi^{\gamma}_{\mathsf{child}} = \emptyset$. The index structure will have one index, which will index the node (c_b, c_b). $\phi^{\gamma}_{\mathsf{emb}}$ maps N_0 to (c_{emb}, N_0).

In the interpretation for α, $\phi^{\alpha}_{\mathsf{Node}} = \{N_0, c_Y\}$, with these nodes labeled with a and "*yes*" respectively. The forest structure will be a forest of two disconnected nodes. The index structure will have a single index (which can be taken to be c_Y), and this will index the node c_Y: $\phi^{\alpha}_{\mathsf{emb}}$ will map N_0 to itself.

Since there is a node in $\phi^{\alpha}_{\mathsf{Ind}}$, we have that $\phi^{\delta}_{\mathsf{Node}} = \{(c_{\mathsf{emb}}, N_0), (c_b, c_b)\}$ with labels a and b respectively, and $\phi^{\delta}_{\mathsf{child}} = \emptyset$. □

4.8 Translation of **union** queries

We now show that **union** can be captured.

THEOREM 4.2. *If $\beta(\$v_1 \dots \$v_{n+1})$ is a $AtomXQ^+$ query, and I^{β} is an indexed-forest interpretation equivalent to β. Then we can compute an indexed forest interpretation equivalent to $\alpha =$* **union** $\$v_{n+1}$ **over** β *.*

To prove this, suppose $\beta(\$v_1 \dots \$v_{n+1})$ is given by $I^{\beta} = (\phi^{\beta}_{\mathsf{Node}}(v_1 \dots v_n; x_1 \dots x_k), \dots)$ where the node arity is k and the index arity is k'. Then we can show that α is given by an indexed forest interpretation over $v_1 \dots v_n$ as follows, where the arity for the nodes is $k+1$ the arity for indices is $k'+1$.

Let c be an additional constant. The interpretation has

$$\begin{aligned}\phi_{\mathsf{Node}}(v_1 \dots v_n, x_1 \dots x_{k+1}) := (&\mathsf{IsInd}(x_1) \wedge\ \exists v_{n+1}\ \mathsf{IsNode}(v_{n+1}) \wedge \\ &\mathsf{ItemOf}(x_1, v_{n+1}) \wedge\ \mathsf{newtuple}^{\beta}(v_1 \dots v_n v_{n+1}; x_2 \dots x_{k+1})) \\ &\vee (x_1 = c \wedge \exists w\ \mathsf{InputNode}(w) \wedge \bigwedge_{2 \le i \le k+1} x_i = w)\end{aligned}$$

This states that a node is either: (i) a new tuple generated by I^{β} for some v_{n+1} in the sequence component of the input, where we distinguish v_{n+1}'s that correspond to different indices by including the associated index as the first element of the sequence, or (ii) an element of the input, padded by an additional constant. The definition guarantees that we get one copy of the input forest and disjoint copies of the new nodes produced for distinct $v_1 \dots v_{n+1}$. The reader should compare with

the definition of the interpretation for sequential composition $\beta\gamma$.

$$\phi_{\mathsf{Haslabel}}(v_1 \dots v_n; x_1 \dots x_k; z) := (x_1 = c \wedge \mathsf{Haslabel}(x_2, z)) \vee$$
$$(x_1 \neq c \wedge \exists v_{n+1}\ \mathsf{ItemOf}(x_1, v_{n+1}) \wedge\ \phi^{\beta}_{\mathsf{Haslabel}}(v_1 \dots v_{n+1}; x_2 \dots x_{k+1}; z))$$

For $\mathsf{axis} \in \{\mathsf{child}, \mathsf{descendant}, \mathsf{next\text{-}sibling}, \mathsf{following\text{-}sibling}\}$ we have

$$\phi_{\mathsf{axis}}(\vec{v}; x_1 \dots x_{k+1}; y_1 \dots y_{k+1}) := (x_1 = c \wedge y_1 = c \wedge \mathsf{axis}(x_2, y_2)) \vee$$
$$(x_1 \neq c \wedge y_1 = x_1 \wedge \exists v_{n+1}\ \mathsf{ItemOf}(x_1, v_{n+1}) \wedge$$
$$\phi^{\beta}_{\mathsf{axis}}(v_1 \dots v_{n+1}; x_2 \dots x_{k+1}; y_2 \dots y_{k+1})$$

$$\phi_{\mathsf{Ind}}(\vec{v}; x'_1 \dots x'_{k'+1}) := \exists v_{n+1}\ \mathsf{ItemOf}(x'_1, v_{n+1}) \wedge \phi^{\beta}_{\mathsf{Ind}}(v_1 \dots v_n v_{n+1}; x'_2 \dots x'_{k'+1})$$

$$\phi_{\mathsf{ItemOf}}(v_1 \dots v_n; x'_1 \dots x'_{k'+1}; x_1 \dots x_{k+1}) := \Big(x_1 = c \wedge \exists w\ \mathsf{InputNode}(w) \wedge$$
$$\Big(\bigwedge_{2 \leq i \leq k'+1} x_i = w\Big) \wedge \exists w_2 \dots w_{k+1} \exists v_{n+1} \mathsf{ItemOf}(x'_1, v_{n+1}) \wedge$$
$$\phi^{\beta}_{\mathsf{ItemOf}}(v_1 \dots v_{n+1}; x'_2 \dots x'_{k'+1}; w_2 \dots w_{k+1}) \wedge$$
$$\phi^{\beta}_{\mathsf{emb}}(v_1 \dots v_{n+1}; w; w_2 \dots w_{k+1})\Big) \vee$$
$$(x_1 = x'_1 \wedge \exists v_{n+1}\ \mathsf{ItemOf}(x'_1, v_{n+1}) \wedge$$
$$\phi^{\beta}_{\mathsf{ItemOf}}(v_1 \dots v_{n+1}; x'_2 \dots x'_{k'+1}; x_2 \dots x_{k+1}))$$

$$\phi_{<_{\mathrm{ind}}}(\vec{v}; x'_1 \dots \vec{x}'_{k'+1}; y'_1 \dots y'_{k'+1}) := <_{\mathrm{ind}}(x'_1, y'_1) \vee (x'_1 = y'_1 \wedge$$
$$\exists v_{n+1}\ \mathsf{ItemOf}(x'_1, v_{n+1}) \wedge \phi^{\beta}_{<_{\mathrm{ind}}}(v_1 \dots v_{n+1}; x'_2 \dots x'_{k'+1}; y'_2 \dots y'_{k'+1}))$$

The rest of the structure (e.g. ϕ_{emb}) is similar.

Correctness: Assume inductively that the interpretation I^{β} represents β. Let $I^{\alpha}(v_1 \dots v_n)$ be the interpretation formed above.

Fix an indexed forest $\mathcal{F}_{\#}$. We will take the underlying index set of $\mathcal{F}_{\#}$ to be an initial segment of the integers and denote the sequence component as $l_i : i \leq s$. Since our queries commute with isomorphism, and our result states only equivalence up to isomorphism, such an assumption is justified. Fix $b_1 \dots b_n$ interpreting $v_1 \dots v_n$. Let ${\mathcal{F}_{\#}}^{\alpha}$ be $[\![\textbf{union}\ \$v_{n+1}\ \textbf{over}\ \beta]\!]_n(\mathcal{F}_{\#}, b_1 \dots b_n)$, and let ${\mathcal{F}_{\#}}^{\alpha,I} = [I^{\alpha}]({\mathcal{F}_{\#}}, b_1 \dots b_n)$. We first describe a mapping H_{α} taking a node b of ${\mathcal{F}_{\#}}^{\alpha}$ to a node of ${\mathcal{F}_{\#}}^{\alpha,I}$. If b is a node of $\mathcal{F}_{\#}$, then $H_{\alpha}(b) = (c, b \dots b)$. If b is not a node of $\mathcal{F}_{\#}$, then there is a unique integer i such that $b \in [\![\beta]\!]_{n+1}(\mathcal{F}_{\#}, b_1 \dots b_n, l_i)$. By the induction hypothesis, there are mappings $H_{\beta,i}$ and $J_{\beta,i}$ from nodes and indices, respectively of $[\![\beta]\!]_{n+1}(\mathcal{F}_{\#}, b_1 \dots b_n, l_i)$ to nodes and indexes of $[I^{\beta}](\mathcal{F}_{\#}, b_1 \dots b_n, l_i)$ that preserve the indexed forest structure. Let $H_{\alpha}(b) = (i, y_1 \dots y_k)$, where $(y_1 \dots y_k) = H_{\beta,i}(b)$.

We now define a mapping J_{α} on indices of ${\mathcal{F}_{\#}}^{\alpha}$. For $i \leq s$, let $sz_{\beta,i}$ be the size of the index set of the indexed forest $[\![\beta]\!]_{n+1}(\mathcal{F}_{\#}, b_1 \dots b_n, l_i)$. Then by the definition of **union** we know that there are $\sum_i sz_{\beta,i}$ indices in ${\mathcal{F}_{\#}}^{\alpha}$, and each such index is thus of the form $\sum_{i' < i} sz_{\beta,i'} + p$ for some $i \leq s$ and some $p < sz_{\beta,i}$. We map such an index to $i' \cdot J_{\beta,i}(p)$. It is easily verified that H_{α}, J_{α} form an isomorphism from ${\mathcal{F}_{\#}}^{\alpha}$ to ${\mathcal{F}_{\#}}^{\alpha,I}$. $\square$

5. COMPOSITION OF INTERPRETATIONS AND THE COMPLETION OF THE PROOF OF THEOREM 3.2

It now remains to show that indexed forest interpretations are closed under the composition operator $\circ$. That is, we must show:

THEOREM 5.1. *Suppose that $I_1(v_1 \dots v_n)$ is an indexed forest interpretation equivalent to XQ^+ query $Q_1(\$v_1 \dots \$v_n)$, and $I_2(v_1 \dots v_n)$ is an indexed forest interpretation equivalent to XQ^+ query $Q_2(\$v_1 \dots \$v_n)$. Then we can form an indexed forest interpretation $I_3(v_1 \dots v_n)$ that is equivalent (for every input forest $\mathcal{F}_\#$ and binding $b_1 \dots b_n$ of the variables to nodes in $\mathcal{F}_\#$) to $Q_2 \circ Q_1(\$v_1 \dots \$v_n)$. The function that produces I_3 from I_1 and I_2 runs in time at most $|I_1| \cdot |I_2|$.*

Interpretations in general only mimic an XQ^+ query up to isomorphism. In particular, if we want to compositionally create an interpretation that returns the indexed forest $[\![Q_2]\!]_n([\![Q_1]\!]_n(\cdot, b_1 \dots b_n), b_1 \dots b_n)$, for some XQ^+ queries Q_1 and Q_2, we must consider that an interpretation I_1 equivalent to Q_1 will produce an output that does not contain $b_1 \dots b_n$, but only isomorphic copies of these nodes. To prove Theorem 5.1, we must thus show the following variant:

THEOREM 5.2. *There is a function that, under the assumptions of Theorem 5.1, produces an interpretation I_3 such that: for every input forest $\mathcal{F}_\#$ and binding $b_1 \dots b_n$ of the variables $v_1 \dots v_n$ to nodes in $\mathcal{F}_\#$, $[I_3](\mathcal{F}_\#, b_1 \dots b_n)$ is isomorphic to $[I_2]([I_1](\mathcal{F}_\#, b_1 \dots b_n), b'_1 \dots b'_n)$ where for $1 \leq i \leq n$, b'_i is the image of b_i under the embedding $\phi^{I_1}_{\mathsf{emb}}(b_1 \dots b_n)$.*

If we have the above, then Theorem 5.1 follows, since $[I_2]([I_1](\mathcal{F}_\#, b_1 \dots b_n), b'_1 \dots b'_n)$ is assumed isomorphic to $[\![Q_2]\!]_n([\![Q_1]\!]_n(\mathcal{F}_\#, b_1 \dots b_n), b_1 \dots b_n)$.

In this subsection, we give a function that composes an interpretation with an additional formula. This function will be used to prove Theorem 5.2 by allowing us to compose an interpretation I_1 with an interpretation I_2 "one formula of I_2 at a time".

We first give some notation for defining a relation within an indexed forest. Let R be a relation symbol of arity d. We consider a formula $\gamma(v_1 \dots v_n; x_1 \dots x_d)$ over vocabulary $\sigma_{nav,ind}$, where again the ';' represents a partition of the free variables into two classes. γ can mention constant symbols $\mathcal{C}'$. For simplicity, we will consider the case where the variables $x_1 \dots x_k$ are constrained to be nodes (by the IsNode predicate). Given such a γ, an indexed forest $\mathcal{F}_\#$, and nodes $b_1 \dots b_n$ in $\mathcal{F}_\#$, we let $R(\mathcal{F}_\#, \gamma, b_1, \dots b_n)$ be the structure for signature $\{R\}$ whose domain is $(\mathcal{F}_\# \cup \mathcal{C}')$ where R is interpreted by the sequence of d-tuples of nodes $\{q_1 \dots q_d \mid (\mathcal{F}_\# \cup \mathcal{C}', b_1 \dots b_n, q_1 \dots q_d) \models \gamma\}$. We will be applying this to the indexed forest $\mathcal{F}_\#{}'$ that results from applying an interpretation.

We now turn to a formula that "pulls back" a definable relation R of arity d on the output of an interpretation to the original domain. Let $I = \langle \phi_{\mathsf{Node}}(v_1 \dots v_n; x_1 \dots x_k), \dots \rangle$ be an indexed forest interpretation with node arity k and index arity l over $v_1 \dots v_n$, using constants $\mathcal{C}$. Suppose that $\gamma'(v_1 \dots v_n; x^1_1 \dots x^1_k, \dots, x^d_1 \dots x^d_k)$ is a formula in signature $\sigma_{nav,ind} \cup \mathcal{C}'$, where $x^j_i : j \leq d, i \leq k$ are variables of combined sort. Intuitively, since I produces an indexed forest output structure in which nodes are k-tuples, γ' determines a d-ary relation in the output of I. Formally, we define $R(\gamma', I)$ to be the interpretation I' for the signature $\{R\}$ based on the constants of I supplemented with any new constants of $\mathcal{C}'$, which takes as input $(\mathcal{F}_\#, b_1 \dots b_n)$ and out-

puts a structure whose domain will be $\{\vec{x} \in {\mathcal{F}_\#}^k : \phi^I_{\mathsf{Node}}(\vec{b};\vec{x}) \vee \bigvee_{c' \in \mathcal{C}'} \bigwedge_{i \leq k} x_i = c'\}$. That is, the domain will be the k-tuples corresponding to nodes of I unioned with the image of the constants under the mapping taking c to $c^k = (c \dots c)$. The relation R will be interpreted using γ': that is, as a set of k-tuples of d-tuples $\vec{m}^1, \dots, \vec{m}^k$ such that $\gamma'(b_1 \dots b_n; \vec{m}^1, \dots, \vec{m}^k)$.

Let $h(\mathcal{F}_\#, k, \mathcal{C}, \mathcal{C}')$ be the mapping from $(\mathsf{Node}(\mathcal{F}_\#) \cup \mathcal{C})^k \cup \mathcal{C}'$ to $(\mathsf{Node}(\mathcal{F}_\#) \cup \mathcal{C} \cup \mathcal{C}')^k$ that is the identity on $(\mathsf{Node}(\mathcal{F}_\#) \cup \mathcal{C})^k$ and takes $c' \in \mathcal{C}'$ to ${c'}^k$.

DEFINITION 5.3. Let I be an interpretation of node arity k based on constants $\mathcal{C}$, and $\gamma(v_1 \dots v_n; x_1 \dots x_d)$ a $\sigma_{nav,ind} \cup \{\mathsf{InputNode}\} \cup \mathcal{C}'$ formula, where the constants in $\mathcal{C}'$ are disjoint from those in $\mathcal{C}$ and the variables are constrained to be nodes. We say that another $\sigma_{nav,ind} \cup \{\mathsf{InputNode}\} \cup \mathcal{C} \cup \mathcal{C}'$ formula $\gamma'(v_1 \dots v_n; x_1^1 \dots x_k^1 \dots x_1^d \dots x_k^d)$ *captures the composition of* γ *and* I if for every indexed forest $\mathcal{F}_\#$ and every interpretation of $v_1 \dots v_n$ by $b_1 \dots b_n$ in $\mathcal{F}_\#$, the mapping $h(\mathcal{F}_\#, k, \mathcal{C}, \mathcal{C}')$ is an isomorphism from $[R(\gamma', I)](\mathcal{F}_\#, b_1 \dots b_n)$ to $R(\gamma, {\mathcal{F}_\#}', b_1' \dots b_n')$, where ${\mathcal{F}_\#}' = [I](\mathcal{F}_\#, b_1 \dots b_n)$, for $i \leq n$, b_i' is the image of b_i under ϕ^I_{emb} (i.e. b_i' is the node corresponding to the unique k-tuple $\vec{t}$ such that $\phi^I_{\mathsf{emb}}(\vec{b}; b_i; \vec{t})$ holds).

Note that both $[R(\gamma', I)](\mathcal{F}_\#, b_1 \dots b_n)$ and $R(\gamma, {\mathcal{F}_\#}', b_1' \dots b_n')$ are structures for a single d-ary relation.

Above, we have assumed, for simplicity, that all the arguments of the relation γ are of the same sort (e.g. all of node sort). The generalization of the definition to formulas $\gamma(x_1 \dots x_d)$ where not all x_i have the same sort is straightforward: for each variable x_i of γ of sort whose arity in I is j, γ' will have variables $x_1^i \dots x_j^i$.

We will show the following "Composition Lemma":

LEMMA 5.4. *There is a function* $\mathsf{Compose}(\gamma, I)$ *that returns a formula* γ' *capturing the composition of* γ *and* I*, which runs in time* $O(|\gamma| \cdot |I|)$.

The proof is by induction on the formula. We give the function Compose that provides the proof of the lemma. We assume that the interpretation I is given by $\langle \phi_{\mathsf{Node}}(v_1 \dots v_n; x_1 \dots x_k), \dots \rangle$. We let $\mathsf{InputNode}^+(x)$ abbreviate the formula $\mathsf{InputNode}(x) \vee \bigvee_{c \in \mathcal{C}} x = c$. For a sequence of variables $\vec{x}$ let $\mathsf{InputNode}^+(\vec{x}) = \bigwedge_i \mathsf{InputNode}^+(x_i)$. In an input structure of the form $\mathcal{F}_\# \cup \mathcal{C} \cup \mathcal{C}'$, $\mathsf{InputNode}^+(x)$ thus enforces that x is not one of the constants of $\mathcal{C}'$. We will normalize γ by assuming that variables v_i occur only in equalities – such a normalization can be done in linear time.

—$\mathsf{Compose}\big(\mathsf{IsNode}(x_i), I\big) := \phi_{\mathsf{Node}}(\vec{v}; x_1^1 \dots x_k^1)$

—$\mathsf{Compose}\big(\mathsf{IsInd}(x_i), I\big) := \phi_{\mathsf{Ind}}(\vec{v}; x_1^1 \dots x_k^1)$

—$\mathsf{Compose}\big(R(x_i, x_j), I\big) := \mathsf{InputNode}^+(\vec{x}^i) \wedge \mathsf{InputNode}^+(\vec{x}^j) \wedge \phi_R(\vec{v}; \vec{x}^i; \vec{x}^j)$, for R any of the navigational predicates

—$\mathsf{Compose}\big(\mathsf{InputNode}(x), I\big) := \mathsf{InputNode}^+(\vec{x})$

—$\mathsf{Compose}(x_i = x_j, I) := \bigwedge_{1 \leq m \leq k} x_m^i = x_m^j$
$\mathsf{Compose}(x_i = c, I) := \bigwedge_{1 \leq m \leq k} x_m^i = c$, where c is a constant in $\mathcal{C}'$.
$\mathsf{Compose}(v_i = v_j, I) := v_i = v_j$ and similarly for constant equalities.
$\mathsf{Compose}(x_i = v_j, I) := \phi_{\mathsf{emb}}(\vec{v}; v_j; \vec{x}^i)$

—$\mathsf{Compose}\big(\mathsf{Haslabel}(x_i, z), I\big) := \mathsf{InputNode}^+(\vec{x}^i) \wedge \phi_{\mathsf{Haslabel}}(\vec{v}; \vec{x}^i; z)$

—$\mathsf{Compose}\big(\mathsf{ItemOf}(x_i, x_j), I\big) := \mathsf{InputNode}^+(\vec{x}^i) \wedge \mathsf{InputNode}^+(\vec{x}^j) \wedge \phi_{\mathsf{ItemOf}}(\vec{v}; \vec{x}^i; \vec{x}^j)$

—$\mathsf{Compose}\big(<_{\mathrm{ind}}(x_i, x_j), I\big) := \mathsf{InputNode}^+(\vec{x}^i) \wedge \mathsf{InputNode}^+(\vec{x}^j) \wedge \phi_{<_{\mathrm{ind}}}(\vec{v}; \vec{x}^i; \vec{x}^j)$

—$\mathsf{Compose}\big((\exists x_i\ \eta(x_1 \dots x_d)), I\big) := \exists \vec{x}^i\ \mathsf{Compose}\big(\eta(x_1 \dots x_d), I\big)$
where $\mathsf{Compose}\big(\eta(x_1 \dots x_d), I\big)$ will have free variable $\vec{x}^1 \dots \vec{x}^d$,where $\vec{x}^i$ is a k-tuple of variables.

—$\mathsf{Compose}$ commutes through Boolean operations.

We now verify that $\mathsf{Compose}$ has the required property. We fix an indexed forest $\mathcal{F}_\#$ and $b_1 \dots b_n$ interpreting $v_1 \dots v_n$. Let $\mathcal{F}_\#{}' = [I](\mathcal{F}_\#, \vec{b})$, and b'_i be the image of b_i under ϕ^I_{emb} for $i \leq n$. Let h abbreviate $h(\mathcal{F}_\#, k, \mathcal{C}, \mathcal{C}')$. Recall that nodes of $[I](\mathcal{F}_\#, b_1 \dots b_n)$ are k-tuples $(n_1 \dots n_k)$ from $\mathcal{F}_\# \cup \mathcal{C}$. For such a k-tuple, h is the identity; for the additional new elements in the domain of $R(\gamma)([I](\mathcal{F}_\#))$ – the constants c in $\mathcal{C}'$ – $h(c)$ is the k-tuple $(c, \dots c)$.

We verify that h is the required isomorphism, by induction on the structure of γ. For brevity, we execute only two of the cases.

—Base cases for axes. Let $\gamma(x_1, x_2) = \mathsf{axis}(x_1, x_2)$. We have to verify that for m_1, m_2 nodes of $R(\gamma, \mathcal{F}_\#{}', \vec{b'})$ we have

$$R(\gamma, \mathcal{F}_\#{}', \vec{b'}) \models R(m_1, m_2) \leftrightarrow [R(\gamma', I)](\mathcal{F}_\#, \vec{b}) \models R(h(m_1), h(m_2)).$$

Unwinding the definition of γ, γ', $R(\gamma, \mathcal{F}_\#{}', \vec{b'})$ and $[R(\gamma', I)]$, we see that this is true iff the following holds for all nodes m_1, m_2 of $\mathcal{F}_\#{}' \cup \mathcal{C} \cup \mathcal{C}'$:

$$\mathcal{F}_\#{}' \cup \mathcal{C}' \models \mathsf{axis}(m_1, m_2) \leftrightarrow$$
$$\mathcal{F}_\# \cup \mathcal{C} \cup \mathcal{C}' \models \mathsf{InputNode}^+(h(m_1), h(m_2)) \wedge \phi^I_{\mathsf{axis}}(\vec{b}; h(m_1); h(m_2)).$$

If m_1 or m_2 is in $\mathcal{C}'$, then $\mathsf{axis}(m_1, m_2)$ cannot hold, since no axis relations hold of the constants (the axes in our signature do not include self), and neither does $\mathsf{InputNode}^+(h(m_1), h(m_2))$ hold, since this would require the components of both arguments to be in $\mathcal{F}_\# \cup \mathcal{C}$. So it suffices to verify the equivalence for $m_1, m_2 \in \mathcal{F}_\# \cup \mathcal{C}$. But there h is just the identity: that is, it maps m_1 to the k-tuple $\vec{m}_1$ that represents it, and similarly for m_2. So we need to show that $\mathcal{F}_\#{}' \models \mathsf{axis}(m_1, m_2) \leftrightarrow \mathcal{F}_\# \cup \mathcal{C} \models \phi^I_{\mathsf{axis}}(\vec{b}; \vec{m}_1; \vec{m}_2)$ But this holds because ϕ^I_{axis} is the interpretation of axis in $[I]$.

—Base cases for equality.
Consider the case of formula $\gamma(v_1; x_1) := v_1 = x_1$ We have to verify that for m_1 a node of $R(\gamma, \mathcal{F}_\#{}', \vec{b'})$ $R(\gamma, \mathcal{F}_\#{}', \vec{b'}) \models R(m_1) \leftrightarrow [R(\gamma', I)](\mathcal{F}_\#, \vec{b}) \models R(h(m_1))$. Unwinding the definitions of the two structures, we see that we have to verify that for all nodes m_1 of $\mathcal{F}_\#{}' \cup \mathcal{C}'$, $\mathcal{F}_\#{}' \cup \mathcal{C}' \models b'_1 = m_1 \leftrightarrow \mathcal{F}_\# \cup \mathcal{C} \cup \mathcal{C}' \models \phi^I_{\mathsf{emb}}(\vec{b}; b_1; h(m_1))$. For $m_1 \in \mathcal{C}'$, both sides of the equivalence are false, so the equivalence holds. For $m_1 \in \mathcal{F}_\#{}'$, we need to verify that $\mathcal{F}_\#{}' \cup \mathcal{C}' \models b'_1 = m_1 \leftrightarrow \mathcal{F}_\# \cup \mathcal{C} \cup \mathcal{C}' \models \phi^I_{\mathsf{emb}}(\vec{b}; b_1; \vec{m}_1)$ where $\vec{m}_1$ represents m_1 in $\mathcal{F}_\#{}'$. But this holds by the definition of b'_1.
Now consider $\gamma := x_1 = c$. We need to show that $[I](\mathcal{F}_\#, \vec{b}) + R(\gamma) \models R(m_1) \leftrightarrow [I + R(\gamma')](\mathcal{F}_\#, \vec{b}) \models R(m_2)$. This unwinds to $[I](\mathcal{F}_\#, \vec{b}) \models m_1 = c \leftrightarrow [I + R(\gamma')](\mathcal{F}_\#, \vec{b}) \models \bigwedge_i m^i_1 = c$. But this again follows from the definition of $[I]$.
The other cases are similar.

We are now ready for the proof of Theorem 5.2. Suppose that $I_1(v_1 \dots v_n)$ and $I_2(v_1 \dots v_n)$ are indexed forest interpretations equivalent to Q_1 and Q_2 respectively. Suppose that $I_1(v_1 \dots v_n)$ is an indexed forest interpretation, and $I_2(v_1 \dots v_n)$ is an indexed forest interpretation.

For each symbol β of $\sigma_{nav,ind}$, we let $\phi_\beta^{I_3}$ be the formula that captures the composition of $\phi_{\mathsf{Node}}^{I_2}$ and I_1 (as produced by Lemma 5.4 in the case where all free variables of $\phi_\beta^{I_3}$ are constrained to be nodes; in the more general case, we apply the obvious generalization of the Lemma).

For each relation R of $\sigma_{nav,ind}$ we let $\phi_R^{I_3}$ be the formula that captures the composition of $\phi_R^{I_2}$ and I_1, as produced by Lemma 5.4. This is the required interpretation, since Lemma 5.4 produces a mapping h that is an isomorphism of the output of the composition with the output of this interpretation (note that the mapping h given by the Lemma does not depend on the formula being composed). □

6. REFINEMENTS AND APPLICATIONS OF THE TRANSLATION FROM $ATOMXQ$ TO LOGIC

Output-node semantics. Recall from Subsection 2.2 that Q_{out} returns the forest consisting of the descendants of nodes in the list component returned by Q. It is easy to revise Theorem 3.1 to preserve the output node semantics. A *forest interpretation* is defined similarly to an index-forest interpretation, but without the index predicates being used in any formulas and without the index formulas and embedding predicate. Then we have:

COROLLARY 6.1. *There is an* EXPTIME *function that maps every AtomXQ query Q with free variables $\$v_1 \dots \$v_n$ to an FO forest interpretation I such that Q_{out} is equivalent to I.*

Proof: Given an indexed forest interpretation I_0 that captures the full semantics, we can easily obtain a forest interpretation I in the sense of subsection 2.3 that captures the output-node semantics of Q. We do this by

—replacing predicates IsInd and ItemOf by false and IsNode by true in any formulas of I_0 – this is clearly justified for inputs where the index structure is empty, which can be taken to be the case,

—dropping all the formulas that interpret the index structure, and

—relativizing the σ_{nav} formulas to the nodes $\vec{x}$ that occur in I_0 inside subtrees of indexed nodes, by conjoining with

$$\exists \vec{w}\vec{w}'\ \phi_{\mathsf{IsInd}}(\vec{v},\vec{w}') \wedge \phi_{\mathsf{ItemOf}}(\vec{v},\vec{w}',\vec{w}) \wedge (\vec{x}=\vec{w} \vee \phi_{\mathsf{descendant}}(\vec{v},\vec{w},\vec{x})).$$

Fragments of $AtomXQ$. We now consider sublanguages of $AtomXQ$, with the goal of seeing what subset of FO they map to. We start with a simple case, that of $PosXQ$. Let $\exists FO$ be the fragment of FO built up from the atomic formulas of σ_{nav} and (in)equalities between constants and variables using positive Boolean operators and existential quantification. As the name implies, $PosXQ$ are positive: they translate into interpretations in which general negation is not needed. A disclaimer is needed here about sibling ordering: in XQ, the sibling ordering is done by relativizing document order to the returned nodes. This requires the use of negation (see Example 2.4). One may worry that when the sibling axis is then

used compositionally, this can lead to negation elsewhere in the interpretation. However, we can show:

PROPOSITION 6.2. *For every PosXQ expression Q that does not use the sibling axes, one can find (in* EXPTIME*) a forest interpretation I equivalent to Q_{out} for which all formulas are in $\exists FO$, other than those for the sibling axes.*

PROOF. The translation that witnesses Theorem 3.2 introduces negation only in a few places.

One is the formula newtuple, which we have defined using negation on ϕ_{emb}. However, we can arrange that for any tuple satisfying ϕ_{Node}, whether or not it is in the image of ϕ_{emb} can be checked with a positive boolean combination of equalities and inequalities with a fixed set of constants on the first component of the tuple – hence newtuple and oldtuple are in fact positive formulas. We can arrange this for constructs other than composition and **union** by construction. In the prior constructions we have sometimes used the product of the identity mapping for ϕ_{emb} (e.g. for sequential composition); instead we could add a new constant and use this as a marker in the first component. For composition, the embedding is the result of applying the composition algorithm to a prior interpretation I and the embedding formula ϕ_{emb}; one can check that the composition of an equality of a variable with a constant $x_1 = c$ is simply the conjunction of $\phi^I_{\mathsf{Node}}(\vec{v} : \vec{x}^1)$ and a conjunction of constant equalities. The conjunct $\phi^I_{\mathsf{Node}}(\vec{v} : \vec{x}^1)$ can be omitted, since the node formula of the composed interpretation will imply $\phi^I_{\mathsf{Node}}(\vec{v} : \vec{x}^1)$, and the variables representing nodes in ϕ_{emb} are automatically relativized to the nodes of the composed interpretation. Hence the embedding formula ϕ_{emb} will again be a positive boolean combination of equalities and inequalities.

Note that the operator Compose does not introduce negation. The only other places where a general negation operator occurs are comparisons with $=_{deep}$ and the construction of the sibling axis of the output, which is computed using relativized document order (see the formula $\phi_{\mathsf{next\text{-}sibling}}$ in Example 2.4). General $=_{deep}$ comparisons are not allowed in $PosXQ$, so this is not an issue. Notice that for queries in which the sibling ordering is not used explicitly in an axis, no formulas in the corresponding interpretations will inductively depend on the sibling axes of subformulas, other than index ordering formulas and sibling axes formulas. The formula for the index ordering will not occur in the final forest interpretation, since (by definition) a forest interpretation outputs only the forest structure. The index ordering in turn is only used to form the sibling axis in the new tree construction operator. Thus, if we exclude the formulas for the sibling-axes (as we do by hypothesis here), we can show by induction that in the construction of Theorem 3.2 the function Compose will never be called on a formula that uses negation. Hence we can see that for an interpretation of sibling-free, $PosXQ$ formulas, we can translate into indexed forest interpretations represented with only $\exists FO$ formulas.

Finally, we need to check not only that the formulas produced by the algorithm of Theorem 3.2 applied to $PosXQ$ queries are in $\exists FO$, but that $\exists FO$ is preserved in the step of going from an interpretation for the general semantics to an interpretation for the output-node semantics. However, $\exists FO$ is closed under relativizing formulas to the nodes within the output list, so this last transformation is not a problem. □

Theorem 3.1 gives an ExpTime translation, due to the need to "inline" the same expression multiple times when translating the **for** clause. It is thus not surprising that XQ queries can be exponentially more succinct than first-order queries. For the translation from $PosXQ$ to existential FO interpretations, we can show a stronger result: there is an exponential blow-up even for boolean queries. Whether *Boolean* $AtomXQ$ queries can be translated in PTime to FO remains open. This is similar to the question of the existence of a PTime translation from nested relational queries on flat structures to relational queries, which has been an open problem in complex-valued databases for some time [Van den Bussche 2005].

Consider any reasonable measure of the size of queries and interpretations, such as the number of bytes required to write them in latex.

Theorem 6.3. *There is no polynomial time function translating $AtomXQ$ queries to FO interpretations. For a $PosXQ^-$ query, there is no PTime translation to $\exists FO$, even for Boolean queries.*

We now give the proof of Theorem 6.3.

We first prove the bound for $AtomXQ$, which uses an argument well-known in functional programming and nested-relational databases. Consider the sequence of queries Q_n given as

$Q_0 := \langle B/\rangle\langle B/\rangle$
$Q_{n+1} := \langle A\rangle$
let $\$v_0$ $:= \langle A\rangle Q_n\langle/A\rangle$
for $\$v_1$ **in** $\$v_0$/child :: $*$
return (**for** $\$v_2$ **in** $\$v_0$/child :: $*$**return** ($\$v_1$ $\$v_2$))
$\langle/A\rangle$

Since the number of nodes returned by Q_n is the square of the number returned by Q_{n-1}, one can verify that Q_n produces an output tree of doubly exponential size in n given a tree with one node as an input. However, an FO interpretation with formulas of arity k using n node constants can produce given an input of size 1, an output of size at most $(n+1)^k$, since nodes in the output are k-tuples of nodes in the input and constants. Hence an interpretation obtained by translation of Q_n bounded in size by polynomial $p(n)$ will be able to produce an output of size at most $(p(n)+1)^{p(n)}$ on an input of size 1. In the limit, this clearly cannot equal the size of the output of Q_n. □

For the second lower bound, we consider the following sequence of queries Q_n: If all labels are distinct, Q_1 will return, for every pair of nodes n_1, n_2 such that there is a vertical path of length 2 from n_1 to n_2, the tree $\langle A\rangle\langle\mathsf{lab}(n_1)/\rangle\langle\mathsf{lab}(n_2)/\rangle\langle/A\rangle$, enclosed within a root node $\langle R/\rangle$.

$Q_{n+1} := \langle R\rangle$ **let** ($\$v$ $:= Q_n$)
for $\$x_1$ **in** $\$v$/descendant :: A **return**
for $\$x_2$ **in** $\$v$/descendant :: A **return**
for $\$m_1$ **in** $\$x_1$/descendant/previous-sibling :: $*$ **return**
for $\$m_2$ **in** $\$x_1$/descendant/following-sibling :: $*$ **return**
for $\$n_1$ **in** $\$x_2$/descendant/previous-sibling :: $*$ **return**
for $\$n_2$ **in** $\$x_2$/descendant/following-sibling :: $*$ **return**

$$\textbf{if } \mathsf{lab}(\$m_2) = \mathsf{lab}(\$n_1) \textbf{ then} \langle A\rangle\langle \mathsf{lab}(m_1)/\rangle\langle \mathsf{lab}(n_2)/\rangle\langle /A\rangle \langle /R\rangle$$

Above we have made some abbreviations: For instance, we use sequences of axes descendant/previous-sibling instead of the equivalent $PosXQ$ expression. We are also using the inverse axis previous-sibling, while in the definition of XQ, we restrict the axes by default to be the forward axes. But clearly the inverse axes are definable in $PosXQ$.

One can verify that, in a tree in which all labels are distinct, Q_n returns a representation of all pairs of nodes that are 2^n apart. From Q_n we can form a Boolean query checking whether the depth of such a tree is at least 2^n. However, an $\exists FO$ formula with $p(n)$ free variables can not distinguish between a chain of depth 2^{n-1} and a chain of depth 2^n. □

We can avoid the blow-up by restricting queries in the source of the translation to be in a special form. The language *composition-free XQ* is formed by restricting the **for** $\$v$ **in** Q **return** Q' construct so that Q must be of the form $\$v/axis :: \nu$, and adding conditions of the form $Q_1 =_{deep} Q_2$, where the Q_i are restricted to return singleton lists: i.e. they must be of the form $\langle a\rangle\beta\langle/a\rangle, \langle \mathsf{lab}(var)\rangle\beta\langle/\mathsf{lab}(var)\rangle$, or var. The semantics is the same as that given by assigning variables v_1 to Q_1 and v_2 to Q_2 and checking if $v_1 =_{deep} v_2$: hence this is expressible in XQ.

The idea is that we do not have the ability to assign a variable to a query result (or to iterate over a query result) — the ability to do this is what allows XQ to reuse subquery results several times, leading to the exponential blow-up. The language *composition-free AtomXQ* is the language formed from $AtomXQ$ by restricting **for** as above, and adding the construct **not** Q with semantics given via **for** $\$v$ **in** $\langle a\rangle Q\langle/a\rangle$ **return** $\$v =_{deep} \langle a/\rangle$. In addition we add $\$v =_{atomic} \langle a/\rangle$ for a a label, since we can no longer simulate this using $\$v =_{atomic} \$v'$ and **for**. In the proof of Theorem 3.1 we can note that the blow-up comes only in the composition step for $\circ$. For the special forms of **for** used above, we do not need to perform any composition. We will thus be able to show:

THEOREM 6.4. *There is a polynomial time function producing for every composition-free AtomXQ query an equivalent first-order interpretation. For Q a composition-free query in PosXQ, one can find in* PTIME *an interpretation that is equivalent to Q, in which all formulas are expressible in* $\exists FO$.

As before (in Proposition 6.2), in the second part of the theorem we assume that the sibling-axes are not given as part of the interpretation.

We give the modification to the algorithm of Theorem 3.2, leaving the verification that it can be performed in polynomial time to the reader.

We will perform the algorithm inductively now only on $AtomXQ$, rather than on $AtomXQ^+$. All the base cases of $AtomXQ$, and all the inductive cases other than **for**, are the same as in the prior algorithm. Rather than reducing to **union** and composition, we perform a new algorithm in the case of the restricted **for** available in Composition-free $AtomXQ$. We will also need to give a direct inductive step for the construct **not** β, which is now a primitive.

We first consider the case of $\alpha =$ **for** $\$v_{n+1}$ **in** $\$v_i/\mathsf{axis} :: \nu$ **return** β. We let I_1 be the interpretation obtained inductively for β, of node arity k and index arity k'. Then the interpretation for α will have:

—arity for the nodes is $k{+}1$, and the arity for indices is $k'{+}1$.

—$\phi_{\mathsf{Node}}(v_1 \dots v_n; x_1 \dots x_{k+1})$ given by the disjunction of two formulas. The first formula is

$$x_1 = c_{\mathsf{emb}} \wedge \mathsf{InputNode}(x_2) \wedge \bigwedge_{2 \le i < j \le k+1} x_i = x_j$$

This states that for every node in the input, we have a node of the output consisting of a constant tuple padded by an extra constant.
The second formula is

$$\mathsf{axis}(v_i, x_1) \wedge \mathsf{newtuple}^{\beta}(v_1 \dots v_n, x_1; x_2 \dots x_{k+1}).$$

This states that we take all new elements produced by β for any parameter value x_1 related to v_i by the axis, distinguishing when the same element is produced by different values of x_1 by including the value of x_1 in the tuple.

—$\phi_{\mathsf{child}}(v_1 \dots v_n; x_1 \dots x_{k+1}; y_1 \dots y_{k+1}) := (x_1 = c_{\mathsf{emb}} \wedge y_1 = c_{\mathsf{emb}} \wedge \mathsf{child}(x_2, y_2)) \vee (x_1 \neq c_{\mathsf{emb}} \wedge y_1 \neq c_{\mathsf{emb}} \wedge \exists v_{n+1} \mathsf{axis}(v_i, v_{n+1}) \wedge \phi^{\beta}_{\mathsf{child}}(v_1 \dots v_{n+1}; x_2 \dots x_{k+1}; y_2 \dots y_{k+1}))$ and similarly for the other σ_{nav} relations.

—$\phi_{\mathsf{Ind}}(v_1 \dots v_n; x'_1 \dots x'_{k'+1}) := \mathsf{axis}(v_i, x'_1) \wedge \phi^{\beta}_{\mathsf{Ind}}(v_1 \dots v_n, x'_1; x'_2 \dots x'_{k'+1})$. That is, the indices are the indices of β applied to a given parameter that is in the appropriate relation, with an extra component for the parameter.

—$\phi_{\mathsf{ItemOf}}(v_1 \dots v_n; x_1 \dots x_{k'+1}; x'_1 \dots x'_{k+1}) := \big(\mathsf{newtuple}^{\beta}(v_1 \dots v_n, x_1; x'_2 \dots x'_{k+1}) \wedge x'_1 = x_1 \wedge \phi^{\beta}_{\mathsf{ItemOf}}(v_1 \dots v_n, x_1; x_2 \dots x_{k'+1}; x'_2 \dots x'_{k+1})\big) \vee \big(x'_1 = c_{\mathsf{emb}} \wedge \exists w'_2 \dots w'_{k+1}$ $\phi^{\beta}_{\mathsf{ItemOf}}(v_1 \dots v_n x_1; x_2 \dots x_{k'+1}; w'_2 \dots w'_{k+1}) \wedge \ \phi^{\beta}_{\mathsf{emb}}(v_1 \dots v_n x_1; x'_2; w'_2 \dots w'_{k+1})\big)$. That is, an index and the associated node it indexes match on their first component, which correspond to a parameter value. The remaining components are either given according to the indexing of β, for a new tuple, or are the pre-image of the indexing of β under the embedding mapping, for an old tuple.

—$\phi_{<_{\mathrm{ind}}}(v_1 \dots v_n; x_1 \dots x_{k'+1}; x'_1 \dots x'_{k'+1}) := (x_1$ comes before x'_1 in document order$) \vee (x_1 = x'_1 \wedge \phi^{\beta}_{<_{\mathrm{ind}}}(v_1 \dots v_n x_1; x_2 \dots x_{k'+1}; x'_2 \dots x'_{k'+1}))$.

Again, the reader will want to compare this construction to the one for sequential composition in Section 4.

Correctness: Fix indexed forest $\mathcal{F}_{\#}$ and nodes $b_1 \dots b_n$ for the variables, and let ${\mathcal{F}_{\#}}^{\alpha} = [\![\alpha]\!]_n(\mathcal{F}_{\#}, b_1 \dots b_n)$. We define an isomorphism on nodes H^{α} on ${\mathcal{F}_{\#}}^{\alpha}$. Any node in ${\mathcal{F}_{\#}}^{\alpha}$ is either in $\mathcal{F}_{\#}$ or in $[\![\beta]\!]_{n+1}(\mathcal{F}_{\#}, b_1 \dots b_{n+1})$ for exactly one b_{n+1} that is a descendant of b_i in $\mathcal{F}_{\#}$. For a node b in ${\mathcal{F}_{\#}}^{\alpha} \cap \mathcal{F}_{\#}$, we let H^{α} be the constant tuple b^{k+1}. For a node b in $[\![\beta]\!]_{n+1}(\mathcal{F}_{\#}, b_1 \dots b_{n+1})$, let H^{α} be $b_{n+1} \cdot H^{\beta}(b)$, where H^{β} is the mapping (assumed to exist inductively) from $[\![\beta]\!]_{n+1}(\mathcal{F}_{\#}, b_1 \dots b_{n+1})$ to $[I^{\beta}](\mathcal{F}_{\#}, b_1 \dots b_{n+1})$. We omit the mapping J_{α} on indices for brevity.

We now turn to the case of $\alpha =$ **not** β. For simplicity, we show how to construct a query that produces the output of β if the list component of the output of β is empty, and otherwise produces the output of β unioned ($\uplus$) with the single-node tree $\mathit{construct}(\text{“}\mathit{yes}\text{”}, (\emptyset, []))$, with the list component containing the root node of the tree $\mathit{construct}(\text{“}\mathit{yes}\text{”}, (\emptyset, []))$. This is not exactly the same as **not** $\beta :=$ (**let** $\$v \ := \ \langle a \rangle \beta \langle /a \rangle$) $\$v =_{\mathit{deep}} \langle a/ \rangle$, since in the latter case we have an additional use of node construction, since the forest component will have an

additional constructed subtree rooted with a node labeled "a". However, the modification to give the exact definition is straightforward.

Again let I_1 be the interpretation obtained inductively for β, of node arity k and index arity k'. Then the interpretation for α will have a new constant c_Y of combined sort (distinct from those in β), label constant "*yes*", node arity k, index arity one, and

—$\phi_{\mathsf{Node}}(v_1 \dots v_n; x_1 \dots x_k) := \phi^{\beta}_{\mathsf{Node}}(\vec{v}, \vec{x}) \vee (\neg\exists w_1 \dots w_{k'}\ \phi^{\beta}_{\mathsf{Ind}}(\vec{v}, \vec{w}) \wedge \bigwedge_{i \leq k} x_i = c_Y)$. That is, the nodes are the nodes of β plus an additional node if the list returned by β is non-empty.

—$\phi_{\mathsf{Haslabel}}(v_1 \dots v_n; x_1 \dots x_k; z) := (\phi^{\beta}_{\mathsf{Node}}(\vec{v}, \vec{x}) \wedge \phi^{\beta}_{\mathsf{Haslabel}}(v_1 \dots v_n; x_1 \dots x_k; z)) \vee (\neg\exists w_1 \dots w_{k'}\ \phi^{\beta}_{\mathsf{Ind}}(\vec{v}, \vec{w}) \wedge \bigwedge_{i \leq k} x_i = c_Y \wedge z = \text{"}yes\text{"})$. That is, the old nodes are labeled as in β and the additional node, if it exists, is labeled with "*yes*".

—$\phi_{\mathsf{Ind}}(\vec{v}; x) := \neg\exists \vec{w}_1 \dots w_{k'}\ \phi^{\beta}_{\mathsf{Ind}}(\vec{v}, \vec{w}) \wedge x = c_Y$ That is, there is exactly one index node iff β returns an empty list, and otherwise there are no indices.

—$\phi_{\mathsf{ItemOf}}(\vec{v}; x; \vec{x}') := \neg\exists \vec{w}_1 \dots w_{k'}\ \phi^{\beta}_{\mathsf{Ind}}(\vec{v}, \vec{w}) \wedge x = c_Y \wedge \bigwedge_{i \leq k} x'_i = c_Y$

The rest of the structure is straightforward. □

In [Anonymous 2008b] we show that composition-free *AtomXQ* queries have the same expressiveness as general *AtomXQ* queries, and composition-free *XQ* queries have the same expressiveness as general *XQ* queries. However composition-free $AtomXQ^-$ and full $AtomXQ^-$ have different expressiveness in the presence of upward and sideways axes (if only downward axes are supported, the two languages again coincide [Koch 2006]).

6.2 Node equality and $FO2$

We turn now to queries that are both composition-free and without $=_{node}$. We can show that composition-free $AtomXQ^-$ maps into a small fragment of FO.

The logic NFO^2, *navigationally two-variable FO*, is built up from atomic formulas via the rules:

—if ϕ_1 and ϕ_2 are navigationally two-variable, then so are $\phi_1 \vee \phi_2$ and $\phi_1 \wedge \phi_2$ and $\exists \vec{w}\ \phi(\vec{w})$ (where $\vec{w}$ is any subset of the free variables).

—if $\phi(\vec{w})$ is navigationally two-variable and if, in the navigational dependency graph of ϕ, no two free variables are connected by a path, and thus every bound variable is connected by a path to at most one free variable, then $\neg\phi(\vec{w})$ is navigationally two-variable.

Here the *navigational dependency graph* of a formula is the graph with nodes for each variable of the formula and with edges connecting two nodes iff the corresponding variables appear together in some axis predicate or equality.

Example 6.5. The FO formulas corresponding to expressions of the navigational core of XPath 1.0 (see e.g. [Benedikt and Koch 2009], i.e., expressions that are built from path expression and conditions which are Boolean combinations of navigational XPath expressions, are in navigational two-variable FO. For instance, XPath expression /descendant::A[not child::B] is equivalent to the navigational two-variable FO formula

$$\exists x\ \mathrm{root}(x) \wedge \mathsf{descendant}(x, y) \wedge P_A(y) \wedge \neg\exists z\ \mathsf{child}(y, z) \wedge P_B(z)$$

where P_A and P_B are unary predicates that are true for those nodes labeled A and B, respectively. Navigational two-variable FO essentially extends beyond the FO fragment corresponding to navigational XPath 1.0 by allowing for conjunctions of unary formulas from that fragment.

On the other hand, a formula such as $\neg\mathsf{child}(x, y)$ are not in navigational two-variable FO. □

We now justify the term "navigationally two-variable" for this fragment, by showing that it is closely related to two-variable logic. For a finite alphabet Σ, we let $\sigma_{nav}(\Sigma)$ be the signature with only a sort for nodes, the same axis predicates as σ_{nav}, and where instead of the labeling function we have unary predicates P_a for every a in Σ. A data forest whose labels all lie in Σ can be interpreted as a $\sigma_{nav}(\Sigma)$ structure in the obvious way. We let $FO^2(\Sigma)$ be two-variable logic over $\sigma_{nav}(\Sigma)$. By an $FO^2(\Sigma)$ formula $\phi(x_1 \dots x_n)$ in many variables, we mean a formula that is a Boolean combination of formulas $\phi(x_i)$, each of which is in FO^2 above. For a σ_{nav} formula ϕ and a finite alphabet Σ, we let $\Sigma(\phi)$ be the restriction of ϕ to structures where the labels come from Σ.

We say that an FO formula ϕ is *almost two-variable* if for every finite alphabet Σ, $\Sigma(\phi)$ is equivalent (modulo the transformation of a σ_{nav} structure to a $\sigma_{nav}(\Sigma)$ structure), to an $FO^2(\Sigma)$ formula. That is, when we restrict to forests whose labels lie in Σ, ϕ is in FO^2.

Claim 6.6. *All navigationally two-variable formulas in one free variable are almost two-variable.*

To prove this, we show a more general claim on the restrictions of navigationally two-variable formulas to a finite alphabet. For any label alphabet Σ, a formula is in $CQ(FO^2)$ if it is of the form $\exists w_1 \dots \exists w_m\ \gamma(w_1 \dots w_n)$ where $m \leq n$ and $\gamma(\vec{w})$ is a conjunction of formulas either of the form $\phi(w)$ where $\phi \in FO^2$ or of form $A(w_1, w_2)$, where A is a navigational predicate. Since such a γ can be made acyclic [Gottlob et al. 2004] a $CQ(FO^2)$ formula in at most one free variable must be in FO^2. For our signatures, in which relations (axes) are antireflexive and at most binary, a formula is acyclic simply if the undirected graph given by the variables as nodes and the binary atoms as edges is acyclic. We will show:

For every navigationally two-variable formula ϕ and every finite alphabet Σ, $\Sigma(\phi)$ is equivalent to a disjunction of $CQ(FO^2)$ formulas.

The proof is by induction on the structure of navigationally two-variable formulas. In the induction, we maintain an additional invariant that in the produced formula $\bigvee \phi_i$, there is an axis predicate connecting two free variables of ϕ_i to some other variable in some ϕ_i iff there is some such predicate in ϕ, and for every existentially quantified variable $w_1 \dots w_m$ in the $CQ(FO^2)$ form of ϕ_i, there is a bound variable in ϕ which occurs in axis predicates with a free variable of ϕ iff w_i occurs in an axis predicate with that free variable in ϕ_i. That is, the navigational dependency structure among variables in $\bigvee \phi_i$ is the same as in ϕ.

For atomic formulas, the claim is clear, since atomic formulas with $=_{atomic}$ can be replaced by disjunctions of conjunctions of atomic formulas $P_a(w)$. The induction steps for positive boolean operators are also straightforward: in the conjunction case, one distributes the conjunction over the disjuncts in each formula.

Consider $\Sigma(\neg\phi(\vec{w})) = \neg\Sigma(\phi)$. By hypothesis, no w_i is connected by a sequence of axis relations with a w_j. By induction we can assume $\Sigma(\phi)$ is a disjunction of

$CQ(FO^2)$ formulas ϕ_i, with each ϕ_i also containing no axis predicates relating the free variables of ϕ. It suffices to show that the negation of each disjunct ϕ_i is a disjunction of conjunctions of FO^2 formulas with one free variable. Fix ϕ_i and let $\phi_i = \exists \vec{x} \bigwedge_j \gamma_{ij}$. Each γ_{ij} is either an axis predicate containing at least one variable from $\vec{x}$, a label predicate, or an FO^2 formula with one free variable. Our inductive invariant and the assumptions on ϕ imply that each free variable of ϕ_i is in a different connected component of the navigational dependency graph of ϕ_i, hence we can split ϕ_i into a conjunction of formulas that correspond to each component, and it suffices to show that the negation of each component is in FO^2. But each component is a conjunctive query over trees with one free variable, and it is known [Gottlob et al. 2004] that each such query can be converted into an acyclic query, and hence into an FO^2 formula $\phi(v)$. Hence $\neg\phi(v)$ is also in FO^2.

This completes the proof of Claim 6.6. That is, navigationally two-variable formulas have two-variable expressive power over any fixed set of labels. □

We are now ready to give the main result of this subsection, that composition-free $AtomXQ^-$ Boolean queries translate to navigationally two-variable interpretations.

THEOREM 6.7. *For every composition-free $AtomXQ^-$ Boolean query, there is an equivalent FO interpretation where all formulas are navigationally two-variable. For every $PosXQ^-$ composition-free Boolean query there is an equivalent interpretation in which all formulas other than those for the sibling axes are $\exists FO$ and navigationally two-variable.*

PROOF. We consider the algorithm of Theorem 3.2 modified via the changes given for Theorem 6.4 but removing the atomic step for $=_{node}$. Recall that for a query Q, Q_{Bool} returns true iff the list returned by Q is nonempty. Hence it suffices to show that all the index formulas ϕ^Q_{Ind} in interpretations produced from such queries are navigationally two-variable. We divide the free variables of these formulas into parameter variables $\vec{v}$ (i.e., those corresponding to parameters of Q), and the remaining free variables, which we refer to as "output variables". We show inductively that ϕ^Q_{Ind} can be written as a disjunction of navigationally two-variable formulas, where in each disjunct the parameter variables are in different components of the navigational dependency graph of the disjunct. The result follows from the first part of this, since navigationally two-variable formulas are closed under disjunction and projection.

For the basic queries this is easy to verify. We deal with the inductive cases:

—If Q is the expression $\langle a\rangle\beta\langle/a\rangle$, then the index structure is degenerate: $\phi^Q_{\mathsf{Ind}}(\vec{v};\vec{x})$ says that each x_i is equal to the same constant. Clearly this is navigationally two-variable and imposes no relationships among the v_i.

—If Q is $\beta\ \gamma$, then $\phi^Q_{\mathsf{Ind}}(\vec{v}; x'_1 \ldots x'_{k'+1}) = (x'_1 = c'_\beta \wedge \phi^\beta_{\mathsf{Ind}}(\vec{v}; x'_2 \ldots x'_{k'+1})) \vee (x'_1 = c'_\gamma \wedge \phi^\gamma_{\mathsf{Ind}}(\vec{v}; x'_2 \ldots x'_{k'+1}))$
and it is easy to see the result by induction.

—If Q is of the form **if** β **then** γ, then $\phi^Q_{\mathsf{Ind}}(\vec{v};\vec{x}') := \exists \vec{y}'\ \phi^\beta_{\mathsf{Ind}}(\vec{v};\vec{y}') \wedge \phi^\gamma_{\mathsf{Ind}}(\vec{v};\vec{x}')$.
Navigationally two-variable formulas are closed under conjunction and projection, so the first part of the theorem is immediate. For the second, notice that no free variables other than the parameter variables are shared between the two conjuncts – hence the second part follows.

—If $Q = \textbf{not}\ \beta$, we have $\phi^Q_{\mathsf{Ind}}(v_1 \ldots v_n; x'_1) := x'_1 = c_Y \wedge \neg\exists \vec{w}'\ \phi^\beta_{\mathsf{Ind}}(\vec{v}; \vec{w}')$ for a constant c_Y. Since the induction hypothesis says that the v_i are in different components of the navigational dependency graph, the resulting formula is navigationally two-variable. The hypothesis on the v_i is clearly preserved.

—In the case $Q = \textbf{for}\ \$v_{n+1}\ \textbf{in}\ \$v_i/axis :: \nu\ \textbf{return}\ \beta$, we have

$$\phi^Q_{\mathsf{Ind}}(\vec{v}; \vec{x}') := \mathsf{axis}(v_i, x'_1) \wedge \phi^\beta_{\mathsf{Ind}}(v_1 \ldots v_n, x'_1; x'_2 \ldots x'_{k'+1}).$$

Clearly, this is navigationally two-variable by induction. The navigational dependency graph of Q is formed from the navigational dependency graph G' of $\phi^\beta_{\mathsf{Ind}}$ by replacing v_{n+1} with x'_1 and then adding an edge from v_i to x'_1. Since $x'_1 = v_{n+1}$ and v_{n+1} was in a different component of G' than the other v_j, the variables $v_1 \ldots v_n$ are still in different components.

The case for positive queries follows by combining the above argument with the prior results about the composition-free translation. □

In [Koch 2006], it is shown that $AtomXQ^-$ queries with only the downward (and self) axes and without node-equality can be converted to composition-free queries (Theorem 7.8). From this result and Theorem 6.7 we have:

COROLLARY 6.8. *For every $AtomXQ^-$ Boolean query (i.e., Q_{Bool} for $Q \in AtomXQ^-$) that uses only downward axes, there is an equivalent navigationally two-variable query.*

This does not hold when all axes are present [Anonymous 2008b].

6.3 Applications to Expressiveness and Complexity.

The following results on the expressiveness of Boolean queries are an immediate consequence of the results above (in the case of $AtomXQ^-$ below, we also use the equivalence of Core XPath and FO^2 shown in [Marx 2004]).

COROLLARY 6.9. *Let Q be an $AtomXQ$ expression, and Q_{Bool} be the Boolean query defined by Q. Then,*

—*(1) Q_{Bool} is expressible in the relational calculus;*

—*(2) If $Q \in PosXQ$ and Q does not use the sibling axes, Q_{Bool} is expressible by a union of conjunctive queries (over all atomic formulas and inequalities);*

—*(3) If $Q \in AtomXQ^-$ and Q is composition-free then for every finite set of labels Σ, there is a Core XPath query equivalent to Q_{Bool} on data trees with labels in Σ. In particular (by [Marx 2004; Etessami et al. 2002]), there are FO queries that are not expressible in composition-free $AtomXQ^-$.*

The proofs follow because the boolean semantics of queries (see Subsection 2.2) is given by checking whether the sequence returned is nonempty, which is just an existential quantification of the formula ϕ_{Ind}. The first two results could be generalized to "relational queries" — queries on tree encodings of relational tables. We will comment on this further in the next section. The results on translation to FO immediately give alternative proofs of the following upper bounds:

COROLLARY 6.10. *We have the following complexity bounds for $AtomXQ$ and its sublanguages:*

—*All AtomXQ queries can be evaluated in data complexity* AC^0 *on a relational representation of the data, and in* ExpSpace *combined complexity.*

—*PosXQ queries can be evaluated in combined complexity* NExpTime.

—*Composition-free AtomXQ is in* PSpace *w.r.t. combined complexity.*

Note that in all the above complexity bounds, the input is a σ_{nav} structure representing the data tree $\mathcal{T}$; this is not the default assumption of [Koch 2006]. However, the combined complexity results follow immediately from [Koch 2006] and the fact that there are LogSpace translations back and forth between XML trees and σ_{nav}-structures.

Proof. It is well-known that FO queries can be evaluated in AC^0 [Immerman 1999], and hence the data complexity bound in the first sentence follows from Theorem 3.1. The combined complexity of FO queries is in PSpace, while the translation of Theorem 3.1 is in ExpTime, hence certainly in ExpSpace. Thus the composition of the translation with the evaluation function for FO is in ExpSpace, and the bound on combined complexity for $AtomXQ$ follows. $\exists FO$ queries can be evaluated in NP, simply by guessing witnesses to existential quantifiers and guessing which disjuncts within a disjunction are satisfied (the sibling axes on the output can be derived in PTime once the remaining parts of the representation are available). Combining this with the ExpTime translation of Proposition 6.2 gives the bound on $PosXQ$ queries.

It is known that the combined complexity of FO queries is in PSpace. Using the PTime (hence PSpace) algorithm of Theorem 6.4, this gives the PSpace bound for composition-free $AtomXQ$. □

7. FROM XQ WITH DEEP EQUALITY TO $FO(\mathbf{CNT})$

When we turn to deep equality, first-order logic no longer suffices, even when we restrict to fixed-depth trees. Consider the query (i.e. condition) Q^1 defined by:

$$\textbf{let } \$v_1 \;:=\; \langle A\rangle(\textbf{for } \$x \textbf{ in } \$v_0/\mathsf{child} :: a \textbf{ return } \langle B/\rangle)\langle/A\rangle$$
$$\textbf{let } \$v_2 \;:=\; \langle A\rangle(\textbf{for } \$x \textbf{ in } \$v_0/\mathsf{child} :: c \textbf{ return } \langle B/\rangle)\langle/A\rangle \;\; \$v_2 =_{deep} \$v_1$$

Q^1_{Bool} holds iff the number of a children of $\$v_0$ is equal to the number of c children. It is easy to show that there is no first-order interpretation equivalent to Q (cf. e.g. [Libkin 2004]).

We will see that the absence of the ability to count is, however, the only obstacle.

Theorem 7.1. *For every* XQ^+ *query* Q*, there is an* $FO(\mathbf{Cnt})$ *indexed forest interpretation equivalent to* Q*, which can be constructed from* Q *in* ExpTime.

Example 7.2. *Consider* Q^1 *above. The output forest returned by* Q^1 *includes several "temporary" trees. For example, whenever* $\langle b/\rangle$ *is called a new tree is formed. In addition, the output forest includes a tree consisting of one node labeled with "yes" exactly when the cardinality of the number of a children equals the number of b children. The sequence component returned by* Q^1 *is either empty (if the cardinalities do not match) or the root of the "yes" tree.*

We give only the index formula in an interpretation for Q^1*. A forest interpretation equivalent to* Q^1 *will include one constant* I_Y *for the (potential) single index of the sequence, and a label constant "yes"; the interpretation will have* $\phi_{\mathsf{Ind}}(x_1) :=$

$\big((\exists i\ \exists^{=i} x\ \mathsf{child}(v_0,x) \wedge \mathsf{Haslabel}(x,a) \wedge \exists^{=i} y\ \mathsf{child}(v_0,y) \wedge \mathsf{Haslabel}(y,b)) \rightarrow x_1 = I_Y\big)$
$\wedge\ \big((\neg\exists j\ \exists^{=j} x\ \mathsf{child}(v_0,x) \wedge \mathsf{Haslabel}(x,a) \wedge \exists^{=j} y\ \mathsf{child}(v_0,y) \wedge \mathsf{Haslabel}(y,a)) \rightarrow$ **false**$\big)$.

We now prove Theorem 7.1 along the lines of the proof of Theorem 3.1. The notion of $FO(\mathbf{Cnt})$-indexed forest interpretation is defined exactly as for FO. We then claim that for every XQ^+ query Q there is an $FO(\mathbf{Cnt})$-indexed forest interpretation that captures the full semantics of Q.

We extend the algorithm from Theorem 3.2, inductively assuming that we have mapped XQ^+ expressions with additional free variables $\$v_1 \ldots \$v_n$ to $FO(\mathbf{Cnt})$ interpretations with additional variables $v_1 \ldots v_n$. The inductive cases are as in Theorem 3.1: we simply extend the function $\mathsf{Compose}$ to include the quantifiers $\exists^{=i}\vec{x}$ and $\exists i$: $\mathsf{Compose}$ commutes with the quantification $\exists i$, and

$$\mathsf{Compose}(\exists^{=i} x_1 \ldots x_s\ \eta(x_1 \ldots x_s, y_1 \ldots y_t), I) := \exists^{=i}\vec{x}_1 \ldots \vec{x}_s\ \mathsf{Compose}(\eta(x_1 \ldots x_s, y_1 \ldots y_t), I).$$

The base cases are also inherited from Theorem 3.1 except for $\$v_i =_{deep} \$v_j$. To translate α of the form $\$v_i =_{deep} \$v_j$, we use the (well-known) fact that isomorphism of trees can be expressed using $FO(\mathbf{Cnt})$. Nodes v_1 and v_2 have isomorphic subtrees iff the following property holds:

For every descendant y_1 of v_1, there is a descendant y_2 of v_2 having the property that for every w_1 lying (non-strictly) between y_1 and v_1, there is w_2 lying between y_2 and v_2 such that: the vertical distance from v_1 to w_1 is the same as the vertical distance of v_2 to w_2, the vertical distance from w_1 to y_1 is the same as the vertical distance of w_2 to y_2, the label of w_1 is the same as the label of w_2, and the number of left-siblings of w_1 is the same as the number of left-siblings of w_2.

Note that the statements that two pairs of nodes are the same vertical distance apart can be expressed in $FO(\mathbf{Cnt})$, as is the statement that two nodes have the same number of right-siblings. Using these facts, one can express the above in $FO(\mathbf{Cnt})$.

Recall that the composition-free fragment of XQ is formed by restricting **for** as in $AtomXQ$ and adding the construct $Q_1 =_{deep} Q_2$. Note that we do not need to introduce negation here, since it will be definable using $=_{deep}$. We now have the analogous result to Theorem 6.4:

THEOREM 7.3. *For every composition-free XQ query, an equivalent $FO(\mathbf{Cnt})$ interpretation can be constructed in* PTIME.

The proof of this is simply to take the algorithm of Theorem 6.4, adding on the construction for the atomic case of $=_{deep}$ given above, and adding also a simple inductive case for $Q_1 =_{deep} Q_2$, which is done similarly to the atomic case.

7.1 Applications to Expressiveness and Complexity.

As in the case of $AtomXQ$, we get a nice bound on the expressiveness of Boolean queries:

COROLLARY 7.4. *Every XQ boolean query is expressible as an $FO(\mathbf{Cnt})$ query over σ_{nav}.*

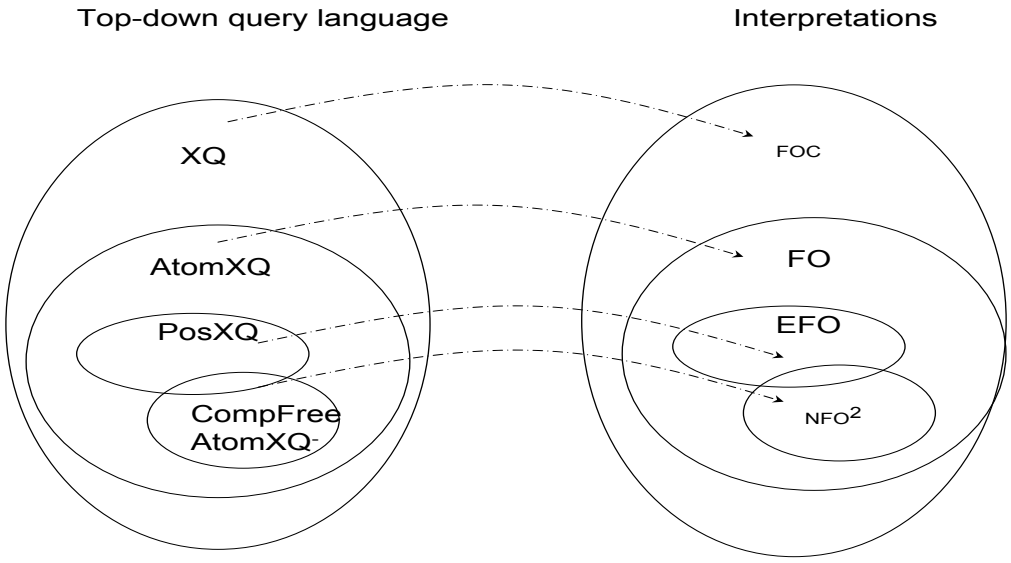

Fig. 3. Expressiveness of languages

A similar result could be stated for relational queries: the relational queries expressible in *XQ* over the data tree coding of relations (i.e. as flat trees whose attributes match the attributes of the relations) are exactly those that are expressible as $FO(\mathbf{Cnt})$ interpretations.

We have the following consequences for complexity:

COROLLARY 7.5. *The data complexity of XQ is in* TC^0. *The combined complexity of XQ is in* EXPSPACE*, while the combined complexity of Composition-free XQ is* PSPACE*-complete.*

The TC^0 bound follows from Theorem 2.5 and Theorem 7.1, while the other results follow from Theorem 7.1 and the PSPACE-complete combined complexity of $FO(\mathbf{Cnt})$ [Immerman 1999]. These upper bounds differ only slightly from the bounds of [Koch 2006]: there they establish a LOGSPACE data complexity bound on a DOM representation of the data; this representation is relational, but does not contain the transitive axes; in contrast, transitive axes are built in to our σ_{nav} representation, and the representation that is built by our operations includes the transitive axes in the output.

8. CONCLUSIONS AND FUTURE WORK

Our main goal was to establish embeddings of XQuery fragments into logics. Our results in terms of expressiveness bounds are shown in Figure 3. On the left are fragments of the top-down tree building languages, and on the right are classes of interpretations that suffice to capture each language.

In the paper [Anonymous 2008b] we show that these relationships are tight, and as a consequence we can deduce expressiveness equivalence of several of our languages. Many of the questions concerning translation complexity to logic remain open, in particular whether *Boolean AtomXQ* queries can be translated in PTIME

to FO. This is similar to the question, initially posed by Vardi, of the existence of a PTIME translation from nested relational queries on flat structures to relational queries, which has been an open problem in complex-valued databases for some time [Van den Bussche 2005]. It is known from [Koch 2006] that unless PSPACE = NEXPTIME (which is considered unlikely), there cannot be such a PTIME translation. This stands in contrast to the (at first sight surprising) fact that nested definitions can be eliminated from FO in polynomial time [Avigad 2003]. While definitions in first-order logic appear to correspond to composition in XQuery, it seems that constructing complex values really adds power.

The FO and $FO(\mathbf{Cnt})$ queries represent natural benchmarks for comparing the expressiveness of tree-structured queries. How does XQ compare to these benchmarks? In this paper, we have shown that XQ and $AtomXQ$ can be captured by these two classes. In the companion paper [Anonymous 2008b], we show that the expressiveness of XQ and $AtomXQ$ is roughly the same as FO and $FO(\mathbf{Cnt})$, up to sibling ordering and indexing issues, and restriction to queries that do not perform deep restructuring of documents.

9. RELATED WORK AND DISCUSSION

The correspondence between relational languages and languages for structured data-types has received considerable attention in the framework of complex-valued query languages. Close in spirit to our translation of XQ to first-order logic are the *normal form theorems* [Wong 1996] and *conservativity theorems* (e.g. the "flat-flat" theorem in [Paredaens and Van Gucht 1988]) for complex-valued query languages. These results show that the building of intermediate results of higher-order than the input is unnecessary for expressiveness, and also show that certain higher-order languages have the same "flat" expressiveness as relational calculus. In translating XQ into first-order interpretations, we are performing a very similar elimination. More generally, a standard technique in complex-valued query languages is to look at a relational representation. In [Suciu 1997; Gyssens et al. 2001; Van den Bussche 1992] such encodings are used to reduce problems concerning complex-valued languages to a relational setting. Probably closest to our work is the approach to the conservativity theorem of [Paredaens and Van Gucht 1988] proposed in [Van den Bussche 2001], showing that nested relational queries can be represented by ordinary relational calculus queries on flat codings of a nested relation. The first results in our paper can be seen as an extension of this approach to tree-structured data.

The main distinction between the complex-valued data model and the tree data model we deal with here is that tree-structured data is not strongly-typed to be of fixed depth. Although non-recursive languages like XQ cannot make changes deeply in the output tree, they can query the entire input tree, making use of the descendant relationship. This distinction leads to the strong connection between XQ and path-oriented tree navigation languages such as XPath; in the world of fixed-depth data types (complex-valued or object-oriented) there is no analog. Even when restricting to fixed depth, there are subtle differences between the main language atomic XQ we deal with here and prior algebras: XQ with atomic equality has node identity on trees as a primitive; in the complex-valued world, the analog is equality in bag languages. However $AtomXQ$ does not have the deep equality that is natural for a bag algebra. Queries in languages such as XQ and $AtomXQ$

have a dual effect of forming new objects and returning a list of subobjects within these newly-created objects – this is seen in the two components of the output of the XQ semantic function (Figure 2). Hence these languages would most naturally be compared to object query languages that have both complex value constructors and object creation. However, OO query languages typically have recursion mechanisms that are much more powerful than those found in XQ.

Our interest is the correspondence between query languages and logical representations; we track precisely how the correspondence between tree languages and relational languages is affected as we add and subtract features from XQ: we know of no analog to these correspondences (e.g. between composition-free XQ without node-equality and two-variable logic) in the nested relational world. Although first-order logic was used as a benchmark for the expressiveness of Boolean and flat queries in complex valued models, there has been, to our knowledge, no prior work using first-order or $FO(\mathbf{Cnt})$ interpretations as a benchmark for completeness of structured object query languages for arbitrary queries.

In the XML literature, the closest work to ours is the article [Koch 2006], which examines the expressiveness of XQuery via a tight correspondence with complex-valued query languages, rather than with relational queries. Mappings are given between XQuery over the child axis to Monad Algebra, and conversely. When only these axes are present and node identity is absent, the semantics of XQuery can be dramatically simplified, since it is not necessary to distinguish between two nodes that are structurally identical. The semantics of [Koch 2006] and the mapping to Monad Algebra do not apply in the presence of all axes. In addition, this work does not strictly give a correspondence in expressiveness, since the mappings between complex values and data trees employed there do not compose to identities. However, the mappings are sufficient to infer upper and lower bounds on complexity and [Koch 2006] exploits this to provide an in-depth study of the complexity of XQuery.

Somewhat surprisingly, some of the issues we deal with here in translating from XQuery to relational logics have not been studied in the context of mapping from XQuery to object query languages (in [Koch 2006], or elsewhere). We also think it is important to study XML query languages via translation to relational languages rather than through richer models. The relationship of XQuery to Relational Calculus and SQL is of independent interest, due to the need to implement XQuery on relational stores. Our results give a manner of seeing many of the XQuery complexity bounds of [Koch 2006] that is more direct and self-contained than the approach via complex-valued queries.

The expressiveness of XQuery has also been studied in the papers [Hidders et al. 2005; Hidders et al. 2005; Hidders et al. 2004]. [Hidders et al. 2005] presents an in-depth study of the expressiveness of XQuery, building on a formalization given in [Hidders et al. 2004]. These papers deal with a more fine-grained model of XQuery, and study the relationship among a much richer assortment of features. While these works are more useful in understanding the current standard (and, e.g. as a step towards arriving at a minimal subset of XQuery for compilation), our work has a different goal. We are interested in comparison of the expressiveness of the core structuring constructs of XQuery and XQuery-like languages against an external benchmark, which [Hidders et al. 2005; Hidders et al. 2004] does not do.

Our results can be seen as dealing with translation of XQuery to SQL, a topic

which has received considerable attention in the context of relational storage for XML [Krishnamurthy et al. 2003; 2004; Krishnamurthy et al. 2004; Fan et al. 2005]. These works consider a number of relational encodings of XML, and are concerned not just with sound translation but efficient evaluation. In these works the target relational language tends to be more powerful than the source language e.g. including recursion even when translating from XQ, in order to deal with the transitive axes, and also including built-in operations such as arithmetic. None of the works cited above deal with translation from non-recursive XQuery over general XML trees to relational calculus, although several deal with special cases (e.g. queries over XML views of relational data [Fernández et al. 2002]), while others translate into SQL-99 [Krishnamurthy et al. 2004; Fan et al. 2005].

The closest paper in this area to our work is [DeHaan et al. 2003], which is relevant to our translation of full XQ into $FO(\mathbf{Cnt})$. This work gives a translation of a large XQuery fragment into SQL-99; the only additional feature over $FO(\mathbf{Cnt})$ that is used is SQL composition (view definitions), along with arithmetic. The semantic model in [DeHaan et al. 2003] is different from ours, since queries return single ordered forests rather than a list of nodes within a forest. However, it is stated that the translation can be extended to handle node identity issues. The analysis of complexity of translation and the translation of fragments of XQ, are not studied in [DeHaan et al. 2003].

REFERENCES

Abiteboul, S., Hull, R., and Vianu, V. 1995. *Foundations of Databases*. Addison-Wesley.

Anonymous. 2008a. Details omitted due to double-blind reviewing.

Anonymous. 2008b. Details omitted due to double-blind reviewing.

Avigad, J. 2003. "Eliminating Definitions and Skolem functions in First-order Logic". *ACM Transactions on Computational Logic* **4**, 3, 402–415.

Barrington, D. A. M., Immerman, N., and Straubing, H. 1990. "On Uniformity within NC1". *Journal of Computer and System Sciences* **41**, 3, 274–306.

Benedikt, M. and Koch, C. 2009. "XPath Leashed". *ACM Computing Surveys*. to appear.

Chamberlin, D., Robie, J., and Florescu, D. 2000. Quilt: An XML query language for heterogeneous data sources. In *Proc. WebDB*. 53–62.

DeHaan, D., Toman, D., Consens, M., and Özsu, M. T. 2003. "A Comprehensice XQuery to SQL Translation using Dynamic Interval Encoding". In *Proc. SIGMOD*. 623–634.

Ebbinghaus, H.-D. and Flum, J. 1999. *Finite Model Theory*. Springer. 2nd edition.

Etessami, K., Vardi, M., and Wilke, T. 2002. "First Order Logic with Two Variables and Unary Temporal Logic". *Information and Computation* **179**, 2, 279–295.

Fan, W., Yu, J. X., Lu, H., Lu, J., and Rastogi, R. 2005. "Query Translation from XPath to SQL in the Presence of Recursive DTDs". In *Proc. VLDB*. 337–348.

Fernández, M., Kadiyska, Y., Morishima, A., Suciu, D., and Tan, W.-C. 2002. "SilkRoute: A Framework for Publishing Relational Data in XML". *ACM Transactions on Database Systems* **27**, 4, 438–493.

Gottlob, G., Koch, C., and Pichler, R. 2005. "Efficient Algorithms for Processing XPath Queries". *ACM Transactions on Database Systems* **30**, 2 (June), 444–491.

Gottlob, G., Koch, C., and Schulz, K. U. 2004. "Conjunctive Queries over Trees". In *Proc. PODS*. 189–200.

Gurevich, Y. and Shelah, S. 1989. "On the Strength of the Interpretation Method". *Journal of Symbolic Logic* **54**, 2, 305–323.

Gyssens, M., Suciu, D., and Gucht, D. V. 2001. "The Restricted and Bounded Fixpoint Closures of the Nested Relational Algebra are Equivalent ". *Information and Computation* **164**, 1, 85–117.

Hidders, J., Marrara, S., Paredaens, J., and Verkammen, R. 2005. "On the Expressive Power of XQuery Fragments". In *Proc. DBPL*. 154–168.

Hidders, J., Paredaens, J., Verkammen, R., and Demeyer, S. 2004. "A Light but Formal Introduction to XQuery". In *Proc. XSYM*. 5–20.

Hidders, J., Paredaens, J., Verkammen, R., Michiels, P., and Page, W. L. 2005. "On the Expressive Power of Node Construction in XQuery". In *Proc. WebDB*. 85–90.

Immerman, N. 1999. *"Descriptive Complexity"*. Springer.

Johnson, D. S. 1990. "A Catalog of Complexity Classes". In *Handbook of Theoretical Computer Science*. Vol. 1. Elsevier Science Publishers B.V., Chapter 2, 67–161.

Koch, C. 2006. "On the Complexity of Non-recursive XQuery and Functional Query Languages on Complex Values". *ACM Transactions on Database Systems* **31**, 4, 1215–1256.

Krishnamurthy, R., Chakaravarthy, V., Kaushik, R., and Naughton, J. 2004. "Recursive XML Schemas, Recursive XML Queries, and Relational Storage: XML-to-SQL Query Translation". In *Proc. ICDE*. 42–53.

Krishnamurthy, R., Kaushik, R., and Naughton, J. 2003. "XML-to-SQL Query Translation Literature: State of the Art and Open Problems ". In *Proc. XSYM*. 1–18.

Krishnamurthy, R., Kaushik, R., and Naughton, J. 2004. "Efficient XML-to-SQL Query Translation: Where to Add the Intelligence". In *Proc. VLDB*. 144–155.

Libkin, L. 2004. *Elements of Finite Model Theory*. Springer.

Marx, M. 2004. "XPath with Conditional Axis Relations". In *Proc. EDBT*. 477–494.

Marx, M. 2005. "First order paths in ordered trees". In *Proc. ICDT*. 114–128.

Paredaens, J. and Van Gucht, D. 1988. "Possibilities and Limitations of Using Flat Operators in Nested Algebra Expressions". In *Proc. PODS*. 29–38.

Schweikardt, N. 2005. "Arithmetic, First-Order Logic, and Counting Quantifiers". *ACM Transactions on Computational Logic* **6**, 3, 634–671.

Suciu, D. 1997. "Bounded Fixpoints for Complex Objects". *Theoretical Computer Science* **176**, 4, 283–328.

Van den Bussche, J. 1992. "Complex Object Manipulation Through Identifiers: An Algebraic Perspective". Technical Report 92-41, University of Antwerp, Department of Mathematics and Computer Science.

Van den Bussche, J. 2001. "Simulation of the Nested Relational Algebra by the Flat Relational Algebra, with an Application to the Complexity of Evaluating Powerset Algebra Expressions". *Theoretical Computer Science* **254**, 1–2, 363–377.

Van den Bussche, J. 2005. Personal Communication.

Wong, L. 1996. "Normal Forms and Conservative Extension Properties for Query Languages over Collection Types". *Journal of Computer and System Sciences* **52**, 3, 495–505.

World Wide Web Consortium. 2002. "XQuery 1.0 and XPath 2.0 Formal Semantics. W3C Working Draft (Aug. 16th 2002). `http://www.w3.org/TR/query-algebra/`.