Expressive and efficient pattern languages for tree-structured data

(extended abstract)

Frank Neven[*]
Limburgs Universitair Centrum

Thomas Schwentick
Johannes Gutenberg-Universität Mainz
Institut für Informatik

ABSTRACT

It would be desirable to have a query language for tree-structured data that is (1) as easily usable as SQL, (2) as expressive as monadic second-order logic (MSO), and (3) efficiently evaluable. The paper develops some ideas in this direction. Towards (1) the specification of sets of vertices of a tree by combining conditions on their induced subtree with conditions on their path to the root is proposed. Existing query languages allow regular expressions (hence MSO logic) in path conditions but are limited in expressing subtree conditions. It is shown that such query languages fall short of capturing all MSO queries. On the other hand, allowing a certain guarded fragment of MSO-logic in the specification of subtree conditions results in a language fulfilling (2), (3) and, arguably, (1).

Categories and Subject Descriptors

F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Computational Logic*; H.2.3 [**Database Management**]: Languages—*Query Languages*

General Terms

Languages, Theory, Algorithms

1. INTRODUCTION

One of the reasons for the success of relational databases is the existence of a robust and well-behaved core query language, first-order logic, which together with its relatives SQL and relational algebra is at the same time user-friendly, reasonably expressive and can be evaluated relatively efficient [2].

In various applications, however, data is structured less uniformly than relational data. One example of particular importance is tree-structured data, e.g., in the context of structured documents or heterogeneous database environments (as modeled, e.g., by semi-structured data [1]). From the more complicated structure of data there arises the need for a more expressive query language for which the evaluation procedures, in turn, are more demanding. Furthermore, a query language for tree-structured data should permit the formulation of queries with tree-valued results. Most of the work in this direction has been done in the context of structured document queries and semi-structured data [1]. We follow this line of research and focus on structured documents, like for instance XML documents, as the prime example of tree-structured data. We model documents as labeled trees where the children of a node are ordered. There is no restriction on the number of children of a vertex.

Although there is no generally accepted transformation language for structured documents, several ones have emerged during the last years, including XML-QL [12], XSLT [10], XQL [26], specifically for XML, and Lorel [3], StruQL [15], and UnQL [7], for the semi-structured data model. Almost all these languages can be divided into a pattern language part and a constructing part [14]. The purpose of the pattern language is to identify the different parts of the document that have to be combined, possibly after some more manipulation, to obtain the output document. The constructing part indicates how the output should be assembled. Pattern languages, therefore, form the basic building blocks of more general query languages transforming documents into other documents. Clearly, the choice of the pattern language can affect tremendously the expressive power of the overall query language.

As we feel that query languages for tree-structured data should be based on an equally robust foundation as those for relational data, we propose monadic second-order logic (MSO) as a benchmark for pattern languages for structured documents. In brief, MSO is first-order logic (FO) extended with set quantification. An MSO formula $\varphi(x_1, \dots, x_n)$ can readily be used to express patterns. Indeed, it just selects all tuples of vertices $v_1, \dots, v_n$ of a tree $\mathbf{t}$ for which $\mathbf{t} \models \varphi[v_1, \dots, v_n]$. Further, MSO is an expressive and versatile logic: on trees, for instance, it captures many robust formalisms, like regular tree languages [31], query automata [22], finite-valued attribute grammars [23; 21], ... Finally, MSO can take the inherent order of children of vertices into account, a desirable property for XML pattern languages [14; 27]. On the negative side, however, MSO suffers from a severe drawback: although MSO properties

[*]Research Assistant of the Fund for Scientific Research, Flanders.

of trees can be evaluated by a linear time algorithm, if they are specified, e.g., by a bottom-up tree automaton, the formula-evaluation problem (i.e., when a tree and a formula are given) is PSPACE-complete and the time of the natural evaluation algorithm is exponential in the size of the tree.

Before we define a fragment of MSO logic which avoids this shortcoming, we first investigate the expressive power of the pattern languages of the structured document transformation languages mentioned above.

Regular path expressions

The current languages can be naturally modeled as FO extended with *vertical regular path expressions.* The latter are the usual constructs expressing regularity conditions on strings formed by the labels of vertices on paths in the tree. To obtain the logic FOREG, we also add *horizontal* regular path expressions which allow to express regularity conditions on strings induced by the labels of siblings. A natural generalization, motivated by the pattern language of XSLT and XQL, is embodied in the logic FOREG*. This logic is the variant of FOREG where regular expressions can be over formulas (see Section 3 for a formal definition). Clearly, FOREG is a sublogic of FOREG*. By employing suitable pebble games we show that FOREG* is strictly more expressive than FOREG. Clearly, the major shortcoming of FO augmented with regular path expressions is that it can only look along paths of the input document. We confirm this intuition by formally proving that FOREG* cannot define the set of all trees representing Boolean circuits evaluating to true (this query can be defined in MSO). We chose this particular query only to facilitate our proof. We give in Section 3 an example of a more realistic query exemplifying the same weakness of FOREG*.

Guarded MSO quantification

Intuitively, FOREG and FOREG* lack expressive power because their second-order quantification is restricted to paths of the input tree. On the other hand, as mentioned above, full MSO logic is not very handy and only expensive evaluation algorithms are known. We introduce a fragment of MSO logic that tries to circumvent these problems as follows: (i) we specify vertices v by combining conditions on the path from the root to v and conditions on the subtree rooted at v; (ii) we allow, in the spirit of FOREG*, the use of vertical and horizontal regular expressions *over formulas*; and (iii) for the specification of subtree properties we propose a guarded fragment of MSO-logic which allows for an efficient evaluation algorithm.

Putting these ingredients together we end up with a logic which can express all MSO patterns on trees and can be evaluated in linear time in the tree-size and exponential time in the formula size. Furthermore, its restricted syntax admits the construction of an equally well-behaved pattern language that allows the specification of queries without explicit use of (first- or second-order) variables.

Related Work

Various topics have been studied w.r.t. vertical regular path expressions: implication of path constraints [4; 8], optimization [19], and rewriting of queries [9]. To the best of our knowledge, the issue of *expressiveness* of FO extended with regular path expressions has not been addressed before. Horizontal regular path expressions are studied before by various authors [11; 24; 21]. Neither of these consider expressibility issues w.r.t. these patterns.

By employing the connection between logic on trees and automata, MSO has been successfully implemented in the MONA project [17]. One of the goals in this paper, however, is not to implement all of MSO but rather to find an efficient fragment of MSO which is, with respect to tree-structured data, equivalent to MSO.

MSO was already considered as a pattern language by Maneth and the first author [18] to enhance the expressiveness of $\mathcal{DTL}^{\text{reg}}$, an initial formal model of XSL. They did not consider efficient versions of MSO or connections with logics with path expressions.

Guarded fragments of first-order and fixpoint logic have been mainly investigated w.r.t. the explanation of decidability questions of various modal logics [6]. Very little work has been devoted to the complexity of their model checking problem. We mention Alechina and Immerman [5], and Gottlob, Grädel, and Veith [16] who defined a guarded transitive closure logic and a guarded fragment of datalog, respectively, admitting linear time model checking. A crucial difference with the above mentioned guarded logics is that we consider trees where children of vertices are ordered.

Overview

In Section 2, we define our basic notation for trees and logics. In Section 3, we investigate the expressive power of first-order logic augmented with regular expressions. Section 4 deals with the efficiently evaluable MSO fragment that we propose. In Section 5, we exemplify a prototypical pattern language. Finally, Section 6 discusses our results and suggests some topics for further research.

2. PRELIMINARIES

Trees

In general, tree-structured data might contain entries of arbitrary types at a single vertex. E.g., in structured documents the content of a single vertex could be a large piece of text. Nevertheless, for many queries against such data only a fixed number of atomic properties of each vertex (or edge) is important.[1] E.g., a query might ask whether at some vertex the word "computer" occurs. The evaluation of such queries can be divided into two steps. First, all the atomic properties of vertices and edges are evaluated. This step can be understood as a reduction of the original tree to a tree which carries only vertex labels and edge labels for all properties that are mentioned in the query. In the second step, the non-atomic part of the query is evaluated against this new tree. In this paper, we focus on the second step and assume that the evaluation of atomic properties in the first step is always efficient. Note that this means that the same data might be viewed as a labeled tree in different ways, depending on the actual query. It also means that

[1]We do not address in this paper the important feature of comparisons of values of different vertices. For a discussion of this point we refer to Section 6.

the queries that we consider do not support join operations based on values at vertices.

As each vertex in a tree has (at most) one incoming edge it does not make a difference whether we represent edge labels at edges or at vertices. For convenience, we will therefore assume that labels occur only at vertices. We will denote the set of vertex labels by Σ. Formally, we define trees as rooted, directed graphs, where the children of a node are ordered. There is no restriction on the number of children of a node. Therefore, we refer to such trees as unranked. A *tree domain* τ *over* $\mathbb{N}$ is a subset of $\mathbb{N}^*$, such that if $v \cdot i \in \tau$, where $v \in \mathbb{N}^*$ and $i \in \mathbb{N}$, then $v \in \tau$. Here, $\mathbb{N}$ denotes the set of natural numbers. If $i > 1$ then also $v \cdot (i-1) \in \tau$. The empty sequence, denoted by ε, represents the root. A Σ-*tree* is a pair $\mathbf{t} = (\text{dom}(\mathbf{t}), \text{lab}_\mathbf{t})$, where $\text{dom}(\mathbf{t})$ is a tree domain over $\mathbb{N}$, and $\text{lab}_\mathbf{t}$ is a function from $\text{dom}(\mathbf{t})$ to Σ. Most of the time we say simply *tree* rather than Σ-tree. The subtree of $\mathbf{t}$ rooted at v is denoted by $\mathbf{t}_v$. If $v_1, \ldots, v_n$ are vertices such that no v_i is an ancestor of a v_j with $i \neq j$, then the *envelope of* $\mathbf{t}$ *at* $v_1, \ldots, v_n$, denoted by $\overline{\mathbf{t}_{v_1,\ldots,v_n}}$, is the tree obtained from $\mathbf{t}$ by deleting the subtrees rooted at $v_1, \ldots, v_n$, but keeping the vertices $v_1, \ldots, v_n$. Note that $\mathbf{t}_v$ and $\overline{\mathbf{t}_v}$ have v in common.

Logic

A Σ-tree $\mathbf{t}$ can be naturally viewed as a finite structure (in the sense of mathematical logic [13]) over the binary relation symbols E and $<$, and the unary relation symbols $(O_\sigma)_{\sigma \in \Sigma}$. E is the edge relation and equals the set of pairs $(v, v \cdot i)$ for every $v, v \cdot i \in \text{dom}(\mathbf{t})$. The relation $<$ specifies the ordering of the children of a node, and equals the set of pairs $(v \cdot i, v \cdot j)$, where $i < j$ and $v \cdot j \in \text{dom}(\mathbf{t})$. By S we denote the corresponding (partial) successor relation. For each σ, O_σ is the set of nodes that are labeled with a σ. We further make use of the binary predicate $\prec$ which is always interpreted as the transitive closure of the edge relation. We denote by $u \preceq v$ that $u = v$ or $u \prec v$, and by $u \leq v$ that $u = v$ or $u < v$.

As mentioned before, Monadic second-order logic (MSO) allows the use of *set variables* ranging over sets of nodes of a tree, in addition to the individual variables ranging over the nodes themselves as provided by first-order logic. We refer the unfamiliar reader to [13; 31].

Queries

As argued by Fernandez, Siméon, and Wadler [14] for the case of XML, queries on tree-structured data consist roughly of a *pattern* clause and a *constructor* clause. The purpose of the pattern language is to identify the different parts of the document that have to be combined to obtain the output document. The constructing part, on the other hand, indicates how the selected parts should be assembled. Such queries can, for instance, be written as

$$\texttt{WHERE}\ \varphi_1(\bar{x}_1), \ldots, \varphi_n(\bar{x}_n),\ \texttt{CONSTRUCT}\ \text{result}(\mathbf{t}),$$

where the φ_i's are patterns selecting vertices and $\mathbf{t}$ is a tree containing at leaves special constructs like yield(x),[2] lab(x), subtree(x) indicating that at this position the yield, the label, or the subtree rooted at the matched vertex for x should be plugged in. In this paper, we restrict attention to pattern languages.

[2]The yield of a vertex is the string obtained by concatenating the labels of the leaves of the subtree rooted at this vertex.

Path formulas

We describe next, the framework for regular path expressions over formulas. It should be noted here, and will be emphasized again in Section 3, that existing query languages for structured documents allow regular path expressions, sometimes even also over formulas. In the definition of *path formulas*, we do not specify the exact formulas over which the regular expressions are defined; they will be specified later.

- If P is a regular expression (in the usual sense) over formulas with (at least) two free variables s, t then $[P]^{\downarrow}_{s,t}(x, y)$ is a (*vertical path*) formula with free variables x, y and those occurring in formulas of P.
- If P is a regular expression over formulas with (at least) one free variable s then $[P]^{\rightarrow}_{s}(x)$ is a (*horizontal path*) formula with free variable x and those occurring in formulas of P.

The semantics of such formulas is defined as follows. Let $\varphi = [P]^{\downarrow}_{s,t}(x, y)$ be a vertical path formula and let the semantics for all formulas that are used in P be already defined. Let $\mathbf{t}$ be a tree and let v, w be vertices of $\mathbf{t}$. We assume interpretations for the free variables occurring in formulas in P. Then, $\mathbf{t} \models \varphi[v, w]$, iff $v \prec w$ and there is a labeling of the edges on the path from v to w with formulas that are used in P, such that (1) each edge (u, u') is labeled with a formula $\theta(s, t)$ such that $\mathbf{t} \models \theta[u, u']$, and (2) the sequence of labels along the path from v to w is matched by P. Let $\psi = [P]^{\rightarrow}_{s}(x)$ be a horizontal path formula and let again the semantics for all formulas in P be already defined. Then $\mathbf{t} \models \psi[v]$, iff there is a labeling of the children of v with formulas that are used in P, such that (1) each child w of v is labeled with a formula $\theta(s)$ such that $\mathbf{t} \models \theta[w]$, and (2) the sequence of labels is matched by P. The formula

$$\exists y \exists z [(s = y)(O_a(s))^* O_b(s)(z = s)]^{\rightarrow}_{s}(x),$$

for instance, says that there are children y and z of x such that the vertex labels between y and z are matched by a^*b.

Ehrenfeucht games

In the the proofs of Theorem 1 and Theorem 2 we use variations of the k-round MSO game which we will define next. For a sequence of vertices $\bar{v}$ of a tree $\mathbf{t}$, and a sequence of sets of vertices $\overline{T}$ of $\mathbf{t}$, we write $(\mathbf{t}, \bar{v}, \overline{T})$ to denote the finite structure that consists of $\mathbf{t}$, the sets $\overline{T}$, and the distinguished vertices $\bar{v}$. Let $\mathbf{t}_1$ and $\mathbf{t}_2$ be two trees, $\bar{v}_1$ a sequence of vertices of $\mathbf{t}_1$, $\bar{v}_2$ a sequence of vertices of $\mathbf{t}_2$, $\overline{T}_1$ a sequence of sets of vertices of $\mathbf{t}_1$, $\overline{T}_2$ a sequence of sets of vertices of $\mathbf{t}_2$, and k a natural number. We write $(\mathbf{t}_1, \bar{v}_1, \overline{T}_1) \equiv^{\text{MSO}}_k (\mathbf{t}_2, \bar{v}_2, \overline{T}_2)$ and say that $(\mathbf{t}_1, \bar{v}_1, \overline{T}_1)$ and $(\mathbf{t}_2, \bar{v}_2, \overline{T}_2)$ are k-*equivalent*, if for each MSO sentence φ of quantifier depth at most k it holds

$$(\mathbf{t}_1, \bar{v}_1, , \overline{T}_1) \models \varphi \Leftrightarrow (\mathbf{t}_2, \bar{v}_2, \overline{T}_2) \models \varphi.$$

That is, $(\mathbf{t}_1, \bar{v}_1, \overline{T}_1)$ and $(\mathbf{t}_2, \bar{v}_2, \overline{T}_2)$ cannot be distinguished by MSO sentences of quantifier depth (at most) k. It follows from the definition that $\equiv_k^{\mathrm{MSO}}$ is an equivalence relation. Moreover, k-equivalence can be nicely characterized by Ehrenfeucht games. The k-round MSO game on two structures $(\mathbf{t}_1, \bar{v}_1, \overline{T}_1)$ and $(\mathbf{t}_2, \bar{v}_2, \overline{T}_2)$ is played by two players, the *spoiler* and the *duplicator*, in the following way. In each of the k rounds the spoiler decides whether he makes a *point move* or a *set move*. If the i-th move is a point move, then the spoiler selects a vertex $u_i \in \mathrm{dom}(\mathbf{t}_1)$ or $w_i \in \mathrm{dom}(\mathbf{t}_2)$, and the duplicator answers with an element of the other structure. If the i-th move is a set move, then the spoiler selects a subset $P_i \subseteq \mathrm{dom}(\mathbf{t}_1)$ or $Q_i \subseteq \mathrm{dom}(\mathbf{t}_2)$, and the duplicator chooses a set of the other structure. After k-rounds, there are elements $u_1, \ldots, u_l$ and $w_1, \ldots, w_l$ that were chosen in the point moves in $\mathrm{dom}(\mathbf{t}_1)$ and $\mathrm{dom}(\mathbf{t}_2)$, respectively, and there are sets $P_1, \ldots, P_n$ and $Q_1, \ldots, Q_n$ that were chosen in the set moves in $\mathrm{dom}(\mathbf{t}_1)$ and $\mathrm{dom}(\mathbf{t}_2)$, respectively. The duplicator wins the game if the mapping which maps u_i to w_i is a partial isomorphism from $(\mathbf{t}_1, \bar{v}_1, \overline{T}_1, \overline{P})$ to $(\mathbf{t}_2, \bar{v}_2, \overline{T}_2, \overline{Q})$. The following is well known [13].

Proposition 1. *The duplicator has a winning strategy in the k-round MSO game on $(\mathbf{t}_1, \bar{v}_1, \overline{T}_1)$ and $(\mathbf{t}_2, \bar{v}_2, \overline{T}_2)$ iff $(\mathbf{t}_1, \bar{v}_1, \overline{T}_1) \equiv_k^{MSO} (\mathbf{t}_2, \bar{v}_2, \overline{T}_2)$.*

3. PATTERN LANGUAGES BASED ON REGULAR EXPRESSIONS

The logic FOREG* is obtained by augmenting first-order logic with formulas of the forms $[P]_s^{\rightarrow}(x, y)$ and $[P]_{s,t}^{\downarrow}(x)$, where the formulas in P are again FOREG* formulas. The logic FOREG is defined similarly, but here the formulas in P are restricted to vertex label predicates only. For convenience, we write in regular expressions, for instance, a^*b rather than $(O_a(x))^*\, O_b(x)$.

The definition of FOREG* is motivated by the pattern language of XSLT and XQL that allow to express *filters*, enclosed between brackets, in their patterns. For instance, the pattern `a[e][b//d]` selects all `a`-labeled vertices that have both an `e`-labeled child and a `b`-labeled child that has a `d`-labeled ancestor. As filter expressions can be nested, FOREG* embodies a powerful generalization of this concept. FOREG on the other hand, forms an abstraction of the pattern language of most other document transformation languages.

If regular expressions are restricted to be star-free,[3] it is readily shown by induction of the structure of FOREG* formulas that FOREG* = FOREG = FO (recall that $\prec$ is available). However, without this restriction FOREG* is strictly more powerful.

Theorem 1. *FOREG* is strictly more expressive than FOREG.*

We only give a hint of the proof. First, we mention that we can encode strings over the alphabet $\{a, b\}$ by binary trees over the alphabet $\{a, b, c\}$. For instance, the string aba is represented by the binary tree $c(a, c(b, c(a, c)))$. By employing suitable Ehrenfeucht games, we can show that FOREG cannot define the set of trees representing strings with an even number of alternations from a's to b's. The intuition is that the regular expressions issued in FOREG formulas have, essentially, no access to the a- and b-labeled vertices: they can only check properties of strings consisting entirely of c-labeled vertices. However, the above query is readily definable in FOREG*.

Intuitively, FOREG* and FOREG, can only look along paths of the input document. By formally proving that FOREG* can not define the class of trees representing Boolean circuits evaluating to true, we confirm the intuition that FOREG* *fails to perceive the input document as a whole.*

Theorem 2. *FOREG* cannot define the class of trees representing Boolean circuits evaluating to true.*

Proof. It is difficult to define Ehrenfeucht games for FOREG* directly. We therefore prove a more general result. We first observe that FOREG* can be defined in chain logic ($\mathrm{MSO}^{\mathrm{chain}}$). This logic is the restriction of MSO that only allows to quantify over sets that are horizontal or vertical chains. We then show that $\mathrm{MSO}^{\mathrm{chain}}$ (for which Ehrenfeucht games are known) cannot define the wanted query. It should be stressed that the vertices in a horizontal chain have to have the same parent vertex. Therefore, quantification of horizontal chains is weaker than quantification of *antichains* in, e.g., [29; 31].

The syntax of $\mathrm{MSO}^{\mathrm{chain}}$ is the same as for MSO. The only difference is that quantified set variables can only be interpreted by chains which are defined as follows. For a tree $\mathbf{t}$ and a set of nodes T, we say that T is a *chain* when one of the following holds: (i) for all $u, v \in T$, $u \preceq v$ or $v \preceq u$; or (ii) for all $u, v \in T$, $u \leq v$ or $v \leq u$. We call the former a *vertical* chain, and the latter a *horizontal* chain.

Using the known fact that regular string languages are definable in MSO (and hence in $\mathrm{MSO}^{\mathrm{chain}}$), one can show by an easy induction on the structure of FOREG* formulas that FOREG* $\subseteq \mathrm{MSO}^{\mathrm{chain}}$.

We now show that the wanted query cannot be defined in $\mathrm{MSO}^{\mathrm{chain}}$. We use the alphabet $\Sigma = \{\mathrm{AND}, \mathrm{OR}, 0, 1\}$ and only consider trees where inner vertices are labeled with AND and OR, and leaves are labeled with 0 and 1. W.l.o.g, we can reduce the vocabulary to E, $<$, and $(O_\sigma)_{\sigma\in\Sigma}$. Assume towards a contradiction that there is an $\mathrm{MSO}^{\mathrm{chain}}$ formula φ of quantifier depth k defining the set of trees representing Boolean circuits evaluating to true. We will use the k-round $\mathrm{MSO}^{\mathrm{chain}}$ game. This game is defined as the k-round MSO game where set moves are restricted to horizontal and vertical chains only. We play the game on the trees $\mathrm{AND}(0, r, h)$ and $\mathrm{AND}(1, r, h)$, and $\mathrm{OR}(0, r, h)$ and $\mathrm{OR}(1, r, h)$ defined as follows. For all $r \geq 0$ and $i \in \{0, 1\}$, define $\mathrm{AND}(i, r, 0) = \mathrm{OR}(i, r, 0) = i$. For all $r \geq 0$ and $h \geq 0$, define

$$\begin{array}{lcl}
\mathrm{AND}(0, r, h+1) & := & \mathrm{AND}(\mathrm{OR}(1, r, h)^{\times r}, \mathrm{OR}(0, r, h), \mathrm{OR}(1, r, h)^{\times r}), \\
\mathrm{OR}(0, r, h+1) & := & \mathrm{OR}(\mathrm{AND}(0, r, h)^{\times 2r+1}), \\
\mathrm{AND}(1, r, h+1) & := & \mathrm{AND}(\mathrm{OR}(1, r, h)^{\times 2r+1}), \text{ and} \\
\mathrm{OR}(1, r, h+1) & := & \mathrm{OR}(\mathrm{AND}(0, r, h)^{\times r}, \mathrm{AND}(1, r, h), \mathrm{AND}(0, r, h)^{\times r}).
\end{array}$$

[3]Star-free regular expressions are regular expressions defined from the alphabet symbols, the empty string, and the empty set, by the operators concatenation, union, and negation.

Here, for a tree $\mathbf{t}$, $\mathbf{t}^{\times i}$ denotes the sequence $\mathbf{t}, \dots, \mathbf{t}$ (i times). We start with some observations concerning these trees. All internal vertices have exactly $2r+1$ children; all vertices of the same height are labeled with the same label; and the labels of the levels alternate between AND and OR. The root of each $\text{AND}(i,r,h)$ is labeled with AND, while the root of each $\text{OR}(i,r,h)$ is labeled with OR. All trees $\text{AND}(0,c,h)$ and $\text{OR}(0,c,h)$ evaluate to 0, while all trees $\text{AND}(1,c,h)$ and $\text{OR}(1,c,h)$ evaluate to 1. We refer to the former as 0-trees and to the latter as 1-trees.

Clearly, for all r and h, the trees $\text{AND}(0,r,h)$ and $\text{AND}(1,r,h)$, and the trees $\text{OR}(0,r,h)$ and $\text{OR}(1,r,h)$ are isomorphic if the labels of vertices are not taken into account. Let π denote this isomorphism (it will always be clear from the context whether we consider AND or OR trees and what the values of r and h are). We can say even more: the trees $\text{OR}(0,r,h)$ and $\text{OR}(1,r,h)$, and $\text{AND}(0,r,h)$ and $\text{AND}(1,r,h)$ are identical apart from the label of one leaf. That is, there is only one vertex that distinguishes these trees. We introduce a special name for the vertices on the path from the root to this leaf: we call them *special vertices.* We invite the reader to check that subtrees rooted at special vertices in $\text{OR}(0,r,h)$ and $\text{AND}(0,c,h)$ are 0-trees while they are 1-trees in $\text{OR}(1,r,h)$ and $\text{AND}(1,r,h)$.

Before we prove the main lemma, we state, without proof, a helpful lemma on strings for which we use the vocabulary consisting of the linear ordering $\leq$ and the symbol O_σ only.

LEMMA 1. *1. Let σ be an arbitrary symbol. For each $k \geq 0$, there are $k_1, k_2 \geq 1$ with $k_2 > k_1$ such that $\sigma^{k_1} \equiv_k^{MSO} \sigma^{k_2}$.*

2. Further, let $f := k_2 - k_1$. Then

$$(\sigma^{2k_2+1}, k_1+1+f, k_2+1+f, \{k_2+1\}) \equiv_k^{MSO} (\sigma^{2k_2+1}, k_1+1, k_2+1, \{k_1+1\}).$$

The result now follows from the next lemma. Stated as such, the lemma is too strong as we only need to show, for instance, that $\text{OR}(0,k_2,h) \equiv_k^{\text{chain}} \text{OR}(1,k_2,h)$. However, we need the distinguished constant and the distinguished set in the inductive proof.

LEMMA 2. *For all $k \geq 1$, $h > 2k$, and k_1 and k_2 satisfying the conditions of Lemma 1(1), the duplicator wins the k-round MSO^{chain} game on*

$$(\text{OR}(0,k_2,h), \varepsilon, \{\varepsilon\}) \text{ and } (\text{OR}(1,k_2,h), \varepsilon, \{\varepsilon\}),$$

and on

$$(\text{AND}(0,k_2,h), \varepsilon, \{\varepsilon\}) \text{ and } (\text{AND}(1,k_2,h), \varepsilon, \{\varepsilon\}).$$

PROOF. (sketch) The proof proceeds by induction on k and clearly holds for $k=1$. Therefore, assume $k>1$. We only discuss moves of the spoiler in $\mathbf{t}$. We start with point moves. Suppose the spoiler picks a vertex u in $\mathbf{t}$. We denote the special vertex in $\mathbf{t}$ of height $2k-1$ by c and $\pi(c)$ by d. We distinguish two cases.

- u does not occur in $\mathbf{t}_c$. Then define v as $\pi(u)$. It suffices to show that the duplicator wins the $(k-1)$-round MSO$^{\text{chain}}$ games on $(\overline{\mathbf{t}_c}, c, u, \{\varepsilon\})$ and $(\overline{\mathbf{s}_d}, d, v, \{\varepsilon\})$, and on $(\mathbf{t}_c, \varepsilon)$ and $(\mathbf{s}_d, \varepsilon)$. The duplicator wins the first game as both structures are isomorphic; he wins the latter game by induction.
- Suppose u occurs in the subtree $\mathbf{t}_c$. We abuse notation. When we say $(\mathbf{t}_c, u)$ then we mean the tree $\mathbf{t}_c$ with the distinguished vertex in $\mathbf{t}_c$ that corresponds to u in $\mathbf{t}$. Let $f := k_2 - k_1$. Observe that the trees rooted at c and $c+f$ are both $\text{AND}(0,k_2,2k-1)$ trees, and that the trees rooted at $d-f$ and at d are an $\text{AND}(0,k_2,2k-1)$ and an $\text{AND}(1,k_2,2k-1)$ tree, respectively. We are going to fool the spoiler by mapping $\mathbf{t}_c$ to $\mathbf{s}_{d-f}$ and $\mathbf{t}_{c+f}$ to $\mathbf{s}_d$. Define v as the vertex in $\mathbf{s}_{d-f}$ occurring in $\mathbf{s}_{d-f}$ on the same position as u occurs in $\mathbf{t}_c$. It suffices to show that the duplicator wins the $(k-1)$-round MSO$^{\text{chain}}$ games on $(\mathbf{t}_c, \varepsilon, u)$ and $(\mathbf{s}_{d-f}, \varepsilon, v)$, on $(\mathbf{t}_{c+f}, \varepsilon)$ and $(\mathbf{s}_d, \varepsilon)$, and on $(\overline{\mathbf{t}_{c,c+f}}, c, c+f, \{\varepsilon\})$ and $(\overline{\mathbf{s}_{d-f,d}}, d-f, d, \{\varepsilon\})$. The duplicator wins the first game as both structures are isomorphic; he wins the second game by induction. By using Lemma 1 it can be shown that the duplicator wins the last game.

We briefly discuss set moves. Suppose the spoiler picks a horizontal chain T in $\mathbf{t}$. Then we have to distinguish two cases: (i) T occurs in $\overline{\mathbf{t}_c}$; and (ii) T occurs in $\mathbf{t}_c$. In the first case we color $\overline{\mathbf{s}_d}$ with $\pi(T)$ and continue playing on the parts above c and d, and below c and d. In the second case we again apply the above mentioned switching trick. If the spoiler picks a vertical chain T in $\mathbf{t}$, then we essentially have to make the same case distinction. □

This concludes the proof of Theorem 2.

The above query was only chosen to facilitate the separation proof. We next give a more realistic presentation of the same example. Suppose in an enterprise we have the hierarchical structure described by the following DTD:

```
<!ELEMENT group    (manager, group+) | employee+ >
<!ELEMENT manager   employee >
<!ELEMENT employee (name, eval) >
<!ELEMENT eval     (good | medium | bad) >
```

An enterprise consists of several groups, each group consists of a manager and a number of subgroups or consists of a number of employees only. Each employee gets an evaluation on a monthly basis. The enterprise employs the following bonus system. A group receives a bonus if one of the following holds: (i) its manager got a good evaluation and at least one of its subgroups received a bonus; (ii) its manager got a medium evaluation but all of its subgroups received a bonus; or (iii) the group consists of employees only and they all got a good evaluation.

FOREG* can not define the set of all groups that receive a bonus, although it can be defined in MSO.

4. EXPRESSIVE AND EFFICIENT FRAGMENTS OF MSO-LOGIC

We have seen in Section 3 that allowing only path conditions falls short of capturing MSO logic. On the other hand, allowing arbitrary MSO formulas would yield a complexity wise unmanagable language. In the rest of this paper, we restrict attention to unary patterns. (Non-unary patterns will be discussed in Section 6.) Intuitively, a unary pattern selects a set of vertices with a certain property from a tree. We consider it natural to specify such properties by using *path formulas* $\varphi(x, y)$ and *subtree formulas* $\psi(x)$. Intuitively, we restrict path and subtree formulas in a way such that whether $\mathbf{t} \models \psi[v]$ only depends on $\mathbf{t}_v$ and whether $\mathbf{t} \models \varphi[u, w]$ only depends on the substructure $(\mathbf{t}_u, w)$.[4] The purpose of this notion is to modularize the formulation of unary patterns. The idea is to formulate all conditions on the subtree rooted at a vertex v in subtree formulas and to collect the necessary information about the envelope of v (i.e., the rest of the tree) along the path from the root to v.

In brief, for path formulas we will use formulas of the form $[P]^{\downarrow}_{s,t}(\text{root}, x)$. In subtree formulas we will allow formulas of the form $(\exists X)(x \prec X \wedge \psi(x, X))$. Here, $x \prec X$ indicates that X only contains vertices that are descendants of x. The advantage of this modularization is two-fold: (i) it facilitates specification of unary patterns (Section 5); (ii) it allows the definition of an efficient logic equivalent to full MSO.

In this section, we introduce a logic that has the full expressive power of MSO logic but can be evaluated rather efficiently: in time linear in the tree size and exponential in the formula size. We call this fragment *efficient tree logic (ETL)*. This logic combines a guarded fragment of MSO logic with horizontal and vertical path expressions. The guarding is such that quantifiers are only allowed to bind sets or vertices that are below the currently visited vertex. This will enable an efficient bottom-up evaluation for formulas.

In a first step, we will introduce a subset of ETL that is able to evaluate properties of vertices that only depend on the subtrees rooted at them. We assume a countable set of set variables $X, Y, Z, X_1, X_2, \ldots$ and a countable set of first-order variables $x, y, z, x_1, x_2, \ldots$. We associate with each formula φ a set free(φ) of its free first-order variables and a set Free(φ) of its free second-order variables. Here, the notion of free variables is as in MSO logic, unless otherwise stated. The following list summarizes the formation rules for ETL formulas.

(i) Atomic formulas are ETL formulas (recall that $E(x, y)$, $x \prec y$, $X(x)$ and $(O_\sigma(x))_{\sigma\in\Sigma}$ are atomic formulas).

(ii) If φ is an ETL formula, free$(\varphi) = \{x, y\}$, and φ contains at most one free set variable, then $\exists y(E(x, y) \wedge \varphi)$ is an ETL formula.

(iii) If φ is an ETL formula, and free$(\varphi) = \{x, y\}$, and φ contains at most one free set variable, then $\exists y(x \prec y \wedge \varphi)$ is an ETL formula.

(iv) If φ is an ETL formula, free$(\varphi) = \{x\}$, Free$(\varphi) = \{X\}$ and φ does not contain subformulas constructed by rule (vi),then $\exists X(x \prec X \wedge \varphi)$ is an ETL formula.

(v) If P is a regular expression over the set of ETL formulas with only the free vertex variable s, then $\psi = [P]^{\rightarrow}_{s}(x)$ is an ETL formula with free$(\psi) = \{x\}$ and Free(ψ) contains the free set variables occurring in formulas of P.

(vi) If P is a regular expression over the set of ETL formulas with only the free variables s and t then $[P]^{\downarrow}_{s,t}(x, y)$ is an ETL formula with free variables x, y.

(vii) Boolean combinations of ETL formulas are ETL formulas.

By $x \prec X$ we express that all vertices in X are descendants of x, i.e, $\forall y(X(y) \rightarrow x \prec y)$. Additionally, we require that each subformula ϕ fulfills $|\text{free}(\phi)| \leq 2$ and $|\text{Free}(\phi)| \leq 1$. Formulas that are constructed by one of the rules (i)-(vi) are called *basic*. We call formulas without free set variables *set-closed*. Note, that we do not allow vertical path expressions within the scope of set quantification. The reason is that otherwise we would not be able to evaluate ETL as efficiently as desired. We study a variant of ETL, where the occurence of vertical path expressions is unlimited at the end of the section. For convenience, we assume that no variable is quantified twice in an ETL formula. As is usual, we use conjunction, implication and universal quantification as abbreviations. It should also be noted that all formulas that can be constructed by the given rules have at least one free vertex variable. Furthermore, it is easily seen that for each such formula φ, each tree $\mathbf{t}$ and each vertex v, it holds $\mathbf{t} \models \varphi[v]$ iff $\mathbf{t}_v \models \varphi[v]$. Hence, intuitively, such formulas can only express properties of subtrees. We show first that ETL formulas can define all subtree properties that can be defined by MSO formulas.

PROPOSITION 2. *For every MSO formula $\varphi(x)$ there is an ETL formula $\psi(x)$ such that, for every tree $\mathbf{t}$ and every vertex v of $\mathbf{t}$, it holds $\mathbf{t}_v \models \varphi[v]$ if and only if $\mathbf{t}_v \models \psi[v]$.*

PROOF. Let $\varphi(x)$ be an MSO formula. First, it is well-known (see [31] for the ranked case and [20] for the unranked case) that there is a deterministic bottom-up tree automaton M, with a distinguished set A of states, which assumes in its unique bottom-up traversal on $\mathbf{t}$ a state from A at a vertex v iff $\mathbf{t}_v \models \varphi[v]$. It should be noted that, in the context of unranked trees, this automaton makes use of several string automata which compute the state of M at a vertex v from the states of the children of v (cf. the proof of Theorem 4). The idea of our proof, therefore, is to construct an ETL formula ψ such that $\mathbf{t}_v \models \psi[v]$ if and only if M assumes a state from A at v.

It should be noted first that the obvious MSO formula for such a simulation is not an ETL formula. This formula would guess the state of the automaton for each vertex of the tree, by existential quantification of some set variables. The number of set variables would depend on the number of

[4]$(\mathbf{t}_u, w)$ denotes the structure consisting of $\mathbf{t}_u$ with w as a distinguished constant.

states of M. This can not be done directly in ETL as ETL formulas can only make use of one set at a time.[5]

However, M can be simulated by a formula which only quantifies (existentially) over one set variable. The proof technique is an adaptation of the proof of Thomas that each regular string language can be described by an existential MSO formula which quantifies only one set variable [28; 25]. The basic idea is to guess states only for a sparse subset of the set of all vertices which is distributed evenly over the tree and to bridge the gap between these vertices in a first-order manner. Of course, the computations of the string automata of M can be simulated by horizontal regular expressions.

Let $q_1, \dots, q_l$ be the states of M, let A denote the set of accepting states, and set $k := 2l+1$. The ETL formula which we construct is of the form $\exists X(x \prec X \wedge \psi)$. Intuitively, a set W which makes $\psi[W,v]$ true should serve two purposes.

First, it should distinguish all vertices of $\mathbf{t}$ that have a depth which is a multiple of k. This is done by putting the right most child of each such vertex into W. Second, it should encode the states of M (of these vertices). This is done in two ways, depending on the degree of a vertex v. If v has large degree ($> l$) then the state q_i of M at v is indicated by putting the i-th child (from left-to-right) of v into W. If v has small degree then its state is only encoded into W if its depth is a multiple of k. In this case, to encode state q_i, all right most children of vertices which are i levels below v are put into W. If v has small height then the state is not encoded at all. We call W the correct set for the computation of M of t if it has these two properties. We say that W accepts v if, for some accepting state q_i of M, either the i-th child of v is in W or the right most child of every vertex i levels below v is in W. If t is a tree of depth greater or equal to l then t is accepted by M iff it has an accepting set W. Trees of smaller depth are handled separately, in a similar way.

It remains to show that there is an ETL formula $\theta(X)$ which checks that a set W is correct and accepts v. The construction of θ is straightforward but tedious.

Basically, θ is the conjunction of three subformulas.

- The first subformula θ_1 checks that the "syntax" of W is ok. I.e.,
  - all right most children of vertices at levels that are multiples of k are in W;
  - if w is a vertex at a level kj of small degree then, for some $j \leq l$, all right most children of the vertices i levels below of w are in W; Between w and the vertices k levels below w there are no other vertices the right most child of which is in W;
  - If w is a vertex of large degree then exactly one of its first l children is in W.
- The second subformula θ_2 checks that W is consistent; Note that, if W is syntactically correct then on each path which goes down from a vertex w there exists a vertex which is at most k levels below w the state of which is indicated by W (or the length of the path is at most $2l$); For each state q of M, we construct a subformula $\psi_q(x)$ which computes the state of a vertex v from the states of vertices below w that are encoded into W. The formula θ_2 checks whether, for each w the truth values $\psi_q(v)$ are consistent with the encoded state for w.
- Finally, θ_3 checks that the state at v is accepting.

Note that we use horizontal expressions to express that, e.g., the right most child of a vertex is in W. Further, horizontal expressions are needed in θ_2 to check the consistency of states for vertices of large degree. On the other hand, ψ does not use any vertical path expressions. $\square$

As a second step, we show that the ETL formulas that we are allowed to construct so far can be evaluated efficiently.

PROPOSITION 3. *There is an algorithm which computes, given a tree* $\mathbf{t}$ *and an ETL formula* $\varphi(x)$, *for each vertex* v *of* $\mathbf{t}$ *whether* $\mathbf{t}_v \models \varphi[v]$ *in time* $O(size(\mathbf{t}))2^{O(size(\varphi))}$.

PROOF. (sketch) If Ψ is a set of formulas then a *truth assignment* for Ψ is a mapping $\alpha : \Psi \to \{0,1\}$. We write Φ for the set of set-closed subformulas of φ of types (i)-(v). For each set variable X, we denote by $\Phi(X)$ the set of basic subformulas of φ of types (i)-(iii) and (v) with free set variable X.

We sketch an algorithm which tests whether $\mathbf{t}_v \models \varphi[v]$. The basic idea is to evaluate φ bottom-up. We compute, for each vertex u of $\mathbf{t}$, the following information.

1. a tuple $b(u) = (\alpha, A)$, where α is a truth assignment for Φ and A is a set of pairs (β, o), where β is a truth assignment for Φ and o will be described below;
2. for each set variable X and for each truth assignment η for $\Phi \cup \Phi(X)$, a set $B_\eta(X,u)$ of pairs (γ, Ψ) where γ is a truth assignment for all of $\Phi(X)$ and Ψ is a set[6] of formulas of type (iii) which occur within the scope of X.

We have to spend some effort to deal with vertical path expressions. To this end, we construct, for each vertical regular expression P a non-deterministic finite automaton, M_P, which accepts the reversal of the language defined by P. We take the reversal because, although the regular expressions describe paths top-down, we have to evaluate them bottom-up. The construction of M_P can be done in time linear in the time of P. In particular, the number of states of M_P is

[5]We remark that the evaluation algorithm would have the same asymptotic behaviour, if we allowed existential quantification of several set variables as long as there is no interaction with other quantifiers. But we are interested in getting rid of multiple set variables to simplify the structure of queries.

[6]We do not make use of a truth assignment here to emphasize the fact that we are interested in formulas which can be made true.

linear in the size of P. The o-components of the pairs (β, o) above map each vertical regular expression P of φ to a set of states of M_P.

Intuitively, for a formula $\theta \in \Phi$, $b(u).\alpha(\theta)$ should be 1 iff $\mathbf{t}_u \models \theta[u]$. A pair (β, o) should be in $b(u).A$, if there is a vertex w below u such that

- for all formulas $\theta \in \Phi$, $\beta(\theta) = 1$ iff $\mathbf{t}_w \models \theta[w]$, and
- for each regular expression P occurring in a vertical expression of φ, $o(P)$ is the set of states that M_P can reach if it runs on the path from w to u.

Finally, a pair (γ, Ψ) should be in $B_\eta(X, u)$ if there is a set C on $\mathbf{t}_u$ such that,

- for each formula $\theta = \exists y(x \prec y \wedge \psi)$ in Ψ there is a strict descendant w of u, such that $\mathbf{t}_w \models \psi'[C, w]$, where ψ' is the formula obtained from ψ by replacing each maximal basic subformula $\chi_1(x)$ and $\chi_2(x, X)$ of ψ by $\eta(\chi_1)$ and $\eta(\chi_2)$, respectively, $x \prec y$ by 1, and all other atomic formulas containing both x and y by 0.[7]

The definition of the sets $B_\eta(X, u)$ deserves some explanation. The γ parts of the elements in such a set are sufficient to evaluate formulas of types (iv). The Ψ parts are used inductively, for the evaluation of subformulas of type $\psi = \exists y(x \prec y \wedge \varphi)$. Intuitively, such a formula is true at a vertex u, if there is a vertex w below u such that φ evaluates to true, if x is bound to u and y is bound to w. Hence, we need information about possible truth assignments to formulas in $\Phi(X)$ at vertices below u. This information depends on the choice of the set of vertices for X. The straightforward approach would maintain a set D consisting of sets F of truth assignments, such that for each choice for X there is a set in D which consists of all truth assignments for vertices under this choice. Unfortunately, this approach would involve objects of double exponential size. Instead, we make use of the fact, that each time at which we need information about vertices below u, there is only one truth assignment for the x-formulas of φ, which we have to consider, namely the one for u. Furthermore, this is available at the time we evaluate φ. Next, we sketch how all this information can be computed. We evaluate the vertices bottom-up. For each vertex v, the information about the subformulas of φ is also computed in a bottom-up manner with respect to the syntax tree of φ.

Let u be a vertex of $\mathbf{t}$, let $w_1, \ldots, w_m$ be the children of u and assume that we already have evaluated all vertices below u. We compute the information corresponding to u in a bottom-up manner w.r.t. the syntax tree of φ. Whenever we evaluate the information corresponding to a subformula θ of φ we may assume that the information about its set-closed subformulas has already been collected in $b(u)$. For a set variable X, we compute the sets $B_\eta(X, u)$ just before we need them to evaluate the corresponding formula of type (iv). Let us first consider a formula $\theta \in \Phi$. In particular, Free$(\theta) = \emptyset$. To determine $b(u).\alpha(\theta)$, we distinguish the following cases.

(i) If θ is atomic, then $b(u).\alpha(\theta)$ can be immediately obtained from the label of u.

(ii) If θ is of the form $\exists y(E(x, y) \wedge \psi)$, then ψ is a Boolean combination of basic formulas with free variable x, basic formulas with free variable y, atomic formulas with free variables x, y, and vertical path expressions with free variables x, y. We refer to these top-level subformulas of θ as the *maximal* subformulas. We set $b(u).\alpha(\theta)$ to 1 iff there is a child w of u such that the formula that results from ψ by replacing

- each maximal basic subformula $\chi(x)$ of ψ by $b(u).\alpha(\chi)$;
- each maximal basic subformula $\chi(y)$ of ψ by $b(w).\alpha(\chi)$;
- each occurrence of $E(x, y)$ and $x \prec y$ by 1 and, each other atomic formula with free variables x, y by 0;
- each occurrence of a vertical path expression
$$[P]^{\downarrow}_{s,t}(x, y)$$
by 1 iff there is a formula $\xi(s, t)$ in the language defined by P such that $\mathbf{t} \models \xi[u, w]$. Note that ξ is a Boolean combination of formulas with free variable s, formulas with free variable t, atomic formulas with free variables s, t, and path expressions with free variables s, t. All these have already been evaluated (at u and w, respectively);

evaluates to true.

(iii) If θ is of the form $\exists y(x \prec y \wedge \psi)$ then ψ can be of the same form, as before. There are now two possibilities: y is interpreted by a child of w of u, or by a strict descendant of the children of u. Therefore, we set $b(u).\alpha(\theta) = 1$ iff

1. there is a child w of u such that the formula obtained from ψ in the same way as in (ii) evaluates to true; or
2. there is a child w of u and a truth assignment $(\beta, o) \in b(w).A$ such that the formula that results from ψ by replacing
   - each maximal basic subformula $\chi(x)$ of ψ by $b(u).\alpha(\chi)$;
   - each maximal basic subformula $\chi(y)$ of ψ by $\beta(\chi)$;
   - each occurrence of $x \prec y$ by 1 and and all other occurrence of atomic formulas containing both variables x and y by 0;
   - each occurrence of a vertical path expression $[P]^{\downarrow}_{s,t}(x, y)$ by 1 just in case there is a state $q \in o(P)$ and a formula $\xi(s, t)$ in P such that $\mathbf{t}_u \models \xi[u, w]$ and M_P can get from state q into an accepting state via a ξ-transition ($\mathbf{t}_u \models \xi[u, w]$ is tested as in case (ii)).

   evaluates to true.

[7]Recall that there are no subformulas of type (vi) inside ψ.

(*iv*) If θ is of the form $\exists X(x \prec X \wedge \psi)$ then ψ consists of a Boolean combination of formulas with free variable x, which may also use the free set variable X. We set $b(u).\alpha(\theta)$ to 1, iff there is a (partial) entry $(\gamma, \Psi) \in B_\eta(X, u)$ (for some[8] η) such that the formula ψ' that results from ψ by replacing

- each maximal basic subformula $\chi(x)$ of ψ by $b(u).\alpha(\chi)$,
- each maximal basic subformula $\chi(X, x)$ of ψ by $\gamma(\chi)$,

becomes true.

(*v*) If θ is of the form $[P]_s^{\rightarrow}(x)$ then P contains only formulas with a single free variable s. So, we set $b(u).\alpha(\theta)$ to 1 iff the truth assignments $b(w_1).\alpha, \ldots, b(w_m).\alpha$ can be matched with P. This can essentially be tested in time $m2^{O(|P|)}$ by simulating the behaviour of the nondeterministic automaton corresponding to P.

Next, we describe the computation of $b(u).A$. An entry (β, o) is put into $b(u).A$ if

1. it holds $\beta = b(u).\alpha$ and, for each P, $o(P)$ exactly contains the initial state of M_P, or
2. there is a child w of u and an entry (β, o') in $b(w).A$ such that, for each P, $o(P)$ is the set of states that M_P can reach from states in $o'(P)$ by one transition according to a formula $\xi(s,t)$ for which $\mathbf{t}_u \models \xi[u, w]$ holds. The truth of such formulas ξ can be tested as above in the evaluation of formulas of type (*ii*).

It remains to explain, how the sets $B_\eta(X, u)$ can be computed. Let X be a set variable that occurs in φ and let η, η' be truth assignments for $\Phi \cup \Phi(X)$, where η' coincides with $b(u).\alpha$ on all set-closed subformulas that occur inside a formula with free X. Here, η' can be viewed as a guess for the formulas which hold at u. We do the following, for each choice of X, η, η'. We already said in Section 4 that a pair (γ, Ψ) should be in $B_\eta(X, u)$ if there is a set C on $\mathbf{t}_u$ such that,

- for each formula $\theta \in \Phi(X)$, $\gamma(\theta)$ is 1 iff $\mathbf{t}_u \models \theta[C, u]$; and
- for each formula $\theta \in \Psi$ there is a strict descendant w of u, such that $\mathbf{t}_w \models \psi'[C, w]$, where ψ' is the formula obtained from ψ by replacing each maximal basic subformula $\chi_1(x)$ and $\chi_2(x, X)$ of ψ by $\eta(\chi_1)$ and $\eta(\chi_2)$, respectively, $x \prec y$ by 1, and all other atomic formulas containing both x and y by 0.

First, we compute, for each child w_i, $i \leq m$, of u a set D_i of pairs (ζ, Ξ), where ζ is a truth assignment to $\Phi(X)$ and Ξ is a set of subformulas of φ of types (*ii*) and (*iii*) with free variable X. A pair (ζ, Ξ) is put into D_i, if there is a pair $(\zeta, \Gamma) \in B_{\eta'}(X, w_i)$ such that for each $\theta \in \Phi(X)$ one of the following holds.

[8]Equivalently, we could require *for all* here.

- θ is of the form $\exists y(E(x,y) \wedge \psi)$ and $\theta \in \Xi$ iff the formula ψ' that results from ψ by replacing
  - each maximal basic subformula $\chi(x)$ of ψ by $b(u).\alpha(\chi)$,
  - each maximal basic subformula $\chi(y)$ of ψ by $b(w_i).\alpha(\chi)$,
  - each maximal basic subformula $\chi(X, x)$ of ψ by $\eta'(\chi)$,
  - each maximal basic subformula $\chi(X, y)$ of ψ by $\zeta(\chi)$,
  - each occurrence of $E(x, y)$ and $x \prec y$ by 1 and, each other atomic formula with free variables x, y by 0

  evaluates to true.
- θ is of the form $\exists y(x \prec y \wedge \psi)$ and $\theta \in \Psi$ iff
  - $\theta \in \Gamma$, or
  - the formula ψ' that results from ψ as in the case before evaluates to true.

Intuitively, a pair (ζ, Γ), tells us that there is a coloring of the tree at u such that

- the formulas $\Phi(X)$ at w_i behave according to ζ, and
- the formulas $\theta \in \Gamma$ hold at u, in the subtree of $\mathbf{t}$ which consists of u, w_i and all vertices below w_i.

Now we describe how the information of the sets D_i can be collected to derive $B_\eta(X, u)$. First, we associate with every horizontal expression P which uses X a nondeterministic automaton M_P equivalent to P. Note that these automata can be chosen such they all together contain at most $O(\text{size}(\varphi))$ many states.

We use a dynamic programming approach and compute, for $i \leq m$ a set C_i consisting of tuples (Θ, Θ', o), where Θ is a set of formulas of types (*ii*) and (*iii*) with free variable X, Θ' is a set of formulas of type (*iii*) with free X and o assigns a set of states to all horizontal expressions P which use X. A tuple (Θ, Θ', o) is in C_i, iff there is a choice of tuples $(\zeta_1, \Xi_1), \ldots, (\zeta_i, \Xi_i)$, where each $(\zeta_j, \Xi_j) \in D_j$, such that the following hold.

- $\Theta = \bigcup_{j=1}^{i} \Xi_j$.
- $\Theta' = \Theta \cup \bigcup_{j=1}^{i} \{\theta | \psi'_{\eta,j} \text{ evaluates to true}\}$, where θ is of the form $\exists y(y \prec x \wedge \psi)$ and $\psi_{\eta,j}$ results from ψ by replacing all subformulas with free x according to η, all subformulas with free y according to ζ_j, $x \prec y$ by 1, and all other atomic formulas by 0.
- For all horizontal expressions P which use X, $o(P)$ contains a state q of the automaton M_P, iff M_P can reach q from the initial state by evaluating the "string" $\zeta_1, \cdots \zeta_i$.

As each tuple (Θ, Θ', o) is of linear size in size(φ), it is not hard to see that the computation of all sets C_i can be done in time $m2^{O(\text{size}(\varphi))}$.

From C_m we can now obtain $B_\eta(X, u)$ as follows. A pair (γ, Ψ) is in $B_\eta(X, u)$ if and only if there is a tuple $(\Theta, \Theta', o) \in C_m$ such that the following hold.

- γ and η' agree on all formulas in $\Phi(X)$.
- For each formula θ of types *(ii)* and *(iii)* with free X, $\gamma(\theta) = 1$ iff $\theta \in \Theta$.
- For each formula θ of type $[P]_s^{\rightarrow}(x)$ with free X, $\gamma(\theta) = 1$ iff $o(P)$ contains an accepting state of M_P.
- $\Psi = \Theta'$.

Altogether, we compute at most $2^{O(\text{size}(\varphi))}$ bits of information per vertex and each bit can be computed in essentially at most $2^{O(\text{size}(\varphi))}$ many steps. This yields the stated time bound. □

With the means which we have introduced so far, ETL formulas can only evaluate properties of vertices that only depend on the subtrees rooted at them. To get the full power of MSO logic we need a bit more. We first introduce a new formation rule for ETL formulas.

(viii) If P is a regular expression over the set of ETL formulas with free variable s but without free set variables (and possibly the formula $s = y$) then $\psi = [P]_s^{\rightarrow}(x)$ is an ETL formula with free$(\psi) = \{x, y\}$ and Free(ψ) = Free(φ).

Intuitively, this new rule allows to compare in horizontal expressions the current child of x with y. In the proof of Proposition 3, it is implicitly shown that the given algorithm also can check whether $\mathbf{t}_v \models \theta[v, w]$, for a formula $\theta(x, y)$, and vertices v and w where w is a child of v. By adapting the algorithm a little bit for the new rules *(viii)* we obtain immediately the following corollary.

Corollary 1. *There is an algorithm which computes, given a tree* $\mathbf{t}$ *and an ETL formula* $\varphi(x, y)$*, for each vertex* v *of* $\mathbf{t}$ *and each child* w *of* v *whether* $\mathbf{t}_v \models \varphi[v, w]$ *in time* $O(size(\mathbf{t}))2^{O(size(\varphi))}$.

We are ready to state the main result of this section.

Theorem 3. *For each MSO formula* $\varphi(x)$ *there is an ETL formula* ψ *which is a Boolean combination of formulas* $\theta(x)$ *and formulas* $[P]_{s,t}^{\downarrow}(root, x)$ *such that, for all trees* $\mathbf{t}$ *and all vertices* v *of* $\mathbf{t}$*, it holds* $\mathbf{t} \models \varphi[v]$ *if and only if* $\mathbf{t} \models \psi[v]$*. Furthermore, there is an algorithm which, given a tree* $\mathbf{t}$ *and such a formula* ψ*, computes in time at most* $O(size(\mathbf{t}))2^{O(size(\psi))}$ *the set of all vertices* v*, for which* $\mathbf{t} \models \psi[v]$.

Proof. (sketch) It is well-known that each MSO formula $\varphi(x)$ is equivalent to a Boolean combination of formulas $\chi(x)$ in which all quantification is restricted to vertices below x (informally, these formulas are evaluated on $\mathbf{t}_x$) and formulas $\rho(x)$ in which all quantification is restricted to vertices above x (talking about the envelope of x). We have already shown in Proposition 2 that the formulas of the former kind can be replaced by ETL formulas. Given this, is it also easy to show that formulas of the latter kind can be replaced by ETL formulas $[P]_{s,t}^{\downarrow}(\text{root}, x)$.

The evaluation algorithm can be obtained by combining the algorithms from Proposition 3 and Corollary 1 in the following way. We compute, for all vertices v and all formulas $\theta(x)$ whether $\mathbf{t}_v \models \theta[v]$. Likewise, we compute, for all subformulas $\sigma(s, t)$ that are used in expressions P of $[P]_{s,t}^{\downarrow}(\text{root}, x)$, and for all vertices v and children w of v, whether $\mathbf{t}_v \models \sigma[v, w]$. Finally, we compute, in a top-down manner, for all vertices v the set of expressions P, for which $\mathbf{t} \models [P]_{s,t}^{\downarrow}(\text{root}, v)$. This is done by simulating the top-down automata M_P' for the expressions P inductively along all paths from the root. □

For efficiency reasons we disallowed in ETL formulas the use of vertical expressions within the scope of second order quantification. The following proposition shows that the complexity of evaluating formulas without this restriction is still feasible. It is quadratic exponential in the size of the formula unlike linear exponential as in the case of ETL formulas.

Proposition 4. *Formulas that are constructed by the rules for ETL formulas with rule (iv) replaced by*

(iv') If φ *is an ETL formula, free*$(\varphi) = \{x\}$ *and Free*$(\varphi) = \{X\}$ *then* $\exists X(x \prec X \wedge \varphi)$ *is an ETL formula.*

can be evaluated in time $O(size(\mathbf{t}))2^{O(size(\varphi)^2)}$.

Proof. (sketch) We only have to adapt the algorithm of Proposition 3. It has to be extended for the case of vertical expressions within the scope of set quantification. The algorithm uses sets $B_\eta(X, u)$ with extended entries as compared with the algorithm of Proposition 3. Let X be a set variable used in formula φ and let $P_1, \ldots, P_l$ be the vertical path expressions inside the scope of the quantification of X. The entries in $B_\eta(X, u)$ are of the form (γ, Ψ), where γ is as before and Ψ is a set of tuples $(\theta, c_1, \ldots, c_l, S_1, \ldots, S_l)$ where each c_i has value 0 or 1, each S_i is a set of states of the nondeterministic automaton associated with P_i, and $\theta \in \Phi(X)$ is of type *(iii)*. Each θ occurs at most once in a tuple of a set Ψ. An entry $(\theta, c_1, \ldots, c_l, S_1, \ldots, S_l)$ should be in Ψ if (for fixed C, η and u) there is a strict descendant w of u such that

- $\mathbf{t}_w \models \psi'[C, w]$, where $\theta = \exists y(x \prec y \wedge \psi)$, ψ' is the formula obtained from ψ by replacing each maximal basic subformula $\chi_1(x)$ and $\chi_2(x, X)$ of ψ by $\eta(\chi_1)$ and $\eta(\chi_2)$, respectively, each maximal vertical path expression $[P_i]^{\downarrow}_{s,t}(x,y)$ by c_i, each $x \prec y$ by 1, and all other atomic formulas containing both x and y by 0, and

- for each $i \leq l$, S_i is the set of states that the automaton associated with P_i can reach after traversing the path from w to u.

The maintenance of this additional information is straightforward. For the complexity, it should be pointed out that each set entry $(\theta, c_1, \ldots, c_l, S_1, \ldots, S_l)$ in a set Ψ can be encoded by a string of length $O(\text{size}(\varphi))$. This is because the overall number of states in automata associated with path expressions is $O(\text{size}(\varphi))$. As, furthermore, each θ occurs at most once in Ψ, a set Ψ can be encoded by a string of length $O(\text{size}(\varphi)^2)$. Therefore, there are at most $2^{O(\text{size}(\varphi)^2)}$ different sets $B_\eta(X,u)$. $\square$

5. TOWARDS A POWERFUL PATTERN LANGUAGE FOR TREE-STRUCTURED DATA

We have just seen in Section 4 that each unary MSO pattern can be expressed by a formula which is a Boolean combination of ETL subtree formulas $\varphi(x)$ and ETL path formulas $\psi(x,y)$. As mentioned before, we believe that this kind of modularization simplifies the formulation of queries.

Nevertheless, it seems unlikely that a pattern language will be considered usable if patterns have to be formulated as logic formulas. However, the very structure of ETL formulas suggests a further step into the direction of a user-friendly pattern language. The reader should verify that, informally, at each position in an ETL formula one can talk about at most three objects, two vertices and one set.

Although these syntactic restrictions where introduced with the goal of efficient evaluation in mind, they can also be exploited to simplify the formulation of queries, in that they make it unnecessary to talk in a pattern explicitly about variables.

To illustrate how a pattern language that uses these concepts could look like we give here an example of a pattern formulated in a style that is reminiscent of SQL. Of course, a lot of work has to be done to develop a real usable pattern language from this initial idea.

The following example shows how the query described in Section 3 which selects all "good" groups in an enterprise could be formulated in such a language. Recall that this query cannot be expressed by using path expressions only. Furthermore, it should be noted that it is almost a one-to-one translation of the natural description of the query given in Section 3.

```
SELECT ALL VERTICES WHERE
  PATH IS (.* group)               (1)
 AND
  HAS TREE IN WHICH
    ALL VERTICES FULFILL
      IS NOT IN TREE
     OR
     (PATH IS (.* group)                (2)
       AND
        ([(HAS VERTEX WHICH FULFILLS
            PATH IS manager.employee     (*)
                            .eval.good)
          AND
          (HAS VERTEX WHICH FULFILLS
            PATH IS group                (*)
            AND
             IS IN TREE)]
        OR
         [(HAS VERTEX WHICH FULFILLS
            PATH IS manager.employee     (*)
                          .eval.medium)
           AND
           (ALL VERTICES FULFILL
             NOT PATH IS group           (*)
             OR
              IS IN TREE)]
        OR
         [ALL VERTICES FULFILL
            PATH IS employee.*           (*)
           AND
             ( NOT PATH is employee
               OR
                HAS VERTEX WHICH FULFILLS
                  PATH IS eval.good)]
        )
     )
```

The semantics of "`PATH IS ...` " is relative to its context: the start of the path is always the second to last mentioned vertex, the end of the path corresponds to the last vertex mentioned. For instance, in `(*)` the start vertex is the one defined by the `ALL VERTICES FULFILL` statement. In `(1)` and `(2)`, the start vertices are the root of the input tree and the root of the quantified tree, respectively. An expression .* stands for an arbitrary sequence of labels.

It remains to explain the statement `HAS SUBTREE`. Let v be a vertex of the input tree $\mathbf{t}$. Note that any set X containing v can be seen as a tree. Indeed, there is an edge from u to w in this tree, if $u \prec w$, both u and w belong to X, and no vertex on the path from u to w is in X. Hence, with the statement `HAS SUBTREE`, we we state the existence of such a tree rooted at the vertex v quantified in the `SELECT ALL VERTICES` statement.

We again emphasize that, although the statement in the example clearly has a flavor of quantification, it has a somewhat simple structure as there are no interactions between variables that are quantified at faraway places, as it might happen in a general MSO formula.

6. DISCUSSION

Our results should be considered as first steps in a whole research program whose goal it is to define a pattern language for tree-structured data that is (1) easy to use, (2) as expressive as MSO-logic, and (3) efficiently evaluable.

Although we concentrated on unary queries in this paper the results can be extended to other kinds of queries. E.g., to allow queries with a k-ary relation as result we could allow the use of variables $x_1, \ldots, x_k$ and evaluate formulas as follows. For each $(k-1)$-tuple of vertices bound to $x_2, \ldots, x_k$ we could call the evaluation algorithm for the free variable x_1 while interpreting the vertices of the $(k-1)$-tuple as labeled with new special labels. This readily results in an evaluation algorithm with time bound $(\text{size}(\mathbf{t}))^k 2^{O(\text{size}(\varphi))}$. To specify such queries in the modular spirit we proposed there is still some work to do. We expect that queries should allow to take least common ancestors of vertices into account.

We plan to consider the following two important generalizations: (i) incorporating value comparisons (that is, joins); and (ii) extending ETL to a language over directed graphs.

As a final remark, we mention that Thomas has studied $\text{MSO}^{\text{chain}}$ extensively on *ranked* trees [29; 30]. From his results it already follows that $\text{MSO}^{\text{chain}}$ is strictly weaker than MSO (Theorem 2). However, although the query he employs (define those trees with an even number of a-labeled leaves) suffices to separate $\text{MSO}^{\text{chain}}$ from MSO, our separation result motivated us to look at quantification over trees to be added to FOREG^* to capture MSO. Indeed, to define the query of Theorem 2, it suffices to define a witness subtree of the input tree that contains only vertices evaluating to true and which contains the root. Define, therefore, TFO^{ver} as the fragment of MSO allowing only quantification restricted to, not necessarily strict, but pure subtrees augmented with the vertical regular path expressions of FOREG. Here, a set of vertices is a pure *pure tree* when its induced subgraph is a tree. Consider, for instance, the tree $a(bb(ccc))$. Then $\{2, 21, 23\}$ (the second b together with the first and the last c) is a pure tree, while $\{\varepsilon, 21, 23\}$ (the previous set with the b replaced by the a) is not. We then obtain that FO extended with these two more intuitive constructs captures MSO (proof omitted).

THEOREM 4. *For every MSO formula $\varphi(\bar{x})$ there exists an equivalent TFO^{ver} formula $\psi(\bar{x})$.*

We can define a variant of ETL only allowing quantification over pure subtrees which would still be equivalent to MSO but whose complexity would be double exponential in the formula size.

7. ACKNOWLEDGEMENTS

We thank Francois Bry, Martin Otto, Jan Van den Bussche, and Wolfgang Thomas for helpful discussions. Helmut Seidl pointed us to filter expressions in XSLT and Michael Benedikt brought [14] to our attention.

8. REFERENCES

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web : From Relations to Semistructured Data and XML.* Morgan Kaufmann, 1999.

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

[3] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. L. Wiener. The lorel query language for semistructured data. *International Journal on Digital Libraries*, 1(1):68–88, 1997.

[4] S. Abiteboul and V. Vianu. Regular path queries with constraints. In *Proc. PODS Conf.*, 1997.

[5] N. Alechina and N. Immerman. Efficient fragment of transitive closure logic. Unpublished, 1999.

[6] H. Andréka, J. van Benthem, and I. Németi. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.

[7] P. Buneman, S. Davidson, G. G. Hillebrand, and D.Suciu. A query language and optimization techniques for unstructured data. In *Proc. SIGMOD Conf.*, 1996.

[8] P. Buneman, W. Fan, and S. Weinstein. Path constraints on semistructured and structured data. In *Proc. PODS Conf.*, 1998.

[9] D. Calvanese, D. G. Giacomo, M. Lenzerini, and M. Y. Vardi. Rewriting of regular expressions and regular path queries. In *Proc. PODS Conf.*, 1999.

[10] J. Clark. XSL transformations version 1.0. http://www.w3.org/TR/WD-xslt, august 1999.

[11] S. Cluet, C. Delobel, J. Siméon, and K. Smaga. Your mediators need data conversion! In *Proc. SIGMOD Conf.*, 1998.

[12] A. Deutsch, M. Fernandez, D. Florescu, A. Levy, and D. Suciu. XML-QL: a query language for XML. In *Proc. WWW8 Conf.*, 1999.

[13] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer, 1995.

[14] M. Fernandez, J. Siméon, and P. Wadler, editors. *XML Query languages: Experiences and Exemplars*, 1999. http://www-db.research.bell-labs.com/user/simeon/xquery.html.

[15] M. F. Fernandez, D. Florescu, J. Kang, A. Y. Levy, and D. Suciu. Catching the boat with strudel: Experiences with a web-site management system. In *Proc. SIGMOD Conf.*, 1998.

[16] G. Gottlob, E. Grädel, and H. Veith. Linear time datalog. Unpublished, 1999.

[17] N. Klarlund. Mona & Fido: The logic-automaton connection in practice. In *Proc. CSL Conf.*, 1998.

[18] S. Maneth and F. Neven. A formalization of tree transformations in XSL. In *Proc. DBPL Conf.*, 1999.

[19] A. O. Mendelzon and P. T. Wood. Finding regular simple paths in graph databases. *SIAM Journal on Computing*, 24(6):1235–1258, 1995.

[20] F. Neven. *Design and Analysis of Query Languages for Structured Documents — A Formal and Logical Approach.* Doctor's thesis, Limburgs Universitair Centrum (LUC), 1999.

[21] F. Neven. Extensions of attribute grammars for structured document queries. In *Proc. DBPL Conf.*, 1999.

[22] F. Neven and T. Schwentick. Query automata. In *Proc. PODS Conf.*, 1999.

[23] F. Neven and J. Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. In *Proc. PODS Conf.*, 1998.

[24] Y. Papakonstantinou and V. Vianu. DTD inference for views of XML data. This proceedings.

[25] A. Potthoff. *Logische Klassifizierung regulärer Baumsprachen.* Doctor's thesis, Institut für Informatik u. Prakt. Math., Universität Kiel, 1994.

[26] J. Robie. The design of XQL. http://www.texcel.no/whitepapers/xql-design.html, 1999.

[27] D. Suciu. Semistructured data and XML. In *Proc. FODO Conf.*, 1998.

[28] W. Thomas. Classifying regular events in symbolic logic. *Journal of Computer and System Sciences*, 25(3):360–376, 1982.

[29] W. Thomas. Logical aspects in the study of tree languages. In *Proc. CAAP Conf.*, 1984.

[30] W. Thomas. On chain logic, path logic, and first-order logic over infinite trees. In *Proc. LICS Conf.*, 1987.

[31] W. Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, chapter 7. Springer, 1997.