Comparing the succinctness of monadic query languages over finite trees

Martin Grohe and Nicole Schweikardt[⋆]

Laboratory for Foundations of Computer Science
University of Edinburgh, Scotland, U.K.
email: {grohe|v1nschwe}@inf.ed.ac.uk

Abstract. We study the *succinctness* of monadic second-order logic and a variety of monadic fixed point logics on trees. All these languages are known to have the same expressive power on trees, but some can express the same queries much more succinctly than others. For example, we show that, under some complexity theoretic assumption, monadic second-order logic is non-elementarily more succinct than monadic least fixed point logic, which in turn is non-elementarily more succinct than monadic datalog.
Succinctness of the languages is closely related to the combined and parameterized complexity of query evaluation for these languages.

Keywords: Finite Model Theory, Monadic Second-Order Logic, Fixed Point Logics, μ-Calculus, Monadic Datalog, Tree-like structures, Succinctness

1. Introduction

A central topic in finite model theory has always been a comparison of the expressive power of different logics on finite relational structures. In particular, the expressive power of fragments of monadic second-order logic and various fixed-point logics has already been investigated in some of the earliest papers in finite model theory [Fag75,CH82]. One of the main motivations for such studies was an interest in the expressive power of query languages for relational databases.

In recent years, the focus in database theory has shifted from relational to semi-structured data and in particular data stored as XML-documents. A lot of current research in the database community is concerned with the design and implementation of XML query languages (see, for example, [FSW00,HP00,GK02] or the monograph [ABS99] for a general introduction into semi-structured data and XML). The languages studied in the present paper may be viewed as node-selecting query languages for XML. They all contain the core of the language XPath, which is an important building block of several major XML-related technologies. Recently, monadic datalog has been proposed as a node-selecting query language with a nice balance between expressive power and very good algorithmic properties [GK02,Koc03].

XML-documents are best modelled by trees, or more precisely, finite labelled ordered unranked trees. It turns out that when studying node-selecting query languages for

[⋆] supported by a fellowship within the Postdoc-Programme of the German Academic Exchange Service (DAAD)

XML-documents, expressive power is not the central issue. Quite to the contrary: Neven and Schwentick [NS02] proposed to take the expressive power of monadic second-order logic (MSO) as a benchmark for node-selecting XML-query languages and, in some sense, suggested that such languages should at least have the expressive power of MSO. However, even languages with the same expressive power may have vastly different complexities. For example, monadic datalog and MSO have the same expressive power over trees [GK02]. However, monadic datalog queries can be evaluated in time linear both in the size of the datalog program and the size of the input tree [GK02], and thus the combined complexity of monadic datalog is in polynomial time, whereas the evaluation of MSO queries is PSPACE complete. The difference becomes even more obvious if we look at parameterized complexity: Unless PTIME $\neq$ NP, there is no algorithm evaluating a monadic second-order query in time $f(\text{size of query})p(\text{size of tree})$ for any elementary function f and polynomial p [FG03]. Similar statements hold for the complexity of the satisfiability problem for monadic datalog and MSO over trees. The reason for this different behaviour is that even though the languages have the same expressive power on trees, in MSO we can express queries much more *succinctly*. Indeed, there is no elementary translation from a given MSO-formula into an equivalent monadic datalog program. We also say that MSO is *non-elementarily more succinct* than monadic datalog. Just to illustrate the connection between succinctness and complexity, let us point out that if there was an elementary translation from MSO to monadic datalog, then there would be an algorithm evaluating a monadic second-order query in time $f(\text{size of query})p(\text{size of tree})$ for an elementary function f and a polynomial p.

In this paper, we study the succinctness (in the sense just described) of a variety of fixed point logics on finite trees. Our main results are the following:

1. MSO *is non-elementarily more succinct than monadic least fixed point logic* MLFP *(see Theorem 2). Unfortunately, we are only able to prove this result under the odd, but plausible complexity theoretic assumption that for some* $i \geqslant 1$, NP *is not contained in* $\text{DTIME}(n^{\log^{(i)}(n)})$, *where* $\log^{(i)}$ *denotes the* i *times iterated logarithm.*
2. MLFP *is non-elementarily more succinct than its* 2*-variable fragment* MLFP^2 *(see Corollary 3).*
3. MLFP^2 *is exponentially more succinct than the full modal* μ*-calculus, that is, the modal* μ*-calculus with future and past modalities (see Theorem 3, Example 2, and Theorem 4).*
4. *The full modal* μ*-calculus is at most exponentially more succinct than stratified monadic datalog, and conversely, stratified monadic datalog is at most exponentially more succinct than the full modal* μ*-calculus (see Theorem 7 and 8). Furthermore, stratified monadic datalog is at most exponentially more succinct than monadic datalog (see Theorem 6).*
   The exact relationship between these three languages remains open.

Of course we are a not the first to study the succinctness of logics with the same expressive power. Most known results are about modal and temporal logics. The motivation for these results has not come from database theory, but from automated verification and model-checking. The setting, however, is very similar. For example, Kamp's well know theorem states that first-order logic and linear time temporal logic have the same expressive power on strings [Kam68], but there is no elementary translation from first-order logic to linear time temporal logic on strings. Even closer to our results, monadic second-order logic and the modal μ-calculus have the same expressive power on (ordered) trees, but again is well-known that there is no elementary translation from the former to the latter. Both of these results can be proved by simple automata theoretic arguments. More refined results are known for various temporal logics [Wil99,AI00,AI01,EVW02]. By and large, however, succinctness has received surprisingly little attention in the finite model theory community. Apart from automata theoretic arguments, almost no good techniques for proving lower bounds on formula sizes are known. A notable exception are Adler and Immerman's [AI01] nice games for proving such lower bounds. Unfortunately, we found that these games (adapted to fixed point logic) were of little use in our context. So we mainly rely on automata theoretic arguments. An exception is the, complexity theoretically conditioned, result that MSO is non-elementarily more succinct than MLFP. To prove this result, we are building on a technique introduced in [FG03].

The paper is organised as follows: In Section 2 we fix the basic notations used throughout the paper. Section 3 concentrates on the translation from MSO to MLFP. In Section 4 we present our results concerning the two-variable fragment of MLFP and the full modal μ-calculus. In Section 5 we concentrate on monadic datalog, stratified monadic datalog, and their relations to finite automata and to MLFP. Finally, Section 6 concludes the paper by pointing out several open questions.

Due to space limitations, we had to defer detailed proofs of our results to the full version of this paper [GS03].

2. Preliminaries

2.1. Basic Notations. Given a set Σ we write Σ^* to denote the set of all finite strings over Σ, and we use ε to denote the empty string. We use $\mathbb{N}$ to denote the set $\{0, 1, 2, ..\}$ of natural numbers. We use lg to denote the logarithm with respect to base 2. With a function f that maps natural numbers to real numbers we associate the corresponding function from $\mathbb{N}$ to $\mathbb{N}$ defined by $n \mapsto \lceil f(n) \rceil$. For simplicity we often simply write $f(n)$ instead of $\lceil f(n) \rceil$.

The function *Tower* $: \mathbb{N} \to \mathbb{N}$ is inductively defined via $\mathit{Tower}(0) := 1$ and $\mathit{Tower}(h{+}1) = 2^{\mathit{Tower}(h)}$, for all $h \in \mathbb{N}$. I.e., $\mathit{Tower}(h)$ is a tower of 2s of height h.

We say that a function $f : \mathbb{N} \to \mathbb{N}$ has bound $f(m) \leqslant \mathit{Tower}\big(o(h(m))\big)$, for some function $h : \mathbb{N} \to \mathbb{N}$, if there is a function $g \in o(h)$ and a $m_0 \in \mathbb{N}$ such that for all $m \geqslant m_0$ we have $f(m) \leqslant \mathit{Tower}\big(g(m)\big)$. Note that, in particular, every *elementary* function f has bound $f(m) \leqslant \mathit{Tower}\big(o(m)\big)$. Indeed, for every elementary function f there is a $h \in \mathbb{N}$ such that, for all $n \in \mathbb{N}$, $f(n)$ is less than or equal to the tower of 2s of height h with an n on top.

2.2. Structures. A *signature* τ is a finite set of relation symbols and constant symbols. Each relation symbol $R \in \tau$ has a fixed arity $ar(R)$. A τ-structure $\mathcal{A}$ consists of a set $\mathcal{U}^{\mathcal{A}}$ called the *universe* of $\mathcal{A}$, an interpretation $c^{\mathcal{A}} \in \mathcal{U}^{\mathcal{A}}$ of each constant symbol $c \in \tau$, and an interpretation $R^{\mathcal{A}} \subseteq (\mathcal{U}^{\mathcal{A}})^{ar(R)}$ of each relation symbol $R \in \tau$. *All structures considered in this paper are assumed to have a finite universe.*

The main focus of this paper lies on the class *Trees* of finite binary trees. Precisely, finite binary trees are particular structures over the signature

$$\tau_{\mathit{Trees}} \ := \ \{\, \mathit{Root},\ 1^{st}\mathit{Child},\ 2^{nd}\mathit{Child},\ \mathit{Has\text{-}No\text{-}}1^{st}\mathit{Child},\ \mathit{Has\text{-}No\text{-}}2^{nd}\mathit{Child}\,\},$$

where *Root*, *Has-No-*1^{st}*Child*, *Has-No-*2^{nd}*Child* are unary relation symbols and 1^{st}*Child*, 2^{nd}*Child* are binary relation symbols. We define *Trees* to be the set of all τ_{Trees}-structures T that satisfy the following conditions:

1. $\mathcal{U}^T \subset \{1,2\}^*$ and for every string $si \in \mathcal{U}^T$ with $i \in \{1,2\}$ we also have $s \in \mathcal{U}^T$.
2. Root^T consists of the empty string ε.
3. $1^{st}\mathit{Child}^T$ consists of the pairs $\big(s, s1\big)$, for all $s1 \in \mathcal{U}^T$.
4. $2^{nd}\mathit{Child}^T$ consists of the pairs $\big(s, s2\big)$, for all $s2 \in \mathcal{U}^T$.
5. *Has-No-*$1^{st}\mathit{Child}^T$ consists of all strings $s \in \mathcal{U}^T$ with $s1 \notin \mathcal{U}^T$.
6. *Has-No-*$2^{nd}\mathit{Child}^T$ consists of all strings $s \in \mathcal{U}^T$ with $s2 \notin \mathcal{U}^T$.

For $T \in$ *Trees* and $t \in \mathcal{U}^T$ we write T_t to denote the subtree of T with root t.

A *schema* σ is a set of unary relation symbols each of which is distinct from *Has-No-*1^{st}*Child*, *Has-No-*2^{nd}*Child*, *Root*. A σ-labelled tree is a $(\tau_{\mathit{Trees}} \cup \sigma)$-structure consisting of some $T \in$ *Trees* and additional interpretations $P^T \subseteq \mathcal{U}^T$ for all symbols $P \in \sigma$. We sometimes write $\mathit{label}(t)$ to denote the set $\{P \in \sigma \ : \ t \in P^T\}$ of labels at vertex t in T.

We identify a string $w = w_0 \cdots w_{n-1}$ of length $|w| = n \geqslant 1$ over an alphabet Σ with a σ-labelled tree T^w in the following way: We choose σ to consist of a unary relation symbol P_a for each letter $a \in \Sigma$, we choose T^w to be the (unique) element in *Trees* with universe $\mathcal{U}^{T^w} = \{\varepsilon, 1, 11, .., 1^{n-1}\}$, and we choose $P_a^{T^w} := \{1^i \ : \ w_i = a\}$, for each $a \in \Sigma$. This corresponds to the conventional representation of strings by structures in the sense that $\langle \mathcal{U}^{T^w}, 1^{st}\mathit{Child}, (P_a^{T^w})_{a\in\Sigma}\rangle$ is isomorphic to the structure $\langle \{0, .., n{-}1\}, \mathit{Succ}, (P_a^w)_{a\in\Sigma}\rangle$ where *Succ* denotes the binary successor relation on $\{0, .., n{-}1\}$ and P_a^w consists of all positions of w that carry the letter a. When reasoning about strings in the context of first-order logic, we sometimes also need the linear ordering $<$ on $\{0, .., n{-}1\}$ (respectively, the transitive closure of the relation 1^{st}*Child*). In these cases we explicitly write FO$(<)$ rather than FO to indicate that the linear ordering is necessary.

XML-documents are usually modelled as ordered unranked trees and not as binary trees. Here *ordered* refers to the fact that the order of the children of a vertex is given. However, a standard representation of ordered unranked trees as relational structures uses binary relations 1^{st}*Child*, *Next-Sibling* and unary relations *Root*, *Leaf*, *Last-Sibling* (for details, see [GK02]) and thus essentially represents ordered unranked trees as binary trees. Therefore, all our results also apply to ordered unranked trees.

2.3. Logics and Queries. We assume that the reader is familiar with *first-order logic*, for short: FO, and with *monadic second-order logic*, for short: MSO (cf., e.g., the textbooks [EF99,Imm99]). We use FO(τ) and MSO(τ), respectively, to denote the class of all first-order formulas and monadic second-order formulas, respectively, of signature τ. We write $\varphi(x_1, .., x_k, X_1, .., X_\ell)$ to indicate that the free first-order variables of the formula φ are $x_1, .., x_k$ and the free set variables are $X_1, .., X_\ell$. Sometimes we use $\overline{x}$ and $\overline{X}$ as abbreviations for sequences $x_1, .., x_k$ and $X_1, .., X_\ell$ of variables.

A formula $\varphi(x)$ of signature τ defines the *unary query* which associates with every τ-structure $\mathcal{A}$ the set of elements $a \in \mathcal{U}^{\mathcal{A}}$ such that $\mathcal{A} \models \varphi(a)$, i.e., $\mathcal{A}$ satisfies φ when interpreting the free occurrences of the variable x by the element a. A *sentence* φ of signature τ (i.e., a formula that has no free variables) defines the *Boolean query* that associates the answer "yes" with all τ-structures that satisfy φ and the answer "no" with all other τ-structures.

Apart from FO and MSO we will also consider *monadic least fixed point logic* MLFP which is the extension of first-order logic by unary least fixed point operators. We refer the reader to [EF99] for the definition of MLFP (denoted by FO(M-LFP) there).

2.4. Formula size and succinctness. In a natural way, we view formulas as finite trees, where leaves correspond to the atoms of the formulas and inner vertices correspond to Boolean connectives, quantifiers, and fixed-point operators. We define the *size* $||\varphi||$ of a formula φ to be the number of vertices of the tree that corresponds to φ.

Note that this measure of formula size is a *uniform cost measure* in the sense that it accounts just 1 cost unit for each variable and relation symbol appearing in a formula, no matter what its index is. An alternative is to define the size of a formula as the length of a binary encoding of the formula. Such a *logarithmic cost measure* is, for example, used in [FG03]. Switching between a uniform and a logarithmic measure usually involves a logarithmic factor.

Definition 1 (Succinctness). *Let L_1 and L_2 be logics, let F be a class of functions from $\mathbb{N}$ to $\mathbb{N}$, and let $\mathcal{C}$ be a class of structures.*
We say that L_1 is F-succinct in L_2 on $\mathcal{C}$ *if there is a function $f \in F$ such that for every formula $\varphi_1 \in L_1$ there is a formula $\varphi_2 \in L_2$ of size $||\varphi_2|| \leqslant f(||\varphi_1||)$ which is equivalent to φ_1 on all structures in $\mathcal{C}$.* $\square$

Intuitively, a logic L_1 being F-succinct in a logic L_2 means that F gives an upper bound for the size of L_1-formulas needed to express *all* of L_2. This definition may seem slightly at odds with the common use of the term "succinctness" in statements such as "L_2 is exponentially *more succinct* than L_1" meaning that there is *some* L_2-formula that is not equivalent to any L_1-formula of subexponential size. In our terminology, we would rephrase this last statement as "L_1 is *not* $2^{o(n)}$-succinct in L_2" (here we interpret subexponential as $2^{o(n)}$, but of course this is not the issue). The reason for defining F-succinctness the way we did is that it makes the formal statements of our results much more convenient. We will continue to use statements such as "L_2 is exponentially more succinct than L_1" in informal discussions.

Example 1. MLFP is $\mathcal{O}(m)$-succinct in MSO on the class of all finite structures, because every formula $[\mathrm{LFP}_{x,X}\varphi(x,X,\overline{y},\overline{Y})](z)$ is equivalent to $\forall X\ \big(Xz \lor \exists x\, \neg Xx \land \varphi(x,X,\overline{y},\overline{Y})\big)$. $\square$

3. From MSO to MLFP

By the standard translation from MSO-logic to tree automata (cf., e.g., [Tho96]) one knows that every MSO-sentence Φ can be translated into a nondeterministic tree automaton with $\mathit{Tower}\big(\mathcal{O}(||\Phi||)\big)$ states that accepts exactly those labelled trees that satisfy Φ. This leads to

Theorem 1 (Folklore). MSO*-sentences are Tower*$\big(\mathcal{O}(m)\big)$*-succinct in* MLFP *on the class of all labelled trees.* □

To show that we cannot do essentially better, i.e., that there is no translation from MSO to MLFP of size *Tower*$\big(o(m)\big)$ we need a complexity theoretic assumption that, however, does not seem to be too far-fetched. Let SAT denote the NP-complete satisfiability problem for propositional formulas in conjunctive normal form. Until now, all known deterministic algorithms that solve SAT have worst-case complexity $2^{\Omega(n)}$ (cf., [DGH$^+$02]). Although not answering the P vs. NP question, the exposition of a deterministic algorithm for SAT with worst-case complexity $\leqslant n^{\lg n}$ would be a surprising and unexpected breakthrough in the SAT-solving community.

In the following, we write $\lg^{(i)}$ to denote the i times iterated logarithm, inductively defined by $\lg^{(1)}(n) := \lg(n)$ and $\lg^{(i+1)}(n) := \lg(\lg^{(i)}(n))$. Moreover, we we write $\lg^*$ to denote the "inverse" of the *Tower* function, that is, the (unique) integer valued function with *Tower*$(\lg^*(n){-}1) < n \leqslant$ *Tower*$(\lg^*(n))$.

Theorem 2. *Unless* SAT *is solvable by a deterministic algorithm that has, for every $i \in \mathbb{N}$, time bound $||\gamma||^{\lg^{(i)}(n)}$ (where γ is the input formula and n the number of propositional variables occurring in γ),* MSO *is not Tower*$\big(o(m)\big)$*-succinct in* MLFP *on the class of all finite strings.* □

The overall proof idea is to assume that the function f specifies the size of the translation from MSO to MLFP and to exhibit a SAT-solving algorithm which

– constructs a string w that represents the SAT-instance γ,
– constructs an MSO-formula $\Phi(z)$ of extremely small size that, when evaluated in w, specifies a canonical satisfying assignment for γ (if γ is satisfiable at all),
– tests, for all MLFP-formulas $\Psi(z)$ of size $\leqslant f(||\Phi||)$, whether Ψ specifies a satisfying assignment for γ.

Before presenting the proof in detail we provide the necessary notations and lemmas: It is straightforward to see

Lemma 1. *There is an algorithm that, given an* MLFP*-formula $\Psi(z)$, a string w, and a position p in w, decides in time $|w|^{\mathcal{O}(||\Psi||)}$ whether $w \models \Psi(p)$.* □

Let us now concentrate on the construction of a string w that represents a SAT-instance γ and of an MSO-formula $\Phi(z)$ that specifies a canonical satisfying assignment of γ (provided that γ is satisfiable at all). Since we want Φ to be extremely short, we cannot choose w to be the straightforward string-representation of γ. Instead, we use the following, more complicated, representation of [FG03]:

For all $h \geqslant 1$ let $\Sigma_h := \big\{0, 1, \texttt{<1>}, \texttt{</1>}, .., \texttt{<h>}, \texttt{</h>}\big\}$. The "tags" `<i>` and `</i>` represent single letters of the alphabet and are just chosen to improve readability. For every $n \geqslant 1$ let $L(n)$ be the length of the binary representation of the number $n{-}1$, i.e., $L(0) = 0$, $L(1) = 1$, and $L(n) = \lfloor\lg(n{-}1)\rfloor + 1$, for all $n \geqslant 2$. By $\mathrm{bit}(i, n)$ we denote the i-th bit of the binary representation of n, i.e., $\mathrm{bit}(i, n)$ is 1 if $\lfloor \frac{n}{2^i} \rfloor$ is odd, and $\mathrm{bit}(i, n)$ is 0 otherwise.
We encode every number $n \in \mathbb{N}$ by a string $\mu_h(n)$ over the alphabet Σ_h, where $\mu_h(n)$ is inductively defined as follows: $\mu_1(0) := \texttt{<1></1>}$, and

$$\mu_1(n) := \texttt{<1>}\,\mathrm{bit}(0, n{-}1)\,\mathrm{bit}(1, n{-}1)\cdots\mathrm{bit}(L(n){-}1, n{-}1)\,\texttt{</1>}\,,$$

for $n \geqslant 1$. For $h \geqslant 2$ we let $\mu_h(0) :=$ `<h></h>` and

$\mu_h(n) :=$ `<h></h>`$\mu_{h-1}(0)\,\text{bit}(0, n{-}1)\ \cdots$ `</h>`$\mu_{h-1}(L(n){-}1)\,\text{bit}(L(n){-}1, n{-}1)$`</h>`,

for $n \geqslant 1$. Here empty spaces and line breaks are just used to improve readability.
To encode a CNF-formula γ by a string we use an alphabet Σ'_h that extends Σ_h by the symbols $+, -, \star$ and a number of additional tags. Let $i \in \mathbb{N}$ and let X_i be a propositional variable. The literal X_i is encoded by the string

$$\mu_h(X_i) \ := \ \texttt{<lit>}\mu_h(i) + \texttt{</lit>},$$

and the literal $\neg X_i$ is encoded by $\mu_h(\neg X_i) \ := \ \texttt{<lit>}\mu_h(i) - \texttt{</lit>}$.
A clause $\delta := \lambda_1 \vee \cdots \vee \lambda_r$ of literals is encoded by

$$\mu_h(\delta) \ := \ \texttt{<clause>}\mu_h(\lambda_1)\cdots\mu_h(\lambda_r)\texttt{</clause>}.$$

A CNF-formula $\gamma := \delta_1 \wedge \cdots \wedge \delta_m$ is encoded by the string

$$\mu_h(\gamma) \ := \ \texttt{<cnf>}\mu_h(\delta_1)\cdots\mu_h(\delta_m)\texttt{</cnf>}.$$

We write CNF(n) to denote the class of all CNF-formulas the propositional variables of which are among $X_0, .., X_{n-1}$. To provide the "infrastructure" for specifying a truth assignment, we use the string

$$\mu_h(X_0, .., X_{n-1}) := \texttt{<ass><val>}\mu_h(0) \star \texttt{</val>}\cdots \\ \texttt{<val>}\mu_h(n{-}1) \star \texttt{</val></ass>}.$$

Remark 1. There is a 1–1-correspondence between assignments $\alpha : \{X_0, .., X_{n-1}\} \to \{\mathit{true}, \mathit{false}\}$, on the one hand, and sets P of positions of $\mu_h(X_0, .., X_{n-1})$ that carry the letter $\star$, on the other hand: Such a set P specifies the assignment α^P that, for each $i < n$, maps the variable X_i to the value *true* iff the $\star$-position directly after the substring $\mu_h(i)$ in $\mu_h(X_0, .., X_{n-1})$ belongs to P. Conversely, a given assignment α specifies the set P^α consisting of exactly those $\star$-positions of $\mu_h(X_0, .., X_{n-1})$ that occur directly after a substring $\mu_h(i)$ where $\alpha(X_i) = \mathit{true}$. □

Finally, we encode a formula $\gamma \in \text{CNF}(n)$ by the string

$$\mu_h(\gamma, \star) \ := \ \mu_h(\gamma)\, \mu_h(X_0, .., X_{n-1}).$$

$\mu_h(\gamma, \star)$ is the string w that we will furtheron use as the representative of a SAT-instance γ. We use the following result of [FG03]:

Lemma 2.

(a) There is an algorithm that, given $h \in \mathbb{N}$ and $\gamma \in CNF(n)$, computes (a binary representation of) the string $\mu_h(\gamma, \star)$ in time $\mathcal{O}\big(h \cdot (\lg h) \cdot (\lg n)^2 \cdot (||\gamma|| + n)\big)$ (cf., [FG03, Lemma 9]).
The string $\mu_h(\gamma, \star)$ has length $|\mu_h(\gamma, \star)| = \mathcal{O}\big(h \cdot (\lg n)^2 \cdot (||\gamma|| + n)\big)$.

(b) There is an algorithm that, given $h \in \mathbb{N}$, computes (the binary representation of) a FO$(<)$*-formula $\varphi_h(Z)$ in time $\mathcal{O}\big(h \cdot \lg h\big)$, such that for all $n \leqslant Tower(h)$, for all $\gamma \in CNF(n)$, and for all sets P of $\star$-positions in the string $\mu_h(\gamma, \star)$ we have*

$$\mu_h(\gamma, \star) \models \varphi_h(P) \quad \text{iff} \quad \alpha^P \text{ is a satisfying assignment for } \gamma$$

(cf., [FG03, Lemma 10]). The formula $\varphi_h(Z)$ has size[1] $||\varphi_h(Z)|| = \mathcal{O}(h)$. □

[1] In [FG03], an additional factor $\lg h$ occurs because there a logarithmic cost measure is used for the formula size, whereas here we use a uniform measure (cf., Section 2.4).

Given a CNF(n)-formula γ and its representative $\mu_h(\gamma,\star)$, we now specify a *canonical* satisfying assignment of γ, provided that γ is satisfiable at all. As observed in Remark 1, every assignment $\alpha : \{X_0,..,X_{n-1}\} \to \{\mathit{true},\mathit{false}\}$ corresponds to a set P^α of positions in $\mu_h(\gamma,\star)$ that carry the letter $\star$. P^α, again, can be identified with the 0-1-string of length $|\mu_h(\gamma,\star)|$ that carries the letter 1 exactly at those positions that belong to P^α. Now, the lexicographic ordering of these strings gives us a linear ordering on the set of all assignments $\alpha : \{X_0,..,X_{n-1}\} \to \{\mathit{true},\mathit{false}\}$. As the canonical satisfying assignment of γ we choose the lexicographically smallest satisfying assignment.

Lemma 3. *There is an algorithm that, given $h \in \mathbb{N}$, computes (the binary representation of) an* MSO*-formula $\Phi_h(z)$ in time $\mathcal{O}(h \cdot \lg h)$, such that for all $n \leqslant \mathit{Tower}(h)$, for all $\gamma \in \mathit{CNF}(n)$, and for all positions p of $\mu_h(\gamma,\star)$ that carry the letter $\star$, we have*

$\mu_h(\gamma,\star) \models \Phi_h(p)$ *iff* *in the lexicographically smallest satisfying assignment for γ, the propositional variable corresponding to position p is assigned the value true.*

The formula $\Phi_h(z)$ has size $||\Phi_h|| = \mathcal{O}(h)$. □

Finally, we are ready for the Proof of Theorem 2:

Proof of Theorem 2.
Let $f : \mathbb{N} \to \mathbb{N}$ be a function such that there is, for every MSO-formula $\Phi(z)$, a MLFP-formula $\Psi(z)$ of size $||\Psi|| \leqslant f(||\Phi||)$ which defines the same query as Φ on the class of all finite strings (recall that such an f does indeed exist, because MSO and MLFP have the same expressive power over the class of finite strings).
Consider the algorithm displayed in Figure 1, which decides if the input formula γ is satisfiable.
The correctness of this algorithm directly follows from Lemma 3 and from the fact that at least one of the formulas $\Psi(z)$ of size $\leqslant f(||\Phi_h||)$ defines the same query as $\Phi_h(z)$.

It remains to determine the worst-case running time of the algorithm. Let γ be an input CNF-formula for the algorithm, let n be the number of propositional variables of γ, and let $h := \lg^*(n)$.
The steps 1–4 of the algorithm will be performed within a number of steps polynomial in $||\gamma||$, and the MSO-formula $\Phi_h(z)$ produced in step 4 will have size $||\Phi_h|| \leqslant c \cdot h$, for a suitable constant $c \in \mathbb{N}$ (cf., Lemma 2 (a) and Lemma 3).
The loop in step 5 will be performed for $2^{c_1 \cdot f(||\Phi_h||) \cdot \lg(f(||\Phi_h||))}$ times, for a suitable constant $c_1 \in \mathbb{N}$. To see this, note that formulas of length $\leqslant f(||\Phi_h||)$ use at most $f(||\Phi_h||)$ different first-order variables and at most $f(||\Phi_h||)$ different set variables. I.e., these formulas can be viewed as strings of length $f(||\Phi_h||)$ over an alphabet of size $c_2 + 2 \cdot f(||\Phi_h||)$, for a suitable constant $c_2 \in \mathbb{N}$. Therefore, the number of such formulas is $\leqslant (c_2 + 2 \cdot f(||\Phi_h||))^{f(||\Phi_h||)} \leqslant 2^{c_1 \cdot f(||\Phi_h||) \cdot \lg(f(||\Phi_h||))}$.
Each performance of the loop in step 5 will take a number of steps polynomial in

$$|\mu_h(\gamma,\star)|^{\mathcal{O}(f(||\Phi_h||))} \leqslant (c_3 \cdot h \cdot (\lg n)^2 \cdot ||\gamma||)^{c_4 \cdot f(c \cdot h)},$$

for suitable constants $c_3, c_4 \in \mathbb{N}$ (cf., Lemma 1 and Lemma 2 (a)). Altogether, for suitable constants $c, d \in \mathbb{N}$, the algorithm will perform the steps 1–6 within

$$||\gamma||^{d \cdot f(c \cdot h) \cdot \lg(f(c \cdot h))}$$

```
Input: a SAT-instance $\gamma$ in CNF

1. Count the number $n$ of propositional variables occurring in $\gamma$, and
   modify $\gamma$ in such a way that only the propositional variables $X_0, .., X_{n-1}$ occur in it.
2. Compute $h := \lg^*(n)$, i.e., choose $h \in \mathbb{N}$ such that $Tower(h-1) < n \leqslant Tower(h)$.
3. Construct the string $\mu_h(\gamma, \star)$ that represents $\gamma$ (see Lemma 2 (a)).
4. Construct an MSO-formula $\Phi_h(z)$ that has the following property:
   Whenever $p$ is a position in $\mu_h(\gamma, \star)$ that carries the letter $\star$, we have

      $\mu_h(\gamma, \star) \models \Phi_h(p)$  iff  in the lexicographically smallest satisfying assignment
                                  for $\gamma$, the propositional variable corresponding to posi-
                                  tion $p$ is assigned the value true

   (cf., Lemma 3).
5. For all MLFP-formulas $\Psi(z)$ of size $||\Psi|| \leqslant f(||\Phi_h||)$ do:
   (a) Initialise the assignment $\alpha := \emptyset$.
   (b) For all positions $p$ in $\mu_h(\gamma, \star)$ that carry the letter $\star$ do
          check whether $\mu_h(\gamma, \star) \models \Psi(p)$;
          if so, then insert the propositional variable corresponding to $p$ into $\alpha$.
   (c) Check whether $\alpha$ is a satisfying assignment for $\gamma$;
       if so, then STOP with output "$\gamma$ is satisfiable via assignment $\alpha$".
6. STOP with output "$\gamma$ is not satisfiable".
```

Fig. 1. A SAT-solving algorithm.

steps.

Now let us suppose that f has bound $f(m) \leqslant Tower\big(o(m)\big)$. From Lemma 4 below we then obtain that our SAT-solving algorithm has, for every $i \in \mathbb{N}$, time bound $||\gamma||^{\lg^{(i)}(n)}$. This finally completes the proof of Theorem 2. ■

Lemma 4. *Let $f : \mathbb{N} \to \mathbb{N}$ be a function with bound $f(m) \leqslant Tower\big(o(m)\big)$, and let $c, d \in \mathbb{N}$. For every $i \in \mathbb{N}$ there is an $n_0 \in \mathbb{N}$ such that for all $n \geqslant n_0$ we have*

$$d \cdot f\big(c \cdot \lg^*(n)\big) \cdot \lg\big(f(c \cdot \lg^*(n))\big) \;\leqslant\; \lg^{(i)}(n)\,. \qquad \square$$

4. The Two-Variable Fragment of MLFP and the Full Modal μ-Calculus

Defining the 2-variable fixed-point logics requires some care: MLFP^2 is the fragment of MLFP consisting of all formulas with just 2 individual variables and no parameters in fixed point operators, i.e., for all subformulas of the form $[\mathrm{LFP}_{x,X}\varphi](y)$, x is the only free first-order variable of φ. This is the monadic fragment of the standard 2-variable least fixed-point logic logic (cf. [GO99]). Without the restriction on free variables in fixed-point operators, we obtain full MLFP even with just two individual variables (we prove this in the full version of this paper [GS03]).

We first note that MLFP^2, and actually FO^2, the two variable fragment of first-order logic, is doubly exponentially more succinct than nondeterministic automata on the class of all finite strings:

Example 2. Let $\sigma := \{L, R, P_1, .., P_n\}$ and

$$\varphi_n := \forall x \Big(Lx \to \exists y \big(Ry \wedge \bigwedge_{i=1}^{n} (P_i x \leftrightarrow P_i y) \big) \Big).$$

We claim that every nondeterministic finite automaton accepting precisely those strings over alphabet 2^σ that satisfy φ has at least 2^{2^n} states. To see this, for every $S \subseteq 2^{\{1,..,n\}}$, we define strings $X_n(S)$ and $Y_n(S)$ such that

– $L^{X_n(S)} = \mathcal{U}^{X_n(S)}$ and $R^{Y_n(S)} = \mathcal{U}^{Y_n(S)}$
– For all $s \in S$ there exists an $x \in \mathcal{U}^{X_n(S)}$ and an $y \in \mathcal{U}^{Y_n(S)}$ such that $s = \{i \mid x \in P_i^{X_n(S)}\} = \{i \mid y \in P_i^{Y_n(S)}\}$.

Let $W_n(S,T) := X_n(S)\,Y_n(T)$ be the concatenation of $X_n(S)$ and $Y_n(T)$. Then $W_n(S,T) \models \varphi \iff S \subseteq T$. Clearly, a nondeterministic finite automaton accepting precisely those strings $W_n(S,T)$ with $S \subseteq T$ needs at least 2^{2^n} states. ☐

Let us return to binary trees now. Following Vardi [Var98], we define the *full modal μ-calculus* FL_μ on binary trees as follows: For each schema σ, an FL_μ-formula of schema σ is either:

– *true*, *false*, P, or $\neg P$, where $P \in \sigma \cup \{$*Root*, *Has-No-*1st*Child*, *Has-No-*2nd*Child*$\}$;
– $\Phi_1 \wedge \Phi_2$ or $\Phi_1 \vee \Phi_2$, where Φ_1 and Φ_2 are FL_μ-formulas of schema σ;
– X, where X is a propositional variable;
– $\langle R \rangle\, \Phi$ or $[R]\, \Phi$, where $R \in \{1^{st}\mathit{Child},\ 2^{nd}\mathit{Child},\ 1^{st}\mathit{Child}^{-1},\ 2^{nd}\mathit{Child}^{-1}\}$ and Φ is an FL_μ-formula of schema σ;
– $\mu X.\Phi$ or $\nu X.\Phi$, where X is a propositional variable and Φ is an FL_μ-formula of schema σ.

The semantics of FL_μ is defined in the usual way interpreting the binary relations over trees. The following is starightforward:

Proposition 1. FL_μ *is $\mathcal{O}(m)$-succinct in* MLFP^2.

Our next result is that there also is a reverse translation from MLFP^2 to FL_μ which only incurs an exponential blow-up in size:

Theorem 3. MLFP^2 *is $2^{poly(m)}$-succinct in* FL_μ *on the class of labelled trees.* ☐

Theorem 4 (Vardi [Var98]). *For every formula Φ of the full modal μ-calculus* FL_μ *there is a nondeterministic tree automaton of size $2^{poly(||\Phi||)}$ that accepts exactly those labelled trees in which Φ holds at the root.* ☐

As a matter of fact, Vardi [Var98] proved a stronger version of this theorem for infinite trees and parity tree automata. But on finite trees, a parity acceptance condition can always be replaced by a normal acceptance for finite tree automata.
The Theorems 3 and 4 directly imply

Corollary 1. *For every* MLFP^2*-formula $\varphi(x)$ there is a nondeterministic tree automaton of size $2^{2^{poly(||\varphi||)}}$ that accepts exactly those labelled trees in which φ holds at the root.* ☐

5. Monadic Datalog and Stratified Monadic Datalog

We assume that the reader is familiar with *datalog*, which may be viewed as logic programming without function symbols (cf., e.g., the textbook [AHV95]). A datalog program is *monadic* if all its IDB-predicates (i.e., its intensional predicates that appear in the head of some rule of the program) are unary. In this paper we restrict attention to monadic datalog programs that are interpreted over labelled trees. A monadic datalog program of schema σ may use as EDB-predicates (i.e., extensional predicates which are determined by the structure the program is interpreted over) the predicates in τ_{Trees}, the predicates in σ, and a predicate $\neg P$ for every $P \in \sigma$ which is interpreted as the complement of P. We use IDB($\mathcal{P}$) to denote the set of IDB-predicates of $\mathcal{P}$, and we write MonDatalog to denote the class of all monadic datalog programs.

More formally, a monadic datalog program $\mathcal{P}$ of schema σ is a finite set of rules of the form $X(x) \leftarrow \gamma(x, \overline{y})$, where γ is a conjunction of atomic formulas over the signature $\tau_{\mathit{Trees}} \cup \sigma \cup \{\neg P : P \in \sigma\} \cup \mathrm{IDB}(\mathcal{P})$. Every program has a distinguished *goal* IDB-predicate that determines the query defined by the program.

We define the *size* $||\mathcal{P}||$ of $\mathcal{P}$ in the same way as we defined the size of formulas.

In [GK02] it was shown that MonDatalog can define the same unary queries on the class of labelled trees as monadic second-order logic. In the remainder of this section we will compare the succinctness of MonDatalog, S-MonDatalog, FL$_\mu$, MLFP, and a particular kind of tree automaton.

5.1. From MonDatalog to Finite Automata. Several mechanisms have been proposed in the literature for specifying *unary* queries by finite automata operating on labelled trees (cf., [NS02]). One such mechanism, introduced in [Nev99] and further investigated in [FGK03,Koc03], is the *selecting tree automaton*:

Definition 2 (STA). *Let σ be a schema. A selecting σ-tree automaton (σ-STA, for short) is a tuple $\mathfrak{A} = (Q, 2^\sigma, F, \delta, S)$, where $S \subseteq Q$ is the set of* selecting states *and $(Q, 2^\sigma, F, \delta)$ is a conventional nondeterministic bottom-up tree automaton (cf., e.g., [Tho96]) with finite state space Q, input alphabet 2^σ, accepting states $F \subseteq Q$, and transition function*

$$\delta \ : \ 2^\sigma \ \cup \ \big(\{1\} \times Q \times 2^\sigma\big) \ \cup \ \big(\{2\} \times Q \times 2^\sigma\big) \ \cup \ \big(Q \times Q \times 2^\sigma\big) \ \to \ 2^Q .$$

A run *of $\mathfrak{A}$ on a σ-labelled tree T is a mapping $\rho : \mathcal{U}^T \to Q$ that has the following property, for all vertices $t, t_1, t_2 \in \mathcal{U}^T$: if t has no children then $\rho(t) = \delta(\mathit{label}(t))$; if $1^{st}\mathit{Child}(t, t_1) \wedge \mathit{Has\text{-}No\text{-}}2^{nd}\mathit{Child}(t)$ then $\rho(t) \in \delta\big(1, \rho(t_1), \mathit{label}(t)\big)$; if $2^{nd}\mathit{Child}(t, t_2) \wedge \mathit{Has\text{-}No\text{-}}1^{st}\mathit{Child}(t)$ then $\rho(t) \in \delta\big(2, \rho(t_2), \mathit{label}(t)\big)$; if $1^{st}\mathit{Child}(t, t_1) \wedge 2^{nd}\mathit{Child}(t, t_2)$ then $\rho(t) \in \delta\big(\rho(t_1), \rho(t_2), \mathit{label}(t)\big)$.*

A run ρ of $\mathfrak{A}$ on T is said to be accepting *if it maps the root of T to a state in F. The* unary query *defined by $\mathfrak{A}$ is the query which maps every σ-labelled tree T to the set of those vertices $t \in \mathcal{U}^T$ that satisfy the following condition: $\rho(t) \in S$ for every* accepting *run ρ of $\mathfrak{A}$ on T.* $\Box$

It was shown in [FGK03,Nev99] that STAs can define exactly those unary queries on the class of labelled trees that are definable in monadic-second order logic.

Theorem 5 ([FGK03,GK02]). MonDatalog *is $2^{\mathcal{O}(m)}$-succinct in STAs on the class of labelled trees.* $\Box$

It is not hard to show that this result is asymptotically optimal, that is, that MonDatalog is not $2^{o(m)}$-succinct in STAs on the class of labelled trees (see [GS03] for details).

5.2. From S-MonDatalog to MonDatalog. In this section we show that S-MonDatalog-programs can be translated into MonDatalog-programs of at most exponential size. It remains open if the exponential size is indeed necessary or if, on the contrary, for every S-MonDatalog-program $\mathcal{P}$ there exists an equivalent MonDatalog-program $\mathcal{P}'$ of size polynomial in $||\mathcal{P}||$.

Lemma 5. *For every σ-STA $\mathfrak{A} = (Q, 2^\sigma, F, \delta, S)$ there is a* MonDatalog-*program $\mathcal{P}$ of size $\mathcal{O}\big(|Q|^3 \cdot |2^\sigma| + |\sigma| \cdot |2^\sigma|\big)$ that defines the complement of the query defined by $\mathfrak{A}$ on the class of all σ-labelled trees.* $\square$

Using Theorem 5 and Lemma 5 one easily obtains

Proposition 2. *For every* MonDatalog-*program $\mathcal{P}$ there is a* MonDatalog-*program $\mathcal{P}'$ of size $2^{\mathcal{O}(||\mathcal{P}||)}$ that defines the complement of the query defined by $\mathcal{P}$ on the class of labelled trees.* $\square$

Using the above proposition, it is not difficult to prove

Theorem 6. S-MonDatalog *is $2^{\mathcal{O}(m)}$-succinct in* MonDatalog *on the class of labelled trees.* $\square$

5.3. S-MonDatalog vs FL$_\mu$. From Theorem 4 and Lemma 5 one directly obtains

Theorem 7. FL$_\mu$ *is $2^{poly(m)}$-succinct in* S-MonDatalog *on the class of labelled trees.* $\square$

Conversely, it is not hard to show the following

Theorem 8. S-MonDatalog *is $2^{\mathcal{O}(m \cdot \lg m)}$-succinct in* FL$_\mu$ *on the class of labelled trees.* $\square$

It remains open whether the above bounds are optimal.

5.4. From MLFP to S-MonDatalog. Similarly to Theorem 1 one easily obtains

Theorem 9 (Folklore). MLFP-*sentences are Tower$\big(\mathcal{O}(m)\big)$-succinct in* S-MonDatalog *on the class of labelled trees.* $\square$

The aim of this section is to show that there are no essentially smaller translations from MLFP to S-MonDatalog. We will use the following well-known observation:

Proposition 3 (Folklore). *There is no function $f : \mathbb{N} \to \mathbb{N}$ with bound $f(m) \leqslant$ Tower$\big(o(m)\big)$ such that for every* FO($<$)-*sentence φ there is a nondeterministic finite automaton $\mathfrak{A}$ with at most $f(||\varphi||)$ states that accepts exactly those strings that satisfy φ.* $\square$

Using Proposition 3 and the results of the Sections 5.1 and 5.2, one obtains the following:

Theorem 10. *There is no function $f : \mathbb{N} \to \mathbb{N}$ with bound $f(m) \leqslant$ Tower$\big(o(m)\big)$ such that for every* FO($<$)-*sentence φ there is a* S-MonDatalog-*program $\mathcal{P}$ of size $||\mathcal{P}|| \leqslant f(||\varphi||)$ and a designated goal predicate $X \in IDB(\mathcal{P})$ such that $(\mathcal{P}, X)$ defines the same Boolean query as φ on the class of all finite strings.* $\square$

Since FO$(<)$ is included in MLFP, the above theorem directly implies the following:

Corollary 2. MLFP *is not* *Tower*$\big(o(m)\big)$*-succinct in* S-MonDatalog *on the class of all finite strings.* $\square$

It remains open if this result remains valid when replacing MLFP with MLFP^2. Note, however, that for the proof of Proposition 3 a small number k of first-order variables suffices. I.e., Proposition 3 remains valid when replacing FO$(<)$ with FO$^k(<)$, and Corollary 2 remains valid when replacing MLFP with MLFP^k.
Together with Corollary 1 and Lemma 5, the above Corollary 2 implies

Corollary 3. MLFP *is not* *Tower*$\big(o(m)\big)$*-succinct in* MLFP^2 *on the class of all finite strings.* $\square$

6. Conclusion

We studied the succinctness of a number of fixed point logics on trees. We believe that the analysis of succinctness, which may be viewed as a refined, "quantitative" analysis of expressive power, is a very interesting topic that deserves much more attention.

Even though we were able to get a good overall picture of the succinctness of monadic fixed point logics on trees, a number of questions remain open. Let us just mention a few of them:

– The exact relationship between monadic datalog, stratified monadic datalog, and the full modal μ-calculus remains unclear. In particular: Is the class of all queries whose complements can be defined by monadic datalog programs polynomially succinct in monadic datalog, or is there an exponential lower bound? (Recall that in Proposition 2 we prove an exponential upper bound.)
– Our proof that MSO is not *Tower*$(o(m))$-succinct in MLFP relies on a complexity theoretic assumption. Is it possible to prove this result without such an assumption?
– We have only considered the 2-variable fragment of MLFP here. What about the k-variable fragments, for $k \geqslant 3$? Do they form a strict hierarchy with respect to succinctness?

References

[ABS99] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML.* Morgan Kaufmann, 1999.

[AHV95] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases.* Addison-Wesley, 1995.

[AI00] N. Alechina and N. Immerman. Reachability logic: An efficient fragment of transitive closure logic. *Logic Journal of the IGPL*, 8(3):325–338, 2000.

[AI01] M. Adler and N. Immerman. An $n!$ lower bound on formula size. In *Proceedings of the 16th IEEE Symposium on Logic in Computer Science*, pages 197–206, 2001.

[CH82] A. Chandra and D. Harel. Structure and complexity of relational queries. *Journal of Computer and Systems Sciences*, 25:99–128, 1982.

[DGH$^+$02] E. Dantsin, A. Goerdt, E.A. Hirsch, R. Kannan, J. Kleinberg, C. Papadimitriou, P. Raghavan, and U. Schöning. A deterministic $(2 - 2/(k+1))^n$ algorithm for k-SAT based on local search. *Theoretical Computer Science*, 289(1):69–83, 2002. Revised version of: Deterministic algorithms for k-SAT based on covering codes and local search, ICALP'00, LNCS volume 1853.

[EF99] Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Springer, New York, second edition, 1999.

[EVW02] K. Etessami, M. Y. Vardi, and Th. Wilke. First-order logic with two variables and unary temporal logic. *Information and Computation*, 179(2):279–295, 2002.

[Fag75] R. Fagin. Monadic generalized spectra. *Zeitschrift für mathematische Logik und Grundlagen der mathematik*, 21:89–96, 1975.

[FG03] Markus Frick and Martin Grohe. The complexity of first-order and monadic second-order logic revisited. Journal version of LICS'02 paper. Available at http://www.dcs.ed.ac.uk/home/grohe/pub.html, 2003.

[FGK03] Markus Frick, Martin Grohe, and Christoph Koch. Query evaluation on compressed trees. In *18th IEEE Symposium on Logic in Computer Science (LICS'03)*, Ottawa, Canada, June 2003.

[FSW00] M. F. Fernandez, J. Siméon, and Ph. Wadler. An algebra for XML query. In S. Kapoor and S. Prasad, editors, *Proceedings of the 20th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'00)*, volume 1974 of *Lecture Notes in Computer Science*, pages 11–45. Springer-Verlag, 2000.

[GK02] Georg Gottlob and Christoph Koch. Monadic datalog and the expressive power of web information extraction languages. *Submitted*, November 2002. Journal version of PODS'02 paper. Available as CoRR report arXiv:cs.DB/0211020.

[GO99] E. Grädel and M. Otto. On Logics with Two Variables. *Theoretical Computer Science*, 224:73–113, 1999.

[GS03] M. Grohe and N. Schweikardt. Comparing the succinctness of monadic query languages over finite trees. Technical Report EDI-INF-RR-0168, School of Informatics, University of Edinburgh, 2003.

[HP00] H. Hosoya and B. C. Pierce. XDuce: A typed XML processing language (preliminary report). In D. Suciu and G. Vossen, editors, *International Workshop on the Web and Databases*, 2000. Reprinted in *The Web and Databases, Selected Papers*, Springer LNCS volume 1997, 2001.

[Imm99] Neil Immerman. *Descriptive complexity*. Springer, New York, 1999.

[Kam68] H. Kamp. *Tense Logic and the theory of linear order*. PhD thesis, University of California, Los Angeles, 1968.

[Koc03] Christoph Koch. Efficient processing of expressive node-selecting queries on XML data in secondary storage: A tree-automata based approach. In *Proceedings of the 29th Conference on Very Large Data Bases*, 2003. To appear.

[Nev99] Frank Neven. *Design and Analysis of Query Languages for Structured Documents – A Formal and Logical Approach*. PhD thesis, Limburgs Universitair Centrum, 1999.

[NS02] Frank Neven and Thomas Schwentick. Query automata over finite trees. *Theoretical Computer Science*, 275(1-2):633–674, 2002. Journal version of PODS'00 paper.

[Tho96] Wolfgang Thomas. Languages, automata, and logic. In G. Rozenberg and A. Salomaa, editors, *Handbook of formal languages*, volume 3. Springer, New York, 1996.

[Var98] Moshe Y. Vardi. Reasoning about the past with two-way automata. In K.G. Larsen, S. Skyum, and G. Winskel, editors, *25th International Colloquium on Automata, Languages and Programming (ICALP'98)*, volume 1443 of *Lecture Notes in Computer Science*, pages 628–641. Springer-Verlag, 1998.

[Wil99] T. Wilke. CTL+ is exponentially more succinct than CTL. In C. P. Rangan, V. Raman, and R. Ramanujam, editors, *Proceedings of the 19th Conference on Foundations of Software Technology and Theoretical Computer Science*, volume 1738 of *Lecture Notes in Computer Science*, pages 110–121. Springer-Verlag, 1999.