SUCCINCTNESS OF LOGICS ON TREES

Nikas Vasilis

December 2007

Contents

1 Introduction **2**

2 Preliminaries **6**
2.1 Logic 6
2.1.1 Fixed-Point Logics 7
2.1.2 Second Order Logic (SO) 10
2.1.3 Rank-k Types 11
2.2 Computational Complexity 14
2.2.1 The Model checking problem 16

3 Logic on Trees **19**
3.1 Trees 19
3.2 XML 22
3.2.1 XML Documents and Schemas 23
3.2.2 XPath 24
3.3 Tree Automata 28
3.3.1 Binary Tree Automata 28
3.3.2 MSO and Tree Automata 29
3.4 Expressive power of logics on trees 31

4 Succinctness **35**
4.1 MSO vs MLFP 36
4.1.1 Succinct Encodings 38
4.1.2 Proof of Theorem 4.2 46

5 Conclusion **50**

Chapter 1

Introduction

Logic has been called "the calculus of computer science". The argument is that logic plays a fundamental role in computer science. Indeed, logic plays an important role in areas of computer science as disparate as architecture (logic gates), software engineering (specification and verification), programming languages (semantics, logic programming), databases (relational algebra and SQL), artificial intelligence (automatic theorem proving), algorithms (complexity and expressiveness), and theory of computation (general notions of computability).

Finite Model Theory (FMT) is the study of logics on classes of finite structures. FMT arose as an independent field of logic for consideration of problems in theoretical computer science. Unfortunately, traditional *Model Theory* (MT), which concentrates on infinite models, cannot help us with finite structures. Most of the theorems that consist the base of MT do not apply in the restriction to finite models. For example the *Compactness Theorem* and in its wake most of the theorems of MT. Moreover other results such as *Löwenheim-Skolem Theorem* are meaningless. Consequently FMT has developed into a very different discipline from MT, with distinct methods, themes and applications of its own.

The first basic result of FMT was found in 1950 by Trakhtenbrot [22]: Validity over finite models is not *recursively enumerable* which means that *Completeness* fails over finite models. In 1960 Büchi showed that regular languages are precisely those definable in *monadic second-order* logic (MSO) over strings [11]. Fagin proved in 1974 that NP=ESO (existential second-order logic) which is remarkable since it is a characterization of the class NP which does not invoke a model of computation such as a *Turing Machine* [23].

The main sources of motivational examples for FMT are found in *Database*

Theory, *Computational Complexity* and *Formal Languages*.

Many of the problems of Database Theory can be formulated as problems of *mathematical logic*, provided that we limit ourselves to finite structures. While early database systems used rather ad hoc data models, from the early 70's the world switched to the *relational model*. Since Codd [5], databases have been modelled as first-order relational structures and database queries as mappings from relational structures to relational structures. It is, hence, not surprising that there is an intimate connection between database theory and FMT. *First order logic* (FO) lies at the core of modern database systems (the standard query languages such as SQL and QBE are syntactic variants of FO). More powerful query languages are based on extensions of FO with recursion (*least fixed-point logics*, etc.). A central topic of FMT has always been the *expressive power* of logics on finite relational structures. Thus a typical application of FMT on databases has to deal with questions of this kind: what can and what cannot be expressed in various *query languages*?

Another central issue in Finite Model Theory is the relationship between logical definability and Computational Complexity. We want to understand how the expressive power of a logical system, such as first-order or second-order logic, least fixed-point logic, or a logic-based database query language such as *Datalog*, is related to its algorithmic properties. Conversely, we want to relate natural levels of Computational Complexity to the defining power of logical languages. The aspects of Finite Model Theory that are related to Computational Complexity are also referred to as *Descriptive Complexity Theory*. Fagin's theorem on NP and ESO is the prototypical result of the field.

We can compare query languages by investigating the complexity of evaluating queries in these languages. There are three ways to measure the complexity of evaluating queries in a specific language. First, one can fix a specific query and study the complexity of applying it to arbitrarily databases. Following Vardi, we call this *data complexity* [32]. Alternatively, one can fix a specific database and study the complexity of applying queries represented by arbitrary expressions in the language (*expression complexity*). Finally, one can study the complexity of applying queries represented by arbitrary expressions in the language to arbitrary databases (*combined complexity*).

Many times languages with the same expressive power may differ a great deal on their complexity on evaluating queries. For example monadic Datalog and MSO have the same expressive power over trees but their combined complexities are very different. Monadic datalog queries can be evaluated in time linear both in the size of the datalog program and the size of the input tree [14]. On the other hand the evaluation of MSO queries is PSPACE-complete. The reason for this different behaviour is that in MSO we can

express queries much more *succinctly*.

Succinctness is a natural measure for comparing the strength of logics that have the same expressive power. Essentially, it is a finer-grained way of being able to say that one logic is "more expressive" than another: either it can define properties not definable in the other logic or it can define the same properties but with shorter formulae.

In this thesis, we study succinctness of logics on trees. Logics over (*unranked*) trees —that is, trees in which nodes can have arbitrary many children— have recently received much attention due to *XML* applications. XML is a data format which has become the lingua franca for information exchange on the World Wide Web. In particular, XML data is typically modelled as *labelled* unranked trees [33] (a labelled tree is a tree in which each node is given a unique label).

When restricting attention to labelled trees, MSO seems to be perfect: it has been proposed as a yardstick for expressiveness of XML query languages [14] and, due to its connection to *finite automata*, the data complexity of MSO-queries on strings and labelled trees is in polynomial time. On finite relational structures in general, however, MSO can express complete problems for all levels of the *polynomial time hierarchy* [15], i.e., MSO can express queries that are believed to be far too difficult to allow efficient *model checking*.

Monadic least fixed-point logic MLFP is a natural logic whose expressiveness lies between that of first-order logic and monadic second-order logic. MLFP is an extension of first-order logic by a mechanism that allows to define unary relations by induction. On finite relational structures in general, MLFP has the nice properties that:

1. The model-checking problem can be solved with *polynomial time* and *linear space* data complexity.

2. MLFP is suitable for the description of many important problems. For example, the transitive closure of a binary relation, or the set of winning positions in games on finite graphs [16] can be specified by MLFP-formulae.

3. On strings and labelled trees, MLFP even has exactly the same expressiveness as MSO [17].

In this dissertation I am bringing out representation of the proof of an interesting theorem concerning succinctness of logics on trees. My purpose is to illustrate as clearly as possible the tightness of the connection between Finite Model Theory and Database Theory. I chose succinctness because it

combines all the central topics of Finite Model Theory: expressiveness, automata and computational complexity. XML was my motivation. The fact that it became the lingua franca of data exchange on the web has drawn the attention of many researchers.

In the beginning we provide the necessary background (from mathematical logic and Complexity Theory). We focus on the logics that we are going to need which are MSO and MLFP. We also make an introduction to the model checking problem. In Chapter 3 after we introduce trees, automata, and XML we study the expressive power of logics on trees. In Chapter 4, which is the main chapter, I present some results that Grohe and Schweikardt have drawn by comparing succinctness of MSO and MLFP [1].

Chapter 2

Preliminaries

2.1 Logic

We do not attempt to give a fully comprehensive introduction to Finite Model Theory. The various facts that are quoted without proof can be found in the standard text books [3, 25]. We introduce some notation:

A *vocabulary* σ is a finite nonempty set of *constant* and *relational symbols* with associated arities.

$$\sigma = \langle R_1^{r_1}, \ldots, R_m^{r_m}, c_1, \ldots, c_n \rangle .$$

A σ-structure $\mathfrak{A}$ is a tuple consisting of a finite set $A \neq \emptyset$, the domain of $\mathfrak{A}$, together with an interpretation $R^{\mathfrak{A}} \subseteq A^r$ for each relation symbol R in σ (r is the arity of R) and an interpretation $c^{\mathfrak{A}} \in A$ for each constant symbol in τ. STRUCT[σ] is the class of all σ-structures.

$$\mathfrak{A} = \langle A, R_1^{\mathfrak{A}}, \ldots, R_m^{\mathfrak{A}}, c_1^{\mathfrak{A}}, \ldots, c_n^{\mathfrak{A}} \rangle .$$

Typical examples of structures are graphs. If E is a symmetric irreflexive binary relation and $\sigma = \langle E \rangle$ then STRUCT[σ] is the class of *undirected simple graphs*. Moreover relational databases are just a finite relational structure over some vocabulary and each table is a relation.

$$\text{Relational Databases} \longleftrightarrow \text{Relational Structures.}$$

The main use of a database is to query its data. A query language is language that allows retrieval and manipulation of data from a database. Logical formulae can express queries.

$$\text{Query Languages} \longleftrightarrow \text{Logics.}$$

We assume that the reader is familiar with first-order logic (FO) [18] and with the basics of Model Theory [7]. FO turned out to be very successful as a query language. The main reason of that is that FO can be efficiently implemented by using *relational algebra*, which provides a set of simple operations on relations expressing all FO queries. Relational algebra as used in the context of databases was introduced by Ted Codd [5]. His realization that the algebra can be used to efficiently implement FO queries gave the initial impetus to the birth of relational database systems.

Unfortunately FO is not the perfect query language. Useful queries, such as *connectivity* of finite graphs, cannot be expressed in FO [3]. Such facts have led to the introduction of a variety of query languages extending FO.

First Order lacks of any mechanism for performing *iterations* and cannot make *recursive definitions*. The observation that an extension of FO allowing iteration and recursion can express queries that cannot be expressed in FO led to *fixed-point logics*.

2.1.1 Fixed-Point Logics

Before we define fixed-point logics we first review the basics of fixed-point theory. We deal only with finite sets although the basic results hold for infinite sets, too.

Given a set S let $\mathcal{P}(S)$ be its powerset. An *operator* on a set S is a map $f : \mathcal{P}(S) \to \mathcal{P}(S)$. An operator f is:

- *monotone* if $X \subseteq Y \Rightarrow f(X) \subseteq f(Y)$, for all $X, Y \in \mathcal{P}(S)$,
- *inflationary* if $X \subseteq f(X)$, for all $X \in \mathcal{P}(S)$,
- *inductive* if $X^i \subseteq X^{i+1}$ for all i where where $X^0 = \emptyset$ and $X^{i+1} = f(X^i)$.

If f is inductive we define

$$X^\infty = \bigcup_{i=0}^{\infty} X^i .$$

As S is finite the sequence $(X^i)_{i\in\mathbb{N}}$ stabilizes after some finite number of steps, so there is a number n such that $X^\infty = X^n$. It's not hard to see that if $X^n = X^{n+1}$ then $X^n = X^{n+k}$ for all $k \geq 0$. Moreover the fact that there are at most $|S|$ subsets $S_1, \ldots, S_k \in \mathcal{P}(S)$ such that $S_i \subset S_{i+1}$ for $i \in \{1, \ldots, k-1\}$ implies that if $n = \min\{m \mid X^m = X^m + 1\}$ then $n \leq |S|$.

Definition 2.1 *Given an operator $f : \mathcal{P}(S) \to \mathcal{P}(S)$, a set $P \subseteq S$ is a fixed-point of f if $f(P) = P$. A fixed-point $X \subseteq S$ is a* least fixed-point *if*

$X \subseteq R$ for every fixed point R of f. We write $\mathbf{lfp}f$ *for f's least fixed-point, if it has one.*

Some of the results of fixed-point theory that we will need are the following [19]:

Proposition 2.1

1. *Every monotone operator is inductive.*
2. *Every monotone operator f has a least fixed-point which can be defined as*

$$\mathbf{lfp}(f) = \bigcap\{Y \mid f(Y) \subseteq Y\}.$$

   (Every set Y such that $f(Y) \subseteq Y$ is called pre-fixed point *of f).*

3. *If f is an inductive operator then $\mathbf{lfp}(f) = X^\infty$, for the sequence X^i that we defined earlier.*

Least Fixed Point Logic (LFP)

Let σ be a vocabulary and X a relation symbol of arity k not in σ. Let $\varphi(X, x_1, \ldots, x_k)$ be a first-order formula of vocabulary $\sigma \cup \{X\}$. X and $x_1, \ldots, x_k$ are considered to be free in the formula. For any fixed σ-structure $\mathfrak{A}$ and any interpretation $R \subseteq A^k$ for X, φ defines the k-ary relation

$$\varphi^{(\mathfrak{A},R)} = \{\bar{a} \in A^k \mid (\mathfrak{A}, R, \bar{a}) \models \varphi\}\,.$$

So for each $\mathfrak{A} \in$STRUCT$[\sigma]$ the formula φ defines an operator

$$f_\varphi^{\mathfrak{A}} : \mathcal{P}(A^k) \to \mathcal{P}(A^k)\,.$$

The idea of fixed-point logics is that we add formulae for computing fixed-points of operators $f_\varphi^{\mathfrak{A}}$.

The next proposition is a consequence of Trakhtenbrot's theorem [22].

Proposition 2.2 *It is undecidable whether a first-order formula $\varphi(X, \bar{x})$ is monotone with respect to X.*

Because it is undecidable whether a first-order formula is monotone, we cannot define a logic based on taking fixed points of monotone formulae, as there is no algorithm to determine which strings are formulae. To rescue the situation we replace the undecidable semantic restriction that we may only write $\mathbf{lfp}\varphi$ if φ is monotone with a decidable syntactic restriction on φ. Given a formula φ that may contain a relational symbol R, we say that an occurrence of R is *negative* if it is under the scope of an odd number of negations, and is *positive* if it is under the scope of an even number of negations. We say that a formula is positive in R if there are no negative occurrences of R in it. Moreover if φ is monotone, there is a positive ψ such that $\mathbf{lfp}\psi = \mathbf{lfp}\varphi$ for every structure of the correct vocabulary. Therefore, making the change from 'monotone' to 'positive' doesn't decrease the expressive power of the system defined.

Proposition 2.3 *Let $\varphi(R, \bar{x}) \in$ FO. If φ is positive in R then $f^{\mathfrak{A}}_{\varphi}$ is monotone [3].*

For any vocabulary σ, LFP extends FO with the following formation rule:

- If $\varphi(R, \bar{x})$ is a formula positive in R, where $R \notin \sigma$ is k-ary and $\bar{t}$ is a tuple of terms, where $|\bar{x}| = |\bar{t}| = k$, then $[\mathbf{lfp}_{R,\bar{x}}\varphi(R, \bar{x})](\bar{t})$ is a formula, whose free variables are those of $\bar{t}$.

The semantics is defined as follows:

$$\mathfrak{A} \models [\mathbf{lfp}_{R,\bar{x}}\varphi(R, \bar{x})](\bar{a}) \text{ iff } \bar{a} \in \mathbf{lfp}(f^{\mathfrak{A}}_{\varphi}) .$$

Monadic least fixed-point logic (MLFP) is defined as the restriction of LFP where we can only take fixed-points of unary relations.

Simultaneous MLFP (S-MLFP)

Let σ be a vocabulary. For $i \in \{1, \ldots, n\}$, let X_i be a new unary relation symbol and let φ_i be a formula with free second-order variables $X_1, \ldots, X_n$ and free first order variables $x_1, \ldots, x_n$.

$$\begin{gathered} \varphi_1(x_1, X_1, \ldots, X_n) , \\ \vdots \\ \varphi_n(x_n, X_1, \ldots, X_n) . \end{gathered}$$

Each formula is positive in all the variables $X_1, \ldots, X_n$. On any σ-structure $\mathfrak{A}$ each of the φ_i defines a monotone operator

$$f^{\mathfrak{A}}_{\varphi_i} : (\mathcal{P}(A))^n \to \mathcal{P}(A) .$$

given by

$$f^{\mathfrak{A}}_{\varphi_i}(R_1, \ldots, R_n) = \{a \in A \mid (\mathfrak{A}, R_1, \ldots, R_n, a) \models \varphi_i\}\,.$$

A tuple $(R_1, \ldots, R_n)$ is called a *simultaneous fixed-point* of $(\varphi_1, \ldots, \varphi_n)$ in $\mathfrak{A}$ iff for all $i \leq n$, $f^{\mathfrak{A}}_{\varphi_i}(R_1, \ldots, R_n) = R_i$.

For all $i \leq n$ and $s \in \mathbb{N}$ we define

$$\begin{aligned} L^0_{\mathfrak{A},\varphi_i} &= \emptyset\,. \\ L^{\ell+1}_{\mathfrak{A},\varphi_i} &= f^{\mathfrak{A}}_{\varphi_i}(L^{\ell}_{\mathfrak{A},\varphi_1}, \ldots, L^{\ell}_{\mathfrak{A},\varphi_n})\,. \end{aligned}$$

As we said $f^{\mathfrak{A}}_{\varphi_i}$ is monotone so $L^{\ell}_{\mathfrak{A},\varphi_i} \subseteq L^{\ell+1}_{\mathfrak{A},\varphi_i}$ for all $\ell \in \mathbb{N}$ and since A is finite for some ℓ_0 we have $L^{\ell_0}_{\mathfrak{A},\varphi_i} = L^{\ell_0+1}_{\mathfrak{A},\varphi_i} = L^{\infty}_{\mathfrak{A},\varphi_i}$ for all $i \leq n$. We define $(L^{\infty}_{\mathfrak{A},\varphi_1}, \ldots, L^{\infty}_{\mathfrak{A},\varphi_n})$ to be the *simultaneous least fixed-point* of $(\varphi_1, \ldots, \varphi_n)$ in $\mathfrak{A}$.

Let S-MLFP be the logic whose syntax is defined by extending the rules for first order with the rule for forming simultaneous least fixed-point as above, with the semantics given by

$$(\mathfrak{A}, \bar{a}) \models [\mathbf{lfp}_{x_1, X_1, \ldots, x_n, X_n}(\varphi_1, \ldots, \varphi_n)]_{X_i}(\bar{y}) \Leftrightarrow \bar{a} \in L^{\infty}_{\mathfrak{A},\varphi_i}\,.$$

2.1.2 Second Order Logic (SO)

Syntax

We assume that for every $k > 0$, there are infinitely many variables $X^k_1, X^k_2, \ldots$ ranging over k-ary relations. A formula of SO can have both first-order and second-order free variables. We define SO terms, formulae and their free variables over a vocabulary σ as follows:

- Each FO variable x and each constant symbol of σ are FO terms. The only free variable of the term x is the variable x. The constant term c has no free variables.
- There are three kinds of atomic formulae:
  1. FO atomic formulae ($t = t', R(\bar{t})$, where t, t' are terms, $\bar{t}$ is a tuple of terms, and R is a relational symbol of σ, that has the same arity as the length of $\bar{t}$). The free first-order variables of the formulae $t = t'$ and $R(\bar{t})$ are the first-order variables of t, t' and $\bar{t}$ respectively.

2. $X(t_1, \ldots, t_k)$, where $t_1, \ldots, t_k$ are terms, and X is a second-order variable of arity k. The free first-order variables of this formula are the free first-order variables of $t_1, \ldots, t_k$. The free second-order variable is X.

- The formulae of SO are closed under the Boolean connectives $\vee, \wedge, \neg$ and first order quantification, with the usual rules for free variables.
- If $\varphi(\bar{x}, Y, \bar{X})$ is a formula, then $\exists Y \varphi(\bar{x}, Y, \bar{X})$ and $\forall Y \varphi(\bar{x}, Y, \bar{X})$ are formulae, whose free variables are $\bar{x}$ and $\bar{X}$.

Semantics

Suppose $\mathfrak{A} \in$STRUCT$[\sigma]$. For each formula $\varphi(\bar{x}, \bar{X})$, we define the notation $(\mathfrak{A}, \bar{b}, \bar{B}) \models \varphi(\bar{x}, \bar{X})$ where $\bar{b}$ is a tuple of elements of A of the same length as $\bar{x}$ and for $\bar{X} = (X_1, \ldots, X_\ell)$, with each X_i being of arity n_i, $\bar{B} = (B_1, \ldots, B_\ell)$, where each B_i is a subset of A^{n_i}. We give the semantics only for constructors that are different from those for FO:

- If $\varphi(\bar{x}, X)$ is $X(t_1, \ldots, t_k)$, where X is k-ary and $t_1, \ldots, t_k$ are terms, with free variables among $\bar{x}$, then $(\mathfrak{A}, \bar{b}, \bar{B}) \models \varphi(\bar{x}, \bar{X})$ iff the tuple $(t_1^{\mathfrak{A}}(\bar{b}), \ldots, t_k^{\mathfrak{A}}(\bar{b}))$ is in B.
- If $\varphi(\bar{x}, \bar{X})$ is $\exists Y \psi(\bar{x}, \bar{X}, Y)$, where Y is k-ary, then $(\mathfrak{A}, \bar{b}, \bar{B}) \models \varphi(\bar{x}, \bar{X})$ if for some $C \subseteq A^k$, it is the case that $(\mathfrak{A}, \bar{b}, \bar{B}, C) \models \psi(\bar{x}, \bar{X}, Y)$
- If $\varphi(\bar{x}, \bar{X})$ is $\forall Y \psi(\bar{x}, \bar{X}, Y)$, where Y is k-ary, then $(\mathfrak{A}, \bar{b}, \bar{B}) \models \varphi(\bar{x}, \bar{X})$ if for all $C \subseteq A^k$, it is the case that $(\mathfrak{A}, \bar{b}, \bar{B}, C) \models \psi(\bar{x}, \bar{X}, Y)$.

Monadic second-order logic (MSO) is defined as the restriction of SO where all second order variables have arity 1.

We will continue the analysis of MLFP and MSO in Chapter 3.

2.1.3 Rank-k Types

Before we continue we need a definition of *quantifier rank* and rank-k types for MSO formulae. Informally quantifier rank counts the number of nested quantifiers of a formula.

Definition 2.2 *The quantifier rank of a formula φ is defined by induction on the formula's structure:*

- *qr(φ)=0 if φ is atomic,*

- $qr(\neg\varphi)=qr(\varphi)$,
- $qr(\varphi \wedge \psi)=qr(\varphi \vee \psi)=\max\{qr(\varphi), qr(\psi)\}$,
- $qr(\exists x\varphi)=qr(\forall x\varphi)=qr(\exists X\varphi)=qr(\forall X\varphi)=1+qr(\varphi)$.

Let MSO$[k]$ be the set of all MSO-formulae of quantifier rank at most k.

Lemma 2.1 *Given a vocabulary σ and for every k,* MSO$[k]$ *over σ contains only finitely many formulae in m variables, up to logical equivalence.*

Proof. The proof is by induction on k.

- The base case is MSO[0]. There are only finite many atomic formulae, and hence only finitely many Boolean combination of those, up to logical equivalence.
- Suppose that it holds for k.
- Each formula $\varphi(x_1, \ldots, x_m, X_1, \ldots, X_l)$ from MSO$[k+1]$ is a Boolean combination of $\exists x\psi(x_1, \ldots, x_m, X_1, \ldots, X_l, x)$ where x is a first or second order variable and $\psi \in$ MSO$[k]$. By the inductive hypothesis the number of MSO$[k]$ is finite (up to logical equivalence) and hence the same can be concluded about MSO$[k+1]$ formulae with m FO free variables and ℓ SO free variables. □

An MSO k, m, ℓ-type is a set $S \subset$ MSO$[k]$ with m free first-order variables and ℓ free second-order variables such that for every $\varphi(x_1, \ldots, x_m, X_1, \ldots, X_\ell)$ from MSO$[k]$ either $\varphi \in S$ or $\neg\varphi \in S$. Given a structure $\mathfrak{A}$, an m-tuple $\bar{a} \in A^m$ and an ℓ-tuple $\bar{V}$ of subsets of A, we define the MSO *rank-k type* of $(\bar{a}, \bar{V})$ in $\mathfrak{A}$ be the set

$$\text{mso-tp}_k(\mathfrak{A}, \bar{a}, \bar{V}) = \{\varphi(\bar{x}, \bar{X}) \in \text{MSO}[k] \mid \mathfrak{A} \models \varphi(\bar{a}, \bar{V})\}\,.$$

When both $\bar{a}, \bar{V}$ are empty, $\text{mso} - \text{tp}_k(\mathfrak{A})$ is the set of all MSO$[k]$ sentences that are true in $\mathfrak{A}$.

Theorem 2.1 *The following hold for any k, ℓ, m:*

1. *There exists only finitely many* MSO k, m, ℓ*-types.*
2. *Let $T_1, \ldots, T_s$ enumerate all the* MSO k, m, ℓ *types. There exist* MSO$[k]$ *formulae $\gamma_i(\bar{x}, \bar{X}), i \in \{1, \ldots, s\}$, such that for every structure $\mathfrak{A}$, every m-tuple $\bar{a}$ of elements of A and every ℓ-tuple $\bar{V}$ of subsets of A, it the case that*

$$\mathfrak{A} \models \gamma_i(\bar{a}, \bar{V}) \text{ iff } \mathrm{mso} - \mathrm{tp}_k(\mathfrak{A}, \bar{a}, \bar{V}) = T_i \,.$$

3. *Each* MSO*[k] formula with m free first-order variables and ℓ free second-order variables is equivalent to a disjunction of some of the γ_i's.*

Proof.

1. We know that up to logical equivalence MSO$[k]$ is finite, for a fixed number m of variables. Let $\varphi_1(\bar{x}, \bar{X}), \ldots, \varphi_L(\bar{x}, \bar{X})$ enumerate all the non-equivalent formulae in MSO$[k]$ with free variables $\bar{x} = (x_1, \ldots, x_m)$ and $\bar{X} = (X_1, \ldots, X_\ell)$. Then an MSO k, m, l-type is uniquely determined by a subset of $\{1, \ldots, L\}$ specifying which of the φ_i's belong to it. But there are finitely many such subsets so the number of different MSO k, m, ℓ-types is finite.

2. Let $K_i \subseteq \{1, \ldots, L\}$ be the set that determines T_i and $\alpha_{K_i}(\bar{x}, \bar{X})$ be the following formula

$$\alpha_{K_i}(\bar{x}, \bar{X}) \equiv \bigwedge_{j \in K_i} \varphi_j \wedge \bigwedge_{k \notin K_i} \neg \varphi_k \,.$$

   For every $\bar{a} \in A^m$ and every $\bar{V} \in (\mathcal{P}(A))^\ell$ if $\mathfrak{A} \models \alpha_{K_i}(\bar{a}, \bar{V})$ then all the φ_j's with $j \in K_i$ are satisfied by $\bar{a}, \bar{V}$ and the φ_j's with $j \notin K_i$ are not satisfied by $\bar{a}, \bar{V}$. So

$$\begin{aligned} \mathrm{mso} - \mathrm{tp}_k(\mathfrak{A}, \bar{a}, \bar{V}) &= \{\varphi(\bar{x}, \bar{X}) \in \mathrm{MSO}[k] \mid \mathfrak{A} \models \varphi(\bar{a}, \bar{V})\} \\ &= \{\varphi_j(\bar{x}, \bar{X}), j \in \{1, \ldots, L\} \mid \mathfrak{A} \models \varphi_j(\bar{a}, \bar{V})\} \\ &= \{\varphi_j(\bar{x}, \bar{X}), j \in K_i\} \\ &= T_i. \end{aligned}$$

3. Every MSO$[k]$ formula is equivalent to some φ_j. We claim that

$$\varphi_j(\bar{x}, \bar{X}) \equiv \bigvee_{j \in K_i} \alpha_{K_i} \,.$$

   Indeed if $\mathfrak{A} \models \varphi_j(\bar{a}, \bar{V})$ then $\varphi_j \in \mathrm{mso} - \mathrm{tp}_k(\mathfrak{A}, \bar{a}, \bar{V})$. But we've already shown that $\mathrm{mso} - \mathrm{tp}_k(\mathfrak{A}, \bar{a}, \bar{V}) = T_{K_i}$ for some i and as result $\mathfrak{A} \models \alpha_i(\bar{a}, \bar{V})$.

   Conversely, if $\mathfrak{A} \not\models \varphi_j$ then $\varphi_j \notin \mathrm{mso} - \mathrm{tp}_k(\mathfrak{A}, \bar{a}, \bar{V})$ for any $\bar{a}, \bar{V}$ so $j \notin K_i$ for any i. □

2.2 Computational Complexity

Computational Complexity is the theory of computer science that contemplates the reasons why some problems are so hard to solve by computers [24]. So it is the theory that tries to answer the question: "As the size of the input to an algorithm increases, how do the running time and memory requirements of the algorithm change and what are the implications and ramifications of that change?"

Turing Machines (TM) are the most general computing devices. Despite their simplicity they can simulate arbitrary algorithms with inconsequential loss of efficiency. They were described in 1936 by Alan Turing [26]. The concept of the TM is based on the idea of a machine executing a well-defined procedure by changing the contents of an unlimited tape, which is divided into squares that can contain one of a finite set of symbols. The machine needs to remember one of a finite set of states and the procedure is formulated in very basic steps in the form of "If your state is 42 and the symbol you see is a "0" then replace this with a "1", move one symbol to the right, and move to state 17."

Definition 2.3 *A (*deterministic*) TM is a quadruple* $M = (K, \Sigma, \delta, s)$ *where*

- K *is a finite set of* states.
- Σ *is a finite set of symbols, (it is also called* alphabet*).* Σ *contains the* blank *(*$\sqcup$*) and the* first *(*$\triangleright$*) symbol.*
- $\delta : K \times \Sigma \to (K \cup \{h,$ *"yes", "no"*$\})\times\Sigma \times \{\leftarrow, \rightarrow, -\}$. *We assume that* h *(the* halting state*), "yes" (the* accepting state*), "no" (the* rejecting state*) and the* cursor directions $\leftarrow$ *for "left",* $\rightarrow$ *for "right" and* $-$ *for "stay", are not elements of* $\Sigma \cup K$.
- $s \in K$ *is the* initial state.

Let M be a TM. The input of M is the initial contents of the tape. It is always of the form $\triangleright w$ where $w \in (\Sigma - \{\sqcup\})^*$. Other than the finite word w, the remainder of the tape is blank. Initially the state is s and the cursor is pointing to the first symbol ($\triangleright$). Then the machine takes a step according to δ, changing its state, printing a symbol and moving the cursor; then it takes another step,... etc. We say that the TM halts if one of the three states "yes", "no", h is reached. At the first case we say that the machine accepts its input ($M(w) =$ "yes"), at the second that it rejects ($M(w) =$ "no") and at the third that the output on input $\triangleright w$ is w' where w' is the string of the

tape at the time of halting ($M(w) = w'$). If none of the above states are reached we say that M diverges and we write $M(x) = \nearrow$.

A *nondeterministic* Turing Machine is defined as a normal (deterministic) TM $N = (K, \Sigma, \Delta, s)$ with the the only difference that Δ is not a function but a relation $\Delta \subset K \times \Sigma \times [(K \cup \{h, \text{“yes”}, \text{“no”}\}) \times \Sigma \times \{\leftarrow, \rightarrow, -\}]$. So if the machine is in state q and sees symbol a, it may be thought of as moving to any state q', writing any character a' and moving the head in any direction d such that $(q, a, q', a', d) \in \Delta$. We say that the machine accepts if at least one possible computation path accepts (regardless of whether the others reject or even terminate), rejects if every path rejects (and, in particular, none accept and none diverge) and otherwise diverges.

Definition 2.4 *Let $L \subset (\Sigma - \{\sqcup\})^*$ be a language (a set of strings of symbols). Let M be a TM such that for any $x \in (\Sigma - \{\sqcup\})^*$*

$$\begin{array}{c} \textit{if } x \in L \textit{ then } M(x) = \textit{“yes” and} \\ \textit{if } x \notin L \textit{ then } M(x) = \textit{“no”}. \end{array}$$

Then we say that M decides *L.*

We say that M accepts *L whenever for any $x \in (\Sigma - \{\sqcup\})^*$*

$$\begin{array}{c} \textit{if } x \in L \textit{ then } M(x) = \textit{“yes” and} \\ \textit{if } x \notin L \textit{ then } M(x) = \nearrow. \end{array}$$

If L is decided by some MT it is called recursive *and if it is accepted by one it is called* recursively enumerable.

Let L be a language decided by a Turing Machine M and $f : \mathbb{N} \to \mathbb{N}$. We say that the machine M operate within time $f(n)$ if the number of transitions M makes before accepting or rejecting an input of size $|w|$ is $f(|w|)$. If M is deterministic we write $L \in \mathbf{TIME}(f)$, if M is nondeterministic then we write $L \in \mathbf{NTIME}(f)$.

Similarly $\mathbf{SPACE}(f(n))$ (respectively $\mathbf{NSPACE}(f(n))$) is the set of languages that are accepted by deterministic (respectively nondeterministic) Turing Machines such as the length of the string at every stage of the computation, appart from the infinite sequence of blanks at the end, is at most $|s| + f(|s|)$ where s is the input. In the case of nondeterministic Turing Machines the bound is on the amount of space used at every stage of every possible computation.

We call the above sets *complexity classes*. The best-known of them are the following:

$$\mathbf{L} = \bigcup_{k \geq =0} \mathbf{SPACE}(\log n^k)$$

$$\begin{aligned}
\mathbf{NL} &= \bigcup_{k\geq=0} \mathbf{NSPACE}(\log n^k) \\
\mathbf{P} &= \bigcup_{k\geq=0} \mathbf{TIME}(n^k) \\
\mathbf{NP} &= \bigcup_{k\geq=0} \mathbf{NTIME}(n^k) \\
\mathbf{PSPACE} &= \bigcup_{k\geq=0} \mathbf{SPACE}(n^k)\,.
\end{aligned}$$

Finally we assume that the reader is familiar with asymptotic notation ($\mathcal{O}, \Omega$, Θ, o and ω-notation) [6].

2.2.1 The Model checking problem

Complexity Theory defines its main concepts via acceptance of string languages by computational devices such as Turing Machines. So if we want to talk about computational complexity of evaluating formulae (or queries) on relational structures we must first *encode* finite structures and logical formulae as strings.

There are several ways to encode a structure. For each vocabulary σ, we will define a coding $\text{enc}_\sigma : \text{STRUCT}[\sigma] \to \{0,1\}^*$. Suppose we have a structure $\mathfrak{A} \in \text{STRUCT}[\sigma]$ where $A = \{a_1, \ldots, a_n\}$. To code $\mathfrak{A}$ we must assume a linear order on its universe. The order has no effect on the result of queries, but we need it to represent the encoding of a structure on the tape of a Turing Machine. Let $a_1 < \ldots < a_n$ enumerate A according to the chosen order. Each k-ary relation $R^{\mathfrak{A}}$ will be encoded by an n^k-bit string $enc_\sigma(R^{\mathfrak{A}})$ as follows. Consider an enumeration of all k-tuples over A, in the *lexicographic order* (i.e., $(a_1, \ldots, a_n) < (b_1, \ldots, b_n)$ if, and only if, there is some $j \leq n$ such that $a_i = b_i$ for all $i < j$ and $a_j < b_j$). Let $\bar{a}_j$ be the jth tuple in this enumeration, then the jth bit of $enc_\sigma(R^{\mathfrak{A}})$ is 1 if $\bar{a}_j \in R^{\mathfrak{A}}$ and 0 if $\bar{a}_j \notin R^{\mathfrak{A}}$. Constants are encoded as unary relations containing one element.

The universe A is coded as the binary string 1^n0, where $n = |A|$. A Turing Machine must know $|\mathfrak{A}|$ in order to use the encoding of a structure. We assume, without loss of generality, that σ contains only relational symbols, $\sigma = \{R_1, \ldots, R_m\}$. The encoding of a structure $\mathfrak{A} \in \text{STRUCT}[\sigma]$ is

$$enc_\sigma(\mathfrak{A}) = 1^n0\ enc_\sigma(R_1^{\mathfrak{A}}) \ldots enc_\sigma(R_m^{\mathfrak{A}})\,.$$

The complexity is usually measured in terms of $|A|$, not in terms of the length of its encoding, but this distinction makes little theoretical difference since it only introduces a polynomial factor.

Since the length of 1^n0 is $n+1$ and the length of $enc_\sigma(R_i^{\mathfrak{A}})$ is n^{p_i} where p_i is the arity of R_i, so the length of $enc_\sigma(\mathfrak{A})$, denoted by $|enc_\sigma(\mathfrak{A})|$ is

$$|enc_\sigma(\mathfrak{A})| = (n+1) + \sum_{i=1}^{p} n^{p_i} .$$

Let $\sigma = \langle R_1, \ldots, R_n \rangle$, to encode a formula we represent its syntax tree as a structure of the vocabulary $\sigma' = \langle \text{VAR}, \text{EXISTS}, \text{AND}, \text{NOT}, \text{EQUALS}, S_1, \ldots, S_n \rangle$ and we code that structure as a string. The above relations are defined as follows

- VAR(a) iff a is a variable,
- EXISTS(a, b, c) iff a codes $\exists x \varphi$, where x is the variable coded by b and φ is the formula coded by c,
- AND(a, b, c) iff a codes $\varphi \wedge \psi$, where φ and ψ are the formulae coded by b and c respectively,
- NOT(a, b) iff a codes $\neg\varphi$, where φ is the formula coded by b,
- EQUALS(a, b, c) iff a codes $b = c$, where b and c are variables,
- $S_i(a, b, \ldots, c)$ iff a codes $R_i(b, \ldots, c)$.

The *model checking problem* asks, given a model $\mathfrak{A}$ and a formula φ of a logic $\mathcal{L}$, whether $\mathfrak{A} \models \varphi$. Depending on which of the two parameters (the formula and the structure) are considered fixed, we get the three definitions of complexity for a logic that we also defined in the introduction.

Definition 2.5 *Let $\mathcal{C}$ be a complexity class and $\mathcal{L}$ a logic. We say that*

- *the combined complexity of $\mathcal{L}$ is $\mathcal{C}$ if the language $\{(enc_\sigma(\mathfrak{A}), enc_\sigma(\varphi)) \mid \varphi \in \mathcal{L}$ and $\mathfrak{A} \models \varphi\}$ is in $\mathcal{C}$,*
- *the data complexity of $\mathcal{L}$ is $\mathcal{C}$ if, for every sentence $\varphi \in \mathcal{L}$, the language $\{enc_\sigma(\mathfrak{A}) \mid \mathfrak{A} \models \varphi\}$ is in $\mathcal{C}$. Moreover if every class of structures that is computable in $\mathcal{C}$ is definable in $\mathcal{L}$, we say that $\mathcal{L}$ captures $\mathcal{C}$,*
- *the expression complexity $\mathcal{L}$ is $\mathcal{C}$ if, for every structure $\mathfrak{A}$ of appropriate vocabulary, the language $\{enc_\sigma(\varphi) \mid \mathfrak{A} \models \varphi\}$ is in $\mathcal{C}$.*

Proposition 2.4 *The data complexity of FO is in* **L**.

Proof.

We fix an FO-formula φ. To prove the proposition we need to construct a deterministic logarithmic space Turing Machine, M_φ that decides the language $L_\varphi = \{enc_\sigma(\mathfrak{A}) \mid \mathfrak{A} \models \varphi\}$. M_φ first computes $\ell = \lceil \log n \rceil$ and writes ℓ zeroes to its work tape for each variable in φ (we assume that every quantifier in φ binds a fresh variable). These ℓ-bit strings will store the current interpretation of the variables. The construction of the machine is by induction on $\mathrm{qr}(\varphi)$.

If $\mathrm{qr}(\varphi)$=0 then φ is a finite Boolean combination of atomic formulae. The machines uses the tape (where the encoding of the structure is stored) to determine the truth of these atomic formulae in turn and to compute the Boolean combination. None of the calculations requires more than logarithmic amount of storage.

Assume that we have machines for all formulae of quantifier rank n. If $\mathrm{qr}(\varphi) = n + 1$ then φ is a finite Boolean combination of formulae $\exists x\psi$ with $\mathrm{qr}(\psi) = n$. For each of these in turn the machine loops through all possible ℓ-bit strings representing the value of variable x and simulates the machine for ψ for each until it either finds a value for x that satisfies ψ or finds that none do. It also remembers the truth value for each subformula in its states and computes the relevant Boolean combination in its state to give the result. □

Theorem 2.2 *The data complexity of* MLFP *is in* **P**.

Proof.

Because of the previous theorem it suffices to show that formulae of the form $\mathbf{lfp}_{R,x}\varphi$ can be evaluated in polynomial time. We observe that if $f : \mathcal{P}(A) \to \mathcal{P}(A)$ is a **P**-computable monotone operator, then $\mathbf{lfp}(f)$ can be computed in polynomial time in $|A|$. Indeed we know that the fixed-point computation stops after at most $|A|$ iterations of which can be evaluated in polynomial time. □

Chapter 3

Logic on Trees

The purpose of this chapter is to explore the connection between trees, automata and XML. We start by making an introduction to tree structures and XML. We then define *XPath*, a popular query language for XML and show that every XPath-definable query is MSO-definable. Furthermore, we define tree automata and we study their relation to MSO. Finally, we use automata to prove that, over finite labelled trees, MSO and MLFP has the same expressive power.

3.1 Trees

Trees arise everywhere in computer science, and there are numerous formalisms in the literature for describing and manipulating trees. Some of these formalisms are declarative and based on logical specifications: for example, first-order logic, monadic second-order logic, and various temporal and fixed-point logics over trees. Others are procedural formalisms such as various flavours of tree automata, or *tree transducers*. All these formalisms have found numerous applications in verification, program analysis, logic programming, constraint programming, linguistics, and databases.

A tree may be *ranked*, if every node which is not a leaf has the same number of children, or unranked, if different nodes can have a different number of children. We call the ranked trees *binary* when they have the property that every node either has exactly two children or is a leaf. Moreover a tree is called *ordered* when the children of any node are ordered by a *sibling ordering*. We will focus on the class of ordered unranked trees.

Nodes in ordered unranked trees are elements of $\mathbb{N}^*$ that is, finite strings whose letters are natural numbers. A string $s = n_0 n_1 \dots$ defines a path from the root to a given node: one goes to the n_0th child of the root, then to the

n_1th child of that element, etc. We shall write $s_1 \circ s_2$ for the concatenation of strings s_1 and s_2, and ε for the empty string.

We now define some basic binary relations on $\mathbb{N}^*$. The *child* relation is

$$s \prec_{ch} s' \Leftrightarrow s' = s \circ i \text{ for some } i \in \mathbb{N}\,.$$

The *next-sibling* relation is given by:

$$s \prec_{ns} s' \Leftrightarrow s = s_0 \circ i \text{ and } s' = s_0 \circ (i+1) \text{ for some } s_0 \in \mathbb{N}^* \text{ and } i \in \mathbb{N}\,.$$

That is, s and s' are both children of the same $s_0 \in \mathbb{N}^*$, and s' is next after s in the natural ordering of siblings.

We shall use the superscript $*$ to denote the reflexive-transitive closure of a relation. Thus, $\prec^*_{ch}$ is the descendant relation (including self): $s \prec^*_{ch} s'$ iff s is a prefix of s' or $s = s'$. The transitive closure of the next-sibling relation, $\prec^*_{ns}$ is a linear ordering on the children of each node: $s \circ i \prec^*_{ns} s \circ j$ iff $i \leq j$. We shall be referring to younger/older siblings with respect to this ordering (the one of the form $s \circ 1$ is the oldest).

A set $D \subseteq \mathbb{N}^*$ is called *prefix-closed* if whenever $s \in D$ and s' is a prefix of s (that is, $s' \prec^*_{ch} s$, then $s' \in D$).

Definition 3.1 *A tree domain D is a finite prefix-closed subset of $\mathbb{N}^*$ such that $s \circ i \in D$ implies $s \circ j \in D$ for all $j < i$.*

Let Σ be a finite alphabet. We define Σ-trees as structures that consist of a universe and a number of relations on the universe.

Definition 3.2 (Σ-tree) *An ordered, unranked Σ-tree T is a structure*

$$T = \langle D, \prec^*_{ch}, \prec^*_{ns}, (P_a)_{a \in \Sigma} \rangle,$$

where D is a tree domain, $\prec^*_{ch}$ and $\prec^*_{ns}$ are the descendant relation and the sibling ordering, and the P_a's are interpreted as disjoint sets whose union is the entire domain D. We let Trees(Σ) *be the set of all Σ-trees.*

An unordered unranked tree is defined as a structure $\langle D, \prec^*_{ch}, (P_a)_{a \in \Sigma} \rangle$, where $D, \prec^*_{ch}$, and P_a's are as above.

Thus, a tree consists of a tree domain together with a *labelling* on its nodes, which is captured by the P_a predicates: if $s \in P_a$, then the label of s is a. In this case we write $\lambda_T(s) = a$ (*the labelling function*). Finally we will define binary trees and definable tree languages.

Definition 3.3 *A binary tree domain is a set $B \subseteq \{1,2\}^*$ that is prefix-closed. For every $s \in D$ either both $s \circ 1$ and $s \circ 2$ are in D or none of them is in D. A Σ-tree is a pair (D, λ) where D is a tree domain and λ is the labelling function (from D to Σ).*

We represent a tree $T = (D, f)$ as a structure

$$\mathfrak{M}_T = \langle D, \prec, (P_a)_{a\in\Sigma}, \mathrm{succ}_1, \mathrm{succ}_2\rangle.$$

The binary relation $\prec$ is interpreted as the prefix relation on D, P_a is interpreted as $\{s \in D \mid f(s) = a\}$, and $\mathrm{succ}_i(s_1 s_2) \Leftrightarrow s_2 = s_1 \circ i$.

The following standard representation of ordered unranked trees, that was introduced by Gottlob and Koch in [14], helps us translate ordered unranked trees into binary trees.

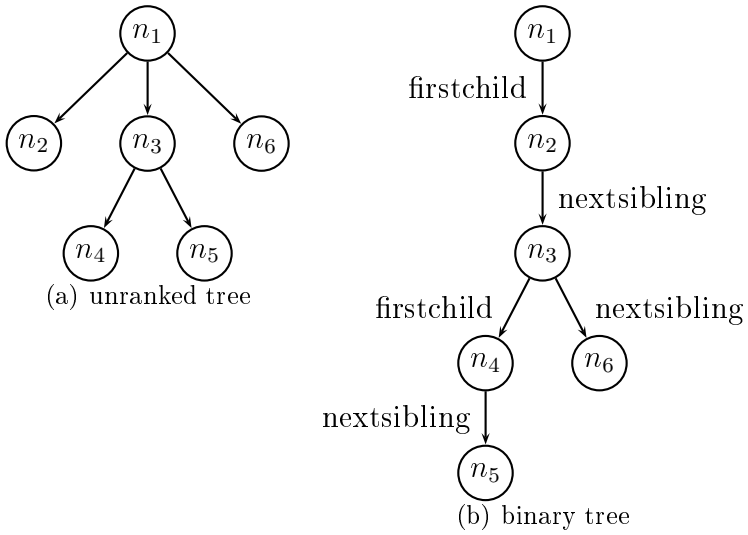


Figure 3.1: Example of translation

An unranked ordered tree can be considered as the structure

$$\mathfrak{T}_{ur} = \langle T, \mathrm{root}, \mathrm{leaf}, (\mathrm{label}_a)_{a\in\Sigma}, \mathrm{firstchild}, \mathrm{nextsibling}, \mathrm{lastsibling}\rangle,$$

where T is the set of nodes in the tree, "root", "leaf", "lastsibling", and the "$label_a$" relations are unary, and "firstchild" and "nextsibling" relations are binary. Most relations are defined according to their intuitive meanings or were defined earlier so we just focus on two of them. "firstchild(n_1, n_2)" is true iff n_2 is the oldest child of n_1, "nextsibling(n_1, n_2)" is true iff, for some i, n_1 and n_2 are the i-th and $(i+1)$-th children of a common parent node,

respectively, counting from the left. Finally, "lastsibling" contains the set of youngest children of nodes. Figure 1 makes the translation clear.

Let φ be a sentence of a logic $\mathcal{L}$ and let $\mathfrak{M}_T$ be the tree structure of the tree T. φ defines the set of trees (tree language) given by

$$L(\varphi) = \{T \mid \mathfrak{M}_T \models \varphi\}.$$

Definition 3.4 *A tree language L is definable in a logic $\mathcal{L}$, or is $\mathcal{L}$-definable, if there exists an $\mathcal{L}$-sentence φ such that $L = L(\varphi)$.*

We will finally define a representation of strings and we will show that we can identify a string with a tree. A string $w = w_0, \ldots, w_{n-1}$ of length $|w| = n$ over an alphabet Σ can be considered as the structure

$$w = \langle \{0. \ldots, n-1\}, \text{succ}, (P_a)_{a \in \Sigma} \rangle \ ,$$

where succ denotes the binary successor relation on $\{0, \ldots, n-1\}$ and $P_a = \{i \mid w_i = a\}$.

Let

$$T_w = \left\langle \{\varepsilon, 1, 11, \ldots, 1^{n-1}\}, \text{succ}_1, (P_a^{T_w})_{a \in \Sigma} \right\rangle \ .$$

It is easy to see that T_w is a unary tree —every node either has exactly one child or is a leaf— and that T_w and w are isomorphic.

3.2 XML

Databases and the Web are connected at many levels. Web pages are increasingly powered by databases. Because of the Web some of the basic assumptions of Database Theory should be revisited. A classical (relational) database is a coherently designed system. The system imposes rigid structure and provides queries in a controlled environment. The Web escapes any such control as it is a free-evolving, ever-changing collection of data sources of various forms, interacting according to a flexible protocol. So the necessity of a database model that would provide a flexible format for data exchange between different types of databases arose [31]. During the last years the World Wide Web Consortium (W3C) has adopted XML (eXtensible Markup Language) as the standard for data exchange on the Web. Hence XML has become a central component of modern data management. Unlike the relational model, whose birth was preceded by solid theoretical foundations, XML is the fruit of an often chaotic design process [30].

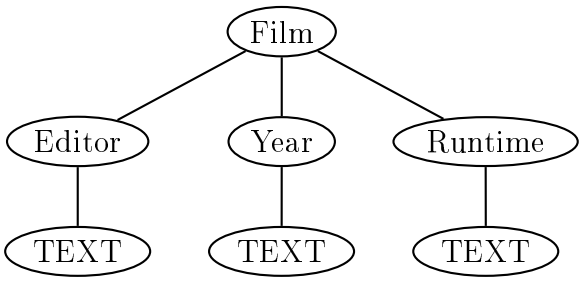
A new data model requires new tools and new techniques. As in the cross-fertilization between logic and databases, XML imposes new challenges on the area of automata and logic, while the latter area can provide new tools and techniques for the benefit of XML research.

3.2.1 XML Documents and Schemas

XML is an augmentation of HTML [27, 28]. HTML, short for HyperText Markup Language, is the predominant markup language for the creation of web pages; markup is the process of taking ordinary text and adding extra symbols. An *XML document* consists of nested elements with ordered sub-elements. Each element has a name (a *tag* or a *label*).

```
<film title= "The Fountain">
    <editor>
         Darren Aronofsky
    </editor>
    <year>
          2006
    </year>
    <runtime>
          97 min
    </runtime>
</film>
```

It can be easily seen that XML data can be modelled as an ordered, labelled, unranked tree.



with:

- $\lambda_T(\varepsilon)$ = "The Fountain"
- $\lambda_T(11)$ = "Darren Aronofsky"
- $\lambda_T(21)$ = "2006"
- $\lambda_T(31)$ = "97 min"

where λ_T is the labelling function.

An *XML schema* is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type, above and beyond the basic constraints imposed by XML itself. A number of standard and proprietary XML schema languages have emerged for the purpose of formally expressing such schemas, and some of these languages are XML-based, themselves. The basic schema mechanism for XML is provided by *Data Type Definitions* (DTDs). This defines the elements that may be included in an XML document, what attributes these elements have, and the ordering and nesting of the elements. A newer XML schema language, described by the W3C as the successor of DTDs, is *XML Schema*, or more informally referred to by the initialism for XML Schema instances, XSD (XML Schema Definition) [29]. While DTD provides a basic grammar for defining an XML Document in terms of the metadata that comprise the shape of the document, an XML Schema provides this, plus a detailed way to define what the data can and cannot contain. As a result XSDs are far more powerful than DTDs in describing XML languages.

3.2.2 XPath

XPath is a simple language for querying an XML tree and returning a set of nodes [8]. It is increasingly popular due to its expressive power and its compact syntax. These two advantages have given XPath a central role both in other key XML specifications and XML applications. It has been introduced by the W3C as the standard query language for retrieving information in XML documents. The current version of the language is XPath 2.0, but version 1.0 is still the more widely-used version.

In their simplest form XPath expressions look like "directory navigation paths". For example, the XPath

/director/genre/film

navigates from the root of a document (designated by the leading slash "/") through the top-level "director" element to its "genre" child elements and on to its "film" child elements. The result of the evaluation of the entire expression is the set of all the "film" elements that can be reached in this manner, returned in the order they occurred in the document. At each step in the navigation the selected nodes for that step can be filtered using qualifiers. A qualifier is a boolean expression between brackets that can test path existence. So if we ask for

/director/genre/film[year]

then the result is all "film" elements that have a least one child element named "year". The situation becomes more interesting when combined with XPath's capability of searching along *axes* other than the shown *children of* axis. Indeed the above XPath is a shorthand for

/child::director/child::genre/child::film[child::year],

where it is made explicit that each path step is meant to search the child axis containing all children of the previous context node. If we instead asked for

/child::director/descendant::*[child::year],

then the last step selects nodes of any kind that are among the *descendants* of the top element "director" and have a "year" sub-element.

Let's introduce the syntax of XPath. The primary syntactic construct in XPath is the *expression*.

Expression $e = /p \mid p$.
Path $p = p_1/p_2 \mid p[q] \mid e_1 \cup e_2 \mid e_1 \cap e_2 \mid (p) \mid a :: n$.
Qualifier $q = q$ and $q \mid q$ or $q \mid$ not $q \mid e$.
Axis a= child | descendant | self | descendant-or-self | parent | ancestor | ancestor-or-self | following-sibling | following | preceding-sibling | preceding.
Node Test $n = \tau \mid *$ where τ is an element of the alphabet of the XML-document.

The formal *semantics functions* S_e and S_p define the set of nodes returned by expressions and paths, starting from a context node x in the tree. The function S_q defines the semantics of qualifiers that basically state the existence or absence of one or more paths from a context node x. The semantics of paths uses the navigational semantics of axes S_a. First we will define the following functions:

- root() returns the root of the tree,
- children(x) returns the set of nodes which are children of the node x,
- parent(x) returns the set $\{y \mid x \in \text{children}(y)\}$,
- the relation $\prec$ defines the ordering:

$x \prec y$ iff the node x is before the node y in the *depth-first* traversal order of the n-ary XML tree,

- name(x) returns the XML labelling of the node x in a tree.

$S_e\,[/p]\,x = S_p\,[p]\,\mathrm{root}()\,.$
$S_e\,[p]\,x = S_p\,[p]\,x\,.$

$S_p\,[p_1/p_2]\,x = \{x_2 \mid x_1 \in S_p\,[p_1]\,x \wedge x_2 \in S_p\,[p_2]\,x_1\}\,.$
$S_p\,[p[q]]\,x = \{x_1 \mid x_1 \in S_p\,[p]\,x \wedge S_q\,[q]\,x_1\}\,.$
$S_p\,[e_1 \cup e_2]\,x = S_e\,[e_1]\,x \cup S_e\,[e_2]\,x\,.$
$S_p\,[e_1 \cap e_2]\,x = \{x_1 \mid x_1 \in S_e\,[e_1]\,x \wedge x_1 \in S_e\,[e_2]\,x\}\,.$
$S_p\,[(p)]\,x = S_p\,[p]\,x\,.$
$S_p\,[a :: s]\,x = \{x_1 \mid x_1 \in S_a\,[a]\,x \wedge \mathrm{name}(x_1) = s\}\,.$
$S_p\,[a :: *]\,x = \{x_1 \mid x_1 \in S_a\,[a]\,x\}\,.$

$S_q[q_1 \,\mathrm{and}\, q_2]x = S_q\,[q_1]\,x \wedge S_q\,[q_2]\,x\,.$
$S_q[q_1 \,\mathrm{or}\, q_2]x = S_q\,[q_1]\,x \vee S_q\,[q_2]\,x\,.$
$S_q[\mathrm{not}\, q]x = \neg S_q\,[q]\,x\,.$
$S_q\,[e]\,x = S_e\,[e]\,x \neq \emptyset\,.$

$S_a[\mathrm{child}]x = \mathrm{children}(x)\,.$
$S_a[\mathrm{parent}]x = \mathrm{parent}(x)\,.$
$S_a[\mathrm{descendant}]x = \mathrm{children}^+(x)\,.$
$S_a[\mathrm{ancestor}]x = \mathrm{parent}^+(x)\,.$
$S_a[\mathrm{self}]x = \{x\}\,.$
$S_a[\mathrm{descendant\text{-}or\text{-}self}]x = S_a[\mathrm{descendant}]x \cup S_a[\mathrm{self}]x\,.$
$S_a[\mathrm{ancestor\text{-}or\text{-}self}]x = S_a[\mathrm{ancestor}]x \cup S_a[\mathrm{self}]x\,.$
$S_a[\mathrm{preceding}]x = \{y \mid y \prec x\} - S_a[\mathrm{ancestor}]x\,.$
$S_a[\mathrm{following}]x = \{y \mid x \prec y\} - S_a[\mathrm{descendant}]x\,.$
$S_a[\mathrm{following\text{-}sibling}]x = \{y \mid \mathrm{parent}(y) = \mathrm{parent}(x) \wedge x \prec y\}\,.$
$S_a[\mathrm{preceding\text{-}sibling}]x = \{y \mid \mathrm{parent}(y) = \mathrm{parent}(x) \wedge y \prec x\}\,.$

As we have already said an XML-document can be seen as an ordered, labelled, unranked tree. We will show that every expression of XPath can be translated into MSO over trees. First we observe that all *navigational primitives* of XPath are MSO-definable. For example x is a descendant or self of y in an XML tree T iff $T \models y \prec_{ch}^* x$. For every axis a of an XML tree T we define the MSO-formula α with the property: x is "a" of y iff $\alpha(x, y)$ holds in T. Finally any XPath query of contex x and result y is translated into an MSO formula using the next figure

$$
\begin{aligned}
W_e[/p]_x^y &= \exists z(\forall w(z \prec^* w)) \wedge W_p[p]_z^y \,. && (3.1)\\
W_e[p]_x^y &= W_p[p]_x^y \,. && (3.2)\\
W_p[p_1/p_2]_x^y &= \exists z(W_p[p_1]_x^z \wedge W_p[p_2]_z^y) \,. && (3.3)\\
W_p[p[q]]_x^y &= W_p[p]_x^y \wedge W_q[q]_y \,. && (3.4)\\
W_p[e_1 \cup e_2]_x^y &= W_e[e_1]_x^y \vee W_e[e_2]_x^y \,. && (3.5)\\
W_p[e_1 \cap e_2]_x^y &= W_e[e_1]_x^y \wedge W_e[e_2]_x^y \,. && (3.6)\\
W_p[(p)]_x^y &= W_p[p]_x^y \,. && (3.7)\\
W_p[a :: \tau]_x^y &= a(x,y) \wedge y \in P_\tau \,. && (3.8)\\
W_p[a :: *]_x^y &= a(x,y) \,. && (3.9)\\
W_q[q_1 \,\text{and}\, q_2]_x &= W_q[q_1]_x \wedge W_q[q_2]_x \,. && (3.10)\\
W_q[q_1 \,\text{or}\, q_2]_x &= W_q[q_1]_x \vee W_q[q_2]_x \,. && (3.11)\\
W_q[\text{not}\, q]_x &= \neg W_q[q]_x \,. && (3.12)\\
W_q[e]_x &= \exists y(W_e[e]_x^y) \,. && (3.13)
\end{aligned}
$$

For example to translate the expression

child::director/descendant::year[parent::film],

we follow the next steps (at the end of each step we give the rule that we used):

- $W\Big[child :: director/descendant :: year[parent :: film]\Big]$,
- $\exists z_1\Big(W[child :: director]_x^{z_1} \wedge W\Big[descendant :: year[parent :: film]\Big]$, (3.3)
- $\exists z_1\Big(child(x,z_1) \wedge z_1 \in P_{director} \wedge W[descendant :: year]_{z_1}^y \wedge W\Big[[parent :: film]\Big]_y\Big)$, (3.8),(3.4)
- $\exists z_1\Big(child(x,z_1) \wedge z_1 \in P_{director} \wedge descendant(z_1,y) \wedge y \in P_{year}\Big) \wedge \exists z_2(W[parent :: film]_y^{z_2})$, (3.8),(3.13)
- $\exists z_1\Big(child(x,z_1) \wedge z_1 \in P_{director} \wedge descendant(z_1,y) \wedge y \in P_{year}\Big) \wedge \exists z_2(parent(y,z_2) \wedge z_2 \in P_{film})$. (3.8)

3.3 Tree Automata

The goal of this section is to study tree automata and their links with logic. We start with a definition of binary tree automata. We then introduce *rank-k types* of MSO formulae and we finish by proving Thatcher and Wright's theorem that the languages accepted by tree automata are exactly the languages defined by MSO formulae. An automaton is a mathematical model for a *finite state machine* (FSM). An FSM is a machine that, given an input of symbols, "jumps" through a series of states according to a transition function.

The connection between automata and logic goes back to work of Büchi [9] and Elgot [10] who showed that finite automata and monadic second-order logic (interpreted over finite words) have the same expressive power and that the transformations from formulae to automata and vice versa are effective, since the translations can be computed by a Turing Machine. Later, in work of Büchi [11], McNaughton [12] and Rabin [13], such an equivalence was shown also between finite automata and monadic second-order logic over infinite words and trees.

Tree automata deal with tree structures, rather than the strings of more conventional state machines. According to how the automata run on the input tree, finite tree automata can be of two types: (a) *bottom-up*, (b) *top-down*. This is an important issue, as although non-deterministic top-down and bottom-up tree automata are equivalent, deterministic top-down automata are strictly less powerful than their deterministic bottom-up counterparts. We will only deal with bottom-up tree automata.

3.3.1 Binary Tree Automata

Definition 3.5 (Tree automata and regular tree languages) *A nondeterministic tree automaton is a tuple $\mathcal{A} = (Q, \delta, F, q_0)$, where Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final (accepting) states, and $\delta : Q \times Q \times \Sigma \to \mathcal{P}(Q)$ is the transition function.*

Given a tree T with domain D a run of $\mathcal{A}$ on T is a function $r : D \to Q$ such that:

- *if s is a leaf labelled a, then $r(s) \in \delta(q_0, q_0, a)$,*
- *if $r(s \circ 1) = q$, $r(s \circ 2) = q'$ and s is a node labelled a, then $r(s) \in \delta(q, q', a)$.*

A run is accepting if $r(\varepsilon) \in F$, that is, the root is in an accepting state. A tree T is accepted by $\mathcal{A}$ if there exists an accepting run. We let $L(\mathcal{A})$ denote the set of all trees accepted by $\mathcal{A}$. Such sets of trees will be called regular.

The definition of a deterministic tree automaton is the same with the restriction that $|\delta(q, q', a)| = 1$ for all $q, q' \in Q$ and for all $a \in \Sigma$. So, in the deterministic case, we will treat δ as a function $Q \times Q \times \Sigma \to Q$.

3.3.2 MSO and Tree Automata

We now show the link between monadic second-order logic and tree automata.

Theorem 3.1 (Thatcher and Wright [34]) *A tree language is regular iff it is* MSO-*definable.*

Proof.

Let L be a regular set of Σ-trees. Then there is a finite tree automaton $\mathcal{A}$ that accepts exactly those trees that are elements of L. Assume that $Q = \{q_0, \ldots, q_{m-1}\}$ is the set of states of $\mathcal{A}$, q_0 is the initial state, $F \subseteq Q$ the set of accepting states and δ the transition function. We will construct an MSO-formula that defines L. Let φ be the sentence

$$\varphi \equiv \exists X_0 X_1 \ldots X_{m-1} \left[\varphi_p \wedge \varphi_\ell \wedge \varphi_t \wedge \varphi_a\right].$$

Suppose that $T = (D, \lambda)$ is a labelled binary tree and

$$\mathfrak{M}_T = \langle D, \prec, (P_a)_{a\in\Sigma}, \mathrm{succ}_1, \mathrm{succ}_2\rangle,$$

is its tree structure. The sets $X_0, X_1, \ldots, X_{m-1}$ code some run r of $\mathcal{A}$ on T in the sense that, for all $n \in D$, we want $n \in X_i \Leftrightarrow r(n) = q_i$.

The FO-formulae $\varphi_p, \varphi_\ell, \varphi_t, \varphi_a$ are defined as follows:

- φ_p asserts that $X_0, \ldots, X_{m-1}$ is a partition of D. An easy way to express that in FO is

$$\varphi_p \equiv \forall x \bigvee_{i=0}^{m-1} \left(X_i(x) \bigwedge_{i\neq j} \neg X_j(x)\right).$$

- φ_ℓ asserts that if s is a leaf labelled a then $r(s) \in \delta(q_0, q_0, a)$.

$$\varphi_\ell \equiv \forall x \Big[(\forall y \neg(x \prec y)) \to \bigvee_{a\in\Sigma} \Big(P_a(x) \wedge \bigvee_{q\in\delta(q_0,q_0,a)} X_q(x)\Big)\Big].$$

- φ_r asserts that if $r(s \circ 1) = q$, $r(s \circ 2) = q'$ and $\lambda(s) = a$ then $r(s) \in \delta(q, q', a)$.

$$\varphi_r \equiv \forall xyz[(\mathrm{succ}_1(x,y) \wedge \mathrm{succ}_2(x,z)) \rightarrow \bigvee_{0\le i,j<m} \bigvee_{a\in\Sigma} (X_i(y) \wedge X_j(z) \quad \wedge \bigvee_{q_k\in\delta(q_i,q_j,a)} X_k(x))] .$$

- φ_a asserts that the root is one of the accepting states.

$$\varphi_a \equiv \forall x((\forall y \neg(y \prec x)) \rightarrow \bigvee_{q_i\in F} X_i(x)) .$$

It is easy to see that $\mathfrak{M}_T \models \varphi \Leftrightarrow T \in L$ so $L(\varphi) = L$.

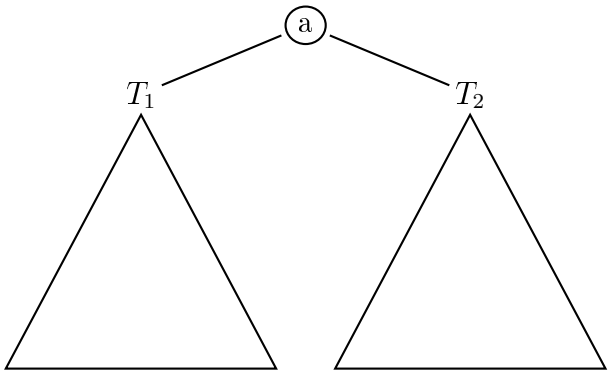
Conversely, let $\varphi \in$ MSO be a sentence of quantifier rank r in the vocabulary σ_Σ for Σ-trees. We construct an automaton $\mathcal{A}_\varphi = \langle Q, q_0, \delta, F\rangle$ that accepts the binary tree T iff $\mathfrak{M}_T \models \varphi$.

Let $\tau_1, \ldots, \tau_k$ be an enumeration of the finite number of rank-r MSO-types of Σ-trees (more precisely, $r, 0, 0$-types or, in other words, sentences). Let ψ_i be an MSO sentence of quantifier rank r defining the type τ_i. Then

$$\mathfrak{M}_T \models \psi_i \Leftrightarrow \text{mso-tp}_k(\mathfrak{M}_T) = \tau_i.$$

Since φ is a sentence of quantifier rank r it must be a disjunction of some subset of the τ_i: $\varphi \equiv \bigvee_{i\in I} \tau_i$. Put $Q = \{\tau_1, \ldots, \tau_k\}$ and let $F = \{\tau_i \mid i \in I\}$.

We further assume that τ_1 is the empty type of $\mathfrak{M}_\varepsilon$ where ε denotes the empty tree. That is, this is the only type among the τ_i's that is consistent with $\neg\exists x(x = x)$. Let $q_0 = \tau_1$.



The transition function $\delta : Q \times Q \times \Sigma \rightarrow 2^Q$ is defined as,

$$\tau_i \in \delta(\tau_j, \tau_l, a) \text{ if, and only if, there are trees } T_1 \text{ and } T_2 \text{ such that } \left(\begin{array}{c} \text{mso-tp}_k(\mathfrak{M}_{T_1}) = \tau_j \text{ and mso-tp}_k(\mathfrak{M}_{T_2}) = \tau_l \\ \text{and if } T \text{ is the tree obtained by "hanging"} \\ T_1 \text{ and } T_2 \text{ as children of a root node} \\ \text{labelled } a \text{ (as above) then mso-tp}_k(\mathfrak{M}_T) = \tau_i \end{array} \right)$$

Using Ehrenfeucht-Fraïssé games [20, 21] it can be easily shown that if T_1, T_2, T_3, T_4 are trees with mso-tp$_k(\mathfrak{M}_{T_1})$ =mso-tp$_k(\mathfrak{M}_{T_3})$ and mso-tp$_k(\mathfrak{M}_{T_2})$ =mso-tp$_k(\mathfrak{M}_{T_4})$ then mso-tp$_k(\mathfrak{M}_T)$ =mso-tp$_k(\mathfrak{M}_{T'})$ where T, T' are the trees obtained by hanging T_1, T_2 and T_3, T_4 respectively as children of a root node labelled a. As a result $\mathcal{A}_\varphi$ is deterministic.

We will prove, by induction that if T_s is the binary subtree that consists of s and every s' such that $s \prec s'$, then $r(s)$ =mso-tp$_k(\mathfrak{M}_{T_s})$. If $s = \varepsilon$ then $r(s) = \tau_1$, the initial state. But τ_1 is the MSO-type of the empty tree as required. Otherwise, s is a node with children s_1 and s_2. For $i \in \{1, 2\}$, let T_i be the subtree rooted at s_i ($T_1 = T_2 = \varepsilon$ in the case that s is a leaf). By the inductive hypothesis, $r(s_i)$ =mso-tp$_r(T_i)$ and, by definition of δ, $r(s)$ =mso-tp$_k(T)$, where T is the subtree rooted at s as required. □

3.4 Expressive power of logics on trees

In this section we will prove that, over finite labelled trees, MSO has the same expressive power as MLFP. First, we will show that for every MLFP-formula there is an equivalent MSO-formula. Conversely, since we have already shown that every MSO-formula can be translated into a tree automaton, we just need to prove that every regular language is MLFP-definable. To do that we will prove that every tree automaton can be translated into an S-MLFP-formula and that

Proposition 3.1 *Over finite structures* MLFP = S − MLFP.

Proof.

Let σ be a vocabulary, $n \in \mathbb{N}$ and let for each $i \leq n$, $\varphi_i(x_1, X_1, \ldots, X_n)$ be an MLFP-formula positive in $X_1, \ldots, X_n$. We will show that for every $i \leq n$ there is an MLFP-formula $\psi_{X_i}(x)$ such that for all $\mathfrak{A} \in$ STRUCT$[\sigma]$, $\mathfrak{A} \models \forall x(\psi_{Xi}(x)) \leftrightarrow [\mathbf{lfp}_{x_1, X_1, \ldots, x_n, X_n}(\varphi_1, \ldots, \varphi_n)]_{X_i}(x)$.

The proof is by induction on n.

- For $n = 1$ there is nothing to prove.
- Suppose that it holds for any $k < n$.

- Fix an $i \leq n$. By the induction hypothesis for every $j \in \{1, \ldots, n\} \setminus \{i\}$ there is an MLFP-formula ψ'_{X_j} in vocabulary $\sigma \cup \{X_i\}$ such that for all $\mathfrak{S} \in \mathrm{STRUCT}[\sigma \cup \{X_i\}]$,

$$\mathfrak{S} \models \forall x \Big(\psi'_{X_j}(x) \leftrightarrow$$

$$[\mathbf{lfp}_{x_1, X_1, \ldots, x_{i-1}, X_{i-1}, x_{i+1}, X_{i+1}, \ldots, x_n, X_n}(\varphi_1, \ldots, \varphi_{i-1}, \varphi_{i+1}, \ldots, \varphi_n)]_{X_i}(x) \Big) .$$

Let $\varphi'_i(x_i, i)$ be the MLFP-formula obtained from $\varphi_i(x_i, X_1, \ldots, X_n)$ by replacing every atom of the form $X_j(x)$, for $j \neq i$, with the formula $\psi'_{X_j}(x)$. We will prove that $[\mathbf{lfp}_{x_i, X_i}(\varphi'_i)](x)$ is the formula we are looking for.

For every $V \subseteq A$ and every $j \neq i$ let

$$\begin{aligned} K_j(V) &= \{a \in A \mid (\mathfrak{A}, V) \models \psi'_{X_j}(a)\} . \\ L_i &= \{a \in A \mid \mathfrak{A} \models [\mathbf{lfp}_{x_i, X_i}(\varphi'_i)](a)\} . \\ M_i &= \{a \in A \mid \mathfrak{A} \models [\mathbf{lfp}_{x_1, X_1, \ldots, x_n, X_n}(\varphi_1, \ldots, \varphi_n)]_{X_i}(a)\} . \end{aligned}$$

Since L_i is a fixed-point of φ'_i and from the definition of φ'_i we have

$$L_i = \{a \in A \mid \mathfrak{A} \models \varphi_i(K_1(L_i), \ldots, K_{i-1}(L_i), L_i, K_{i+1}(L_i), \ldots, K_n(L_i)\} .$$

Moreover from the definition of ψ'_{X_j} we have

$$K_j(L_i) = \{a \in A \mid \mathfrak{A} \models \varphi_j(K_1(L_i), \ldots, K_{i-1}(L_i), L_i, K_{i+1}(L_i), \ldots, K_n(L_i)\} ,$$

so $(K_1(L_i), \ldots, K_{i-1}(L_i), L_i, K_{i+1}(L_i), \ldots, K_n(L_i))$ is a simultaneous fixed-point of $(\varphi_1, \ldots, \varphi_n)$ in $\mathfrak{A}$. But $(M_1, \ldots, M_n)$ is the least fixed-point of $(\varphi_1, \ldots, \varphi_n)$ in $\mathfrak{A}$ so $M_i \subseteq L_i$.

On the other hand $(M_1, \ldots, M_{i-1}, M_{i+1}, \ldots, M_n)$ is a simultaneous fixed-point of $(\varphi_1, \ldots, \varphi_{i-1}, \varphi_{i+1}, \ldots, \varphi_n)$ in $(\mathfrak{A}, M_i)$ which least simultaneous fixed-point is $(K_1(M_i), \ldots, K_{i-1}(M_i), K_{i+1}(M_i), \ldots, K_n(M_i))$ so $K_j(M_i) \subseteq M_j$, for all $i \neq j$. We will show that M_i is a pre fixed-point of φ'_i

$$\{a \in \mathfrak{A} \mid (\mathfrak{A}, M_i) \models \varphi'_i(a)\} =$$

$$\{a \in A \mid \mathfrak{A} \models \varphi_i(K_1(M_i), \ldots, K_{i-1}(M_i), M_i, K_{i+1}(M_i), \ldots, K_n(M_i)\} \subseteq$$
$$\{a \in A \mid \mathfrak{A} \models \varphi_i(M_1, \ldots, M_{i-1}, M_i, M_{i+1}, \ldots, M_n\} = M_i .$$

By Proposition 2.1 we know that the least fixed-point of an operator is the intersection of all pre fixed-points, as a result $L_i \subseteq M_i$.

Since $L_i = M_i$

$$\mathfrak{A} \models \forall x \psi_{X_i}(x) \leftrightarrow [\mathbf{lfp}_{x_1, X_1, \ldots, x_n, X_n}(\varphi_1, \ldots, \varphi_n)]_{X_i}(x),$$

where $\psi_{X_i}(x) \equiv [\mathbf{lfp}_{x_i, X_i} \varphi_i'](x)$. □

Theorem 3.2 *Over finite labelled trees* MSO = MLFP.

Proof. Easily every formula of the form $[\mathbf{lfp}_{x,X} \varphi(x, X, \bar{y}, \bar{Y})](z)$ is equivalent to $\forall X \Big(\forall x \Big(\varphi(x, X, \bar{y}, \bar{Y}) \rightarrow X(x) \Big) \rightarrow X(z) \Big)$.

For the converse it suffices to prove that regular tree languages are S-MLFP definable. Let L be a regular tree language accepted by an automaton $\mathcal{A}$. Assume that $Q = \{q_0, \ldots, q_{m-1}\}$ is the set of states of $\mathcal{A}$, q_0 is the initial state, $F \subseteq Q$ the set of accepting states and δ the transition function. Similarly to the proof of Theorem 3.2 we construct an S-MLFP formula that defines L. Suppose that $T = (D, f)$ is a labelled binary tree and

$$\mathfrak{M}_T = \langle D, \prec, (P_a)_{a \in \Sigma}, \mathrm{succ}_1, \mathrm{succ}_2 \rangle \ ,$$

its tree structure. For $i \in \{0, \ldots, m-1\}$, the set X_i will be the set of nodes v of the tree such that there is a run r of $\mathcal{A}$ on T with $r(v) = q_i$. Let

$$\alpha_i(x) \equiv (\forall y \neg (x \prec y)) \wedge \bigvee_{a \in \Sigma \text{ s.t. } q_i \in \delta(q_0, q_0, a)} P_a(x) \,,$$

$$\beta_i(x) \equiv \exists y z \Big(\mathrm{succ}_1(x, y) \wedge \mathrm{succ}_2(x, z) \wedge \bigvee_{(a, q_j, q_k) \text{ s.t. } q_i \in \delta(q_j, q_k, a)} (P_a(x) \wedge X_j(y) \wedge X_k(z)) \Big) \,.$$

Let also

$$\varphi_i(x) \equiv \alpha_i(x) \vee \beta_i(x) \,,$$

$$\varphi(y) \equiv \exists x (\forall z \neg (z \prec x) \wedge \bigvee_{q_i \in F} X_i(x)) \,.$$

Formula α_i says that x is a leaf of the tree and it is labelled with a symbol a such that the automaton can move into state q_i when it reads a at a leaf. β_i says that x has left child y and right child z and the states q_j of y and q_k of z are such that, when the character a is read at x, the machine can move into state q_i. Finally φ says that there is a position in the word that is the last position and that the automaton can be in an accepting state when it reaches that node.

It is easy to see that $\varphi_i(x)$'s, φ are positive. The simultaneous fixed-point of the above formulae computes the relations: $X_i^\infty(a)$ holds iff there is a run r such that $r(a) = q_i$. Thus

$$\mathfrak{M}_T \models \exists z \Big(\mathbf{lfp}_{Y,X_1,\ldots,X_{m-1}}(\varphi, \varphi_1, \ldots, \varphi_{m-1})_Y(z) \Big) \Leftrightarrow T \in L\,. \quad \square$$

Chapter 4

Succinctness

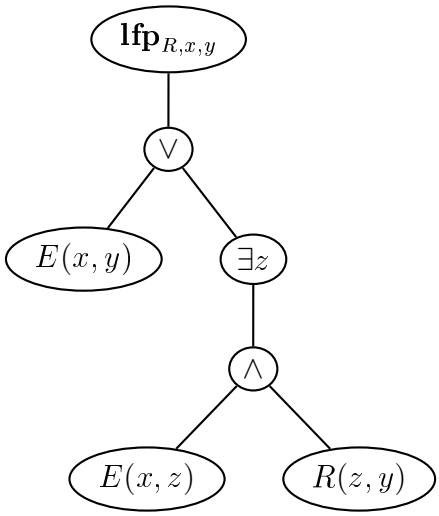
Succinctness is a natural measure for comparing the strength of logics that have the same expressive power. Intuitively, a logic L_1 is more succinct than another logic L_2 if all properties that can be expressed in L_2 can be expressed in L_1 by formulae of approximately the same size, but some properties can be expressed in L_1 by significantly smaller formulae.

In a natural way, we view formulae as their syntax trees, where leaves correspond to the atoms of the formulae, and inner vertices correspond to Boolean connectives, quantifiers and fixed-point operators. We define the size (or, length) $\|\varphi\|$ of a formula φ as the number of vertices in the syntax tree of φ.

For example the syntax tree of

$$\psi(u,v) \equiv [\mathbf{lfp}_{R,x,y} E(x,y) \vee \exists z(E(x,z) \wedge R(z,y))](u,v)\,.$$

is the following



So $\|\varphi\| = 7$.

Data complexity can be used as a measure of expressiveness as it replies to the question: "how difficult are the individual questions asked in this language?" Combined complexity, on the other hand, is a measure of succinctness of the language because it answers to the question: "How difficult is answering different questions asked in this language?"

Definition 4.1 *Let L_1 and L_2 be logics, let F be a class of functions from $\mathbb{N}$ to $\mathbb{N}$, and let C be a class of structures. We say that L_1 is F-succinct in L_2 on C iff there is a function $f \in F$ such that for every L_1-sentence φ_1 there is an L_2-sentence φ_2 of size $\|\varphi_2\| \leq f(\|\varphi_1\|)$ which is equivalent to φ_1 on all structures in C.*

Succinctness has received little attention so far. Most known results are about temporal and modal logics [35, 36, 37]. The motivations for these results has come from automated verification and model-checking.

4.1 MSO vs MLFP

We have already shown that, over finite trees, MSO and MLFP has the same expressive power. In this section we will compare their succinctness. After we define some functions that we will need in our proofs, we use the translation from MSO to tree automata, which we introduced in the previous chapter, to prove that MSO is Tower($\mathcal{O}(m)$)–succinct in MLFP. Afterwards we show that we cannot do essentially better by proving that (under a complexity theoretic assumption) there is no translation from MSO to MLFP of size Tower($o(m)$).

Definition 4.2 *Let* $\mathrm{Tower} : \mathbb{N} \to \mathbb{N}$ *be the function defined via*

$$\begin{aligned} \mathrm{Tower}(0) &= 1 \\ \mathrm{Tower}(n+1) &= 2^{\mathrm{Tower}(n)}. \end{aligned}$$

Let $T : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ be the function defined via

$$\begin{aligned} T(0,n) &= n \\ T(h+1,n) &= 2^{T(h,n)}. \end{aligned}$$

Let also

$$\begin{aligned} \log^{(0)}(k) &= k \\ \log^{(n+1)}(k) &= \log(\log^{(n)}(k)) \,. \end{aligned}$$

Finally let $\log^* : \mathbb{N} \to \mathbb{N}$ *be the function with the following property:*

$$\text{Tower}(\log^*(n) - 1) < n \leq \text{Tower}(\log^*(n)) \,.$$

In other words $\log^*$ *is the inverse function of* Tower.

Theorem 4.1 MSO*-sentences are* Tower$(\mathcal{O}(m))$*-succinct in* MLFP *on the class of all labelled trees.*

Proof. We have to prove that there is a function $f : \mathbb{N} \to \mathbb{N}$, with bound $f(m) \leq \text{Tower}(\mathcal{O}(m))$ such that for every MSO-formula φ_1 there is an equivalent MLFP-formula φ_2 of size $\|\varphi_2\| \leq f(\|\varphi_1\|)$.

In the previous chapter we showed that we can translate an MSO-formula, φ_1, of quantifier rank k into an automaton $\mathcal{A}_\varphi = \langle Q, q_0, \delta, F \rangle$ where $|Q|$ is equal to the number of MSO$[k]$-sentences.

We claim that there are Tower$(\mathcal{O}(k))$ MSO$[k]$-sentences. By Lemma 3.1 we know that there are finitely many MSO[0]-formulae. Let $\ell = |\text{MSO}[0]|$. We assume that there are $T(n, \ell)$ MSO$[n]$ types. Each formula $\varphi(x_1, \ldots, x_s, X_1, \ldots, X_d)$ from MSO$[n+1]$ is a Boolean combination of formulae of the form $\exists x \psi(x_1, \ldots, x_s, X_1, \ldots, X_d, x)$ where x is a first- or-second order variable and $\psi \in \text{MSO}[n]$. So there are $2^{T(n,\ell)} = T(n+1, \ell)$ MSO$[n+1]$ types and our claim is proved.

Using the method that we used in the proof of Theorem 3.3 we can translate an automaton with Tower$(\mathcal{O}(k))$ states into an S-MLFP-formula φ_3, where $\|\varphi_3\| \leq \text{Tower}(\mathcal{O}(k))$. Finally, by Proposition 3.1, φ_3 is equivalent to an MLFP-formula φ_2 of length $\|\varphi_2\| \leq \text{Tower}(\mathcal{O}(k))$.

The function that specifies the length of the above translation has the desired properties so the proof is completed. □

A formula is in conjunctive normal form (CNF) if it is a conjunction of clauses, where a clause is a disjunction of literals. We write CNF(n) to denote the class of all CNF-formulae the propositional variables of which are among $X_0, \ldots, X_{n-1}$.

The following theorem is due to Grohe and Schweikardt [1].

Theorem 4.2 *Let* $\gamma \in \text{CNF}(n)$*, unless* SAT *is solvable by a deterministic algorithm that has, for every* $i \in \mathbb{N}$ *time bound* $\|\gamma\|^{log^{(i)}(n)}$ MSO *is not* Tower$(o(m))$*-succinct in* MLFP *on the class of all finite trees.*

To prove the above theorem we will show that if MSO is Tower($o(m)$)-succinct in MLFP then we can construct a deterministic algorithm solving SAT that has, for every $i \in \mathbb{N}$, time bound $\|\gamma\|^{\log^{(i)}(n)}$. It is generally thought that there is no SAT-solver with such time bounds. The following figure shows the main steps of the algorithm.

$$\gamma \stackrel{1}{\to} w \stackrel{2}{\to} \varphi \in \text{MSO} \stackrel{3}{\to} \psi \in \text{MLFP} \stackrel{4}{\to} \text{Yes/No}\,.$$

The SAT-instance γ is the input. In (1) we construct a string w that represents γ. The algorithm in (2) computes an MSO-formula $\varphi(x)$ such that φ specifies (when evaluated in w) a canonical satisfying assignment for γ. Let f be the function that specifies the size of the translation from MSO to MLFP. In step (3) we check for all MLFP-formulae ψ where $\|\psi\| \leq f(\|\varphi\|)$ whether ψ specifies a satisfying assignment for γ. If it does the algorithm accepts its input otherwise it rejects.

It is obvious that $\|\varphi\|$ should be as short as possible so that the computation can proceed within the stated total time bounds. To achieve that we will use an encoding that Frick and Grohe introduced in [2].

4.1.1 Succinct Encodings

First of all we will give, for every $h \geq 0$, an encoding of natural numbers such that for every n there is a first order formula of length (of the binary encoding of the formula) $\mathcal{O}(n)$ stating that two words encode the same number k where $k \leq T(h, n)$. We will define a sequence of encodings μ_h, for $h \geq 1$, of natural numbers by words over certain finite alphabets.

For all $h \geq 1$ we let $\Sigma_h = \{0, 1, <1>, </1>, \ldots, <h>, </h>\}$. The tags $<i>, </i>$ represent single letters of the alphabet and are just chosen to improve readability. Let $L : \mathbb{N} \to \mathbb{N}$ such that $L(n)$ is the length of the binary representation $n-1$. So $L(0) = 0$, $L(1) = 1, L(n) = \lfloor \log(n-1) \rfloor + 1$ for $n \geq 2$. By $\text{bit}(i, n)$ we denote the i-th bit of the binary representation of n.

We encode every number $n \in \mathbb{N}$ by a string $\mu_h(n)$, so μ_h is a function from natural numbers to finite strings of alphabet Σ_h which is inductively defined as follows:

$$\begin{aligned} \mu_1(0) &= <1></1>\,, \\ \mu_1(n) &= <1>\,\text{bit}(0, n-1)\,\text{bit}(1, n-1)\ldots\,\text{bit}(L(n)-1, n-1)</1>\,, \end{aligned}$$

for $n \geq 1$. For $h \geq 2$, we let

$$\mu_h(0) = <h></h>\,,$$

$$\begin{aligned}\mu_h(n) &= < h > \\ &\quad \mu_{h-1}(0)\mathrm{bit}(0, n-1) \\ &\quad \mu_{h-1}(1)\mathrm{bit}(1, n-1) \\ &\quad \vdots \\ &\quad \mu_{h-1}(L(n)-1)\mathrm{bit}(L(n)-1, n-1) < /h > \ .\end{aligned}$$

Lemma 4.1

a) $|\mu_h(n)| \in \mathcal{O}(h \log^2 n)$.
b) There is an algorithm that, given $h, n \in \mathbb{N}$*, computes* $\mu_h(n)$ *in time* $\mathcal{O}(|\mu_h(n)|)$.

Proof.
a) We define functions $L_i : \mathbb{N} \to \mathbb{N}$ as follows: $L_1(n) = L(n)$, for all $n \in \mathbb{N}$ and $L_i(n) = L_{i-1}(L(n))$ for all $i, n \in \mathbb{N}$ with $i \geq 2$. We also define $P_i : \mathbb{N} \to \mathbb{N}$ for $i \geq 1$ where

$$P_i(n) = \prod_{j=1}^{i} L_j(n) \, .$$

It is easy to see that for all $i \geq 2$ and $n \geq 0$ we have $P_i(n) = L(n)P_{i-1}(L(n))$. We first prove, by induction on $h \geq 1$, that for all $n \geq 0$,

$$|\mu_h(n)| \leq 4h \cdot P_h(n) \, . \tag{4.1}$$

- For $h = 1$ we have $\mu_1(n) = 2 + L(n) \leq 4L(n) = 4P_1(n)$.
- Suppose that (4.1) holds for $h - 1$, where $h \geq 2$.
- We need to show that (4.1) holds for h:

$$\begin{aligned}|\mu_h(n)| &= 2 + L(n) + \sum_{i=0}^{L(n)} |\mu_{h-1}(i)| \\ &= 2 + L(n) + 2 + \sum_{i=1}^{L(n)} |\mu_{h-1}(i)| \\ &\leq 4 + L(n) + \sum_{i=1}^{L(n)} 4(h-1)P_{h-1}(i)\end{aligned}$$

$$\begin{aligned} &\leq 4 + L(n) + 4(L(n)-1)(h-1)P_{h-1}(L(n)) \\ &\leq L(n) + 4(h-1)L(n)P_{h-1}(L(n)) \\ &\leq L(n) + 4(h-1)P_h(n) \\ &\leq 4hP_h(n). \end{aligned}$$

As we know $L(n) \in \Theta(\log n)$ so to complete the proof of the lemma it suffices to show that there is a constant c such that for all $h \geq 1, n \geq 0$ we have $P_h(n) \leq cL(n)^2$. Since $L(L(n)) \in \mathcal{O}(\log\log n)$ and $L(n) \in \Omega(\log n)$, there is an n_0 such that for all $n \geq n_0$ we have

$$L(L(n))^2 \leq L(n)\,.$$

Let $P = \{P_h(m) \mid m < n_0, h \geq 1\}$. For $h > \log^*(n_0)$ we have that $L_h(m) \leq L_h(n_0) = 0$ and as a result $L_h(m) = P_h(m) = 0$. So there are only finitely many values of $m < n_0$ and h for which $P_h(m) > 0$. Since P is a finite set we can define $c = max(P)$. We prove that $P_h(n) \leq c \cdot L(n)^2$ by induction on $h \geq 1$:

- $P_1(n) = L(n)$, so this it is true for $h = 1$.
- Suppose that it also holds for $h-1$, where $h \geq 2$.
- We have $P_h(n) = L(n)P_{h-1}(L(n))$. If $L(n) < n_0$, then $P_{h-1}(L(n)) \leq c$ and thus $P_h(n) \leq cL(n)$. On the other hand if $L(n) \geq n_0$, we have $L(L(n))^2 \leq L(n)$. By induction hypothesis, $P_{h-1}(L(n)) \leq cL(L(n))^2$ so

$$P_h(n) = L(n)P_{h-1}(L(n)) \leq L(n)cL(L(n))^2 \leq cL(n)^2\,.$$

b) The algorithm computes $\mu_h(n)$ in a straightforward recursive manner. We get the following recurrence for the running time $R(h,n)$:

$$R(h,n) \leq \mathcal{O}(L(n)) + \sum_{i=0}^{L(n)} R(h-1, L(i))\,.$$

This recurrence is very similar to the one we used earlier and can easily be solved using the same methods. □

Lemma 4.2 *Let $h \geq 1$, $\ell \geq 0$. There is an FO(<)-formula $\chi_{h,\ell}(x,y)$ of size $\mathcal{O}(h\log h + \ell)$ such that for all words $\mathcal{W}, a, b$, and $m, n \in \{0, \ldots, T(h,\ell)\}$ the following holds: If a is the first position of a subword $\mathcal{U} \sqsubseteq \mathcal{W}$ with $\mathcal{U} \cong \mu_h(m)$ and b is the first position of a subword $\mathcal{V} \sqsubseteq \mathcal{W}$ with $\mathcal{V} \cong \mu_h(n)$, then*

$$(\mathcal{W}, a, b) \models \chi_{h,\ell}(x, y) \Leftrightarrow m = n\,.$$

Furthermore, the formula $\chi_{h,\ell}$ can be computed from h and ℓ in time $\mathcal{O}(h \log h + \ell)$.

Proof.

The proof is by induction on h. Let $h = 1$. Recall that the μ_1-encoding of an integer $p \geq 1$ is just the binary encoding of $p-1$ enclosed in <1>, </1>. Hence to say that x and y are the first positions of the μ_1-encodings of the same numbers, we have to say that if $x_1, \ldots, x_k$ are the positions between <1> and </1> and $y_1, \ldots, y_k$ are the positions between <1> and </1> then x_i and y_i should be both 1s or 0s. For numbers $p \in \{0, ..., T(1, \ell)\}$, there are at most $L(p) \leq \ell$ positions to be investigated. To express this, we let be the following formula:

$$\chi_{1,\ell}(x, y) \equiv \exists x_1 \ldots x_\ell y_1 \ldots y_\ell$$

$$\Big(\mathrm{succ}(x, x_1) \wedge \bigwedge_{i=1}^{\ell-1} \Big((P_{</1>}(x_i) \wedge x_i = x_{i+1}) \quad \vee \quad (\neg P_{</1>}(x_i) \wedge \mathrm{succ}(x_i, x_{i+1}))\Big) \wedge$$

$$\mathrm{succ}(y, y_1) \wedge \bigwedge_{i=1}^{\ell-1} \Big((P_{</1>}(y_i) \wedge y_i = y_{i+1}) \quad \vee \quad (\neg P_{</1>}(y_i) \wedge \mathrm{succ}(y_i, y_{i+1}))\Big) \wedge$$

$$P_{<1>}(x) \wedge P_{<1>}(y) \wedge \bigwedge_{i=1}^{\ell} \Big((P_0(x_i) \leftrightarrow P_0(y_i)) \quad \wedge \quad (P_1(x_i) \leftrightarrow P_1(y_i))\Big)\Big)\,.$$

Suppose that we have already defined $\chi_{h-1,\ell}(x, y)$.

First we define the following formulae

$$\chi^h_{\mathrm{int}}(x, y) \equiv P_{<h>}(x) \wedge (x < y) \wedge \forall z \Big((x < z \wedge z \leq y) \rightarrow \neg P_{</h>}(z)\Big)\,.$$

$$\chi^h_{\mathrm{last}}(x, y) \equiv P_{<h>}(x) \wedge (x < y) \wedge P_{</h>}(y) \wedge \forall z \Big((x < z \wedge z < y) \rightarrow \neg P_{</h>}(z)\Big)\,.$$

The first formula says that y is in the interior of the subword of the form $\mu_h(p)$ starting at x and the second one that y is the last position of the subword of the form $\mu_h(p)$ starting at x, provided such a subword indeed starts at x.

To say that the subwords starting at x and y are μ_h-encodings of the same numbers, we have to say that for all positions w between x and the next closing </h> and all positions z between y and the next closing </h>, if w and z are first positions of subwords isomorphic to $\mu_{h-1}(q)$ for some $q \in \mathbb{N}$, then the positions following these two subwords are either both 1s or both 0s. For all subwords of $\mu_h(p)$ of the form $\mu_{h-1}(q)$ we have $q \in \{0, \ldots, L(p)\}$.

In order to apply the formula $\chi_{h-1,\ell}$ to test equality of such subwords, we must have $q \leq L(p) \leq T(h-1,\ell)$. Observe that for all $h, \ell \geq 1$ we have

$$T(h,\ell) = \max\{n \in \mathbb{N} \mid L(n) \leq t(h-1,\ell)\} .$$

So the last inequality holds for all $p \leq T(h,\ell)$. Thus for such p we can use the formula χ_{h-1}, ℓ to test equality of subwords of $\mu_h(p)$ of the form $\mu_{h-1}(q)$. As a first approximation to our formula $\chi_{h,\ell}$, we let $\chi'_{h,\ell}(x,y)$ be the following

$$\begin{aligned}
&\forall w\Big(\Big(\chi^h_{\text{int}}(x,w) \wedge P_{<h-1>}(w)\Big) \rightarrow \exists z\Big(\chi^h_{\text{int}}(y,z) \wedge P_{<h-1>}(z) \wedge \chi_{h-1,\ell}(w,z)\Big)\Big) \wedge \\
&\forall z\Big(\Big(\chi^h_{\text{int}}(y,z) \wedge P_{<h-1>}(z)\Big) \rightarrow \exists w\Big(\chi^h_{\text{int}}(x,w) \wedge P_{<h-1>}(w) \wedge \chi_{h-1,\ell}(w,z)\Big)\Big) \wedge \\
&\forall wz\Big(\Big(\chi^h_{\text{int}}(x,w) \wedge P_{<h-1>}(w) \wedge \chi^h_{\text{int}}(y,z) \wedge P_{<h-1>}(z) \wedge \chi_{h-1,\ell}(w,z)\Big) \\
&\rightarrow \exists w'z'(\chi^{h-1}_{\text{last}}(w,w') \wedge \chi^{h-1}_{\text{last}}(z,z') \wedge \Big(P_1(\text{succ}(z')) \leftrightarrow P_1(\text{succ}(w'))\Big)\Big) .
\end{aligned}$$

The first line of this formula says that every subword of the form $\mu_{h-1}(q)$ in the subword of the form $\mu_h(p)$ starting at x also occurs in the subword of the form $\mu_h(p)$ starting at y. The second line says that every subword of the form $\mu_{h-1}(q)$ in the subword of the form $\mu_h(p)$ starting at y also occurs in the subword of the form $\mu_h(p)$ starting at x. The third and fourth line say that if w and z are the first positions of isomorphic subwords of the form $\mu_{h-1}(q)$, then they are either both followed by a 1 or both by a 0 (since the only two letters that can appear immediately after a subword $\mu_{h-1}(q)$ in a word $\mu_h(p)$ are 0 and 1). This formula says what we want, but unfortunately it is too large to achieve the desired bounds. The problem is that there are three occurrences of the subformula $\chi_{h-1,\ell}(w,z)$. We we can easily overcome this problem. We let

$$\begin{aligned}
&\zeta(w,z) \equiv \exists w'z'(\chi^{h-1}_{\text{last}}(w,w') \wedge \chi^{h-1}_{\text{last}}(z,z') \wedge P_1(\text{succ}(z')) \leftrightarrow P_1(\text{succ}(w'))) \text{ and} \\
&\chi_{h,\ell}(x,y) \equiv \forall w \exists z\Big(\Big(\chi^h_{int}(x,w) \rightarrow \chi^h_{int}(y,z)\Big) \\
&\qquad \wedge \Big(\chi^h_{\text{int}}(y,w) \rightarrow \chi^h_{\text{int}}(x,z)\Big) \wedge (P_{<h-1>}(w) \rightarrow P_{<h-1>}(z)) \\
&\qquad \wedge \Big(\Big((\chi^h_{\text{int}}(y,w) \vee \chi^h_{\text{int}}(x,w)) \wedge P_{<h-1>}(w)\Big) \rightarrow \\
&\qquad\qquad \chi_{h-1,\ell}(w,z) \wedge \zeta(w,z)\Big)\Big).
\end{aligned}$$

Observing that $\|\chi_{1,\ell}\| \in \mathcal{O}(\ell)$ and that $\|\chi_{h,\ell}\| = \|\chi_{h-1,\ell}\| + c \log h$ for some constant c, we obtain the desired bound on the size of the formulae. To see

why we need the factor $\log h$ here, recall that $\|\varphi_{h,\ell}\|$ is the length of a binary encoding of $\varphi_{h,\ell}$. The vocabulary of the formula $\varphi_{h,\ell}$ is of size $\mathcal{O}(h)$, thus the binary encoding of the symbols in this vocabulary will require $\mathcal{O}(\log h)$ bits. The fact that $\chi_{h,\ell}$ can be computed in time linear in the size of the output is immediate from the construction. □

Our goal is to encode a SAT instance or in other words a CNF formula so why do we care so much about encodings of natural numbers? We will show that we can use the encoding we defined to introduce an encoding of CNF-formulae where the numbers we code are used as the names of the variables in codings of formulae.

To encode a CNF-formula γ by a string we use the alphabet

$$\Sigma'_h = \Sigma_h \cup \{< \text{asn} >, < /\text{asn} >, < \text{val} >, < /\text{val} >, +, -, < \text{lit} >, < /\text{lit} >, \\ < \text{clause} >, < /\text{clause} >, < \text{cnf} >, < /\text{cnf} >, \star\}.$$

The literals $X_i, \neg X_i$, $i \in \mathbb{N}$, are encoded by the strings

$$\mu_h(X_i) =< \text{lit} > \mu_h(i) + < /\text{lit} > .$$
$$\mu_h(\neg X_i) =< \text{lit} > \mu_h(i) - < /\text{lit} > .$$

A clause $C = \ell_1 \vee \ldots \vee \ell_m$, where ℓ_i are literals, is encoded by the string

$$\mu_h(C) =< \text{clause} > \mu_h(\ell_1) \ldots \mu_h(\ell_m) < /\text{clause} >.$$

A CNF-formula $\gamma \equiv C_1 \wedge \ldots \wedge C_k$ is encoded by

$$\mu'_h(\gamma) =< \text{cnf} > \mu_h(C_1) \ldots \mu_h(C_k) < /\text{cnf} >.$$

Finally to make the manipulation of the encoding easier, we add the following string to it.

$$\begin{aligned} \mu_h(X_0, \ldots, X_{n-1}) = \; & < \text{asn} > \\ & \quad < \text{val} > \mu_h(0) \star < /\text{val} > \\ & \quad \vdots \\ & \quad < \text{val} > \mu_h(n-1) \star < /\text{val} > \\ & < /\text{asn} >. \end{aligned}$$

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

$$\mu_h(\gamma) = \mu'_h(\gamma) \mu_h(X_0, \ldots, X_{n-1}) .$$

We call any function $\alpha : \{X_0, \ldots, X_{n-1}\} \to \{\text{TRUE}, \text{FALSE}\}$ an assignment. Let P be a set of positions that carry the letter $\star$ in $\mu_h(\gamma)$. We will call p_i the

$\star$-position directly after the substring $\mu_h(i)$. It is easy to see that P specifies an assignment:

$$\alpha^P(X_i) = \text{TRUE} \Leftrightarrow p_i \in P\,.$$

Moreover any assignment α specifies a set:

$$P^\alpha = \{p_i \mid \alpha(X_i) = \text{TRUE}\}\,.$$

First we will show that our encoding is succinct and that it can be computed by a "fast" algorithm. By the definition of $\mu_h(\varphi)$ and $\mu_h(\alpha)$ and using Lemma 4.1 we can easily prove the following

Lemma 4.3 *Let $h \in \mathbb{N}$ and $\gamma \in \mathrm{CNF}(n)$ then $|\mu_h(\gamma)| = \mathcal{O}(h \log^2 n(\|\gamma\| + n))$. Moreover there is an algorithm that computes $\mu_h(\gamma)$ in time $\mathcal{O}(h \log^2 n\,(\|\varphi\| + n))$(that is, linear in the size of the output).*

Lemma 4.4 *For all $h, \ell \in \mathbb{N}$ there is a FO(<)-formula $\psi_{h,\ell}(P)$, where P is a free second-order variable, of size $\mathcal{O}(h \log h + \ell)$ such that for all $n \leq T(h,\ell)$, $\gamma \in \mathrm{CNF}(n)$*

$$\mu_h(\gamma) \models \psi_{h,\ell}(P) \Leftrightarrow \alpha^P \text{ is a satisfying assignment for } \gamma.$$

Furthermore, the formula $\psi_{h,\ell}$ can be computed in time $\mathcal{O}(h \log h + \ell)$.

Proof.

Let $\chi_{h,\ell}(x,y)$ be the formula defined in Lemma 4.2. We are going to use it to check if two subwords of $\mu_h(\gamma, \alpha)$ that represent variables actually represent the same variable.

Also recall the formula $\chi^h_{\text{last}}(x,y)$, defined in the proof of Lemma 4.2, which says that y is the last position of the subword of the form $\mu_h(n)$ starting at x.

We first define a formula $\psi^{\text{lit}}_{h,\ell}(x,P)$ such that the subword of φ starting at x is the encoding of the literal with the following property: If the literal is of the form X_i the $\star$-position p that occurs exactly after the substring $\mu_h(i)$ belongs to P, otherwise if the literal is of the form $\neg X_i$, $p \notin P$. So $\psi^{\text{lit}}_{h,\ell}(x,P)$ is true if the literal that starts at position x evaluates to true under the assignment P.

We let

$$\psi^{\text{lit}}_{h,\ell}(x,P) \equiv \exists y x' y' \Big(P_{<\text{val}>}(y) \wedge \chi_{h,\ell}(\text{succ}(x), \text{succ}(y)) \wedge \chi^h_{\text{last}}(\text{succ}(x), x') \wedge \chi^h_{\text{last}}(\text{succ}(y), y') \wedge \Big(P_+(\text{succ}(x')) \leftrightarrow P(\text{succ}(y')) \Big) \Big).$$

$\psi^{\text{lit}}_{h,\ell}(x,P)$ says that there are positions x', y and y' in the input such that the segment $x \dots x'$ defines a literal and $y \dots y'$ is the part of the assignment that gives a value to the variable in the literal $x \dots x'$ and that this value makes the literal true. In both formulae, succ is used as an abbreviation; we could rewrite them without it.

Next, we define a formula $\psi^{\text{clause}}_{h,\ell}(x,P)$ such that if the subword of φ starting at x is the encoding of a clause, then it contains a y that satisfies $\gamma^{\text{lit}}_{h,\ell}$, in other words the clause contains a satisfied literal. We let

$$\psi^{\text{clause}}_{h,\ell}(x,P) \equiv P_{<\text{clause}>}(x) \wedge \Big(\exists y(\forall z((x < z \wedge z \leq y) \rightarrow \neg P_{<\text{/clause}>}(z)) \wedge P_{<\text{lit}>}(y) \wedge \psi^{\text{lit}}_{h,\ell}(y,P)\Big).$$

Finally, we let

$$\psi_{h,\ell}(P) \equiv \forall y(P_{<\text{clause}>}(y) \rightarrow \psi^{\text{clause}}_{h,\ell}(y,P)).$$

It is easy to see that if there is a set of positions P that satisfies $\psi_{h,\ell}$ then the assignment α^P satisfies γ and that if α is a satisfying assignment for γ then the set $P = P^\alpha$ satisfies $\psi_{h,\ell}$. $\square$

Lemma 4.5 *There is an algorithm that given $h, \ell \in \mathbb{N}$ computes (the binary representation of) an* MSO*-formula $\Phi_{h,\ell}(z)$ in time $\mathcal{O}(h \log h)$, such that for all $n \leq \text{Tower}(h)$, for all $\Phi \in \text{CNF}(n)$ and for all positions p of $\mu_h(\gamma)$ we have*

$$\mu_h(\gamma) \models \Phi_h(p) \text{ iff } \begin{array}{c} \textit{in the lexicographically} \\ \textit{smallest satisfying assignment for } \varphi, \\ \textit{the propositional variable corresponding} \\ \textit{to position } p \textit{ is assigned the value true.} \end{array}$$

The (binary representation of the) formula $\Phi_h(z)$ has size $\|\Phi_h\| = \mathcal{O}(h \log h)$.

Proof.

As we proved in the previous lemma we can construct an FO($<$) formula $\psi_{h,\ell}(P)$ which holds in $\mu_h(\gamma)$ iff α^P is a satisfying assignment for γ. Before we construct Φ we have to define the relation $\leq_{\text{lex}}$. We let

$$Z \leq_{\text{lex}} W \equiv (\forall x Z(x) \leftrightarrow W(x)) \vee \exists y\Big(\neg Z(y) \wedge W(y) \quad \wedge \quad \forall x\Big(x < y \rightarrow (Z(x) \leftrightarrow W(x))\Big)\Big).$$

The following formula has the desired properties of Φ_h

$$\Phi'_h(z) \equiv \exists Z\Big(\forall x(Z(x) \to P_\star(x)) \quad \wedge \quad Z(z) \wedge \psi_{h,l}(Z) \wedge \\ \forall W\Big(\forall x(W(x) \to P_\star(x)) \quad \wedge \quad \psi_{h,l}(Z)\Big) \to Z \leq_{\text{lex}} W\Big)\,.$$

To get rid of the atoms $x < y$ in $\Phi'_h(z)$ we replace them by the formula

$$\exists Y\,(Y(y) \wedge \neg Y(x) \wedge \forall z_1 z_2\Big(\text{succ}(z_1, z_2) \wedge Y(z_1) \to Y(z_2)\Big)\,.$$

The resulting formula is $\Phi_h(z)$. □

4.1.2 Proof of Theorem 4.2

As we said our goal is to construct a fast SAT-solving algorithm. So far we know that the time we need for step 1 is $\mathcal{O}(h \log^2 n(\|\varphi\| + n))$, and for step 2 $\mathcal{O}(h \log h)$.

$$\gamma \xrightarrow{1} w \xrightarrow{2} \varphi \in \text{MSO} \xrightarrow{3} \psi \in \text{MLFP} \xrightarrow{4} \text{Yes/No}\,.$$

Let f be a function such that for every MSO-formula φ there is an equivalent MLFP-formula ψ of size $\leq f(\|\varphi\|)$. As we said in step 3 we check for all ψ where $\|\psi| \leq f(\|\varphi\|)$ whether it specifies a satisfying assignment for γ. But how many MLFP-formulae of length less than or equal to $f(\|\varphi\|)$ are there? Assume that our vocabulary has c_1 symbols. First and second order variables can't be more than $f(\|\varphi\|)$. So each letter of our string can be filled with $c_1 + f(|\,\|\varphi\|)$ ways. Therefore the number of such formulae is less than or equal to $\Big(c_1 + f(\|\varphi\|)\Big)^{f(\|\varphi\|)} \leq 2^{c_2 f(\|\varphi\|)\log(f(\|\varphi\|))}$.

We must also know how much time we need to find out whether ψ specifies a satisfying assignment. For each MLFP-formula ψ where $\|\psi\| \leq f(\|\varphi\|)$ we construct an assignment. For all $\star$-positions p_i in $\mu_h(\gamma)$ we check whether $\mu_h(\gamma) \models \psi(p_i)$, if so we set $\alpha(X_i) = \text{TRUE}$. Finally we check whether α is a satisfying assignment for γ.

Lemma 4.6 *There is an algorithm that, given an* MLFP*-formula* $\Psi(z)$*, a string* w*, a position* p *in* w*, decides in time* $|w|^{O(\|\Psi\|)}$ *whether* $w \models \Psi(p)$*.*

Proof.

We will prove that given an MLFP-formula $\Psi(x_1, \ldots, x_k, X_1, \ldots, X_\ell)$, a string w, a sequence $p_1, \ldots, p_k$ of positions in w, and sets $P_1, \ldots, P_\ell$ of positions in w we can decide in time $|w|^{\mathcal{O}(\|\Psi\|)}$ whether $w \models \Psi(p_1, \ldots, p_k, P_1, \ldots,$

P_ℓ). The algorithm operates by recursion on the construction of Ψ. We omit the trivial cases to examine the case when $\Psi \equiv [\mathbf{lfp}_{y,Y}\Psi'(y, Y, x_1, \ldots, x_k, X_1, \ldots, X_\ell)]_Y(x_i)$. The algorithm computes the stages of Ψ''s least fixed-point:

- $L^0 = \emptyset$.
- For $s = 1$ to $|w|$ do
  1. $L^s = \emptyset$.
  2. For $q = 0$ to $|w| - 1$ do
     check whether $w \models \Psi'(q, L^{s-1}, p_1, \ldots, p_k, P_1, \ldots, P_\ell)$. if so, then insert q into L^s.
- Check whether $p_i \in L^{|w|}$. if so, then "yes", otherwise "no".

Easily this computation takes $\mathcal{O}(|w|^2\, |w|^{\mathcal{O}(\|\Psi'\|)}) = |w|^{\mathcal{O}(\|\Psi\|)}$ steps. □

So it will take $|\mu_h(\gamma)|^{\mathcal{O}(f(\|\varphi\|)}$ steps to check whether $\mu_h(\gamma) \models \psi(p_i)$. Since steps 1 and 2 can be performed within a number of steps polynomial in $\|\gamma\|$, the total number of steps will be

$$|\mu_h(\gamma)|^{\mathcal{O}(f(\|\varphi\|)} \cdot 2^{c_2 f(\|\varphi\|)\log(f(\|\varphi\|))}.$$

which, by Lemmata 4.3 and 4.5, is less than or equal to

$$\Big(c_3 \cdot h \cdot (\log n)^2 \cdot (\|\gamma\| + n)\Big) \cdot 2^{c_2 f(c\cdot h)\log(f(c\cdot h))} \leq \|\gamma\|^{b\cdot f(c\cdot h)\log(f(c\cdot h))}.$$

So if we prove that $\|\gamma\|^{b\cdot f(c\cdot h)\log(f(c\cdot h))} \leq \|\gamma\|^{\log^{(i)}(n)}$ we are done.

Lemma 4.7 *Let $f : \mathbb{N} \to \mathbb{N}$ such that $f(\ell) \leq \mathrm{Tower}(o(\ell))$, let also $c, d \in \mathbb{N}$. For every $i \in \mathbb{N}$ there is an $n_0 \in \mathbb{N}$ such that for all $n \geq n_0$*

$$d \cdot f\Big(c \cdot \log^*(n)\Big) \cdot \log\Big(f(c \cdot \log^*(n))\Big) \leq \log^{(i)}(n).$$

Proof.

We know that since $f(\ell) \leq \mathrm{Tower}(o(\ell))$ there is an $m_0 \in \mathbb{N}$ and a function $g : \mathbb{N} \to \mathbb{N}$ with $g \in \omega(1)$ such that $f(m) \leq \mathrm{Tower}(\frac{m}{g(m)})$, for all $m \geq m_0$. So, for $t = c \cdot \log^*(n)$ we have

$$\begin{aligned} d \cdot f(t) \cdot \log(f(t)) &\leq d \cdot \mathrm{Tower}\Big(\frac{t}{g(t)}\Big) \cdot \mathrm{Tower}\Big(\frac{t}{g(t)} - 1\Big) \\ &\leq \mathrm{Tower}\Big(\frac{t}{g(t)} + 1\Big). \quad \textbf{(1)} \end{aligned}$$

We will show that $\log^{(i)}(n) \geq \text{Tower}(\log^*(n) - (i+1))$ **(2)**. By induction on i:

- For $i = 0$ it holds by the definition of $\log^*$.
- Suppose that it holds for $i = k$.
- By induction hypothesis

$$\begin{aligned} \log^{(k)}(n) &\geq \text{Tower}\Big(\log^*(n) - (k+1)\Big) \Rightarrow \\ \log(\log^{(k)})(n) &\geq \log\left(\text{Tower}\Big(\log^*(n) - (k+1)\Big)\right) \Rightarrow \\ \log^{(k+1)}(n) &\geq \text{Tower}\Big(\log^*(n) - ((k+1)+1)\Big)\,. \end{aligned}$$

Moreover since $g \in \omega(1)$ there is some $k_0 \geq m_0$ such that $g(m) \geq c \cdot (i+3)$, for all $m \geq k_0$. So

$$\begin{aligned} \frac{c \cdot \log^*(n)}{g(c \cdot \log^*(n))} + 1 &= \frac{c \cdot \Big(\log^*(n) - (i+3)\Big)}{g(c \cdot \log^*(n))} + \frac{c \cdot (i+3)}{g(c \cdot \log^*(n))} + 1 \\ &\leq \log^*(n) - (i+1)\,. \quad \textbf{(3)} \end{aligned}$$

Finally

$$\textbf{(1)}, \textbf{(2)}, \textbf{(3)} \Rightarrow d \cdot f\Big(c \cdot \log^*(n)\Big) \cdot \log\Big(f(c \cdot \log^*(n))\Big) \leq \log^{(i)}(n)\,. \qquad \square$$

We proved that the following algorithm can solve SAT in time $\mathcal{O}(\|\gamma\|^{\log^{(i)}(n)})$ which contradicts to our hypothesis.

- Input:γ (a SAT-instance).
- Set n =the number of variables occurring in γ.
- Rename the variables of γ such that only the variables $X_0, \ldots, X_{n-1}$ occur in it.
- Set $h = \log^*(n)$.
- Construct the string $\mu_h(\gamma)$.
- Construct the MSO-formula φ_h (Lemma 4.5).

- For all MLFP-formulae ψ where $\|\psi\| \leq f(\|\varphi_h\|)$.
    1. $\alpha = \emptyset$.
    2. check whether $\mu_h(\gamma) \models \psi(p_i)$. If so set $\alpha(X_i) =$ TRUE else set $\alpha(X_i) =$ FALSE.
    3. check whether α satisfies γ. If so Output: "α satisfies γ".
- Output: "γ is not satisfiable.

We have proven that, unless there is a deterministic SAT-solver that determines the satisfiability of a sentence γ in time $\|\gamma\|^{\log^{(i)} n}$ for all i (and it is generally thought that there isn't), then MSO is not Tower($o(m)$)-succinct in MLFP on trees.

Chapter 5

Conclusion

We studied succinctness of MSO and MLFP on trees and tried to make clear the link between Databases and logic. The reason that we restricted our attention on trees was because of XML. Moreover we chose studing MSO and MLFP because they both have very interesting properties over finite trees.

Our main result was that MSO is non-elementarily more succinct than MLFP. To prove that theorem we made a complexity theoretic assumption about the SAT problem. But how arbitrary was the assumption that we made? Finding a deterministic algorithm solving SAT, with worst case complexity less that or equal to $n^{\log(n)}$ although not answering the P=NP would be a surprising and unexpected breakthrough. The question whether there is a proof without such assumption remains open.

We close this dissertation with the following interesting results concerning logics over finite trees [1] together with some open questions:

1. MLFP is non-elementary more succinct than its 2-variable fragment MLFP^2. The question of what happens with the k-variable fragments, for $k \geq 3$ remains open.

2. MLFP^2 is exponentially more succinct than the full modal μ-calculus, that is, the modal μ-calculus with future and past modalities.

3. The full modal μ-calculus is at most exponentially more succinct than stratified monadic datalog. Conversely, stratified monadic datalog is at most exponentially more succinct than the full modal μ-calculus. Finally, stratified monadic datalog is at most exponentially more succinct than monadic datalog. The exact relationship between these three languages remains open.

Bibliography

[1] M. Grohe, N. Schweikardt. Comparing the succinctness of monadic query languages over finite trees, RAIRO Inf. Theor. Appl. 38 (2004) 343–373.

[2] Markus Frick , Martin Grohe. The Complexity of First-Order and Monadic Second-Order Logic Revisited. Proceedings of the 17th Annual IEEE Symposium on Logic in Computer Science, p.215–224, July 22–25, 2002

[3] Leonid Libkin. Elements of Finite Model Theory. Springer 2004.

[4] 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.

[5] Codd. E.F. Relational completeness of database sublanguage. In Data Base Systems (Rustin, Ed.), Prentice Hall, 1972, pp. 6508.

[6] Cormen Thomas, Leiserson Charles, Rivest Ronald, Stein Clifford. Introduction to Algorithms. MIT Press 2001.

[7] C. C. Chang and H. J. Keisler. Model Theory. Studies in Logic and the Foundations of Mathematics, Vol. 73, North-Holland, Amsterdam, 1973.

[8] Clark, J. and DeRose, S. 1999. XML path language (XPath) version 1.0. W3C Recommendation, World Wide Web Consortium.

[9] J.R. Büchi. Weak second-order arithmetic and finite automata. Zeit. Math. Logik Grundl. Math. 6 (1960), 66–92

[10] C.C. Elgot. Decision problems of finite automata design and related arithmetics, Trans. Amer. Math. Soc. 98, (1961) 21–52.

[11] J.R. Buchi. On a decision method in restricted second order arithmetic, in Proc. 1960 Int. Congr. for Logic, Methodology and Philosophy of Science, Stanford Univ. Press, Stanford, 1962, pp. 1–11.

[12] R. McNaughton. Testing and generating infinite sequences by a finite automaton, Inform. Contr. 9 (1966), 521–530

[13] M.O. Rabin. Decidability of second order theories and automata on infinite trees, Trans. Amer. Math. Soc. 141 (1969), 1–35.

[14] G. Gottlob, C. Koch, Monadic datalog and the expressive power of web information extraction languages, J. ACM 51 (1) (2004) 74–113 (Journal version of PODS' 02 paper).

[15] M. Ajtai, R. Fagin, L. Stockmeyer, The closure of Monadic NP, J. Comput. System Sci. 60 (3) (2000) 660-716 (Journal version of STOC'98 paper).

[16] E. Gradel, Model checking games, in: Proc. of WOLLIC 02, Electronic Notes in Theoretical Computer Science, Vol. 67. Elsevier, Amsterdam, 2002.

[17] W. Thomas, Languages, automata, and logic, in: G. Rozenberg, A. Salomaa (Eds.), Handbook of Formal Languages, Vol. 3, Springer, Berlin, 1996.

[18] E.Mendelson. Introduction To Mathematical Logic. CRC Press Inc 1987.

[19] Y.Moschovakis. Elementary Induction on Abstract Structures. North-Holland, 1974.

[20] A.Ehrenfeucht. An application of games to the completeness problem for formalized theories. Fundamenta Mathimaticae, 49 (1961), 129-141.

[21] R.Fraïssé. Sur quelques classifications des systemes de relations. Universite d'Alger, Publications Scientifiques, Serie A,1 (1954), 35-182.

[22] Boris A. Trakhtenbrot. Impossibility of an Algorithm for the Decision Problem in Finite Classes. Doklady Akademii Nauk SSSR 70:569–572, 1950. (In Russian; English translation: American Mathematical Society Translations, Series 2, 23: 1–5, 1963.)

[23] Ronald Fagin. Generalized First-Order Spectra and Polynomial-Time Regognizable Sets. In Richard Karp (ed), Complexity of Computation, SIAM-AMS Proceedings volume 7, pp.43–73, 1974.

[24] Christos H. Papadimitriou. Computational Complexity. Addison-Wesley, 1994.

[25] Neil Immerman. Descriptive Complexity. Springer, 1998.

[26] Alan Turing. "On Computable Numbers, With an Application to the Entscheidungsproblem". Proceedings of the London Mathematical Society, Series 2, Volume 42 (1936). Eprint. Reprinted in The Undecidable pp.115-154.

[27] Norman Walsh. "A Technical Introduction to XML", 1998. Available at http://nwalsh.com/docs/articles/xml/.

[28] T. Berners-Lee. "Hypertext Markup Language - 2.0", 1995. Available at http://www.ietf.org/rfc/rfc1866.txt.

[29] David C. Fallside, Priscilla Walmsley. "XML Schema Part 0: Primer", 2004. Available at http://www.w3.org/TR/xmlschema-0/

[30] Victor Vianu. XML: From Practice to Theory. Simposio Brasileiro de Bancos de Dados (SBBD): 11–25, 2003.

[31] Victor Vianu. A Web Odyssey: From Codd to XML. Symposium on Principles of Database Systems (PODS) , 2001.

[32] Moshe Y. Vardi. The Complexity of Relational Query Languages (Extended Abstract). In Proceedings of 14th ACM Symposium on Theory of Computing (STOC '82), pp. 137–146.

[33] F. Neven. Automata, logic, and XML. In CSL 2002, pages 2–26.

[34] J.W. Thatcher, J.B. Wright. Generalized finite automata with an application to a decision problem of second order logic, Math. Syst. Theory 2 (1968), 57–82.

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

[36] 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.

[37] 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.