Foundations of Regular Expressions in XML Schema Languages and SPARQL

Katja Losemann*
Universität Bayreuth
supervisor Wim Martens
expected date of graduation 2015

ABSTRACT

Regular expressions can be found in a wide array of technology for data processing on the web. We are motivated by two such technologies: schema languages for XML and query languages for graph-structured or linked data. Our focus is on theoretical aspects of regular expressions in these contexts.

Categories and Subject Descriptors

F.4.3 [**Mathematical Logic and Formal Languages**]: Formal Languages; H.2.3 [**Database Management**]: Languages—*query languages*

Keywords

regular expressions, schema languages, query evaluation

1. INTRODUCTION

Regular expressions have been studied for a long time in the context of database theory. They are used in *XPath* [4, 11], *graph databases* [2], and *regular path expressions* [13]. Often, they are augmented with additional semantic or syntactical extensions. These extensions may be syntactic sugar, such that the set of expressible languages remains the same. There are also constraints which restrict the set of regular expressions, such that only a subclass of regular languages is definable. Moreover, evaluation complexity depends on the used expressions and descriptive complexity can differ between two classes. Here, descriptive complexity means that languages which have large expressions in one class, can be expressible exponentially more succinct in another. Our main goal is to examine the characteristics of such classes of regular expressions in detail. By that we hope to optimize query and schema processing which is an overall long time goal. This work combines database research with formal language theory. Therefore we also aim at publishing some of our work in general theory conferences.

In our current work we focus on theoretical aspects of two applications of regular expressions in particular.

On the one hand, we investigate XML schema languages, like *DTDs* and *XML Schema* [36], where only deterministic regular expressions are allowed [9]. At the moment we want to determine the descriptive complexity of deterministic regular expressions. Then future work may include the design of algorithms that can generate preferably small deterministic expressions.

Our second field of interest is the *SPARQL Protocol and RDF Query Language (SPARQL)*. The *Resource Description Framework (RDF)* is used to represent linked data on the Web and is developed by the World Wide Web Consortium (W3C). The W3C gave a recommendation for a query language for RDF data, which is SPARQL. Recently the W3C decided to introduce *property paths* in SPARQL queries and thereby extend the navigational capabilities of queries by allowing the use of regular expressions with numerical occurrence indicators in queries [21]. Further they introduced a non-standard semantics on the evaluation of these regular expressions on graphs. These semantics require that some subexpressions of the query expression have to be matched onto simple paths. Then SPARQL queries have the ability of counting all different results of a query. At the moment our focus is on these special semantics of the query language, which are not without any problems [3, 25]. In [25] we have examined the complexity of evaluating property paths in SPARQL.

In the first part of the paper we will define the class of deterministic regular expressions. After that we will review related work on schema languages and summarize some of the basic ideas of our work. In the second half of this paper we will introduce the model of RDF and SPARQL more formally. Then we give a short overview of considered problems and further work on that topic.

2. DEFINITIONS

By Σ we denote a finite alphabet of symbols. A *word* w is a finite sequence of symbols $a_1 \cdots a_n$ for some $n \in \mathbb{N}$. The empty string is denoted by ε. *Regular expressions* (or *REs*) over Σ are defined as follows: Every symbol ε and $a \in \Sigma$ is a regular expression. Let r and s be regular expressions. Then so are $(r{\cdot}s)$, $(r{+}s)$, and $(s)^*$. For readability, we usually omit parentheses. The *size* $|r|$ of a regular expression r is defined to be the total number of occurrences of alphabet symbols,

*Supported by grant number MA 4938/2–1 from the Deutsche Forschungsgemeinschaft (Emmy Noether Nachwuchsgruppe).

SIGMOD/PODS'12 PhD Symposium, May 20, 2012, Scottsdale, AZ, USA.

epsilons, and operators. We say that a regular expression r is *minimal* if there does not exist another regular expression r' with $L(r') = L(r)$ and $|r'| < |r|$.

The language defined by an expression r, denoted by $L(r)$, is inductively defined as follows: $L(\varepsilon) = \{\varepsilon\}$; $L(a) = \{a\}$; $L(r \cdot s) = L(r) \cdot L(s)$; $L(r + s) = L(r) \cup L(s)$ and, $L(r^*) = \{\varepsilon\} \cup \bigcup_{i=1}^{\infty} L(r)^i$.

A *nondeterministic, finite automaton (NFA)* A is a tuple $(Q, \Sigma, \delta, q_0, F)$, where Q is a finite set of states, $\delta : Q \times \Sigma \to \mathcal{P}(Q)$ is a transition function, $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the set of accepting states. An automaton is deterministic (or a *DFA*), if $|\delta(q, a)| \leq 1$, for all $q \in Q, a \in \Sigma$. by reading w. As usual, we extend δ to words, i.e., $\delta(q, w)$ is the set of states that can be reached from q by reading w. A word w is *accepted* by A if $\delta(q_0, w) \cap F \neq \emptyset$. The language of A, which we denote by $L(A)$, consists of all words accepted by A. The *size* $|A|$ of an automaton is defined to be $|Q|$.

3. DESCRIPTIVE COMPLEXITY OF DETERMINISTIC EXPRESSIONS

Intuitively, a regular expression is *deterministic* (or *one-unambiguous* [9]) when the following holds. When reading the input string from left to right, a deterministic expression always allows to match each symbol of that string uniquely against a position in the expression, without looking ahead. Formally, let $\bar{r}$ stand for the regular expression obtained from r by replacing the i-th occurrence of alphabet symbol a in r (counting from left to right) by a_i, for every i and a. For example, for $r = (a + b)^*a$ we have $\bar{r} = (a_1 + b_1)^*a_2$.

Definition 3.1. *A regular expression r is* deterministic *(or a* DRE*) if there are no words wa_iv and wa_jv' in $L(\bar{r})$ such that $i \neq j$.*

The expression $(a + b)^*a$ is not deterministic, because a_1 and a_2 are both in $L((a_1 + b_1)^*a_2)$ (w, v, and v' are all empty). Brüggemann-Klein and Wood [9] showed that deterministic regular expressions define a strict subset of the regular languages. Thus we say, a regular language L is *DRE-definable* if there exists a DRE r with $L(r) = L$. The canonical example for a language that is not DRE-definable is $L((a + b)^*a(a + b))$.

In 1998 Brüggemann-Klein and Wood [9] present a decision algorithm that, given a minimal DFA A, decides whether $L(A)$ is DRE-definable or not in time polynomial in A. Further, they construct a second algorithm that, given the same input, outputs a DRE r for $L(A)$ if and only if $L(A)$ is DRE-definable. Thereby the size of r can be exponential in A.

3.1 Towards small Deterministic Expressions

Our main goal is to understand the class of DREs in more detail. As we have mentioned, we hope that the results on the descriptive complexity and therefore developed proof methods will help to optimize XML schema languages. Later we want to be able to construct small deterministic expressions for a given DRE-definable language.

Related work on XML schema languages and automata can be found in [17, 26, 27, 30, 31]. The complexity of regular languages for schemas was investigated in [28]. Further work on deterministic expressions can be found in [5, 19]. In [29] it was investigated that in most of the available schemas the used DREs are syntactically restricted. These restricted expressions are named SOREs and CHAREs. In [5, 7, 6, 8, 14] efficient learning algorithms for this class of regular expressions are given. Bex et al. [5] gave a variant of the decision algorithm of Brüggemann-Klein and Wood that outputs smaller DREs than the original.

In the future we want to optimize more and more general DREs. Therefore we want to develop approximation algorithms for finding preferably small DREs for a given language. A first step in this direction is determining the possible blow-up which can occur when translating a minimal DFA to a minimal DRE.

3.2 An exponential blow-up from RE to DRE

At the moment we are interested in the following fundamental questions in particular:

(A) Let A be a minimal DFA for a DRE-definable language. Is it possible that the minimal DRE is at least exponentially larger?

(B) Do there exist DRE-definable languages for which the smallest DRE is exponentially larger than the smallest RE?

In 1976, Ehrenfeucht und Zeiger [12] showed that translating a minimal DFA to a minimal RE can give rise to an exponential blow-up. They showed that for every $n \in \mathbb{N}$ there exists a language Z_n which has a minimal DFA of size $O(n^2)$ and a minimal RE with size $2^{\Theta(n-1)}$. However, these languages are not DRE-definable. Further work on descriptive complexity of regular expressions can be found in [19, 20]. We have already examined their methods and came to the conclusion that they cannot be applied for DRE-definable languages naively.

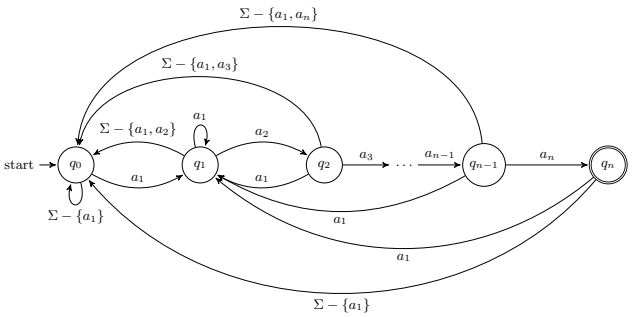

Figure 1: A minimal DFA A for $L(\Sigma^*a_1 \cdots a_n)$.

In [9], Brüggemann-Klein and Wood claimed that the blow-up when going from the minimal DFA for $L(\Sigma^*a_1 \cdots a_n)$ (see Figure 1) to a minimal DRE for this language can be exponential. Since the minimal DFA for $L(\Sigma^*a_1 \cdots a_n)$ has size $\Theta(n)$, this claim implies that both questions (A) and (B) would have a positive answer. However, to the best of our knowledge, no proof for this result exists in the literature and our investigations suggested that it is non-trivial. In my master thesis we prove this claim by showing that every minimal DRE for $L(\Sigma^*a_1 \cdots a_n)$ contains at least $2^{\Theta(n)}$ concatenations.

The main challenge behind the proof is that minimal DREs are not yet well-understood. The main idea of our proof is to examine how to identify a concatenation of a minimal DRE in the minimal DFA for the language. Thereby, one

has to search for states in the automaton through which every accepting run needs to go. We refer to such states as *bottleneck states* and prove that bottleneck states identify concatenations in a minimal DRE.

LEMMA 3.2. *Let $A = (\Sigma, \delta, q_0, F)$ be a minimal DFA for a DRE-definable regular language L and let q be a bottleneck state in A. Then every minimal DRE r for L is a concatenation $s.t$, such that $\delta(q_0, w) = q$ for every word $w \in L(s)$.*

By Lemma 3.2 we know that the state q_n in Figure 1 will cause a concatenation $s.t$ in every minimal DRE for the language. In a second step we examined the remaining minimal automata for $L(s)$ and $L(t)$. They have each only one state less, but we can prove that they contain the same recursive structure. This will cause an exponential blow-up of minimal DREs for $L(\Sigma^* a_1 \cdots a_n)$. Further this result implies the following:

COROLLARY 3.3. *Let L_1, L_2 be DRE-definable languages.*

- *For each $m \in \mathbb{N}$, there exist languages L_1 and L_2, such that they can be defined by DREs of size $O(m)$ but the minimal DRE for $L_1 \cdot L_2$ has size $2^{\Omega(m)}$.*
- *For each $m \in \mathbb{N}$, there exists a language L_1, such that L_1 can be defined by a DRE of size $O(m)$ but the minimal DRE for the reversal language of L_1 has size $2^{\Omega(m)}$.*

3.3 Boolean operations on DRE-definable languages

Next we give an overview of open questions regarding the descriptive complexity of boolean operations $(\cup, \cap, \setminus)$ on DRE-definable languages.

Therefore we review that DRE-definable languages are not closed under any boolean operation. It has been observed that they are not closed under union [9] or complement [19]. In my master thesis we established that they are also not closed under intersection in general [24].

LEMMA 3.4. *DRE-definable regular languages are not closed under intersection.*

In [10] it is proved that they are not closed under intersection even under a binary alphabet. This also implies that results on the descriptive complexity of regular languages do not simply carry over to DRE-definable languages.

To investigate the descriptive complexity of boolean operations on DRE-definable languages, we examine the next question regarding only the minimal DFAs, first.

(C) Let A_1, A_2 be minimal DFAs for DRE-definable languages. Is it possible that a minimal DFA for the union, intersection, or complement is at least exponentially larger and still describing a DRE-definable language?

Towards an answer of this question we start with studying results from the literature about descriptive complexity of subsets of regular languages which are less complex or in some way related to DRE-definable languages.

Since every finite language is DRE-definable [5], we can directly apply the knowledge on the descriptive complexity of boolean operations on finite languages. Although not every infinite regular language over a unary alphabet is DRE-definable, these class of regular languages is less complex

	$\lvert\Sigma\rvert = 1$		$\lvert\Sigma\rvert \geq 1$	
	1	n	1	n
$\setminus$	m [37]		m [37]	
$\cap$	$\min\{m_1, m_2\}$ [37]	$\min\{m_1, \ldots, m_n\}$	$O(m_1 m_2)$ [37]	exp
$\cup$	$\max\{m_1, m_2\}$ [37]	$\max\{m_1, \ldots, m_n\}$	$O(m_1 m_2)$ [37]	exp

Figure 2: Descriptive Complexity of Operations on Minimal DFAs for Finite Languages.

such that we can use the knowledge on these classes on DRE-definable languages, too.

Figure 2 gives an overview on the descriptive complexity of minimal DFAs describing finite languages. We summarize the results for a unary and an arbitrary alphabet where in each case we consider a single use of a boolean operation and an n-times application. Results given in O-notation mark where the exact bounds are still open. We simply list the results of Yu [37] on minimal DFAs for the single use application. Thereby follow the results for an n-times application on unary alphabets immediately.

The exponential upper bound for an n-times application of union or intersection on finite deterministic languages over an arbitrary alphabet follows directly by the standard product construction [22]. We can obtain a proof for the exponential lower bound, too.

For boolean operations on minimal DFAs for all DRE-definable regular languages we know the following.

For minimal DFAs it is well-known that we can obtain a minimal DFA for the complement by simply switching the accepting states to non-accepting states and vice versa. Therefore there is no blow-up for this operation [22].

For the intersection and union all upper bounds follow directly from the standard product construction. Thus for the intersection or union of two DRE-definable languages the upper bound is quadratic and for the n-times application it is exponential.

We now want to examine the lower bounds. For unary alphabets minimal automata for DRE-definable languages can only be of very restricted form, which alleviate the proofs. However we can show that for the intersection the lower bounds are still quadratic for two DRE-definable languages and exponential for the n-times application on DRE-definable languages. This holds even when the alphabet is unary.

The lower bounds for the union of two or n DRE-definable languages are, as far as we know, still open. This is due to the fact that it is not trivial in which cases the standard product automaton for the union of two DRE-definable languages is a DRE-definable language again. Admittedly more of theoretical point of view, knowing the lower bounds for this operation would be very interesting. Nevertheless, we hope that by finding these bounds we will gain more insights in the structure of DRE-definable regular languages.

4. THE COMPLEXITY OF EVALUATING PATH EXPRESSIONS IN SPARQL

In this section we give a brief introduction on RDF graphs and the SPARQL query language. Related work on SPARQL can be found in [1, 3, 33, 34, 35]. A large collection of SPARQL query examples can be found here [21]. We give some definitions first.

4.1 Regular expression with numerical occurrence indicators and graphs

For the rest of the paper, we consider a countably infinite alphabet Δ, which is due to the fact that RDF data models data on the web where the alphabet is not known beforehand. In SPARQL queries the following additional operators for regular expressions are allowed:

Numerical Occurrence Indicators: If $k \in \mathbb{N}$ and $\ell \in \mathbb{N}^+ \cup \infty$ with $k \leq \ell$, then $(r^{k,\ell})$ is a regular expression.

Limited Negation: If $a_1, \ldots, a_n \in \Delta$, then $!(a_1 + \cdots + a_n)$ is a regular expression.

Wildcard: The symbol $\bullet \notin \Delta$ is a regular expression.

For all operators appearing in Section 2, $L(r)$ is defined analogously for the infinite alphabet Δ. The newly introduced operators are inductively defined as follows:

$L(\bullet) = \Delta$;

$L(r^{k,\ell}) = \bigcup_{i=k}^{\ell} L(r)^i$; and,

$L(!(a_1 + \cdots + a_n)) = \Delta \backslash \{a_1, \ldots, a_n\}$.

We consider RDF-graphs as edge-labeled source-target-graphs. A graph G will be denoted as $G = (V, E, x, y)$, where V is the set of nodes of G and $E \subseteq V \times \Delta \times V$ is the set of edges. We denote x as *source node* and y as *target node*. A *path* from node x to node y in G is a sequence of edges going from x to y. We say that path p has *length* n if it consists of n edges. A path p *matches* a regular expression r if the concatenation of all edge-labels in p is in $L(r)$.

4.2 SPARQL Queries with Path Expressions

In [25] we were mainly interested in the following problems:

Evaluation: Given a graph (V, E, x, y) and a regular expression r, is there a path from x to y that matches r?

Finiteness: Given a graph (V, E, x, y) and a regular expression r, are there only finitely many different paths from x to y that match r?

Counting: Given a graph (V, E, x, y), a regular expression r and a natural number max in unary, how many different paths of length at most max between x and y match r?

First, the limited negation (!) and wildcard symbol ($\bullet$) do not significantly increase the complexity of SPARQL query evaluation. Numerical occurrence indicators in regular expression allow for exponentially shorter expressions [23], which might increase the evaluation complexity very well. Important work on regular expressions with numerical occurrence indicators can be found in [15, 18, 16, 23, 32]. Nevertheless membership testing for regular expressions with counting can be done in polynomial time [23] and we showed that Evaluation is in polynomial time, too [25]. Thus, in this case numerical occurrence indicators do not prevent efficient query evaluation.

The definition of the SPARQL property paths semantics on graphs, which is given in a non-standard manner, is more problematic. By this definition some subexpresssions of the property path expression have to be matched onto simple walks[1]. Basically subexpressions which can match onto infinitely many paths have to be matched onto simple walks, thereby preventing the result from being infinite itself. Unfortunately these semantics increases evaluation complexity significantly [3, 25].

At the moment the focus is on finding powerful semantics for SPARQL, where the query evaluation complexity is still manageable. In fact, a good way to improve the evaluation complexity of SPARQL may be the use of DREs, which we discussed in the first part of the paper. Currently they are used in XML schema language, because of their efficient processing qualities. For example, deterministic regular expressions can be translated to DFAs in polynomial time [9] and for DFAs Counting can be evaluated in polynomial time. In contrast, Counting for REs is #P-complete.

Further in SPARQL, as well as in XML Schema, numerical occurrence indicators are allowed. Therefore both applications will benefit from investigations on this class of regular expressions.

[1]A simple walk is a path that does not visit the same node twice, but is allowed to return to its first node.

5. CONCLUSION AND FURTHER WORK

The core motivation of our future work comes basically from two applications, namely XML schema languages and the RDF and SPARQL framework.

In XML schema languages regular expressions with numerical occurrence indicators are allowed. The expression itself is forced to be deterministic. This allows for processing deterministic regular expression more efficiently than general regular expressions.

In XML theory our main goal is on the optimization of patterns for XML schema languages. In our case this means we want to optimize deterministic regular expressions.

Therefore we want to examine the descriptive complexity of DREs and of boolean operations on DRE-definable languages, first. Although some results directly carry over from standard regular expressions, there are still some non trivial open cases.

Further we want to exhibit more insights in the structure of DRE-definable languages and their representation as deterministic regular expressions. As far as we know, the common proof methods for standard regular expression fail on their counterparts for deterministic regular expressions, such that new methods have to be developed.

Our second field of interest is SPARQL, a query language for RDF data on the web. Regarding the SPARQL query language we examine regular expressions with numerical occurrence indicators and investigate the evaluation of regular expressions on labelled graphs [25]. At the moment SPARQL semantics prevent efficient evaluation. Therefore we focus on finding powerful semantics for SPARQL property paths, where the query evaluation complexity is still manageable. Further we want to be able to provide efficient evaluation algorithms for expressions, since RDF graphs are on web-scale.

6. REFERENCES

[1] F. Alkhateeb, J.-F. Baget, and J. Euzenat. Extending SPARQL with regular expression patterns (for querying RDF). *J. Web Sem.*, 7(2):57–73, 2009.

[2] R. Angles and C. Gutierrez. Survey of graph database models. *ACM Comput. Surv.*, 40:1:1–1:39, 2008.

[3] M. Arenas, S. Conca, and J. Pérez. Counting beyond a yottabyte, or how SPARQL 1.1 property paths will prevent the adoption of the standard. In *World Wide Web Conference (WWW)*, 2012. To appear.

[4] M. Benedikt and C. Koch. XPath leashed. *ACM Comp. Surveys*, 41(1), 2008.

[5] G. J. Bex, W. Gelade, W. Martens, and F. Neven. Simplifying XML Schema: effortless handling of nondeterministic regular expressions. In *International Symposium on Management of Data (SIGMOD)*, pages 731–744, 2009.

[6] G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning deterministic regular expressions for the inference of schemas from XML data. In *World Wide Web Conference (WWW)*, pages 825–834, 2008.

[7] G. J. Bex, F. Neven, T. Schwentick, and S. Vansummeren. Inference of concise regular expressions and DTDs. *ACM Transactions on Database Systems*, 2010.

[8] G. J. Bex, F. Neven, and S. Vansummeren. Inferring XML Schema Definitions from XML data. In *International Conference on Very Large Data Bases (VLDB)*, pages 998–1009, 2007.

[9] A. Brüggemann-Klein and D. Wood. One-unambiguous regular languages. *Information and Computation*, 142(2):182–206, 1998.

[10] P. Caron, Y. Han, and L. Mignot. Generalized one-unambiguity. In *Developments in Language Theory*, volume 6795 of *Lecture Notes in Computer Science*, pages 129–140. Springer Berlin / Heidelberg, 2011.

[11] J. Clark and S. DeRose. XML Path Language (XPath) version 1.0. Technical report, World Wide Web Consortium, 1999. http://www.w3.org/TR/xpath/.

[12] A. Ehrenfeucht and H. Zeiger. Complexity measures for regular expressions. *Journal of Computer and System Sciences*, 12(2):134–146, 1976.

[13] M. F. Fernandez and D. Suciu. Optimizing regular path expressions using graph schemas. In *ICDE*, pages 14–23, 1998.

[14] D. Freydenberger and D. Reidenbach. Inferring descriptive generalisations of formal languages. In *COLT*, pages 194–206, 2010.

[15] W. Gelade. Succinctness of regular expressions with interleaving, intersection and counting. *Theoretical Computer Science*, 411(31–33):2987–2998, 2010.

[16] W. Gelade, M. Gyssens, and W. Martens. Regular expressions with counting: Weak versus strong determinism. *SIAM J. Comput.*, 41(1):160–190, 2012.

[17] W. Gelade, T. Idziaszek, W. Martens, and F. Neven. Simplifying XML Schema: Single-type approximations of regular tree languages. In *International Symposium on Principles of Database Systems (PODS)*, 2010.

[18] W. Gelade, W. Martens, and F. Neven. Optimizing schema languages for XML: Numerical constraints and interleaving. *SIAM J. Comput.*, 38(5), 2009.

[19] W. Gelade and F. Neven. Succinctness of the complement and intersection of regular expressions. In *Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 325–336, 2008.

[20] H. Gruber and M. Holzer. Finite automata, digraph connectivity, and regular expression size. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 39–50, 2008.

[21] S. Harris and A. Seaborne. SPARQL 1.1 query language. Technical report, World Wide Web Consortium (W3C), 2010.

[22] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 3 edition, 2007.

[23] P. Kilpeläinen and R. Tuhkanen. Regular expressions with numerical occurrence indicators — preliminary results. In *Symposium on Programming Languages and Software Tools (SPLST)*, pages 163–173, 2003.

[24] K. Losemann. Boolesche Operationen auf deterministischen regulären Ausdrücken. master thesis, TU Dortmund, October 2010.

[25] K. Losemann and W. Martens. The complexity of evaluating path expressions in SPARQL. In *PODS*, 2012.

[26] W. Martens. *Static analysis of XML transformation and schema languages*. PhD thesis, Hasselt University, 2006. Advisor: Frank Neven.

[27] W. Martens, F. Neven, and T. Schwentick. Simple off the shelf abstractions of XML Schema. *Sigmod RECORD*, 36(3):15–22, 2007.

[28] W. Martens, F. Neven, and T. Schwentick. Complexity of decision problems for XML schemas and chain regular expressions. *SIAM J. Comput.*, 39(4):1486–1530, 2009.

[29] W. Martens, F. Neven, T. Schwentick, and G.J. Bex. Expressiveness and complexity of XML Schema. *ACM Transactions on Database Systems*, 31(3):770–813, 2006.

[30] W. Martens and J. Niehren. On the minimization of XML Schemas and tree automata for unranked trees. *Journal of Computer and System Sciences*, 73(4):550–583, 2007.

[31] W. Martens, M. Niewerth, and T. Schwentick. Schema design for XML repositories: Complexity and tractability. In *International Symposium on Principles of Database Systems (PODS)*, 2010. To appear.

[32] A. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *IEEE Symposium on Foundations of Computer Science*, pages 125–129, 1972.

[33] J. Pérez, M. Arenas, and C. Gutierrez. Semantics and complexity of SPARQL. *ACM Trans. Database Syst.*, 34(3), 2009.

[34] J. Pérez, M. Arenas, and C. Gutierrez. nSPARQL: A navigational language for RDF. *J. Web Sem.*, 8(4):255–270, 2010.

[35] M. Schmidt, M. Meier, and G. Lausen. Foundations of SPARQL query optimization. In *Proceedings of the 13th International Conference on Database Theory*, ICDT '10, pages 4–33. ACM, 2010.

[36] C.M. Sperberg-McQueen and H. Thompson. XML Schema. http://www.w3.org/XML/Schema, 2005.

[37] S. Yu. State complexity of regular languages. *Journal of Automata, Languages and Combinatorics*, 6(2):221–, 2001.