Succinctness of the Complement and Intersection of Regular Expressions

WOUTER GELADE
Hasselt University and Transnational University of Limburg
and
FRANK NEVEN
Hasselt University and Transnational University of Limburg

We study the succinctness of the complement and intersection of regular expressions. In particular, we show that when constructing a regular expression defining the complement of a given regular expression, a double exponential size increase cannot be avoided. Similarly, when constructing a regular expression defining the intersection of a fixed and an arbitrary number of regular expressions, an exponential and double exponential size increase, respectively, can not be avoided. All mentioned lower bounds improve the existing ones by one exponential and are tight in the sense that the target expression can be constructed in the corresponding time class, i.e., exponential or double exponential time. As a by-product, we generalize a theorem by Ehrenfeucht and Zeiger stating that there is a class of DFAs which are exponentially more succinct than regular expressions, to a fixed alphabet. When the given regular expressions are one-unambiguous, as for instance required by the XML Schema specification, the complement can be computed in polynomial time whereas the bounds concerning intersection continue to hold. For the subclass of single-occurrence regular expressions, we prove a tight exponential lower bound for intersection.

Categories and Subject Descriptors: F.4.3 [**Theory of Computation**]: Mathematical Logic and Formal Languages—*Formal Languages*; I.7.2 [**Computing Methodologies**]: Document and Text Processing—*Document Preparation*; F.1.1 [**Theory of Computation**]: Computation by Abstract Devices—*Models of Computation*

General Terms: Languages, Theory

Additional Key Words and Phrases: Complexity, Regular expressions, Intersection, Complement, Succinctness, XML Schema languages

1. INTRODUCTION

The two central questions addressed in this paper are the following. Given regular expressions (REs) $r, r_1, \ldots, r_k$ over an alphabet Σ,

(1) what is the complexity of constructing a regular expression $r_\neg$ defining $\Sigma^* \setminus L(r)$, that is, the complement of r?

(2) what is the complexity of constructing a regular expression $r_\cap$ defining $L(r_1) \cap \cdots \cap L(r_k)$?

Wouter Gelade is a Research Assistant of the Fund for Scientific Research - Flanders (Belgium). We acknowledge the financial support of FWO-G.0821.09 and the Future and Emerging Technologies (FET) programme within the Seventh Framework Programme for Research of the European Commission, under the FET-Open grant agreement FOX, number FP7-ICT-233599.
An extended abstract of this paper appeared in the proceedings of the 25th International Symposium on Theoretical Aspects of Computer Science (STACS'08).

In both cases, the naive algorithm takes time double exponential in the size of the input. Indeed, for the complement, transform r to an NFA and determinize it (first exponential step), complement it and translate back to a regular expression (second exponential step). For the intersection there is a similar algorithm through a translation to NFAs, taking the crossproduct and a retranslation to a regular expression. Note that, in the worst case, both algorithms do not only take double exponential time but also result in a regular expression of double exponential size. In this paper, we exhibit classes of regular expressions for which this double exponential size increase cannot be avoided. Furthermore, when the number k of regular expressions is fixed, $r_\cap$ can be constructed in exponential time and we prove a matching lower bound for the size increase. In addition, we consider the fragments of one-unambiguous and single-occurrence regular expressions relevant to XML schema languages [Bex et al. 2010; Bex et al. 2007; Ghelli et al. 2007; Martens et al. 2006]. Our main results are summarized in Table I.

The main technical part of the paper is centered around the generalization of a result in [Ehrenfeucht and Zeiger 1976]. They exhibit a class of languages $(Z_n)_{n\in\mathbb{N}}$ each of which can be accepted by a DFA of size $\mathcal{O}(n^2)$ but cannot be defined by a regular expression of size smaller than 2^{n-1}. The most direct way to define Z_n is by the DFA that accepts it: the DFA is a graph consisting of n states, labeled 0 to $n-1$, which is fully connected and the edge between state i and j carries the label $a_{i,j}$. It now accepts all paths in the graph, that is, all strings of the form $a_{i_0,i_1}a_{i_1,i_2}\cdots a_{i_k,i_{k+1}}$. Note that the alphabet over which Z_n is defined grows quadratically with n. We generalize their result to a fixed alphabet. In particular, we define K_n as the binary encoding of Z_n using a suitable encoding for $a_{i,j}$ and prove that every regular expression defining K_n should be at least of size 2^n. As integers are encoded in binary the complement and intersection of regular expressions can now be used to separately encode K_{2^n} (and slight variations thereof) leading to the desired results. It should be noted that [Waizenegger 2000] already claimed a similar generalization using the straightforward binary encoding of Z_n. Unfortunately, we believe that a more sophisticated encoding as presented here is necessary, and hence the proof in [Waizenegger 2000] to be incorrect, as we discuss at the end of Section 3.

Although the succinctness of various automata models have been investigated in depth [Globerman and Harel 1996] and more recently those of logics over (unary alphabet) strings [Grohe and Schweikardt 2005], the succinctness of regular expressions had, up to recently, hardly been addressed. For the complement of a regular expression an exponential lower bound is given in [Ellul et al. 2005]. For the intersection of an arbitrary number of regular expressions Petersen gave an exponential lower bound [Petersen 2002], while [Ellul et al. 2005] mentions a quadratic lower bound for the intersection of two regular expressions. In fact, in [Ellul et al. 2005], it is explicitly asked what the maximum achievable blow-up is for the complement of one and the intersection of two regular expressions (Open Problems 4 and 5), and whether an exponential blow-up in the translation from DFA to RE is also unavoidable when the alphabet is fixed (Open Problem 3).

More recently, there have been a number of papers concerning succinctness of regular expressions and related matters [Gruber and Johannsen 2008; Gruber and Holzer 2008b; 2008a; Gelade 2010]. Most related is [Gruber and Holzer 2008a],

where, independently, a number of problems similar to the problems in this paper are studied. They also show that in constructing a RE for the intersection of two expressions, an exponential blow-up can not be avoided. However, we only give a $2^{\Omega(\sqrt{n})}$ lower bound whereas they obtain $2^{\Omega(n)}$, which in [Gruber and Holzer 2008b] is shown to be almost optimal. For the complementation of an RE we both obtain a double exponential lower bound. Here, however, they obtain $2^{2^{\Omega(\sqrt{n \log n})}}$ whereas we prove a (tight) $2^{2^{\Omega(n)}}$ lower bound. Finally, as a corollary of our results we obtain that in a translation from a DFA to an RE an exponential size increase can not be avoided, also when the alphabet is fixed. This yields a lower bound of $2^{\Omega(\sqrt{n/\log n})}$ which in [Gruber and Holzer 2008a] is improved to the (tight) bound of $2^{\Omega(n)}$. Together, these results settle open problems 3, 4, and 5 of [Ellul et al. 2005].

Succinctness of complement and intersection relate to the succinctness of semi-extended (RE($\cap$)) and extended regular expressions (RE($\cap$,$\neg$)). These are regular expressions augmented with intersection and both complement and intersection operators, respectively. Their membership problem has been extensively studied [Jiang and Ravikumar 1991; Kupferman and Zuhovitzky 2002; Myers 1992; Petersen 2002; Rosu and Viswanathan 2003; Rosu 2007]. Furthermore, non-emptiness and equivalence of RE($\cap$,$\neg$) is non-elementary [Stockmeyer and Meyer 1973]. For RE($\cap$), inequivalence is EXPSPACE-complete [Fürer 1980; Hunt III 1973; Robson 1979], and non-emptiness is PSPACE-complete [Fürer 1980; Hunt III 1973] even when restricted to the intersection of a (non-constant) number of regular expressions [Kozen 1977]. Several of these papers hint upon the succinctness of the intersection operator and provide dedicated techniques in dealing with the new operator directly rather than through a translation to ordinary regular expressions [Kupferman and Zuhovitzky 2002; Petersen 2002]. Our results present a double exponential lower bound in translating RE($\cap$) to RE and therefore justify even more the development for specialized techniques.

A final motivation for this research stems from its application in the emerging area of XML-theory [Libkin 2005; Neven 2002; Schwentick 2007; Vianu 2003]. From a formal language viewpoint, XML documents can be seen as labeled unranked trees and collections of these documents are defined by schemas. A schema can take various forms, but the most common ones are Document Type Definitions (DTDs) [Bray et al. 2004] and XML Schema Definitions (XSDs) [Sperberg-McQueen and Thompson 2005] which are grammar based formalisms with regular expressions at right-hand sides of rules [Martens et al. 2006; Murata et al. 2005]. Many questions concerning schemas reduce to corresponding questions on the classes of regular expressions used as right-hand sides of rules as is exemplified for the basic decision problems studied in [Gelade et al. 2009] and [Martens et al. 2009]. Furthermore, the lower bounds presented here are utilized in [Gelade and Neven 2007] to prove, among other things, lower bounds on the succinctness of existential and universal pattern-based schemas on the one hand, and single-type EDTDs (a formalization of XSDs) and DTDs, on the other hand. As the DTD and XML Schema specification require regular expressions occurring in rules to be *deterministic*, formalized by Brüggemann-Klein and Wood in terms of one-unambiguous regular expressions [Brüggemann-Klein and Wood 1998], we also investigate the complement

	complement	intersection (fixed)	intersection (arbitrary)
regular expression	2-exp	exp	2-exp
one-unambiguous	poly	exp	2-exp
single-occurrence	poly	exp	exp

Table I. Overview of the size increase for the various operators and subclasses. All non-polynomial complexities are tight.

and intersection of those. In particular, we show that a one-unambiguous regular expressions can be complemented in polynomial time, whereas the lower bounds concerning intersection carry over from unrestricted regular expressions. A study in [Bex et al. 2010] reveals that most of the one-unambiguous regular expression used in practice take a very simple form: every alphabet symbol occurs at most once. We refer to those as single-occurrence regular expressions (SOREs) and show a tight exponential lower bound for intersection.

Outline. In Section 2, we introduce the necessary definitions concerning (one-unambiguous) regular expressions and automata. In Section 3, we extend the result by Ehrenfeucht and Zeiger to a fixed alphabet using the family of languages $(K_n)_{n\in\mathbb{N}}$. In Section 4, we consider the succinctness of complement. In Section 5, we consider the succinctness of intersection of several classes of regular expressions. We conclude in Section 6.

2. PRELIMINARIES

2.1 Regular expressions

By $\mathbb{N}$ we denote the natural numbers without zero. For the rest of the paper, Σ always denotes a finite alphabet. A Σ-*string* (or simply string) is a finite sequence $w = a_1 \cdots a_n$ of Σ-symbols. We define the *length* of w, denoted by $|w|$, to be n. We denote the empty string by ε. By $w_1 \cdot w_2$ we denote the *concatenation* of two strings w_1 and w_2. As usual, for readability, we denote the concatenation of w_1 and w_2 by w_1w_2. The set of all strings is denoted by Σ^* and the set of all non-empty strings by Σ^+. A *string language* is a subset of Σ^*. For two string languages $L, L' \subseteq \Sigma^*$, we define their concatenation $L \cdot L'$ to be the set $\{w \cdot w' \mid w \in L, w' \in L'\}$. We abbreviate $L \cdot L \cdots L$ (i times) by L^i.

The set of *regular expressions* over Σ, denoted by RE, is defined in the usual way: $\emptyset$, ε, and every Σ-symbol is a regular expression; and when r_1 and r_2 are regular expressions, then $(r_1 \cdot r_2)$, $(r_1 + r_2)$, and (r_1^*) are also regular expressions.

By RE($\cap$,$\neg$) we denote the class of *extended regular expressions*, that is, RE extended with intersection and complementation operators. So, when r_1 and r_2 are RE($\cap$,$\neg$)-expressions then so are $(r_1 \cap r_2)$ and $(\neg r_1)$. By RE($\cap$) and RE($\neg$) we denote RE extended solely with the intersection and complement operator, respectively.

The language defined by an extended regular expression r, denoted by $L(r)$, is inductively defined as follows: $L(\emptyset) = \emptyset$; $L(\varepsilon) = \{\varepsilon\}$; $L(a) = \{a\}$; $L((r_1r_2)) = L(r_1) \cdot L(r_2)$; $L((r_1+r_2)) = L(r_1) \cup L(r_2)$; $L((r^*)) = \{\varepsilon\} \cup \bigcup_{i=1}^{\infty} L(r)^i$; $L((r_1 \cap r_2)) = L(r_1) \cap L(r_2)$; and $L((\neg r_1)) = \Sigma^* \setminus L(r_1)$.

By $\bigcup_{i=1}^{k} r_i$, and r^k, with $k \in \mathbb{N}$, we abbreviate the expression $r_1 + \cdots + r_k$, and $rr \cdots r$ (k-times), respectively. For a set $S = \{a_1, \ldots, a_n\} \subseteq \Sigma$, we abbreviate by S

the regular expression $a_1 + \cdots + a_n$.

We define the *size* of an extended regular expression r over Σ, denoted by $|r|$, as the number of Σ-symbols and operators occurring in r disregarding parentheses. This is equivalent to the length of its (parenthesis-free) reverse Polish form [Ziadi 1996]. Formally, $|\emptyset| = |\varepsilon| = |a| = 1$, for $a \in \Sigma$, $|(r_1 r_2)| = |(r_1 \cap r_2)| = |(r_1 + r_2)| = |r_1| + |r_2| + 1$, and $|(\neg r)| = |(r^*)| = |r| + 1$.

Other possibilities considered in the literature for defining the size of a regular expression are: (1) counting all symbols, operators, and parentheses [Aho et al. 1974; Ilie and Yu 2002]; or, (2) counting only the Σ-symbols [Gruber and Holzer 2008a]. However, it is known (see, for instance [Ellul et al. 2005]) that for regular expressions (so, without $\neg$ and $\cap$), provided they are preprocessed by syntactically eliminating superfluous $\emptyset$- and ε-symbols, and nested stars, the three length measures are identical up to a constant multiplicative factor. For extended regular expressions, counting only the Σ-symbols is not sufficient, since for instance the expression $(\neg\varepsilon)(\neg\varepsilon)(\neg\varepsilon)$ does not contain any Σ-symbols. Therefore, we define the size of an expression as the length of its reverse Polish form.

2.2 One-unambiguous regular expressions and SOREs

As mentioned in the introduction, several XML schema languages restrict regular expressions occurring in rules to be *deterministic*, formalized in terms of one-unambiguity [Brüggemann-Klein and Wood 1998]. We introduce this notion next.

To indicate different occurrences of the same symbol in a regular expression, we mark symbols with subscripts. For instance, the *marking* of $(a + b)^*a + bc$ is $(a_1 + b_2)^*a_3 + b_4c_5$. We denote by $r^\flat$ the marking of r and by $\mathrm{Sym}(r^\flat)$ the subscripted symbols occurring in $r^\flat$. When r is a marked expression, then $r^\natural$ over Σ is obtained from r by dropping all subscripts. This notion is extended to words and languages in the usual way.

Definition 1. A regular expression r is *one-unambiguous* if for all strings $w, u, v \in \mathrm{Sym}(r^\flat)^*$, and all symbols $x, y \in \mathrm{Sym}(r^\flat)$, the conditions $uxv, uyw \in L(r^\flat)$ and $x \neq y$ imply $x^\natural \neq y^\natural$.

For instance, the regular expression $r = a^*a$, with marking $r^\flat = a_1^*a_2$, is not one-unambiguous. Indeed, the marked strings a_1a_2 and $a_1a_1a_2$ both in $L(r^\flat)$ do not satisfy the conditions in the previous definition. The equivalent expression aa^*, however, is one-unambiguous. The intuition behind the definition is that positions in the input string can be matched in a deterministic way against a one-unambiguous regular expression without looking ahead. For instance, for the expression aa^*, the first a of an input string is always matched against the leading a in the expression, while every subsequent a is matched against the last a. Unfortunately, one-unambiguous regular languages do not form a very robust class as they are not even closed under complement or union [Brüggemann-Klein and Wood 1998].

The following subclass captures the class of regular expressions occurring in XML schemas on the Web [Bex et al. 2010]:

Definition 2. A *single-occurrence regular expression (SORE)* is a regular expression where every alphabet symbol occurs at most once.

For instance, $(a+b)^+c$ is a SORE while $a^*(a+b)^+$ is not. Clearly, every SORE is one-unambiguous. Note that SOREs define local languages and that over a fixed alphabet there are only finitely many of them.

2.3 Finite automata

A non-deterministic finite automaton (NFA) A is a 4-tuple (Q, q_0, δ, F) where Q is the set of states, q_0 is the initial state, F is the set of final states and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation. We write $q \Rightarrow_{A,w} q'$ when w takes A from state q to q'. So, w is accepted by A if $q_0 \Rightarrow_{A,w} q'$ for some $q' \in F$. The set of strings accepted by A is denoted by $L(A)$. The size of an NFA is $|\delta|$. An NFA is *deterministic* (or a DFA) if for all $a \in \Sigma, q \in Q$, $|\{(q, a, q') \in \delta \mid q' \in Q\}| \leq 1$.

We make use of the following known results.

THEOREM 3. *Let A be an NFA over Σ with m states, and let $|A| = n$ and $|\Sigma| = k$.*

(1) *A regular expression r, with $L(r) = L(A)$, can be constructed in time $\mathcal{O}(m^2 k 4^m)$ [McNaughton and Yamada 1960; Ellul et al. 2005].*
(2) *A DFA B with 2^m states, such that $L(B) = L(A)$, can be constructed in time $\mathcal{O}(2^n)$ [Yu 1997].*
(3) *A DFA B with 2^m states, such that $L(B) = \Sigma^* \setminus L(A)$, can be constructed in time $\mathcal{O}(2^n)$ [Yu 1997].*
(4) *Let $r \in RE$. An NFA B with $|r| + 1$ states, such that $L(B) = L(r)$, can be constructed in time $\mathcal{O}(|r|^2)$ [Brüggemann-Klein 1993].*
(5) *Let $r \in RE(\cap)$. An NFA B with $2^{|r|}$ states, such that $L(B) = L(r)$, can be constructed in time $\mathcal{O}(2^{\ell \cdot |r|})$, for some constant ℓ [Fürer 1980].*

3. A GENERALIZATION OF A THEOREM BY EHRENFEUCHT AND ZEIGER TO A FIXED ALPHABET

We first introduce the family $(Z_n)_{n \in \mathbb{N}}$ of string languages defined by Ehrenfeucht and Zeiger over an alphabet whose size grows quadratically with the parameter n [Ehrenfeucht and Zeiger 1976]:

DEFINITION 4. *Let $n \in \mathbb{N}$ and $\Sigma_n = \{a_{i,j} \mid 0 \leq i, j \leq n-1\}$. Then, Z_n contains exactly all strings of the form $a_{i_0,i_1} a_{i_1,i_2} \cdots a_{i_{k-1},i_k}$ where $k \in \mathbb{N}$.*

A way to interpret Z_n is to consider the DFA with states $\{0, \ldots, n-1\}$ which is fully connected and where the edge between state i and j is labeled with $a_{i,j}$. The language Z_n then consists of all paths in the DFA. [1]

Ehrenfeucht and Zeiger obtained the succinctness of DFAs with respect to regular expressions through the following theorem:

THEOREM [EHRENFEUCHT AND ZEIGER 1976]. *For $n \in \mathbb{N}$, any regular expression defining Z_n must be of size at least 2^{n-1}. Furthermore, there is a DFA of size $\mathcal{O}(n^2)$ accepting Z_n.*

[1] Actually, in [Ehrenfeucht and Zeiger 1976], only paths from state 0 to state $n-1$ are considered. We use our slightly modified definition as it will be easier to generalize to a fixed arity alphabet suited for our purpose in the sequel.

We will now define the language K_n as a binary encoding of Z_n over a four-letter alphabet, and generalize Theorem 5 to K_n.

The language K_n will be defined as the straightforward binary encoding of Z_n that additionally swaps the pair of indices in every symbol $a_{i,j}$. Thereto, for $a_{i,j} \in \Sigma_n$, define the function ρ_n as

$$\rho_n(a_{i,j}) = \text{enc}(j)\$\text{enc}(i)\#,$$

where $\text{enc}(i)$ and $\text{enc}(j)$ denote the $\lceil \log(n) \rceil$-bit binary encodings of i and j, respectively. Note that since $i, j < n$, i and j can be encoded using only $\lceil \log(n) \rceil$-bits. We extend the definition of ρ_n to strings in the usual way: $\rho_n(a_{i_0,i_1} \cdots a_{i_{k-1},i_k}) = \rho_n(a_{i_0,i_1}) \cdots \rho_n(a_{i_{k-1},i_k})$.

We are now ready to define K_n.

DEFINITION 6. *Let $\Sigma_K = \{0, 1, \$, \#\}$. For $n \in \mathbb{N}$, let $K_n = \{\rho_n(w) \mid w \in Z_n\}$.*

For instance, for $n = 5$, $w = a_{3,2}a_{2,1}a_{1,4}a_{4,2} \in Z_5$ and thus

$$\rho_n(w) = 010\$011\#001\$010\#100\$001\#010\$100\# \in K_5.$$

REMARK 7. *Although it might seem a bit artificial to swap the indices in the binary encoding of Z_n, it is definitely necessary. Indeed, consider the language K'_n obtained from Z_n like K_n but without swapping the indices of the symbols in Z_n. This language can be defined by a regular expressions of size $\mathcal{O}(n \log(n))$:*

$$\bigcup_{i<n} enc(i)\$\Big(\bigcup_{i<n} enc(i)\#enc(i)\$\Big)^* \bigcup_{i<n} enc(i)\#.$$

Therefore, K'_n is not suited to show exponential or double-exponential lower bounds on the size of regular expressions.

We now show that, by using the encoding that swaps the indices, it is possible to generalize Theorem 5 to a four-letter alphabet as follows:

THEOREM 8. *For any $n \in \mathbb{N}$, with $n \geq 2$,*

(1) there is a DFA A of size $\mathcal{O}(n^2 \log n)$ defining K_n; and

(2) any regular expression defining K_n is of size at least 2^n.

We first show (1). Let $n \in \mathbb{N}$ and $n \geq 2$. We compose A from a number of subautomata which each will be able to read one block of a string. That is, strings defined by the regular expression $(0+1)^{\lceil \log n \rceil}\$(0+1)^{\lceil \log n \rceil}$. These DFAs are:

—$B = (Q_B, q_B, \delta_B, \{f_0, \ldots, f_{n-1}\})$, with $L(B) = \{w\$w' \mid w, w' \in (0+1)^{\lceil \log n \rceil} \wedge 0 \leq \text{enc}(w') < n\}$, such that for any $j < n$, $q_B \Rightarrow_{B,w\$w'} f_j$ iff $\text{enc}(w) = j$. That is, B reads strings of the form $w\$w'$, checks whether w' encodes a number between 0 and $n-1$, and remembers the value of w in its accepting state.

—For any $i < n$, $B_i = (Q_i, q_i, \delta_i, \{f_{i,0}, \ldots, f_{i,n-1}\})$, with $L(B_i) = \{w\$w' \mid w, w' \in (0+1)^{\lceil \log n \rceil} \wedge \text{enc}(w') = i\}$, such that for any $j < n$, $q \Rightarrow_{B_i,w\$w'} f_{i,j}$ iff $\text{enc}(w) = j$. That is, B_i reads strings of the form $w\$w'$, checks whether w' encodes i and remembers the value of w in its accepting state.

The construction of B and each of the B_i is very similar. We illustrate the construction of B_2, for $n = 4$, in Figure 1.

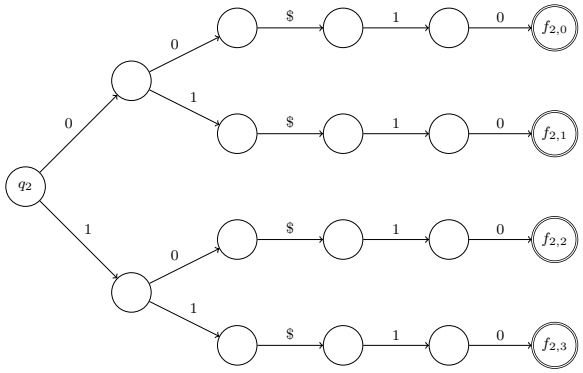

Fig. 1. The automaton B_2, for $n = 4$.

Now, let $A = (Q, q_B, \delta, \{q_0, \ldots, q_{n-1}\})$ where $Q = Q_B \cup \bigcup_{i<n} Q_i$ and δ contains $\delta_B \cup \bigcup_{i<n} \delta_i$ plus for every $i, j < n$, $(f_j, \#, q_j)$, $(f_{i,j}, \#, q_j) \in \delta$. So, A works as follows: first B reads the first block of the string and remembers the first integer therein, say j. Then it passes control to B_j which reads the next block in which it remembers the first integer, say k, and checks whether the second integer is j. If so, it passes control to B_k, and so on. Whenever A reaches an initial state of one of the B_i, a valid string has been read and A can accept.

Finally, for the size of A, consider the subautomata B_i as illustrated in Figure 1. The first, tree-like, part consists of at most $n + n/2 + n/4 + \cdots + 1$ nodes and transitions, which is bounded by $2n$. Further, the *linear* automata following this part are each of size $\lceil \log n \rceil$ and there are n of these. So, the total size of any B_i is $\mathcal{O}(n \log n)$. Further, B can be constructed similarly as the B_i. It has the same tree-like structure, of size $\mathcal{O}(n)$, as initial part. However, whereas the subsequent linear automata in the B_i each only have to read one encoded number, the subsequent automata in B have to check whether the number after the \$ is smaller then n. This, however, can again be done by such a tree-like automaton of size $\mathcal{O}(n)$. As B consists of n such parts, its size is $\mathcal{O}(n^2)$. Since A consists of one copy of B and a linear number of B_i subautomata, A is of size $\mathcal{O}(n^2 \log n)$. This concludes the proof of Theorem 8(1).

We now prove Theorem 8(2). It follows the structure of the proof of Ehrenfeucht and Zeiger but is technically more involved as it deals with binary encodings of integers.

We start by introducing some terminology. Fix $n \in \mathbb{N}$. We say that a language L *covers* a string w if there exist strings $u, u' \in \Sigma^*$ such that $uwu' \in L$. A regular expression r covers w when $L(r)$ covers w. Let $w = a_{i_0,i_1} a_{i_1,i_2} \cdots a_{i_{k-1},i_k} \in Z_n$. We say that i_0 is the *start-point* of w and i_k is its *end-point*. Furthermore, we say that w *contains* i or i *occurs in* w if i occurs as an index of some symbol in w. That is, $a_{i,j}$ or $a_{j,i}$ occurs in w for some j. For instance, $a_{0,2}a_{2,2}a_{2,1} \in Z_5$, has start-point 0, end-point 1, and contains 0, 1 and 2.

The notions of contains, occurs, start- and end-point of a string w are also extended to K_n. For $w \in Z_n$, the start- and end-points of $\rho_n(w)$ are the start and end-points of w. Hence, the start-point of $\rho_n(w)$ is the integer occurring between

the first \$ and # signs, and the notions of start- and end-points is only extended to strings of K_n. The notion of containing an integer, on the other hand, is extended to every string v which is covered by K_n. That is, v *contains* any integer encoded by $\lceil \log n \rceil$ consecutive bits in v. Clearly, for any $w \in Z_n$, $\rho_n(w)$ contains exactly the integers contained in w, but the notion is more general. For instance, for $n = 4$, the string 0\$01#11\$10 contains 1, 2, and 3 but its start- and end-point are undefined as it is not in K_n.

For a regular expression r, we say that i is a *sidekick* of r when it occurs in every non-empty string defined by r. A regular expression s is a starred subexpression of a regular expression r when s is a subexpression of r and is of the form t^*. We say that a regular expression r is *proper* if every starred subexpression of r has a sidekick.

LEMMA 9. *Any regular expression defining K_n is proper.*

PROOF. Let r be a regular expression defining K_n and s be a starred subexpression of r. We prove this lemma by a detailed examination of the structure of strings defined by s. We start by making the following observation. For any string $w \in L(s)$, there exist strings u, u' such that $uwu' \in L(r)$. Furthermore, w can be pumped in uwu' and still form a string defined by r. That is, for every $j \in \mathbb{N}$, $uw^j u' \in L(r)$. In addition, for every other $w' \in L(s)$, $uw'u' \in L(r)$.

Let w be a non-empty string in $L(s)$ and let u, u' be such that $uwu' \in L(r)$. Then w must contain at least one \$-symbol. Towards a contradiction, suppose it does not. If w contains a # then $uwwu' \in L(r)$ but $uwwu' \notin K_n$ which leads to the desired contradiction. If w contains no # and therefore only consists of 0's and 1's, then $uw^n u' \in L(r)$ but $uw^n u' \notin K_n$ which again is a contradiction. In a similar way, one can show that (i) w contains at least one #-symbol; (ii) w contains an equal number of \$ and #-symbols; (iii) the \$ and #-symbols must alternate; and (iv) between any consecutive \$ and #-symbol there is a string of length $\lceil \log(n) \rceil$ containing only 0's and 1's.

From the above it follows that w matches one of the following expressions:

(1) $\alpha_1 = (0+1)^*\$(0+1)^{\lceil \log n \rceil}\#\Sigma_K^*$
(2) $\alpha_2 = \Sigma_K^*\#(0+1)^{\lceil \log n \rceil}\$(0+1)^*$.

We refer to the strings defined by α_1 and α_2 as strings of type-1 and type-2, respectively. We next show that all strings defined by s are either all of type-1 or all of type-2. Towards a contradiction assume there is a type-1 string w_1 and a type-2 string w_2 in $L(s)$. Then, $w_2 w_1 \in L(s)$ and thus there exist $u, u' \in \Sigma_K^*$ such that $uw_2 w_1 u' \in L(r)$. However, because of the concatenation of w_2 and w_1, there are two \$-symbols without an intermediate #-symbol and therefore $uw_2 w_1 u' \notin K_n$.

Assume that all strings defined by s are of type-1. We next argue that the substring of length $\lceil \log n \rceil$, that is, the integer i, between the first \$ and #-symbol is the same for every $w \in L(s)$ which gives us our sidekick. For a type-1 string, we refer to this integer as the start block. Towards a contradiction, suppose that w, w_1 and w_2 are non-empty strings in $L(s)$ such that w_1 and w_2 have different start blocks. Let u, u' be such that $uww_1 u' \in L(r)$ and therefore $uww_1 u' \in K_n$. Now, uw contains at least one \$ and #-symbol. Therefore, by definition of K_n, the value of the start block of w_1 is uniquely determined by uw. That is, it must be equal

to the integer preceding the last \$-symbol in uw. Now also $uww_2u' \in L(r)$ as s is a starred subexpression, but $uww_2u' \notin K_n$ as w_2 has a different start block, which yields the desired contradiction.

The same kind of reasoning can be used to show that s has a sidekick when all defined strings are of type-2. □

If there is a greatest integer m for which r covers w^m, we call m the *index* of w in r and denote it by $I_w(r)$. In this case we say that r is *w-finite*. Otherwise, we say that r is *w-infinite*. The index of a regular expression can be used to give a lower bound on its size according to the following lemma.

LEMMA [EHRENFEUCHT AND ZEIGER 1976]. [2] *For any regular expression r and string w, if r is w-finite, then $I_w(r) < 2|r|$.*

Now, we can state the most important property of K_n.

LEMMA 11. *Let $n \geq 2$. For any $C \subseteq \{0, \ldots, n-1\}$ of cardinality k and $i \in C$, there exists a string $w \in K_n$ with start- and end-point i only containing integers in C, such that any proper regular expression r which covers w is of size at least 2^k.*

PROOF. The proof is by induction on k. For $k = 1$, $C = \{i\}$. Then, define $w = \text{enc}(i)\$\text{enc}(i)\#$, which satisfies all conditions and any expression covering w must definitely have size at least 2.

For the inductive step, let $C = \{j_1, \ldots, j_k\}$ and $i \in C$. Define $C_\ell = C \setminus \{j_{(\ell \mod k)+1}\}$ and let w_ℓ be the string given by the induction hypothesis with respect to C_ℓ (of size $k-1$) and j_ℓ. Note that $j_\ell \in C_\ell$. Further, define $m = 2^{k+1}$ and let w be

$$\text{enc}(j_1)\$\text{enc}(i)\#w_1^m\text{enc}(j_2)\$\text{enc}(j_1)\#w_2^m\text{enc}(j_3)\$\text{enc}(j_2)\#\cdots w_k^m\text{enc}(i)\$\text{enc}(j_k)\#.$$

Then, $w \in K_n$, has i as start and end-point and only contains integers in C. It only remains to show that any expression r which is proper and covers w is of size at least 2^k.

Fix such a regular expression r. If r is w_ℓ-finite, for some $\ell \leq k$, then $I_{w_\ell}(r_k) \geq m = 2^{k+1}$, by construction of w. By Lemma 10, $|r| \geq 2^k$ and we are done.

Therefore, assume that r is w_ℓ-infinite for every $\ell \leq k$. For every $\ell \leq k$, consider all subexpressions of r which are w_ℓ-infinite. We will see that all minimal elements in this set of subexpressions must be starred subexpressions. We say that an expression is minimal with respect to a set simply when no other expression in the set is a subexpression. Indeed, a subexpression of the form a or ε can never be w_ℓ-infinite and a subexpression of the form r_1r_2 or $r_1 + r_2$ can only be w_ℓ-infinite if r_1 and/or r_2 are w_ℓ-infinite and is thus not minimal with respect to w_ℓ-infinity. Hence, all minimal w_ℓ-infinite subexpressions are starred. Among these minimal starred subexpressions for w_ℓ, choose one and denote it by s_ℓ. Let $E = \{s_1, \ldots, s_k\}$. Note that since r is proper, all its subexpressions are also proper. As in addition each s_ℓ covers w_ℓ, by the induction hypothesis the size of each s_ℓ is at least 2^{k-1}.

[2] In fact, in [Ehrenfeucht and Zeiger 1976] the length of an expression is defined as the number of Σ-symbols occurring in it. However, since our length measure also contains these Σ-symbols, this lemma still holds in our setting.

Now, choose from E some expression s_ℓ such that s_ℓ is minimal with respect to the other elements in E.

As r is proper and s_ℓ is a starred subexpression of r, there is an integer j such that every non-empty string in $L(s_\ell)$ contains j. As, by the induction hypothesis, w_ℓ only contains integers in C, s_ℓ is w_ℓ-infinite, and s_ℓ is chosen to be minimal with respect to w_ℓ infinity, it follows that $j \in C = \{j_1, \ldots, j_k\}$ must hold. Let k' be such that $j = j_{k'}$. By definition of the inductively obtained strings $w_1, \ldots, w_k$, we have that for $p = k' - 1$ (if $k' \geq 2$, and $p = k$, otherwise), w_p does not contain j, because $j \notin C_p$. Denote by s_p the starred subexpression from E which is w_p-infinite. In particular, as j is a sidekick of s_ℓ, and s_p defines strings not containing j (recall that it is w_p-infinite, and minimal in this respect), s_ℓ and s_p cannot be the same subexpression of r.

Now, there are three possibilities:

—s_ℓ and s_p are completely disjoint subexpressions of r. That is, they are both not a subexpression of one another. By induction they must both be of size 2^{k-1} and thus $|r| \geq 2^{k-1} + 2^{k-1} = 2^k$.

—s_p is a strict subexpression of s_ℓ. This is not possible since s_ℓ is chosen to be a minimal element from E.

—s_ℓ is a strict subexpression of s_p. We show that if we replace s_ℓ by ε in s_p, then s_p is still w_p-infinite. It then follows that s_p still covers w_p, and thus s_p without s_ℓ is of size at least 2^{k-1}. As $|s_\ell| \geq 2^{k-1}$ as well it follows that $|r| \geq 2^k$.
To see that s_p without s_ℓ is still w_p-infinite, recall that any non-empty string defined by s_ℓ contains j and j does not occur in w_p. Therefore, a full iteration of s_ℓ can never contribute to the matching of any number of repetitions of w_p. Or, more specifically, no non-empty word $w \in L(s_\ell)$ can ever be a substring of a word w_p^i, for any i. So, s_p can only lose its w_p-infinity by this replacement if s_ℓ contains a subexpression which is itself w_p-infinite. However, this then also is a subexpression of s_p and s_p is chosen to be minimal with respect to w_p-infinity, a contradiction. We can only conclude that s_p without s_ℓ is still w_p-infinite.

□

Since by Lemma 9 any expression defining K_n is proper, Theorem 8(2) directly follows from Lemma 11 by choosing $i = 0$, $k = n$. This concludes the proof of Theorem 8(2).

It remains to discuss Waizenegger's proof of Theorem 8. He takes an approach similar to ours and defines a binary encoding of Z_n as follows. For $n \in \mathbb{N}$, Σ_n consists of n^2 symbols. If we list these, in some unspecified order, and associate a natural number from 0 to $n^2 - 1$ to each of them, we can encode Z_n using $\log(n^2)$ bits. For instance, if $n = 2$, then $\Sigma_n = \{a_{0,0}, a_{0,1}, a_{1,0}, a_{1,1}\}$ and we can encode these by the numbers $0, 1, 2, 3$, respectively. Now, a string in Z_n is encoded by replacing every symbol by its binary encoding and additionally adding an a and b to the start and end of each symbol. For instance, $a_{0,0}a_{0,1}$ becomes $a00ba01b$, which gives a language, say W_n, over the four letter alphabet $\{0, 1, a, b\}$.

Then, Waizenegger shows that W_n can be described by an NFA of size $\mathcal{O}(k^2 \cdot \log k^2)$, but every regular expression defining it must be of size at least exponential

in n. The latter is done by using the same proof techniques as in [Ehrenfeucht and Zeiger 1976]. However, in this proof it is claimed that if r_n is an expression defining W_n, then every string defined by a starred subexpression of r_n must start with an a. This statement is incorrect. For instance, if r_2 is an expression defining W_2, and we still use the same encoding as above, then $((a0(0ba0)^*0b) + \varepsilon)r_2$ still defines W_2 but does not have this property.[3] Albeit small, the mistake has serious consequences. It follows from this claim that every starred subexpression has a sidekick (using our terminology), and the rest of the proof builds upon this fact. To see the importance, consider the language K'_n obtained from Z_n like K_n but without swapping the indices of the symbols in Z_n. As illustrated in Remark 7, K'_n can be defined by a regular expression of size $\mathcal{O}(n \log(n))$. However, if we assume that every string defined by a starred subexpression starts with a #, we can reuse our proof for K_n and show that any regular expression defining K'_n must be of at least exponential size which is clearly false.

To summarize, Waizenegger's claim that the specific encoding of the n^2 symbols in Σ does not matter in the proof for Theorem 8 is false. Our encoding used in defining K_n allows to prove the sidekick lemma (Lemma 9) in a correct way.

4. COMPLEMENTING REGULAR EXPRESSIONS

It is known that extended regular expressions are non-elementary more succinct than classical ones [Dang 1973; Stockmeyer and Meyer 1973]. Intuitively, each exponent in the tower requires nesting of an additional complement. In this section, we show that in defining the complement of a single regular expression, a double exponential size increase cannot be avoided in general. In contrast, when the expression is one-unambiguous its complement can be computed in polynomial time.

THEOREM 12. *(1) For every regular expression r over Σ, a regular expression s with $L(s) = \Sigma^* \setminus L(r)$ can be constructed in time $\mathcal{O}(2^{2|r|+2} \cdot |\Sigma| \cdot 4^{2^{|r|+1}})$.*

(2) Let Σ be a four-letter alphabet. For every $n \in \mathbb{N}$, there is a regular expressions r_n of size $\mathcal{O}(n)$ such that any regular expression r defining $\Sigma^* \setminus L(r_n)$ is of size at least 2^{2^n}.

PROOF. (1) Let $r \in$ RE. We first construct a DFA A, with $L(A) = \Sigma^* \setminus L(r)$ and then construct the regular expression s equivalent to A. According to Theorem 3(3) and (4) A contains at most $2^{|r|+1}$ states and can be constructed in time exponential in the size of r. Then, by Theorem 3(1), the total algorithm is in time $\mathcal{O}(2^{2|r|+2} \cdot |\Sigma| \cdot 4^{2^{|r|+1}})$.

(2) Take Σ as Σ_K, that is, $\{0, 1, \$, \#\}$. Let $n \in \mathbb{N}$. We define an expression r'_n of size $\mathcal{O}(n)$, such that $\Sigma^* \setminus L(r'_n) = K_{2^n}$. By Theorem 8, any regular expression defining K_{2^n} is of size exponential in 2^n, that is, of size 2^{2^n}. This hence already proves that the theorem holds for languages over an alphabet of size four. Afterwards, we show that it also holds for alphabets of size two.

By $r^{[0,n-1]}$ we abbreviate the expression $(\varepsilon + r(\varepsilon + r(\varepsilon \cdots (\varepsilon + r))))$, with a nesting depth of $n-1$. We then define r'_n as the disjunction of the expressions below. Note

[3]For convenience, we assume here that all strings in W_n must have start point 0, an additional constraint which is actually present in [Ehrenfeucht and Zeiger 1976; Waizenegger 2000]

that, in these expressions, we only consider words which are not yet defined by one of the foregoing expressions. For instance, the last expression is only properly defined over words of the form $((0+1)^n\$(0+1)^n\#)^*$, as all other strings are already defined by the previous expressions.

—all strings that do not start with a prefix in $(0+1)^n\$$:

$$\Sigma^{[0,n]} + (0+1)^{[0,n-1]}(\$+\#)\Sigma^* + (0+1)^n(0+1+\#)\Sigma^*$$

—all strings where a \$ is not followed by a string in $(0+1)^n\#$:

$$\Sigma^*\$\big(\Sigma^{[0,n-1]}(\#+\$) + \Sigma^n(0+1+\$)\big)\Sigma^*$$

—all strings where a non-final # is not followed by a string in $(0+1)^n\$$:

$$\Sigma^*\#\big(\Sigma^{[0,n-1]}(\#+\$) + \Sigma^n(0+1+\#)\big)\Sigma^*$$

—all strings that do not end in #:

$$\Sigma^*(0+1+\$)$$

—all strings where the corresponding bits of corresponding blocks are different:

$$((0+1)^* + \Sigma^*\#(0+1)^*)0\Sigma^{3n+2}1\Sigma^* + ((0+1)^* + \Sigma^*\#(0+1)^*)1\Sigma^{3n+2}0\Sigma^*.$$

It should be clear that a string over $\{0,1,\$,\#\}$ is matched by none of the above expressions if and only if it belongs to K_{2^n}. So, the complement of r'_n defines exactly K_{2^n}.

□

The previous theorem essentially shows that in complementing a regular expression, there is no better algorithm than translating to a DFA, computing the complement and translating back to a regular expression which includes two exponential steps. However, when the given regular expression is one-unambiguous, a corresponding DFA can be computed in quadratic time through the Glushkov construction [Brüggemann-Klein and Wood 1998] eliminating already one exponential step. It should be noted that this construction, which we refer to as Glushkov construction, was introduced by Book et al. [Book et al. 1971] based on [Glushkov 1961] and [McNaughton and Yamada 1960], and often goes by the name position automata [Hromkovic et al. 2001]. As in the context of one-unambiguous expressions the name Glushkov construction is the most common, we will consistently use this naming.

The proof of the next theorem shows that the complement of the Glushkov automaton of a one-unambiguous expression can directly be defined by a regular expression of polynomial size.

THEOREM 13. *For any one-unambiguous regular expression r over an alphabet Σ, a regular expression s defining $\Sigma^* \setminus L(r)$ can be constructed in time $\mathcal{O}(n^3)$, where n is the size of r.*

PROOF. Let r be a one-unambiguous expression over Σ. We introduce some notation.

—The set Not-First(r) contains all Σ-symbols which are not the first symbol in any word defined by r, that is, Not-First$(r) = \Sigma \setminus \{a \mid a \in \Sigma \wedge \exists w \in \Sigma^*, aw \in L(r)\}$.

—For any symbol $x \in \mathrm{Sym}(r^\flat)$, the set Not-Follow$(r, x)$ contains all Σ-symbols of which no marked version can follow x in any word defined by $r^\flat$. That is, Not-Follow$(r, x) = \Sigma \setminus \{y^\natural \mid y \in \mathrm{Sym}(r^\flat) \wedge \exists w, w' \in \mathrm{Sym}(r^\flat)^*, wxyw' \in L(r^\flat)\}$.

—The set Last(r) contains all *marked* symbols which are the last symbol of some word defined by $r^\flat$. Formally, Last$(r) = \{x \mid x \in \mathrm{Sym}(r^\flat) \wedge \exists w \in \Sigma^*, wx \in L(r^\flat)\}$.

We define the following regular expressions:

—$\mathrm{init}(r) = \begin{cases} \text{Not-First}(r)\Sigma^* & \text{if } \varepsilon \in L(r); \text{ and} \\ \varepsilon + \text{Not-First}(r)\Sigma^* & \text{if } \varepsilon \notin L(r). \end{cases}$

—For every $x \in \mathrm{Sym}(r^\flat)$, $r^\flat_x$ will denote an expression defining $\{wx \mid w \in \mathrm{Sym}(r^\flat)^* \wedge \exists u \in \mathrm{Sym}(r^\flat)^*, wxu \in L(r^\flat)\}$. That is, all prefixes of strings in $r^\flat$ ending in x. Then, let r_x define $L(r^\flat_x)^\natural$.

We are now ready to define s:

$$\mathrm{init}(r) + \bigcup_{x \notin \mathrm{Last}(r)} r_x(\varepsilon + \text{Not-Follow}(r, x)\Sigma^*) + \bigcup_{x \in \mathrm{Last}(r)} r_x \text{Not-Follow}(r, x)\Sigma^*.$$

We conclude by showing that s can be constructed in time cubic in the size of r and that s defines the complement of r. We prove that $L(s) = \Sigma^* \setminus L(r)$ by highlighting the correspondence between s and the complement of the Glushkov automaton G_r of r. The Glushkov automaton G_r is the DFA (Q, q_0, δ, F), where

—$Q = \{q_0\} \cup \mathrm{Sym}(r^\flat)$;

—$F = \mathrm{Last}(r^\flat)$ (plus q_0 if $\varepsilon \in L(r)$); and

—for $x, y \in \mathrm{Sym}(r^\flat)$, there is

  —a transition $(q_0, x^\natural, x) \in \delta$ if x is the first symbol in some word defined by $r^\flat$; and,

  —a transition $(x, y^\natural, y) \in \delta$ if y follows x in some word defined by $r^\flat$.

It is known that $L(r) = L(G_r)$ and that G_r is deterministic whenever r is one-unambiguous [Brüggemann-Klein and Wood 1998].

The complement automaton $\overline{G_r} = (\overline{Q}, q_0, \overline{\delta}, \overline{F})$ is obtained from G_r by making it complete and interchanging final and non-final states. Formally, $\overline{Q} = Q \cup \{q_-\}$ (with $q_- \notin Q$) and $\overline{\delta}$ contains δ plus the triples (q_-, a, q_-), for every $a \in \Sigma$, and (q, a, q_-) for every state $q \in Q$ and symbol $a \in \Sigma$ for which there is no $q' \in Q$ with $(q, a, q') \in \delta$. Finally, $\overline{F} = \{q_-\} \cup (Q \setminus F)$. Clearly, $L(\overline{G_r}) = \Sigma^* \setminus L(G_r)$.

Now, we show that $L(s) = L(\overline{G_r})$. First, by definition of s, $\varepsilon \in L(s)$ iff $\varepsilon \notin L(r)$ iff $\varepsilon \in L(\overline{G_r})$. We prove that for any non-empty word w, $w \in L(s)$ iff $w \in L(\overline{G_r})$, from which the lemma follows. Thereto, we show that the non-empty words defined by the different disjuncts of s correspond exactly to subsets of the language $\overline{G_r}$.

—$\mathrm{init}(r)$ defines exactly the non-empty words for which $\overline{G_r}$ immediately goes from q_0 to q_- and reads the rest of the word while in q_-.

—For any $x \in \text{Last}(r)$, $r_x(\text{Not-Follow}(r,x)\Sigma^*)$ defines exactly all strings w for which $\overline{G_r}$ arrives in x after reading a part of w, goes to $q_\_$ and reads the rest of w there.

—For any $x \notin \text{Last}(r)$, $r_x(\varepsilon + \text{Not-Follow}(r,x)\Sigma^*)$ defines exactly all strings w for which $\overline{G_r}$ either (1) arrives in x after reading w and accepts because x is an accepting state; or (2) arrives in x after reading a part of w, goes to $q_\_$ and reads the rest of w there.

Note that because there are no incoming transitions in q_0, and $q_\_$ only has transitions to itself, we have described exactly all accepting runs of $\overline{G_r}$. Therefore, any non-empty string $w \in L(s)$ iff $w \in \overline{G_r}$.

We now show that s can be computed in time cubic in the size of r. By a result of Brüggemann-Klein [Brüggemann-Klein 1993] the Glushkov automaton G_r corresponding to r as defined above can be computed in time quadratic in the size of r. Using G_r, the sets Not-First(r), Not-Follow(r,x) and Last(r) can be computed in time linear in the size of r. So, all three sets can be computed in time quadratic in the size of r. The expression init(r) can be constructed in linear time. We next show that for any $x \in \text{Sym}(r^\flat)$, the expression $r^\flat_x$ can be constructed in time quadratic in the size of r. As $r_x = (r^\flat_x)^\natural$, it follows that s can be constructed in cubic time. The expression $r^\flat_x$ is inductively defined as follows:

—For $r^\flat = \varepsilon$ or $r^\flat = \emptyset$, $r^\flat_x = \emptyset$.

—For $r^\flat = y \in \text{Sym}(r^\flat)$,

$$r^\flat_x = \begin{cases} x & \text{if } y = x \\ \emptyset & \text{otherwise.} \end{cases}$$

—For $r^\flat = \alpha\beta$,

$$r^\flat_x = \begin{cases} \alpha_x & \text{if } x \text{ occurs in } \alpha \\ \alpha\beta_x & \text{otherwise.} \end{cases}$$

—For $r^\flat = \alpha + \beta$,

$$r^\flat_x = \begin{cases} \alpha_x & \text{if } x \text{ occurs in } \alpha \\ \beta_x & \text{otherwise.} \end{cases}$$

—For $r^\flat = \alpha^*$, $r_x = \alpha^*\alpha_x$

The correctness is easily proved by induction on the structure of $r^\flat$. Note that there are no ambiguities in the inductive definition of concatenation and disjunction since the expressions are marked and therefore every marked symbol occurs only once in the expression. For the time complexity, notice that all steps in the above inductive definition, except for the case $r^\flat = \alpha^*$, are linear. Further, for $r^\flat = \alpha^*$, $r_x = \alpha^*\alpha_x$, and hence α is doubled. However, as the inductive construction continues on only one of the two operands it follows that the complete construction is at most quadratic. □

We illustrate the construction in the previous proof by means of an example. Let $r = a(ab^*c)^*$, and $r^\flat = a_1(a_2b_3^*c_4)^*$. Then,

—$r^\flat_{a_1} = a_1$,

—$r^{\flat}_{a_2} = a_1(a_2b_3^*c_4)^*a_2$,
—$r^{\flat}_{b_3} = a_1(a_2b_3^*c_4)^*a_2b_3^*b_3$,
—$r^{\flat}_{c_4} = a_1(a_2b_3^*c_4)^*a_2b_3^*c_4$, and
—$\mathrm{init}(r) = \varepsilon + (b+c)\Sigma^*$.

Then the complement of r is defined by

$$\begin{aligned}\varepsilon + (b+c)\Sigma^* &\\ &+ a(ab^*c)^*a(\varepsilon + a\Sigma^*) + a(ab^*c)^*ab^*b(\varepsilon + a\Sigma^*)\\ &\qquad + a((b+c)\Sigma^*) + a(ab^*c)^*ab^*c((b+c)\Sigma^*).\end{aligned}$$

We conclude this section by remarking that one-unambiguous regular expressions are not closed under complement and that the constructed s is therefore not necessarily one-unambiguous.

5. INTERSECTING REGULAR EXPRESSIONS

In this section, we study the succinctness of intersection. In particular, we show that the intersection of two (or any fixed number) and an arbitrary number of regular expressions are exponentially and double exponentially more succinct than regular expressions, respectively. Actually, the exponential bound for a fixed number of expressions already holds for single-occurrence regular expressions, whereas the double exponential bound for an arbitrary number of expressions only carries over to one-unambiguous expressions. For single-occurrence expressions this can again be done in exponential time.

In this respect, we introduce a slightly altered version of K_n.

Definition 14. *Let $\Sigma_L = \{0, 1, \$, \#, \triangle\}$. For all $n \in \mathbb{N}$, $L_n = \{\rho_n(w)\triangle \mid w \in Z_n \wedge |w| \text{ is even}\}$.*

We also define a variant of Z_n which only slightly alters the $a_{i,j}$ symbols in Z_n. Thereto, let $\Sigma_n^\circ = \{a_{i^\circ,j}, a_{i,j^\circ} \mid 0 \leq i, j < n\}$ and set $\hat{\rho}(a_{i,j}a_{j,k}) = \triangleright_i a_{i,j^\circ}a_{j^\circ,k}$ and $\hat{\rho}(a_{i_0,i_1}a_{i_1,i_2}\cdots a_{i_{k-2},i_{k-1}}a_{i_{k-1},i_k}) = \hat{\rho}(a_{i_0,i_1}a_{i_1,i_2})\cdots\hat{\rho}(a_{i_{k-2},i_{k-1}}a_{i_{k-1},i_k})$, where k is even.

Definition 15. *Let $n \in \mathbb{N}$ and $\Sigma_M^n = \Sigma_n^\circ \cup \{\triangleright_0, \triangle_0, \ldots, \triangleright_{n-1}, \triangle_{n-2}\}$. Then, $M_n = \{\hat{\rho}(w)\triangle_i \mid w \in Z_n \wedge |w| \text{ is even} \wedge i \text{ is the end-point of } w\} \cup \{\triangle_i \mid 0 \leq i < n\}$.*

Note that words in M_n are those in Z_n where every odd position is promoted to a circled one ($^\circ$), and triangles labeled with the non-circled positions are added. For instance, the string $a_{2,4}a_{4,3}a_{3,3}a_{3,0} \in Z_5$ is mapped to the string $\triangleright_2 a_{2,4^\circ}a_{4^\circ,3} \triangleright_3 a_{3,3^\circ}a_{3^\circ,0}\triangle_0 \in M_5$.

We make use of the following property:

Lemma 16. *Let $n \in \mathbb{N}$.*

(1) Any regular expression defining L_n is of size at least 2^n.
(2) Any regular expression defining M_n is of size at least 2^{n-1}.

PROOF. (1) Analogous to Lemma 9, any regular expression r defining L_n must also be proper and must furthermore cover any string $w \in K_n$. By choosing $i = 0$ and $k = n$ in Lemma 11, we see that r must be of size at least 2^n.

(2) Let $n \in \mathbb{N}$ and r_M be a regular expression defining M_n. Let r_Z^2 be the regular expression obtained from r_M by replacing $\triangleright_i$ and $\triangle_i$, with $i < n$, by ε and any $a_{i^\circ,j}$ or a_{i,j°, with $0 \leq i, j < n$, by $a_{i,j}$. Then, $|r_Z^2| \leq |r_M|$ and r_Z^2 defines exactly all strings of even length in Z_n plus ε, and thus also covers every string in Z_n. Since the proof in [Ehrenfeucht and Zeiger 1976] also constructs a string $w \in Z_n$ such that any proper expression covering w must be of size at least 2^{n-1}, it immediately follows that r_Z^2 and thus r_M must be of size at least 2^{n-1}. □

The next theorem shows the succinctness of the intersection operator.

THEOREM 17. *(1) For any $k \in \mathbb{N}$ and regular expressions $r_1, \ldots, r_k$, a regular expression defining $\bigcap_{i \leq k} L(r_k)$ can be constructed in time $\mathcal{O}((m+1)^{2k} \cdot |\Sigma| \cdot 4^{(m+1)^k})$, where $m = \max\{|r_i| \mid 1 \leq i \leq k\}$.*

(2) For every $n \in \mathbb{N}$, there are SOREs r and s of size $\mathcal{O}(n^2)$ such that any regular expression defining $L(r) \cap L(s)$ is of size at least 2^{n-1}.

(3) For each $r \in RE(\cap)$ an equivalent regular expression can be constructed in time $\mathcal{O}(2^{2|r|} \cdot |\Sigma| \cdot 4^{2^{|r|}})$.

(4) For every $n \in \mathbb{N}$, there are one-unambiguous regular expressions $r_1, \ldots, r_m$, with $m = 2n + 1$, of size $\mathcal{O}(n)$ such that any regular expression defining $\bigcap_{i \leq m} L(r_i)$ is of size at least 2^{2^n}.

(5) Let $r_1, \ldots, r_n$ be SOREs. A regular expression defining $\bigcap_{i \leq n} L(r_n)$ can be constructed in time $\mathcal{O}(m^2 \cdot |\Sigma| \cdot 4^m)$, where $m = \sum_{i \leq n} |r_i|$.

PROOF. (1) First, construct NFAs $A_1, \ldots, A_k$ such that $L(A_i) = L(r_i)$, for any $i \leq k$. If $m = \max\{|r_i| \mid 1 \leq i \leq k\}$, then by Theorem 3(4) any A_i has at most $m+1$ states and can be constructed in time $\mathcal{O}(m^2)$. Then, an NFA A with $(m+1)^k$ states, such that $L(A) = \bigcap_{i \leq k} L(A_i)$, can be constructed in time $\mathcal{O}((m+1)^k)$ by means of a product construction. By Theorem 3(1), a regular expression defining $L(A)$ can then be constructed in time $\mathcal{O}((m+1)^{2k} \cdot |\Sigma| \cdot 4^{(m+1)^k})$.

(2) Let $n \in \mathbb{N}$. By Lemma 16(2), any regular expression defining M_n is of size at least 2^{n-1}. We define SOREs r and s of size quadratic in n, such that $L(r) \cap L(s) = M_n$. We start by partitioning Σ_M^n in two different ways. To this end, for every $i < n$, define $\mathrm{Out}_i = \{a_{i,j^\circ} \mid 0 \leq j < n\}$, $\mathrm{In}_i = \{a_{j^\circ,i} \mid 0 \leq j < n\}$, $\mathrm{Out}_{i^\circ} = \{a_{i^\circ,j} \mid 0 \leq j < n\}$, and $\mathrm{In}_{i^\circ} = \{a_{j,i^\circ} \mid 0 \leq j < n\}$. Then,

$$\Sigma_M^n = \bigcup_i \mathrm{In}_i \cup \mathrm{Out}_i \cup \{\triangleright_i, \triangle_i\} = \bigcup_{i^\circ} \mathrm{In}_{i^\circ} \cup \mathrm{Out}_{i^\circ} \cup \{\triangleright_i, \triangle_i\}.$$

Further, define

$$r = \Big((\triangleright_0 + \cdots + \triangleright_{n-1}) \bigcup_{i^\circ} \mathrm{In}_{i^\circ} \mathrm{Out}_{i^\circ}\Big)^* (\triangle_0 + \cdots + \triangle_{n-1})$$

and

$$s = \Big(\bigcup_i (\mathrm{In}_i + \varepsilon)(\triangleright_i + \triangle_i)(\mathrm{Out}_i + \varepsilon)\Big)^*.$$

Now, r checks that every string consists of a sequence of blocks of the form $\triangleright_i a_{j,k^\circ} a_{k^\circ,\ell}$, for $i, j, k, \ell < n$, ending with a $\triangle_i$, for $i < n$. It thus sets the format of the strings and checks whether the circled indices are equal. Further, s checks whether the non-circled indices are equal and whether the triangles have the correct indices. Since the alphabet of M_n is of size $\mathcal{O}(n^2)$, also r and s are of size $\mathcal{O}(n^2)$.

(3) For an expression r in RE($\cap$), an NFA A with $2^{|r|}$ states defining $L(r)$ can be constructed in time $\mathcal{O}(2^{\ell \cdot |r|})$ for some constant ℓ (Theorem 3(5)). Then, by Theorem 3(1), a regular expression (so, without $\neg$ and $\cap$) defining $L(A)$ can be constructed in time $\mathcal{O}(2^{2|r|} \cdot |\Sigma| \cdot 4^{2^{|r|}})$.

(4) Let $n \in \mathbb{N}$. We define $m = 2n + 1$ one-unambiguous regular expressions of size $\mathcal{O}(n)$, such that their intersection defines L_{2^n}. By Lemma 16(1), any regular expression defining L_{2^n} is of size at least 2^{2^n} and the theorem follows. For ease of readability, we denote Σ_L simply by Σ. The expressions are as follows.

—There should be an even length sequence of blocks:

$$\big((0+1)^n\$(0+1)^n\#(0+1)^n\$(0+1)^n\#\big)^*\triangle.$$

—For all $i \in \{0, \ldots, n-1\}$, the $(i+1)$th bit of the two numbers surrounding an odd # should be equal:

$$\big(\Sigma^i(0\Sigma^{3n+2}0 + 1\Sigma^{3n+2}1)\Sigma^{n-i-1}\#\big)^*\triangle.$$

—For all $i \in \{0, \ldots, n-1\}$, the $(i+1)$th bit of the two numbers surrounding an even # should be equal:

$$\Sigma^{2n+2}\Big(\Sigma^i(0\Sigma^{2n-i+1}(\triangle + \Sigma^{n+i+1}0\Sigma^{n-i-1}\#) + (1\Sigma^{2n-i+1}(\triangle + \Sigma^{n+i+1}1\Sigma^{n-i-1}\#)))\Big)^*.$$

Clearly, the intersection of the above expressions defines L_{2^n}. Furthermore, every expression is of size $\mathcal{O}(n)$ and is one-unambiguous as the Glushkov construction translates them into a DFA [Brüggemann-Klein and Wood 1998].

(5) We show that given SOREs $r_1, \ldots, r_n$, we can construct an NFA A with $|\Sigma| + 1$ states defining $\bigcap_{i \leq n} L(r_i)$ in time cubic in the sizes of $r_1, \ldots, r_n$. It then follows from Theorem 3(1) that an expression defining $\bigcap_{i \leq n} L(r_i)$ can be constructed in time $\mathcal{O}((m+1)^2 \cdot |\Sigma| \cdot 4^{(m+1)})$, where $m = \sum_{i \leq n} |r_i|$ since $m \geq |\Sigma|$.

We first construct NFAs $A_1, \ldots, A_n$ such that $L(A_i) = L(r_i)$, for any $i \leq n$, by using the Glushkov construction [Brüggemann-Klein 1993]. This construction creates an automaton which has an initial state, with only outgoing transitions, and additionally one state for each symbol in the regular expressions. Furthermore, all incoming transitions for that state are labeled with that symbol. So, we could also say that each state is labeled with a symbol, and that all incoming transitions carry the label of that state. Since $r_1, \ldots, r_n$ are SOREs, for every symbol there exists at most one state labeled with that symbol in any A_i. Now, let $A_i = (Q_i, q_0^i, \delta_i, F_i)$ then we say that $Q_i = \{q_0^i\} \cup \{q_a^i \mid a \in \Sigma\}$, where q_a^i is the state labeled with a. For ease of exposition, if a symbol a does not occur in an expression r_i, we add a state q_a^i to Q_i which does not have any incoming or outgoing transitions.

Now, we are ready to construct the NFA $A = (Q, q_0, \delta, F)$ defining the intersection of $A_1, \ldots, A_n$. First, Q has again an initial state and one state for each symbol: $Q = \{q_0\} \cup \{q_a \mid a \in \Sigma\}$. A state is accepting if all its corresponding states are accepting: $F = \{q_a \mid \forall i \leq n, q_a^i \in F_i\}$. Here, a can denote 0 or an alphabet symbol. Finally, there is a transition between q_a and q_b if there is a transition between q_a^i and q_b^i, in every A_i: $\delta = \{(q_a, b, q_b) \mid \forall i \leq n, (q_a^i, b, q_b^i) \in \delta^i\}$. Now, $L(A) = \bigcap_{i \leq n} L(A_i)$. Since the Glushkov construction takes quadratic time [Brüggemann-Klein 1993], and we have to construct n automata, the total construction can be done in cubic time. □

6. CONCLUSION

In this paper we showed that the complement and intersection of regular expressions are double exponentially more succinct than ordinary regular expressions. For complement, complexity can be reduced to polynomial for the class of one-unambiguous regular expressions although the obtained expressions could fall outside that class. For intersection, restriction to SOREs reduces complexity to exponential. It remains open whether there are natural classes of regular expressions for which both the complement and intersection can be computed in polynomial time.

ACKNOWLEDGMENT

We thank Juraj Hromkovič for sending us reference [Waizenegger 2000].

REFERENCES

Aho, A., Hopcroft, J., and Ullman, J. 1974. *The Design and Analysis of Computer Algorithms.* Addison-Wesley.

Bex, G. J., Neven, F., Schwentick, T., and Vansummeren, S. 2010. Inference of concise regular expressions and DTDs. *ACM Trans. Database Syst. 35,* 2.

Bex, G. J., Neven, F., and Vansummeren, S. 2007. Inferring XML Schema Definitions from XML data. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB'07)*. ACM, 998–1009.

Book, R., Even, S., Greibach, S., and Ott, G. 1971. Ambiguity in graphs and expressions. *IEEE Transactions in computers c-20,* 2, 149–153.

Bray, T., Paoli, J., Sperberg-McQueen, C., Maler, E., and Yergeau, F. 2004. Extensible Markup Language (XML). Tech. rep., World Wide Web Consortium. February. http://www.w3.org/TR/REC-xml/.

Brüggemann-Klein, A. 1993. Regular expressions into finite automata. *Theoretical Computer Science 120,* 2, 197–213.

Brüggemann-Klein, A. and Wood, D. 1998. One-unambiguous regular languages. *Information and Computation 142,* 2, 182–206.

Dang, Z. R. 1973. On the complexity of a finite automaton corresponding to a generalized regular expression. *Dokl. Akad. Nauk SSSR*.

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

Ellul, K., Krawetz, B., Shallit, J., and Wang, M. 2005. Regular expressions: New results and open problems. *Journal of Automata, Languages and Combinatorics 10,* 4, 407–437.

Fürer, M. 1980. The complexity of the inequivalence problem for regular expressions with intersection. In *Automata, Languages and Programming, 7th Colloquium (ICALP'80)*, J. W. de Bakker and J. van Leeuwen, Eds. Lecture Notes in Computer Science, vol. 85. Springer, 234–245.

Gelade, W. 2010. Succinctness of regular expressions with interleaving, intersection and counting. *Theor. Comput. Sci. 411,* 31-33, 2987–2998.

Gelade, W., Martens, W., and Neven, F. 2009. Optimizing schema languages for xml: Numerical constraints and interleaving. *SIAM J. Comput. 38,* 5, 2021–2043.

Gelade, W. and Neven, F. 2007. Succinctness of pattern-based schema languages for xml. In *DBPL*, M. Arenas and M. I. Schwartzbach, Eds. Lecture Notes in Computer Science, vol. 4797. Springer, 201–215. To appear in Journal of Computer and System Sciences.

Ghelli, G., Colazzo, D., and Sartiani, C. 2007. Efficient inclusion for a class of xml types with interleaving and counting. In *Database Programming Languages, 11th International Symposium (DBPL'07)*, M. Arenas and M. I. Schwartzbach, Eds. Lecture Notes in Computer Science, vol. 4797. Springer, 231–245.

Globerman, N. and Harel, D. 1996. Complexity results for two-way and multi-pebble automata and their logics. *Theoretical Computer Science 169,* 2, 161–184.

Glushkov, V. M. 1961. The abstract theory of automata. *UMN 16,* 5(101), 3–62.

Grohe, M. and Schweikardt, N. 2005. The succinctness of first-order logic on linear orders. *Logical Methods in Computer Science 1,* 1.

Gruber, H. and Holzer, M. 2008a. Finite automata, digraph connectivity, and regular expression size. In *35th International Colloquium on Automata, Languages and Programming (ICALP 2008)*, L. Aceto, I. Damgård, L. A. Goldberg, M. M. Halldórsson, A. Ingólfsdóttir, and I. Walukiewicz, Eds. Lecture Notes in Computer Science, vol. 5126. Springer, 39–50.

Gruber, H. and Holzer, M. 2008b. Provably shorter regular expressions from deterministic finite automata. In *Developments in Language Theory (DLT 2008)*, M. Ito and M. Toyama, Eds. Lecture Notes in Computer Science, vol. 5257. Springer, 383–395.

Gruber, H. and Johannsen, J. 2008. Optimal lower bounds on regular expression size using communication complexity. In *11th International Conference on Foundations of Software Science and Computational Structures (FoSSaCS 2008)*, R. M. Amadio, Ed. Lecture Notes in Computer Science, vol. 4962. Springer, 273–286.

Hromkovic, J., Seibert, S., and Wilke, T. 2001. Translating regular expressions into small epsilon-free nondeterministic finite automata. *J. Comput. Syst. Sci. 62,* 4, 565–588.

Hunt III, H. B. 1973. The equivalence problem for regular expressions with intersection is not polynomial in tape. Tech. rep., Department of Computer Science, Cornell University.

Ilie, L. and Yu, S. 2002. Algorithms for computing small NFAs. In *Mathematical Foundations of Computer Science, 27th International Symposium (MFCS'02)*, K. Diks and W. Rytter, Eds. Lecture Notes in Computer Science, vol. 2420. Springer, 328–340.

Jiang, T. and Ravikumar, B. 1991. A note on the space complexity of some decision problems for finite automata. *Information Processing Letters 40,* 1, 25–31.

Kozen, D. 1977. Lower bounds for natural proof systems. In *18th Annual Symposium on Foundations of Computer Science (FOCS'77)*. IEEE, 254–266.

Kupferman, O. and Zuhovitzky, S. 2002. An improved algorithm for the membership problem for extended regular expressions. In *Mathematical Foundations of Computer Science 2002, 27th International Symposium (MFCS'02)*. Lecture Notes in Computer Science, vol. 2420. Springer, 446–458.

Libkin, L. 2005. Logics for unranked trees: An overview. In *Automata, Languages and Programming, 32nd International Colloquium (ICALP'05)*, L. Caires, G. F. Italiano, L. Monteiro, C. Palamidessi, and M. Yung, Eds. Lecture Notes in Computer Science, vol. 3580. Springer, 35–50.

Martens, W., Neven, F., and Schwentick, T. 2009. Complexity of decision problems for xml schemas and chain regular expressions. *SIAM J. Comput. 39,* 4, 1486–1530.

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

McNaughton, R. and Yamada, H. 1960. Regular expressions and state graphs for automata. *IEEE Transactions on Electronic Computers 9,* 1, 39–47.

Murata, M., Lee, D., Mani, M., and Kawaguchi, K. 2005. Taxonomy of XML schema languages using formal language theory. *ACM Transactions on Internet Technologies 5,* 4, 660–704.

Myers, G. 1992. A four russians algorithm for regular expression pattern matching. *Journal of the ACM 39,* 2, 432–448.

Neven, F. 2002. Automata, logic, and XML. In *Computer Science Logic, 16th International Workshop (CSL'02)*, J. C. Bradfield, Ed. Lecture Notes in Computer Science, vol. 2471. Springer, 2–26.

Petersen, H. 2002. The membership problem for regular expressions with intersection is complete in LOGCFL. In *19th Annual Symposium on Theoretical Aspects of Computer Science (STACS'02)*, H. Alt and A. Ferreira, Eds. Lecture Notes in Computer Science, vol. 2285. Springer, 513–522.

Robson, J. M. 1979. The emptiness of complement problem for semi extended regular expressions requires c^n space. *Information Processing Letters 9,* 5, 220–222.

Rosu, G. 2007. An effective algorithm for the membership problem for extended regular expressions. In *Foundations of Software Science and Computational Structures, 10th International Conference (FOSSACS'07)*, H. Seidl, Ed. Lecture Notes in Computer Science, vol. 4423. Springer, 332–345.

Rosu, G. and Viswanathan, M. 2003. Testing extended regular language membership incrementally by rewriting. In *Rewriting Techniques and Applications, 14th International Conference (RTA'03)*, R. Nieuwenhuis, Ed. Lecture Notes in Computer Science, vol. 2706. Springer, 499–514.

Schwentick, T. 2007. Automata for XML - a survey. *Journal of Computer and System Sciences 73,* 3, 289–315.

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

Stockmeyer, L. J. and Meyer, A. R. 1973. Word problems requiring exponential time: Preliminary report. In *Conference Record of Fifth Annual ACM Symposium on Theory of Computing (STOC'73)*. ACM, 1–9.

Vianu, V. 2003. Logic as a query language: From frege to XML. In *20th Annual Symposium on Theoretical Aspects of Computer Science (STACS'03)*, H. Alt and M. Habib, Eds. Lecture Notes in Computer Science, vol. 2607. Springer, 1–12.

Waizenegger, V. 2000. Uber die Effizienz der Darstellung durch reguläre Ausdrücke und endliche Automaten. Diplomarbeit, RWTH Aachen.

Yu, S. 1997. Regular languages. In *Handbook of formal languages*, G. Rozenberg and A. Salomaa, Eds. Vol. 1. Springer, Chapter 2, 41–110.

Ziadi, D. 1996. Regular expression for a language without empty word. *Theoretical Computer Science 163,* 1&2, 309–315.

Received December 2008; revised March 2010; accepted November 2010