SIAM J. COMPUT.
Vol. 14, No. 3, August 1985

ON THE EQUIVALENCE AND CONTAINMENT PROBLEMS FOR UNAMBIGUOUS REGULAR EXPRESSIONS, REGULAR GRAMMARS AND FINITE AUTOMATA*

R. E. STEARNS† AND H. B. HUNT III†

Abstract. The known proofs that the equivalence and containment problems for regular expressions, regular grammars and nondeterministic finite automata are PSPACE-complete [SM] depend upon consideration of highly unambiguous expressions, grammars and automata. Here, we prove that such dependence is inherent.

Deterministic polynomial-time algorithms are presented for the equivalence and containment problems for unambiguous regular expressions, unambiguous regular grammars and unambiguous finite automata. The algorithms are then extended to ambiguity bounded by a fixed k. Our algorithms depend upon several elementary observations on the solutions of systems of homogeneous linear difference equations with constant coefficients and their relationship with the number of derivations of strings of a given length n by a regular grammar.

Key words. unambiguous regular expressions, regular grammars, finite automata, finite state transducers, equivalence and containment problems, homogeneous linear difference equations

1. Introduction.

The equivalence and containment problems for regular expressions, regular grammars and nondeterministic finite automata have been extensively studied in the technical literature. Both problems are known to be PSPACE-complete [SM], and thus are probably computationally intractable. Several possible ways to circumvent this intractability have been proposed in the literature. One way is to restrict attention to grammars, expressions and automata that only denote proper subfamilies of the regular sets [B], [Mc], [McP], [Z]. However in [H], we showed that the complexity of the equivalence and containment problems does *not* in general depend upon the structure of the languages denoted. A second way, proposed in [HRS], [H], is to place restrictions on the structure of the grammars, expressions and automata considered, rather than on the languages they denote.

In this paper we consider the structural restriction that the grammars, expressions and automata be *unambiguous*. (Informally, a language descriptor is unambiguous if each of the elements of the language it denotes can only be obtained in one way.) This restriction is very natural since strings in a language often have only one meaning, and one expects that the derivation of a string is attached to its meaning.

We present deterministic polynomial-time algorithms for the equivalence and containment problems for unambiguous regular expressions, regular grammars and nondeterministic finite automata. These algorithms are then generalized to deterministic polynomial-time algorithms for the equivalence and containment problems for regular expressions, regular grammars and nondeterministic finite automata of any *fixed* degree of ambiguity. Evidence is presented that our results are close to the best possible. For example, we show that the equivalence and containment problems for regular expressions, regular grammars and nondeterministic finite automata of bounded degree of ambiguity are CoNP-hard.

Section 2 presents the basic definitions, § 3 the relevant ideas from the theory of difference equations, and § 4 contains the basic results including the polynomial-time equivalence and containment algorithms.

* Received by the editors December 6, 1983 and in final form April 10, 1984.

† Computer Science Department, State University of New York at Albany, Albany, New York 12222.

2. Notation and definitions. We assume that the reader is familiar with the standard notation and terminology for regular expressions, regular grammars, deterministic and nondeterministic finite automata, the complexity classes P, NP and PSPACE, and the concepts of polynomial reducibility, NP-hard problems and PSPACE-hard problems. Otherwise see [AHU], [AU]. We denote the sets of natural numbers and real numbers by N and R, respectively.

DEFINITION 2.1. CoNP is the set of all languages over $\{0, 1\}$ that are complements of languages recognizable by nondeterministic polynomially time-bounded Turing machines. A language is CoNP-hard if every language in CoNP is polynomially reducible to it.

It is easily seen that a language L over an alphabet Σ is CoNP-hard if and only if its complement is NP-hard.

Since most of our results are presented in terms of finite automata, we give the definitions of the relevant concepts for them.

DEFINITION 2.2. A nondeterministic finite automaton $M = (S, I, \delta, s_1, F)$, where

(1) S is a finite nonempty set of *states*;

(2) I is a finite nonempty set of *input letters*;

(3) δ is a function from $S \times (I \cup \{\lambda\})$ into the power set of S;

(4) $s_1 \in S$ is the *start state*; and

(5) $F \subset S$ is the set of *accepting states.*

If there is $s \in S$ for which $\delta(s, \lambda)$ is a nonempty subset of S, then the automaton M is said to *have λ-transitions.* If the machine has no λ-transitions, then δ may be regarded as a function from $S \times I$ into the power set of S. If furthermore each $\delta(s, a)$ is a one element set, δ is regarded as a function from $S \times I$ into S and the automaton M is said to be a *deterministic finite automaton.*

The function δ is extended to the domain $S \times I^*$ in the standard manner. The *size of M*, denoted by $|M|$, is defined to equal $|S| \cdot |I|$.

Our definition of the size of M is a bit misleading since it represents the number of values of $\delta(s, a)$ that must be specified but not the space required for the actual specification. Our results hold as long as the size of M is bounded by some fixed polynomial function of $|S|$ and $|I|$.

DEFINITION 2.3. Let $M = (S, I, \delta, s_1, F)$ be a nondeterministic finite automaton. The *language accepted* by M, denoted by $L(M)$, is the set

$$\{w \in I^* \mid \delta(s_1, w) \cap F \neq \varnothing\}.$$

A string w is said to be *accepted by* the automaton M if and only if $w \in L(M)$.

DEFINITION 2.4. Let $M = (S, I, \delta, s_1, F)$ be a nondeterministic finite automaton. By a *state transition sequence* for M, we mean a finite nonempty sequence $\sigma = ((q_1, a_1), \cdots, (q_n, a_n), q_{n+1})$, where

(1) $(q_i, a_i) \in S \times (I \cup \{\lambda\})$ for $1 \leqq i \leqq n$;

(2) $q_{n+1} \in S$; and

(3) $q_{i+1} \in \delta(q_i, a_i)$ for $1 \leqq i \leqq n$.

The sequence σ is said to be a *state transition sequence for w*, where $w = a_1 \cdots a_n$, *that takes M from state q_1 to state q_{n+1}*. If $q_1 = s_1$ and $q_{n+1} \in F$, then the sequence σ is said to be an *accepting state transition sequence for w*. The *length* of the sequence σ, denoted by $|\sigma|$, equals n. (Thus if the automaton M does *not* have λ-transitions, then $|\sigma| = |w|$.)

Let $k \geqq 1$. If, for all $w \in L(M)$, there exist at most k accepting state transition sequences for w, then M is said to be *ambiguous of degree $\leqq k$.* If M is ambiguous of degree $\leqq 1$, then M is said to be *unambiguous.* If there exists $k \in N$ for which M is ambiguous of degree $\leqq k$, than M is said to have bounded degree of ambiguity.

DEFINITION 2.5. Let $M(=S, I, \delta, s_1, F)$ be a nondeterministic finite automaton. The function TRAN-SEQ$_M$ is the function from $S \times N$ to N defined by:

TRAN-SEQ$_M(s, k)$ for $s \in S$ and $k \in N$ equals the number of state transition sequences of length k which take state s into an accepting state.

The function ACC-SEQ $_M$ is the function from N to N defined by:

ACC-SEQ$_M(k)$ for $k \in N$ equals the number of accepting state transition sequences of length k.

DEFINITION 2.6. The *equivalence* and *containment problems* for a class C of regular expressions, regular grammars or nondeterministic finite automata are the problems of determining, given $M, N \in C$, if $L(M) = L(N)$ or $L(M) \subset L(N)$, respectively.

We also need the following elementary definition from the difference calculus.

DEFINITION 2.7. Let A be a function from N to R. We say that A *satisfies a homogeneous linear difference equation with constant coefficients of degree n* if and only if there exist constants $c_i \in R$, for $1 \leqq i \leqq n$, with $c_n \neq 0$ such that

$$\sum_{i=0}^{n} c_i \cdot A(k+i) = 0 \quad \text{for all } k \geqq 0.$$

Henceforth, we abbreviate "homogeneous linear difference equations with constant coefficients" by "difference equation".

Analogues of the concepts of unambiguity, ambiguity of degree $\leqq k$, and bounded degree of ambiguity can be defined for regular expressions and for the regular grammars. There exist well-known deterministic polynomial-time algorithms for converting a regular expression or a regular grammar into a nondeterministic finite automaton that accepts the same language [AU], [AHU]. These algorithms preserve the properties of unambiguity, ambiguity of degree $\leqq k$ and bounded degree of ambiguity. Hence, we omit further discussion of these facts and present our results in terms of nondeterministic finite automata.

3. Lemmas on difference equations. We present three elementary lemmas on difference equations. In §§ 4 and 5 we use these lemmas to prove the correctness of our deterministic polynomial-time algorithms for equivalence and containment problems.

LEMMA 3.1. *Let S be a nonempty finite set. For all $s \in S$, let A_s be a function from N to R such that*

$$A_s(k+1) = \sum_{t \in S} d^1_{s,t} \cdot A_t(k) \quad \textit{for all } k \geqq 0, \tag{3.1}$$

where $d^1_{s,t}$ is a real constant for all $s, t \in S$. Then for all $s \in S$, the function A_s satisfies a difference equation of degree $\leqq |S|$.

Proof. We show how to derive the difference equation for one particular A_s. The first step is to obtain $|S|$ difference equations of the form

$$A_s(k+j) = \sum_{t \in S} d^j_{s,t} \cdot A_t(k) \quad \text{for all } k \geqq 0$$

one equation for each value of j between 1 and $|S|$.

The equation for $j = 1$ is simply the corresponding equation from set (3.1) (i.e., the equation whose left-hand side is the particular A_s under consideration).

The equations for subsequent j are obtained inductively. Given the equation for $A_s(k+j)$, replace k by $k+1$ to obtain

$$A_s(k+j+1) = \sum_{t \in S} d^j_{s,t} \cdot A_t(k+1).$$

Then, replace each $A_t(k+1)$ using (3.1). The resulting equation is

$$A_s(k+j+1)=\sum_{t\in S} d_{s,t}^{j+1}\cdot A_t(k)$$

where

$$d_{s,t}^{j+1}=\sum_{u\in S} c_{s,u}\cdot d_{u,t}^{j}.$$

We thus have the desired equation for $j+1$.

The derived $|S|$ equations relate the quantities $A_s(k+1),\cdots$, and $A_s(k+|S|)$ with the $|S|$ quantities $A_t(k)$ for $t\in S$. Standard elimination techniques for systems of linear equations can be used to eliminate the quantities $A_t(k)$ for $t\in S-\{s\}$. This is because there are $|S|$ equations and only $|S|-1$ quantities to be eliminated. The resulting difference equation is the equation whose existence is asserted in the statement of the lemma. □

The construction in the proof of Lemma 3.1 is a standard construction (see any text on difference equations). The resulting equation can be used to obtain a "closed form" formula for $A_s(k)$. Since this formula has constants which are roots of a polynomial equation, the formula can not be considered "closed" from a computational point of view. However, we use the existence of the difference equations only for proofs and not for computation.

LEMMA 3.2. *Let A and B be functions from N to R such that A and B satisfy difference equations of degrees a and b, respectively. Then the function D from N to R defined by, for all $k\in N$, $D(k)=A(k)-B(k)$, satisfies a difference equation of degree $\leqq a+b$. Hence, if for $0\leqq k\leqq a+b-1$, $A(k)=B(k)$, then $A(k)=B(k)$ for all $k\in N$.*

Proof. We first show that the function D satisfies a difference equation of degree $\leqq a+b$.

By definition for all $k\geqq 0$, the function D satisfies the equation

$$D(k+j)=A(k+j)-B(k+j)\quad \text{for } 0\leqq j\leqq a+b.$$

Moreover the difference equations satisfied by A and by B allow;

(1) the replacement of any $A(k+j)$ by a linear combination of $A(k+i)$ for $0\leqq i\leqq j$ provided $j\geqq a$; and

(2) the replacement of any $B(k+j)$ by a linear combination of $B(k+i)$ for $0\leqq i\leqq j$ provided $j\geqq b$. Hence, by repeated applications of [1] and of [2] for all $k\geqq 0$,

$$D(k+j)=\sum_{i=0}^{a-1} a_i^j\cdot A(k+i)+\sum_{i=0}^{b-1} b_i^j\cdot B(k+i)$$

for all $j\geqq 0$.

As in the proof of Lemma 3.1, elimination of the $a+b$ quantities $A(k+i)$ for $0\leqq i\leqq a-1$ and $B(k+i)$ for $0\leqq i\leqq b-1$ from the $a+b+1$ equations for $D(k),\cdots$, and $D(k+a+b)$ yields a difference equation of degree $\leqq a+b$ relating $D(k),\cdots$, and $D(k+a+b)$. This equation is the equation whose existence is affirmed in the statement of the lemma.

Now consider the last statement in the lemma. If $A(k)=B(k)$ for $0\leqq k\leqq a+b-1$, then $D(k)=0$ over this range. But since $D(k)$ satisfies an equation of degree $\leqq a+b$, all subsequent values of $D(k)$ must be zero and $A(k)-B(k)$ is zero for all k. □

LEMMA 3.3. *Let $k\geqq 1$. Let $A_1,\cdots$, and A_k be functions from N to R such that each function A_i satisfies a difference equation of degree a_i. Let $c_1,\cdots$, and c_k be elements of R. Then the function A from N to R defined by $A(n)=\sum_{i=1}^{k} c_i\cdot A_i(n)$ for all $n\in N$, satisfies a difference equation of degree $\leqq\sum_{i=1}^{k} a_i$.*

Proof. The proof is a straightforward extension of the techniques used in the proof of Lemma 3.2. □

4. Unambiguous regular descriptions.

4. Unambiguous regular descriptions. We use the observations about systems of difference equations in § 3 to derive deterministic polynomially time-bounded algorithms for the equivalence and containment problems for unambiguous regular expressions, regular grammars and finite automata. For automaton M, let $\mathrm{ACC}_M\ (k)$ represent the number of strings of length k accepted by M. The correctness of our algorithm is based upon the following fact:

Let M be an unambiguous finite automaton with start state s_0 and with *no* λ-transitions. Then for all $k \geqq 0$,

$$\mathrm{ACC}_M\ (k) = \text{ACC-SEQ}_M\ (k).$$

Our algorithms are derived and their correctness is proven by a sequence of lemmas. The techniques apply directly to automata without λ-transitions. However, we start with more general automata because they are the output of the standard procedures which produce automata from expressions and grammars. The first lemma in the sequence allows us to consider just the case without λ-transitions.

LEMMA 4.1. *There exists a deterministic polynomially time-bounded algorithm that takes an unambiguous finite automaton $M = (S, I, \delta, s_0, F)$ as input and outputs an equivalent unambiguous finite automaton M' with no λ-transitions and with at most $|S|$ states.*

Proof. The existence of such a deterministic polynomially time-bounded algorithm is implied by the following:

(1) Let $\rightarrow_M$ be the binary relation on S defined by, $s \rightarrow_M t$ for $s, t \in S$ if and only if $t \in \delta(s, \lambda)$. Let $\rightarrow_M^*$ be the transitive reflexive closure of $\rightarrow_M$. Relations $\rightarrow_M$ and $\rightarrow_M^*$ are computable deterministically from M in polynomial time.

(2) Let M' be the nondeterministic finite automaton (S, I, δ', s_0, F'), where δ' is defined by, for all $s \in S$ and $a \in I$, $\delta'(s, a) = \{t'' \in S \mid \exists t' \in S$ for which $s \rightarrow_M^* t'$ and $t'' \in \delta(t', a)\}$, and $F' = \{s \in S \mid \exists t \in F$ for which $s \rightarrow_M^* t\}$. Machine M' is unambiguous, has *no* λ-transitions, and is equivalent to M. □

LEMMA 4.2. *Let $M = (S, I, \delta, s_0, F)$ be a nondeterministic finite automaton with no λ-transitions. Let $s \in S$. Let $k \in N$. Then*

$$\text{TRAN-SEQ}_M(s, k+1) = \sum_{t \in S} a_{s,t} \cdot \text{TRAN-SEQ}_M\ (t, k), \quad \textit{where } a_{s,t} = |\{a \in I \mid t \in \delta(s, a)\}|.$$

Proof. The proof is immediate from the definition of an accepting state transition sequence, since by assumption the automaton M has *no* λ-transitions. □

LEMMA 4.3. *Let $n \in N$. For any nondeterministic n state finite automaton M with no λ-transitions, the function* ACC-SEQ$_M$ *satisfies a difference equation of degree $\leqq n$.*

Proof. The equations in the statement of Lemma 4.2 are of the form of Lemma 3.1 if we let the set S be the set of M's states. For all $t \in S$, let $A_t(k) = \text{TRAN-SEQ}_M\ (t, k)$. Letting s_0 be the start state of M, $A_{s_0}(k) = \text{ACC-SEQ}_M\ (k)$. □

Lemma 4.3 is illustrated by Example 4.10. The reader may want to study this example before continuing.

LEMMA 4.4. *There exists a deterministic polynomially time-bounded algorithm that, given as input a nondeterministic finite automaton $M = (S, I, \delta, s_0, F)$ with no λ-transitions and a string 1^n, gives as output the first n values of* ACC-SEQ$_M$.

Proof. The obvious way to compute the first n values of ACC-SEQ$_M$ is:

(1) to first compute the values of TRAN-SEQ$_M$ $(s, 0)$ for all $s \in S$; and then,

(2) to perform an iteration n times in which the values of TRAN-SEQ$_M$ $(s, i+1)$

are computed from the values of TRAN-SEQ$_M$ (s, i) for $s, t \in S$ by means of Lemma 4.2. For each $s \in S$,

$$\text{TRAN-SEQ}_M\ (s, 0) = \begin{cases} 1 & \text{if } s \in F, \\ 0 & \text{otherwise.} \end{cases}$$

The coefficients $a_{s,t}$ from Lemma 4.2 have size at most $|I|$ and the sum has $|S|$ terms. Since the TRAN-SEQ$_M$ $(s, 0)$ are at most 1, it follows by induction that the TRAN-SEQ$_M$ (s, k) are at most $(|I| \cdot |S|)^k$. Thus all values computed during the algorithm are positive integers whose length in binary is at most $n \cdot \log_2 (|I| \cdot |S|)$. Thus, all multiplications and additions can be accomplished in time polynomial in $|M|$ and n. Since the number of operations executed during the computation is also bounded by a polynomial in $|M|$ and n, the computation can be accomplished deterministically in time polynomial in $|M|$ and n. □

LEMMA 4.5. *There exists a deterministic polynomially time-bounded algorithm that, given as input an n_1 state unambiguous finite automaton M_1 and an n_2 state unambiguous finite automaton M_2 both with no λ-transitions, gives as output an $n_1 \cdot n_2$ state unambiguous finite automaton M with no λ-transitions such that*

$$L(M) = L(M_1) \cap L(M_2).$$

Proof. Letting $M_1 = (S_1, I_1, \delta_1, s_1, F_1)$ and $M_2 = (S_2, I_2, \delta_2, s_2, F_2)$, construct

$$M = (S, I, \delta, s, F) \quad \text{where } S = S_1 \times S_2, \quad I = I_1 \cup I_2,$$

$\delta((t_1, t_2), a) = (\delta_1(t_1, a), \delta_2(t_2, a))$ for $a \in I_1 \cap I_2$, $s = (s_1, s_2)$, and $F = F_1 \times F_2$.

The accepting state transition sequences for this machine are precisely those that project into accepting state transition sequences for M_1 and M_2, so the number of ways M accepts a string w is the product of the ways M_1 accepts w and M_2 accepts w. Because M_1 and M_2 are unambiguous, they each accept a string in either zero or one ways and so M accepts in one way if both M_1 and M_2 accept and M does not accept otherwise. Thus M is unambiguous and $L(M) = L(M_1) \cap L(M_2)$. □

THEOREM 4.6. *There exists a deterministic polynomially time-bounded algorithm that, given as input an n_1 state unambiguous finite automaton M_1 and an n_2 state unambiguous finite automaton M_2 such that $L(M_1) \subset L(M_2)$, decides if the set containment is proper.*

Proof. By Lemma 4.1 the automata M_1 and M_2 can be converted deterministically in polynomial-time into equivalent unambiguous finite automata M_1' and M_2' with *no* λ-transitions such that M_1' is also an n_1 state automaton and M_2' is also an n_2 state automaton.

Since $L(M_1) \subset L(M_2)$ by assumption, the set containment is proper if and only if ACC-SEQ$_{M_1'}$ $(k) \neq$ ACC-SEQ$_{M_2'}$ (k) for some k. By Lemma 4.3 the functions ACC-SEQ$_{M_1'}$ and ACC-SEQ$_{M_2'}$ satisfy difference equations of degrees n_1 and n_2, respectively. Thus by Lemma 3.2 the functions ACC-SEQ$_{M_1'}$ and ACC-SEQ$_{M_2'}$ are equal if and only if, for all natural $k < n_1 + n_2$, ACC-SEQ$_{M_1'}$ $(k) =$ ACC-SEQ$_{M_2'}$ (k). By Lemma 4.4 this can be checked deterministically in polynomial-time. □

COROLLARY 4.7. *The equivalence and containment problems for unambiguous finite automata are decidable deterministically in polynomial-time.*

Proof. The corollary follows immediately from Lemma 4.5 and Theorem 4.6, since for finite automata M_1 and M_2, $L(M_1) \subset L(M_2)$ if and only if the language $L(M_1) \cap L(M_2)$ is *not* properly contained in the language $L(M_1)$. □

This corollary combines with a result of Gurari and Ibarra [GI] to give the following result: There is a deterministic polynomially time-bounded algorithm which decides if two single-valued unambiguous finite state transducers are equivalent.

Finally, let M_1 and M_2 be unambiguous finite automata such that $L(M_2) - L(M_1) \neq \emptyset$. It is natural to ask the question:

Can a string $w \in L(M_2) - L(M_1)$ be found deterministically in time polynomial in $|M_1| + |M_2|$?

The answer to this question is "yes" as shown by the next two theorems.

THEOREM 4.8. *There exists a deterministic polynomial-time algorithm that, given as input an n_1 state unambiguous finite automaton M_1 and an n_2 state unambiguous finite automaton M_2 such that $L(M_1)$ is properly contained in $L(M_2)$, gives as output a string $w \in L(M_2) - L(M_1)$ of minimal length such that $|w| < n_1 + n_2$.*

Proof. As in the proof of Theorem 4.6, we can assume that the automata M_1 and M_2 have *no* λ-transitions. As shown in the proof of Theorem 4.6, the language $L(M_1)$ is properly contained in the language $L(M_2)$ if and only if there exists a string $w \in L(M_2) - L(M_1)$ such that $|w| < n_1 + n_2$. Moreover such a string w of length $j < n_1 + n_2$ exists if and only if ACC-SEQ$_{M_1}(j) \neq$ ACC-SEC$_{M_2}(j)$.

Let $M_1 = (S_1, I, \delta_1, s_1, F_1)$ and $M_2 = (S_2, I, \delta_2, s_2, F_2)$ be unambiguous finite automata with *no* λ-transitions such that $L(M_2)$ properly contains $L(M_1)$. Let $I = \{a_1, \cdots, a_m\}$ be of cardinality m. The following algorithm, given inputs M_1 and M_2, outputs a string w of minimal length such that $w \in L(M_2) - L(M_1)$ and $|w| < |S_1| + |S_2|$:

```
(1) Compute $j_0 = \min\{j \mid \text{ACC-SEQ}_{M_1}(j) \neq \text{ACC-SEQ}_{M_2}(j)\}$.
(2) $x \leftarrow \lambda$.
(3) $A_x \leftarrow \{s_1\}$ and $B_x \leftarrow \{s_2\}$.
(4) If $|x| = j_0$, then halt with output "$x$".
(5) For $1 \leqq i \leqq m$,
        $A_{xa_i} \leftarrow \{s' \in S_1 \mid \text{there exists } s \in A_x \text{ for which } s' \in \delta_1(s, a_i)\}$
        $B_{xa_i} \leftarrow \{t' \in S_2 \mid \text{there exists } t \in B_x \text{ for which } t' \in \delta_2(t, a_i)\}$.
(6) For $1 \leqq i \leqq M$,
        $d_i = \sum_{t' \in B_{xa_i}} \text{ACC-SEQ}_{M_2}(t, j_0 - |x| - 1) - \sum_{s' \in A_{xa_i}} \text{ACC-SEQ}_{m_1}(s', j_0 - |x| - 1)$.
(7) Letting $i_0$ be the least $i$ such that $d_i \neq 0$, $x \leftarrow xa_{i_0}$.
(8) Go to 4.
```

The algorithm is deterministic and polynomially time-bounded since:

(a) $j_0 < |S_1| + |S_2|$; and j_0 can be computed from M_1 and M_2 deterministically in polynomial-time by Lemma 4.4 and Theorem 4.6.

(b) The loop consisting of statements (4), (5), (6), (7) and (8) is executed j_0 times.

(c) For all $s \in S$, $t \in S_2$ and $i, k \leqq j_0$, the integers ACC-SEC$_{M_1}(s, i)$ and ACC-SEQ$_{m_2}(t, k)$ can be computed from M_1 and M_2 deterministically in polynomial-time by arguments analogous to the proofs of Lemmas 4.3 and 4.4. The correctness of the algorithm follows from the induction hypothesis:

for $0 \leqq j \leqq j_0$, after j times around the loop consisting of statements (4), (5), (6), (7) and (8), $|x| = j$ and there exists a string y of length $j_0 - j$ such that $xy \in L(M_2) - L(M_1)$.

THEOREM 4.9. *There exists a deterministic polynomial-time algorithm that, given unambiguous finite automata M_1 and M_2 such that $L(M_2) - L(M_2) \neq \emptyset$, outputs a string $w \in L(M_2) - L(M_1)$ of minimal length.*

Proof. The theorem follows from Lemma 4.5 and Theorem 4.8 since $L(M_2)-L(M_1)\neq\emptyset$ if and only if the containment of $L(M_1)\cap L(M_2)$ in $L(M_2)$ is proper. □

An example. We present an example of an unambiguous finite automaton M, together with the computation of the difference equation for ACC-SEQ$_M$.

Problem. Find the difference equation for the function ACC-SEQ$_M$ for the unambiguous finite automaton M without λ-transitions, with the start state A, input alphabet $\{0, 1\}$ and set of accepting states $\{C\}$ whose state transition function δ is given by

$$\delta(A,0)=\{A\},\quad \delta(A,1)=\{A,B\},\quad \delta(B,0)=\{C\},\quad \delta(B,1)=\{C\},$$
$$\delta(C,0)=\emptyset,\quad \delta(C,1)=\emptyset.$$

It is easily verified that $L(M)=\{0,1\}^*\cdot\{1\}\cdot\{0,1\}$.

Step 1. Write the equations from Lemma 4.2.

$$\text{TRAN-SEQ}_M\ (A,k+1)=2\cdot\text{TRAN-SEQ}_m\ (A,k)+\text{TRAN-SEQ}_M\ (B,k).$$
$$\text{TRAN-SEQ}_M\ (B,k+1)=2\cdot\text{TRAN-SEQ}_M\ (C,k).$$
$$\text{TRAN-SEQ}_M\ (C,k+1)=0.$$

Step 2. Obtain the equations for TRAN-SEQ$_M$ $(A, k+1)$, TRAN-SEQ$_M$ $(A, k+2)$ and TRAN-SEQ$_M$ $(A, k+3)$ as in the proof of Lemma 3.1.

$$\text{TRAN-SEQ}_M\ (A,k+1)=2\cdot\text{TRAN-SEQ}_M\ (A,k)+\text{TRAN-SEQ}_M\ (B,k).$$
$$\begin{aligned}\text{TRAN-SEQ}_M\ (A,k+2)&=2\cdot\text{TRAN-SEQ}_M\ (A,k+1)+\text{TRAN-SEQ}_M\ (B,k+1)\\&=4\cdot\text{TRAN-SEQ}_M\ (A,k)+2\cdot\text{TRAN-SEQ}_M\ (B,k)\\&\qquad+2\cdot\text{TRAN-SEQ}_M\ (C,k).\end{aligned}$$
$$\begin{aligned}\text{TRAN-SEQ}_M\ (A,k+3)&=2\cdot\text{TRAN-SEQ}_M\ (A,k+2)+\text{TRAN-SEQ}_M\ (B,k+2)\\&=2\cdot[4\cdot\text{TRAN-SEQ}_M\ (A,k)+2\cdot\text{TRAN-SEQ}_M\ (B,k)\\&\qquad+2\cdot\text{TRAN-SEQ}_M\ (C,k)]+0\\&=8\cdot\text{TRAN-SEQ}_M\ (A,k)+4\cdot\text{TRAN-SEQ}_M\ (B,k)\\&\qquad+4\cdot\text{TRAN-SEQ}_M\ (C,k).\end{aligned}$$

Step 3. Eliminate TRAN-SEQ$_M$ (B, k) and TRAN-SEQ$_M$ (C, k).

In this case, we need only subtract twice the second equation from the third to obtain

$$\text{TRAN-SEQ}_M\ (A,k+3)-2\cdot\text{TRAN-SEQ}_M(A,k+2)=0.$$

Since ACC-SEQ$_M$ $(k)=$ TRAN-SEQ$_M$ (A, k) for all $k\geqq 0$, the difference equation for ACC-SEQ$_M$ is

$$\text{ACC-SEQ}_M\ (k+3)-2\cdot\text{ACC-SEQ}_M\ (k+2)=0\quad\text{for } k\geqq 0.$$

By direct observation, ACC-SEQ$_M(0)=$ ACC-SEQ$_M$ $(1)=0$ and ACC-SEQ$_M$ $(k)=2^{k-1}$ for $k\geqq 2$, thus satisfying the derived difference equation.

5. Descriptions of ambiguity $\leqq k$.

For all $k\geqq 1$, we show that there exists deterministic polynomially time-bounded algorithms for the equivalence and containment problems for regular expressions, regular grammars and nondeterministic finite automata of degree of ambiguity $\leqq k$.

The basic idea of this section is that, although $\mathrm{ACC}_M(k)$ is no longer equal to $\mathrm{ACC\text{-}SEQ}_M(k)$, we can still find a difference equation for ACC_M of sufficiently low degree that the ideas of the previous section carry over.

We begin with two observations which help establish the difference equation for ACC_M. The observations are proved in Lemma 5.3.

Observation 5.1. Let $k \geqq 1$. Let $M = (S, I, \delta, s_1, F)$ be a nondeterministic finite automaton of degree of ambiguity $\leqq k$ with *no* λ-transitions. Let $|S| = n$. Let $1 \leqq l \leqq k$. Then there exists an $O(n^l)$ state nondeterministic finite automaton M_l constructable from M deterministically in polynomial-time such that, for all $m \in N$,

$$\mathrm{ACC\text{-}SEQ}_{M_l}(m) = \sum_{j=1}^{k} \binom{j}{l} \cdot l! \cdot M(m,j),$$

where $M(m,j)$ is the number of words of length m accepted by M by *exactly* j distinct derivations.

Observation 5.2. Let M, k, l, m and M_l be as in Observation 5.1. Then there exist rational constants $c_1, \cdots, c_k$, depending *only* on k and *not* on M, such that, for all $m \in N$, $\mathrm{ACC}_M(m) = c_1 \cdot \mathrm{ACC\text{-}SEQ}_{M_1}(m) + \cdots + c_k \cdot \mathrm{ACC\text{-}SEQ}_{M_k}(m)$.

LEMMA 5.3. *Observations* 5.1 *and* 5.2 *are correct.*

Proof. We first sketch the proof of Observation 5.1.

Let $M = (S, I, \delta, s_1, F)$, l, k and m be as in the statement of Observation 5.1. The nondeterministic finite automaton $M_l = (S', I, \sigma', s_1', F')$, where

(1) $S' = S \times S \times \cdots \times S \times (0,1) \times (0,1) \times \cdots \times (0,1)$, i.e. S' consists of l copies of S, l^2 copies of $(0,1)$;

(2) $s_1' = (s_1, s_1, \cdots, s_1, a_{11}, \cdots, a_{ll})$, where $a_{ij} = 1$, if $i = j$ and $a_{ij} = 0$, otherwise;

(3) $F' = \{(s_{i_1}, \cdots, s_{i_l}, a_{11}, \cdots, a_{ll}) \mid \text{where } s_{i_1}, \cdots, s_{i_l} \in F \text{ and each } a_{ij} = 1\}$; and

(4) the function δ' is defined by, $\delta'((t_1, \cdots, t_l, a_{11}, \cdots, a_{ll}), a) = \{(u_1, \cdots, u_l, b_{11}, \cdots, b_{ll}) \mid u_i \in \delta(t_i, a), b_{rs} = 1 \text{ if and only if } a_{rs} = 1 \text{ or } u_r \neq u_s\}$.

Clearly, $|M_l| = O(2^{l^2} \cdot |M|^l) = O(|M|^l)$, since $l \leqq k$ is a constant. Also the automaton M_l is constructable from M deterministically in polynomial time.

Intuitively, a state of S' represents the status of l derivations of M. The first l components represent the current state of the individual derivations and the subsequent l^2 components represent bits which indicate if two individual derivations were ever distinct. Bits a_{ii} are included so as to keep the notation from becoming too obscure and are always set to 1. The automaton starts with all derivations in the starting state and all distinct pairs marked as identical by setting the corresponding bits to 0. A sequence is accepted if all derivations end in an accepting state and all pairs have been marked as distinct.

By construction, $w \in L(M_1)$ if and only if there are $j \geqq 1$ distinct accepting transition sequences of M for w. In which case, there are $\binom{j}{l} \cdot l!$ distinct accepting transition sequences of M_l for w. This follows since the accepting transition sequences of M_l for w correspond exactly to the 1! permutations of the $\binom{j}{l}$ different combinations of l different accepting transition sequences of M for w. Hence for all $n \in N$,

$$\mathrm{ACC\text{-}SEQ}_{M_l}(n) = \sum_{j=1}^{k} \binom{j}{l} \cdot l! \cdot M(n,j)$$

as claimed.

To prove Observation 5.2, we first observe that

$$\mathrm{ACC}_M(n) = \sum_{j=1}^{k} M(n,j)$$

since the ambiguity of M is bounded by K. This equation together with the k equations of Observation 5.1 can be used to eliminate the $M(n, j)$ and obtain an equation of the form required in Observation 5.2. □

LEMMA 5.4. *There exists a deterministic polynomially time-bounded algorithm that, given as input a nondeterministic finite automaton $M = (S, I, \delta, s, F)$ outputs an equivalent nondeterministic finite automata M' with no λ-transitions having at most $|S|$ states and degree of ambiguity $\leqq$ the degree of ambiguity of M.*

Proof. The proof is identical to that of Lemma 4.1. □

LEMMA 5.5. *Let $k \geqq 1$. Let $n \in N$. For any nondeterministic n state finite automaton M of degree of ambiguity $\leqq k$ with no λ-transitions, the function* $\mathrm{ACC}_M(n)$ *satisfies a difference equation of degree $O(n^k)$.*

Proof. By Observation 5.2, the function $\mathrm{ACC}_M(n)$ is a linear combination of the functions $\mathrm{ACC\text{-}SEQ}_{M}, \cdots$, and $\mathrm{ACC\text{-}SEQ}_{M_k}$. By Lemma 4.3 and Observation 5.1, the functions $\mathrm{ACC\text{-}SEQ}_{M}, \cdots$, and $\mathrm{ACC\text{-}SEQ}_{M_k}$ satisfy difference equations of degrees $O(n), \cdots$, and $O(n^k)$, respectively. Thus by Lemma 3.3, the function $\mathrm{ACC}_M(n)$ satisfies a difference equation of order $O(k \cdot n^k) = O(n^k)$. □

LEMMA 5.6. *Let $k \geqq 1$. There exists a deterministic polynomially time-bounded algorithm that, given as input a nondeterministic finite automaton $M = (S, I, \delta, s_1, F)$ of degree of ambiguity $\leqq k$ with no λ-transitions and a string 1^n, outputs the first n values of the function* ACC_M.

Proof. The machines M_2 described in Observation 5.1 can be constructed, and the algorithm from the proof of Lemma 4.4 can be used to compute the first n values of $\mathrm{ACC\text{-}SEQ}_{M_l}$ for $1 \leqq l \leqq k$. These values are combined using Observation 5.2 to obtain the values for ACC_M. □

LEMMA 5.7. *Let $k \geqq 1$. There exists a deterministic polynomially time-bounded algorithm that, given as input an n_1 state nondeterministic finite automaton M_1 and an n_2 state nondeterministic finite automaton M_2, both of degree of ambiguity $\leqq k$ with no λ-transitions, outputs an $n_i \cdot n_2$ state nondeterministic finite automaton M with no λ-transitions and of degree of ambiguity $\leqq k^2$ such that*

$$L(M) = L(M_1) \cap L(M_2).$$

Proof. The proof is identical to that of Lemma 4.5. □

THEOREM 5.8. *Let $k \geqq 1$. There exists a deterministic polynomially time-bounded algorithm that, given as input an n_1 state nondeterministic finite automaton M_1 and an n_2 state nondeterministic finite automaton M_2, both of degree of ambiguity $\leqq k$ and such that $L(M_1) \subset L(M_2)$, the algorithm decides if the set containment is proper.*

Proof. By Lemma 5.4 we may assume that M_1 and M_2 have *no* λ-transitions. Since $L(M_1) \subset L(M_2)$ by assumption, the set containment is proper if and only if $\mathrm{ACC}_{M_1}(m) \neq \mathrm{ACC}_{M_2}(m)$ for some $m \in N$. By Lemma 5.5 the functions ACC_{M_1} and ACC_{M_2} satisfy difference equations of degrees $\leqq c \cdot n_1^k$ and $c \cdot n_2^k$, respectively, where c is a constant independent of M_1 or of M_2. Thus by Lemma 3.2, the functions ACC_{M_1} and ACC_{M_2} are equal if and only if, for all natural numbers $m < c \cdot n_1^k + c \cdot n_2^k$, $\mathrm{ACC}_{M_1}(m) = \mathrm{ACC}_{M_2}(m)$. By Lemma 5.6 this can be checked deterministically in polynomial-time. □

COROLLARY 5.9. *Let $k \geqq 1$. The equivalence and containment problems for the nondeterministic finite automata of degree of ambiguity $\leqq k$ are decidable deterministically in polynomial-time.*

Proof. For finite automata M_1 and M_2, $L(M_1) \subset L(M_2)$ if and only if the language $L(M_1) \cap L(M_2)$ is *not* properly contained in the language $L(M_2)$. Thus the corollary follows immediately from Lemma 5.7 and Theorem 5.8. □

Finally, we note the following:

THEOREM 5.10. *For all $k \geqq 1$, there is a deterministic polynomially time-bounded algorithm to decide if a nondeterministic finite automaton is ambiguous of degree $\leqq k$.*

Proof. Let $k \geqq 1$. Let M be a nondeterministic finite automaton. Then M is ambiguous of degree $\leqq k$ if and only if $L(M_{k+1}) = \emptyset$, where M_{k+1} is the finite automaton constructed from M as in the proof of Lemma 5.3. For a fixed k, this construction can be accomplished deterministically in polynomial time. It is well known that the emptiness problem for nondeterministic finite automata is decidable deterministically in polynomial-time. □

6. Extensions that fail. In § 5, we extended the results of § 4 to k-bounded ambiguity. It is natural to ask if the results of § 4 can be extended further by considering other classes of finite automata that include the unambiguous automata but do not include all finite automata. Here we show that two extensions suggested by our earlier results fail (unless P = NP). This provides evidence that our deterministic polynomially time-bounded algorithms are close to being as generally applicable as possible.

First, we consider the class B of nondeterministic finite automata of bounded degree of ambiguity. The algorithms of § 5 are not directly applicable since the degrees of the polynomials that bound their runtimes grow unboundedly with k. Unless P = NP = CoNP, *no* deterministic polynomially time-bounded algorithm exists for the equivalence problem or for the containment problem for the class B.

THEOREM 6.1. *The equivalence and containment problems for the class B are* CoNP-*hard.*

Proof. In [HRS] we showed that the equivalence and containment problems for regular expressions containing only the operators $\cup$ and $\cdot$ are CoNP-complete. Clearly such expressions are of bounded degree of ambiguity. It is easy to show that there is a deterministic polynomially time-bounded algorithm that converts a regular expression containing only the operators $\cup$ and $\cdot$ into an equivalent nondeterministic finite automata of bounded degree of ambiguity. (The standard constructions for converting expressions preserve bounded degree of ambiguity.) □

The next extension is based on the following definition.

DEFINITION 6.2. Let M be a nondeterministic finite automaton with input alphabet Σ. Then **STATE**(M) equals the number of states of M; and **SHORT**(M) equals -1, if $L(M) = \Sigma^*$, and equals the length of a shortest string in $\Sigma^* - L(M)$, otherwise.

In those cases where we have polynomial algorithms, STATE and SHORT are polynomially related.

THEOREM 6.3.

(1) *For all unambiguous finite automata M,* SHORT$(M) \leqq$ STATE(M).

(2) *Let $k \geqq 1$. Then there exists a polynomial p_k such that, for all nondeterministic finite automata M of degree of ambiguity $\leqq k$,* SHORT$(M) \leqq p_k($STATE$(M))$.

Proof. These assertions follow from the proofs of Theorems 4.6 and 5.8 taking machine M_2 to be the one state machine which accepts all input sequences. □

This result is in sharp contrast to the general case described by the following result.

THEOREM 6.4. *There exists $c > 0$ such that, for infinitely many nondeterministic finite automata M,*

$$\text{SHORT}(M) > 2^{c \cdot \text{STATE}(M)}$$

Proof. This result is suggested by the work of other authors (see [H] and [N]). The construction in [SM] used to prove regular expression equivalence PSPACE-complete can be made into such an example by using an lba which accepts 1^n only after making 2^n moves. In [H, Prop. 3.9] this is discussed. [N] shows that an exponential distinguishing sequence is required to distinguish two arbitrary automata, but not to distinguish one automaton from the automaton which accepts all sequences. However, we have found a variation on the example given in [N] which proves our result. Because of [H], we consider the proof to be already in the literature and omit further discussion. □

It is also true that, even for machines with single letter alphabets, there is no polynomial relationship between SHORT (M) and SIZE (M). An easy variation on an example in [N] proves this. This is also implied by the proof in [SM] that the predicate "$L(R)=\{0\}^*$" is CoNP-complete for regular expressions over the alphabet $\{0\}$.

Theorem 6.3 suggests that some condition on class C of nondeterministic finite automata such as:

$$\text{(6.1)} \qquad \text{For all } M \in C, \quad \text{SHORT}(M) \leqq \text{STATE}(M)$$

might suffice to insure that the equivalence and containment problems for C are decidable deterministically in polynomial-time. Again unless $P = NP = \text{CoNP}$, this is not the case.

THEOREM 6.5. *Let C be the class of nondeterministic finite automata that simultaneously are of bounded degree of ambiguity and satisfy condition* (6.1). *Then the equivalence and containment problems for C are* CoNP-*hard.*

Proof. The class C contains the class of all nondeterministic finite automata of bounded degree of ambiguity that recognize finite sets. (A sequence accepted by such an automaton cannot repeat a state and so an accepted sequence of n inputs must visit $n+1$ distinct states and all accepted strings must be shorter than STATE (M).) Therefore in particular, C contains the set of machines obtained from expressions involving $\cup$ and $\cdot$ as used in the proof of Theorem 6.1. But this same proof showed that the equivalence and containment problems for this latter class of finite automata were shown CoNP-hard. □

7. Relative economy of description.

7. Relative economy of description. We now contrast three descriptions of regular sets: the nondeterministic finite automata, the unambiguous finite automata and the deterministic finite automata. For each pair of descriptions, the economy of description is exponential. These results extend and complement the results on the relative economy of description of regular set descriptions in [MF].

THEOREM 7.1. *There exists $c>0$ such that for infinitely many nondeterministic finite automata M and any equivalent unambiguous finite automaton N, $|N|>2^{c\cdot|M|}$.*

Proof. The value of c and set of automata satisfying Theorem 6.4 also satisfy this theorem. To see this, let M be one of those automata and N an equivalent unambiguous automaton. Then by Theorems 6.3 and 6.4 and the definition of equivalence,

$$|N| \geqq \text{STATE}(N) \geqq \text{SHORT}(N) = \text{SHORT}(M) > 2^{c\cdot|M|}. \qquad □$$

THEOREM 7.2. *For all $n \geqq 1$, there is an unambiguous finite automaton M_n with $n+2$ states and of size $2\cdot(n+2)$ such that any equivalent deterministic finite automaton N_n has $\geqq 2^{n+1}$ states and is of size $\geqq 2^{n+2}$.*

Proof. For all $n \geqq 1$, $M_n = (S_n, \{0, 1\}, \delta_n, s_0, \{s_{n+1}\})$, where

(1) $S_n = \{s_0, s_1, \cdots, s_{n+1}\}$; and

(2) $\delta_n(s_0, 0) = \{s_0\}$, $\delta_n(s_0, 1) = \{s_0, s_1\}$, for $1 \leqq i \leqq n$ $\delta_n(s_i, 0) = \delta_n(s_i, 1) = \{s_{i+1}\}$, and $\delta_n(s_{n+1}, 0) = \delta_n(s_{n+1}, 1) = \varnothing$.

The only way M_n accepts is to stay in state s_0 for some initial string, move to state s_1 using input 1, and reach a single final state s_{n+1} after exactly n move inputs. Thus $L(M_n) = \{0, 1\}^* \cdot \{1\} \cdot \{0, 1\}^n$ and strings can be accepted in only one way.

Automaton M_n has $n+2$ states. An equivalent deterministic automaton must remember the last $n+1$ inputs so as to be able to give the right answer for the current input and the next n inputs. Hence the deterministic machine needs 2^{n+1} states. □

The example in the above proof has been known since at least the early 1970's; here we add the observation that the M_n are in fact unambiguous.

The third exponential gap between deterministic and nondeterministic descriptors is implied by either of the above theorems and has been known for a long time. It is given in [MF, Prop. 1] where its early history is discussed. This exponential difference is also the maximum possible because it is achieved as an upper bound by the standard subset construction. Thus there is no class of regular sets which simultaneously satisfy the exponential gap between deterministic and unambiguous finite automata and the exponential gap between unambiguous and arbitrary nondeterministic finite automata. The possibility of simultaneous nonpolynomial gaps remains open.

Finally, we present one corollary of Theorem 7.1 for context-free grammars.

DEFINITION 7.3. Let $G = (N, \Sigma, P, S)$ be a context-free grammar defined as in [AU]. Then the *size of G*, denoted by $|G|$, equals

$$\sum_{A \to w \in P} (|w|+1).$$

COROLLARY 7.4. *There exists $c > 0$ such that for infinitely many context-free grammars G and for any structurally equivalent structurally unambiguous context-free grammar H,*

$$|H| > 2^{c \cdot |G|}.$$

Proof. The corollary follows from Theorem 7.1, by noting that:

(1) two regular grammars are structurally equivalent if and only if they are equivalent [HRS];

(2) a regular grammar is structurally unambiguous if and only if it is unambiguous [T]; and

(3) there is an algorithm for converting a nondeterministic finite automaton M into an equivalent regular grammar G such that $|G|$ is $O(|M|)$ (see [AU]). □

REFERENCES

[AHU] A. V. AHO, J. E. HOPCROFT AND J. D. ULLMAN, *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[AU] A. V. AHO AND J. D. ULLMAN, *The Theory of Parsing, Translation, and Compiling*, Vol. 1, Prentice-Hall, Englewood Cliffs, NJ, 1972.

[B] J. A. BRZOZOWSKI, *Canonical regular expressions and minimal state graphs for definite events*, in Proc. Symposium on Mathematical Theory of Automata, Polytechnic Press of the Polytechnic Institute of Brooklyn, Brooklyn, NY, 1963, pp. 529-562.

[CI] T. CHAN AND O. H. IBARRA, *On the finite-valuedness problem for sequential machines*, Theoret. Comput. Sci., 23 (1983), pp. 95-101.

[GI] E. M. GURARI AND O. H. IBARRA, *A note on finite-values transducers*, Math. Systems Theory, 16 (1983), pp. 61-66.

[HU] J. E. HOPCROFT AND J. D. ULLMAN, *Formal Languages and Their Relation to Automata*, Addison-Wesley, Reading, MA, 1969.

[H] H. B. HUNT, III, *Observations on the complexity of regular expression problems*, J. Comput. System Sci., 19 (1979), pp. 222-236.

[HRS] H. B. HUNT, III, D. J. ROSENKRANTZ AND T. G. SZYMANSKI, *On the equivalence, containment, and covering problems for the regular and context-free languages*, J. Comput. System Sci., 12 (1976), pp. 222-268.

[Mc] R. MCNAUGHTON, *The loop complexity of regular events*, Inform. Sci., 1, 1969, pp. 305-328.

[McP] R. MCNAUGHTON AND S. PAPERT, *Counter-Free Automata*, MIT Press, Cambridge, MA, 1971.

[MF] A. R. MEYER AND M. J. FISCHER, *Economy of description by automata, grammars, and formal systems*, in Conference Record, Twelfth Annual IEEE Symposium on Switching and Automata Theory, East Lansing, MI, October 1971, pp. 144-152.

[N] A. NOZAKI, *Equivalence problems of non-deterministic finite automata*, J. Comput. System Sci., pp. 18 (1979), 8-17.

[PU] M. C. PAULL AND S. H. UNGER, *Structural equivalence of context-free grammars*, J. Comput. System Sci., 2 (1968), pp. 427-463.

[RH] D. J. ROSENKRANTZ AND H. B. HUNT, III, *Efficient algorithms for structural similarity of grammars*, in Proc. 7th ACM Symposium on Principles of Programming Languages, Las Vegas, NV, January 1980, pp. 213-219.

[SM] L. J. STOCKMEYER AND A. R. MEYER, *Word problems requiring exponential time: preliminary report*, in Proc. 5th Annual ACM Symposium on Theory of Computing, Austin, TX, May 1973, pp. 1-9.

[T] J. W. THATCHER, *Tree automata: an informal survey*, in Currents in the Theory of Computing, A. V. Aho, Ed., Prentice-Hall, Englewood Cliffs, NJ, 1973, pp. 143-172.

[Z] Y. ZACKSTEIN, *On star-free events*, in Conference Record, the Eleventh Annual IEEE Symposium on Switching and Automata Theory, 1970, pp. 76-80.