The Complexity of Word Problems – This Time with Interleaving

Alain J. Mayer[1]
Dept. of Computer Science
Columbia University
New York, NY 10027

Larry J. Stockmeyer
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

Abstract. We consider regular expressions extended with the interleaving operator, and investigate the complexity of membership and inequivalence problems for these expressions. For expressions using the operators union, concatenation, Kleene star, and interleaving, we show that the inequivalence problem (deciding whether two given expressions do not describe the same set of words) is complete for exponential space. Without Kleene star, we show that the inequivalence problem is complete for the class Σ_2^p at the second level of the polynomial-time hierarchy. Certain cases of the membership problem (deciding whether a given word is in the language described by a given expression) are shown to be NP-complete. It is also shown that certain languages can be described exponentially more succinctly by using interleaving.

This paper appears in *Information and Computation*, Vol. 115, No. 2, 1994, pp. 293–311.

[1]Part of this work was done at Brown University and the IBM T.J. Watson Research Center. Supported by ONR grant N00014-91-J-1613 at Brown University.

1 Introduction

There has been considerable progress in classifying the computational complexity of decision problems involving regular expressions, possibly including operators on languages other than the usual operators union, concatenation, and star. Problems which have been studied include *inequivalence*, i.e., deciding whether two given expressions do not describe the same set of words, and *membership*, i.e., deciding whether a given word is in the language described by a given expression. For example, the inequivalence problem for regular expressions is complete for polynomial space (Meyer and Stockmeyer, 1972). If intersection is also included, the complexity increases to exponential-space-complete (Hunt, 1973a; Fürer, 1980). Additional work on this subject can be found, for example, in (Hunt, 1973b; Hunt, Rosenkrantz, and Szymanski, 1976; Stockmeyer and Meyer, 1973); see also (Hopcroft and Ullman, 1979; Wagner and Wechsung, 1986).

In this paper, we focus on the *interleaving* operator (sometimes called *shuffle* in the literature). The interleaving of words x and y, denoted $x|y$, is the set of all words of the form

$$x_1 y_1 x_2 y_2 \dots x_k y_k$$

where $k > 0$, $x = x_1 x_2 \dots x_k$, $y = y_1 y_2 \dots y_k$, and where the words x_i and y_i, $1 \leq i \leq k$, can be of arbitrary length (including the empty word).

The motivation to investigate the interleaving operator is twofold. Our primary motivation is that the interleaving operator can be interpreted as the simplest case of the *composition* operator used in algebraic approaches to modeling concurrent computation. Interleaving represents the case where processes run concurrently in such a fashion that their atomic steps can be arbitrarily interleaved but where no communication between them takes place. One of the best known formalisms for specifying and verifying concurrent systems is CCS (Milner, 1980). In (Milner, 1984), a restricted set of algebraic operations, $\{\cup, \cdot, *\}$, is used to form the *star expressions* in CCS. These expressions are syntactically identical to regular expressions, but instead of having as semantics "sets of strings", their semantics is "equivalence classes of processes". Kanellakis and Smolka (1990) show that the observational equivalence problem for star expressions is solvable in polynomial time. They leave open the question of determining the complexity of this problem for expressions extended by a composition operator. This led us to investigate how interleaving affects the complexity of decision problems for regular expressions.

Secondly, as we discovered while doing this work, the interleaving operator has some interesting properties of its own:

Succinctness: The use of the interleaving operator can shorten a regular expression by an exponential amount.

Simulation of Integer Addition and Intersection: Under certain format restrictions, addition of positive integers and intersection of expressions can be simulated by the use of the interleaving operator.

Complexity: The inequivalence problem for expressions with interleaving, but without star, is one of the few natural problems known to be Σ_2^p-complete.

Motivated by the connection between interleaving and concurrency, there has been some study of how interleaving affects the complexity of word problems. Ogden, Riddle and

	without interleaving	with interleaving
inequivalence of regular expressions	PSPACE	ESPACE
inequivalence of regular expressions without star	NP	Σ_2^p

Table 1: The effect of interleaving on complexity.

Rounds (1978) state that the membership problem for regular expressions with interleaving is NP-complete. Mansfield (1983) and Warmuth and Haussler (1984) show that it is NP-complete to decide, given words w and $w_1, w_2, \ldots, w_n$, whether w can be written as an interleaving of $w_1, \ldots, w_n$. (Warmuth and Haussler (1984) prove this even for the case $w_1 = w_2 = \cdots = w_n$.) In contrast, the membership problem for regular expressions (without interleaving) can be solved in polynomial time.

One of our main results is that the inequivalence problem for regular expressions with interleaving is complete for exponential space. As noted above, without interleaving the problem is known to be complete for polynomial space. Without star, we show that the inequivalence problem for regular expressions, using only union, concatenation and interleaving, is complete in the class Σ_2^p at the second level of the polynomial-time hierarchy. With only union and concatenation, the inequivalence problem is known to be NP-complete (Hunt, 1973b; Stockmeyer and Meyer, 1973). These results are summarized in Table 1, where ESPACE is the class of languages which can be recognized in space 2^{cn} for some constant c. In the case with star, we show that interleaving is powerful enough to simulate intersection under certain format restrictions. ESPACE-hardness of the inequivalence problem for expressions with interleaving then follows, by a polynomial-time reduction, from the ESPACE-hardness of this problem for expressions with intersection. In the case without star, we show that interleaving can simulate addition of integer vectors, again under appropriate format restrictions. We can then emulate a proof of Stockmeyer (1977) to show that the inequivalence problem is Σ_2^p-complete.

We now outline the paper. To illustrate the nature of the interleaving operator, we begin in Section 2 by reviewing a polynomial-time algorithm for checking whether a given word z can be written as an interleaving of given words $u_1, u_2, \ldots, u_k$ where k is a constant. Definitions are given in Section 3. In Section 4, we present a language for which a succinct expression with interleaving exists but every regular expression (without interleaving) is longer by an exponential amount. In Section 5, several cases of the membership problem are shown to be NP-complete. For example, although the membership problem for regular expressions can be solved in polynomial time, the combination of interleaving with any one of the three regular operators leads to an NP-complete membership problem; as noted above, the case with interleaving and concatenation was known previously. Sections 6 and 7 contain our results, described above, on the complexity of the inequivalence problem for expressions without and with star, respectively.

2 Example: Interleaving of a Constant Number of Strings

To illustrate the interleaving operator, we show how to answer the following question in a straightforward way for any constant k: Given words $z, u_1, u_2, \ldots, u_k$ over some alphabet Σ, does z belong to $u_1 \,|\, u_2 \,|\, \ldots \,|\, u_k$, the set of interleavings of $u_1, \ldots, u_k$? The algorithm is essentially the simple dynamic programming algorithm of van Leeuwen and Nivat (1982) and Mansfield (1983), presented in terms of a nondeterministic finite-state automaton (NFA). To simplify the description, assume that all u_i have the same length n and that the length of z is kn.

1. Construct the NFA M for $u_1 \,|\, u_2 \,|\, \ldots \,|\, u_k$. Its transition diagram will be an $(n+1) \times (n+1) \ldots \times (n+1)$ (k times) grid of states. Thus we can think of it as a k-dimensional hypercube of side $n+1$, where every state $(l_1, \ldots, l_k)$ corresponds to having consumed l_i symbols of u_i. Thus, for every state $(l_1, \ldots, l_k)$ with $0 \leq l_i \leq n$ for $1 \leq i \leq k$, and every j with $l_j \geq 1$, there is a transition from $(l_1, \ldots, l_j - 1, \ldots, l_k)$ to $(l_1, \ldots, l_j, \ldots, l_k)$ labeled with the l_j-th symbol of u_j. The initial state is $(0, 0, \ldots, 0)$ and the only accepting state is $s_a = (n, n, \ldots, n)$. Every state has at most k successors, each of which has one coordinate closer to s_a. Thus there are $O(n^k)$ states and $O(n^k)$ transitions. Note that every path from the initial state to the accepting state has length kn.

2. Let S be a set of states. Simulate M on input z by storing in S the states which M can reach after reading the prefix of z consumed so far. After reading at most kn symbols, one of the following two conditions will become true: (i) $S = \{\}$ and thus REJECT or (ii) $S = \{s_a\}$ and thus ACCEPT. Note that the size of S can be at most $O(n^{k-1})$, since there are at most $O(n^{k-1})$ states at distance l ($1 \leq l \leq kn$) from the initial state.

It can easily be verified that the above procedure can be carried out using $O(n^k)$ time and $O(n^{k-1})$ space on a unit-cost RAM. Van Leeuwen and Nivat (1982) improved the time performance to $O(n^k / \log^{1/(k-1)} n)$. This was further improved by Ibarra, Palis and Chang (1985) to $O(n^k / \log^{k/(k-1)} n)$.

3 Definitions

Basic familiarity with regular expressions, time and space complexity, polynomial-time reducibility, and complete problems is assumed. The necessary background, if needed, can be found in (Hopcroft and Ullman, 1979) or (Wagner and Wechsung, 1986), for example.

We now define more precisely the types of expressions and problems of interest. Let ϵ denote the empty word. Let Σ be a finite alphabet and let S be a subset of the operators $\{\cup, \cdot, *, \cap, |\}$. We define the S-*expressions* (over Σ) and simultaneously define the operator L which maps each S-expression to a subset of Σ^*:

1. For every $\sigma \in \Sigma \cup \{\epsilon\}$, σ is an S-expression, and $L(\sigma) = \{\sigma\}$;

2. If r_1 and r_2 are S-expressions and $@ \in S - \{*\}$, then $(r_1 \mathbin{@} r_2)$ is an S-expression, and $L((r_1 \mathbin{@} r_2)) = L(r_1) \mathbin{@} L(r_2)$;

3. If r is an S-expression, then (r^*) is an S-expression, and $L((r^*)) = (L(r))^*$.

In 2, the interleaving operator is extended to sets of words in the obvious way, i.e., $L_1|L_2$ is the union of the sets $w_1|w_2$ taken over all $w_1 \in L_1$ and $w_2 \in L_2$. When writing expressions in the text, extraneous parentheses are often omitted. Although it is sometimes convenient to use ϵ when writing expressions, our results do not change if expressions cannot contain ϵ.

Letting S be as above, the problem MEMBER-S is the problem of deciding, given an S-expression r and a word $w \in \Sigma^*$, whether $w \in L(r)$. The problem INEQ-S is the problem of deciding, given two S-expressions r_1 and r_2, whether $L(r_1) \neq L(r_2)$. The problem NEC-S is to decide, for a given r, whether $L(r) \neq \Sigma^*$. (So NEC-S is a special case of INEQ-S if S contains union and star.)

$|w|$ denotes the length of the word w, and $|r|$ denotes the length of the expression r.

By a straightforward cross-product construction, we can define $|$ also as an operator on nondeterministic finite automata (NFA's) M_1 and M_2 in such a way that $L(M_1 \,|\, M_2) = L(M_1) \,|\, L(M_2)$. Here is the relevant definition (see (Eilenberg, 1974)):

> Let $M_i = \langle Q_i, \Sigma, \delta_i, p_{0i}, F_i \rangle$ $(i = 1, 2)$ be an NFA with (in notation of Hopcroft and Ullman (1979)) state set Q_i, input alphabet Σ, transition function δ_i, initial state p_{0i}, and accepting states F_i. Then $M = M_1 \,|\, M_2$ is defined as follows:
> $M = \langle Q_1 \times Q_2, \Sigma, \delta, (p_{01}, p_{02}), F \rangle$ where the new transition relation δ is defined as $\delta((q_1, q_2), a) = (\delta_1(q_1, a) \times \{q_2\}) \cup (\{q_1\} \times \delta_2(q_2, a))$, and the new set of accepting states F is defined by $F((q_1, q_2)) = F_1(q_1) \wedge F_2(q_2)$.

Note that the number of states of $M_1|M_2$ is the product of the number of states of M_1 and the number of states of M_2. It follows that any $\{\cup, \cdot, *, |\}$-expression r can be converted to an equivalent NFA having $2^{O(|r|)}$ states.

We will also use a "shuffle resistant" code of Warmuth and Haussler (1984) for coding symbols from a large alphabet as binary words. Using this code, all of our results hold for expressions over an alphabet Σ of size two, provided that concatenation is an allowed operation. Given an alphabet $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_k\}$, let h be the homomorphism $h : \Sigma^* \to \{0,1\}^*$ defined by $h(\sigma_i) = 0^{i+1}1^i$ for $1 \leq i \leq k$. If r is an S-expression over Σ, let $h(r)$ be the expression obtained from r by replacing each occurrence of an alphabet symbol σ by $h(\sigma)$. If S contains interleaving, then $L(h(r))$ can contain words which are not codewords, i.e., not in $h(\Sigma^*)$. The following proposition shows, however, that if we restrict attention to codewords, then the language of $h(r)$ is the code of $L(r)$.

Proposition 3.1 *For any $S \subseteq \{\cup, \cdot, *, \cap, |\}$ and any S-expression r,*

$$L(h(r)) \cap h(\Sigma^*) = h(L(r)).$$

Proof (sketch). The proof is by induction on the structure of r, using a stronger inductive hypothesis which also states that the non-codewords in $L(h(r))$ have a special form. Let B be the set of words of the form $u0^i1^jz$ where $u \in h(\Sigma^*)$, $i \geq j+2$, and $z \in \{\epsilon\} \cup 0 \cdot \{0,1\}^* \cdot 1$. So every word $w \in B$ is not a codeword and, moreover, the first "mistake" in w consists of a string of 0's followed by a string of 1's where the length of the 0's string is at least two more than the length of the 1's string. It is easy to show by induction on the structure of r that $L(h(r)) = h(L(r)) \cup W_r$ for some $W_r \subseteq B$, which proves the result. ∎

4 Succinctness

Consider the alphabet $\Sigma_n = \{\sigma_1, \sigma_2, \ldots, \sigma_n\}$ and the language L_n of all permutations of length n in Σ_n^*, i.e., L_n is the set of words of length n in which each symbol σ_i appears exactly once. Obviously, we have $L_n = L(\sigma_1|\sigma_2|\ldots|\sigma_n)$. But as the following easy proposition shows, there is no NFA, and therefore no standard regular expression, of subexponential length denoting L_n.

Proposition 4.1 *If M is an NFA such that $L(M) = L_n$, then M has at least 2^n states.*

Proof. For $S \subseteq \Sigma_n$, let the word $w(S)$ be the concatenation of the symbols in S in order of increasing index. Let $\overline{S}$ denote the complement of S with respect to Σ_n. Note that for any S, $w(S)w(\overline{S}) \in L_n$. We claim that the number of subsets of Σ_n, namely 2^n, is a lower bound on the number of states of any NFA accepting L_n.

Assume that there is an NFA M with fewer states. Then there are two subsets S and T with $S \neq T$ and a state q such that there is a computation path of M on input $w(S)$ from the initial state to q, a path on input $w(T)$ from the initial state to q, a path on input $w(\overline{S})$ from q to an accepting state, and a path on input $w(\overline{T})$ from q to an accepting state. Assuming (without loss of generality) that S is not a subset of T, there must be a σ_j in S which is not in T, so σ_j is in $\overline{T}$. Therefore, M accepts the word $w(S)w(\overline{T})$ which contains two occurrences of σ_j. Thus any NFA accepting L_n must have at least 2^n states. ∎

Remark. This result remains true for the language $L'_n = L(h(\sigma_1)\,|\,h(\sigma_2)\,|\,\ldots\,|\,h(\sigma_n))$ over the alphabet $\{0, 1\}$, where h is the code described in Section 3. The proof is virtually identical to the proof just given. By Proposition 3.1, we can restrict attention to words in $h(\Sigma^*)$, and consider words of the form $h(w(S))$ in the proof.

5 Instances of MEMBER which are NP-Complete

The problem MEMBER-$\{\cup, \cdot, *, \cap\}$ is known to be solvable in polynomial time (see the solution to Problem 3.23 in (Hopcroft and Ullman, 1979)). As noted above, it is also known that MEMBER-$\{\cup, \cdot, *, \cap, |\}$ is NP-hard, even when restricted to expressions containing only concatenation and interleaving (Mansfield, 1983; Warmuth and Haussler, 1984). In this section, we identify other restrictions which are NP-hard, and show that MEMBER-$\{\cup, \cdot, *, \cap, |\}$ belongs to NP in general.

Theorem 5.1 *MEMBER-$\{\cup, \cdot, *, \cap, |\}$ is NP-complete. The problem remains NP-hard even if: (1) only $\{\cup, |\}$ are used in the expression, (2) only $\{*, |\}$ are used, or (3) $\{\cup, \cdot, \cap, |\}$ are used and $|$ appears only once.*

We prove this theorem by a sequence of lemmas.

Lemma 5.2 *MEMBER-$\{\cup, |\}$ and MEMBER-$\{*, |\}$ are NP-hard.*

Proof. The lemma is proved by doing reductions from the NP-hard 3-dimensional matching problem:

Given disjoint sets $W = \{w_1, w_2, \ldots, w_q\}$, $X = \{x_1, x_2, \ldots, x_q\}$, $Y = \{y_1, y_2, \ldots, y_q\}$, and given a set $M \subseteq W \times X \times Y$, say $M = \{m_1, m_2, \ldots, m_k\}$, does M have a matching? I.e., does there exist a set $M' \subseteq M$ in which every element of $W \cup X \cup Y$ appears exactly once?

Let $m_i = (a_i, b_i, c_i)$ for $1 \leq i \leq k$, and let d be a symbol not in $W \cup X \cup Y$. Define the expressions $\mu_1, \ldots, \mu_k, r$ and the string z over the alphabet $\Sigma = W \cup X \cup Y \cup \{d\}$ as follows:

$$\mu_i = ((a_i \mid b_i \mid c_i) \cup d)$$

$$r = \mu_1 \mid \mu_2 \mid \ldots \mid \mu_k$$

$$z = w_1\, w_2 \,\ldots\, w_q\, x_1\, x_2 \,\ldots\, x_q\, y_1\, y_2 \,\ldots\, y_q\, d^{k-q}.$$

It should be obvious that $z \in L(r)$ iff M contains a matching.

The reduction for $\{*, |\}$ is similar. Take instead

$$\mu_i = (a_i \mid b_i \mid c_i)^*$$

$$z = w_1\, w_2 \,\ldots\, w_q\, x_1\, x_2 \,\ldots\, x_q\, y_1\, y_2 \,\ldots\, y_q.$$

∎

MEMBER-$\{\cdot, |\}$ remains NP-hard when restricted to an alphabet of size 2. We note, however, that when restricted to an alphabet Σ of size k, MEMBER-$\{\cup, |\}$ can be solved in time $n^{O(k)}$ by dynamic programming.

We next consider the case where the number of occurrences of interleaving is bounded. Using the cross-product machine construction described in Section 3 it is easy to show that, for each constant k, the problem MEMBER-$\{\cup, \cdot, *, |\}$ restricted to expressions having at most k occurrences of $|$ can be solved in polynomial time, since any such expression can be converted to an equivalent NFA having $n^{O(k+1)}$ states. In contrast, we have the following if intersection is included.

Lemma 5.3 *MEMBER-$\{\cup, \cdot, \cap, |\}$ is NP-hard even if $|$ appears just once in the expression.*

Proof. We prove this lemma by doing a reduction from the well known NP-hard problem 3SAT. Assume that we are given a formula $C = \{c_1, c_2, \ldots, c_m\}$ as a collection of m clauses on a set $\{v_1, v_2, \ldots, v_n\}$ of variables such that $|c_i| = 3$ $(1 \leq i \leq m)$.

We will use the following notation: P_i $(1 \leq i \leq m)$ is the set of indices of the variables appearing positively in c_i, and N_i $(1 \leq i \leq m)$ is the set of indices of the variables appearing negatively in c_i.

We construct a string z and an expression r over the alphabet $\Sigma = \{v_1, v_2, \ldots, v_n\}$ such that $z \in L(r)$ iff C is satisfiable.

Let C_i $(1 \leq i \leq m)$ be the regular expression defined as follows:

$$C_i = \bigcup_{k \in N_i} ((\Sigma - v_k) \cup \epsilon)^n \;\cup\; ((\Sigma \cup \epsilon)^n \cdot (\bigcup_{l \in P_i} v_l) \cdot (\Sigma \cup \epsilon)^n).$$

Thus $C_i \cap (\Sigma \cup \epsilon)^n$ contains exactly all words of length at most n in which (1) at least one symbol whose index is in N_i does not appear, or in which (2) at least one symbol whose index is in P_i appears. Now define the expression r and the string z as

$$r = (C_1 \cap C_2 \cap \ldots \cap C_m) \mid (\Sigma \cup \epsilon)^n$$

$$z = v_1\, v_2\, \ldots\, v_n.$$

(1) C is satisfiable $\Rightarrow z \in L(r)$:
Let T be a satisfying truth assignment for C. Let the partitioning of z be such that the symbol v_j belongs to the LHS of "$|$" iff $T(v_j) = 1$. In other words $z = x_1 y_1 \ldots x_k y_k$, where $x = x_1 x_2 \ldots x_k$ is exactly the sequence of all variables true under T in ascending order. Since T is satisfying, we know that for all i $(1 \leq i \leq m)$ the word x (with $|x| \leq n$) either (i) contains at least one symbol with index in P_i or (ii) does not contain all the symbols with index in N_i. From this it easily follows that x is an element of every C_i $(1 \leq i \leq m)$ and we are done.

(2) $z \in L(r) \Rightarrow C$ is satisfiable:
Let the partitioning of z, by which its membership in $L(r)$ is shown, be $z = x_1 y_1 \ldots x_k y_k$. Thus the word $x = x_1 x_2 \ldots x_k$ is a member of every C_i $(1 \leq i \leq m)$. Thus we can define a truth assignment:

$$T(v_j) = \begin{cases} 1 & \text{if } v_j \in x \\ 0 & \text{if } v_j \notin x\ . \end{cases}$$

T obviously satisfies every clause. ∎

We now prove the NP upper bound.

Lemma 5.4 *MEMBER-*$\{\cup, \cdot, *, \cap, |\}$ *is in NP.*

Proof. Let z be a word in Σ^* and let E be an expression over Σ. We define a "proof" that $z \in L(E)$ recursively as follows. First, if $z = \epsilon$, then the symbol ϵ is a proof of (z, E) if $\epsilon \in L(E)$. In the remaining cases, we assume $z \neq \epsilon$. (i) If $z \in \Sigma$, then z is a proof of (z, z); (ii) if P_1 is a proof of (z_1, E_1), P_2 is a proof of (z_2, E_2), and $z = z_1 \cdot z_2$, then $(z, P_1 \cdot P_2)$ is a proof of $(z, (E_1 \cdot E_2))$; (iii) if P is a proof of (z, E) then P is a proof of $(z, (E \cup E'))$ and of $(z, (E' \cup E))$ for any expression E'; (iv) if P_1 is a proof of (z, E_1) and P_2 is a proof of (z, E_2), then $(z, P_1 \cap P_2)$ is a proof of $(z, (E_1 \cap E_2))$; (v) if P_1 is a proof of (z_1, E_1), P_2 is a proof of (z_2, E_2), and $z \in L(z_1 \,|\, z_2)$, then $(z, P_1 \,|\, P_2)$ is a proof of $(z, (E_1 \,|\, E_2))$; (vi) if $z = z_1 z_2 \ldots z_k$ for some $k \geq 1$ and words $z_i \neq \epsilon$ for $1 \leq i \leq k$, and if P_i is a proof of (z_i, E) for $1 \leq i \leq k$, then $(z, P_1, \ldots, P_k)$ is a proof of $(z, (E^*))$.

Let Q be the relation $Q(z, E, P)$ iff P is a proof of (z, E). The question "$\epsilon \in L(E)$?" can be solved in polynomial time. Since also the question "$z \in L(z_1 \,|\, z_2)$?" can be decided in polynomial time (see Section 2), it is easy to see that Q can be computed in polynomial time. By induction on the structure of E it is not hard to verify that, if P is a proof of (z, E) and $z \neq \epsilon$, then $|P| \leq 2|z||E|$. We illustrate the induction step for case (vi) (star):

$$\begin{aligned}
|P| &\leq |z| + k + 2 + \sum_{i=1}^{k} |P_i| \\
&\leq |z| + k + 2 + \sum_{i=1}^{k} 2|z_i||E| \qquad \text{by induction} \\
&\leq 2|z| + 2 + 2|z||E| \qquad \text{since } k \leq |z| \text{ and } z = z_1 \ldots z_k \\
&\leq 2|z|(|E| + 3) \qquad \text{since } |z| \geq 1 \\
&= 2|z||(E^*)|.
\end{aligned}$$

Now we can write $z \in L(E)$ iff $(\exists P)[Q(z, E, P)]$. It follows that the membership problem belongs to NP. ∎

Lemmas 5.2–5.4 prove Theorem 5.1.

6 Inequivalence for Expressions without Star

There are few natural problems known to be complete in the class Σ_2^p of the polynomial-time hierarchy (Stockmeyer, 1977). In this section, we add another problem to this list by showing that INEQ-$\{\cup, \cdot, |\}$ is Σ_2^p-complete. The proof will make use of the fact that interleaving is powerful enough to simulate addition of positive integers.

Theorem 6.1 *INEQ-$\{\cup, \cdot, |\}$ is Σ_2^p-complete.*

Proof. We will prove this theorem in two parts, first that the problem belongs to Σ_2^p, and then that it is Σ_2^p-hard. Both parts of this proof are similar to the proof that the inequivalence problem for integer expressions is Σ_2^p-complete (Stockmeyer, 1977).

Membership
By induction on the structure of E, it is easy to show that, if E is a $\{\cup, \cdot, |\}$-expression and $z \in L(E)$, then $|z| \leq |E|$. Let the notion of a "proof" and the predicate Q be defined as in the proof of Lemma 5.4. Then we can write:

$$(E_1, E_2) \in \text{INEQ}\{\cup, \cdot, |\} \text{ iff } (\exists z)[(\exists P_1)[Q(z, E_1, P_1)] \leftrightarrow \neg(\exists P_2)[Q(z, E_2, P_2)]].$$

Standard manipulation of quantifiers and Theorem 3.1 of Stockmeyer (1977) imply now that INEQ-$\{\cup, \cdot, |\}$ is in Σ_2^p.

Hardness
We first show that, using certain format requirements, we can simulate addition of vectors of positive integers by interleaving.

Let $a_{i,k}$, for $1 \leq i \leq n+1$ and $1 \leq k \leq m$, be positive integers. For each k, let s_k be the sum of $a_{i,k}$ for $1 \leq i \leq n+1$. Let E be the expression:

$$\begin{aligned}
E \quad = \quad & 1^{a_{1,1}} \cdot b \cdot 1^{a_{1,2}} \cdot b \cdots 1^{a_{1,m}} \cdot b \;| \\
& 1^{a_{2,1}} \cdot b \cdot 1^{a_{2,2}} \cdot b \cdots 1^{a_{2,m}} \cdot b \;| \\
& \ldots \\
& 1^{a_{n+1,1}} \cdot b \cdot 1^{a_{n+1,2}} \cdot b \cdots 1^{a_{n+1,m}} \cdot b.
\end{aligned}$$

Let R be the set of words over alphabet $\{1, b\}$ such that every block of consecutive b's has length at least $n + 1$.

Lemma 6.2 *$L(E) \cap R$ contains the single word* $1^{s_1} \cdot b^{n+1} \cdot 1^{s_2} \cdot b^{n+1} \cdots 1^{s_m} \cdot b^{n+1}$.

Proof. For a word in $L(E)$ the only way to build $n+1$ consecutive b's is to first interleave all leading 1's from all $n+1$ arguments of the interleaving operator (i.e., $1^{a_{1,1}}, 1^{a_{2,1}}, \ldots, 1^{a_{n+1,1}}$), and then all first b's of the arguments, etc. ∎

We now show the desired hardness result by doing a reduction from $(B_2 \cap \mathrm{DNF})$, which is known to be Σ_2^p-hard (Stockmeyer, 1977; Wrathall, 1977). An instance of $(B_2 \cap \mathrm{DNF})$ is a Boolean formula $G(X_1, X_2)$ where X_j ($j = 1, 2$) is a set of variables $\{x_{j1}, x_{j2}, \ldots, x_{jn}\}$, and where G is in disjunctive normal form, i.e., $G = C_1 \vee C_2 \vee \ldots \vee C_m$, where each C_k is a conjunction of literals; the question is whether $\exists X_1 \forall X_2 (G(X_1, X_2) = 1)$. We can assume that a variable and its negation do not both appear in the same clause.

In order to show the Σ_2^p-hardness of INEQ-$\{\cup, \cdot, |\}$ we will construct expressions E_1, E_2, and $\overline{R}$ such that

$$\begin{array}{c} \forall X_1 \exists X_2 (G(X_1, X_2) = 0) \\ \text{iff} \\ L(E_1 \cup \overline{R}) \subseteq L(E_2 \cup \overline{R}). \end{array} \tag{1}$$

Letting R be defined as in Lemma 6.2, the expression $\overline{R}$ will have the properties that $L(\overline{R}) \cap R = \emptyset$ and $L(E_1) - R \subseteq L(\overline{R})$. It is easy to verify that these two properties imply that

$$L(E_1 \cup \overline{R}) \subseteq L(E_2 \cup \overline{R}) \;\text{ iff }\; L(E_1) \cap R \subseteq L(E_2) \cap R.$$

Therefore, to prove (1) it suffices to show

$$\begin{array}{c} \forall X_1 \exists X_2 (G(X_1, X_2) = 0) \\ \text{iff} \\ L(E_1) \cap R \subseteq L(E_2) \cap R. \end{array}$$

Let us first define $\overline{R}$. Since all words in $L(E_1)$ are bounded in length by $M := |E_1|$, the following will do:

$$\overline{R} \;=\; ((b \cup 1 \cup \epsilon)^M \cdot 1 \cup \epsilon) \cdot b \cdot (b \cup \epsilon)^{n-1} \cdot (1 \cdot (b \cup 1 \cup \epsilon)^M \cup \epsilon).$$

Let [...] be 11 if the expression in the brackets is true and 1 if it is false. Let $\overline{[...]}$ be the opposite.

The expression E_1 is now:

$$E_1 = E_{11} \mid E_{12} \mid \ldots \mid E_{1n} \mid (1^{n+1} \cdot b)^m$$

where, for $1 \leq i \leq n$,

$$\begin{aligned} E_{1i} \;=\; (&\overline{[x_{1i} \in C_1]} \cdot b \cdot \overline{[x_{1i} \in C_2]} \cdot b \cdots \overline{[x_{1i} \in C_m]} \cdot b \\ &\cup \overline{[\neg x_{1i} \in C_1]} \cdot b \cdot \overline{[\neg x_{1i} \in C_2]} \cdot b \cdots \overline{[\neg x_{1i} \in C_m]} \cdot b). \end{aligned}$$

Let

$$F = (1 \cup 1^2 \cup 1^3 \cup \ldots \cup 1^{2n}).$$

The expression E_2 is:

$$E_2 = E_{21} \mid E_{22} \mid \ldots \mid E_{2n} \mid (F \cdot b)^m$$

where, for $1 \leq i \leq n$,

$$\begin{aligned} E_{2i} \quad = \quad & ([x_{2i} \in C_1] \cdot b \cdot [x_{2i} \in C_2] \cdot b \cdots [x_{2i} \in C_m] \cdot b \\ & \cup [\neg x_{2i} \in C_1] \cdot b \cdot [\neg x_{2i} \in C_2] \cdot b \cdots [\neg x_{2i} \in C_m] \cdot b). \end{aligned}$$

We now restrict the words in $L(E_1)$ and $L(E_2)$ to be in R. We can use Lemma 6.2 to conclude that all words in $L(E_1) \cap R$ and in $L(E_2) \cap R$ are of the form

$$y = 1^{s_1} \cdot b^{n+1} \cdot 1^{s_2} \cdot b^{n+1} \cdots 1^{s_m} \cdot b^{n+1}.$$

It is useful to write numerical expressions for the numbers s_k, $1 \leq k \leq m$. For words in $L(E_1) \cap R$, the expressions are functions of 0-1 valued variables p_{1i} for $1 \leq i \leq n$. Setting $p_{1i} = 0$ (resp., $p_{1i} = 1$) means that we choose the LHS (resp., RHS) of the union in E_{1i} to produce the corresponding word in $L(E_1) \cap R$. We also interpret [...] as being either 1 or 2 (instead of 1 or 11, respectively). Now $y \in L(E_1) \cap R$ iff there are $p_{1i} \in \{0, 1\}$ such that, for $1 \leq k \leq m$,

$$s_k = \sum_{i=1}^{n} ((1 - p_{1i})\overline{[x_{1i} \in C_k]} + p_{1i}\overline{[\neg x_{1i} \in C_k]}) + (n + 1). \tag{2}$$

The numerical expressions for words in $L(E_2) \cap R$ involve 0-1 valued variables p_{2i} which, as above, indicate whether the LHS or RHS of the union in E_{2i} is used. These expressions also involve variables f_k for $1 \leq k \leq m$, where $1 \leq f_k \leq 2n$ for all k; here f_k indicates which word is taken from the kth occurrence of F in E_2. Now $y \in L(E_2) \cap R$ iff there are $p_{2i} \in \{0, 1\}$ and $f_k \in \{1, 2, \ldots, 2n\}$ such that, for $1 \leq k \leq m$,

$$s_k = \sum_{i=1}^{n} ((1 - p_{2i})[x_{2i} \in C_k] + p_{2i}[\neg x_{2i} \in C_k]) + f_k. \tag{3}$$

We now can, as in (Stockmeyer, 1977), identify four facts about E_1 and E_2. The following terminology is used. If X is a set of variables, an X-*assignment* is an assignment of truth values to the variables in X. We say that an X-assignment α *kills* the clause C_k if either some literal x appears in C_k and x is assigned value *false* by α or some literal $\neg x$ appears in C_k and x is assigned value *true* by α.

(a) For each $y \in L(E_1) \cap R$, $2n + 1 \leq s_k \leq 3n + 1$ for $1 \leq k \leq m$; and there is an X_1-assignment such that, for $1 \leq k \leq m$, $s_k = 3n + 1$ iff the assignment does not kill C_k.

(b) For each X_1-assignment there is a $y \in L(E_1) \cap R$ such that, for $1 \leq k \leq m$, $s_k = 3n+1$ iff the assignment does not kill C_k.

(c) For each $y \in L(E_2) \cap R$ there is an X_2-assignment such that, for $1 \leq k \leq m$, if $s_k = 3n+1$ then the assignment kills C_k.

(d) Let A_2 be an X_2-assignment and y be a word over $\{1,b\}^*$ having the form $1^{s_1} \cdot b^{n+1} \cdot 1^{s_2} \cdot b^{n+1} \ldots$ such that $2n+1 \leq s_k \leq 3n+1$ and $(s_k = 3n+1) \Rightarrow (A_2$ kills $C_k)$ for $1 \leq k \leq m$. Then $y \in L(E_2) \cap R$.

The proofs of (a)–(d) are not difficult. In each case, we must draw a correspondence between a truth assignment and a word y. As just noted, each word corresponds to values for the 0-1 variables p_{1i} or p_{2i}. The correspondence between these variables and the Boolean variables in G is that $p_{ji} = 1$ iff x_{ji} is assigned value *true*. We illustrate this for (a), leaving the other cases to the reader.

Let $y \in L(E_1) \cap R$. Since each expression $\overline{[...]}$ is either 1 or 2, it is obvious that $2n+1 \leq s_k \leq 3n+1$ for all k. Consider the X_1-assignment obtained from y via the p_{1i} as just described. In the sum (2), note that $s_k = 3n+1$ iff each of the first n terms contributes 2 to the sum. Suppose that $x_{1i} \in C_k$. Then $\overline{[x_{1i} \in C_k]}$ has (integer) value 1. Therefore, the ith term contributes 2 to the sum iff $p_{1i} = 1$ iff x_{1i} is *true*. Similarly, if $\neg x_{1i} \in C_k$, then the ith term contributes 2 to the sum iff $p_{1i} = 0$ iff x_{1i} is *false*. It follows that $s_k = 3n+1$ iff C_k is not killed.

Remember that our goal was to show

$$\forall X_1 \exists X_2 (G(X_1, X_2) = 0)$$
$$\text{iff}$$
$$L(E_1) \cap R \subseteq L(E_2) \cap R.$$

Since $G(X_1, X_2) = 0$ iff all clauses are killed, it is easy to prove "only if" from (a) and (d), while "if" follows from (b) and (c). As noted above, this proves (1). Finally, from (1) we have

$$\exists X_1 \forall X_2 (G(X_1, X_2) = 1)$$
$$\text{iff}$$
$$L(E_1 \cup E_2 \cup \overline{R}) \neq L(E_2 \cup \overline{R}).$$

∎

7 Inequivalence for Expressions with Star

The problem NEC-$\{\cup, \cdot, *, \cap\}$ is known to be ESPACE-complete. This was first proved by Hunt (1973a) who also proved that this problem requires space $c^{\sqrt{n/\log n}}$ for some constant $c > 1$. The proof was simplified by Fürer (1980) and the lower bound was improved to c^n. We show in this section that ESPACE-completeness of NEC and INEQ holds also if the intersection operator is replaced by the interleaving operator.

Theorem 7.1 *INEQ-$\{\cup, \cdot, *, |\}$ and NEC-$\{\cup, \cdot, *, |\}$ are ESPACE-complete.*

Proof.

(1) INEQ-$\{\cup, \cdot, *, |\} \in$ ESPACE.
Given $\{\cup, \cdot, *, |\}$-expressions E_1 and E_2 of length at most n, it is possible to build NFA's M_1 and M_2 with $O(2^n)$ states which accept $L(E_1)$ and $L(E_2)$, respectively. The product construction of Section 3 is used to handle interleaving. It is known (and easy to show) that equivalence of NFA's can be decided by a nondeterministic Turing machine within space proportional to the size of the NFA's (see, for example, the proof of Thm. 13.14 in (Hopcroft and Ullman, 1979)).

(2) NEC-$\{\cup, \cdot, *, |\}$ is ESPACE-hard.
Fürer (1980) proves the ESPACE-hardness of NEC-$\{\cup, \cdot, *, \cap\}$ by doing a generic reduction from an exponential-space Turing machine. In a preliminary version of our paper (Mayer and Stockmeyer, 1991), we modified this proof and showed that by adding new format requirements for words describing accepting computations we could simulate the intersection operator by the interleaving operator. One of the referees observed that our method for simulating intersection by interleaving can be used to give a polynomial-time reduction from NEC-$\{\cup, \cdot, *, \cap\}$ to NEC-$\{\cup, \cdot, *, |\}$, thus providing a simpler proof of our result.

We first describe the words having the restricted format useful in the reduction. Let $\Gamma = \{\gamma_1, ..., \gamma_l\}$ be an alphabet. Let c be a symbol not in Γ. If $w = w_1 w_2 \dots w_m$ where $w_i \in \Gamma$ for $1 \le i \le m$, and if k is a positive integer, then

$$w^{(k)} = w_1^k c^k w_2^k c^k ... w_m^k c^k.$$

Also, $\epsilon^{(k)} = \epsilon$. Letting A be any language over Γ, define $A^{(k)} = \{\, w^{(k)} : w \in A \,\}$. Words having the required format are in the set $R^{(k)}$ defined as:

$$R^{(k)} = (\Gamma^*)^{(k)} = ((\gamma_1^k \cup ... \cup \gamma_l^k) c^k)^*.$$

Before giving the reduction in detail, it is instructive to see how interleaving can simulate intersection in the special case $E = F \cap G$, where F and G are ordinary ($\{\cup, \cdot, *\}$) regular expressions. Suppose that E is over the alphabet Γ. Let F' be obtained from F by replacing each occurrence of a symbol $\gamma \in \Gamma$ in F by the expression $\gamma \cdot c$. Define G' similarly in terms of G. We claim that, when restricted to words in $R^{(2)}$, the language described by $F' \,|\, G'$ equals $(L(F \cap G))^{(2)}$. That is, for every $w \in \Gamma^*$,

$$w \in L(F \cap G) \ \text{ iff } \ w^{(2)} \in L(F' \,|\, G').$$

Note first that $w \in L(F)$ iff $w^{(1)} \in L(F')$, and $w \in L(G)$ iff $w^{(1)} \in L(G')$. If $w \in L(F \cap G)$, then $w^{(1)} \in L(F')$ and $w^{(1)} \in L(G')$, from which it is easy to see that $w^{(2)} \in L(F' \,|\, G')$. The other direction is the more interesting one. Assume $w^{(2)} \in L(F' \,|\, G')$. The case $w = \epsilon$ is easy, so assume $w = w_1 w_2 \dots w_m$ where $w_i \in \Gamma$ for all i. Each word in $L(F')$ and $L(G')$ belongs to $R^{(1)}$. Therefore, for some $u_1 c u_2 c \dots u_p c \in L(F')$ and $v_1 c v_2 c \dots v_q c \in L(G')$, where $u_1, \dots, u_p, v_1, \dots, v_q \in \Gamma$,

$$w_1^2 c^2 w_2^2 c^2 \dots w_m^2 c^2 \in L(u_1 c u_2 c \dots u_p c \mid v_1 c v_2 c \dots v_q c).$$

Since $c \notin \Gamma$, it clear that we must have $m = p = q$ and $w_i = u_i = v_i$ for all $1 \le i \le m$. Therefore, $w^{(1)} \in L(F')$ and $w^{(1)} \in L(G')$, so $w \in L(F)$ and $w \in L(G)$.

We now describe the general transformation. Let $I(E)$ denote the number of occurrences of $\cap$ in E. For any $\{\cup, \cdot, *, \cap\}$-expression E over alphabet Γ and any $k \geq I(E) + 1$, the expression $E^{(k)}$ over alphabet $\Sigma = \Gamma \cup \{c\}$ is defined inductively as follows:

$$\begin{aligned}
\epsilon^{(k)} &= \epsilon \\
\gamma^{(k)} &= \gamma^k c^k \quad \text{for } \gamma \in \Gamma \\
(F \cup G)^{(k)} &= (F^{(k)} \cup G^{(k)}) \\
(F \cdot G)^{(k)} &= (F^{(k)} \cdot G^{(k)}) \\
(F^*)^{(k)} &= (F^{(k)*}) \\
(F \cap G)^{(k)} &= (F^{(i)} \mid G^{(j)}) \ \ \text{where } i = I(F) + 1 \text{ and } j = k - i.
\end{aligned}$$

In the last case $E = F \cap G$, note that $j = k - i \geq (I(E) + 1) - (I(F) + 1) = I(G) + 1$, so $G^{(j)}$ is defined.

Important properties of $E^{(k)}$ are stated in the next lemma, following two definitions. If $w \in (\Gamma \cup \{c\})^*$, let $M(w)$ be the maximum length of a subword u of w such that $u \in \Gamma^* \cup \{c\}^*$. For a language A, let $A^+ = A^* - \{\epsilon\}$.

Lemma 7.2 *For every $\{\cup, \cdot, *, \cap\}$-expression E and every $k \geq I(E) + 1$:*

(1) $|E^{(k)}| \leq 8k|E|$,

(2) $L(E^{(k)}) \subseteq (\Gamma^+ c^+)^*$,

(3) $M(w) \leq k$, *for all* $w \in L(E^{(k)})$,

(4) $w \in L(E)$ *iff* $w^{(k)} \in L(E^{(k)})$, *for all* $w \in \Gamma^*$.

Proof. The proof of each statement is by induction on the structure of E. The base cases $E = \epsilon$ and $E = \gamma \in \Gamma$ are all obvious. (In (1), we chose the constant factor 8 so that the base case $E = \gamma$ is true when the expression $\gamma^k c^k$ is written in fully parenthesized form.) The induction step for each statement has four cases. Many of the cases are similar to one another, and most are easy to check. For this reason, in the following we leave the verification of some cases to the interested reader.

(1). We illustrate the induction step for $E = (F \cup G)$. The other cases are virtually identical.

$$\begin{aligned}
|(F \cup G)^{(k)}| &= |(F^{(k)} \cup G^{(k)})| \\
&= |F^{(k)}| + |G^{(k)}| + 3 \\
&\leq 8k|F| + 8k|G| + 3 \quad \text{by induction} \\
&\leq 8k(|F| + |G| + 3).
\end{aligned}$$

(2). It is enough to note that the set $(\Gamma^+ c^+)^*$ is closed under union, concatenation, star, and interleaving.

(3). The induction step for $E = F \cup G$ is obvious. For $E = F \cap G$, it is enough to note that, if $w \in L(u|v)$, then $M(w) \leq M(u) + M(v)$. For the cases concatenation and star, note that if $w = u_1 u_2 \ldots u_m$ where $u_1, \ldots, u_m \in (\Gamma^+ c^+)^*$, then $M(w) = \max_i M(u_i)$.

(4). We first show that $w \in L(E)$ implies $w^{(k)} \in L(E^{(k)})$.

Let $E = F \cdot G$ and $w \in L(E) = L(F \cdot G)$. Then $w = uv$ where $u \in L(F)$ and $v \in L(G)$. Using the induction hypothesis,

$$w^{(k)} = u^{(k)} v^{(k)} \in L(F^{(k)} \cdot G^{(k)}) = L(E^{(k)}).$$

The proof for star is similar to that for concatenation, and the case $E = F \cup G$ is straightforward.

Consider now $E = F \cap G$. If $w \in L(E)$, $i = I(F) + 1$ and $j = k - i$, then $w \in L(F)$ and $w \in L(G)$, so $w^{(i)} \in L(F^{(i)})$ and $w^{(j)} \in L(G^{(j)})$ by induction. If $w = \epsilon$, then $\epsilon \in L(F^{(i)} \,|\, G^{(j)})$ is clear. Letting $w = w_1 w_2 \ldots w_m$ where $w_i \in \Gamma$ for all i,

$$w_1^i c^i w_2^i c^i \ldots w_m^i c^i \in L(F^{(i)}) \quad \text{and} \quad w_1^j c^j w_2^j c^j \ldots w_m^j c^j \in L(G^{(j)}).$$

So

$$w^{(k)} = w_1^{i+j} c^{i+j} w_2^{i+j} c^{i+j} \ldots w_m^{i+j} c^{i+j} \in L(F^{(i)} \,|\, G^{(j)}) = L(E^{(k)}).$$

We now show that $w^{(k)} \in L(E^{(k)})$ implies $w \in L(E)$.

The proof for $E = F \cup G$ is straightforward and is omitted.

Let $E = F \cdot G$, and assume $w^{(k)} \in L(F^{(k)} \cdot G^{(k)})$. So $w^{(k)} = yz$ where $y \in L(F^{(k)})$ and $z \in L(G^{(k)})$. Since $y, z \in (\Gamma^+ c^+)^*$ by (2), it must be that $y = u^{(k)}$ and $z = v^{(k)}$ for some $u, v \in \Gamma^*$ with $w = uv$. By induction, $u \in L(F)$ and $v \in L(G)$, so $w = uv \in L(F \cdot G)$.

The proof for star is similar to that for concatenation.

Let $E = F \cap G$, and assume $w^{(k)} \in L(F^{(i)} \,|\, G^{(j)})$, where $i = I(F) + 1$ and $j = k - i$. If $w = \epsilon$ then $\epsilon \in L(F^{(i)})$ and $\epsilon \in L(G^{(j)})$, so $\epsilon \in L(F)$ and $\epsilon \in L(G)$ by induction. Now suppose $w^{(k)} = w_1^k c^k \ldots w_m^k c^k$. Let y and z be such that $w^{(k)} \in L(y|z)$, $y \in L(F^{(i)})$ and $z \in L(G^{(j)})$. Since $w^{(k)} \in L(y|z)$, $M(y) \leq i$, $M(z) \leq j$, and $k = i + j$, a little thought shows that we must have $y = w^{(i)}$ and $z = w^{(j)}$. For example, the only way to produce the prefix w_1^k of $w^{(k)}$ is to take w_1^i from the front of y and w_1^j from the front of z. Then the only way to produce c^k is to take c^i from y and c^j from z, and so on. Therefore, by induction, $w \in L(F)$ and $w \in L(G)$. ∎

Another way of viewing part (4) of the lemma is that $L(E^{(k)})$, when restricted to words in $R^{(k)}$, is precisely $(L(E))^{(k)}$. The reduction of NEC-$\{\cup, \cdot, *, \cap\}$ to NEC-$\{\cup, \cdot, *, |\}$ transforms E to $E^{(k)} \cup E_0$, where $k = I(E) + 1$ and $L(E_0) = \Sigma^* - R^{(k)}$. To complete the proof, we describe the expression E_0. We can view a word in Σ^* as a concatenation of "blocks", where a block is a maximal length subword of the form σ^i for some $\sigma \in \Sigma$ and some $i \geq 1$. E_0 describes "mistakes" that cause a word not to belong to $R^{(k)}$. We split these into four categories, i.e., $E_0 = \bigcup_{j=1}^4 E_{0j}$. The expression E_{01} (E_{02}) describes the mistake of a block being too short (long), E_{03} takes care of the mistake where not every other block is composed of c's, and E_{04} describes words which start and end wrong:

$$E_{01} = \textstyle\bigcup_{\sigma \in \Sigma} (\epsilon \cup (\Sigma^* \cdot (\Sigma - \{\sigma\}))) \cdot \sigma \cdot (\sigma \cup \epsilon)^{k-2} \cdot (\epsilon \cup ((\Sigma - \{\sigma\}) \cdot \Sigma^*))$$

$$E_{02} = \textstyle\bigcup_{\sigma \in \Sigma} \Sigma^* \cdot \sigma^{k+1} \cdot \Sigma^*$$

$$E_{03} = \Sigma^* \cdot (\Sigma - \{c\})^{k+1} \cdot \Sigma^*$$

$E_{04} = c \cdot \Sigma^* \cup \Sigma^* \cdot (\Sigma - \{c\})$. ∎

Since Fürer (1980) proves a lower bound of 2^{cn} on the space complexity of NEC-$\{\cup, \cdot, *, \cap\}$ and since the transformation above maps an expression E to an expression of length $O(|E|^2)$, the following corollary is immediate.

Corollary 7.3 *There is a constant $c > 1$ such that no deterministic Turing machine with space bound $c^{\sqrt{n}}$ can accept NEC-$\{\cup, \cdot, *, |\}$ or INEQ-$\{\cup, \cdot, *, |\}$.*

By using the coding h of Section 3, Theorem 7.1 and Corollary 7.3 remain true for expressions over an alphabet of size 2.

Acknowledgements: The first author wishes to thank his advisor Paris Kanellakis for the help during this work. We thank David Harel for pointing out the remark following Proposition 4.1, and we are grateful to one of the referees for suggesting the alternate proof of Theorem 7.1.

References

Eilenberg, S. (1974), "Automata, Languages, and Machines," Vol. A, Academic Press, New York.

Fürer, M. (1980), The complexity of the inequivalence problem for regular expressions with intersection, in "Proceedings, 7th International Colloquium on Automata, Languages, and Programming," Lecture Notes in Computer Science, Vol. 85, Springer-Verlag, New York, pp. 234–245.

Hopcroft, J. E., and Ullman, J. D. (1979), "Introduction to Automata Theory, Languages, and Computation," Addison-Wesley, Reading, MA.

Hunt, H. B., III (1973a), The equivalence problem for regular expressions with intersection is not polynomial in tape, Tech. Report TR 73-161, Cornell University, Ithaca, NY.

Hunt, H. B., III (1973b), On the time and tape complexity of languages I, in "Proceedings, 5th ACM Symposium on Theory of Computing," pp. 10–19.

Hunt, H. B., III, Rosenkrantz, D. J., and Szymanski, T. G. (1976), On the equivalence, containment, and covering problems for the regular and context-free languages, *J. Comput. System Sci.* 12, 222–268.

Ibarra, O. H., Palis, M. A., and Chang, J. H. (1985), On efficient recognition of transductions and relations, *Theoret. Comput. Sci.* 39, 89–106.

Kanellakis, P. C., and Smolka, S. A. (1990), CCS expressions, finite state processes, and three problems of equivalence, *Inform. and Comput.* 86, 43–68.

Mansfield, A. (1983), On the computational complexity of a merge recognition problem, *Disc. Appl. Math.* 5, 119–122.

Mayer, A. J., and Stockmeyer, L. J. (1991), Word problems – this time with interleaving, Report RJ8180, IBM Research Division, San Jose, CA, 1991.

Meyer, A. R., and Stockmeyer, L. J. (1972), The equivalence problem for regular expressions with squaring requires exponential space, in "Proceedings, 12th IEEE Symposium on Switching and Automata Theory," pp. 125–129.

Milner, R. (1980), "A Calculus of Communicating Systems," Lecture Notes in Computer Science, Vol. 92, Springer-Verlag, New York.

Milner, R. (1984), A complete inference system for a class of regular behaviors, *J. Comput. System Sci.* 28, 439–466.

Ogden, W. F., Riddle, W. E., and Rounds, W. C. (1978), Complexity of expressions allowing concurrency, in "Proceedings, 5th ACM Symposium on Principles of Programming Languages," pp. 185–194.

Stockmeyer, L. J. (1977), The polynomial-time hierarchy, *Theoret. Comput. Sci.* 3, 1–22.

Stockmeyer, L. J., and Meyer, A. R. (1973), Word problems requiring exponential time, in "Proceedings, 5th ACM Symposium on Theory of Computing," pp. 1–9.

Van Leeuwen, J., and Nivat, M. (1982), Efficient recognition of rational relations, *Inform. Process. Lett.* 14, 34–38.

Wagner, K., and Wechsung, G. (1986), "Computational Complexity," D. Reidel, Dordrecht.

Warmuth, M. K., and Haussler, D. (1984), On the complexity of iterated shuffle, *J. Comput. System Sci.* 28, 345–358.

Wrathall, C. (1977), Complete sets and the polynomial-time hierarchy, *Theoret. Comput. Sci.* 3, 23–33.