THE COMPLEXITY OF THE INEQUIVALENCE PROBLEM FOR REGULAR EXPRESSIONS WITH INTERSECTION

Martin Fürer
Department of Computer Science
University of Edinburgh
Edinburgh, Scotland.

Abstract

The nondeterministic lower space bound $\sqrt{n}$ of Hunt, for the problem if a regular expression with intersection describes a non-empty language, is improved to the upper bound n. For the general inequivalence problem for regular expressions with intersection the lower bound c^n matches the upper bound except for the constant c. And the proof for this tight lower bound is simpler than the proofs for previous bounds. Methods developed in a result about one letter alphabets are extended to get a complete characterization for the problem of deciding if one input-expression describes a given language. The complexity depends only on the property of the given language to be finite, infinite but bounded, or unbounded.

1. Introduction and previous results

Semi-extended regular expressions are built as regular expressions, but in addition they contain the intersection. The problem if two regular-like expressions describe different languages (inequivalence problem) is of interest in connection with pattern-matching algorithms. The inequivalence problem for semi-extended regular expressions has an upper space bound d^n, because it is easy to build a nondeterministic finite automaton with 2^n states, which accepts the language described by a semi-extended regular expression E of length n. A product construction can be used for every intersection.

Hunt [1973] has given a lower space bound $c^{\sqrt{n/\log n}}$ for the inequivalence problem. This theorem which also appeared in Aho, Hopcroft and Ullman [1974], has a pretty complicated proof. The better lower bound $(c^{n/\log n})$ of Stockmeyer [1974] does not change the proof techniques. We give a tight lower space bound (c^n) by a proof, which is easier to understand.

The general method to prove lower bounds for word problems of regular-like expressions introduced by Meyer and Stockmeyer [1972] is to describe computations or their complement by regular-like expressions. Here a computation is a sequence of subsequent ID's (instantaneous descriptions) of a Turing machine. In describing such a computation we have in particular to compare the j^{th} position in an ID with the j^{th} position in the following ID. But especially for regular-like expressions which contain intersections it would be much easier to compare the first position of one ID with the last position of the other ID and so on. This idea was used in

Fürer [1978] to improve the lower bound of the inequivalence problem for star-free expressions. So we change the code of a computation, instead of trying to find a better description of a fixed code.

The method is well illustrated by the following example:

Describe the set $L_n = \{ww \mid w \in \Sigma^* , |w| = n\}$ by a short semi-extended regular expression!

This can easily be done by a semi-extended regular expression of length $O(n^2)$. With a dive-and-conquer approach, we get an expression of length $O(n \log n)$ and our conjecture is: It cannot be done better.

To the contrary it is not hard to describe the set $L'_n = \{w^R w \mid w \in \Sigma^* , |w| = n\}$ by a semi-extended regular expression of linear size:

We define the expression E_n which describes L'_n inductively by

$E_0 = \lambda$

$$E_{i+1} = \Sigma E_i \Sigma \cap \bigcup_{\sigma \in \Sigma} \sigma \Sigma^* \sigma$$

Strictly speaking, the right side of E_{i+1} is not an expression, but a notation to describe an expression. For example, for $\Sigma = \{0,1\}$ the expression E_1 looks more like $((0 \cup 1) \cdot \lambda \cdot (0 \cup 1)) \cap (0 \cdot (0 \cup 1)^* \cdot 0 \cup 1 \cdot (0 \cup 1)^* \cdot 1)$, in fact even with more parenthesis.

Instead of comparing the regular languages described by two expressions, we can take one particular regular language L_0, and decide if a semi-extended regular expression given as input describes this language. If we choose L_0 to be Σ^* we get the problem NEC $(\Sigma, \{\cup, \cdot, *, \cap\})$ (non-empty complement), if L_0 is empty we get the problem NE$(\Sigma, \{\cup, \cdot, *, \cap\})$ (non-empty). In general we get the problem SINEQ(Σ, L_0) (semi-extended inequivalence).

For $|\Sigma| \geq 2$ NEC$(\Sigma, \{\cup, \cdot, *, \cap\})$ has the same complexity as the general inequivalence problem, but NE$(\Sigma, \{\cup, \cdot, *, \cap\})$ is CSL-complete. Hunt [1973] has shown that this problem is POLYSPACE-complete by showing that every nondeterministic Turing machine M needs at least space $c\sqrt{n}$ (for some constant $c > 0$ depending on M) to decide this problem. We get cn for the same bound. This implies a nontrivial lower time bound.

We want to give a characterization of the complexity of SINEQ (Σ, L_0) for every regular L_0. It is easy to see that for finite L_0, the problem has the same difficulty as for L_0 empty, and that L_0 unbounded (see section 2) is as difficult as $L_0 = \Sigma^*$. The easiest case with L_0 infinite but bounded is $L_0 = \{0\}^*$. So we have to deal with regular-like expressions over one letter alphabets. The periodicity of these languages yields polynomial space upper bounds for most of the inequivalence problems. The lower bound for the inequivalence problem for regular expressions over a singleton alphabet is nondeterministic polynomial time by Stockmeyer and Meyer [1973]. It is an open problem to improve this lower bound for semi-

extended regular expressions. But for $|\Sigma| \geq 2$ we have a polynomial space lower bound for SINEQ $(\Sigma,\{0\}^*)$, because it is hard to decide if a semi-extended regular expression describes also words containing other letters than 0.

It is very complicated to show that SINEQ (Σ,L_0) has about the same difficulty for every infinite but bounded regular language L_0. This proof needs other techniques than the proof by Hunt, Rosenkrantz and Szymanski [1976] of the corresponding classification for regular expressions (without intersection).

2. Notation

The notation is mostly the same as in Aho, Hopcroft and Ullman [1974] and Stockmeyer [1974].

$\leq_{\log\text{-}\lin}$-complete means that a transformation can be computed in logarithmic space and the length of the output is linear. CSL-complete means $\leq_{\log\text{-}\lin}$-complete in NSpace $(n+1)$. Hence by Seiferas, Fischer and Meyer [1973] every CSL-complete set really needs linear space to be recognized.

Subsets of the operators $\{\cup,\cdot,*,\cap,\neg,^2\}$ are denoted by ψ. $\{\cup,\cdot,*\}$ are the regular operators: union, concatenation and Kleene star, $\cap$ is the intersection, $\neg$ is the complement relative to Σ^*, and ^{2} is the squaring operator defined by $L(E^2) = L(EE)$. NEC (Σ,ψ) (resp. NE (Σ,ψ)) is the set of those regular-like expressions over the alphabet Σ, built with the operators in ψ, which don't describe Σ^* (resp. ϕ). INEQ (Σ,ψ) is the set of pairs (E,E') of regular-like ψ-expressions which describe different languages (i.e. $L(E) \neq L(E')$). SINEQ (Σ, L_0) is the set of semi-extended regular expressions which don't describe L_0.

Definition: A language is bounded if it is contained in $W_1^* \ldots W_k^*$ for some words $W_1,\ldots,W_k$.

3. The non-empty problem

Theorem 1

NE $(\{0,1\}, \{\cup,\cdot,*,\cap\})$ is CSL-complete.

Proof

a) NE $(\{0,1\}, \{\cup,\cdot,*,\cap\}) \in$ CSL.
This result of Hopcroft is an exercise in Aho, Hopcroft and Ullman [1974, p.424]. In fact the powerset of the parenthesis-positions in a semi-extended regular expression can be used as the set of states of a nondeterministic finite automaton which accepts the language described by the expression. (A state in $(E_1 \cap E_2)$ is determined by a pair of states, one in E_1 and one in E_2.)

b) Given a lba M (linear-bounded automaton = nondeterministic $n+1$ space-bounded Turing machine) with the tape alphabet T, the state set S, the accepting states

F, the start state q_o and the input word x of length $|x| = n$, we construct a semi-extended regular expression which describes the set of accepting computations of M with input x. For this theorem we define a computation as a word of the following kind:

$\& \; ID_1^R \# ID_1 \; \& \; ID_2^R \# ID_2 \; \ldots\ldots \; \& \; ID_{k-1}^R \# ID_{k-1} \; \& \; ID_k^R \# ID_k \; \&$

w^R is the reversed of the word w. $ID_1, ID_2, \ldots, ID_k$ are subsequent instantaneous descriptions $(ID_j \vdash ID_{j+1})$ of the lba M. All ID's have length $n+2$ ($n+1$ tape symbols and 1 state).

Let Σ be the alphabet $S \cup T \cup \{\&,\#\}$. We describe the set of accepting computations by the semi-extended regular expression $E = \bigcap_{i=1}^{4} E_i$.

E_1: Reflection

We define the subexpressions RE_j by

$RE_o = \#$

$RE_{j+1} = \Sigma \; RE_j \Sigma \cap \bigcup_{\sigma \in S \cup T} \sigma \; \Sigma^* \sigma \quad$ for $j \in N \quad N = \{0,1,2,\ldots\}$

$E_1 = \& \; (RE_{n+2} \&)^*$

With E_1 we demand the correct format (positions of the separating symbols & and #) and the reflection at the symbols #.

E_2: Computation Step

We define the subexpression CS_j by

$CS_1 = T \; \& \; T$

$CS_2 = \Sigma \; CS_1 \; \Sigma$

$CS_{j+1} = \Sigma \; CS_j \; \Sigma \cap \bigcup_{Next} \sigma_1\sigma_2\sigma_3\Sigma^*\sigma_6\sigma_5\sigma_4$ for $j = 2,3,\ldots$

Where $Next = \{(\sigma_1\sigma_2\sigma_3, \sigma_4\sigma_5\sigma_6) \mid$ there exist $u,u',v,v' \in (S \cup T)^*$ such that $u\sigma_1\sigma_2\sigma_3 v \vdash u'\sigma_4\sigma_5\sigma_6 v'\}$

$CS = CS_{n+2} \cap (\Sigma^* T \cup (S \cup \Sigma S)\Sigma^* S)$ (No head is allowed to walk in from the left end.)

$E_2 = (\Sigma - \{\#\})^* \# (CS \#)^* (\Sigma - \{\#\})^*$. $L(E_1 \cap E_2)$ is the set of computations (of the lba M) which start and stop with any ID's.

E_3: ID_1 is the Start-ID

$E_3 = (\Sigma - \{\#\})^* \# q_o \; x \; b \; \& \; \Sigma^*$ where x is the input.

E_4: Accepting Computation

$E_4 = \Sigma^* \; F \; \Sigma^*$

where $F \subseteq \Sigma$ is the set of accepting states of the lba M.

Given any lba M, the transformation from an input x to the corresponding semi-extended regular expression $E = \bigcap_{i=1}^{4} E_i$ can be done by a Turing machine with a two-way read-only input tape in linear time without using any space on the work tapes.

In particular the length of E is linear in the length of the input x.

Since the size of Σ is constant (independent of the input x) there is no problem in changing the expression E to an expression which describes a binary encoding of $L(E)$. □

Corollary

NE $(\{0,1\},\{\cup,\cdot,*,\cap\}) \notin$ D Time(cn)

Proof

D Time$(t) \subseteq$ D Space$(t/\log t)$ by Hopcroft, Paul and Valiant [1977]. □

4. The non-empty complement problem

Definition: A marked binary number x is a word (over the four letter alphabet $\{0,\underline{0},1,\underline{1}\}$) described by the regular expression

$$(0\cup 1)^* \, \underline{1}\underline{0}^* \cup \underline{0}^* .$$

Example: $0100\underline{1}\underline{0}\underline{0}$ is a marked binary number of length 7. Note that, if i has the binary representation 0100100 (of length 7), then $i - 1 \bmod 2^7$ has the binary representation 0100011. The representations of i and $i - 1 \bmod 2^7$ differ in the underlined digits. For two marked binary numbers the successor relation can be tested locally.

Theorem 2

NEC $(\{0,1\}, \{\cup,\cdot,*,\cap\})$ is $\leq_{\text{log-lin}}$ - complete in EXPSPACE.

Proof

a) It is well known that NEC $(\{0,1\}, \{\cup,\cdot,*,\cap\})$ is in EXSPACE.

b) Let M be a nondeterministic 2^n-2 space-bounded Turing machine. We choose another code for sequences of subsequent ID's of M, and call this code a computation of M. Here we combine an idea of Stockmeyer [1974] (numbering positions helps to "find" them with regular-like expressions) with an idea of Fürer [1978] (a reflection is easier to describe by regular-like expressions than a translation).

Let $ID_1, ID_2, \ldots, ID_k$ be a sequence of subsequent instantaneous descriptions of the Turing machine M, where ID_1 is the initial ID corresponding to the input x and ID_k is an accepting ID. We define a_{ij} to be the j^{th} symbol in ID_i $(i = 1,2,\ldots,k \; ; \; j = 1,2,\ldots,2^n-1)$.

Let $[j]$ be the marked binary number j of length n and $[j]^R$ be the reversed word of it.

Then we define (for this proof) the word

$$\& [0]^R \neq [0] \& [1]^R \; a_{11} \; [1] \& [2]^R \; a_{12} \; [2] \& \ldots$$
$$\ldots [2^n-1]^R \; a_{1,2^n-1} \; [2^n-1] \& [0]^R \neq [0] \& [1]^R \; a_{21} \; [1] \; \& \ldots\ldots$$
$$\ldots\ldots [2^n-1]^R \; a_{k,2^n-1} \; [2^n-1] \& [0]^R \neq [0] \;\&$$

to be an accepting computation of the Turing machine M with input x.

To define a transformation from the language accepted by M in the set of semi-extended regular expressions which do not describe Σ^*, we map each input x of M in an expression E describing the complement of the accepting computations of the Turing machine M with input x. With our code of computations this can be done in a straightforward way, by enumerating the mistakes which imply that a word is not a computation of M with input x. Each mistake is described by a semi-extended regular expression E_i and $E = \bigcup_{i=1}^{10} E_i$.

The possible mistakes are:

1. Two symbols with distance $n+1$ do not match.

$E_1 = \Sigma^* \; (\bigcup_{\sigma \in \Sigma} \sigma \; \Sigma^n (\Sigma - \text{match}(\sigma))) \Sigma^*$
with match (σ) defined by

σ	match(σ)
$0,\underline{0},1,\underline{1}$	$\{0,\underline{0},1,\underline{1}\}$
&	$S \cup T \cup \{\neq\}$
$\sigma \in S \cup T \cup \{\neq\}$	$\{\&\}$

2. The word does not start correctly.

$E_2 = (\Sigma \cup \lambda)^{n+1} \cup (\Sigma - \{\&\})\Sigma^*$
$\cup \; \Sigma \; (\Sigma \cup \lambda)^{n-1} \; (\Sigma - \{\underline{0}\})\Sigma^*$
$\cup \; \Sigma^{n+1} \; (\Sigma - \{\neq\})\Sigma^*$

So all words in $\Sigma^* - L(E_2)$ start with $\& \; \underline{0}^n \neq$

3. The word does not end correctly. E_3 is symmetric to E_2.

4. A subword of length $\leq 2n+1$ with an a_{ij} or $\neq$ in the middle is not symmetric:

Let $SY_0 = \{\neq\} \cup S \cup T$

$SY_{j+1} = \Sigma \; SY_j \Sigma \; \cup \; SY_0$ for $j \in N$.

Then $E_4 = \Sigma^* \; (SY_n \cap \bigcup_{\sigma \in \Sigma} \sigma \Sigma^* (\Sigma - \{\sigma\}))\Sigma^*$

5. A binary number is not correctly marked.

$E_5 = \Sigma^* \; (S \cup T \cup \{\neq\}) (0 \cup 1 \cup \underline{0} \cup \underline{1})^* [(0 \cup 1)(\underline{0} \cup \&) \cup (\underline{0} \cup \underline{1})(\underline{1} \cup 0 \cup 1)] \Sigma^*$

6. A subword of length $2n+1$ with & in the middle is not of the form $[j]\ \&\ [j+1]^R$ with $0 \le j \le 2^n - 1$ and addition mod 2^n.

$$E_6 = \Sigma^* \left(CN_n \cap \bigcup_{\sigma \in \{0,1,\underline{0},\underline{1}\}} \sigma\, \Sigma^* (\Sigma - \mathrm{succ}(\sigma))\Sigma^*\right)$$

with $CN_0 = \&$

$$CN_{j+1} = \Sigma\ CN_j\ \Sigma \cup CN_0 \quad \text{for } j \in N$$

and $\mathrm{succ}(0) = \mathrm{succ}(\underline{0}) = \{0,\underline{1}\}$

$\mathrm{succ}(1) = \mathrm{succ}(\underline{1}) = \{1,\underline{0}\}$

The remaining mistakes are:

7. $\neq$ or $[0]$ appear not only as a pair $\neq[0]$.

8. The initial ID is wrong.

9. A computation step is wrong, i.e. two triples $a_{ij-1}\ a_{ij}\ a_{ij+1}$ and $a_{i+1,j-1}\ a_{i+1,j}\ a_{i+1,j+1}$ are not in the relation "Next". All pairs of such triples are found, because a_{ij} is to the left of $[j]$, $a_{i+1,j}$ is to the right of $[j]^R$, and there is exactly one $\neq$ between them.

10. No ID is accepting.

The reader can find similar expressions for these mistakes.

As in the previous theorem the length of the expression $E = \cup E_i$ is $O(n)$ and the transformation from an input to the corresponding expression E can be computed on a Turing machine without using the work tapes. □

Corollary

The inequivalence problem for semi-extended regular expressions is $\le_{\text{log-lin}}$-complete in EXPSPACE.

5. Regular-like expressions over a one letter alphabet

Known results are:

a) $\mathrm{INEQ}(\{0\},\{\cup,\cdot,\neg\}) \in P$

b) $\mathrm{INEQ}(\{0\},\{\cup,\cdot,^*\})$ is NP-complete

c) $\mathrm{INEQ}(\{0\},\{\cup,\cdot,^2,\neg\})$ is POLYSPACE-complete

d) $\mathrm{INEQ}(\{0\},\{\cup,\cdot,^*,\neg,^2\}) \in$ EXPSPACE

b) is proved and c) is claimed in Stockmeyer and Meyer [1973]. In both cases, the lower bound is difficult.

d) follows from the fact that there is a bound for the length of the initial segment and the period (of the described ultimately periodic languages) which is only squared with each operation. A double exponential time bound is implicit in Rangel [1974].

We extend the list by

e) NE $(\{0\},\{\cup,\cdot,^*,\cap\})$ is NP-complete

f) INEQ $(\{0\},\{\cup,\cdot,^*,\cap,^2\}) \in$ POLYSPACE

The proof of the upper bound in e) is not presented here. The lower bound is very similar to the lower bound of b). f) is the next theorem.
Open problems suggest themself by the incompleteness of the list.

For regular-like expressions E over a one letter alphabet, let $i(E)$ be the length of the shortest initial segment and let $p(E)$ be the length of the shortest period of the language $L(E)$. I.e. $i(E) \geq 0$ and $p(E) > 0$ are the smallest integers such that for all $j \geq i(E)$ $0^j \in L(E) \Leftrightarrow 0^{j+p(E)} \in L(E)$.

For $i_j \geq i(E_j)$, and p_j a positive multiple of $p(E_j)$ $(j=1,2)$, we define i and p by the following table (depending on E, i_j and p_j).

E	i	p
$E_1 \cup E_2$, $E_1 \cap E_2$	$\max(i_1,i_2)$	$\operatorname{lcm}(p_1,p_2)$
E_1E_2	$i_1+i_2+\operatorname{lcm}(p_1,p_2)$	$\operatorname{lcm}(p_1,p_2)$
E_1^2	$2i_1+p_1$	p_1
E_1^* for $L(E_1)$ infinite	$(i_1+p_1)p_1/p(E_1^*)$	p_1
E_1^* for $L(E_1)$ finite	$i_1^2 / p(E_1^*)$	$p(E_1^*)$

i is an initial segment length and p is a period of $L(E)$ (maybe not the smallest). ($\operatorname{lcm}(s,t)$ is the least common multiple of s and t.)

Lemma 1

For E equal to $E_1 \cup E_2$, $E_1 \cap E_2$, E_1E_2 or E_1^2 we have $i(E) \leq i$ and $p(E) \mid p$.
This Lemma can be checked easily.

Lemma 2

For $E = E_1^*$ and $L(E_1)$ infinite, p is a positive multiple of $p(E_1^*)$ and $i(E_1^*) \leq i$.

Proof

The smallest period $p(E_1^*)$ of $L(E_1^*)$ is the greatest common divisor of all lengths of words in $L(E_1)$. Furthermore $L(E_1^*)$ is contained in $P = \{0^{jp(E_1)} \mid j \in N\}$ and $P - L(E_1^*)$ is finite.

Let 0^k be any word of $L(E_1)$ with $i_1 < k \leq i_1 + p_1$, and let E_2 be $0^k(0^{p_1})^*E_1^*$. Then $L(E_2) \subseteq L(E_1^*) \subseteq P$ and $P - L(E_2)$ is finite too. Therefore $i(E_2) \geq i(E_1^*)$.
So it is enough to show that $i = (i_1+p_1)p_1/p(E_1^*) \geq i(E_2)$. If $0^j \in L(E_2)$ then also $0^{j+p_1} \in L(E_2)$. We want to investigate, where words of new lengths modulo p_1

come to the set $L(E)$.

Let S be the set of new word lengths modulo p_1 in $L(E_2)$, i.e.

$$S = \{j \mid O^j \in L(E_2) \quad \text{but} \quad O^{j-p_1} \notin L(E_2)\}$$

Then S has the properties:

1) $k \in S$ and the difference between any two adjacent elements of the ordered set S is less than $i_1 + p_1$.

2) $|S| = p_1/p(E_1^*)$

To see property 1), assume $j \in S$ and $j \neq k$.

$O^j \in L(E_2)$ and $j \neq k$ imply that there exist j_1, j_2, j_3 with

a) $j = j_1 + j_2 + j_3$

b) $j_1 = k + np_1$ for some $n \in N$

c) $O^{j_2} \in L(E_1^*)$

d) $O^{j_3} \in L(E_1)$ and $j_3 > 0$

$j \in S$ and b) imply $j_1 = k$

$j \in S$ and c) imply $k + j_2 \in S$

$j \in S$ and d) imply $j_3 - p_1 \notin L(E_1)$ and therefore

$j_3 \leq i_1 + p_1$.

Hence we have shown, if $j \in S$ then either $j = k$ or there exists j_2 with $k + j_2 \in S$ and $1 \leq j - (k + j_2) \leq i_1 + p_1$.

Property 2) of S follows from the definition of S, and the fact that modulo p_1 exactly the multiples of $p(E_1^*)$ appear as word lengths in $L(E_2)$.

The properties 1) and 2) of S imply that

$$i(E_2) \leq k + (|S|-1)(i_1 + p_1) \leq |S|(i_1 + p_1) \leq (i_1 + p_1)\, p_1/p(E_1^*)$$

□

Lemma 3

For $E = E_1^*$ and $L(E_1)$ finite $p(E_1^*) \leq \max(i_1, 1)$ and $i(E_1^*) \leq 2i$.

Proof

The Lemma is trivial for $L(E_1) \subseteq \{\lambda\}$. Otherwise choose p_2 such that $0 < p_2 \leq i_1$ and $O^{p_2} \in L(E_1)$. Let E_2 be $(O^{p_2})^* E_1$. Then $L(E_2^*) = L(E_1^*)$, but $L(E_2)$ is infinite with $p(E_2) \leq p_2 \leq i(E_1)$ and $i(E_2) \leq i(E_1)$. Now Lemma 2 is applied to E_2. (A direct proof would give the bound $i = i_1^2 / p(E_1^*)$ (instead of $2i$) for $i(E_1^*)$.

We want to show that $i(E)$ and $p(E)$ are bounded by $2^{c|E|}$. But the star operation makes difficulties in our induction proof. E.g. for $L(E) = \{O^j, O^{j+1}\}$, $i(E^*) \approx (i(E))^2$. We avoid these difficulties by proving bounds for the product $i(E)p(E)$.

Lemma 4

If E is an expression with the operators $\{\cup,\cdot,^*,\cap,^2\}$ over the alphabet $\{0\}$ then

$$p(E) \leq 2^{\frac{1}{2}|E|} \quad \text{and} \quad i(E)p(E) \leq 2^{|E|} .$$

Proof

The induction step for $f(E) \leq c^{|E|}$ is to show for each unary operation op_1 and each binary operation op_2 :

$$f(E_1^{op}) \leq c\, f(E_1)$$

$$f(E_1\ op\ E_2) \leq c\, f(E_1) f(E_2)$$

This is trivial, except for the case "E_1^* for $L(E_1)$ finite", where the induction does not go through and a separate proof is necessary. But this problem has a surprisingly elegant solution.

Every ultimately periodic language L can be represented by three sets P,A,B, where P is periodic, A and B are disjunct finite sets and $L = (P - B) \cup A$. Let $max(A)$ be the length ℓ of the longest word 0^ℓ in L. By induction on the structure of a regular-like expression E (without $\neg$), it is easy to see that $max(A) \leq 2^{|E|}$ (for A corresponding to $L(E)$). Without the operation ^{2}, $max(A)$ is even less than n. □

Problem: Can $max(B)$ grow faster than $2^{|E|}$? Otherwise the upper bound of Theorem 3 would hold for the operation $\neg$ too.

Theorem 3

INEQ $(\{0\}, \{\cup,\cdot,^*,\cap,^2\})$ is in POLYSPACE.

Proof

If $L(E_1) \not\subseteq L(E_2)$ then an integer $k \leq max(i(E_1),i(E_2)) + lcm(p(E_1),p(E_2))$ exists such that $0^k \in L(E_1) - L(E_2)$. By Lemma 4, $k \leq 2^n$ (for $n = |E_1| + |E_2|$).

An $O(n^2)$ time bounded alternating Turing machine (see Chandra and Stockmeyer [1976] or Kozen [1976]) can guess a binary representation of such a k and check if it has this property. The only difficult case is to check for a number p and a subexpression E^* if $0^p \in L(E^*)$. This is done by

```
if p = 0 then accept and stop
guess an exponent e = 2^j < 2p such that 0^p ∈ L((E ∪ λ)^e)
for i := j down to 1 do
        choose universally q with 0 ≤ q ≤ p
        choose universally p := q or p := p - q
if p = 0 then accept and stop
check if 0^p ∈ L(E)
```

Note: To guess q with $0^q \in L(E)$ and $0^{p-q} \in L(E^*)$ needs too much time. □

6. Equivalence to a fixed language

Theorem 4

SINEQ $(\{0,1\}, L_0)$ is

a) CSL-complete for L_0 finite

b) POLYSPACE-complete for L_0 infinite but bounded (Definition in Section 2).

c) $\leq_{\text{log-lin}}$-complete in EXPSPACE for L_0 unbounded.

Proof

Let E_0, $\bar{E}_0$ be semi-extended regular expressions with $L(E_0) = L_0$ and $L(\bar{E}_0) = \{0,1\}^* - L_0$.

From $L(0^*E \cup E_0) \neq L_0 \Leftrightarrow L(E) \neq \emptyset$ and Theorem 1 follows that all three cases (except for $L_0 = \{0,1\}^*$) are CSL-hard. Also the containment problem $L(E) \subseteq L_0$ is CSL-hard.

The EXPSPACE lower bound for L_0 unbounded follows from Theorem 2 and the fact that an unbounded L_0 contains a homomorphic image of $\{0,1\}^*$. This method was used by Hunt, Rosenkvantz and Szymanski [1976] to show that the same problem for regular expressions is CSL-hard.

To get the upper bounds, we use $L(E) \neq L_0 \Leftrightarrow L(E) \not\subseteq L_0 \vee L_0 \not\subseteq L(E)$ and give upper bounds for both containment problems.

From $L(E) \not\subseteq L_0 \Leftrightarrow L(E \cap \bar{E}_0) \neq \emptyset$ and Theorem 1 follows that the first containment problem is in CSL for all L_0. The upper bounds of $L_0 \not\subseteq L(E)$ depend on L_0.

a) $\{E \mid L_0 \not\subseteq L(E)\} \in P$ for L_0 finite by Stockmeyer and Meyer [1973]. To test membership of a word x in the language described by a regular-like expression, a list of all memberships of subwords of x in languages described by subexpressions is built bottom-up.

c) From $L_0 \not\subseteq L(E) \Leftrightarrow L(E \cup \bar{E}_0) \neq \{0,1\}^*$ and Theorem 2 follows the EXPSPACE upper bound.

b) This is by far the most difficult part, and only the main points are sketched here. The bounded language L_0 is contained in $w_1^* \ldots w_k^*$ with $w_i = a_{i1} \ldots a_{i\ell_i}$.

We call $L \cap a_{ij}\, a_{ij+1} \ldots a_{i\ell_i}\, w_{i+1}^* \ldots w_{i'-1}^*\, a_{i'1} \ldots a_{i'j'}$ the (i,j,i',j')-part of the language L. For all subexpressions E' of E, all parts of $L(E')$ are represented by the corresponding set of $(i'-i+1)$-tuples of exponents for $w_i, \ldots, w_{i'}$. These sets of $(i'-i+1)$-tuples are ultimately periodic, and in a way similar to Theorem 3, it is shown that the initial segment length and periods are not too big (only $2^{c|E|\log|E|}$).

Difficult to handle are those operations which involve more than one part of the

language, namely the concatenation and especially the Kleene star. Here Kleene's method of constructing a regular expression from a finite automaton is used to get a survey of all possible decompositions of one part of $L(E^*)$ in finitely many parts of $L(E)$. □

Open problem

What is the complexity of the containment problem $\{E \mid E$ is a semi-extended regular expression with $L_0 \not\subseteq L(E)\}$ for L_0 infinite but bounded? The upper bound is polynomial space, the lower bound is nondeterministic polynomial time.

This problem is closely related to the easier problem about the complexity of $INEQ(\{0\},\{\cup,\cdot,^*,\cap\})$ (also between NP and POLYSPACE).

References

Aho, A.V., J.E. Hopcroft and J.D. Ullman, "The Design and Analysis of Computer Algorithms", Addison-Wesley, Reading, Mass., 1974.

Chandra, A.K. and L.J. Stockmeyer, "Alternation", Proc. 17th Annual IEEE Symposium on Foundations of Comp. Sci., 98-108, 1976.

Fürer, M., "Non-elementary lower bounds in automata theory", (in German), Diss. ETH 6122 (Ph.D. Thesis), Zürich, 1978.

Hopcroft, J.E., W.J. Paul and L.G. Valiant, "On Time Versus Space", Journal of the Association for Computing Machinery 24, 332-337, 1977.

Hunt III, H.B., "The equivalence problem for regular expressions with intersection is not polynomial in tape", Tech. Report TR 73-161, Dept. of Computer Science, Cornell University, 1973.

Hunt III, H.B., D.J. Rosenkrantz and T.G. Szymanski, "On the Equivalence, Containment, and Covering Problems for the Regular and Context-Free Languages", J. of Computer and System Sciences 12, 222-268, 1976.

Kozen, D., "On parallelism in Turing machines", Proc. 17th Annual IEEE Symposium on Foundations of Comp. Sci., 89-97, 1976.

Meyer, A.R. and L.J. Stockmeyer, "The equivalence problem for regular expressions with squaring requires exponential space", Proc. 13th Annual IEEE Symposium on Switching and Automata Theory, 125-129, 1972.

Rangel, J.L. "The Equivalence Problem for Regular Expressions over One Letter is Elementary", 15th Annual Symposium on Switching and Automata Theory, 24-27,1974.

Sieferas, J.I., M.J. Fischer and A.R. Meyer, "Refinements of the Nondeterministic Time and Space Hierarchies", Proc. 14th Annual IEEE Symposium on Switching and Automata Theory, 130-137, 1973.

Stockmeyer, L.J., "The complexity of decision problems in automata theory and logic". Report TR-133, M.I.T., Project MAC, Cambridge, Mass., 1974.

Stockmeyer, L.J. and A.R. Meyer, "Word Problems Requiring Exponential Time: Preliminary Report", Proc. 5th Annual ACM Symposium on the Theory of Computing, 1-9, 1973.