THE COMPLEXITY OF REGULAR EXPRESSIONS WITH GOTO AND BOOLEAN VARIABLES

by

Karl Abrahamson

TECHNICAL REPORT 82-3

March 1982

ABSTRACT

Regular expressions can be extended by adding gotos and Boolean variables. Although such extensions do not increase the class of expressible languages, they do permit shorter expressions for some languages. The output space complexity of eliminating Boolean variables is shown to be double exponential. The complexity of eliminating a single goto from a regular expression is shown to be $\Omega(n \log n)$, a surprising result considering that n gotos can be eliminated in single exponential space.

1. INTRODUCTION

It has long been recognized that the basic control structures of structured programs — while-do, if-then-else, sequencing — are weak. From the beginning, those control mechanisms have been augmented by a) gotos (where needed), and b) assignments and tests of Boolean variables (fairly freely). It is natural to ask how much power gotos and Boolean variables add to structured programs.

Regular expressions are a convenient abstraction of structured programs. We view a program as a language, whose alphabet is the instruction set, and whose members represent the sequences of instructions which the program might execute. It is possible to add gotos and Boolean variable assignments and tests to regular expressions in a natural way. Then we can ask how difficult it is to eliminate them.

Regular expressions afford several advantages over the usual structured programs for theoretical studies. An obvious one is that we can restrict our study to languages, without being concerned with the semantics which lies behind structured programs. But at least as compelling is that the extensions we are interested in — gotos and Boolean variables — can in fact be eliminated from regular expressions. In contrast, it is not always possible to eliminate gotos from a structured program without adding variables (see Greibach, p. 4-62), or, dually, to eliminate Boolean variables without adding gotos. It appears to be the nondeterminism of regular expressions — viewed as the "backtrack" variety, as opposed to the "irrevocable choice" of Dijkstra's guarded commands — which gives regular expressions their added power. Indeed,

we can define an expression E to be deterministic if checking $s \in L(E)$ never requires that we backtrack or remember more than one position in E. It can be shown that the deterministic expressions with gotos represent all regular languages, while $(ab)^*(a \cup b)$ is equivalent to no ordinary deterministic expression.

The results presented in this paper all have a strong form, namely that the problem of eliminating a given extension (gotos or variables) is of a given complexity, even when the input is constrained to be deterministic, while the output need not be. However, for simplicity, we ignore determinism, explicitly stating and proving only the form with unrestricted input. The reader should have no difficulty modifying the proofs for deterministic inputs.

The first problem we consider is that of eliminating gotos from regular expressions. It is well known that gotos can be eliminated at the cost of an exponential length blowup, by converting the expression to a finite automaton, where gotos are nothing special, and then converting the finite automaton to an equivalent regular expression, by standard techniques. Conversely, a result of Ehrenfeucht and Zeiger implies that gotos cannot always be eliminated at polynomial cost.

We present a new result concerning elimination of a single goto from an expression. Although intuition suggests that one goto can be eliminated in linear space - which would be consistent with an exponential space requirement to eliminate all gotos, one at a time - we show that the worst case size increase incurred in eliminating a single goto from a size n regular expression is at best $\Omega(n \log n)$, and at worst $O(n^2)$. The preceding results can be found in section 4.

In section 5 we look at the question of whether changing the names of alphabetic symbols, by using a string to represent each symbol, can make a language easier to represent by a regular expression. That is to say, is there

a homomorphism h such that, although L is hard to represent, h(L) is easy? h is required to be one-one, so that any string is recoverable from its image under h, permitting us to imagine a post-processor following the expression, whose output is just the members of L. In some cases, h(L) is far easier to represent than L. But in others no one-one homomorphism seems to help; we can prove that if neither of $h(\sigma)$ and $h(\delta)$ is a prefix or suffix of the other, for $\sigma \neq \delta$, then, for a certain language X, h(X) is no easier than X. The central result, the basis of all of our lower bounds for Boolean variable elimination, can be described as an extension of a result of Ehrenfeucht and Zeiger concerning a hard language K for regular expressions. We extend their theorem to certain homomorphic images of K.

By encoding each of the n^2 symbols of K as a sequence of 0's and 1's, we can find a size O(log n) extended expression for h(K), while, by our theorem, h(K) remains exponentially difficult for ordinary expressions.

Results concerning elimination of variables are presented in section 6. They are based on the main result of section 5. For the case of eliminating O(n) variables from a size n expression, both upper and lower bounds are double exponential in n. But for eliminating only one variable, the bounds differ greatly; the lower bound is quadratic, although the author knows of no polynomial space method of eliminating a single Boolean variable.

We begin with a description of extended expressions, and some further preliminary definitions and results.

2. EXTENDED EXPRESSIONS

The syntax and semantics of extended expressions is given in Table 1. a is an alphabetic symbol, x is a variable name, and A and B are extended

expressions. A completely formal semantics is complex, and serves no purpose for this paper. Instead, we give the semantics in terms of side effects, either on the current location in the expression, in the case of gotos, or on an auxiliary storage. It should be emphasized that the tests x? and $\bar{x}$? are dynamic, testing the current value of x. Thus a given instance of x? may be λ at one encounter and ϕ at another. ϕ should be thought of as a fence, implying that a wrong choice was made at some previous time.

The size measure size (E) is the number of alphabetic symbols, gotos, destinations and variable operations in E. In some cases we also refer to the length of E, that being the total number of characters used to write E.

Two expressions are equivalent if they represent the same language.

Example 2.1. $G_1(a \cdot d_1 \cdot b)^*$ has size 4 and is equivalent to $b \cdot (a \cdot b)^*$.

Example 2.2. $x\uparrow \cdot y\uparrow \cdot (y? \cdot E \cdot (x? \cdot x\downarrow \cup \bar{x}? \cdot y\downarrow))^* \cdot \bar{y}?$ has size 8 + size (E), and is equivalent to $E \cdot E$.

We now define the function COST(n,g,v), which it is our goal to bound. Let ERE(n,g,v) be the class of extended regular expressions of length at most n, containing at most g gotos and at most v different variables. An ordinary expression contains no gotos or variables. Let COST(n,g,v) be the maximum over ERE(n,g,v) of the size of the smallest equivalent ordinary regular expression. Our bounds on COST(n,g,v) are summarized below.

Expression E	Size S(E)	Language L(E)	side effects/comments
ϕ	0	ϕ	empty set
λ	0	$\{\lambda\}$	empty string
a	1	$\{a\}$	
$x\uparrow$	1	$\{\lambda\}$	set x = true
$x\downarrow$	1	$\{\lambda\}$	set x = false
$x?$	1	ϕ if x = false, $\{\lambda\}$ if x = true	
$\bar{x}?$	1	$\{\lambda\}$ if x = false, ϕ if x = true	
d_i	1	$\{\lambda\}$	
G_i	1	$\{\lambda\}$	jump to d_i
$A \cup B$	$S(A) + S(B)$	$L(A) \cup L(B)$	
$A \cdot B$	$S(A) + S(B)$	$L(A) \cdot L(B)$	concatenation
A^*	$S(A)$	$L(A)^*$	Kleene closure

Table 1

1. $\Omega(2^{\sqrt{n/2}}) \leq \mathrm{COST}(n,n,0) \leq 6\cdot 4^{2n-2}$

2. $\Omega(n \log n) \leq \mathrm{COST}(n,1,0) \leq O(n^2)$

3. $(n/cm)^{2^{2^{\frac{m}{2}-1}}} \leq \mathrm{COST}(n,0,m) \leq 6\cdot 4^{2n2^m-2}$ for some constant c.

4. $\mathrm{COST}(n,0,n/16) \geq 2^{2^{2^{n/16-1}}-1}$

5. $\mathrm{COST}(n,0,1) \geq n^{2-\varepsilon}$ for every $\varepsilon > 0$, and infinitely many n.

Results 1 and 2 can be found in section 4. Results 3, 4 and 5 are proved in section 6, based on the results of section 5.

3. PRELIMINARIES

This section presents some definitions and results concerning ordinary regular expressions and their relationships to graphs.

Labelled Graphs

An arc-labelled graph is a 4-typle (V,E,Σ,Λ), where V is a set of vertices, $E \subseteq V \times V$ is a set of directed arcs, Σ is a set of labels, and $\Lambda: E \to \Sigma$ assigns a label to each arc. Arc-labelled graphs are hereafter referred to simply as graphs.

A _path_ from u to v is a sequence $u = w_0, w_1, \ldots, w_k = v$, where $(w_{i-1}, w_i) \in E$ for $i = 1,\ldots,k$. The _trace_ of path $p = w_0, w_1, \ldots, w_k$ is the string $\Lambda(w_0,w_1)\Lambda(w_1,w_2)\ldots\Lambda(w_{k-1},w_k)$. The set _traces_ (G,u,v) is the set of all traces of paths in G from u to v.

A graph is (forward) deterministic if no two distinct arcs with the same initial vertex have the same label. A graph is backward deterministic if its transpose is deterministic.

Lemma 3.1. For every size $n > 0$ ordinary expression E there is a 2n vertex graph g with vertices u and v such that L(E) = traces(G,u,v).

The proof of lemma 3.1 is by standard finite automaton techniques, and is omitted. (See Aho, Hopcroft and Ullman, p. 318).

Covering and Index

The definitions of covering and index, as well as lemma 3.2, are from Ehrenfeucht and Zeiger (1976), with minor modifications.

Expression E covers string s iff s is a contiguous substring of some member of L(E). Let s^k denote $s \cdot s \cdot \ldots \cdot s$ (k times). The index $I_s(E)$ of s in E is defined as

0 if E does not cover s,

k if E covers s^k but not s^{k+1},

∞ if E covers s^k for all k.

If $I_s(E) \neq \infty$ then E is said to be s-finite.

Lemma 3.2. If $s \neq \lambda$ and E is s-finite, then $I_s(E) \leq 2 \cdot \text{size}(E)$.

Proof. Let n = size(E). Construct the 2n-vertex graph G of lemma 3.1 for E. Suppose there is a path p in G whose trace is s^k, and which is a subpath of some path from u to v. Then either p has length at most $2n \cdot \text{size}(s)$ (and so $k \leq 2n$) or p reaches the same vertex twice, at the same position in s. In the latter case, a pumping argument shows that G, and so E, is s-infinite. □

Normality

The definition of normality is from Ehrenfeucht and Zeiger. Expression

E is normal w.r.t. graph G provided there are functions i and f from subexpressions of E to vertices of G such that, for every subexpression F of E,

1) $L(F) \subseteq traces(G,i(F),f(F))$;
2) if $F = A \cup B$, then $i(F) = i(A) = i(B)$ and $f(F) = f(A) = f(B)$;
3) if $F = A \cdot B$, then $i(F) = i(A)$, $f(A) = i(B)$ and $f(B) = f(F)$;
4) if $F = A^*$, then $i(F) = i(A) = f(A) = f(F)$. In this case $i(F)$ is said to be the base point of F.

A graph G is normal, if, whenever $L(E) \subseteq traces(G,u_E,v_E)$ for some vertices u_E and v_E, E is normal w.r.t. G.

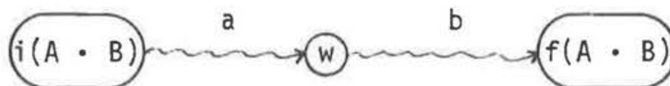
Lemma 3.3. If G is forward and backward deterministic, then G is normal.

Proof. The functions i and f are constructed inductively on the structure of E, from longer to shorter subexpressions.

Basis: Let $i(E) = u_E$ and $f(E) = v_E$.

$A \cup B$: Given $i(A \cup B)$ and $f(A \cup B)$, let $i(A) = i(B) = i(A \cup B)$ and $f(A) = f(B) = f(A \cup B)$.

$A \cdot B$: Given $i(A \cdot B)$ and $f(A \cdot B)$, let $i(A) = i(A \cdot B)$, $f(B) = f(A \cdot B)$. It remains to find $w = f(A) = i(B)$. Assume w.l.g. that $L(A)$ and $L(B)$ are nonempty, and $a \in L(A)$, $b \in L(B)$. Then G contains the following subgraph.



By forward determinism at $i(A \cdot B)$, w is independent of the choice of B. By backward determinism at $f(A \cdot B)$, w is independent of the choice of a. Let $f(A) = i(B) = w$.

A*: Let $i(A) = i(A^*)$ and $f(A) = f(A^*)$. We must show that $i(A^*) = f(A^*)$. Assume the contrary. Then L(A) contains a nonempty string s which traces a path from i(A) to f(A), and $s, s \cdot s \in L(A^*)$. G must contain the following subgraph.

By forward determinism, no other path for $s \cdot s$, starting at $i(A^*)$, is possible. By backward determinism, $i(A^*) = f(A^*)$.

□

4. ELIMINATING GOTOS

We begin by reviewing known results concerning the difficulty of eliminating all of the gotos from an expression. Then we prove lower ($\Omega(n \log n)$) and upper ($O(n^2)$) bounds on the space required to eliminate a single goto, in the worst case.

Multiple Gotos

The following procedure eliminates the gotos from a size n expression E.

1. Apply lemma 3.1 to find 2n vertex graph G for E, treating gotos and destinations as if they were alphabetic symbols.
2. Convert G to a standard graph G' whose traces from u to v are just L(E), by altering goto and destination arcs in the obvious way. G' has 2n vertices.

3. Convert G' to an ordinary expression by the standard technique. The expression has size $6 \cdot 4^{2n-2}$ (see Ehrenfeucht and Zeiger (1976)).

Thus we have

Theorem 4.1. $COST(n,n,0) \leq 6 \cdot 4^{2n-2}$.

A lower bound on COST(n,n,0) can be obtained as follows. Let K_n be the complete graph of n vertices, each arc bearing a distinct label. Ehrenfeucht and Zeiger prove the following. □

Theorem 4.2. There is a path p in K_n such that, if E is any ordinary regular expression which is normal w.r.t. K_n, and E covers the trace of p, then $size(E) \geq 2^{n-1}$.

Theorem 4.2 is a special case of theorem 5.2; hence its proof is deferred.

Theorem 4.3. $COST(n,n/2,0) \geq \Omega(2^{\sqrt{n/2}})$.

Proof. Consider the language $L = traces(K_m,0,0)$. We construct an extended expression E whose language is L. Let a_{ij} be the label of arc (i,j).

$$F_i = d_i \cdot (\bigcup_{j=0}^{m-1} a_{ij} \cdot G_j), \qquad i \neq 0,$$

$$F_0 = d_0 \cdot (G_m \cup \bigcup_{j=0}^{m-1} a_{ij} \cdot G_j),$$

$$E = G_0(\bigcup_{j=0}^{m-1} F_j) \cdot d_m.$$

E has size $2m^2 + m + 3$. If E' is an ordinary expression equivalent to E, then E' is normal w.r.t. K_m by lemma 3.3, so size (E') $\geq 2^{m-1}$ by theorem 4.2. Choosing $2m^2 + m + 3 \leq n < 2(m+1)^2 + (m+1) + 3$ gives the desired result. □

Remark. The above proof assumes an alphabet of at least m^2 symbols. By making use of the stronger theorem 5.2, it can be shown that COST(n,n,0) $\geq \Omega(2^{\sqrt{n/c}})$ for some $c > 0$, even for a two element alphabet.

Eliminating a Single Goto

The remainder of this section is concerned with the difficulty of eliminating one goto from a regular expression, and is independent of later sections. It seems reasonable that it should be no more difficult to eliminate all of the O(n) gotos from a size n expression one at a time, than to do so en masse. Then, according to our exponential space method of eliminating all gotos, it should be possible to eliminate each one at the cost of a constant factor length increase.

Whether or not there is an algorithm for eliminating one goto which, over the course of eliminating all of the gotos from an expression, averages a constant factor increase per goto eliminated, I do not know. But it is shown below that it is not possible to uniformly eliminate one goto in linear space.

Theorem 4.4. COST(n,1,0) $\geq \Omega(n \log n)$.

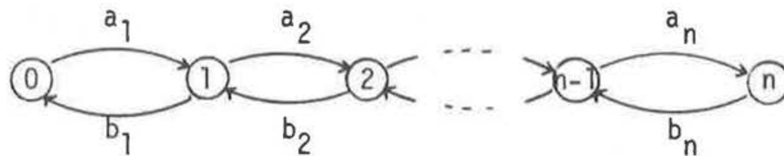
Proof. Consider expression E_n defined by

$$F_0 = d,$$

$$F_n = (a_n \cdot F_{n-1} \cdot b_n)^* \qquad \text{for } n > 0,$$

$$E_n = G \cdot F_n,$$

where G is the single goto, and d is its destination. For example, $E_3 = G(b_3(b_2(b_1da_1)^*a_2)^*a_3)^*$. The reader can check that E_n represents exactly the traces from vertex 0 to vertex n in graph G_n, drawn below.



Graph G_n

If we can show that every ordinary regular expression whose language is traces(G_n,0,n) has size $\Omega(n \log n)$, then we are finished, for E_n has only linear size. Note that the choice of start and stop vertices is critical; indeed, F_n is a linear size expression for the traces in G_n from vertex 0 to itself.

<u>Claim</u>. If E is an ordinary regular expression and L(E) = traces(G_n,0,n), then size(E) $\geq$ n log(n + 2).

<u>Proof of Claim</u>. The proof is by induction on n. The cases n = 0,1 are trivial.

Because each arc has a distinct label, it is possible to equate traces and paths, which is done in what follows. Let p be the loop path $a_1 a_2 \ldots a_n b_n \ldots b_1$. Clearly E is p-infinite. From among the finitely many subexpressions of E, choose a minimal length one F which is p-infinite. Two facts about F are apparent: 1) size(F) $\geq$ 2n, and 2) F is a star, say F = A*. By its forward and backward determinism, G_n is normal, and A* has a base point w. Clearly, A* can neither be entered nor exited except by passing through w. But E must allow both for the possibility of following any arbitrarily long path among vertices 0,...,w-1 before entering w for the first time, and for following any path among w+1,...,n after leaving w for the last time. It is clear that A* cannot be used in covering those pre-w or post-w paths, and in fact that if B is obtained from E by replacing A* by λ, and replacing each occurrence of a_j and b_j by λ, for $j \geq w$, then B represents just the paths from 0 to w-1 in G_{w-1}. By induction, size(B) $\geq$ (w-1)log(w+1). Similarly, B', obtained by replacing A* by λ, and a_j and b_j by λ, for $j \leq w$, represents just the paths from w+1 to n, which is isomorphic to the paths from 0 to n-w-1 in G_{n-w-1}. By induction, size(B') $\geq$ (n-w-1)log(n-w+1). But the occurrences of alphabetic symbols in A*, B and B' are disjoint, so

$$\begin{aligned} \text{size}(E) &= \text{size}(A^*) + \text{size}(B) + \text{size}(B'), \\ &\geq 2n + (w-1)\log(w+1) + (n-w-1)\log(n-w+1). \end{aligned}$$

Size(E) is minimized for w = n/2, giving

$$\begin{aligned} \text{Size}(E) &\geq 2n + 2(\tfrac{n}{2}-1)\log(\tfrac{n}{2}+1), \\ &\geq n\log(n+2) \qquad \text{for } n \geq 2. \qquad \square \end{aligned}$$

A single goto can be eliminated from an expression at the cost of a quadratic size increase, as the following construction shows. We begin by defining expressions FIRST(E) and LAST(E). FIRST(E) is defined only when E contains exactly one goto, and represents all strings which can take E from its start to its goto. LAST(E) is defined only when E contains exactly one destination symbol, and represents the strings generated by E if E is started at its destination symbol. We define FIRST(E) and LAST(E) inductively on the structure of E, leaving it up to the reader to verify the properties claimed for them.

$$\mathrm{FIRST}(G) = \lambda;$$

$$\mathrm{FIRST}(A \cup B) = \begin{cases} \mathrm{FIRST}(A) & \text{if } A \text{ contains } G, \\ \mathrm{FIRST}(B) & \text{if } B \text{ contains } G; \end{cases}$$

$$\mathrm{FIRST}(A \cdot B) = \begin{cases} \mathrm{FIRST}(A) & \text{if } A \text{ contains } G, \\ A \cdot \mathrm{FIRST}(B) & \text{if } B \text{ contains } G; \end{cases}$$

$$\mathrm{FIRST}(A^*) = (A')^* \cdot \mathrm{FIRST}(A);$$

where A' is obtained from A by replacing G by ϕ.

$$\mathrm{LAST}(d) = \lambda;$$

$$\mathrm{LAST}(A \cup B) = \begin{cases} \mathrm{LAST}(A) & \text{if } A \text{ contains } d, \\ \mathrm{LAST}(B) & \text{if } B \text{ contains } d; \end{cases}$$

$$\mathrm{LAST}(A \cdot B) = \begin{cases} \mathrm{LAST}(A) \cdot B & \text{if } A \text{ contains } d, \\ \mathrm{LAST}(B) & \text{if } B \text{ contains } d; \end{cases}$$

$$\mathrm{LAST}(A^*) = \mathrm{LAST}(A) \cdot A^*.$$

It is easily checked that length $(\mathrm{FIRST}(E)) = O(\mathrm{length}(E)^2)$, and similarly for LAST. The goto-free expression ELIM(E) equivalent to E is then defined as follows. Trivial cases are left out. Let A/ϕ be obtained from A by replacing G by ϕ.

A $\cup$ B. Assume w.l.g. that A contains G and B contains d. Then

$$ELIM(A \cup B) = (A/\phi) \cup B \cup FIRST(A) \cdot LAST(B).$$

A $\cdot$ B. Case 1: A contains G, B contains d. Then

$$ELIM(A \cdot B) = (A/\phi) \cdot B \cup FIRST(A) \cdot LAST(B).$$

Case 2: A contains d, B contains G. Then

$$ELIM(A \cdot B) = A \cdot (FIRST(B) \cdot LAST(A))^* \cdot (B/\phi)$$

A*. $$ELIM(A^*) = ELIM(A)^*$$

The reader can check that ELIM(E) is equivalent to E, and length(ELIM(E)) = $O(\text{length}(E)^2)$. When all redundant symbols are eliminated, length and size are related by a constant factor. We have shown

Theorem 4.5. $COST(n,1,0) \leq O(n^2)$. □

5. RECODING

We have seen that the complete graph K_n has a useful property for studying the power of regular expressions, namely that every ordinary expression whose language is traces(K_n,0,0) has size at least 2^{n-1}. If it were possible to find a size O(log n) expression with Boolean variables for the traces in K_n, then we would immediately have a double exponential lower bound on the space required to eliminate all of the variables from an expression.

Unfortunately, K_n has n^2 different labels, killing all hope of any kind of expression with size less than n^2 for traces $(K_n,0,0)$. But suppose each arc is labelled not by a distinct symbol, but by a distinct binary number; that is, a string of 0's and 1's. There is hope of finding a very small expression with Boolean variables for the traces in such a graph. On the other hand, it is not clear that the traces in that graph remain hard for ordinary expressions. Indeed, it is quite possible that the recoding completely invalidates the 2^{n-1} lower bound of theorem 4.2, as the following observation shows.

Let h be the one-one homomorphism which maps symbol a_{ij} to string $b_i \cdot b_j$, over alphabet $\{b_0,\ldots,b_{n-1}\}$. Let L = traces $(K_n,0,0)$. By theorem 4.2, no ordinary regular expression for L has size less than 2^{n-1}. But

$$E = b_0 (\bigcup_{i=0}^{n-1} b_i b_i)^* b_0 \cup \lambda$$

is a size 2n + 2 ordinary expression for h(L). Thus, recoding each symbol by a distinct string permits ordinary expressions to economically describe the traces in K_n.

The preceding observation makes two suggestions:

1) regular expressions may not be quite as weak as suggested by theorem 4.2, in the sense that all that is required for them to be economical, at least in some cases, is a simple renaming of symbols;
2) any lower bound proof which relies on homomorphically recoding K_n will have to take care in how the recoding is done.

Later in this section we show that interpretation (1) is probably not warranted, by exhibiting a language which is hard for regular expressions, and

remains hard even under any homomorphism in a reasonably broad class, namely the class of homomorphisms h for which neither $h(\sigma)$ nor $h(\delta)$ is a prefix or suffix of the other whenever $\sigma \neq \delta$. In fact, the language we exhibit is itself the image of traces $(K_n,0,0)$ under just such a homomorphism.

K_n-Expansions

Our goal is to generalize theorem 4.2 to a larger class of graphs, by distilling from K_n those properties which seem to be required by Ehrenfeucht and Zeiger's proof. Those properties are listed below, as properties of K_n-expansions.

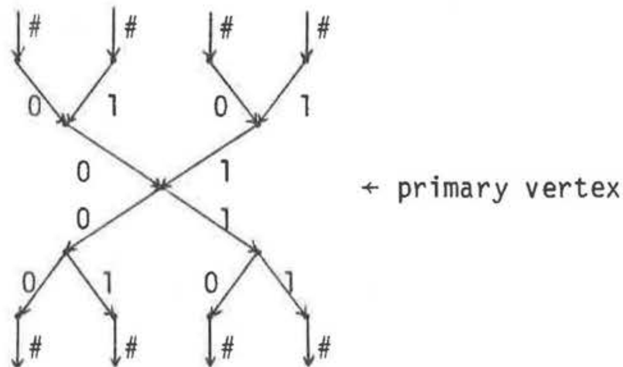
Definition. A K_n-expansion G is a 5-typle (V,P,E,Σ,Λ), where (V,E,Σ,Λ) is a labelled graph, and $P \subseteq V$ is a set of exactly n primary vertices. In addition, G must satisfy

1. Every subgraph of G, including G itself, is normal.
2. Every loop path is uniquely determined by its trace; that is, $t \in$ traces $(G,u,u) \Rightarrow t$ is the trace of exactly one path in G.
3. For every pair of primary vertices $u,v \in P$, there is a path $p = (u=u_0,u_1,\ldots,u_{k-1},u_k=v)$ in G, where none of $u_1,\ldots,u_{k-1}$ are in P. Path p is called a primitive path from u to v;
4. For every non-primary vertex u, there is a primary vertex v such that every loop path from u to itself passes through v.

Note that K_n is itself a K_n-expansion, with no non-primary vertices. There are other K_n-expansions as well.

Example 5.1. Let $\bar{u}$ be the log n bit binary representation of u. We have seen

that by coding a_{ij}, the label on arc (i,j), by $\bar{i}$ # $\bar{j}$, the traces in K_n become easy for regular expressions. But suppose the order is reversed, with a_{ij} coded as $\bar{j}$ # $\bar{i}$. That recoding maps the traces in K_n into the traces in a certain K_n-expansion C_n. For n a power of 2, C_n has n primary and 2n(2n - 2) non-primary vertices. One of those primary vertices and its surrounding non-primary vertices is drawn below for n = 4.



The #-arcs are connected in such a way that there is a path with trace $\bar{j}$ # $\bar{i}$ from primary vertex i to primary vertex j, $i,j \in \{0,1,2,3\}$. This results in at most one #-arc entering or leaving any vertex. C_n is forward and backward deterministic, and hence is normal. Property 2 follows from the fact that every loop unambiguously names the primary vertices visited. Other properties of K_n-expansions are simple to check.

Definition. If G is a K_n-expansion, say that H is a K_m-subexpansion of G iff

1. H is a K_m-expansion;
2. H is a subgraph of G;
3. the primary vertices of H are a subset of those of G.

Note that if G is a K_n-expansion and H is obtained from G by deleting any k primary vertices and associated arcs, then H is a K_{n-k}-subexpansion of G.

We are now ready to prove the main theorem about K_n-expansion, generalizing theorem 4.2. The proof closely follows the lines of Ehrenfeucht and Zeiger's proof of theorem 4.2, but of course requires more careful consideration on certain points, in particular on the difference between traces and paths.

Theorem 5.2. Let G be a K_n-expansion, H be a K_m-subexpansion of G, and v be any primary vertex of H. There is a loop path p in H from v to itself such that, if E is any ordinary regular expression covering p, and E is normal w.r.t. G, then size (E) $\geq 2^{m-1}$.

Proof. The proof is by induction on m for $m \leq n$. The case m = 1 is trivial. Assume m > 1. Let $\oplus$ and $\ominus$ denote addition and subtraction modulo m. Let $u_0,\ldots,u_{m-1}$ be the primary vertices of H, and let H_i be the K_{m-1}-subexpansion of G obtained by deleting $u_{i\ominus 1}$ from H. By induction, there is a path p_i in H_i from u_i to itself such that, if E_i covers the trace of p_i, and E_i is normal w.r.t. p_i, then size $(E_i) \geq 2^{m-2}$. Let $s_{i,j}$ be a primitive path from u_i to u_j in H. For $i = 0,\ldots,m-1$, define

$$q_i = (p_i)^k s_{i,i\oplus 1}(p_{i\oplus 1})^k s_{i\oplus 1,i\oplus 2} \cdots (p_{i\oplus(m-1)})^k s_{i\oplus(m-1),i})$$

where $k = 2^m$, and p^k denotes k-fold iteration of loop path p. q_i is a path in H from u_i to itself. We must show that if E is normal w.r.t. G, and E covers the trace of q_i, then size (E) $\geq 2^{m-1}$. Let t_j be the trace of p_j.

Case 1. Suppose E is t_j-finite for some $j \in \{0,\ldots,m-1\}$. Let $t = t_j$. The definition of q_i implies that $I_t(E) \geq 2^m$. By lemma 3.2, size (E) $\geq 2^{m-1}$.

<u>Case 2</u>. Suppose E is t_j-finite for all $j \in \{0,\ldots,m-1\}$. The finitely many subexpressions of E can be divided into those which are t_j-finite and those which are not. For each j, let F_j be a minimal length subexpression of E which is t_j-infinite. It is clear that $F_j = A_j^*$ for some A_j. From the set $\{F_0,\ldots,F_{m-1}\}$, choose a minimal length member F_r.

F_r inherits E's normality, and F_r covers t_r. Thus, by induction, size $(F_r) \geq 2^{m-2}$.

<u>Claim</u>. There is a $k \in \{0,\ldots,m-1\}$ such that, if $x \in L(F_r)$, and p is a path in G with trace x, then p passes through u_k (a primary vertex of H).

Given the claim, we finish the proof of theorem 5.2 as follows. By property 2 of K_m-expansions, $t_{k \oplus 1}$ is the trace of no path except $p_{k \oplus 1}$. But $p_{k \oplus 1}$ is a path in $H_{k \oplus 1}$, and so does not pass through u_k. Since, by the claim, any member of F_r defines a path which does pass through u_k, E can cover $t_{k \oplus 1}^d$ for arbitrary d (as it must) only if E covers $t_{k \oplus 1}^d$ without entering F_r, or some subexpression of F_r is $t_{k \oplus 1}$-infinite. But the latter possibility violates the minimality of F_r.

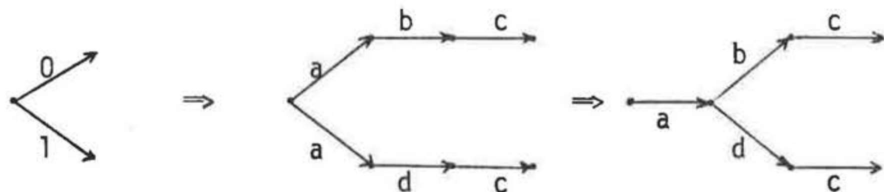
Let B be obtained from E by replacing F_r by λ. By the preceding argument, B is $t_{k \oplus 1}$-infinite; in particular, B covers $t_{k \oplus 1}$. By induction, size $(B) \geq 2^{m-2}$. But size (E) = size (B) + size $(F_r) \geq 2^{m-2} + 2^{m-2} \geq 2^{m-1}$.

<u>Proof of Claim</u>. $F_r = A_r^*$ has a base point b in G. By property 2 of K_n-expansions, every member of $L(F_r)$ traces exactly one path in G, namely one beginning and ending at b. If we can show that b is in H, then property 4 of K_m-expansions guarantees the existence of u_k.

F_r was chosen to be t_r-infinite, where t_r is the trace of p_r. If p_r

does not pass through b, then some proper subexpression of F_r must be t_r-infinite, violating the minimality of F_r. Thus p_r contains b, and, since p_r was defined as a path in H, b is a vertex in H. □

Before proceeding to Boolean variables, let us briefly examine the question of whether it is possible to recode any graph so as to make its traces economically representable by a regular expression. Call a homomorphism h simple, for lack of a better term, if, for $\sigma \neq \delta$, neither $h(\sigma)$ nor $h(\delta)$ is a prefix or a suffix of the other. Simple homomorphism are one-one, so that if $L(E) = h(L)$, every member of E corresponds to a unique member of L. We know by theorem 5.2 that L = traces $(C_n,0,0)$ cannot be represented by any expression of size less than 2^{n-1}, where C_n is the K_n-expansion of example 5.1. We show that h(L) is no easier, for any simple homomorphism h, by showing that $h(L) = \text{traces}(C_n',0,0)$ for some K_n-expansion C_n'. Given h, add non-primary vertices to C_n so that each arc labelled σ is replaced by a straight line path labelled $h(\sigma)$. In doing so, the only important property of C_n which might have changed is its forward and backward determinism. Restore determinism by merging vertices, as illustrated for h(0) = abc, h(1) = adc.



Backward determinism can be restored similarly. The important observation is that for simple homomorphisms, the vertex merging cannot propogate to the #-arcs in C_n, and so no other property of K_n-expansions is compromised.

Not all one-one homomorphisms are simple, and it remains open whether or not any one-one homomorphism exists to simplify any given language for regular expressions. However, given the weakness of regular expressions, it appears unlikely that such homomorphisms always exist.

6. ELIMINATING BOOLEAN VARIABLES

We begin with a simple minded upper bound on COST(n,0,m), the space required to eliminate m Boolean variables from a length n expression. Then we turn to lower bounds. We are unable to prove a completely satisfying lower bound on COST(n,0,m), and settle instead for a few special cases.

The variables can be eliminated from expression E in a straightforward manner: 1) apply lemma 3.1 to find a graph G for L(E), treating each variable operation as if it were an alphabetic symbol; 2) encode the variable values into the vertices of G by making 2^m copies of G; and 3) convert back to a regular expression. Analysis of that process gives

Theorem 6.1. $COST(n,0,m) \leq 6 \cdot 4^{p-2}$, where $p = 2n2^m$. □

There is no reason to believe that the above procedure is optimal. Indeed, when m = 0, the procedure introduces a potential exponential size increase when no work at all is required. But for the case of m close to n, we can prove a lower bound which coincides with the above upper bound to the point that both are double exponential.

The lower bound proofs depend on theorem 5.2, and are proved simply by exhibiting short expressions, with variables, for the traces in a certain K_s-expansion. We have already seen one K_s-expansion, namely C_s of example 5.1.

In order to write an expression E_s for traces $(C_s,0,0)$, we introduce some abbreviations. Let $s = 2^k$, and let $x_1,\ldots,x_k,y_1,\ldots,y_k$ be Boolean variables.

1. $\Pi e_i = e_1 \cdot e_2 \cdot \ldots \cdot e_k$;
2. $x \leftarrow 0 = \Pi x_i \downarrow$;
3. $x = 0? = \Pi \bar{x}_i ?$;
4. $x \leftarrow y = \Pi(\bar{y}_i ? \cdot x_i \downarrow \cup y_i ? \cdot x_i \uparrow)$;
5. $y \leftarrow \text{random} = \Pi(y_i \downarrow \cup y_i \uparrow)$;
6. $\text{gen } x = \Pi(\bar{x}_i ? \cdot 0 \cup x_i ? \cdot 1)$.

Then let

$$E_s = (x \leftarrow 0)((y \leftarrow \text{random})(\text{gen } u)\#(\text{gen } x)(x \leftarrow y))^*(x = 0?).$$

E_s represents traces $(C_s,0,0)$, but has size only $16k + 1$. Choosing $16K + 1 \leq n < 16(k+1) + 1$, we arrive at

<u>Theorem 6.2</u>. $\text{COST}(n,0,n/16) \geq 2^{P-1}$, where $p = 2^{n/16 - 1}$. □

Theorem 6.2 can be generalized by using fewer variables, at the cost of a longer expression. To that end, we define the $K_{r\cdot s}$-expansion C_s^r, which is constructed by completely connecting r copies of C_s. More precisely, make r copies of C_s. Draw an arc labelled a_{ij} from vertex u in copy i to vertex v in copy j if there is an arc labelled # from u to v in C_s. Finally, erase all of the #-arcs.

An expression for traces $(C_s^r,0,0)$ can be constructed as follows, for $s = 2^k$. Let E be a length at most 4^r expression for traces $(K_r,0,0)$. Replace each occurrence of a_{ij} in E by the expression

(*) $(y \leftarrow \text{random})(\text{gen } y)a_{ij}(\text{gen } x)(x \leftarrow y)$,

where x and y are k bit counters, as before. Calling the new expression E', let

$$E'' = (x \leftarrow 0)E'(x = 0?).$$

E" represents traces(C_s^r,0,0), and has size at most $4^r(14k + 1) + 2k$. Applying theorem 5.2 yields

Theorem 6.3. There is a constant $c > 0$ such that, for all n and all $k > 0$, $\text{COST}(n,0,2k) \geq (n/ck)^{s/2}$, where $s = 2^k$. □

It is worth noting that theorem 6.3 can be strengthened somewhat for small values of k. Counter y can be eliminated by using

$$\bigcup_{i=0}^{s-1} (\bar{k} \cdot a_{ij} \cdot (\text{gen } x)(x \leftarrow k))$$

in place of (*). Although the resulting expression is much larger than E", it has half as many variables. Analysis shows that the bound becomes $\text{COST}(n,0,k) \geq (n/cks)^{s/2}$, where $s = 2^k$, $c > 0$.

Our last lower bound is for a single variable.

Theorem 6.4. For every $\varepsilon > 0$, for infinitely many n, $\text{COST}(n,0,1) \geq n^{2-\varepsilon}$.

Proof. Otherwise theorem 6.2 could be violated by eliminating variables one at a time. □

Theorem 6.4 is not very satisfying, for while its bound is only quadratic, the best known upper bound is not even polynomial. The question of which bound needs to move is significant, for there are times when a few variables are a great convenience, but are only affordable if COST(n,0,1) is polynomial in n. On the other hand, finding COST(n,0,1) to be worse than polynomial is a very strong statement about the power of regular expressions.

7. CONCLUSION

It is disappointing to find the lower bounds so high. Even a quadratic increase must be considered a high price for eliminating just one variable, and the actual cost may be far greater. Since a few variables can be very helpful, for such uses as holding a character for later reference, it would be worthwhile to learn whether a fixed number of variables can be eliminated in polynomial space. Through considerable work, the author has managed to lose all intuition on the matter.

Of course, regular expressions are a convenient form of representation for some languages. Simple homomorphisms increase the power of regular expressions, in a quantitative sense. It is conceivable that arbitrary one-one homomorphisms increase their power still further, though very unlikely that they can help in every case.

References

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

2. Ehrenfeucht, A. and Zeiger, P., "Complexity Measures for Regular Expressions", JCSS 12, 134-146 (1976)).

3. Greibach, S.A., Theory of Program Structures: Schemes, Semantics, Verification, Lecture Notes in Computer Science 36, Springer-Verlag, Berlin, Heidelberg, New York.