CLOSURE ALGORITHMS

AND THE STAR-HEIGHT PROBLEM OF

REGULAR LANGUAGES

by

ROLAND CARL BACKHOUSE

A thesis submitted for the degree of Doctor of Philosophy
in the Faculty of Engineering in the University of London.

Department of Computing and Control
Imperial College
London SW7 1LU

September, 1975.

ABSTRACT

The main motivation for this work is the star-height problem of regular languages proposed by Eggan.

We begin by investigating in some detail analogies between regular algebra and linear algebra. It is shown how, using a few simple tautologies of regular algebra, one can derive product forms for the closure A* of a matrix A analogous to the Gauss and Jordan product forms for $(I-A)^{-1}$ in linear algebra. The product forms yield algorithms for calculating A* analogous to the Gauss and Jordan elimination methods of linear algebra. Methods for finding A* analogous to the iterative methods of linear algebra are also briefly discussed.

Following this we present various results in the theory of factors of regular languages. We show that there is a unique minimal starth root G_Q of the factor matrix $\lceil Q \rceil$ of a regular language Q, and give an algorithm to calculate G_Q. It is then shown that the factor matrix of a factor H of Q is a submatrix of the factor matrix $\lceil Q \rceil$. This leads to an algorithm to calculate the closure G_Q^* of G_Q. The algorithm yields a regular expression for Q which is proven to have star-height always less than or equal to (and in some cases strictly less than) the rank of G_Q. Moreover the algorithm does not have any analogous method in linear algebra.

The algorithm to find G_Q^* does not always give a minimal star-height expression for Q and the star-height problem is still open. But it is suggested that the algorithm may be more generally applicable (than just to factor graphs), thus possibly giving fresh insight into the star-height problem.

ACKNOWLEDGEMENTS

The work presented in this thesis is entirely the author's own work except for parts of Chapter II, or where otherwise indicated in the text.

The material in Chapter II represents work done in collaboration with Dr. B. A. Carré, and is essentially an adapted version of [2]. An attempt has been made to omit those parts of [2] which are definitely attributable to Dr. Carré, but it should also be mentioned that the derivation of the formulae in Chapter II §4.1 and §4.2 (but excluding §4.2.1), using the tautology $(AB)^* = E+A(BA)^*B$, is due to Dr. Carré. Aitken's method (§4.2.2) is also due to Dr. Carré and has been included here so that the author's comparison of Aitken's method and Jordan's method (§4.5) could be included. I would like to take the opportunity of thanking Bernard Carré for the very enjoyable and stimulating opportunity of working with him.

I would also like to thank my supervisor Jim Cunningham for much valuable criticism and encouragement while I was doing this work, and M. S. Paterson for his criticisms and suggestions regarding this work.

CONTENTS

INTRODUCTION

The objective of the work presented here is to study in depth certain aspects of the * operator in a regular algebra. Although not immediately apparent, such studies are relevant to many aspects of Computing Science, because the star operator is just one instance of a closure operator.

A closure operator is any operator J which is transitively closed, i.e. J.J = J or, in words, the effect of applying J twice (or any number of times) is the same as that of applying J just once. A trivial example is a while loop, since obviously while p do (while p do a) is equivalent to while p do a. Closure operations occur wherever an operation is iterated indefinitely. For instance a* can be interpreted as meaning "iterate a, unconditionally" whereas while p do a means "iterate a, conditional on p" or "iterate (test p then do a)".

Sometimes the iteration is concealed and it is not immediately obvious that a closure operation is involved. One instance is in the recursive definition of a function (e.g. Fx = if p then fx else Ffx). Recent work by Scott [40] shows clearly that what is involved implicitly in such a definition is indeed a closure operation - the operation of finding the least fixed point of a function g which in Scott's notation is

$$\bigsqcup_{n=0}^{\infty} g^n(\perp),$$

or in the notation of regular algebra $g^*(\perp)$. (g in the above example maps Fx onto fx or Ffx depending on p).

A particular example of this is in the study of context-free languages. We may regard the context-free grammar G = (V,N,T,P,S) having productions

p1: $S \to aSb$ p2: $S \to ab$

as defining two operators p1 and p2 mapping subsets of V^* into subsets of V^*. p1 is defined by

$$uaSbv \in p1(uSv) \quad \forall\ u,v \in V^*$$

and p2 by

$$uabv \in p2(uSv) \quad \forall\ u,v \in V^*.$$

From p1 and p2 a composite operator L = (p1+p2)* is formed, where p1+p2 applied to w is the union of p1 applied to w and p2 applied to w, and p* applied to w is the union of all sets found by applying p iteratively to w. The language generated by G is $T^* \cap L(S)$, i.e. the set of all terminal strings formed by applying the operator L to S. Hence of course the usual notation $S \to^* w$.

Closure operations also arise in the study of path-finding problems. A typical problem here is, given a network of points such that certain points are connected by arcs to which a cost is attached, what is the least cost path between any two points. It can be shown [2,6] that this problem can be formulated as finding the closure A* of a matrix A in some regular algebra. Other related problems, such as finding a route between two points having the least probability of blockage, can also be formulated in the same way. Rather

interestingly the algorithms we present in Chapter II can be used uniformly in the solution of such problems.

As a final example of where closure operators are important, let us observe that lattice theory is a study of relations which are anti-symmetric, and reflexive and transitive. Quite often in the practical application of lattice theory, one is not given such a relation but must construct it from other known relations on the elements being considered, and this invariably involves a closure operation. Non-trivial evidence of this can be seen in the construction of the lattice of "partitions with the substitution property" in the study of sequential machine decompositions [24].

The regular algebras are important as a step towards a more universal study of closure operators, since they would appear to offer the "simplest" examples of algebras having a non-trivial closure operator. The . and + operators, which are iterated to form the * operator of regular algebra, have quite simple properties - for instance distributivity

$a.(b + c) = a.b + a.c$ of . over +, and associativity of +

$a + (b + c) = (a + b) + c$. In comparison the while operator mentioned earlier is much more unmanageable - distributivity does not hold:-

a ; if p then b else c; ≠ if p then (a;b) else (a;c);

and associativity does not hold:-

if q then (if p then a else b) else c;

≠ if p then a else (if q then b else c);

(Note; this is clearer if we use infix notation:
Writing $a +_p b$ for *if* p *then* a *else* b; the above inequalities become

$$a ; (b +_p c) \neq (a;b) +_p (a;c)$$

and

$$(a +_p b) +_q c \neq a +_p (b +_q c) \quad).$$

Even though the star operator of regular algebra is thus a relatively simple closure operator, the study of its properties is not simple - in fact quite the opposite.

To avoid any confusion later, we should make explicit various terms we shall use. In Chapter I we give a purely formal list of axioms and a rule of inference involving the formal operator symbols +, . and *. By a regular algebra we understand any algebra having operators +,. and * which obey the rules given in Chapter I. Such an algebra is an interpretation of the system of axioms. A regular expression is simply a well-formed formula involving the symbols (,),*,+ and ., and symbols a,b,... from a finite vocabulary V. Regular expressions are used to denote elements of a regular algebra. In particular the familiar regular languages form just one instance of a regular algebra (a very important one, nevertheless) in which + is interpreted as set union and . as concatenation of words. In the algebra of regular languages a regular expression is a denotation for a regular language.

The problem we consider in Chapters III and IV is the star-height problem for regular languages. (We assume

of course that the reader has some familiarity with this problem. Those readers not already acquainted with the problem should consult the excellent paper by McNaughton [29], in which a highly significant amount of empirical information is presented.) The problem is concerned with finding the "simplest" possible regular expression denoting a particular regular language. In tackling this problem a possible application we envisage is as follows. Assume one is given a regular algebra R (not necessarily the regular languages) in which, for each element Q, there is some canonical denotation for Q. Let us call this denotation the "value" of Q, and suppose, given a regular expression f(a,b,...) denoting Q, it is required to "evaluate" Q. We presume that there is some procedure to do this. Usually one would expect that it is more difficult to evaluate starred terms A* in f. So, if before evaluating f we can preprocess f and find some regular expression g(a,b,...), such that in any interpretation f(a,b,...) = g(a,b,...) and the starred terms in g are simpler to evaluate, then evaluate g rather than f, we will clearly optimise on the computational effort. Since the regular languages form a free regular algebra, this is equivalent to finding the simplest possible denotation of a given regular language.

The approach we have adopted in tackling the star-height problem is to regard it as a problem of how best to calculate the closure A* of a matrix A whose entries are elements of a regular algebra. The approach is essentially similar to that of Eggan [18] in his pioneering work on the

star-height problem. Eggan considered the following problem. Given a (non-deterministic) finite-state recogniser of a regular language Q, what is the minimum star-height of all the regular expressions denoting Q which one can obtain by using an elimination method to solve the associated system of equations defining Q. Eggan succeeded in solving this problem, showing that a minimal star-height expression, obtained by applying an elimination method, equals the "rank" of the graph. Eggan showed how to calculate the rank of a graph, and how to order the nodes of the graph in the elimination process in order to obtain the best possible expression.

A rather simple converse to this result is that, given any regular expression g(a,b,...) denoting the language Q, there is naturally associated with g a recogniser of Q which has rank equal to the star-height of g. Thus one arrives at Eggan's theorem (see Chapter III): the star-height of a language Q equals the minimum rank of all transition graphs which recognise Q. Subsequent investigations of the star-height problem [9,10,11,12,28,29] begin with this theorem and aim to find a recogniser of Q having the least rank.

In contrast the main aim of this work is, given a recogniser of Q, to invent new methods of "solving" for Q which do better than a simple elimination method, i.e. to invent systematic methods of obtaining regular expressions for Q, from the given recogniser, which have star-height less than or equal to the rank of the recogniser - and in some cases strictly less than the rank of the recogniser.

This approach was suggested to the author by work done in collaboration with B.A. Carré on analogies between regular algebra and linear algebra [2]. The sections of this paper relevant to the present work are included in Chapter II. First we note in Chapter I that the problem of finding a regular expression denoting the language Q from a recogniser of Q is equivalent to finding certain entries in the closure A* of a matrix A. Now A* is analogous to $(I-A)^{-1}$ in linear algebra, since the former is the minimal solution of Y = AY+E, where E is the unit matrix in regular algebra, and the latter is the solution of Y = AY+I, where I is the unit matrix in linear algebra. Moreover the algebra $M_p(R)$, of all p×p matrices over the regular algebra R, is itself a regular algebra and thus any tautologies of regular algebra apply equally well to matrices. Beginning with these two observations, we note in Chapter II a few simple tautologies of regular algebra which have analogues in linear algebra and then show how these tautologies are sufficient to enable one to derive algorithms for calculating A*, which are analogous to the well-known elimination methods of Gauss and Jordan in linear algebra.

In addition, as an aside to the main theme of this thesis, we show how a number of other "path-finding algorithms", having analogues in the iterative techniques of linear algebra, can be succinctly described and compared using regular algebra.

Not all regular tautologies have analogues in linear algebra, and so, in restricting oneself to simple elimination methods for finding the closure of a matrix, one is under-

utilising the properties of regular events. An aim of the research for this thesis was therefore to seek methods of calculating A^*, for a given matrix A, which do not have analogues in linear algebra, and which improve on the elimination methods.

Unfortunately we have been unable to discover a generally applicable algorithm of this nature, but in Chapters III and IV we show that for a particular class of graph - the factor graph - such an algorithm does exist. Moreover the algorithm achieves one of our objectives, namely that the regular expressions produced always have star-height less than or equal to the star-height of those produced by any elimination method.

The algorithm, when applied to factor graphs, does not always yield minimal star-height expressions and the star-height problem remains unsolved. However, we suggest in the Conclusions how the algorithm might be extended to other important classes of graphs, and that a further attack along these lines might eventually lead to a solution of the problem.

I REGULAR ALGEBRAS

Regular languages, which we define in §2, have been studied by various authors [13,22,38,39] each of whom has given axiomatic formulations of their properties. In the next section we also give an axiomatic formulation of regular languages which is similar to that given by Salomaa [38] and then proceed to discuss some of the elementary consequences of the axioms.

1. THE SYSTEM OF AXIOMS F1

1.1 Axioms

The algebras we shall consider are of the form $R = (S,+,\cdot,*)$, where S is a set on which are defined two binary operators + and $\cdot$, and one unary operator *. The following are assumed as axiomatic.

A1	$(\alpha+\beta)+\gamma$	$=$	$\alpha+(\beta+\gamma)$	A4	$\alpha\cdot(\beta+\gamma)$	$=$	$(\alpha\cdot\beta)+(\alpha\cdot\gamma)$
A2	$\alpha\cdot(\beta\cdot\gamma)$	$=$	$(\alpha\cdot\beta)\cdot\gamma$	A5	$(\alpha+\beta)\cdot\gamma$	$=$	$(\alpha\cdot\gamma)+(\beta\cdot\gamma)$
A3	$\alpha+\beta$	$=$	$\beta+\alpha$	A6	$\alpha+\alpha$	$=$	α

where $\alpha, \beta, \gamma \in S$.

The set S contains a zero element ϕ such that

A7	$\alpha+\phi$	$=$	α	A8	$\phi\cdot\alpha$	$=$	ϕ	$=\ \alpha\cdot\phi$.

Finally the star (or closure) operator * obeys:

A9	$\phi^*\cdot\alpha$	$=$	α	$=$	$\alpha\cdot\phi^*$	}			
A10	α^*	$=$	ϕ^*	$+$	$\alpha\cdot\alpha^*$	}	for all	$\alpha \in S$.	
A11	α^*	$=$	$(\phi^* + \alpha)^*$			}			

We shall normally denote ϕ^* by e. The axioms A9-A11 then become:

A9' $\quad e \cdot \alpha = \alpha = \alpha \cdot e$

A10' $\quad \alpha^* = e + \alpha \cdot \alpha^*$

A11' $\quad \alpha^* = (e + \alpha)^*$.

1.2 Partial Ordering

In view of A1, A3 and the idempotency law A6 we can define a partial ordering $\subseteq$ on the set S by

$$\alpha \subseteq \beta \Leftrightarrow \alpha + \beta = \beta$$

and a strict ordering $\subset$ by

$$\alpha \subset \beta \Leftrightarrow \alpha \subseteq \beta \text{ and } \alpha \neq \beta .$$

It is easily verified that

$$\alpha \subseteq \beta \Rightarrow \alpha+\gamma \subseteq \beta+\gamma$$

$$\alpha \cdot \gamma \subseteq \beta \cdot \gamma$$

and $\quad \gamma \cdot \alpha \subseteq \gamma \cdot \beta$ for all $\alpha, \beta, \gamma \in S$.

Note that we do not assume the cancellative property

$$\alpha \cdot \gamma = \beta \cdot \gamma \Rightarrow \alpha = \beta ,$$

and in consequence we cannot in general infer that if $\alpha \subset \beta$ then $\alpha \cdot \gamma \subset \beta \cdot \gamma$.

1.3 Solution of Equations

We define an element α of S to be definite if and only if $t \subseteq \alpha \cdot t \Rightarrow t = \phi$. Then we assume the following rule of inference.

R1 $\quad \alpha = \beta \cdot \alpha + \gamma \Rightarrow \alpha \supseteq \beta^* \cdot \gamma$,

and furthermore, if β is definite then

$$\alpha = \beta \cdot \alpha + \gamma \Rightarrow \alpha = \beta^* \cdot \gamma .$$

It will be observed that for any given β and γ the equation $\alpha=\beta\cdot\alpha+\gamma$ always has a solution $\alpha=\beta^*\cdot\gamma$. The first part of our rule R1 postulates that $\alpha=\beta^*\cdot\gamma$ is the minimal solution, and the second part gives a condition under which this solution is unique.

Henceforth we shall denote the set of axioms A1-A11 and the rule of inference R1 by F1, and we shall call any algebra $R = (S,+,\cdot,^*)$ such that the set F1 is valid in R a regular algebra.

2. INTERPRETATIONS

It is important to realise that the above system of axioms is a purely formal system in which no meaning has been attached to the symbols of S or to the operator symbols +, $\cdot$ and *. We now describe a number of interpretations of $S,+,\cdot$ and * which give rise to a regular algebra and which are particularly important. Note that the list is by no means exhaustive.

2.1 Regular Languages

Consider any finite non-empty set $V = \{v_1,v_2,\ldots,v_m\}$ which we call an alphabet or vocabulary, and whose elements we call letters. A word over V is a finite string of zero or more letters of V; the string consisting of zero letters is called the empty word.

The set of all words over V is denoted by V*. A language over V is any subset of V*. The symbol ϕ denotes the empty set, and $\phi^* = e$ denotes the set consisting of the empty word.

The sum $\alpha+\beta$ of two languages α and β is their set union, and the product or concatenation $\alpha\cdot\beta$ is the set of all words formed by concatenating a word in α with a word in β. The powers of a language α are defined by

$$\alpha^0 = e, \quad \alpha^k = \alpha\cdot\alpha^{k-1} \qquad (k = 1,2,\ldots)$$

and the closure α^* of α is defined to be

$$\alpha^* = \sum_{k=0}^{\infty} \alpha^k \quad .$$

In the ensuing paragraphs we shall use three terms - regular expression, regular event and regular language - with quite different meanings. A regular expression over a vocabulary V is defined to be a well-formed formula constructed from the set $V\cup\{\phi\}$, the operators + · and *, and the parentheses (and) . Thus $(v_1 + ((v_2+v_1)\cdot v_3))^*$ is a regular expression. Also allowed are formulae in which the dot is omitted, being denoted by juxtaposition, and parentheses are omitted. In this latter case operator precedence is in the order *, ·, + .

We define a regular event to be any element of a regular algebra which can be denoted by a regular expression over some finite vocabulary V.

We also define a regular language to be any set of words over some finite vocabulary V which can be denoted by a regular expression.

Finally, following Conway [13], we shall use the term regular tautology to mean any equation f(a,b,...) = g(a,b,...) between two regular expressions which is universally true in any regular algebra.

Regular languages are particularly important in the study of regular algebras, since the axiom system F1 is consistent with the regular languages and is a complete system for proving valid equations between regular languages. Essentially this was proved by Salomaa [38], although Salomaa's axiom system differs from ours, particularly in the rule R1 and in the condition for uniqueness of solution of equations. For regular languages it is however trivial to establish that the two conditions (for uniqueness of solutions) are identical, and all other differences between our and Salomaa's axioms are inessential. (For further discussion see §2.2.3, below, where we consider the uniqueness of solution of matrix equations over the regular languages.)

The regular languages over V thus form a free regular algebra, signifying that the algebra is free of any equalities holding between its elements which cannot be deduced using the system F1. Hence we call the algebra of the regular languages over vocabulary V the free regular algebra generated by V, and denote it by $R_F(V)$.

2.2 Matrix Algebras

2.2.1 The Algebra $M_p(R)$

If one examines the system F1 one may observe that a number of operators satisfy the properties of + and • . For example the operators min and max defined on the real numbers obey the properties of +, and real addition, min and max obey

the properties of $\cdot$. These interpretations do not have any practical significance when S is the set of all real numbers, but they are very significant when we consider matrices over the real numbers. First, however, a number of preliminaries are necessary.

Given a regular algebra R we may form an algebra $M_p(R)$ consisting of all p×p matrices whose elements belong to R. In the algebra $M_p(R)$ the operators + and $\cdot$ and the order relation $\subseteq$ are defined as follows:

Let

$$A = [a_{ij}] \quad \text{and} \quad B = [b_{ij}]$$

be any p×p matrices with elements in R; then

$$A+B = \left[a_{ij} + b_{ij} \right], \qquad A \cdot B = \left[\sum_{k=1}^{p} a_{ik} \cdot b_{kj} \right]$$

and

$$A \subseteq B \text{ if and only if } a_{ij} \subseteq b_{ij} \text{ for all } i,j.$$

The unit matrix $E = [e_{ij}]$ is defined as that p×p matrix with $e_{ij} = e$ if $i=j$ and $e_{ij} = \phi$ if $i \neq j$. The rows and columns of this matrix are described as <u>unit</u> vectors. The <u>zero</u> or <u>null</u> matrix N is that matrix all of whose entries are ϕ.

Powers of A are defined by

$$A^0 = E, \quad A^k = A \cdot A^{k-1}, \quad k = 1,2, \ldots$$

and finally A* is defined informally as

$$\sum_{k=0}^{\infty} A^k \quad .$$

(Note, a formal definition of A* is given in Appendix A. However the above definition of A* is much more useful intuitively. We do not assume this form of the definition in any proofs.)

If R is a regular algebra it is a logical problem to prove that $M_p(R)$ is a regular algebra, that is that all the axioms A1-A11 and the rule of inference R1 remain valid in $M_p(R)$. For our axiomatisation this is a fairly simple problem; the main parts of the proof are given in Appendix A. Moreover for most practical applications one can also infer the validity of F1 from the previous literature.

2.2.2 Graphs

It is well-known that a p-node graph G can be described by a p×p matrix A, and, conversely, a p×p matrix can always be visualised as a p-node graph. We shall now take the opportunity to make this precise and to introduce some terminology which will be of use later.

A labelled graph G = (X,A) consists of a set X of p elements $x_1, x_2, \ldots, x_p$ together with a p×p matrix $A = [a_{ij}]$ with elements in some regular algebra R. An arc is a pair (x_i, x_j) such that the arc label a_{ij} is non-null, and is said to be directed from x_i to x_j. A sequence of t arcs

$$\mu = (x_i, x_{k_1}), (x_{k_1}, x_{k_2}), \ldots, (x_{k_{t-1}}, x_j)$$

such that the terminal node of each arc coincides with the initial node of the following arc is called a path from x_i to x_j, of path length t. If $i \neq j$ the path is said to be open; whereas if $i=j$, μ is called a closed path or cycle. A path (open or closed) is elementary if it does not traverse any of its nodes more than once. The path product $w(\mu)$ of a path μ is defined as the product of its arc labels:

$$w(\mu) = a_{ik_1} a_{k_1k_2} \cdots a_{k_{t-1}j} .$$

A graph without any cycles is called acyclic.

2.2.3 Regular Languages

If the arc labels are regular languages and, in particular, if they are subsets of $V \cup \{e\}$ we call G a (nondeterministic) transition graph. In general for any path μ from x_i to x_j the path product $w(\mu)$ will be a set of words. A word v is said to be a word taking node x_i to node x_j if either $v \in w(\mu)$ for some path μ from x_i to x_j, or $v=e$ and $i=j$. The closure $A^*=[a^*_{ij}]$ is such that a^*_{ij} is the set of all words which take node i to node j.

The consistency of $M_p(R_F(V))$ (the algebra of $p \times p$ matrices whose elements are regular languages over the vocabulary V) with F1 has been studied by Salomaa [39]. Salomaa's axiom system differs from ours in a number of respects, but mainly in rule R1. Firstly, our rule R1 is the reverse of Salomaa's rule R1 (i.e. Salomaa gives the solution of equations of the form $\alpha = \alpha \cdot \beta + \gamma$, not $\alpha = \beta \cdot \alpha + \gamma$) and, correspondingly, our axiom A10 ($\alpha^* = \phi^* + \alpha \cdot \alpha^*$) differs from Salomaa's ($\alpha^* = \phi^* + \alpha^* \cdot \alpha$). This change is not essential in that Salomaa's proofs of the consistency and completeness of his axiom system can easily be modified to apply to our own axiom system. Thus, henceforth, we shall assume any theorems proved by Salomaa to be true and leave the reader to convert them to proofs in our own axiom system.

Secondly, Salomaa's condition for uniqueness of solution of equations also differs (outwardly) from ours. The relevant definition and theorem are given below.

Definition 2.1 (Salomaa): A $p\times p$ matrix $M=[m_{ij}]$ possesses the empty word property (ewp) iff there is a sequence of numbers $i_1,i_2,\ldots,i_k$ $(k\geq 1)$ such that $e\varepsilon m_{i_v,i_{v+1}}$ for all v, $1\leq v\leq k-1$, and $e\varepsilon m_{i_k i_1}$.

Theorem 2.2 (Salomaa): If the matrix M does not possess ewp then the equation $Y=MY+R$ has a unique solution, namely $Y=M^*R$.

For 1×1 matrices, i.e. regular languages, it is obvious that we have the equivalence: M does not possess ewp $\Leftrightarrow$ M is definite. The equivalence is not so obvious for larger matrices, but is nevertheless easily proved.

Theorem 2.3 Let $A\varepsilon M_p(R_F(V))$. Then A does not possess ewp $\Leftrightarrow$ A is definite.

Proof (i) $\Leftarrow$. Suppose $\exists$ $T\neq N$ such $T=AT$. Then the equation $T = A\cdot T+N$ has more than one solution, namely T and N. Therefore A possesses ewp.

(ii) $\Rightarrow$. Suppose A possesses ewp. Then the graph of A contains a cycle.

$$\gamma = (x_{i_1},x_{i_2})\ (x_{i_2},x_{i_3}),\ldots,(x_{i_k},x_{i_1})$$

such that

$$e \varepsilon a_{i_1 i_2} a_{i_2 i_3} \cdots a_{i_k i_1} \qquad (1)$$

Let B be the submatrix of A consisting solely of the arc labels $a_{i_v i_{v+1}}$ $(v = 1,\ldots,k-1)$ and $a_{i_k i_1}$. $B\subseteq A$, so we may choose a matrix C such that $B+C=A$.

Now let $T=B^+$. By (1)

$$(b_{i_1 i_2} b_{i_2 i_3} \ldots b_{i_k i_1})(b_{i_1 i_2} \ldots b_{i_k i_1})^* = (b_{i_1 i_2} \ldots b_{i_k i_1})^*$$

and it follows that $B \cdot B^+ \supseteq B$ and hence $T = BT$. But then

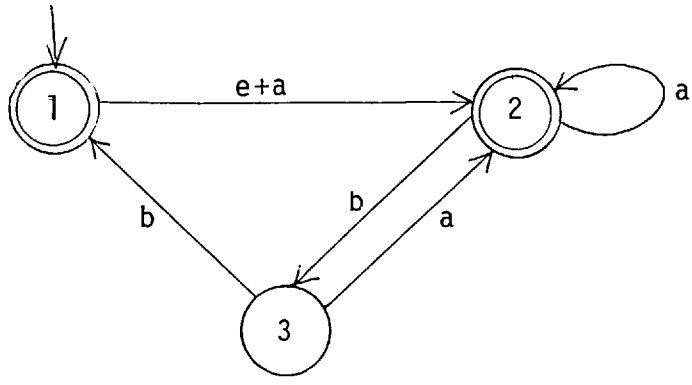
$$T \subseteq BT + CT = AT$$

and therefore A is not definite.

Following Conway [13] we call a matrix A, all of whose non-null entries are e, a <u>constant matrix</u> and a matrix all of whose non-null entries are subsets $a_{i_1} + a_{i_2} + \ldots a_{i_k}$ of the letters in V, a <u>linear matrix</u>. A <u>constant + linear</u> matrix is, as the terminology suggests, one which is the sum of a constant matrix and a linear matrix. Thus a transition graph always has a constant + linear matrix. A <u>recogniser</u> (G,S,T) is a transition graph G = (X,A) for which two subsets S,T of the set of nodes X are designated as <u>start</u> and <u>terminal</u> nodes, respectively; the language recognised is the set of all words which take some start node $s \in S$ to some terminal node $t \in T$. A recogniser is <u>all-admissible</u> if for any node $x \in X$ there is a path from a start node $s \in S$ to x, and a path from x to a terminal node $t \in T$.

Finally we define a graph to be <u>deterministic</u> if for all words w and all nodes x_i, there are no two distinct nodes x_j and $x_{j'}$ such that w takes x_i to x_j and x_i to $x_{j'}$. A recogniser is deterministic if its associated graph is deterministic and S has cardinal 1.

As usual, we shall specify a recogniser diagrammatically, as illustrated below.



A Recogniser

The graph of this recogniser has nodes X = {1,2,3} and matrix

$$A = \begin{bmatrix} \phi & e+a & \phi \\ \phi & a & b \\ b & a & \phi \end{bmatrix}$$

There is only one start node (node 1) indicated by an unlabelled arrow pointing to the node. Terminal nodes (nodes 1 and 2) are indicated by double circles.

Because of the very natural correspondence between graphs and matrices we shall in future use the two words synonymously. Thus we shall refer to the closure G* of a graph G, where, more exactly, we mean the graph of the closure A* of the matrix A of G.

2.2.4 Boolean Matrices

In table 1, in which R denotes the real numbers, we have shown a number of different interpretations of S,+,· and * which have practical value. The table is adapted from one given by Carré [6] by adding a column giving an interpretation of the * operator. In each case the star operator has no real significance; it becomes important only when we begin to consider matrices.

The algebra R_B is the familiar Boolean algebra in which 0 and 1 represent <u>false</u> and <u>true</u>. We may regard a p×p matrix $A = [a_{ij}]$ with elements in this algebra as specifying a relation on the integers 1,2,...,p. Thus

$$iAj \qquad \text{(read } i \text{ is related by A to } j\text{)}$$

$$\iff \quad a_{ij} = 1 \qquad 1 \le i,j \le p.$$

In $M_p(R_B)$ the operator * is highly significant since A* is the <u>reflexive and transitive closure</u> of A. That is A* is the least relation containing A and having the properties

$$iA^*i \qquad \text{(reflexivity)}$$

and $\quad (iA^*j \text{ and } jA^*k) \Rightarrow iA^*k \qquad \text{(transitivity)}.$

It is this particular application of regular algebras from which one gets the terminology "closure" operation for the * operation.

The applications of this algebra are numerous and are so familiar that they need hardly be mentioned. Suffice it to say that A* represents the connectivity of a directed graph and wherever graph theory finds application, then so does $M_p(R_B)$.

TABLE 1

	S	+	·	e	ϕ	⊃	*
R_{SP}		min			∞	<	$a^* = \begin{cases} 0 & \text{if } a \geq 0 \\ -\infty & \text{if } a < 0 \end{cases}$
	$R \cup \{-\infty, \infty\}$		+	0			
R_{SC}					$-\infty$		$a^* = \begin{cases} 0 & \text{if } a \leq 0 \\ \infty & \text{if } a > 0 \end{cases}$
R_B	{0,1}	max					1
			×	1	0	>	
R_C							∞
	$\{a \in R \mid a \geq 0\} \cup \{\infty\}$						
R_{CP}			min	∞	0		∞

2.2.5 Finding Shortest Path

If the arc labels of a graph A are real numbers representing costs or distances the algebra $M_p(R_{Sp})$ can be used to find the least cost or the shortest distance from one node to another. Indeed it was only through reading the paper by Carré [6] and by observing the tremendous similarity between Carré's algebra and regular algebra that the author first realised the usefulness of regular algebra in the context of path-finding problems. However, before we can make a precise comparison between regular algebra and Carré's algebra we must remove the obstacles caused by differences between the definitions used here and in [6].

Carré's paper was concerned exclusively with the solution of extremal path problems, and its definition of A* was tailored to this purpose. Let us here define $\tilde{A}$ to be the matrix whose (i,j)th element is the sum of the path products of all elementary paths from x_i to x_j on the graph of A; the matrix $\tilde{A}$ as defined here corresponds to A* in [6]. Now to relate $\tilde{A}$ to the closure A* of A we recall Carré's definitions of definiteness and semi-definiteness, which can be paraphrased as follows: Let A be a p×p matrix; then A is semi-definite iff there is no closed path γ in A with path-product $w(\gamma) \supset e$; A is definite iff there is no closed path γ in A with path-product $w(\gamma) \supseteq e$. (Note that Carré's definition of definiteness is identical to Salomaa's definition of ewp and hence is equivalent to our own.)

In addition to A1-A9, Carré assumed the following:

(a) commutativity of multiplication, $\alpha\cdot\beta = \beta\cdot\alpha \quad \forall\alpha,\beta$.

(b) the order relation $\leq$ is total, i.e. for any α,β either $\alpha\leq\beta$ or $\beta\leq\alpha$,

(c) the cancellation property:

$\forall\alpha,\beta,\gamma\neq-\infty,\alpha\neq\infty; \quad \alpha\cdot\beta = \alpha\cdot\gamma \Rightarrow \beta=\gamma$.

With the above properties holding one can prove the following well-known theorem [Theorem 4.1 of 6].

Theorem 2.4 Let A be a p×p semi-definite matrix. Then the series $E+A+A^2+\ldots$ is finitely convergent, with $E+A+A^2+\ldots+A^r = \tilde{A}$ for all $r\geq p-1$.

Corollary $\tilde{A} = A^*$.

Proof $\tilde{A} \leq A^*$ by p applications of A10. But by Theorem 2.4, $\tilde{A} = E+A\cdot\tilde{A}$ and hence by R1, $\tilde{A} \geq A^*$. Thus they are equal.

Thus when A is semi-definite, shortest distances in A are given by A*. When A is not semi-definite A must contain at least one cycle of negative length and the concept of distance becomes meaningless. Thus in this case A* has no real significance.

Regular algebra may also be used to find shortest routes between any two points in a graph as follows. Consider a vocabulary $V = \{v_1, v_2, \ldots, v_p\}$ and let R_{SR} be the free regular algebra generated by V on which the following generating relations are imposed:

$$v_i\cdot\alpha\cdot v_i = v_i\cdot v_i = \phi \ , \quad (i = 1,2,\ldots p) \text{ for all } \alpha.$$

In this algebra one can prove that $\alpha^* = e+\alpha$, and so the star operator may be discarded.

To enumerate the elementary paths on an unlabelled p-node graph $G=(X,\Gamma)$, we give a name v_i to each node $x_i \in X$, and we label each arc of G with the name of its terminal node, i.e. we set $a_{ij} = v_j$ for all $(x_i,x_j) \in \Gamma$. Then within the algebra $M_p(R_{SR})$ the closure A* of the matrix A gives all elementary paths on G: specifically, each product $\{v_i\} \cdot a^*_{ij}$ is a set of sequences of node names, each of these sequences defining an elementary path from x_i to x_j.

If the graph G is labelled with costs or distances, the two algebras $M_p(R_{SP})$ and $M_p(R_{SR})$ may be combined to give least cost routes (and their cost) through G. Specifically each arc of G is labelled with a pair (c,r) where c is the cost of traversing the arc and r is the name of the node on which the arc terminates. The product and sum operations which are appropriate to finding shortest routes are:

$(c,r) \cdot (c',r') = (c \cdot c',\ r \cdot r')$

$(c,r) + (c',r') = (c+c',$ if $c+c' = c$ then r else $r')$.

In practice it is of course inadvisable to store the entire shortest route from node i to node j in the (i,j)th element of A*. Instead it suffices to store the number of the node immediately following node i on the shortest route from node i to node j [7]. To do this it is simply necessary to redefine the product operation to:

$$(c,r) \cdot (c',r') = (c \cdot c',r) .$$

2.2.6 Other Applications

Finally we make brief mention of three other applications of regular algebra of practical importance.

If arc labels a_{ij} represent the probability of going directly from node x_i to node x_j then in $M_p(R_C)$ (Table 1), a^*_{ij} represents the maximum probability of going from node x_i to x_j.

If nodes represent tasks, and there is an arc from node x_i to node x_j labelled with the duration of the task x_i, if x_i must be completed before x_j may begin, then $M_p(R_{SC})$ (Table 1) may be used in scheduling the various tasks.

Finally $M_p(R_{CP})$ (Table 1) may be used to find critical paths through routes with some likelihood of blockage. For example if arc labels represent the width of a bridge on a path from node x_i to x_j, this algebra may be used to find the minimum width bridge on a route between two given points x_i and x_j which minimises the likelihood of a blockage.

We refer the reader to [6] for detailed references on these topics.

2.3 Semigroups

As a final example of a regular algebra which we shall exploit in Chapters III and IV we now mention the regular algebra defined by a finite semigroup.

Let $\underline{S} = (S,\cdot)$ be any finite semigroup having a unit element e. Consider all subsets 2^S of S and let + be the operation of set union defined on 2^S. Extend the operation

• in the usual way to apply to elements of 2^S by

$$\left(\sum_{i=1}^{m} a_i\right)\cdot\left(\sum_{j=1}^{n} b_j\right) = \sum_{i=1}^{m}\sum_{j=1}^{n} a_i\cdot b_j \quad .$$

For any $\alpha\varepsilon 2^S$, define $\alpha^0=e$ (the unit of S) and

$$\alpha^n = \alpha\cdot\alpha^{n-1} \quad , \quad n = 1,2,\ldots$$

Since S is finite it is clear that for sufficiently large N

$$e+\alpha+\alpha^2+\ldots+\alpha^N = e+\alpha+\ldots+\alpha^N+\ldots+\alpha^{N+n}$$

for any $n\geq 0$, and all $\alpha\varepsilon 2^S$.

We define α^* to be this sum. In this way the semigroup $\underline{S}$ is naturally extended to become a regular algebra. We will normally denote this algebra by $R(\underline{S})$.

II ELIMINATION METHODS FOR FINDING CLOSURE MATRICES

Everyone knows how to solve simultaneous equations in linear algebra! The equations

$$x = ax + by + e \tag{1}$$

$$y = cx + dy \tag{2}$$

are solved by eliminating x from the right hand side of (1), forward substituting this value in (2) thus finding y, and finally back substituting in (1) to get x. By this process one would get

$$x = (1-a)^{-1}[b(1-(c(1-a)^{-1}b+d))^{-1}c(1-a)^{-1}e+e] \tag{3}$$

$$y = (1-(c(1-a)^{-1}b+d))^{-1}c(1-a)^{-1}e \tag{4}$$

In regular algebra, one uses the rule R1 to eliminate variables in exactly the same way. Thus if a,b,c are letters and e is the empty word, one obtains as minimal solutions of the above equations

$$x = a^*[b(ca^*b+d)^*ca^*+e] \tag{3}'$$

$$y = (ca^*b+d)^*ca^* \tag{4}'$$

(3)' and (4)' are identical to (3) and (4) if one replaces the symbolism m^* by $(1-m)^{-1}$, and assumes e=1.

In both linear algebra and regular algebra we may express the equations (1) and (2) in matrix form, as

$$Y = AY + \underline{e}_{01}$$

where

$$Y = \begin{bmatrix} x \\ y \end{bmatrix}, \quad A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

and $\underline{e}_{01} = \begin{bmatrix} e \\ 0 \end{bmatrix}$ is the first column of the unit

matrix

$$\begin{bmatrix} e & 0 \\ 0 & e \end{bmatrix} \quad \text{(assume } e=1 \text{ in linear algebra).}$$

The solutions are

$$Y = (I-A)^{-1}\underline{e}_{01} \text{ in linear algebra}$$

$$\text{and } Y = A^*\underline{e}_{01} \text{ in regular algebra}$$

(more precisely this is the minimal solution).

This analogy between linear algebra and regular algebra has surely been noticed by many others interested in regular languages (for example, Aho and Ullman [1]), but apart from a casual reference none has bothered to investigate the analogy further. In linear algebra quite significant economies can be made in simplifying proofs and avoiding long, involved formulae if one uses the so-called matrix methods [21]. These methods are based on the simple concept that matrices have inverses which satisfy properties identical to the inverses of real numbers. But in regular algebra one can also talk meaningfully about the star A* of a matrix A, and, moreover, it has the same properties as α^* for a language α (as well as others). In this chapter, our objective is to investigate fully the analogy between linear algebra and regular algebra, and, in particular, to show how matrix methods can equally be employed in regular algebra as well as in linear algebra.

The impetus for this investigation was provided by Carré [6], who, in studying some path-finding problems,

obtained product forms for A* which are entirely analogous to the Gauss and Jordan product forms for $(I-A)^{-1}$ in linear algebra. Carré's paper, however, falls short of our objective, as he also used the usual "school" methods for deriving the product forms, and, at the time, did not realise the possibility of using matrix methods directly.

1. Comparison Between Uniqueness of Solutions

Before proceeding to discuss algorithms for finding A*, for a matrix A, it is worth observing the analogy between the conditions for uniqueness of solution of equations in linear algebra and regular algebra.

By R1, A* is a solution of the equation

$$Y = AY + E \tag{1.1}$$

and, moreover, it is the unique solution if A is definite. Now, in linear algebra, the equation

$$Y = AY + I \tag{1.2}$$

has the unique solution $Y=(I-A)^{-1}$ if and only if I-A is non-singular, or, if and only if $T=AT \Rightarrow T=0$.

Let us compare this with our definition of definiteness: $A \in \mathcal{M}_p(R)$ is definite iff $T \subseteq AT \Rightarrow T=N$. The analogy is not quite complete, but this is soon rectified by Lemma 1.1 below. Note that in the proof we anticipate the proof of the identity (2.1): $AA^* = A^*A$, but we assure the reader that we are not arguing tautologously.

Lemma 1.1 ($\exists$ $T \neq N$ such that $T \subseteq AT$) $\Leftrightarrow$ ($\exists$ $T \neq N$ such that $T = AT$).

Proof $\Leftarrow$ is trivial. So let us assume $\exists$ $T \neq N$ such that $T \subseteq AT$.

Then $T \subseteq AT \Rightarrow A^*T \subseteq A^*AT = AA^*T$ (by 2.1).

But by A10, $A^*T \supseteq AA^*T$.

Hence $A^*T = A(A^*T)$,

i.e. $T' = A^*T = AT'$, and the lemma is proved.

It may also be possible to establish an analogue to the determinant, det(B), of a real matrix B. Consider for simplicity a 2×2 matrix

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} .$$

In regular algebra, the condition of non-definiteness of A reduces to two conditions (cf. Appendix A).

Either (a) $a_{11} \supseteq e$ or (b) $a_{22} + a_{21} a_{11}^* a_{12} \supseteq e$.

Now suppose A is a real matrix, and consider B=I-A.

$$\det(B) = (1-a_{11})(1-a_{22}) - a_{21}a_{12}$$
$$= (1-a_{11}) \left[1 - a_{22} - a_{21} \cdot \frac{1}{1-a_{11}} \cdot a_{12}\right] .$$

Replacing $(1-m)^{-1}$ by m^*, we get:

$$a_{11}^* \; (a_{22} + a_{21} a_{11}^* a_{12})^* \det(B) = 1 .$$

Superficially, $\det(B) = 0$ if and only if either

(a)' $a_{11} = 1$

or (b)' $a_{22} + a_{21} a_{11}^* a_{12} = 1$.

Of course, as anyone can see, this is not true and the analogy between (a), (b) and (a)', (b)' has been fiddled. Nevertheless, one cannot dispute that there is a similarity and one which is quite interesting to observe.

The condition for definiteness takes various concrete forms under different interpretations of a regular algebra. For instance, as we saw in Chapter I §2.2.3, for matrices whose entries are regular languages, non-definiteness corresponds to the empty word property defined by Salomaa. For further discussion of this see section 5 of [2].

2. Some Regular Tautologies and their Analogues

Rather surprisingly, with A^* replacing $(I-A)^{-1}$, much of the theory of real matrices also holds in regular algebra. In this section we shall establish a few simple regular tautologies, all of which have analogues in linear algebra, and which are sufficient to enable us to obtain analogues in regular algebra of all the direct methods of solving linear equations and inverting matrices. The tautologies are listed below, together with their proofs, following which we give their analogues in linear algebra.

Tautology (2.1): $A^*A = AA^*$

Proof
$$\begin{aligned} A(A^*A + E) + E &= AA^*A + A + E \\ &= (AA^* + E)A + E \\ &= A^*A + E, \text{ by A1O'.} \end{aligned}$$

Thus by R1, $A^*A + E \geq A^*$

hence, $A(A^*A + E) \geq AA^*$

therefore, $(AA^* + E)A \geq AA^*$

so, by A10', $A^*A \geq AA^*$. (2.1.1)

But, by A10', $AA^* = A + A(AA^*)$

hence, by R1, $AA^* \geq A^*A$. (2.1.2)

From (2.1.1) and (2.1.2), $A^*A = AA^*$.

Tautology (2.2): $A(BA)^* = (AB)^*A$

Proof Let $X = A(BA)^*$, $Y = (AB)^*A$, and $P = (BA)^*$. Then

by A10' $P = E + BAP$

hence $X = AP = A + ABAP$

$= A + ABX$.

Therefore, by R1, $X \geq (AB)^*A = Y$. (2.2.1)

Also, since $Y = (AB)^*A$,

$Y = (E + AB(AB)^*)A$

$= A + AB(AB)^*A = A + ABY$.

Hence, $BY = BA + BA(BY)$.

Therefore, by R1, $BY \geq (BA)^*BA$

$= BA(BA)^*$ by 2.1.

So $Y = A + ABY$ gives $Y \geq A + ABA(BA^*)$

$= A(E + BA(BA)^*)$

$= A(BA)^* = X$. (2.2.2)

From (2.2.1) and (2.2.2), $X = Y$.

Tautology (2.3): $(A + B)^* = A^*(BA^*)^*$

Proof Let $X = (A + B)^*$ and $Y = A^*(BA^*)^*$.

Now $E + (A + B)Y = E + AA^*(BA^*)^* + BA^*(BA^*)^*$

$= (BA^*)^* + AA^*(BA^*)^*$

$= A^*(BA^*)^* = Y$.

Hence, by R1, $Y \geq (A + B)^* = X.$ (2.3.1)

Also, by A1O', $X = E + (A + B)X,$

so $X = AX + (E + BX),$

hence, by R1, $X \geq A^*(E + BX)$

$= A^*BX + A^* .$

Now $X \geq A^*BX + A^* \Rightarrow X \geq (A^*B)^*A^* .$ (2.3.2)

From (2.3.1) and (2.3.2), $X = Y.$

Tautology (2.4): $(A + B)^* = (A^*B)^*A^*$

Proof This is immediate from (2.2) and (2.3).

Tautology (2.5): $(AB)^* = E + A(BA)^*B$

Proof This is immediate from A1O' and (2.2).

The identities (2.1) - (2.5) all have analogues in linear algebra, which are the following:

(2.1a) : $(I-A)^{-1}A = A(I-A)^{-1}$

(2.2a) : $A(I-BA)^{-1} = (I-AB)^{-1}A$

(2.3a) : $(I-(A+B))^{-1} = (I-A)^{-1}[I-B(I-A)^{-1}]^{-1}$

(2.4a) : $(I-(A+B))^{-1} = [I-(I-A)^{-1}B]^{-1}(I-A)^{-1}$

(2.5a) : $(I-AB)^{-1} = I + A(I-BA)^{-1}B .$

The identities (2.3) and (2.4) will find great prominence in the following sections, because they will enable us to express any closure matrix as a product of elementary transformation matrices. In regular algebra these identities are well-known, indeed they are often listed as axioms (e.g. Conway [13]).

3. Product Forms for Closure Matrices

In this section we shall use the tautologies 2.3 - 2.5 to derive product forms for the closure A* of a p×p matrix A, analogous to the Jordan product forms and triangular factor representations of inverse matrices in linear algebra. These product forms yield algorithms analogous to the direct methods of linear algebra, for calculating the minimal solution Y = A*B of a set of equations of the form Y = AY + B.

In this Chapter, the typical elements of a matrix M and its closure M* will be denoted by m_{ij} and m^*_{ij} respectively, and the closure of an element m_{ij} of M will be denoted by $(m_{ij})^*$. The i^{th} row and the j^{th} column will be denoted by $\underline{m}_{io}$ and $\underline{m}_{oj}$ respectively.

3.1 Row and Column Matrices

Our techniques for deriving product forms are based on the following simple idea: Given a matrix $A = A^{(0)}$, we can split $A^{(0)}$ into two matrices $C^{(1)}$ and $S^{(1)}$, and using (2.4), write $A^* = A^{(0)*} = (C^{(1)} + S^{(1)})^* = (C^{(1)*}S^{(1)})^*C^{(1)*} = A^{(1)*}C^{(1)*}$, say. In doing so, the problem of determining the closure of $A^{(0)}$ is turned into the problem of finding the closure of two matrices $A^{(1)}$ and $C^{(1)}$. Our strategy is to choose $C^{(1)}$ and $S^{(1)}$ such that

(a) the closure of $C^{(1)}$ can be immediately calculated and

(b) $A^{(1)}$ has a "simpler" form than $A^{(0)}$.

We can repeat the process for $A^{(1)}$, reducing the problem of finding $A^{(1)*}$ to that of finding $C^{(2)*}$ and $A^{(2)*}$ and so on. The process is terminated when, after p steps, $A^{(p)}$ is of such a form that its closure $A^{(p)*}$ can also be immediately calculated. Using alternative methods of splitting, combined with different identities of regular algebra, we can derive different product forms for A*.

The requirement (a) above is achieved if we choose the matrices $C^{(k)}$ always to be column matrices, that is matrices of the form

$$C^{(k)} = \begin{bmatrix} \phi \ldots \phi & c_{1k} & \phi \ldots \phi \\ \phi \ldots \phi & c_{2k} & \phi \ldots \phi \\ \ldots\ldots & \ldots & \ldots\ldots \\ \phi \ldots \phi & c_{pk} & \phi \ldots \phi \end{bmatrix}$$

which are non-null in one column only (in this case the kth column).

$C^{(k)*}$ is found either by inspecting its graph (see the following figures)

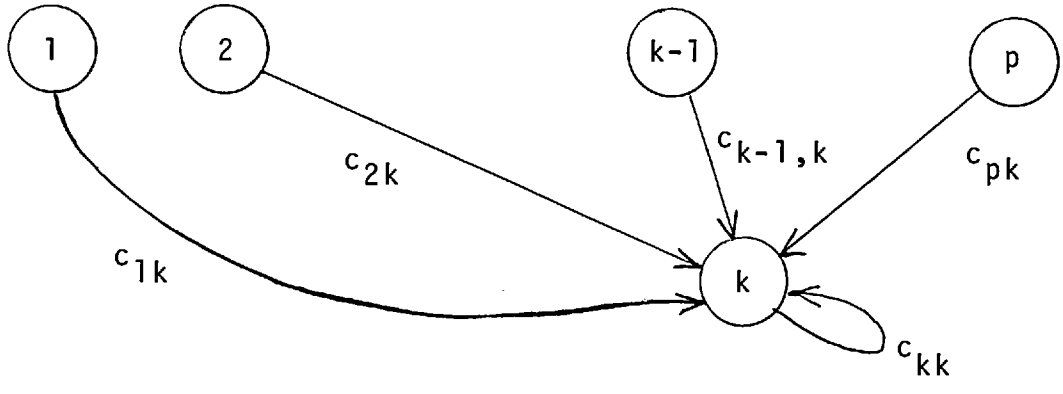


Fig. 1(a) - Graph of $C^{(k)}$

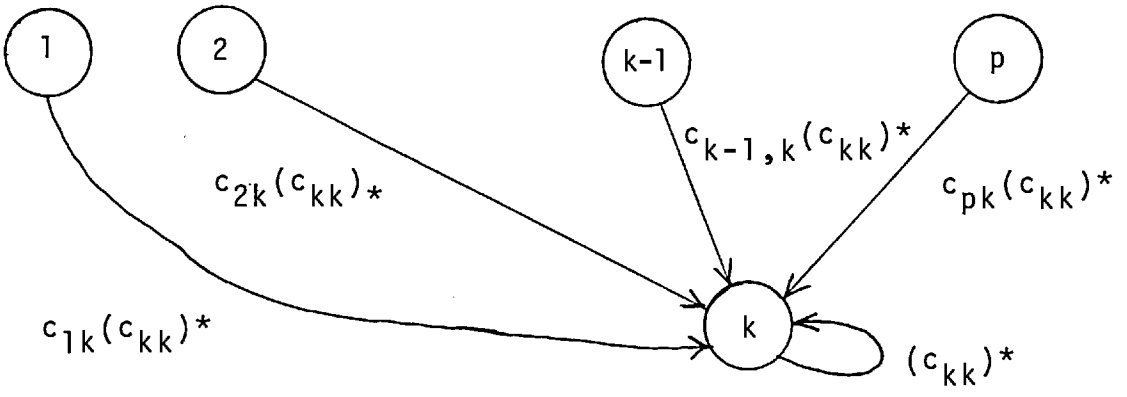


Figure 1(b) - Graph of $C^{(k)*}$

or algebraically as follows:

We express $C^{(k)}$ as

$$C^{(k)} = \underline{c}_{ok}\underline{e}_{ko}$$

Then $$C^{(k)*} = (\underline{c}_{ok}\underline{e}_{ko})^*$$

which, by 2.5, $$= E + \underline{c}_{ok}(\underline{e}_{ko}\underline{c}_{ok})^*\underline{e}_{ko} \ .$$

$$\therefore \quad C^{(k)*} = E + \underline{c}_{ok}(c_{kk})^*\underline{e}_{ko} \ . \qquad (3.1)$$

Thus $C^{(k)*}$ differs from the unit matrix only in its kth column.

Similarly the star of a row matrix

$$R^{(k)} = \underline{e}_{ok}\underline{r}_{ko}$$

which is non-null in the kth row only is

$$R^{(k)*} = E + \underline{e}_{ok}(r_{kk})^*\underline{r}_{ko} \ . \qquad (3.2)$$

We shall now show how (3.1) and (3.2) are exploited to derive the standard elimination methods of linear algebra.

3.2 The Jordan Product Forms

(i) Column Decomposition

To obtain a product form for A*, we first set $A^{(0)}=A$ and express $A^{(0)}$ in terms of its column vectors:

$$A^{(0)} = \sum_{j=1}^{p} \underline{a}_{oj}^{(0)} \underline{e}_{jo} . \tag{3.3}$$

Then we express $A^{(0)}$ as the matrix sum

$$A^{(0)} = C^{(1)} + S^{(1)}$$

$$\text{where} \quad C^{(1)} = \underline{a}_{o1}^{(0)} \underline{e}_{1o} \text{ and } S^{(1)} = \sum_{j=2}^{p} \underline{a}_{oj}^{(0)} \underline{e}_{jo} . \tag{3.4}$$

Hence, from (2.4),

$$\left.\begin{aligned} A^{(0)*} &= (C^{(1)} + S^{(1)})^* = A^{(1)*}C^{(1)*} \\ \text{where} \quad A^{(1)} &= C^{(1)*}S^{(1)} . \end{aligned}\right\} \tag{3.5}$$

Now since the first column of $S^{(1)}$ is null, the first column of $A^{(1)}$ is null also, so $A^{(1)}$ can in turn be expressed as the sum

$$A^{(1)} = C^{(2)} + S^{(2)}$$

$$\text{where} \quad C^{(2)} = \underline{a}_{o2}^{(1)} \underline{e}_{2o} \quad \text{and} \quad S^{(2)} = \sum_{j=3}^{p} \underline{a}_{oj}^{(1)} \underline{e}_{jo} \tag{3.6}$$

and applying 2.4 again,

$$A^{(1)*} = (C^{(2)} + S^{(2)})^* = A^{(2)*}C^{(2)*}$$

$$\text{where} \quad A^{(2)} = C^{(2)*}S^{(2)} . \tag{3.7}$$

Continuing in a similar manner, setting

$$\left.\begin{aligned} C^{(k)} &= \underline{a}_{ok}^{(k-1)} \underline{e}_{ko}, \quad S^{(k)} = \sum_{j=k+1}^{p} \underline{a}_{oj}^{(k-1)} \underline{e}_{jo} \\ A^{(k)} &= C^{(k)*}S^{(k)}, \quad (k = 1,2,\ldots p) \end{aligned}\right\} \tag{3.8}$$

we obtain

$$A^{(k-1)*} = A^{(k)*}C^{(k)*}, \quad (k = 1,2,\ldots p) . \qquad (3.9)$$

Now $A^{(p)}$ is null, so $A^{(p)*} = E$. Therefore 3.9 gives

$$\boxed{A^* = C^{(p)*}\, C^{(p-1)*} \ldots C^{(1)*} .} \qquad (3.10)$$

(ii) Row Decomposition

The product form (3.10) was derived by repeated application of the relation (2.4), $(P+Q)^* = (P^*Q)^*P^*$, to column decompositions. Alternatively, it is possible to apply (2.3), $(P+Q)^* = P^*(QP^*)^*$, to row decompositions: Corresponding to (3.8) - (3.10), if we set

$$R^{(k)} = \underline{e}_{ok}\underline{a}_{ko}^{(k-1)}, \quad T^{(k)} = \sum_{i=k+1}^{p} \underline{e}_{oi}\underline{a}_{io}^{(k-1)} ,$$

and

$$A^{(k)} = T^{(k)}R^{(k)*}, \quad (k = 1,2,\ldots p) \qquad (3.11)$$

then

$$A^{(k-1)*} = R^{(k)*}A^{(k)*}, \quad (k = 1,2,\ldots p) \qquad (3.12)$$

hence

$$\boxed{A^* = R^{(1)*}R^{(2)*} \ldots R^{(p)*} .} \qquad (3.13)$$

We describe (3.10) and (3.13) as the Jordan product forms for A^*.

3.3 The Gauss Product Form

In order to derive a product form analogous to the Gauss product form it is sufficient to apply the row and column decomposition methods, defined above, alternately to the successive matrices $A^{(k)}$.

We again consider a p×p matrix $A^{(0)}$, to which we first apply the row decomposition

$$A^{(0)} = R^{(1)} + T^{(1)}$$

where $R^{(1)} = \underline{e}_{o1}\underline{a}_{1o}^{(0)}$ and $T^{(1)} = \sum_{i=2}^{p} \underline{e}_{oi}\underline{a}_{io}^{(0)}$. (3.14)

Whence

$$A^{(0)*} = (R^{(1)} + T^{(1)})^* = R^{(1)*}S^{(1)*}$$

where

$$S^{(1)} = T^{(1)}R^{(1)*} . \tag{3.15}$$

We note that since the first row of $T^{(1)}$ is null, the first row of $S^{(1)}$ is null also. We now perform the column decomposition

$$S^{(1)} = C^{(1)} + A^{(1)}$$

where

$$C^{(1)} = \underline{s}_{o1}^{(1)}\underline{e}_{1o} \quad \text{and} \quad A^{(1)} = \sum_{j=2}^{p} \underline{s}_{oj}^{(1)}\underline{e}_{jo} . \tag{3.16}$$

Whence

$$S^{(1)*} = (C^{(1)} + A^{(1)})^* = (C^{(1)*}A^{(1)})^*C^{(1)*} . \tag{3.17}$$

Here both the first column and the first row of $A^{(1)}$ are null; and since only the first column of $C^{(1)}$ is non-null, in (3.17) we have $C^{(1)*}A^{(1)} = (E + C^{(1)*}C^{(1)})A^{(1)} = A^{(1)}$. Therefore (3.17) simplifies to

$$S^{(1)*} = A^{(1)*}C^{(1)*} \tag{3.18}$$

so (3.15) gives

$$A^{(0)*} = R^{(1)*}A^{(1)*}C^{(1)*} . \tag{3.19}$$

Continuing in a similar fashion, setting

$$R^{(k)} = \underline{e}_{ok}\underline{a}_{ko}^{(k-1)}, \quad T^{(k)} = \sum_{i=k+1}^{p} \underline{e}_{oi}\underline{a}_{io}^{(k-1)}, \quad S^{(k)} = T^{(k)}R^{(k)*}$$

$$C^{(k)} = \underline{s}_{ok}^{(k)}\underline{e}_{ko}, \quad A^{(k)} = \sum_{j=k+1}^{p} \underline{s}_{oj}\underline{e}_{jo} \tag{3.20}$$

we obtain

$$A^{(k)*} = R^{(k+1)*} A^{(k+1)*} C^{(k+1)*} \tag{3.21}$$

$$(k = 1,2,\ldots p) .$$

At the p^{th} stage $A^{(p)}$ and $C^{(p)}$ are null, so (3.21) gives

$$\boxed{A^* = R^{(1)*} R^{(2)*} \ldots R^{(p)*} C^{(p-1)*} C^{(p-2)*} \ldots C^{(1)*}.} \tag{3.22}$$

This decomposition process is illustrated in Figure 2, which shows the disposition of the non-null elements of the successive $R^{(k)}$, $C^{(k)}$ and $A^{(k)}$ matrices.

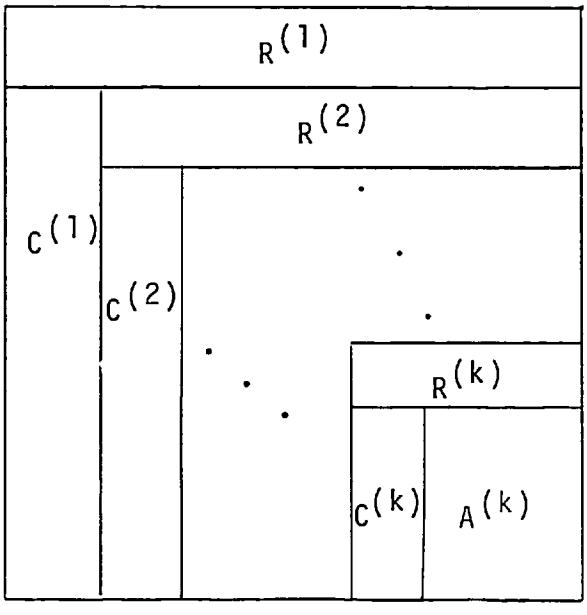


Fig. 2 Triangular Decomposition

4. Algorithms from the Product Forms

The product forms (3.10), (3.13) and (3.22) immediately yield algorithms for computing the minimal solution Y=A*B of the equation Y=AY+B. These algorithms are now presented in more detail.

4.1 Jordan Elimination Method

Firstly, we note that if we substitute $C^{(k)} = \underline{a}_{ok}^{(k-1)} \underline{e}_{ko}$ (from (3.8) into (3.1)), and then use the definition of $A^{(k)}$ in (3.8), we get

$$A^{(k)} = (E + \underline{a}_{ok}^{(k-1)} (a_{kk}^{(k-1)})^* \underline{e}_{ko}) (\sum_{j=k+1}^{p} \underline{a}_{oj}^{(k-1)} \underline{e}_{jo})$$

$$= \sum_{j=k+1}^{p} (\underline{a}_{oj}^{(k-1)} + \underline{a}_{ok}^{(k-1)} (a_{kk}^{(k-1)})^* a_{kj}^{(k-1)}) \underline{e}_{jo} \quad . \quad (4.1)$$

The non-null columns of $A^{(k)}$ can therefore be obtained directly from those of $A^{(k-1)}$, using

$$\underline{a}_{oj}^{(k)} = \underline{a}_{oj}^{(k-1)} + \underline{a}_{ok}^{(k-1)} (a_{kk}^{(k-1)})^* a_{kj}^{(k-1)}$$

$$\text{for } k<j\leq p \quad (k = 1,2,\ldots p-1) \quad . \quad (4.2)$$

To find $A^*B = C^{(p)*} C^{(p-1)*} \ldots C^{(1)*}B$, we form the sequence

$$B^{(0)} = B, \; B^{(k)} = C^{(k)*}B^{(k-1)} \quad (k = 1,2,\ldots p) \quad . \quad (4.3)$$

The final term gives the required solution: $B^{(p)} = A^*B$. We note that from (4.1), the successive $B^{(k)}$ matrices are given by

$$B^{(0)} = B, \; B^{(k)} = B^{(k-1)} + \underline{a}_{ok}^{(k-1)} (a_{kk}^{(k-1)})^* \underline{b}_{ko}^{(k-1)}$$

$$(k = 1,2,\ldots,p) \quad . \quad (4.4)$$

Hence the solution can be obtained by performing p successive transformations of A and B, the matrices $A^{(k)}$ and $B^{(k)}$ at each stage being obtained from (4.1) and (4.4) respectively. This method is analogous to the Jordan method of solving Y=AY+B in linear algebra, with $Y=A^{(k)}Y+B^{(k)}$ (k = 1,2,...,p) being the equivalent system obtained after the elimination of the kth Y- variable.

Note that once p transformations of the original matrix A have been completed the contents of A have been overwritten by $C^{(1)}$, $C^{(2)}$, ... , $C^{(p)}$. I.e. after p transformations

$$A \leftarrow \left[C^{(1)} \middle| C^{(2)} \middle| \ \ldots \ \middle| C^{(p)} \right] \qquad (4.5)$$

This is very convenient if later one wishes to calculate the solution of an equation Y=AY+B for a new value of B since one need only compute the product $C^{(p)}* C^{(p-1)}* \ldots C^{(1)}*B$ using (4.4).

In linear algebra this is very often used to advantage. The matrix (4.5) is referred to by Tewarson [42] as the product form of the inverse or PFI.

When it is required to find $A^{+}=A*A$ the equations (4.2) and (4.4) take on a rather simpler form. Substituting $B^{(0)}=A$ into (4.4) we note that the final p-k columns of $B^{(k)}$ are identical to the final p-k columns of $A^{(k)}$. Moreover, since the first k columns of $A^{(k)}$ are null we can store $A^{(k)}$ and $B^{(k)}$ in the same matrix. Thus to calculate A^{+} the above

method reduces to the rather simpler form $A^+=A^{(p)}$, where

$$A^{(0)} = A,\ A^{(k)} = A^{(k-1)} + \underline{a}_{ok}^{(k-1)}\,(a_{kk}^{(k-1)})^*\underline{a}_{ko}^{(k-1)}$$
$$(k = 1,2,\ldots,p)\ . \qquad (4.6)$$

This method has been rediscovered by many authors. On the two-element Boolean algebra it is commonly known as Warshall's algorithm [43], for finding shortest paths it is often attributed to Floyd [19].

4.1.1 Triangular Matrices

For triangular matrices the relations (3.8) - (3.10) and (3.11) - (3.13) defining the Jordan product forms can be greatly simplified. Specifically, in applying the column decomposition method to a lower triangular matrix L, we have in (3.8) $C^{(k)*}\,S^{(k)} = S^{(k)}$ and hence

$$L^{(k)} = \sum_{j=k+1}^{p} \ell_{oj}^{(0)}\underline{e}_{jo}, \quad (k = 1,2,\ldots,p-1) \qquad (4.7)$$

which is simply the original matrix L with its first k columns nullified. Thus $C^{(k)}$ is formed directly from the kth column of L, and from (3.8) and (3.9) we have

$$L^* = C^{(p)*}\,C^{(p-1)*}\ldots C^{(1)*} \qquad (4.8)$$

where $C^{(k)*} = E + \underline{\ell}_{ok}(\ell_{kk})^*\,\underline{e}_{ko}$, $(k = 1,2,\ldots,p)$.

Similarly for an upper triangular matrix U, (3.11) - (3.13) give

$$U^* = R^{(1)*}R^{(2)*}\ldots R^{(p)*} \qquad (4.9)$$

where $R^{(k)*} = E + \underline{e}_{ok}(u_{kk})^*\underline{u}_{ko}$, $(k = 1,2,\ldots,p)$.

To obtain the solution $Y=L^*B$ of a lower triangular system $Y=LY+B$, (4.4) and (4.8) give the familiar <u>forward substitution method</u>

$$B^{(0)} = B,\ B^{(k)} = B^{(k-1)} + \underline{\ell}_{ok}(\ell_{kk})^*\underline{b}_{ko}^{(k-1)} \qquad (4.10)$$
$$(k = 1,2,\ldots,p)$$

which does not involve any modifications of L. For an upper triangular system Y=UY+B, (4.9) enables us to express the minimal solution as

$$U^*B = R^{(1)*} R^{(2)*} \ldots R^{(p)*}B \tag{4.11}$$

which leads to the back-substitution method

$$B^{(0)} = B,\ B^{(k)} = R^{(p-k+1)*} B^{(k-1)} \qquad (k = 1,2,\ldots,p)\,. \tag{4.12}$$

From (4.9), the $B^{(k)}$ matrices here are given by

$$B^{(k)} = B^{(k-1)} + \underline{e}_{oq}\,(u_{qq})^*\,\underline{u}_{qo}\,B^{(k-1)} \tag{4.13}$$

where $q = p-k+1$; hence they have elements

$$b_{ij}^{(k)} = \begin{cases} b_{ij}^{(k-1)} & \text{for } i \neq q \\ b_{ij}^{(k-1)} + (u_{qq})^* \sum_{r=q}^{p} u_{qr} b_{rj}^{(k-1)} & \text{for } i = q\,. \end{cases} \tag{4.14}$$

4.2 The Gaussian Elimination Method

To obtain a convenient method of calculating the successive $C^{(k)}$, $A^{(k)}$ and $R^{(k)}$ matrices we first use (3.2) in (3.20) to obtain:

$$S^{(k)} = T^{(k)}R^{(k)*} = \sum_{i=k+1}^{p} \underline{e}_{oi}(\underline{a}_{io}^{(k-1)} + a_{ik}^{(k-1)}(a_{kk}^{(k-1)})^*\underline{a}_{ko}^{(k-1)})\,. \tag{4.15}$$

Therefore, from (3.20), the $C^{(k)}$ and $A^{(k)}$ matrices are given by

$$\begin{aligned} C^{(k)} &= \underline{s}_{ok}^{(k)}\underline{e}_{ko} \\ &= \sum_{i=k+1}^{p} \underline{e}_{oi}(a_{ik}^{(k-1)} + a_{ik}^{(k-1)}(a_{kk}^{(k-1)})^* a_{kk}^{(k-1)})\underline{e}_{ko} \\ &= \sum_{i=k+1}^{p} \underline{e}_{oi}\, a_{ik}^{(k-1)}(a_{kk}^{(k-1)})^*\underline{e}_{ko} \end{aligned} \tag{4.16}$$

and

$$A^{(k)} = \sum_{j=k+1}^{p} \underline{s}_{oj}^{(k)} \underline{e}_{jo}$$

$$= \sum_{j=k+1}^{p} \sum_{i=k+1}^{p} \underline{e}_{oi} (a_{ij}^{(k-1)} + a_{ik}^{(k-1)} (a_{kk}^{(k-1)})^* a_{kj}^{(k-1)}) \underline{e}_{jo}$$

$$= \sum_{j=k+1}^{p} \sum_{i=k+1}^{p} \underline{e}_{oi} (a_{ij}^{(k-1)} + c_{ik}^{(k)} a_{kj}^{(k-1)}) \underline{e}_{jo} \; . \qquad (4.17)$$

The matrix $R^{(k)}$ is already defined directly in terms of $A^{(k-1)}$ by (3.20).

Since the non-null elements of $R^{(1)}$, $R^{(2)},\ldots,R^{(k)}$ and $C^{(1)}$, $C^{(2)},\ldots,C^{(k)}$ and $A^{(k)}$ all occupy different positions (see figure 2), all the $R^{(k)}$ and $C^{(k)}$ matrices can be computed and recorded simply by performing p-1 successive transformations of the original A- matrix. Writing $M^{(0)}=A$, we compute $M^{(k)}$ $(k = 1,2,\ldots,p-1)$, where the elements of $M^{(k)}$ are obtained using successively

$$m_{ij}^{(k)} = \begin{cases} m_{ik}^{(k-1)} \; (m_{kk}^{(k-1)})^* & \text{for } k<i\leq p; \quad j=k \\ m_{ij}^{(k-1)} + m_{ik}^{(k)} \; m_{kj}^{(k-1)} & \text{for } k<i,j\leq p \\ m_{ij}^{(k-1)} & \text{otherwise.} \end{cases} \qquad (4.18)$$

On termination $M^{(p-1)}$ contains the non-null elements of $R^{(1)},R^{(2)},\ldots,R^{(p)}$ and $C^{(1)},C^{(2)},\ldots,C^{(p-1)}$ in their appropriate positions (see figure 3). In linear algebra $M^{(p-1)}$ is considered an extremely useful form of the inverse, particularly for sparse matrices and is referred to as the elimination form of the inverse or EFI [42].

$M^{(p-1)}$ =

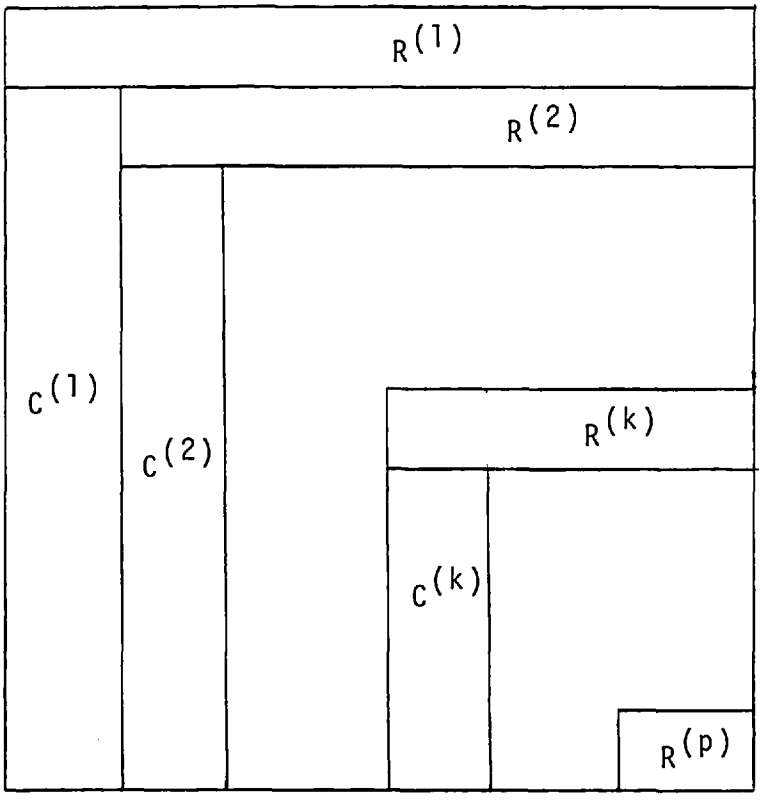


Fig. 3

To complete the solution of the equation Y=AY+B we need to calculate the product (3.22). It will be observed that the $R^{(k)}$ matrices together form an upper triangular matrix

$$U = \sum_{k=1}^{p} R^{(k)}$$

whose closure by (4.9) is

$$U^* = R^{(1)*}\ldots R^{(p)*} \tag{4.19}$$

and that the $C^{(k)}$ matrices form the strictly lower triangular matrix

$$L = \sum_{k=1}^{p-1} C^{(k)}$$

whose closure by (4.8) is

$$L^* = C^{(p-1)*}C^{(p-2)*}\ldots C^{(1)*} . \tag{4.20}$$

Hence the minimal solution of $Y = AY+B$ is

$$Y = A^*B = U^*L^*B. \tag{4.21}$$

The required solution can therefore be derived by applying the forward substitution method (4.10) followed by the back substitution method (4.14) which give in turn L^*B and $Y = U^*L^*B$. The above procedure is analogous to the Gauss elimination method in linear algebra.

4.2.1 Calculating A*

To complete the calculation of A^* we need to calculate the product (3.22) which we can do in the order

$$A^* = (R^{(1)*}(R^{(2)*}(\ldots(R^{(p)*}(C^{(p-1)*}(\ldots(C^{(2)*}C^{(1)*})\ldots)$$

Note that (other than e elements on the diagonal) the non-null elements of the accumulated product and the non-null elements of the remaining factors do not intersect at any stage of the product. Thus the product can be performed by transforming $M^{(p-1)}$ to A^* in 2p-2 steps, overwriting the contents of $M^{(p-1)}$ at each step.

Accordingly, let

$$B^{(1)} = M^{(p-1)}$$

then $B^{(k)}$, $k = 2,3,\ldots\ p-1$, are obtained by applying the forward substitution method (4.10):

$$b_{ij}^{(k)} = \begin{cases} b_{ij}^{(k-1)} + b_{ik}^{(k-1)}\ b_{kj}^{(k-1)} & j<k<i \\ b_{ij}^{(k-1)} & \text{otherwise .} \end{cases} \qquad (4.22)$$

Finally $A^* = B^{(2p-1)}$ where $B^{(k)}$ for $k = p,p+1,\ldots,2p-1$ are obtained by applying the back substitution method (4.14):

$$b_{ij}^{(k)} = \begin{cases} b_{ij}^{(k)} & i\neq q \\ b_{qj}^{(k-1)} + (b_{qq}^{(k-1)})^* \sum_{r=q}^{p} b_{qr}^{(k-1)}\ b_{rj}^{(k-1)}, & i=q,\ j<q \\ b_{qj}^{(k-1)} + (b_{qq}^{(k-1)})^* \sum_{r=q+1}^{p} b_{qr}^{(k-1)}\ b_{rj}^{(k-1)}, & i=q,\ j\geq q \end{cases}$$

where $q = 2p-k$. (4.23)

4.2.2 Calculation of Submatrices of A* - Aitken's Method

It is often necessary to find the intersection of each row $\underline{a}^*_{io}$ with each column $\underline{a}^*_{oj}$ where $i\in V$ and $j\in W$ for some subsets V and W of $\{1,2,\ldots,p\}$.

To solve this problem we observe that any $s\times t$ submatrix of A* can be expressed as

$$H = PA^*Q \qquad (4.24)$$

where P is an $s\times p$ matrix composed of the s unit row vectors corresponding to nodes $i\in V$, and Q is a $p\times t$ matrix composed of the t unit column vectors corresponding to nodes $j\in W$.

The expression (4.24) can be evaluated by a method analogous to the Aitken method of linear algebra, which can be very easily explained in graph-theoretic terms. Consider the graph $G = (X,A)$ of the matrix A and suppose this graph is augmented to form a graph $G^a = (X \cup V' \cup W', A+\psi_1+\psi_2)$ where $V' = \{x'_{i_1}, x'_{i_2}, \ldots, x'_{i_s}\}$ and $W' = \{x'_{j_1}, \ldots, x'_{j_t}\}$ are sets of "duplicates" of the nodes of V and W, and ψ_1 and ψ_2 are sets of unit arcs joining nodes $x'_{i_k} \in V'$ to the corresponding node $x_{i_k} \in V$, and nodes $x_{j_k} \in W$ to the corresponding node $x'_{j_k} \in W'$. This graph (illustrated in figure 4) has matrix

$$\tilde{M}^{(0)} = \begin{bmatrix} M^{(0)} & \phi_{12} & Q \\ P & \phi_{22} & \phi_{23} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix} . \qquad (4.25)$$

where $M^{(0)} = A$.

Applying the transformations (4.18) to this matrix it is easily verified that after k steps

$$\tilde{M}^{(k)} = \begin{bmatrix} M^{(k)} & \phi_{12} & Q^{(k)} \\ P^{(k)} & \phi_{22} & H^{(k)} \\ \phi_{31} & \phi_{32} & \phi_{33} \end{bmatrix}$$

where $P^{(k)} = PR^{(1)*} \ldots R^{(k)*}$

$$Q^{(k)} = C^{(k)*} \ldots C^{(1)*} Q \qquad (4.26)$$

and $$H^{(k)} = \sum_{i=1}^{k} \underline{p}_{oi}^{(i)} \underline{q}_{io}^{(i)} .$$

Thus on termination,

$$H^{(p)} = PA^*Q. \qquad (4.27)$$

Since a number of elements of the matrices $\tilde{M}^{(k)}$ are always null we may dispense with them, and hence the algorithm reduces to the following.

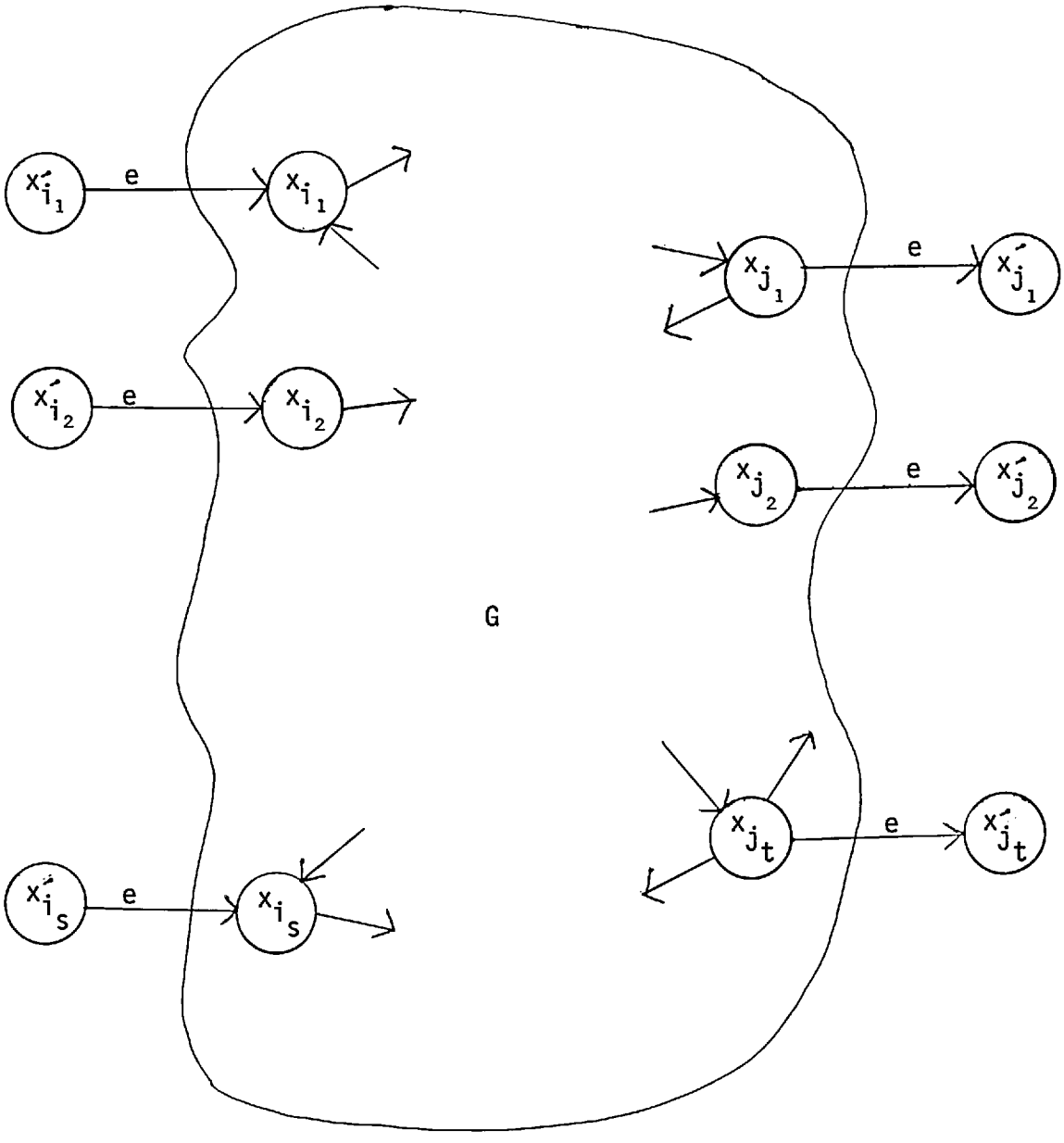


Fig. 4

Let

$$\hat{M}^{(0)} = \begin{bmatrix} M^{(0)} & Q \\ P & \phi \end{bmatrix} \tag{4.28}$$

then we form $\hat{M}^{(k)}$ $(k = 1,2,\ldots,p)$ where

$$\hat{m}_{ij}^{(k)} = \begin{cases} \hat{m}_{ik}^{(k-1)} \; (\hat{m}_{kk}^{(k-1)})^* & \text{for } k<i\leq p+s, \quad j=k \\ \hat{m}_{ij}^{(k-1)} + \hat{m}_{ik}^{(k)} \; \hat{m}_{kj}^{(k-1)} & \text{for } \begin{cases} k<i\leq p+s \\ k<j\leq p+t \end{cases} \\ \hat{m}_{ij}^{(k-1)} & \text{otherwise .} \end{cases} \tag{4.29}$$

At the pth stage of this algorithm

$$\hat{M}^{(p)} = \begin{bmatrix} M^{(p)} & Q^{(p)} \\ P^{(p)} & H^{(p)} \end{bmatrix}$$

where $H^{(p)} = PA^*Q$ is the required submatrix of A^*.

The Gaussian elimination method was first described by Carré [6,7], and is particularly important when handling sparse matrices [8]. The method we have just described for finding submatrices of A^* first appeared in Backhouse and Carré [2].

4.3 The Escalator Method

An alternative form of the formulae describing the Gaussian elimination method gives rise to the escalator method, which we describe briefly below.

Consider, once again, the first step in the derivation of the Gauss product form (eqns. (3.14)-(3.19)). Let us define the matrices A_{11}, A_{12}, A_{21} and A_{22} by

$$A = A^{(0)} = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right] = \left[\begin{array}{c} R^{(1)} \\ \hline T^{(1)} \end{array}\right]$$

Then, using (3.15), we get

$$S^{(1)} = \left[\begin{array}{c|cc} \phi & \phi & \cdots \quad \phi \\ \hline A_{21}A_{11}^{*} & A_{21}A_{11}^{*}A_{12} + A_{22} & \end{array}\right]$$

whence from (3.18) and (3.19),

$$A^{*} = \left[\begin{array}{c|c} C_{11} & C_{12} \\ \hline C_{21} & C_{22} \end{array}\right]$$

where

$$\begin{aligned} C_{11} &= A_{11}^{*} + A_{11}^{*}A_{12}C_{22}A_{21}A_{11}^{*} \\ C_{12} &= A_{11}^{*}A_{12}C_{22} \\ C_{21} &= C_{22}A_{21}A_{11}^{*} \end{aligned} \qquad (4.31)$$

and

$$C_{22} = (A_{21}A_{11}^{*}A_{12} + A_{22})^{*} .$$

These formulae applied recursively to a sequence of matrices of orders p×p, (p-1)×(p-1),...,1×1 yield a method of calculating A* equivalent to Gaussian elimination.

Alternatively we can apply the formulae (4.31) to find successively the star of a 1×1 matrix, a 2×2 matrix

etc. as shown in figure 5. In this form the corresponding shortest path algorithm is known as Dantzig's algorithm [14].

Finally, as with all our formulae, we are not constrained to split the matrix A into a row, a column and the remainder in applying (4.31). Instead we could use a "binary" splitting technique to split the matrix A (roughly) into 4 equal sized matrices. Munro [32] has applied this technique to triangular matrices in order to take advantage of a recently introduced method of matrix multiplication [41].

4.4 Woodbury's Formula

Our method of deriving product forms for closure matrices, using (2.3)-(2.5), is based on the same principles as a method discussed by Householder [26] for finding inverse matrices in linear algebra, involving repeated use of the formula:

$$(B + URV^T)^{-1} = B^{-1} - B^{-1}U(R^{-1} + V^TB^{-1}U)^{-1}V^TB^{-1} . \qquad (4.32)$$

Indeed, by combining our relations (2.4) and (2.5) we obtain the analogous formula:

$$(A + USV^T)^* = A^* + A^*U(SV^TA^*U)^*SV^TA^* \qquad (4.33)$$

which can be verified as follows:

$$\begin{aligned} (A + USV^T)^* &= (A^*USV^T)^*A^* && \text{(by (2.4))} \\ &= (E + A^*U(SV^TA^*U)^*SV^T)A^* && \text{(by (2.5))} \\ &= A^* + A^*U(SV^TA^*U)^*SV^TA^*. \end{aligned}$$

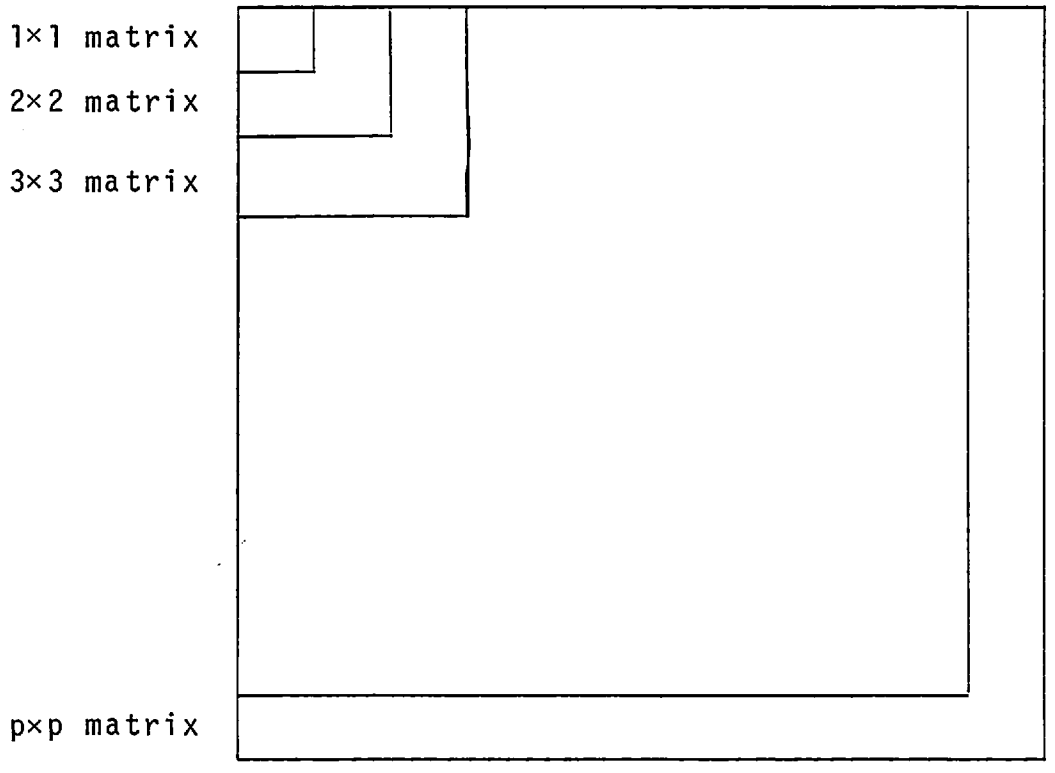


Fig. 5

To demonstrate that (4.32) and (4.33) are analogous, we replace the symbolism M* by $(I-M)^{-1}$ in (4.33) which gives

$$(I-A-USV^T)^{-1} = (I-A)^{-1}+(I-A)^{-1}U(I-SV^T(I-A)^{-1}U)^{-1}SV^T(I-A)^{-1} \tag{4.34}$$

or

$$(I-A-USV^T)^{-1} = (I-A)^{-1}+(I-A)^{-1}U(S^{-1}-V^T(I-A)^{-1}U)^{-1}V^T(I-A)^{-1} \tag{4.35}$$

If in (4.35) we set I-A = B, and S = -R, we immediately obtain (4.32). It would have been possible to derive our product forms from (4.33), but it is more convenient to apply (2.3) - (2.5) separately.

As in linear algebra, the direct application of (4.33) is not usually to be recommended as a practical method of computing closure matrices, but it is sometimes useful for finding the modification of a closure matrix A* which results from a change of a single element of A. In particular, from (4.33) the modification of A* caused by adding σ to the element a_{ij} of A is given by

$$(A + \underline{e}_{oi}\sigma\underline{e}_{jo})^* = A^* + \underline{a}^*_{oi}(\sigma a^*_{ji})^*\sigma\underline{a}^*_{jo} \; . \tag{4.36}$$

A concrete form of (4.36) has been derived from graph-theoretic considerations by Murchland [33] and Rodionov [36] who used it to calculate the changes in distances in a transportation network when one of its arc lengths is reduced.

Note also that (4.32) forms the basis of "Kron's method of tearing", which is sometimes found useful in linear algebra for finding the inverse of matrices having particular structural properties [42].

4.5 A Comparison of Aitken and Jordan Elimination

In this section we shall compare the Jordan elimination method ((4.3) and (4.4)) with Aitken's method ((4.29)) for finding particular submatrices of A*. The relationship between the two methods has been extensively studied by linear algebraists [42] and our contribution is merely to translate an important theorem [42, Pp.97-100], well-known to linear algebraists, into its appropriate form in regular algebra.

Recall ((4.30)) that in Aitken's method we need to store the non-null elements of a matrix $\hat{M}^{(k)}$ ($k = 1,2,\ldots,p$) which consists of the matrix $M^{(k)}$ bordered by the three matrices $P^{(k)}$, $Q^{(k)}$ and $H^{(k)}$. We shall assume that a relatively small number of elements of A* are required and hence that the storage required by and the manipulations on $P^{(k)}$, $Q^{(k)}$ and $H^{(k)}$ may be disregarded.

Similarly using Jordan elimination (§4.1) we assume that the matrix $B^{(k)}$ may be disregarded. We therefore need to investigate the relationship between the matrices

$$J^{(k)} = \left[\begin{array}{c|c|c|c|c} C_J^{(1)} & C_J^{(2)} & \cdots & C_J^{(k)} & A_J^{(k)} \end{array}\right] \qquad (4.37)$$

and $M^{(k)}$ (see figure 2) for $k = 1,2,\ldots,p-1$.

Note that, for sparse matrices, the fill-in (i.e. creation of new non-null entries) in the matrices $J^{(k)}$ usually exceeds the number of non-null entries in $C_J^{(k)}$ (which is no longer required) and thus the release of this storage is not

normally exploited in linear algebra. A similar statement applies to the matrix $M^{(k)}$.

In order to compare the storage requirements of $M^{(k)}$ and $J^{(k)}$ it is technically simpler to compare the matrices

$$M^{(k)'} = \left[\begin{array}{c|c|c|c} R_G^{(1)*} & & & \\ \hline & R_G^{(2)*} & & \\ \hline C_G^{(1)*} & & & \\ \hline & C_G^{(2)*} & & \\ \hline & & R_G^{(k)*} & \\ \hline & & C_G^{(k)*} & A_G^{(k)} \end{array}\right] \qquad (4.38)$$

and

$$J^{(k)'} = \left[\begin{array}{c|c|c|c|c} C_J^{(1)*} & C_J^{(2)*} & \cdots\cdots & C_J^{(k)*} & A_J^{(k)} \end{array}\right] \qquad (4.39)$$

(For convenience in the above matrices we use the notation C* (R*) to denote the non-null elements of the closure of the column (row) matrix C (R) excluding the e's on the diagonal.)

Note that $M^{(k)'}$ ($J^{(k)'}$) has the same number of non-null elements as $M^{(k)}$ ($J^{(k)}$) because, other than e's on the diagonal, R* (C*) has the same number of non-null elements as R (C) for any row (column) matrix R (C).

The analogue in linear algebra of the following lemma is given in [42,Pp97-100].

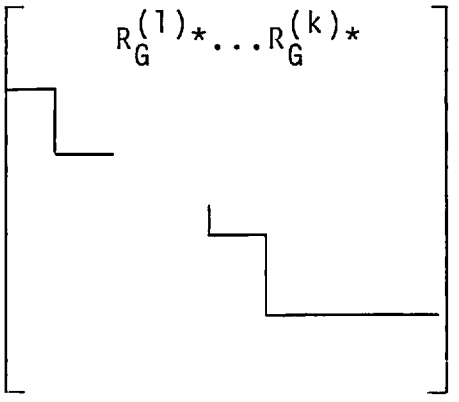
Lemma 4.1 If $M^{(k)'}$ is given by (4.38) then

$$J^{(k)'} = \begin{bmatrix} C_G^{(1)*} & & & R_G^{(1)*}R_G^{(2)*}\ldots R_G^{(k)*} \\ & C_G^{(2)*} & & \\ & & C_G^{(k)*} & A_G^{(k)} \end{bmatrix}$$

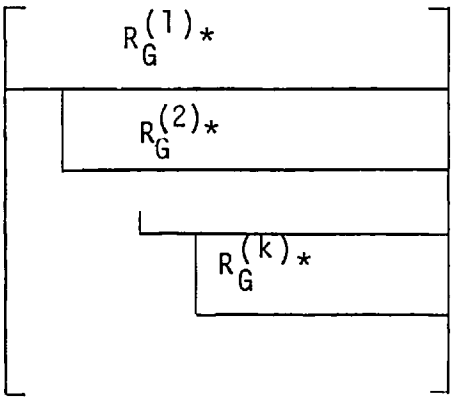
Proof The lemma is very easily proved by induction on k. In fact intuitively the result is obvious, since in Jordan elimination one performs the back-substitution as soon as possible, whereas in Gaussian elimination back-substitution is delayed until all elimination steps have been completed.

For k=0 the result is immediate. If it is true for k-1, then certainly $M^{(k)'}$ and $J^{(k)'}$ are equal below the diagonal and in rows below row k, simply by comparing (4.2) and (4.18). Indeed from (4.2) and (4.18), $M^{(k)'}$ and $J^{(k)'}$ differ only in those entries (i,j) where $i \leq k$ and $j \geq k$.

This is where the matrix



differs from



Now in $J^{(k)\prime}$ these elements are given by (c.f. (4.2))

$$a_{ij}^{\prime(k)} = \begin{cases} (a_{kk}^{\prime(k-1)})^* & \text{if } i=j=k \\ (a_{kk}^{\prime(k-1)})^* \, a_{kj}^{\prime(k-1)} & \text{if } i=k,\ j>k \\ a_{ij}^{\prime(k-1)} + a_{ik}^{\prime(k-1)} \, (a_{kk}^{\prime(k-1)})^* \, a_{kj}^{\prime(k-1)} & \text{if } i<k,\ j>k. \end{cases} \tag{4.40}$$

But since, by the induction hypothesis,

$$a_{kj}^{\prime(k-1)} = [R_G^{(k)}]_{kj} \qquad \text{for } j \geq k$$

and $$a_{ij}^{\prime(k-1)} = [R_G^{(1)*}\ldots R_G^{(k-1)*}]_{ij} \qquad \text{for } i<k,\ j \geq k$$

it is clear by inspection of (4.40) that

$$\begin{aligned} a_{ij}^{\prime(k)} &= [R_G^{(1)*}\ldots R_G^{(k-1)*}]_{ij} \\ &\quad + [R_G^{(1)*}\ldots R_G^{(k-1)*}]_{ik} [R_G^{(k)*}]_{kj} \\ &= [R_G^{(1)*}\ldots R_G^{(k-1)*} R_G^{(k)*}]_{ij} \ . \end{aligned}$$

Hence the lemma is also true for k, and so is true for all $k = 1,\ldots,p$.

For comparable implementations of Jordan elimination and Aitken's method the implications of lemma 4.1 must be carefully considered. Essentially the lemma indicates that Aitken's method will generally be better than Jordan elimination both in time and space. It is better with respect to storage because wherever the matrix $J^{(k)}$ is non-null then so also is the corresponding element in $M^{(k)}$. It is better with respect to time in that at each stage at least as many • and + operations need to have been performed (either explicitly or implicitly) using Jordan elimination as using Aitken's method.

5. The Iterative Techniques

In many applications Theorem I2.4 holds. For such applications one can use iterative techniques analogous to those in linear algebra to solve the equation

$$Y = A \cdot Y + B \tag{5.1}$$

for semi-definite matrices A. Although such techniques are not applicable to regular languages, and hence out of the scope of this thesis, they are extremely important in many practical applications and so worthy of a brief mention.

The simplest iterative technique is to set

$$Y^{(0)} = B \tag{5.2}$$

and then perform successively the transformations

$$Y^{(k)} = AY^{(k-1)} + B, \quad k = 1,2,\ldots \quad . \tag{5.3}$$

Assuming A is semi-definite and Theorem I2.4 holds, this process may be terminated after at most p steps giving

$$Y^{(p)} = A^*B \tag{5.4}$$

More generally, to solve the equation (5.1) we may split the matrix A into two matrices C and D such that

$$A = C + D \tag{5.5}$$

and with initial condition given by (5.2), iterate successively the transformation

$$Y^{(k)} = CY^{(k)} + DY^{(k-1)} + B \quad . \tag{5.6}$$

Note that the minimal solution of (5.6),

$$Y^{(k)} = C^*DY^{(k-1)} + C^*B \tag{5.7}$$

involves the closure matrix C*. It is advantageous to choose C so that the calculation of $Y^{(k)}$ does not involve modifying C. One such choice is when A is split into a strictly lower triangular matrix L and an upper triangular matrix U. This gives the method

$$\left.\begin{aligned} Y^{(0)} &= B \\ Y^{(k)} &= LY^{(k)} + UY^{(k-1)} + B \end{aligned}\right\} \tag{5.8}$$

which can be solved using the forward substitution method (4.10) to find $L^*UY^{(k-1)} + L^*B$.

The method (5.3) is analogous to Jacobi's method and (5.8) is analogous to the Gauss-Seidel method in linear algebra, as was first observed by Carré [6].

Yen [45] has described a method which consists essentially of iterating the procedure

$$\left.\begin{aligned} Y^{(2k+1)} &= UY^{(2k+1)} + Y^{(2k)} \\ Y^{(2k+2)} &= LY^{(2k+2)} + Y^{(2k+1)} \end{aligned}\right\} \tag{5.9}$$

for k = 0,1,... with initial approximation $Y^{(0)} = B$. This involves applying the back-substitution method (4.14) to find $Y^{(2k+1)}$ from $Y^{(2k)}$ followed by applying the forward substitution method (4.10) to find $Y^{(2k+2)}$.

One can easily prove that

$$Y_{5.8}^{(k)} = (L^*U + L^*)Y_{5.8}^{(k-1)} + L^*B \qquad (5.10)$$

and

$$Y_{5.9}^{(2k)} = L^*U^*Y_{5.9}^{(2k-2)} + L^*B \ . \qquad (5.11)$$

Hence, as $L^*U^* \geq L^*U + L^* \geq A$, it is clear by comparing (5.11),(5.10) and (5.3) that

$$Y \geq Y_{5.9}^{(2k)} \geq Y_{5.8}^{(k)} \geq Y_{5.3}^{(k)} \ . \qquad (5.12)$$

Since method (5.3) converges after at most p iterations if A is semi-definite we have

Lemma 5.1 If A is semi-definite and Theorem I2.4 holds, methods (5.8) and (5.9) converge after at most p iterations†.

We can strengthen this lemma by saying that Yen's method (method (5.9)) will always converge at least as quickly as, if not more quickly than, method (5.8) which in turn converges at least as quickly as, if not more quickly than, method (5.3). In fact the number of iterations of Yen's method will always be less than about half the number of iterations of method (5.3) as the next lemma states.

Lemma 5.2 Suppose $A^* = E+A+\ldots+A^m$ (i.e. method (5.3) converges after at most m iterations), then Yen's method converges after $\lfloor m/2 \rfloor + 1$ iterations.

Proof After k iterations of (5.9),

$$Y_{5.9}^{(2k)} = (L^*U^*)^k B$$

and after k iterations of (5.3)

$$Y_{5.3}^{(k)} = (E + A + A^2 + \ldots + A^k)B \ .$$

---

† A single iteration of method (5.9) consists of calculating both $Y^{(2k-1)}$ and $Y^{(2k)}$ from $Y^{(2k-2)}$.

But $A = L + U$,

$L^* = E + L + L^2 + \ldots + L^p$

and $U^* = E + U + U^2 + \ldots + U^p$.

Hence

$$Y_{5.9}^{(2(\lfloor m/2 \rfloor + 1))}$$

$$= \{(E+L+\ldots+L^p)\ (E+U+\ldots+U^p)\}^{\lfloor m/2 \rfloor + 1} B$$

$$\geq Y_{5.3}^{(m)} = \{E + (L+U)^2 + (L+U)^3 + \ldots + (L+U)^m\}B$$

(by comparing terms on both sides of the inequality).

Note that Yen's method may require much less than half the number of iterations of methods (5.3) or (5.8) for example if A=U is strictly upper triangular.

Finally we observe:

Lemma 5.3 The number of $\cdot$ and + operations used by each of the methods (5.3), (5.8) and (5.9) per iteration is the same even when sparsity of A is exploited.

Proof Each of the elements of A is multiplied into once per column of B, the methods differing only in the order in which the multiplication is performed. Each such multiplication which can lead to a modification of the (i,j)th element of Y must be added to every other multiplication which can lead to a modification of the same element of Y. Since there are identical numbers of multiplications there are also identical numbers of additions.

Corollary 5.4 Yen's method (method (5.9)) invariably uses a number of operations less than or equal to the number of operations of methods (5.3) and (5.8).

The comparison between Yen's method and methods (5.3) and (5.8) was first observed by Carré [unpublished work].

6. A Brief Comparison of the Methods

The algorithms we have given in this chapter have, as we have shown, a very wide range of applications. In the previous paragraphs we have given comparisons of the elimination methods and of the iterative methods, which are independent of the input and of the application. To compare the elimination methods with the iterative methods is however a much more difficult task since the methods differ widely in the manner in which they depend on both the input and the application. The difficulties are also compounded because we would need to compare the generally applicable methods we have given with methods tailored to a particular application (e.g. Dijkstra's [16] algorithm for finding shortest paths). This is beyond the scope of this thesis, and so we shall merely try to give a brief review of the literature on this topic.

In practice the problem of finding a particular row or column of a closure matrix should be considered separately from the problem of finding the whole closure matrix. The reason for this is that, in practice, the matrix A may be very large but may also be very sparse (i.e. contain relatively few non-null entries). The matrix A* will, however, usually be quite full. In calculating particular rows or columns it is therefore important to preserve sparsity as much as possible, in calculating A* this is just not possible.

The iterative techniques are particularly suited to preserving storage space (when calculating particular rows or

columns of A*) since they involve no modification of the original matrix A. On the other hand the elimination methods do incur an overhead on the storage space since the elimination form of the inverse (§4.2) will usually contain more non-null elements than the original matrix A. In the experience of linear algebraists [5,37,42] the elimination form of the inverse is often sparse also, and the overhead manageable. Note, however, that in the worst-case the EFI is full as figure 6 illustrates. (Indeed after eliminating node 1, $A^{(1)}$ has no null elements.) This example should not deter one from using elimination techniques since the experience of many linear algebraists [5,37,42] indicates that the worst-case analysis is of minor importance. The escalator method used with a binary splitting technique is not generally to be recommended for finding particular elements of A* for large sparse matrices A since it involves calculating the closure A^*_{11} of a large submatrix A_{11}, and A^*_{11} will usually be full. The technique is nevertheless sometimes used by linear algebraists when the use of backing store is essential [37]. Here, however, A^*_{11} is not calculated explicitly but its EFI is calculated and stored and (3.22) is used wherever A^*_{11} is required.

Comparisons of closure algorithms that appear in the literature are usually restricted to the "worst case". Such analyses can only be used as an indicator of the bottlenecks within an algorithm and should not be used as the sole measure of the algorithm's efficiency. On a worst-case analysis Yen's

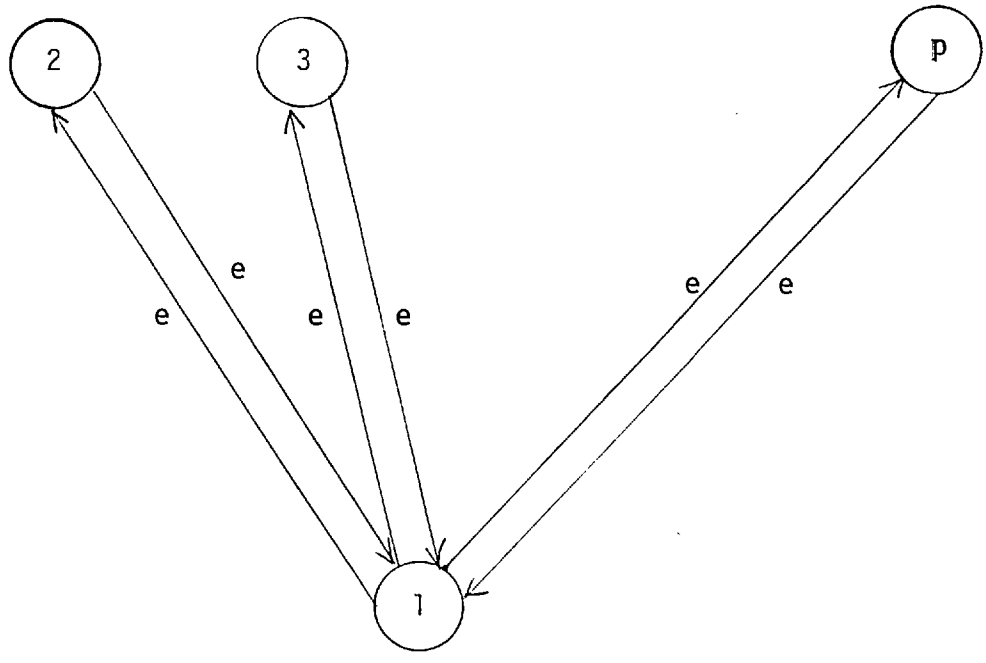


Fig. 6

iterative technique requires $\frac{1}{2}p\alpha$ • and + operations [27], where p is the number of nodes of the graph and α is the number of non-null elements. In the Jordan method the elimination process (4.2) requires $\frac{1}{2}p^3$ • and + operations (in the worst-case) and then completing the solution of Y=A•Y+B using (4.4) requires an additional p^2 • and + operations per column of B [21]. To find A* the algorithm given in §4.2.1 requires p^3 • and + operations. Using Gauss's method the elimination process (4.18) requires $\frac{1}{3}p^3$ • and + operations, and once again completing the solution of Y=AY+B requires an additional p^2 • and + operations per column of B.

Considerable effort has recently been put into trying to find algorithms to compute A* which are asymptotically better in the worst-case than the above algorithms. For Boolean matrices Munro [32] has given an algorithm which requires $O(p^{2.81})$ • and + operations, and for shortest paths Hoffmann and Winograd [25] have given an algorithm which requires $O(p^3)$ comparisons and $O(p^{5/2})$ (approx.) real additions and subtractions.

Worst-case analyses are, however, particularly derogatory to the elimination methods. Figure 6 is an example given by Johnson [27] to illustrate that Jordan's method is a $O(p^3)$ algorithm. If p is large the matrix of this graph is large and sparse and so this would imply that the elimination methods cannot exploit sparsity within a matrix. On the other hand the elimination methods are usually recommended

for solving large sparse sets of equations in linear algebra [42] and many linear algebraists consider the worst-case analyses to be of minor importance [5,37,42]. We have no reason to suspect that their experience will not also apply to path-finding problems.

Unfortunately other than worst-case analyses there is very little in the literature comparing the algorithms. In an interesting paper Fontan [20] has compared Yen's, Gauss's and Dijkstra's algorithms for large rectangular grid networks of various sizes with arc labels uniformly distributed within the range [0,10]. These networks are sparse and are similar to those commonly occurring in traffic flow problems. Fontan's results indicated that for this type of network Dijkstra's method took significantly longer than Yen's for large matrices. Gauss's method took almost as long as Dijkstra's to find one row of A*, most of the time being taken by the elimination process, but if more than one row of A* was required Gaussian elimination became rapidly more advantageous. These results are in complete contradiction to a worst-case analysis, since Dijkstra's algorithm is well-known to be $O(p^2)$ and, theoretically, should be superior to Yen's and Gauss's methods. Recently Johnson [27] has presented a new method of implementing Dijkstra's algorithm which is asymptotically better than, for example, Yen's [44] implementation of Dijkstra's algorithm. It remains to be seen whether Johnson's method is better in practice than Yen's iterative technique.

In addition Grassin and Minoux [23] have compared Dantzig's method (the escalator method) and Floyd's algorithm [19] (Jordan elimination) with a "new" algorithm for finding shortest paths through large sparse symmetric networks. Their new algorithm is a variation on Dantzig's algorithm which exploits sparsity and some additional properties of symmetric matrices. Their algorithm shows significant improvements but it is not clear to what extent this is due to exploiting sparsity and to what extent to exploiting symmetry.

Finally mention should be made of review articles and books. The paper by Dreyfus [17] is an early review of shortest-path algorithms, and an excellent recent review of the iterative techniques and Dijkstra's algorithm is given by Johnson [27]. Neither Johnson nor Dreyfus mention the elimination methods. A very full discussion of the elimination methods in the context of linear algebra can be found in the book by Tewarson [42].

7. Conclusions

In this chapter we have demonstrated the very strong connection between problems in linear algebra and various path-finding problems. This relationship is important because we can now draw upon the very considerable practical experience of linear algebraists [5,37,42] in solving such problems. Previously this analogy had been observed by Carré [6] and others. Our contribution has been to introduce an algebra

which is applicable to both matrix operations as well as operations on individual elements. This has enabled us to justify the elimination methods in a new way which, we feel, adds much greater depth to our understanding of these algorithms. We have also given an algebraic characterisation of the uniqueness of solution of equations which shows unequivocably the relationship between e.g. negative cycles in a distance matrix and singularity of a linear equation. We have also been able to give a simple comparison of the two major elimination methods (§4.5) not previously given elsewhere, and have given simple, concise comparisons of the iterative techniques (§5).

It is remarkable how much of the theory of real matrices holds in regular algebra. However one should not suppose that all concepts of linear algebra have analogues in regular algebra. The notion of linear dependence in a p-dimensional vector space is one such concept, and hence the orthogonalisation methods of linear algebra cannot be used for finding closure matrices. Nor should one suppose that all regular tautologies have analogues in linear algebra. The axiom A6 $\alpha=\alpha+\alpha$ is one example, and all other tautologies which involve this axiom in their proof do not have analogues in linear algebra. This has prompted the author to look for other generally applicable closure algorithms which improve on the elimination methods with respect to the star-height of the resulting regular expressions. The explanation of "star-height" and the results of this search are contained in the next two chapters.

III THE FACTOR MATRIX AND FACTOR GRAPH

1. Motivation - The Star-Height Problem

In this chapter and the next we present some new results in the theory of factors of regular languages. The term "factor", as we shall use it, was introduced by Conway [13], and to him are due all the fundamental results of "factor theory". Although we would hope that our results will be of value in their own right as a contribution to factor theory, our interest in this theory was motivated by an interest in the "star-height problem" of regular languages.

Any regular language may be denoted by an unbounded number of different regular expressions. Thus (a+b)* and (a*b)*a* are two expressions denoting the same language (consisting of the set of all strings of a's and b's). Different expressions denoting the same language may of course differ rather trivially, but often they are remarkably "unalike". For example (b+a(aa*b)*b)* and (b+ab)*+(a+b)*b(b+ba)*b both denote the same language, (see example 1 of this chapter), but are quite different in form from each other. It is therefore natural to seek some canonical expression denoting a particular language - wherein the "canonicality" of an expression signifies that it is the "simplest" of all expressions which denote that language. Little, if any, progress has been made in finding a canonical form for regular languages, and so, as an intermediate step, efforts have been directed towards finding a way of assigning to each regular language some measure of the "complexity" of the language.

As a measure of the complexity of a language,

Eggan's [18] definition of "star-height" would appear to be very reasonable and is generally accepted [9, 10, 11, 12, 15, 28, 29]. To define this, one first defines the star-height of a regular expression to be the maximum depth of embedded starred terms in the expression. In our earlier example $(b+a(aa^*b)^*b)^*$ has star-height 3 and $(b+ab)^*+(a+b)^*b(b+ba)^*b$ has star-height 1. The star-height of a regular language is the minimum star-height of all regular expressions which denote that language. The star-height problem is then just the problem of finding the star-height of any given regular language.

This problem was first posed in 1963 by Eggan, and has been tackled by various authors [9, 10, 11, 12, 15, 28, 29]. But, in common with many mathematical problems which are quite simply stated, its solution has not been forthcoming and it would appear to be a very difficult problem. In the next few paragraphs we have summarised those results which we consider an essential part of the repertoire of anyone who wishes to tackle this problem. We then continue to discuss, quite briefly, other results on this problem which have appeared, and indicate why these results led us to feel that Conway's factor theory was pertinent to the problem.

1.1 Previous Work

A concept which is fundamental to any study of the star-height problem, is Eggan's [18] notion of the rank of a transition graph. The rank is a measure of the loop complexity of a graph, but it is also very closely related to

the notion of star-height.

In order to define rank some additional terminology is needed. A subgraph of a graph G is a graph G_Y determined by a set $Y \subseteq X$ of the nodes of G, having just those arcs (x_i, x_j) of G between nodes x_i and x_j, both of which are in Y. A subgraph is strongly connected if there is a path from x_1 to x_2 for every ordered pair (x_1, x_2) of its nodes. A section of a graph G is a strongly connected subgraph that is not a proper subgraph of any strongly connected subgraph of G.

The rank r(G) of a transition graph G is then defined as follows:

(i) If G is not strongly connected then

a) if G has no strongly connected subgraph r(G)=0, otherwise

b) r(G) is the maximum rank of all the sections of G.

(ii) If G is strongly connected r(G) = n+1 if and only if

a) it does not have rank i for any $i \leq n$, and

b) it has a node x whose deletion from G results in a subgraph of rank n.

The above recursive definition of rank is not particularly enlightening; readers not familiar with the notion should refer to McNaughton's paper [29], (from which the above definitions were taken), for a more detailed discussion.

Now consider any recogniser (G, S, T) of the language Q. If we use, for example, the escalator method to calculate G* we obtain some regular expression α for Q. Re-

ordering the nodes of G and reapplying the escalator method to calculate G* will result in a different regular expression β for Q which, moreover, will often have a different star-height to that of α. Thus there will be some optimal ordering of the nodes of G which, when using the escalator method to calculate G*, will yield some minimal star-height expression γ for Q from the graph G. The definition of the rank of the graph G is so contrived that the star-height of γ equals the rank of G. This is expressed by Eggan's theorem [18] which is essentially the following:

Eggan's Theorem

Consider the use of the escalator method to calculate G* from a given graph G. Then

(i) for a suitable ordering of the nodes of G the resulting regular expressions for those entries $[G^*]_{ij}$ for which G is an all-admissible recogniser have star-height equal to the rank of G. For other entries (ones for which G is not an all-admissible recogniser) the resulting regular expressions have star-height less than or equal to the rank of G.

(ii) For all other orderings of the nodes the resulting regular expressions for entries $[G^*]_{ij}$, for which G is an all-admissible recogniser, have star-height greater than or equal to the rank of G.

A converse to this result was also observed by Eggan, namely that to every regular expression there natur-

ally corresponds a graph G having rank equal to the star-height of the expression. Thus one obtains the following corollary:

Corollary [18, 29] The star-height of a regular language equals the smallest rank of all transition graphs which recognise the language.

This corollary to Eggan's theorem immediately suggests an approach to the problem of determining the star-height of a regular language which is to find a method of obtaining a graph of least rank which is a recogniser of the language; it is this approach that almost all papers on the star-height problem have adopted. (Note that in the literature [29] the above corollary is usually referred to as "Eggan's theorem" - for reasons which will emerge we would like to remove the emphasis from this corollary.)

The most significant contribution to the star-height problem has been made in two papers by McNaughton [28,29]. In the first of the two, McNaughton studies languages whose semi-group is a pure group. For this class of languages McNaughton solved the star-height problem completely, although his solution involved enumerating a possibly rather large number of different graphs. However, for the subclass of this class consisting of languages for which the finite-state machine has a unique terminal state, he showed that the star-height of the language equals the rank of the finite-state machine. In spite of this result any connection between the structure of the semigroup of the language and its star-height would seem to be very illusory.

In order to establish his method McNaughton introduced the idea of a pathwise homomorphism between two graphs, and then proved a simple but fundamental theorem on the ranks of the graphs. As we shall need this result in the next chapter we state the theorem below.

Definition A pathwise homomorphism is defined as a mapping γ from the nodes and arcs of the transition graph G onto the nodes and arcs of G', such that $\gamma(x)$, for any node x of G, is a node of G' and the following two conditions hold between the arcs of G and G':

(PH1): For each arc B of G labelled b and leading from node x_1 to node x_2, either $\gamma(B)$ is a node of G' and $\gamma(x_1) = \gamma(x_2) = \gamma(B)$ or there is an arc $\gamma(B)$ in G' labelled b and leading from $\gamma(x_1)$ to $\gamma(x_2)$.

(PH2): If w is a word taking node x_1' to node x_2' in G' there are nodes x_1 and x_2 in G with $\gamma(x_1) = x_1'$ and $\gamma(x_2) = x_2'$ such that w takes node x_1 to node x_2 in G.

McNaughton's theorem is the following:

McNaughton's Pathwise Homomorphism Theorem If there is a pathwise homomorphism γ from G onto G' then the rank of G' is less than or equal to the rank of G.

Following McNaughton's work a number of papers were written by Cohen [9,10,11] and by Cohen and Brzozowski [12]. Some of these papers were concerned with extending McNaughton's work on pure group languages (to "reset-free" and "permutation-free" languages and to languages with the "finite intersection property"), the basic idea being to apply a combination of McNaughton's pathwise homomorphism theorem and the corollary to Eggan's theorem. Others pro-

vided more empirical results on the star-height problem.

However, with the exception of Eggan's theorem and McNaughton's pathwise homomorphism theorem, progress towards solving the star-height problem has been rather slow and fragmentary.

Our own first step in tackling the problem was as follows: Eggan's work showed that one closure algorithm (the escalator method) yielded regular expressions having star-height characterised by the rank of the graph. Is it possible that other closure algorithms yield regular expressions characterised by some other property of the graph and possibly even offer an improvement over the escalator method? In particular, do any of the elimination methods of Chapter II (e.g. Jordan elimination) offer such an improvement?

It is not long before one realises that this is not so, and that the rank is indeed the appropriate characteristic of a graph when applying any "elimination method". This statement is made precise in Appendix B where we give a general formulation of an "elimination method" and use this to prove the following theorem.

Theorem B4 If an elimination method is used to find G^* for a graph G, then G^* will contain regular expressions having star-height at least equal to the rank of G.

We have observed, however, that the elimination methods are all based on regular tautologies having analoques in linear algebra, but that not all regular tautologies have such analogues. This suggested two problems:

a) Can we invent new closure algorithms which do not have analogues in linear algebra? and

b) does the rank of a graph represent the "best" one can do using these new algorithms, or can we do better than the rank (i.e. obtain regular expressions for the entries $[G^*]_{ij}$ of star-height less than the rank of G)?

The main stumbling block to this approach is problem a), since it would appear extremely difficult to solve this problem in full generality. (Otherwise, no doubt, such algorithms would already have been published.) We were therefore obliged to seek a new closure algorithm which could be applied to particular classes of graphs, e.g. finite-state machines, each graph in the class being somehow naturally defined by a given language. Yet once again, for graphs such as the finite-state machine or semigroup machine, such algorithms seem impossible to find; thus we were forced to look for other "naturally defined" graphs to which such an algorithm could be applied. Cohen and Brzozowski [12] introduced the notion of a "subset automaton" but the difficulty in studying this class of graphs is that even for very simple regular languages the size of the "subset automaton" may be immense, thus precluding any empirical investigations.

1.2 The Relevance of Factor Theory

The class of graphs which we eventually decided to study are called "factor graphs". The idea of studying factor graphs came from reading McNaughton's paper [29] and

the chapter in Conway's book [13] on factor theory. Let us use Conway's terminology and call G.H.K a subfactorization of a language Q if G.H.K $\subseteq$ Q, and call H a factor of Q if there is no H' $\supset$ H such that G.H'.K $\subseteq$ Q. (Thus H is in a sense maximal). In his paper, McNaughton presented an extremely useful technique for establishing a lower bound on the star-height of a given language. The technique involves spotting particular regular languages and showing that in any recogniser of Q these languages define nodes of the recogniser which are connected by arcs having loop complexity at least equal to the conjectured lower bound. Now, in general, subfactorizations of a language Q are mathematically unmanageable; but Conway showed that factors are manageable and, moreover, exhibit some remarkable properties. One such property particularly relevant to our aims is that the factors are all the entries in a matrix, denoted $\boxed{Q}$ and called the factor matrix, which is the closure $(C_{max} + L_{max})^*$ of a constant + linear matrix. Thus the factors naturally define a transition graph, which is, moreover, a recogniser for Q.

The matrix $C_{max} + L_{max}$, as it turns out, is not very useful for our purposes, but it is a stepping stone to proving that there is a unique minimal constant + linear matrix G_Q, which we call the factor graph of Q, such that $G_Q^* = \boxed{Q}$.

One of the main reasons for studying a problem like the star-height problem is the possible side-benefits that one can gain on the way. In trying to attack the problem using factor theory, we have been particularly on the lookout

for such benefits; but also we are expressing a belief that a mathematical solution to the problem, which is not an impracticable "enumerative" solution, does exist. Rather disappointingly, the algorithm we shall present does not always yield a minimal star-height expression for a given regular language. Nevertheless we feel it is important as a contribution to our understanding of factor theory and because it offers a new approach to the problem, one which may well be more successful than earlier approaches using Eggan's theorem.

The presentation of the algorithm to determine G_Q^* occupies both this chapter and the next. This chapter is devoted to the fundamental properties of factors (due to Conway [13]) and to introducing the factor graph, G_Q, of a regular language Q and providing an algorithm to calculate G_Q. In the next chapter we introduce the notion of separability of factors and exploit this notion to derive an algorithm to calculate the closure G_Q^* of the factor graph. We also prove that the algorithm yields regular expressions for Q of star-height less than or equal to the rank of G_Q, and, as we demonstrate, in many cases strictly less than the rank of G_Q. Finally we discuss how the results could be extended in a further attack on the star-height problem.

2. ℓ-classes, r-classes and c-classes

We shall assume that the reader is familiar with the basic results on finite-state machines, to be found in Rabin and Scott [35], and the method of derivatives due to Brzozowski [3].

The purpose of this section is merely to summarise those results which we shall require later, and to define the ℓ, r and c-classes of a regular language Q.

2.1 Machine, Anti-machine and Semigroup

Let $Q \subseteq V^*$ be any language. Q naturally defines three equivalence relations on V^* - Q_ℓ, Q_r and Q_c - given by:

$$xQ_\ell y \Leftrightarrow (\forall z \in V^*,\ zx \in Q \Leftrightarrow zy \in Q)$$

$$xQ_r y \Leftrightarrow (\forall z \in V^*,\ xz \in Q \Leftrightarrow yz \in Q)$$

$$xQ_c y \Leftrightarrow (\forall u,v \in V^*,\ uxv \in Q \Leftrightarrow uyv \in Q).$$

These are, of course, the usual left-invariant equivalence relation, right-invariant equivalence relation and congruence relation introduced by Rabin and Scott [35].

The fundamental theorem linking these relations to regular languages is the following:

Theorem 2.1 A language $Q \subseteq V^*$ is regular $\Leftrightarrow$ the relation Q_ℓ is of finite index $\Leftrightarrow$ the relation Q_r is of finite index $\Leftrightarrow$ the relation Q_c is of finite index.

Definition Let Q be a regular language. By theorem 2.1, each of the relations Q_ℓ, Q_r and Q_c partitions V^* into a finite number of equivalence classes. We shall call an equivalence class modulo Q_ℓ an r-class of Q, an equivalence class modulo Q_r an ℓ-class of Q and an equivalence class modulo Q_c a c-class of Q.

Note the peculiar switch: an equivalence class modulo Q_ℓ is an r-class of Q. The reason for this will become evident later.

We shall also write $\ell(x)$ for the ℓ-class containing x, $r(x)$ for the r-class containing x, and $c(x)$ for the c-class containing x.

Definition The machine of a regular language Q is the unique deterministic recogniser of Q having the least number of nodes.

The anti-machine of Q is the machine of $\overleftarrow{Q}$, where $\overleftarrow{Q}$ denotes the set of all words which are the reverse of words in Q.

Nodes of the machine and anti-machine will usually be called states.

The semigroup of Q is the quotient of the free semigroup V^* with respect to the congruence relation Q_c.

The machine and the ℓ-classes of Q, and the anti-machine and the r-classes of Q are connected by the following theorem.

Theorem 2.2 Let Q be a regular language. Let the states of the machine for Q be $\{\ell_1, \ldots, \ell_n\}$ and the states of the anti-machine be $\{r_1, \ldots, r_m\}$. Suppose that ℓ_1 and r_1 are the start states of the respective machines and let $x \in V^*$. Then we have:

(a) If x takes the start state ℓ_1 to state ℓ_i of the machine, then the ℓ-class containing x, $\ell(x)$, is the set of all words which also take state ℓ_1 to state ℓ_i.

(b) If $\overleftarrow{x}$ takes the start state r_1 to state r_j of the anti-machine, then the r-class containing x, $r(x)$, is the set of the reverse of all words which take state r_1 to state r_j.

Corresponding to the semigroup we can always construct a semigroup machine, whose states correspond to elements c_i of the semigroup, and where, for all $a \varepsilon V$, there is an arc labelled a from state c_i to c_j if $c_i \cdot c(a) = c_j$. Let c_1 be the identity element of the semigroup. We then have:

(c) If x takes the state c_1 to state c_t of the semigroup machine, then the c-class containing x, c(x), is the set of all words which also take state c_1 to state c_t.

Corollary The ℓ, r and c-classes of Q are regular if Q is regular.

Because of this theorem, we shall henceforth use the symbols ℓ_1, ℓ_2, ... to denote states of a machine for a regular language Q and also to denote the ℓ-classes of Q to which they correspond. (And similarly of course with the symbols r_1, r_2, ... and c_1, c_2, ...).

(2.2) Derivatives, Anti-derivatives and Contexts

Let us consider the relation Q_r. We note that any word $x \varepsilon V^*$ partitions V^* into two sets, denoted $D_x Q$ and $\sim D_x Q$, where

$$D_x Q = \{y \mid xy \varepsilon Q\}$$

$$\sim D_x Q = \{y \mid xy \notin Q\} .$$

$D_x Q$ is called the derivative of Q with respect to x.

We then have:

Lemma 2.3 $x \; Q_r \; y \Leftrightarrow D_x Q = D_y Q$.

This is the basis of the method of derivatives for calculating the machine of a language Q [3].

Similarly the relation Q_ℓ leads one to define anti-derivatives: The anti-derivative of Q with respect to x, denoted $\partial_x Q$ is

$$\partial_x Q = \{y \mid xy \in \overleftarrow{Q}\} = \{y \mid \overleftarrow{yx} \in Q\} .$$

Lemma 2.4 $\quad x\ Q_\ell\ y \Leftrightarrow \partial_{\overleftarrow{x}} Q = \partial_{\overleftarrow{y}} Q .$

Finally, the relation Q_c partitions the set $V^* \times V^*$ into $C_x Q$, the context of x in Q, and $\sim C_x Q$ where

$$C_x Q = \{(u,v) \mid uxv \in Q\} .$$

Lemma 2.5 $\quad x\ Q_c\ y \Leftrightarrow C_x Q = C_y Q .$

The following observation, although rather elementary, is quite important in the sequel.

Theorem 2.6 (a) The word derivatives $D_x Q$ of a language Q are unions of r-classes of Q, where $D_x Q \supseteq r(y)$ if and only if $xy \in Q$.

(b) The reverse of anti-derivatives of Q, i.e. languages of the form $\overleftarrow{\partial_{\overleftarrow{y}} Q}$, are unions of ℓ-classes of Q, where $\overleftarrow{\partial_{\overleftarrow{y}} Q} \supseteq \ell(x)$ if and only if $xy \in Q$.

(c) The contexts $C_x Q$ of a language Q are unions of subsets of $V^* \times V^*$ of the form $\ell \times r$, where ℓ is an ℓ-class of Q and r is an r-class of Q, where $C_x Q \supseteq \ell(u) \times r(v)$ if and only if $uxv \in Q$.

Proof Let Q be a language and let $x \in V^*$.

Then $y \in D_x Q \Leftrightarrow xy \in Q \Leftrightarrow \overleftarrow{x} \in \partial_{\overleftarrow{y}} Q$.

But by lemma 2.4, $\partial_{\overleftarrow{y}} Q = \partial_{\overleftarrow{y'}} Q$ for all y' such that $y'\ Q_\ell\ y$.

$\therefore\ y \in D_x Q \Leftrightarrow y' \in D_x Q$ for all y' such that $y'\ Q_\ell\ y$.

i.e. $D_x Q = \sum_{y \in D_x Q} r(y)$, and part (a) is proved.

Part (b) is proved similarly.

Consider now $C_x Q$. The pair $(u,v) \in C_x Q \Leftrightarrow uxv \in Q$ $\Leftrightarrow v \in D_{ux} Q$ and $\overleftarrow{u} \in C\left|\overleftarrow{xv}\right. Q$.

But then, by an identical argument to that above, this implies that $u'xv' \in Q$ for all $u' \in \ell(u)$ and $v' \in r(v)$.

i.e. $C_x Q \supseteq \ell(u) \times r(v)$.

Whence $C_x Q = \sum_{(u,v) \in C_x Q} \ell(u) \times r(v)$, and we have proved (c).

Note that although the displayed unions are over an infinite set, the number of distinct terms is finite when Q is regular, and so the unions themselves may be taken over only a finite set of words.

3. The Fundamentals of Factor Theory

The following definitions are taken from Conway [13].

Definitions Let F, G, H, ... , K, Q denote arbitrary languages (not necessarily regular).

F.G...H...J.K is a subfactorization of Q if and only if

$$F.G...H...J.K \subseteq Q. \qquad (*)$$

$\overline{F}.\overline{G}...\overline{H}...\overline{J}.\overline{K}$ dominates it if it is also a subfactorization of Q and $F \subseteq \overline{F}$, $G \subseteq \overline{G}$, ... , $K \subseteq \overline{K}$.

A term H is maximal if it cannot be increased without violating the inequality (*).

A factorization of Q is a subfactorization in which every term is maximal.

A factor of Q is any language which is a term in some factorization of Q.

A left (right) factor is one which can be the leftmost (rightmost) term in a factorization of Q.

Next we state two lemmas, due to Conway, which are quite fundamental to future results. The proofs are quite simple and can be found in Conway's book [13].

Lemma 3.1 Any subfactorization of Q is dominated by some factorization in which all terms originally maximal remain unchanged.

Lemma 3.2 Any left factor is the left factor in some 2-term factorization. Any right factor is the right factor in some 2-term factorization. Any factor is the central term in some 3-term factorization. The condition that L.R be a factorization of Q defines a (1-1) correspondence between left and right factors of Q.

We shall now give a characterisation of the factors of Q which gives some insight into their properties. Recall (§2) that an $\mathcal{L}$-class of Q is a right-invariant equivalence class, an r-class is a left-invariant equivalence class and a c-class is a congruence class of Q.

Theorem 3.3 The left factors of any language Q are either ϕ (the empty set) or are sums of $\mathcal{L}$-classes of Q. The right factors of Q are either ϕ or are sums of r-classes of Q and the factors are ϕ or are sums of c-classes of Q.

Corollary (Conway) A language Q is regular if and only if it has a finite number of factors. The factors are regular for regular Q.

Proof Let L be a left factor in the two term factorization $L.R \subseteq Q$ of Q. If $L \neq \phi$, let $x \in L$ and consider any $y \in \mathcal{L}(x)$. Since $L.R \subseteq Q$, $R \subseteq D_x Q = D_y Q$ (by Lemma 2.3). Therefore $y.R \subseteq Q$, and so, since L is maximal, $y \in L$. Hence $L \supseteq \mathcal{L}(x)$, and $L = \sum_{x \in L} \mathcal{L}(x)$, i.e. L is a sum of $\mathcal{L}$-classes

of Q. Similarly any non-empty right factor is a sum of r-classes of Q.

If H is any factor of Q it is the central term in a factorization $LHR \subseteq Q$ (lemma 3.2). If $H \neq \phi$, let $x \in H$. Then the set $C_x Q = \{(u,v) | uxv \in Q\} \supseteq L \times R = \{(u,v) | u \in L, v \in R\}$. But if $y \in c(x)$, $C_y Q = C_x Q \supseteq L \times R$. Thus, as above, $y \in H$ and $H = \sum_{x \in H} c(x)$.

The corollary follows from the corollary to Theorem 2.2.

The above characterisation of the factors of Q is different to Conway's. The advantage will be seen later when we consider the problem of calculating the factors of Q.

From now on, unless otherwise stated, we shall only consider the case when Q is regular.

4. The Factor Matrix

Following Conway, let us index the left and right factors as $L_1, L_2, \ldots, L_q$ and $R_1, R_2, \ldots, R_q$ wherein corresponding factors (see lemma 3.2) are given the same index. We now define Q_{ij} $(1 \leq i,j \leq q)$ by the condition that $L_i Q_{ij} R_j$ is a subfactorization of Q in which Q_{ij} is maximal. (It is important to note that $L_i Q_{ij} R_j$ may not be a factorization of Q). We note that, by lemmas 3.1 and 3.2, H is a factor of Q if and only if it is some Q_{ij}. Thus the factors of Q are organised into a $q \times q$ matrix which is called the factor matrix of Q and is denoted $\boxed{Q}$.

Various properties of the factor matrix may be observed, some of which are summarised below.

Theorem 4.1

(i) H is a factor of $Q \Leftrightarrow$ H is some entry Q_{ij} in the factor matrix $\boxed{Q}$.

(ii) Q_{ij} is maximal in the subfactorizations $L_i \cdot Q_{ij} \subseteq L_j$ and $Q_{ij} \cdot R_j \subseteq R_i$. Thus Q_{ij} is a right factor of L_j and a left factor of R_i.

(iii) $\exists$ unique indices s and t such that $Q = L_t = R_s = Q_{st}$, $L_i = Q_{si}$ and $R_i = Q_{it}$.

(iv) $\boxed{Q} = \boxed{Q}^*$.

(v) If $A_1 \cdot A_2 \ldots A_m \subseteq Q_{ij}$ is a subfactorization of Q_{ij}, then $\exists$ indices $k_1, k_2, \ldots, k_{m-1}$ such that $A_1 \subseteq Q_{ik_1}$, $A_2 \subseteq Q_{k_1k_2}$, $\ldots$, $A_m \subseteq Q_{k_{m-1}j}$.

Proof Although the proofs of all parts of this theorem can be found in Conway's book, the proof technique is so fundamental that it is worth repeating.

The proof of (i) is contained in the preamble to the theorem.

To prove (ii), we observe that the subfactorization $(L_iQ_{ij}) \cdot R_j \subseteq Q$ is dominated by $L_j \cdot R_j \subseteq Q$. Therefore $L_i \cdot Q_{ij} \subseteq L_j$ is a subfactorization of L_j in which Q_{ij} must be maximal. Similarly $Q_{ij} \cdot R_j \subseteq R_i$ is a subfactorization of R_i in which Q_{ij} is maximal. The rest follows from lemmas 3.1 and 3.2.

The indices s and t in part (iii) are chosen by the condition that $L_t \cdot R_t \subseteq Q$ dominates the subfactorization $Q \cdot e \subseteq Q$, and $L_s \cdot R_s \subseteq Q$ dominates the subfactorization $e \cdot Q \subseteq Q$. Then, by definition, $L_t \supseteq Q$; but also $Q \supseteq L_t \cdot R_t \supseteq L_t \cdot e = L_t$. Therefore $L_t = Q$. Similarly

$R_s = Q$. The index s (t) is then unique, because all the left (right) factors are distinct. To prove that $Q_{st} = Q$, we note that Q_{st} is maximal in $L_s \cdot Q_{st} \cdot R_t \subseteq Q$ implies that it is also maximal in the subfactorization $L_s \cdot Q_{st} \subseteq L_t$. But $L_t = Q$, and R_s is maximal in $L_s \cdot R_s \subseteq Q$. Therefore $Q_{st} = R_s = Q$. Now $L_s \cdot Q \subseteq Q$ and $L_i \cdot R_i \subseteq Q$ are both factorizations of Q; so $L_s \cdot L_i \cdot R_i \subseteq Q$ is a subfactorization of Q in which L_i and R_i are maximal. Therefore, by definition of Q_{si}, $L_i = Q_{si}$. Similarly, $R_i = Q_{it}$.

Part (iv) can now be proved quite simply. We observe

(a) $Q_{ii} \supseteq e$ (Since $L_i \cdot R_i = L_i \cdot e \cdot R_i \subseteq Q$). and

(b) $Q_{ij} \supseteq Q_{ik} \cdot Q_{kj}$, for all $k = 1, 2, \ldots, q$. This follows because, by (ii), $L_k \supseteq L_i Q_{ik}$, and $R_k \supseteq Q_{kj} \cdot R_j$. Therefore $L_i \cdot (Q_{ik} Q_{kj}) \cdot R_j \subseteq L_k \cdot R_k \subseteq Q$. So, by definition, $Q_{ij} \supseteq Q_{ik} \cdot Q_{kj}$.

In matrix terms (a) and (b) are

(a)' $\overline{|Q|} \supseteq E$, (b)' $\overline{|Q|} \supseteq \overline{|Q|} \cdot \overline{|Q|}$.

Therefore $\overline{|Q|} \supseteq E + \overline{|Q|} \cdot \overline{|Q|}$, and so by R1, $\overline{|Q|} \supseteq \overline{|Q|}^*$.

But $\overline{|Q|}^* \supseteq \overline{|Q|}$. Therefore $\overline{|Q|} = \overline{|Q|}^*$.

We shall prove (v) for the case $m = 2$; for $m > 2$ the result follows by simple induction. Suppose then that $A \cdot B \subseteq Q_{ij}$. Then $(L_i A) \cdot (B R_j) \subseteq Q$ is a subfactorization of Q and so must be dominated by some factorization $L_k \cdot R_k \subseteq Q$. I.e. $L_k \supseteq L_i \cdot A$ and $R_k \supseteq B \cdot R_j$. But then, by (ii), $A \subseteq Q_{ik}$ and $B \subseteq Q_{kj}$.

4.1 is an extremely interesting and powerful theorem, from which most results on factors can be deduced immediately. Particularly useful is 4.1 (iv), which we shall often apply in its alternative form (a) $Q_{ii} \supseteq e$ and (b) $Q_{ij} = \sum_k Q_{ik} \cdot Q_{kj}$.

Part (iii) tells us that the s th column of $\overline{|Q|}$ contains all the left factors and the t th row all the right factors, and the intersection of this row and column is the language Q itself. This and (iv), $\overline{|Q|} = \overline{|Q|}^*$, suggest very strongly that there is some recogniser of Q, $(G,\{s\},\{t\})$, consisting of a graph G with start node s and terminal node t, such that L_i is the set of all words taking node s to node i, and R_j is the set of all words taking node j to node t. In fact there is often more than one such G, but we shall show that there is a unique minimal one.

Note, also, that (iii) does not imply that Q only occurs once in its factor matrix. For instance,

$Q = (11)^*$ has factor matrix $\overline{|Q|} = \begin{bmatrix} (11)^* & (11)^*1 \\ (11)^*1 & (11)^* \end{bmatrix}$

in which Q occurs twice. (We mention this because there is a misprint in Conway's book [13,p 49], in which Conway says "The theorem does prevent Q from occurring twice in its matrix ...". This should, of course, read "does not prevent").

As we shall see, the combination of Theorems 3.3 and 4.1 (ii) is sufficient to enable one to calculate $\overline{|Q|}$. Various "brute force" methods can be used, but the method we give appears to be the most straightforward and easiest to apply.

5. The Factor Graph

Our objective in this section is to find a method of determining the factor matrix of a regular language Q. We shall prove that there is a unique minimal matrix G_Q such

that $\lceil Q \rceil = G_Q^* . G_Q$ is a constant + linear matrix and so is called the factor graph of Q.

The proof technique we use may seem rather roundabout, and so it is worth while explaining the difficulty. Consider any element A of a regular algebra R. We shall call any X such that $X^* = A^*$ a starth root of A^*. Our aim is to prove that there is a unique minimal starth root of $\lceil Q \rceil$. In a free regular algebra it is quite easy to prove that there is always a unique minimal starth root of any element α^* in the algebra (see Brzozowski [4]). The proof, however, relies on length considerations and does not apply to all regular algebras. Indeed it is not generally true. Consider the matrix $M = \begin{bmatrix} e & e & e \\ e & e & e \\ e & e & e \end{bmatrix}$

This matrix has starth roots $A_1 = \begin{bmatrix} \phi & e & \phi \\ \phi & \phi & e \\ e & \phi & \phi \end{bmatrix}$ and

$A_2 = \begin{bmatrix} \phi & \phi & e \\ e & \phi & \phi \\ \phi & e & \phi \end{bmatrix}$, which are both minimal.

Thus there is no unique minimal starth root of M.

In order to prove that $\lceil Q \rceil$ nevertheless does have a unique minimal starth root we first prove that $\lceil Q \rceil$ has a starth root which is a constant + linear matrix, and then that this matrix can be reduced to one which is minimal.

Lemma 5.1 $\exists$ unique maximal constant and linear matrices C_{max} and L_{max} such that $\lceil Q \rceil \geq (C_{max} + L_{max})^*$.

Proof Define C_{max} and L_{max} to be the unique maximal constant and linear matrices (respectively) such that

$\lceil Q \rceil \geq C_{max}$ and $\lceil Q \rceil \geq L_{max}$. Then $\lceil Q \rceil \geq C_{max} + L_{max}$ and, as $\lceil Q \rceil = \lceil Q \rceil^*$, $\lceil Q \rceil \geq (C_{max} + L_{max})^*$.

We shall now prove that the inequality in the above lemma can be changed to an equality.

Theorem 5.2 (Conway) Let C_{max} and L_{max} be the unique maximal constant and linear matrices of lemma 5.1 such that $\lceil Q \rceil \geq (C_{max} + L_{max})^*$. Then $\lceil Q \rceil = (C_{max} + L_{max})^*$.

Proof Suppose $x \in Q_{ij}$. If $x = e$ then (a) $x \in [C_{max}]_{ij}$, since C_{max} is maximal. Otherwise (b) $x = a_1 a_2 \ldots a_m$ is a word of length $m \geq 1$, in which each a_p is a letter. But then applying Theorem 4.1(v), $\exists$ integers $k_1, k_2, \ldots, k_{m-1}$ such that $a_1 \in Q_{ik_1}$, $a_2 \in Q_{k_1k_2}$, $\ldots$, $a_m \in Q_{k_{m-1}j}$. But then $a_1 \in [L_{max}]_{ik_1}$, $a_2 \in [L_{max}]_{k_1k_2}$, $\ldots$, $a_m \in [L_{max}]_{k_{m-1}j}$. I.e. $x \in [L_{max}.L^*_{max}]_{ij}$. But (a) and (b) imply $\lceil Q \rceil \leq C_{max} + L_{max}.L^*_{max}$. But, using lemma 5.1, $\lceil Q \rceil \leq C_{max} + L_{max}.L^*_{max} \leq (C_{max} + L_{max})^* \leq \lceil Q \rceil$. Hence the theorem.

We have already mentioned that in a free regular algebra R_F any event A^* has a unique minimal starth root. This is given by $(A\backslash E)\backslash(A\backslash E)^{2+*}$, where E is the unit element of R_F, $\backslash$ denotes set difference and x^{2+*} denotes $x^2+x^3+x^4+\ldots$. In the algebra $M_p(R_F)$, of $p \times p$ matrices over the free regular algebra R_F, the most we can say is that if $(A\backslash E)\backslash(A\backslash E)^{2+*}$ is a starth root then A^* does have a unique minimal starth root which is given by the above expression. More formally:

Theorem 5.3 Let A be an element of $M_p(R_F)$ where R_F is a free regular algebra. Let $M_p(R_F)$ have unit element E.

Let $[B\backslash C]_{ij} = [b_{ij}\backslash c_{ij}]$ where $\backslash$ denotes set difference.
If $A^* = \{(A\backslash E)\backslash(A\backslash E)^{2+*}\}^*$ then $(A\backslash E)\backslash(A\backslash E)^{2+*}$ is the unique minimal starth root of A^*.

Proof Let $X = (A\backslash E)\backslash(A\backslash E)^{2+*}$. By assumption X is a starth root of A^*. Suppose Y is also a starth root. We must show that $X \subseteq Y$.

Suppose $w \in X_{ij}$.

Clearly $w \in Y^*_{ij} = [(Y\backslash E)^*]_{ij}$, because Y is a starth root of A^* and $A^* \supseteq X$. Hence $w \in [(Y\backslash E)^n]_{ij}$ for some n where, by definition of X, $n \geq 1$.

Now $Y \subseteq A^* = (A\backslash E)^*$.

Hence $Y\backslash E \subseteq (A\backslash E)^+$.

$\therefore$ $w \in [(Y\backslash E)^n]_{ij} \subseteq [((A\backslash E)^+)^n]_{ij}$.

But $w \in X_{ij} = [(A\backslash E)\backslash(A\backslash E)^{2+*}]_{ij}$

$\Rightarrow$ $n = 1$

$\Rightarrow$ $w \in [Y\backslash E]_{ij} \subseteq Y_{ij}$.

I.e. $X \subseteq Y$ and the theorem is proved.

Considering the matrix M mentioned at the beginning of this section, we find that

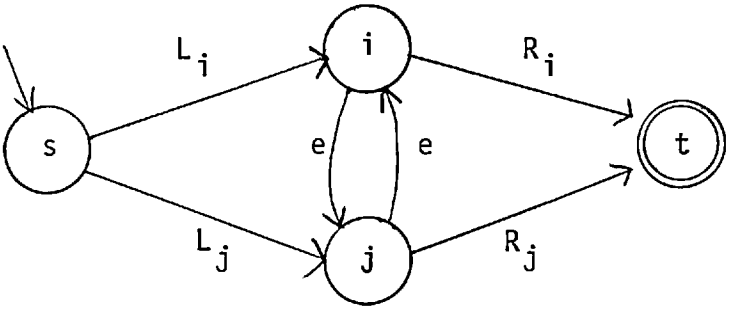
$$(M\backslash E)\backslash(M\backslash E)^{2+*} = \begin{bmatrix} \phi & \phi & \phi \\ \phi & \phi & \phi \\ \phi & \phi & \phi \end{bmatrix}$$

This is clearly not a starth root of M.

We note however that M has e-cycles which pass through more than one node. This cannot be true of the factor matrix as the next lemma states. This observation together with theorems 5.2 and 5.3 enable us to proceed to the proof of our main theorem.

Lemma 5.4 $C_{max}\backslash E$ is acyclic.

Proof Suppose $C_{max}\backslash E$ is cyclic. Then there must be two distinct nodes i and j such that $[C_{max}]_{ij} = e = [C_{max}]_{ji}$. Now consider the matrix $\overline{|Q|}$. It will help if the reader refers to the figure below, which shows part of the matrix $\overline{|Q|}$. The nodes labelled s and t are the nodes mentioned in Theorem 4.1.



We have indicated by labelled arrows that

$$Q_{si} = L_i, \quad Q_{it} = R_i$$

$$Q_{sj} = L_j \text{ and } Q_{jt} = R_j .$$

Since $\overline{|Q|} \geq C_{max}$, $Q_{ij} \geq e$ and $Q_{ji} \geq e$.

This has also been indicated. We shall now prove that $L_i = L_j$ and $R_i = R_j$.

We have $\overline{|Q|} = \overline{|Q|}^*$, (4.1(iv)), and $\overline{|Q|} \geq C_{max}$, by Theorem 5.2.

Therefore $\overline{|Q|} = \overline{|Q|} \cdot \overline{|Q|} \geq \overline{|Q|} \cdot C_{max}$.

Hence $Q_{si} \geq Q_{sj} \cdot [C_{max}]_{ji} = Q_{sj}$

and $Q_{sj} \geq Q_{si} \cdot [C_{max}]_{ij} = Q_{si}$.

Therefore $Q_{si} = Q_{sj}$ i.e. $L_i = L_j$.

Similarly, using $\overline{|Q|} \geq C_{max} \cdot \overline{|Q|}$, we get $R_i = R_j$.

But then nodes i and j cannot be distinct and we have a contradiction. Therefore the initial assumption is incorrect and $C_{max}\backslash E$ must be acyclic.

Corollary 5.5 Let $\lceil Q \rceil$ be a $q \times q$ matrix. Then $(C_{max} \backslash E)^p = N$ for all $p \geq q$.

Finally we come to the main theorem of this chapter.

Theorem 5.6 Let Q be a regular language, and let C_{max} and L_{max} be as defined in Theorem 5.2. Then there is a unique minimal matrix G_Q such that $G_Q^* = \lceil Q \rceil$, given by $G_Q = ((C_{max} + L_{max}) \backslash E) \backslash ((C_{max} + L_{max}) \backslash E)^{2+*}$. Moreover the triple $(G_Q, \{s\}, \{t\})$ (where s and t are given by Theorem 4.1 (iii)) is a recogniser for Q.

G_Q is a constant + linear matrix and so its graph will be called the factor graph of Q.

Proof Using Theorem 5.3, we need only prove that $G_Q^* = \lceil Q \rceil$ where $G_Q = ((C_{max} + L_{max}) \backslash E) \backslash ((C_{max} + L_{max}) \backslash E)^{2+*}$. In turn this only requires proving that $G_Q^* \supseteq (C_{max} + L_{max}) \backslash E$, since then $G_Q^* = (G_Q^*)^* \supseteq (C_{max} + L_{max})^* = \lceil Q \rceil$, by Theorem 5.2.

Let p1 : $w \in [(C_{max} + L_{max}) \backslash E]_{ij}$.

Then p2 : w has length 0 or 1.

Suppose p3 : $w \notin [G_Q^*]_{ij}$.

Then p4 : $w \notin [G_Q]_{ij}$

and so p5 : $w \in [((C_{max} + L_{max}) \backslash E)^{2+*}]_{ij}$.

Hence $\exists$ indices $i = k_1, k_2, \ldots, k_{m+1} = j$ and words $a_1, a_2, \ldots, a_m$ s.t. $m \geq 2$, $w = a_1 a_2 \ldots a_m$

and p1 : $a_h \in [(C_{max} + L_{max}) \backslash E]_{k_h k_{h+1}}$

and hence p2 : a_h has length 0 or 1, for all $h = 1, \ldots, m$.

Now for some h we must have

p3 : $a_h \notin [G_Q^*]_{k_h k_{h+1}}$

(otherwise $w \in [G_Q^*]_{ij}$).

Hence for this h

$$p4 : a_h \notin [G_Q]_{k_h k_{h+1}}$$

and so, for this h

$$p5 : a_h \in [((C_{max} + L_{max})\backslash E)^{2+*}]_{k_h k_{h+1}} .$$

Let this a_h be v. v has the same properties as w and so can in turn be expressed as the product of two or more words $b_1 b_2 \ldots b_n$, where by the same argument one of the b_g's = u, say, also has the same properties as w. In this way we can express w as a product

$$w = y_1 y_2 \ldots y_x$$

of an unbounded number x of words y_f, where

$$y_f \in [(C_{max} + L_{max})\backslash E]_{n_f n_{f+1}}$$

for some nodes $n_1, \ldots, n_{x+1}$. But the product of two linear matrices is either null or non-linear. Therefore at most one y_f has length one, and we conclude that $(C_{max}\backslash E)^p$ is non-null for all p. But this contradicts corollary 5.5. Hence the initial assumption that property p3 holds for w must be false. Hence $G_Q^* \geq (C_{max} + L_{max})\backslash E$, and by our earlier argument $G_Q^* = \lceil Q \rceil$. Finally, applying Theorem 5.3, G_Q is the minimal starth root of $\lceil Q \rceil$. The last part of the theorem follows immediately from Theorem 4.1 (iii), $Q = Q_{st} = [G_Q^*]_{st}$.

6. AN EXAMPLE

The previous results embody an algorithm for calculating $\lceil Q \rceil$, which we shall develop with the aid of an example. Essentially our aim is to calculate $C_{max} + L_{max}$; we could then use any one of the algorithms of chapter II to calculate

$\lceil Q \rceil$ from G_Q, using $\lceil Q \rceil = G_Q^*$ and $G_Q = ((C_{max} + L_{max})\backslash E)\backslash((C_{max} + L_{max})\backslash E)^{2+*}$, although the next chapter will develop a rather better algorithm to determine G_Q^*.

Example 1: Let $Q = (b+a(aa^*b)^*b)^*$

6.1 Machine, Anti-machine and Semigroup

Theorem 3.3 suggests that we begin by calculating the machine, anti-machine and semigroup of Q. This we have done, using standard methods, in Figures 1(a), (b) and (c).

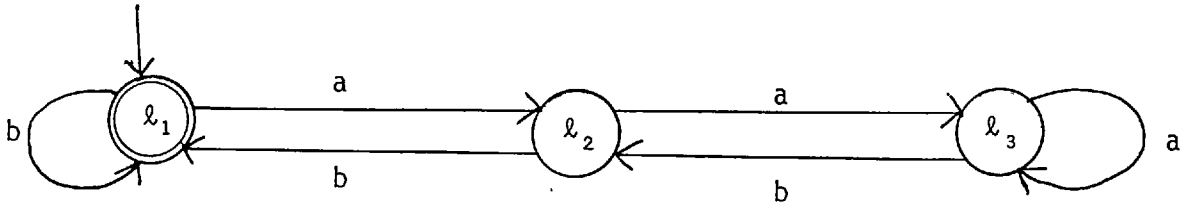


Fig. 1(a) Machine of Q.

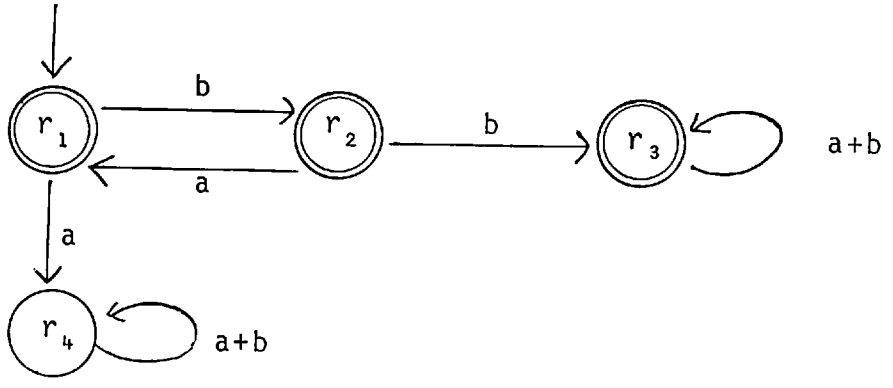


Fig. 1(b) Anti-machine of Q.

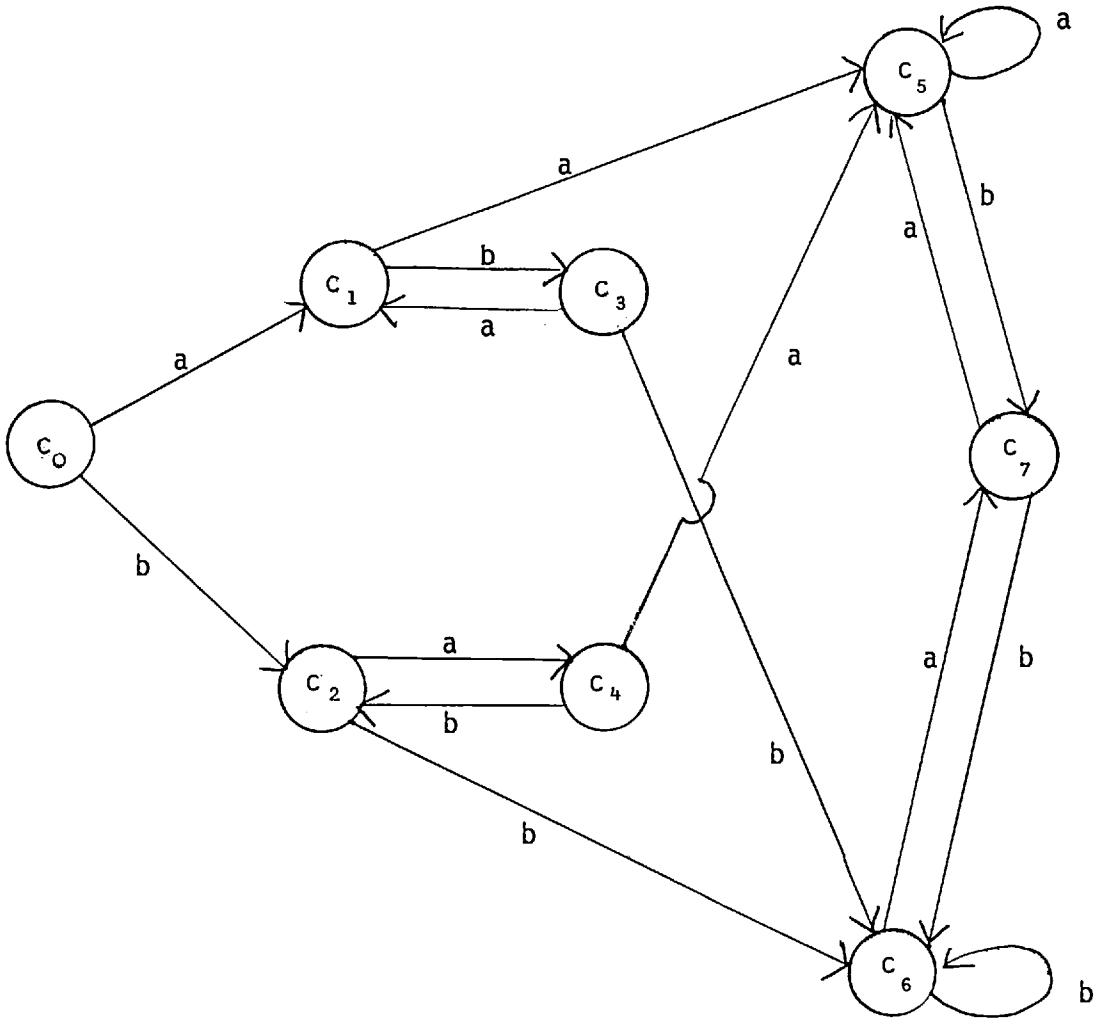


Fig. 1(c) Semigroup of Q.

In these figures we have labelled the nodes of the machine ℓ_1, ℓ_2 and ℓ_3, but, as stated after the corollary to theorem 2.2, we shall also use the symbols ℓ_1, ℓ_2, ℓ_3 to denote the sets $(b+a(aa^*b)^*b)^*$ $(=Q)$, $Qa(aa^*b)^*$ and $Qa(aa^*b)^*aa^*$, respectively, these being the right-invariant equivalence classes to which they correspond. Similarly r_1 is used to label a node of the anti-machine, but also denotes the set $(ab)^*$, this being the set of the reverse of all words

which take node r_1 to itself in the anti-machine (=(ba)*) .

6.2 Calculating the Derivatives etc.

In tables 1(a) and (b) we have used Theorem 2.6 to determine the derivatives and anti-derivatives of Q as sums of r-classes and sums of ℓ-classes of Q, respectively. Thus in the first column of each table we have listed the ℓ- and r-classes, and the third column shows the corresponding derivative or (reverse of) anti-derivative as a union of r-classes or ℓ-classes of Q, as the case may be.

Node/ ℓ-class	Representative element	Derivative
ℓ_1	e	$r_1+r_2+r_3$
ℓ_2	a	r_2+r_3
ℓ_3	aa	r_3

Table 1(a) Machine.

Node/ r-class	Representative element	Reverse of anti-derivative
r_1	e	ℓ_1
r_2	b	$\ell_1 + \ell_2$
r_3	bb	$\ell_1 + \ell_2 + \ell_3$
r_4	a	ϕ

Table 1(b) Anti-machine.

A very important point to note here is that, since each ℓ-, r- or c-class consists of words which are equivalent with respect to some relation, the properties which follow from the equivalences can be found by considering representative elements of the equivalence classes only. In order to calculate the derivatives of Q as sums of r-classes we have chosen in column 2 of each table a representative element of each equivalence class. The choice is quite arbitrary. Then we have used Theorem 2.6 (a) directly to determine the derivative. For instance the class ℓ_2 has representative a

and since $a \cdot e \notin Q$ (e is the representative of r_1),

and $a \cdot a \notin Q$ (a ——— " ——— r_4),

but $a \cdot b \in Q$ (b ——— " ——— r_2),

and $a \cdot bb \in Q$ (bb ——— " ——— r_3),

the derivative corresponding to ℓ_2 is $r_2 + r_3$. Thus it is quite unnecessary in these calculations to calculate regular expressions representing the various ℓ-, r- and c-classes.

A similar table, illustrating part (c) of Theorem 2.6, could be constructed for the c-classes of Q. The part of this table for those c-classes containing e or a word consisting of a single letter is shown in Table 1(c). Once again column 1 lists the c-class and column 2 gives a representative element of each c-class. The third column expresses the corresponding context of Q, C_xQ (where x is the representative), as a union of direct products of the form $\ell_i \times r_j$. A simple way of calculating the appropriate entry is as follows. Suppose one is considering the c-class c_k having as representative the element x. Consider each state ℓ_i of

the machine in turn. If under input x state ℓ_i goes to state ℓ_j, read off the entry in column 3 of the ℓ_jth row of Table 1(a). Suppose this is $r_{j_1} + r_{j_2} + \ldots + r_{j_h}$. Then add $\ell_i \times (r_{j_1} + \ldots + r_{j_h})$ to the entry already in the third column of c_k.

For example consider the class c_1 having representative a. Let us write $\ell_i \xrightarrow{x} \ell_j$ if input x takes state ℓ_i to state ℓ_j of the machine. Then we have

$\ell_1 \xrightarrow{a} \ell_2$ and ℓ_2 has derivative $r_2 + r_3$. Hence enter $\ell_1 \times (r_2 + r_3)$

$\ell_2 \xrightarrow{a} \ell_3$ and ℓ_3 ——— " ——— r_3. —— " —— $\ell_2 \times r_3$

$\ell_3 \xrightarrow{a} \ell_3$ and ℓ_3 ——— " ——— r_3. —— " —— $\ell_3 \times r_3$.

Thus the entry in column 3 for c_1 is

$$\ell_1 \times (r_2 + r_3) + \ell_2 \times r_3 + \ell_3 \times r_3 .$$

The reader should ignore column 4 of Table 1(c) for the time being.

Node/ Congruence class	Representative element	Context Form 1.	Context Form 2.
c_0	e	$\ell_1 \times (r_1 + r_2 + r_3)$	$L_1 \times R_1 + L_2 \times R_2 + L_3 \times R_3$
		$+ \ell_2 \times (r_2 + r_3)$	$+ L_1 \times R_2 + L_2 \times R_3 + L_1 \times R_3$
		$+ \ell_3 \times r_3$	$+ L_4 \times R_i \quad (i = 1,2,3,4)$
c_1	a	$\ell_1 \times (r_2 + r_3)$	$L_3 \times R_3 + L_1 \times R_2$
		$+ \ell_2 \times r_3$	$+ L_2 \times R_3 + L_1 \times R_3$
		$+ \ell_3 \times r_3$	$+ L_4 \times R_i \quad (i = 1,2,3,4)$
c_2	b	$\ell_1 \times (r_1 + r_2 + r_3)$	$L_3 \times R_2 + L_2 \times R_1$
		$+ \ell_2 \times (r_1 + r_2 + r_3)$	$+ L_2 \times R_2 + L_3 \times R_2$
		$+ \ell_3 \times (r_2 + r_3)$	$+ L_1 \times R_2 + L_2 \times R_3 + L_1 \times R_3$

Table 1(c)

6.3 The Left and Right Factors

We can now deduce the left factors and right factors of Q from Table 1(a). If we consider any sum $\ell_{i_1} + \ell_{i_2} + \ldots + \ell_{i_k}$ of ℓ-classes of Q, including the empty sum ϕ, and then determine those r-classes $r_{j_1} + \ldots + r_{j_m}$ common to the third column of all the i_1th, i_2th, ... , i_kth rows of Table 1(a), then

$$(\ell_{i_1} + \ell_{i_2} + \ldots + \ell_{i_k}) \cdot (r_{j_1} + \ldots + r_{j_m}) \subseteq Q$$

will be a subfactorization of Q. By inspection of all such subfactorizations we can deduce those which are also factorizations of Q. Thus for our example we would get the following subfactorizations:

$(\ell_1 + \ell_2 + \ell_3) \cdot r_3$, $(\ell_2 + \ell_3) \cdot r_3$, $(\ell_1 + \ell_3) \cdot r_3$, $\ell_3 \cdot r_3$,

$(\ell_1 + \ell_2) \cdot (r_2 + r_3)$, $\ell_2 \cdot (r_2 + r_3)$,

$\ell_1 \cdot (r_1 + r_2 + r_3)$,

$\phi \cdot (r_1 + r_2 + r_3 + r_4)$.

Table 2 Subfactorizations of Q.

We have displayed these subfactorizations in such a way as to make it evident that only those in the first column are also factorizations of Q.

This information is summarised in Table 3, in which we have also named the left and right factors L_1, L_2, L_3, L_4 and R_1, R_2, R_3, R_4.

Left factors		Right factors	
$L_t = L_1$	ℓ_1	$R_s = R_1$	$r_1 + r_2 + r_3$
L_2	$\ell_1 + \ell_2$	R_2	$r_2 + r_3$
L_3	$\ell_1 + \ell_2 + \ell_3$	R_3	r_3
L_4	ϕ	R_4	$r_1 + r_2 + r_3 + r_4$

Table 3 Left and right factors

In the table we have also indicated that the indices s and t of Theorem 4.1(iii) are both equal to 1. This is because, from the machine (Fig. 1(a)), $Q = \ell_1$, and from the anti-machine (Fig. 1(b)), $Q = r_1 + r_2 + r_3$; but $L_1 = \ell_1$ and $R_1 = r_1 + r_2 + r_3$.

6.4 Construction of $C_{max} + L_{max}$ and G_Q

The penultimate step in the construction of the factor graph is to construct $C_{max} + L_{max}$. In our example the graph of $C_{max} + L_{max}$ will have four nodes (see Fig. 2). In order to fill in the arc labels there are two approaches we can adopt.

(i) In Table 1(c), column 4, we have expressed the context $C_x Q$ of each constant or linear term x (i.e. $x = e$ or $x \in V$) in all possible ways in terms of direct products of left and right factors of Q. There is then an arc labelled x from node i to node j of $C_{max} + L_{max}$ if and only if there is an entry $L_i \times R_j$ in Column 4 of Table 1(c) of the row corresponding to x.

(ii) Alternatively, we can apply Theorem 4.1(ii), which can be restated as, $x \in Q_{ij} \Leftrightarrow L_i x \subseteq L_j$. To do this we use the representation of L_i as $\ell_{i_1} + \ell_{i_2} + \ldots + \ell_{i_k}$, given in Table 3. C_{max} can be calculated immediately using $[C_{max}]_{ij} = e \Leftrightarrow L_i \subseteq L_j$. To find L_{max} consider each element $x \in V$ in turn. Under input x the state ℓ_{i_m} of the machine goes to state ℓ_{h_m}, say. Thus $L_i x = (\ell_{i_1} + \ell_{i_2} + \ldots + \ell_{i_k}) \cdot x \subseteq \ell_{h_1} + \ell_{h_2} + \ldots + \ell_{h_k}$, and $[L_{max}]_{ij} \supseteq x$ for all those j such that

$$L_j \supseteq \ell_{h_1} + \ell_{h_2} + \ldots + \ell_{h_k} .$$

Using either of these techniques, we get the graph of $C_{max} + L_{max}$ shown in Fig. 2.

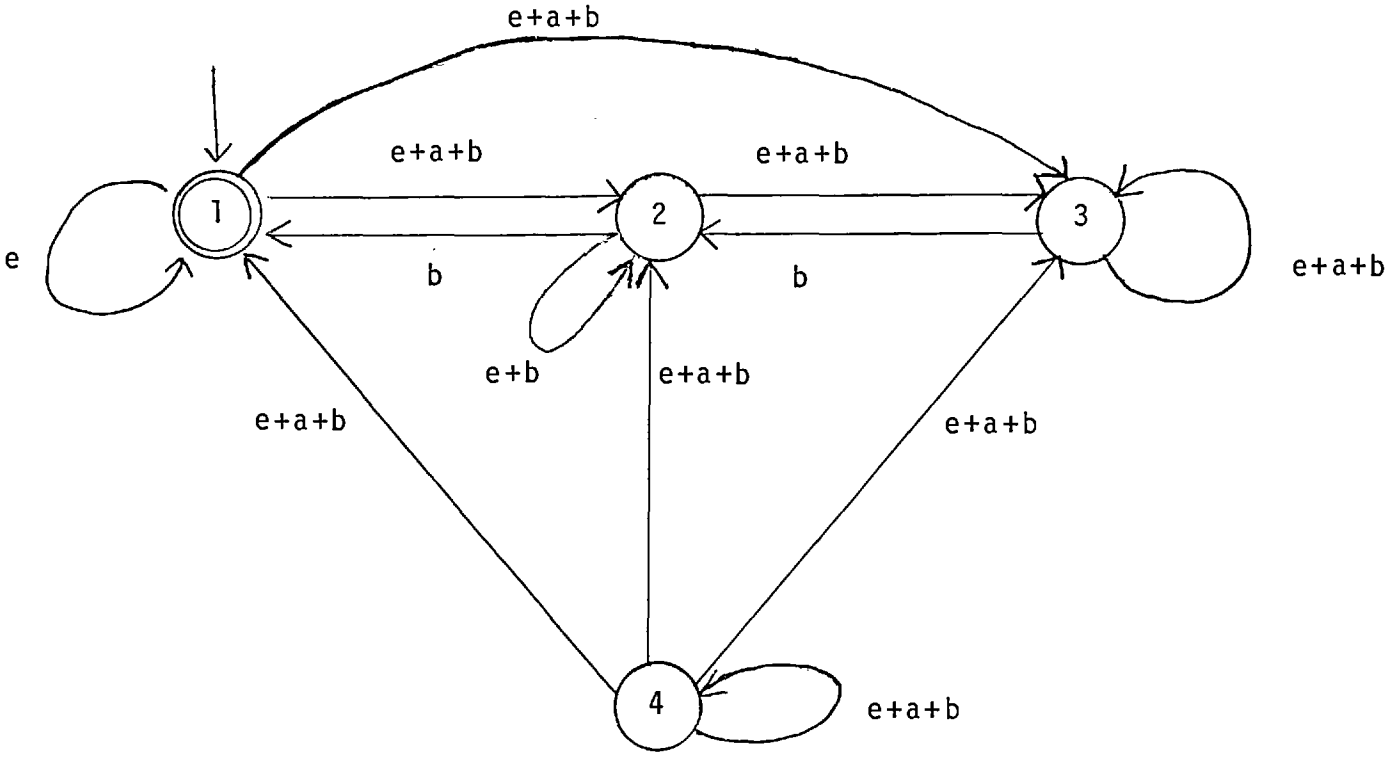


Fig. 2 $C_{max}+L_{max}$.

Finally to get the factor graph we remove those arc labels in $C_{max} + L_{max}$ which are in E or may be decomposed into paths in $((C_{max} + L_{max})\backslash E)^{2+*}$. The factor graph for this example is shown in Fig. 3. In both Figs. 2 and 3 we have indicated, in the usual way, that the graphs are recognisers of Q with start node s = 1 and terminal node t = 1.

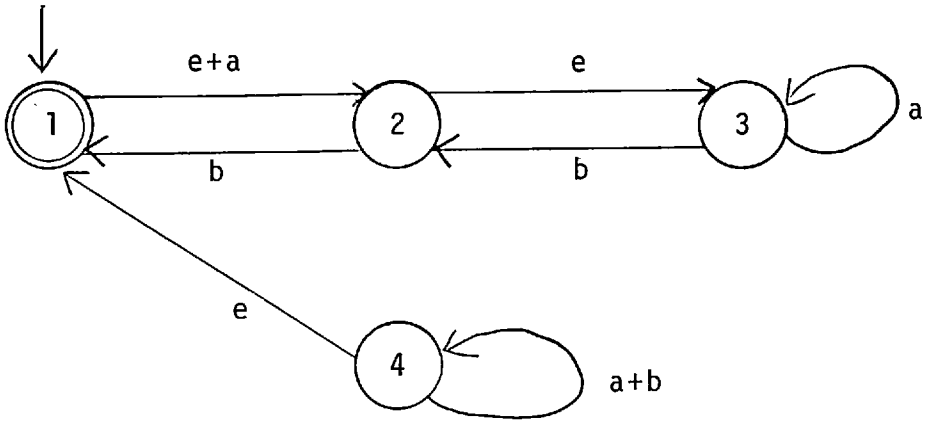


Fig. 3 The factor graph of Q.

6.5 The matrix $\overline{|Q|}$

From G_Q we could now calculate $\overline{|Q|} = G_Q^*$ using one of the standard methods of chapter II. However we can get the same information about $\overline{|Q|}$ by determining each entry Q_{ij} as a sum of c-classes of Q. To do this we replace each entry in G_Q by the c-class of which it is a representative (see Fig. 1(c)). Thus G_Q is represented by

$$\begin{bmatrix} \phi & c_0 + c_1 & \phi & \phi \\ c_2 & \phi & c_0 & \phi \\ \phi & c_2 & c_1 & \phi \\ c_0 & \phi & \phi & c_1 + c_2 \end{bmatrix}$$

and then calculate G_Q^* in the algebra $M_q(R(\underline{S}_Q))$ where $\underline{S}_Q$ is the semigroup of Q, and $R(\underline{S}_Q)$ is the regular algebra

generated by $\underline{S}_Q$ (c.f. I§2.3). For this example one can verify that G_Q^* is represented by

$$\begin{bmatrix} c_0+c_2+c_3+c_6 & c_0+c_1+c_2+c_3 & S & \phi \\ & +c_4+c_6+c_7 & & \\ c_2+c_6 & c_0+c_2+c_3 & S & \phi \\ & +c_4+c_6+c_7 & & \\ c_6 & c_2+c_3+c_6 & S & \phi \\ & +c_7 & & \\ S & S & S & S \end{bmatrix}$$

where S denotes the whole semigroup i.e. $S = \sum_{i=0}^{7} c_i$.

6.6 Some Remarks

There are various minor improvements one can make to the above method of calculating G_Q.

First of all, if it is only required to calculate G_Q, it is unnecessary to calculate the semigroup of the language. However, the representation of the matrix $\boxed{Q}$ in terms of c-classes, as in the last section, is (as we shall see) very useful and also much more informative than calculating regular expressions denoting each of the factors.

A second point concerns Tables 1(a) and (b). It should be noted that the third column of Table 1(b) can be deduced directly from the third column of Table 1(a) (or vice-versa), and thus gives redundant information.

This small amount of redundancy can, however, be quite useful in checking hand calculations and so is probably worth retaining.

Thirdly, we note that by length considerations,

$$G_Q = C_{min} + L_{min}, \text{ where}$$

$$C_{min} = (C_{max}\backslash E)\backslash(C_{max}\backslash E)^{2+*}$$

and $$L_{min} = L_{max}\backslash(L_{max} + C_{min})^{2+*} \quad .$$

The above formulae suggest that one first calculates C_{max} and from it C_{min}, and then determines L_{max} and from it L_{min}. The sum of C_{min} and L_{min} is then the factor graph G_Q. In fact one rarely needs calculate C_{max} and L_{max} explicitly because one can remove arcs from these matrices by inspection as they are being constructed. Note also that the second method of calculating C_{max} and L_{max} (§6.4(ii)) is preferable to the first, and hence the construction of Table 1(c) is unnecessary - although it does add some insight into what is happening.

Finally, a minor technical nuisance in the study of factors is that ϕ may be a factor. In this example $L_4=\phi$ is a left factor, but ϕ is not a right factor. If ϕ is a factor then the factor graph can have up to two "useless" nodes, i.e. nodes such that there is no path from node s to the node, (for example node 4 of Fig.3), or no path from the node to node t. If we are interested in the graph G_Q as a recogniser for Q, we can always ignore these nodes and consider the resulting all-admissible recogniser for Q. In

all future calculations we will take the liberty of disregarding this technical problem, and all the factor graphs we display will be all-admissible factor graphs.

7. An algorithm to calculate the factor graph

We are now in a position to summarise the steps in an algorithm to determine the (all-admissible) factor graph for a given regular language Q. We assume naturally that Q is given either as a regular expression or by a system of left (or right)-linear equations. Following the algorithm we have worked through another example, which shows explicitly the various steps of the algorithm.

Algorithm 1 To calculate the factor graph G_Q of a given regular language Q.

Step 1 Calculate the machine and anti-machine for the language Q. (Use the method of derivatives [3]). Label the states of the machine (anti-machine) $\ell_1, \ell_2, \ldots, \ell_m$ ($r_1, r_2, \ldots, r_{am}$) and use these labels to denote the corresponding ℓ-class (r-class).

Step 2 Construct two tables, the first listing the ℓ-classes of Q and the second the r-classes of Q. Each table has 3 columns. Construct first of all the first two columns of these tables, the first column containing simply a list of the labels ℓ_i (r_j) given to the ℓ-classes (r-classes) of Q, and the second column containing an arbitrary representative element of the corresponding class. The third column of each table is now constructed. In the first table this

column represents the various derivatives of Q as unions of r-classes of Q, and in the second table it represents the reverse of the various anti-derivatives of Q as unions of ℓ-classes of Q. Suppose the ℓ-class ℓ_i has representative x_i and the r-class r_j has representative y_j. Then r_j appears as a term in the ℓ_ith row of Table 1 if and only if $x_i y_j \in Q$, and similarly ℓ_i appears as a term in the r_jth row of Table 2 if and only if $x_i y_j \in Q$.

Step 3 Deduce the corresponding left and right factors of Q and label them $L_1, L_2, \ldots, L_q, R_1, R_2, \ldots, R_q$. Find the unique indices s and t such that $Q = L_t = R_s$.

To do this, one considers all subsets $\{\ell_{i_1}, \ldots, \ell_{i_k}\}$ (excluding the empty subset) of the ℓ-classes of Q and finds for each subset those classes $r_{j_1}, \ldots r_{j_n}$ common to the ℓ_{i_1}th, ℓ_{i_2}th, ..., ℓ_{i_k}th entries in the third column of Table 1. One then has $(\ell_{i_1} + \ldots + \ell_{i_k}) \cdot (r_{j_1} + \ldots + r_{j_n}) \subseteq Q$ is a subfactorization of Q in which $(r_{j_1} + \ldots + r_{j_n})$ is maximal. By inspecting all such subfactorizations one may deduce the left and right factors. $L_t = \ell_{t_1} + \ell_{t_2} + \ldots + \ell_{t_k}$ is that left factor such that the ℓ-class $\ell_{t_i} \subseteq L_t$ if and only if it corresponds to a terminal node of the machine for Q. Similarly $R_s = r_{s_1} + r_{s_2} + \ldots + r_{s_p}$ is that right factor such that the r-class $r_{s_j} \subseteq R_s$ if and only if it corresponds to a terminal node of the anti-machine for Q.

Step 4 Calculate C_{max}. Whence deduce

$$C_{min} = (C_{max}\backslash E)\backslash(C_{max}\backslash E)^{2+*} .$$

$[C_{max}]_{ij} \geq e$ if and only if $L_j \geq L_i$, and this can easily be deduced from the representation of L_i and L_j as unions of ℓ-classes of Q.

Step 5 Calculate L_{max}. Whence deduce

$$L_{min} = L_{max}\backslash((C_{max}+ L_{max})\backslash E)^{2+*}.$$

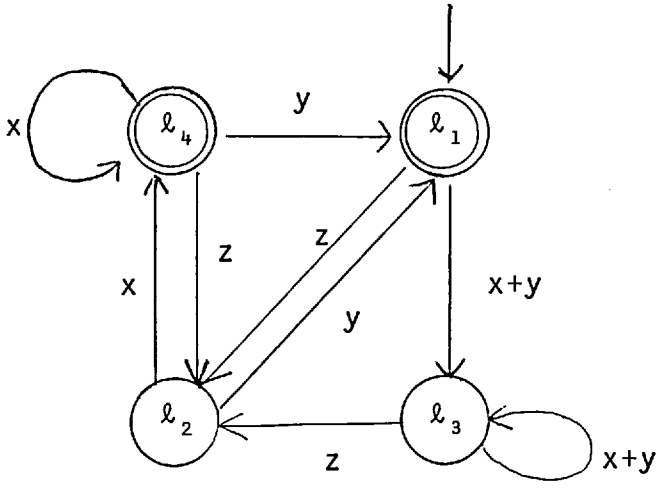
$[L_{max}]_{ij} \geq a$ if and only if $a \in V$ and $L_i.a \subseteq L_j$. This can also be easily deduced from the representation of L_i and L_j as unions of ℓ-classes of Q and the knowledge that $\ell_k.a \subseteq \ell_j$ if and only if under input a the ℓ_kth state of the machine for Q goes to state ℓ_j.

Finally $G_Q = C_{min} + L_{min}$.

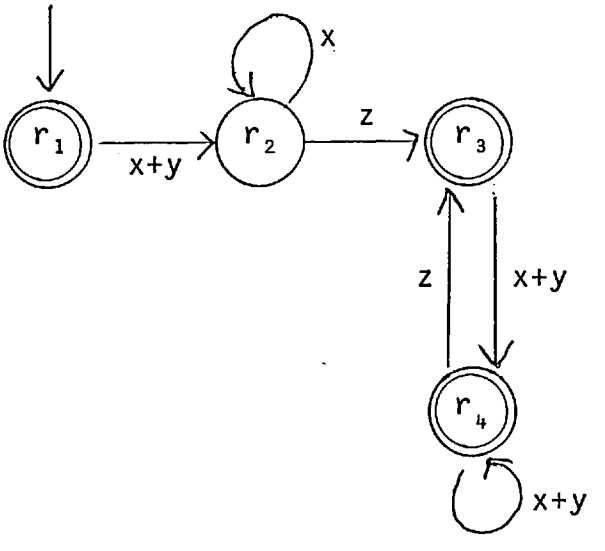
We shall now illustrate the various steps in the above algorithm by a second example.

Example 2 $Q = [(x+y)^*zx^*(x+y)]^*$

Step 1



(All-admissible) machine



(All-admissible) anti-machine

Step 2.

ℓ-class	Representative	Derivative
ℓ_1	e	$r_1 + r_3 + r_4$
ℓ_2	z	$r_2 + r_4$
ℓ_3	x	$r_3 + r_4$
ℓ_4	zx	$r_1 + r_2 + r_3 + r_4$

Table 1.

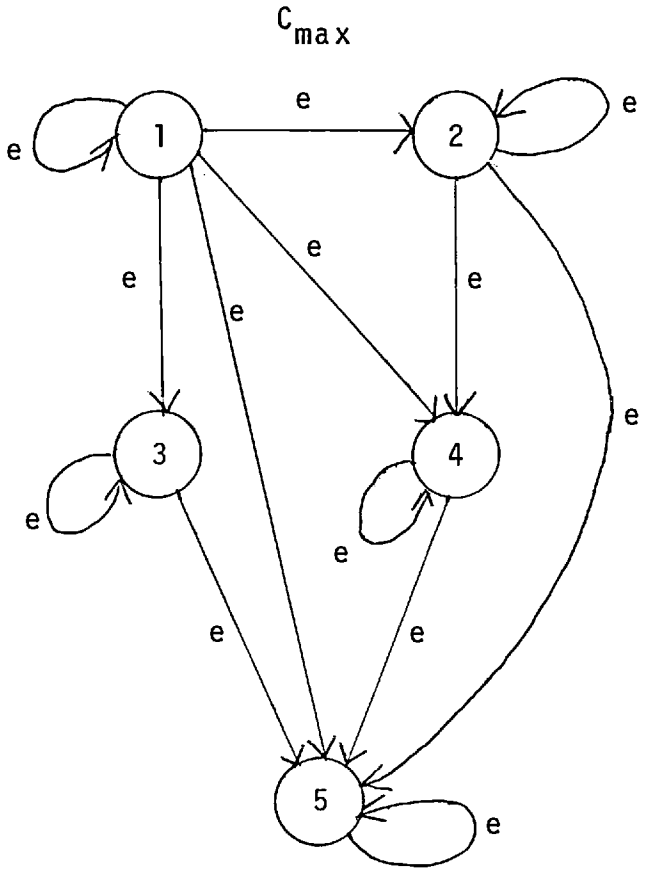
r-class	Representative	Reverse of anti-derivative
r_1	e	$\ell_1 + \ell_4$
r_2	x	$\ell_2 + \ell_4$
r_3	zx	$\ell_1 + \ell_3 + \ell_4$
r_4	xzx	$\ell_1 + \ell_2 + \ell_3 + \ell_4$

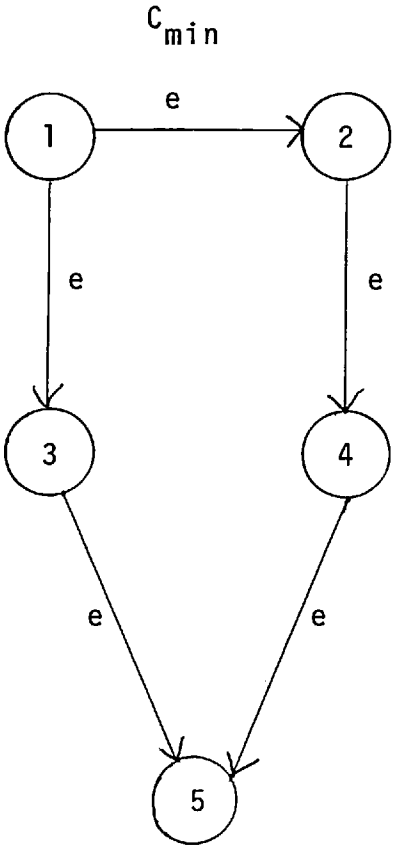
Table 2.

Step 3.

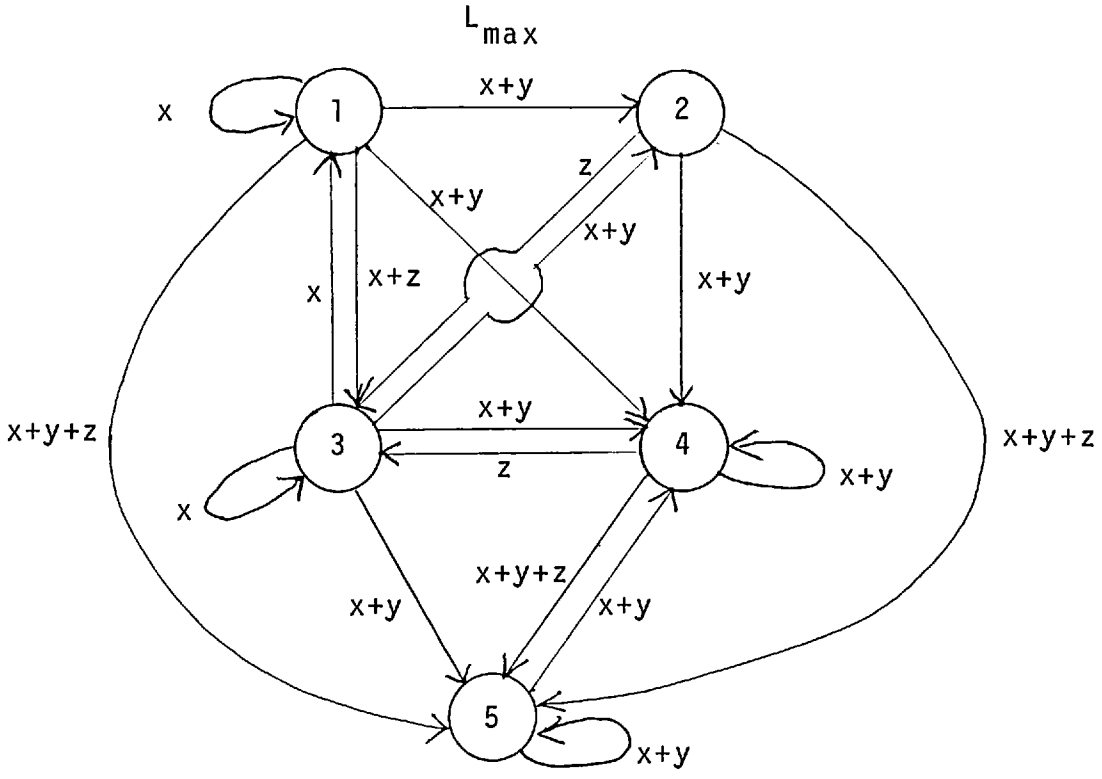
	Left factors			Right factors	
	L_1	ℓ_4		R_1	$r_1 + r_2 + r_3 + r_4$
$L_t =$	L_2	$\ell_1 + \ell_4$	$R_s =$	R_2	$r_1 + r_3 + r_4$
	L_3	$\ell_2 + \ell_4$		R_3	$r_2 + r_4$
	L_4	$\ell_1 + \ell_3 + \ell_4$		R_4	$r_3 + r_4$
	L_5	$\ell_1 + \ell_2 + \ell_3 + \ell_4$		R_5	r_4

Step 4

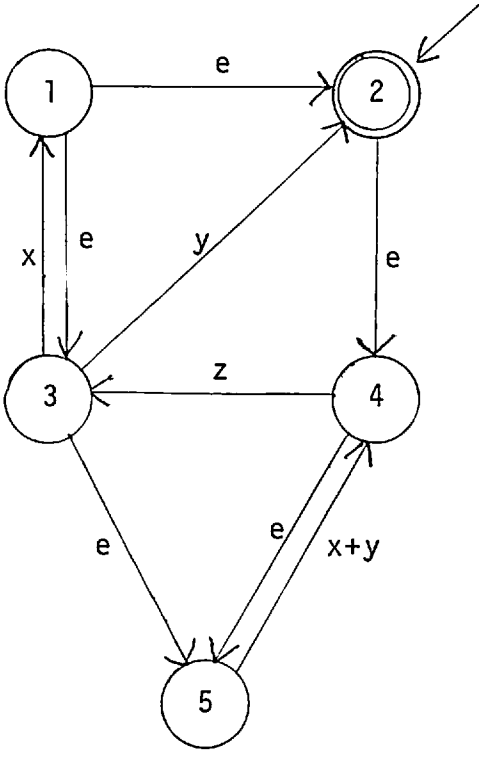




Step 5



(All admissible) Factor Graph $G_Q = C_{min} + L_{min}$



IV CALCULATING THE CLOSURE OF A FACTOR GRAPH

1. Introduction

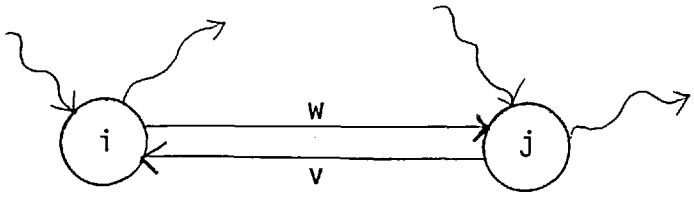
We recall that our original objective in studying Conway's factor matrix was to try to obtain a method of finding the star-height of a given regular language. It is not long, however, before one realises that this cannot be obtained directly from the factor graph for the language Q. Thus for the language $Q = (b + a(aa^*b)^*b)^*$ of Example 1 one obtains directly from the anti-machine for Q (see III §6, fig. 1(b)) the regular expression

$$Q = [(a + b)^*bb + b + e]\ (ab)^*$$

showing that Q has star-height one. However the factor graph of Q (III §6,fig. 3) has rank two.

Yet a very enigmatic feature of factor graphs, observed by examining just a few examples, is that very often one can see ad hoc ways of determining regular expressions for the languages which they recognise, which have star-height less than the rank of the factor graph. The purpose of this chapter is to develop a systematic way of finding the closure G_Q^* of the factor graph G_Q, which does have the property of often yielding expressions of star-height less than the rank of the graph G_Q.

The intuitive approach adopted to tackle this problem is based on the recursive nature of the definition of a language Q in terms of its factors (we use recursive here in the computer scientists sense, not the mathematicians). We observe that if there is a loop



such as the one shown above in the factor graph for Q we would get equations

$$Q_{it} = w \cdot Q_{jt} + \ldots$$
$$Q_{jt} = v \cdot Q_{it} + \ldots$$

in the system of equations which define $Q = Q_{st}$. In this way Q is defined recursively in terms of its factors. The question we ask is "when is a factor necessarily defined in terms of Q?" A clue to answering this question is given by Theorem 2.1 below, due to Conway, which states "factors of factors are themselves factors". This means that the relation "factor of" is a transitive relation, and so it can be naturally reduced to an equivalence relation on the factors, which we call "inseparable from". (Note that this is no different to considering the relation "is connected to" on nodes of a graph, and reducing it to "is strongly connected to", which is an equivalence relation on the nodes.) Examining the properties of factors further (section 3), we prove that the factor matrix of a factor F is a submatrix of $|\overline{Q}|$, and, moreover, is equal to $|\overline{Q}|$ if and only if F is inseparable from Q. Having made this observation an algorithm for

determining G_Q^* (sections 4 and 5) which exploits separability of factors is then obvious. The remaining sections are then concerned with discussing the applicability of the algorithms to the star-height problem.

2. Inseparable Factors

Theorem 2.1 (Conway) Let Q be any language, and let F be a factor of Q. Then any factor of F is also a factor of Q.

Corollary The relation "factor of" is a reflexive and transitive relation on the factors of any language Q.

Proof If F is a factor of Q it is maximal in some subfactorization $LFR \subseteq Q$ of Q. If H is a factor of F it is also maximal in some subfactorization $GHJ \subseteq F$. But then H is maximal in the subfactorization $LGHJR \subseteq Q$ of Q and so is a factor of Q. The corollary follows because Q is a factor of Q, i.e. the relation is reflexive. (That "factor of" is transitive is merely a restatement of the above theorem.)

Definition 2.2 Let F and H be factors of any language Q. We say F is inseparable from H if and only if F is a factor of H and H is a factor of F. Otherwise we say F and H are separable.

Lemma 2.3 Inseparability is an equivalence relation on the factors of Q.

3. Factor Matrices of Factors

Let the matrix M have nodes $N = \{1,2,\ldots,n\}$ and let $N' \subseteq N$ be any subset of this set. Then we shall call the matrix M' derived from M by simply removing the rows and columns corresponding to nodes $i \notin N'$ the submatrix of M defined by N'. If $N' \neq N$ we say M' is a proper submatrix of M.

Consider, now, any factor F of Q. Then F is some entry Q_{ij} of $\boxed{Q}$ (III4.1(i)), and each two term product $Q_{ik} \cdot Q_{kj}$ is, by III4.1(iv), a subfactorization of Q_{ij} (i.e. $Q_{ik} \cdot Q_{kj} \subseteq Q_{ij}$). Moreover, by III4.1(v), all the L.R factorizations of $F = Q_{ij}$ are included in the subfactorizations $Q_{ik} \cdot Q_{kj} \subseteq Q_{ij}$. These observations are highly suggestive that the factor matrix $\boxed{F}$ of F is a submatrix of $\boxed{Q}$, and, indeed, we shall show in this section that this is the case. Complications arise inevitably in the proof because factors of Q do not necessarily appear uniquely in the factor matrix, but often appear repeatedly (as in example 1).

To avoid confusion we shall henceforth always need to use subscripts or superscripts to identify the factor under consideration. Thus we shall use N_Q to denote the set of nodes of the factor graph G_Q, s_Q and t_Q (where we previously used just s and t) for the nodes mentioned in Theorem III4.1 (iii), and so on.

The proof of the main theorem, that the factor matrix $\boxed{F}$ of a factor $F = Q_{ij}$ of Q is a submatrix of $\boxed{Q}$, follows

from four simple lemmas. The first lemma recognises that F may occur more than once in $\lceil Q \rceil$, and so identifies unique nodes s_F and t_F such that $F = Q_{s_F t_F}$ and which will play the same role in $\lceil F \rceil$ as s_Q and t_Q (see III4.1(iii)). Following this we define a subset N_F of the nodes N_Q of the matrix $\lceil Q \rceil$, and in lemmas 3.3 and 3.4 show that $k \in N_F \Rightarrow Q_{s_F k} \cdot Q_{k t_F} \subseteq Q_{s_F t_F} = F$ is a factorization of F and that these include all the L·R factorizations of F. The final step is to show that if k and $m \in N_F$, Q_{km} is maximal in $Q_{s_F k} \cdot Q_{km} \cdot Q_{m t_F} \subseteq F$. Then by the definition of $\lceil F \rceil$ the submatrix of $\lceil Q \rceil$ defined by the set of nodes N_F is the matrix $\lceil F \rceil$.

Lemma 3.1 Let $F = Q_{ij}$ be a factor of Q. Then $\exists$ indices s_F and t_F such that $F = Q_{s_F t_F}$ and $Q_{s_F s_F}$ and $Q_{t_F t_F}$ are both factors of F.

Proof By III4.1(ii), $L_i Q_{ij} \subseteq L_j$ is a subfactorization in which Q_{ij} is maximal. Let this be dominated by the factorization $L_{s_F} \cdot Q_{s_F j} \subseteq L_j$. (Note that III4.1(v) is being used implicitly here.) Then, by Lemma III3.1, $Q_{ij} = Q_{s_F j}$, and by III4.1(iii) the index s_F is uniquely defined. Now let t_F be that unique index defined by $Q_{s_F t_F} \cdot R_{t_F} \subseteq R_{s_F}$ is a factorization which dominates the subfactorization $Q_{s_F j} \cdot R_j \subseteq R_{s_F}$. Then also $Q_{s_F t_F} = Q_{s_F j}$ and hence $Q_{s_F t_F} = Q_{ij} = F$. To prove the last part, we note that, by construction, $L_{s_F} \cdot Q_{s_F t_F} \cdot R_{t_F} \subseteq Q$ is a factorization of Q. Thus, using III4.1(ii),

$$L_{s_F} \cdot Q_{s_F s_F} \cdot Q_{s_F t_F} \cdot Q_{t_F t_F} \cdot R_{t_F} \subseteq Q \text{ is a factorization,}$$

and hence $Q_{s_F s_F}$ and $Q_{t_F t_F}$ must be maximal in $Q_{s_F s_F} \cdot Q_{s_F t_F} \cdot Q_{t_F t_F} \trianglelefteq Q_{s_F t_F}$ and so are factors of $Q_{s_F t_F} = F$.

Definition 3.2 The subset N_F of the set N_Q of nodes of the factor matrix $\lceil Q \rceil$ is defined by $k \in N_F \Leftrightarrow Q_{kt_F}$ is a factor of F and $L_k \cdot Q_{kt_F} \trianglelefteq L_{t_F}$ is a factorization of L_{t_F}.

Lemma 3.3 $k \in N_F \Rightarrow Q_{s_F k} \cdot Q_{kt_F} \trianglelefteq Q_{s_F t_F}$ is a factorization of F.

Proof By definition $k \in N_F$ implies $L_k \cdot Q_{kt_F} \trianglelefteq L_{t_F}$ is a factorization, which implies, by III4.1(ii), that $L_{s_F} \cdot Q_{s_F k} \cdot Q_{kt_F} \trianglelefteq L_{t_F}$ is a subfactorization in which $Q_{s_F k}$ is maximal. But by the definition of t_F in the proof of lemma 3.1, $L_{s_F} \cdot Q_{s_F t_F} \trianglelefteq L_{t_F}$ is a factorization. Therefore, $Q_{s_F k}$ must also be maximal in $Q_{s_F k} \cdot Q_{kt_F} \trianglelefteq Q_{s_F t_F}$ and so is a left factor of F. Q_{kt_F} is a factor by assumption, so the lemma follows immediately.

Lemma 3.4 If $L^F \cdot R^F \trianglelefteq F$ is a factorization of F, $\exists$ a unique node $k \in N_F$ such that $Q_{s_F k} = L^F$ and $Q_{kt_F} = R^F$.

Proof By III4.1(v), $L^F \cdot R^F \trianglelefteq F = Q_{s_F t_F}$ implies $L^F \trianglelefteq Q_{s_F p}$ and $R^F \trianglelefteq Q_{pt_F}$ for some p. Moreover, as L^F and R^F are factors, the last two inequalities must be equalities. Suppose $L_p \cdot Q_{pt_F} \trianglelefteq L_{t_F}$ is dominated by the factorization $L_k \cdot Q_{kt_F} \trianglelefteq L_{t_F}$. Now, by III4.1(ii), $Q_{kt_F} = Q_{pt_F} = R^F$, and $L_k \trianglerighteq L_p \Rightarrow$, by III4.1(ii), $Q_{pk} \trianglerighteq e \Rightarrow Q_{s_F k} \trianglerighteq Q_{s_F p} = L^F$. Hence $Q_{s_F k} \cdot Q_{kt_F} \trianglelefteq F$ dominates $L^F \cdot R^F \trianglelefteq F$; but, as the latter is a factorization, $Q_{s_F k} = L^F$ and $Q_{kt_F} = R^F$. Moreover, by definition 3.2, $k \in N_F$. Finally, k is unique follows directly from $Q_{kt_F} = Q_{pt_F}$ and $L_k \cdot Q_{kt_F} \trianglelefteq L_{t_F}$ is a factorization of L_{t_F}.

Lemma 3.5 Let k and m ε N_F. Then Q_{km} is maximal in $Q_{s_F k} \cdot Q_{km} \cdot Q_{m t_F} \subseteq Q_{s_F t_F}$.

Proof By definition, $m \varepsilon N_F \Rightarrow L_m \cdot Q_{m t_F} \subseteq L_{t_F}$ is a factorization; hence, by III4.1(ii), $L_k \cdot Q_{km} \cdot Q_{m t_F} \subseteq L_{t_F}$ is a subfactorization in which Q_{km} is maximal. But $L_k \cdot Q_{k t_F} \subseteq L_{t_F}$ is also a factorization of L_{t_F}, from which we conclude that Q_{km} must be maximal in $Q_{km} \cdot Q_{m t_F} \subseteq Q_{k t_F}$. Thus, by lemma 3.3, Q_{km} is maximal in $Q_{s_F k} \cdot Q_{km} \cdot Q_{m t_F} \subseteq Q_{s_F t_F}$.

Theorem 3.6 F is a factor of Q$\Leftrightarrow$ the factor matrix $\boxed{F}$ of F is a submatrix of the factor matrix $\boxed{Q}$ of Q.

Proof Let F be a factor of Q. Then if we compare lemmas 3.3 to 3.5 with the definition of the factor matrix $\boxed{F}$ of F at the beginning of Section III 4, we see immediately that the submatrix of $\boxed{Q}$ defined by the set N_F is indeed $\boxed{F}$. Conversely if $\boxed{F}$ is a submatrix of $\boxed{Q}$, F is an entry in $\boxed{Q}$ and so is a factor of Q (see III4.1(i)).

Corollary 1 Let F and H be two factors of a regular language Q. Then F is inseparable from H

$\Leftrightarrow$ they have the same factor matrix

$\Leftrightarrow$ they have the same factor graph.

Proof F is a factor of H$\Leftrightarrow$ $\boxed{F}$ is a submatrix of $\boxed{H}$. H is a factor of F$\Leftrightarrow$ $\boxed{H}$ is a submatrix of $\boxed{F}$. Hence F is inseparable from H$\Leftrightarrow$ $\boxed{F} = \boxed{H}$. The rest follows from the uniqueness of the factor graph.

Corollary 2 Let F be a regular language, and let C^F_{max} and L^F_{max} be the maximal constant and linear matrices such that $(C^F_{max} + L^F_{max})^* = \boxed{F}$. Then F is a factor of Q if and only if $C^F_{max} + L^F_{max}$ is a submatrix of $C^Q_{max} + L^Q_{max}$.

Proof If F is a factor of Q, $C^F_{max} + L^F_{max} \subseteq \overline{|F|}$, which is a submatrix of $\overline{|Q|}$. Thus by maximality of $C^Q_{max} + L^Q_{max}$, $C^F_{max} + L^F_{max}$ is a submatrix of it. $\Leftarrow$ is obvious.

Finally, we recall that inseparability of factors was defined as a symmetric closure of the relation "factor of". The other "half" of this relation - the anti-symmetric half - is a partial ordering on the classes of inseparable factors, or equivalently, by Corollary 1 of the last theorem, a partial ordering on the factor graphs of factors. This is now defined.

Definition 3.7 Let F and H be two factors of a regular language Q, and let G_F and G_H be their factor graphs. Then we define the relation $\lesssim$ on the factor graphs of factors of Q by

$$G_F \lesssim G_H \quad \text{iff } \overline{|F|} \text{ is a submatrix of } \overline{|H|}.$$

Theorem 3.8 $\lesssim$ is a (reflexive) partial ordering on the factor graphs of factors of Q.

The proof is obvious.

4. Example 1 again

The above theorem immediately suggests a new method of calculating $\overline{|Q|}$ when Q has a factor H which is separable from Q. For, using our knowledge that $\overline{|H|}$ is a submatrix of $\overline{|Q|}$, we can write

$$\overline{|Q|} = \begin{bmatrix} C_{11} & C_{12} \\ C_{21} & \overline{|H|} \end{bmatrix} \qquad (1)$$

where C_{11} is a square matrix. The factor graph G_Q can be decomposed into corresponding submatrices

$$G_Q = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \tag{2}$$

Using this notation the escalator method (II § 4.3) is given by the formulae

$$C_{11} = A_{11}^* + A_{11}^* A_{12} \overline{|H|} A_{21} A_{11}^* \tag{3}$$

$$C_{12} = A_{11}^* A_{12} \overline{|H|} \tag{4}$$

$$C_{21} = \overline{|H|} A_{21} A_{11}^* \tag{5}$$

where $\overline{|H|}$ is usually given as $\overline{|H|} = (A_{22} + A_{21}\ A_{11}^*\ A_{12})^*$. However $\overline{|H|}$ is the factor matrix of the language H, and so

$$\overline{|H|} = G_H^* \tag{6}$$

where G_H is the factor graph of H.

Formulae (3), (4), (5) and (6) form the basis of an algorithm to compute $\overline{|Q|}$. We shall first use these formulae to calculate the factor matrix of Example 1.

For ease of reference the all-admissible factor graph of $Q = Q_{11}$ is reproduced below.

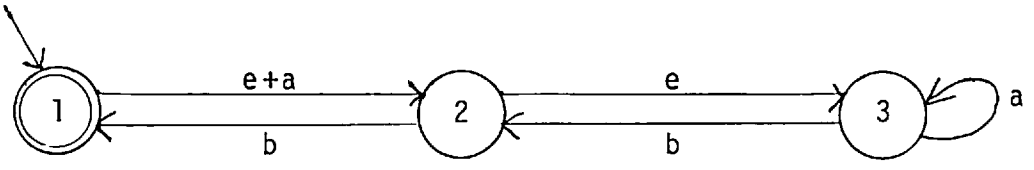


Fig. 1

The first step is to find the factor graphs of factors of Q. We already have an algorithm to do this, contained in the proof of Theorem 3.6. If $F = Q_{ij}$ is a factor of Q, this involves locating other entries $Q_{i'j'}$ which are also equal to F, and using lemma 3.1 to choose s_F and t_F. Then the L R factorizations of F are found by checking whether any one subfactorization $Q_{s_F p} \cdot Q_{p t_F} \subseteq F$ is dominated by another subfactorization $Q_{s_F k} \cdot Q_{k t_F} \subseteq F$; finally definition 3.2 is used to choose those $k \in N_F$. It is here that the representation of each entry in $G_Q^* = \overline{|Q|}$ as a union of c-classes of Q is particularly useful. For this example we have already shown that G_Q^* may be represented by

$$\begin{bmatrix} c_0 + c_2 + c_3 + c_6 & c_0 + c_1 + c_2 + c_3 + c_4 + c_6 + c_7 & S \\ c_2 + c_6 & c_0 + c_2 + c_3 + c_4 + c_6 + c_7 & S \\ c_6 & c_2 + c_3 + c_6 + c_7 & S \end{bmatrix}$$

(Refer to III§6fig.1(c) for the meaning of $c_0, \ldots, c_7$). Using this representation of G_Q^* to determine whether $Q_{ip} \cdot Q_{pj} \subseteq Q_{ij}$ dominates $Q_{ik} \cdot Q_{kj} \subseteq Q_{ij}$ it is only necessary to compare finite subsets of the set of elements in the semigroup against one another.

This example has been chosen for its simplicity. Only one factor, $Q_{33} = Q_{13} = Q_{23}$, appears more than once in the matrix, and this, from fig. 1, is obviously equal to (a+b)*, and so has the factor graph shown below.

Fig. 2

All other factors Q_{ij} can be easily shown to have the same factor graph as $Q = Q_{11}$. For j=1 and for all i, one need only check that $Q_{i1}.Q_{11} \subseteq Q_{i1}$ is not dominated by any subfactorization $Q_{ik}.Q_{k1} \subseteq Q_{i1}$. This is clearly impossible since $Q_{11} = c_0 + c_2 + c_3 + c_6 \supseteq Q_{21} = c_2 + c_6 \supseteq Q_{31} = c_6$. Thus $Q = Q_{11}$ is a factor of Q_{i1}, and so they are inseparable. Similarly Q_{12} can be shown to be inseparable from Q. This only leaves Q_{22}, but since $Q_{12} \supset Q_{22}$ and $Q_{12} \supset Q_{32}$ ($c_1 \subseteq Q_{12}$, $c_1 \notin Q_{22} + Q_{32}$), Q_{12} is a factor of Q_{22}. But Q_{12} is inseparable from Q, hence so is Q_{22}. Now in order to use formulae (3), (4), (5) and (6), we write

$$G_Q = \left[\begin{array}{cc|c} b & & e+a \\ & b & a \end{array}\right] = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

A^*_{11} is calculated by a standard elimination method and found to be

$$A^*_{11} = \begin{bmatrix} (b+ab)^* & (b+ab)^*(e+a) \\ (b+ba)^*b & (b+ba)^* \end{bmatrix}$$

The factor H in (3) - (6) is Q_{33}, and its factor matrix is determined from fig.2.

$$\boxed{H} = [(a+b)^*]$$

We now have all the information necessary to apply the formulae (3) - (6), giving

$$\boxed{Q} = \begin{bmatrix} (b+ab)^*(e+a)(a+b)^*b(b+ba)^*b + (b+ab)^* & (b+ab)^*(e+a)(a+b)^*b(b+ba)^* + (b+ab)^*(e+a) & (b+ab)^*(e+a)(a+b)^* \\ (b+ba)^*(a+b)^*b(b+ba)^*b + (b+ba)^*b & (b+ba)^*(a+b)^*b(b+ba)^* + (b+ba)^* & (b+ba)^*(a+b)^* \\ (a+b)^*b(b+ba)^*b & (a+b)^*b(b+ba)^* & (a+b)^* \end{bmatrix}$$

The regular expressions appearing in $\overline{|Q|}$ could be made much simpler had we used the knowledge that $Q_{33} = Q_{31} = Q_{32} = (a+b)^*$. However this is irrelevant to our aim which is simply to obtain regular expressions of smallest star-height. Indeed for this example we have achieved this aim, since all the expressions appearing in $\boxed{Q}$ are of star-height one. Moreover this is <u>strictly less than</u> the rank of the factor graph. The final expression for Q is

$$Q = Q_{11} = (b+ab)^*(e+a)(a+b)^*b(b+ba)^*b+(b+ab)^*$$

which simplifies to

$$Q = (a+b)^*b(b+ba)^*b+(b+ab)^* .$$

5. <u>An Algorithm for Calculating $\boxed{Q}$</u>

We shall now formulate the algorithm for calculating $\overline{|Q|}$. In general the algorithm is not quite as simple as in the example above. In the above example all the factor graphs G_H of factors of Q were totally ordered by the relation $\preccurlyeq$ (there were only two!). Technical difficulties arise in the algorithm because in general the relation $\preccurlyeq$ is a partial ordering on the factor graphs associated with Q.

Algorithm 2. To calculate $\overline{|Q|}$ for a given regular language Q.

Step 1 Find the factor graphs G_H of all factors H of Q (including G_Q). A method of doing this is given in the next section. Associate with each factor graph G_H a subset $N_H = \{i_1, i_2, \ldots, i_h\}$ of the nodes $\{1, 2, \ldots, q\}$ of the factor graph G_Q, where the submatrix of $\overline{|Q|}$ defined by the set N_H is the factor matrix $\overline{|H|}$ of H. Note that for some factors H there may be more than one submatrix of $\overline{|Q|}$ which is equal to $\overline{|H|}$ (see e.g. the next example). For the purposes of exposition, we shall assume these factor graphs to be distinct.

Step 2 Calculate the upper semi-lattice defined by the partial ordering $\lesssim$ on the distinct factor graphs G_H of factors H of Q. We shall call G_H a minimal element of this lattice if there is no other factor graph G_F such that $G_F \prec G_H$. G_Q is of course the only maximal element.

Step 3 Choose any path $G_Z \prec G_Y \ldots G_S \prec G_R \prec G_Q$ from a minimal element G_Z of the semi-lattice to the maximal element G_Q. This defines a sequence $N_Z \subset N_Y \subset \ldots \subset N_S \subset N_R \subset N_Q$ of the nodes of G_Q. Reorder the nodes of G_Q such that the nodes in the set $N_Q \backslash N_R$ are numbered from 1 to $|N_Q \backslash N_R|$, the nodes of $N_R \backslash N_S$ are numbered from $|N_Q \backslash N_R| + 1$ to $|N_Q \backslash N_S|$ etc. Within any of sets $N_R \backslash N_S$ the order is immaterial.

Step 4 For the minimal element G_Z of the path calculate $G_Z^* = \overline{|Z|}$ using a standard elimination method.

Step 5 Suppose the current factor matrix that has been calculated is $G_H^* = \boxed{H}$. If $G_H = G_Q$, stop; otherwise let G_F be the next point in the chain. Split G_F as shown below

$$G_F = \left[\begin{array}{c|c} A_{FF} & A_{FH} \\ \hline A_{HF} & A_{HH} \end{array}\right] \begin{array}{l} \} \text{ Nodes in } N_F \backslash N_H \\ \} \text{ Nodes in } N_H \end{array} \Big\} \text{ Nodes in } N_F$$

and correspondingly define C_{FF}, C_{FH}, C_{HF} and C_{HH} by

$$G_F^* = \boxed{F} = \left[\begin{array}{c|c} C_{FF} & C_{FH} \\ \hline C_{HF} & C_{HH} \end{array}\right]$$

Compute A_{FF}^* using a standard elimination method.

Compute all entries of G_F^* using

$$C_{HH} = \boxed{H} \quad \text{(which has already been calculated)}$$

$$C_{FF} = A_{FF}^* + A_{FF}^* A_{FH} \boxed{H} A_{HF} A_{FF}^*$$

$$C_{FH} = A_{FF}^* A_{FH} \boxed{H}$$

$$C_{HF} = \boxed{H} A_{HF} A_{FF}^* .$$

Step 6 Repeat Step 5.

The above algorithm requires that one calculate the factor graphs of factors of Q. Once the factor graph of Q has been calculated it is not necessary to repeat all the steps of the algorithm given in section 7 of Chapter III to find the factor graph of any factor F of Q. Instead one essentially uses the proof of lemma 3.1 to find nodes s_F and

and t_F such that $F = Q_{s_F t_F}$, and then definition 3.2 and corollary 2 to theorem 3.6 enable one to deduce $C^F_{max} + L^F_{max}$ directly from $C^Q_{max} + L^Q_{max}$. For ease of reference the steps in this algorithm are given below.

Algorithm 3 To calculate factor graphs G_H of factors H of Q.

Suppose $H = Q_{ij}$ is the (i,j)th entry of $\lceil Q \rceil$.

Step 1 Calculate $C^Q_{max} + L^Q_{max}$ and deduce G_Q.

Step 2 Calculate $(C^Q_{max} + L^Q_{max})^*$ in the algebra $M_q(R(\underline{S}_Q))$. ($R(\underline{S}_Q)$ is the regular algebra generated by the semigroup $\underline{S}_Q$ of the language Q). In other words calculate each entry of $\lceil Q \rceil$ as a union of c-classes of Q. Let this matrix be denoted $\lceil C(Q) \rceil$.

In the following steps, in order to check that $Q_{k\ell} \supseteq Q_{mn}$, one checks that

$$C(Q)_{k\ell} = c_{i1} + c_{i2} + \ldots + c_{ix} \supseteq C(Q)_{mn} = c_{j1} + c_{j2} + \ldots + c_{jy}.$$

This involves comparing two finite sets for set inclusion.

Step 3 Consider $H = Q_{ij}$, and consider all nodes i' such that $Q_{ij} = Q_{i'j}$. Let s_H be that node i' such that L_{s_H} is maximal. Now consider $Q_{s_H j}$ and all nodes j' such that $Q_{s_H j} = Q_{s_H j'}$. Let t_H be that node j' such that R_{t_H} is maximal.

Step 4 We now have $H = Q_{s_H t_H}$. Compare all subfactorizations $Q_{s_H k} \cdot Q_{k t_H} \subseteq H$ and $Q_{s_H m} \cdot Q_{m t_H} \subseteq H$ for one dominating the other; thus deduce the right factors $Q_{k t_H}$ of H.

Step 5 For all k such that $Q_{k t_H}$ is a right factor of H, let k' be that node such that $Q_{k t_H} = Q_{k' t_H}$ and $L_{k'}$ is maximal.

Let N_H be the set of all such k'.

Step 6 $C^H_{max} + L^H_{max}$ is the submatrix of $C^Q_{max} + L^Q_{max}$ defined by the set of nodes N_H. Calculate G_H using

$$G_H = ((C^H_{max} + L^H_{max})\backslash E)\backslash((C^H_{max} + L^H_{max})\backslash E)^{2+*}.$$

Needless to say in practical applications it is not necessary to go to quite these lengths to calculate G_H, and various ad hoc techniques, such as were used in example 1, can be acquired with practice.

6. Two More Examples

Consider Q = a(a+b)*b(a+b)*a.

We shall apply algorithm 2 to determine the factor matrix $|\overline{Q}|$.

Step 1 The factor graph and semigroup of Q are shown below.

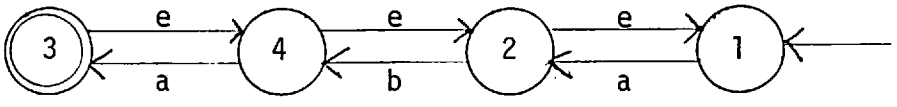


Fig. 3 Factor Graph of $Q = Q_{13}$.

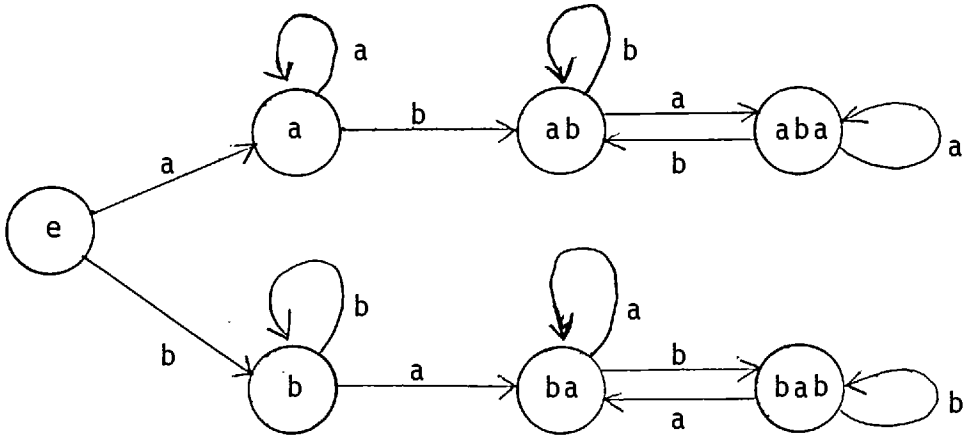


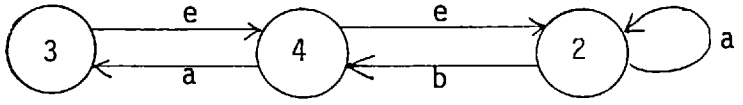
Fig. 4 Semigroup of Q.

The matrix $\overline{|C(Q)|}$ which exhibits each entry of $\overline{|Q|}$ as a union of c-classes of Q is easily found to be:

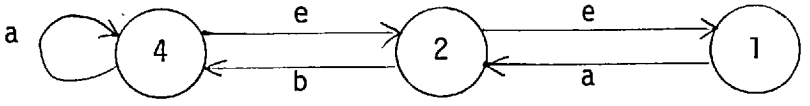
$$\overline{|C(Q)|} = \begin{bmatrix} e+a+ab+aba & a+ab+aba & aba & aba+ab \\ S & S & aba+ba & b+ab+ba+bab+aba \\ S & S & e+a+ba+aba & S \\ S & S & a+ba+aba & S \end{bmatrix}$$

where S denotes the whole semigroup.

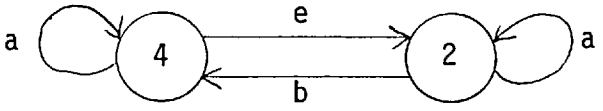
Applying algorithm 3 we can deduce the following factor graphs for factors Q_{ij} of Q.



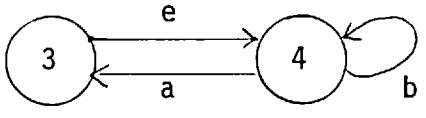
(a) Factor graph of Q_{23}



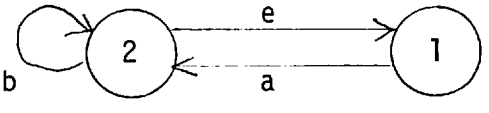
(b) Factor Graph of Q_{14}



(c) Factor graph of Q_{24}

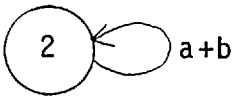


(d) Factor Graph of Q_{43}



(e) Factor Graph of Q_{12}

(f) Factor Graph of Q_{44}



(g) Factor Graph of Q_{22}

Fig. 5

Step 2 The semi-lattice defined by the relation $\prec$ is shown graphically below. Each node contains a representative element of the class of inseparable factors to which the node corresponds, together with the set of nodes which define the submatrix of $\boxed{Q}$ which equals the particular factor matrix.

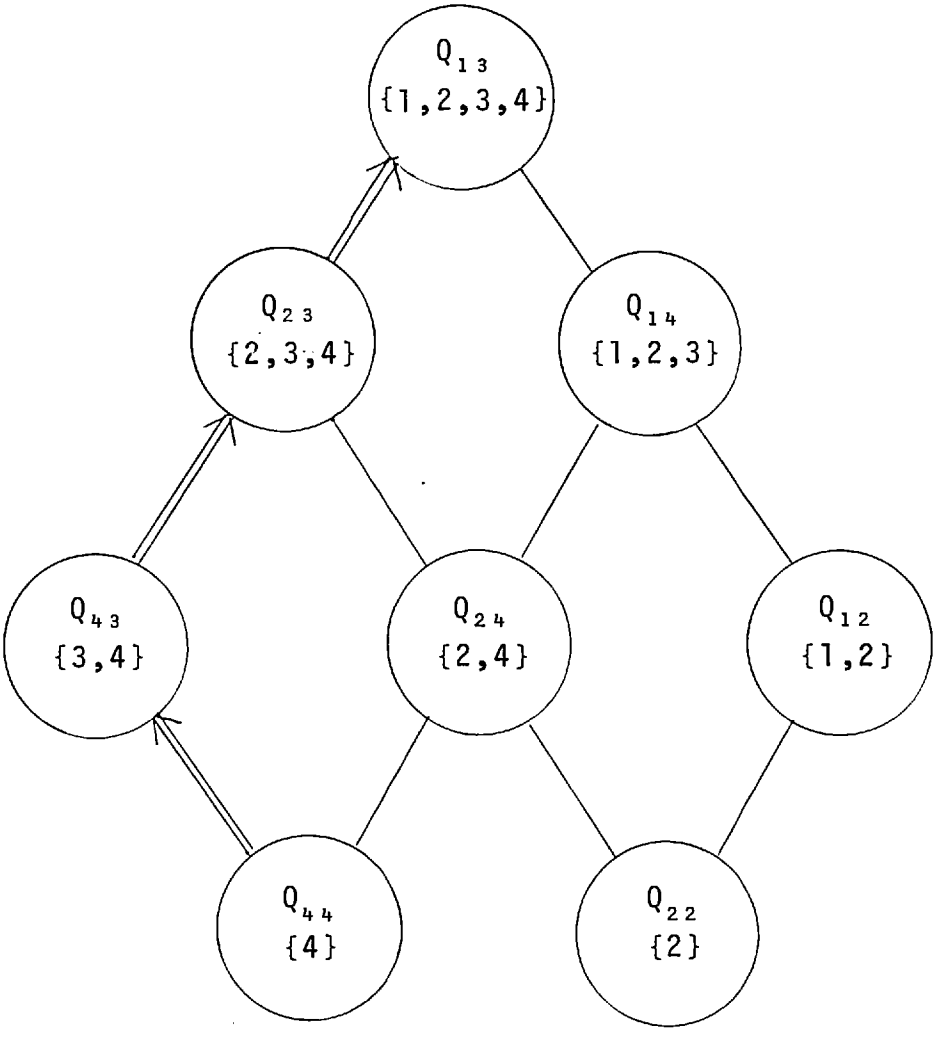


Fig. 6 Semi-lattice $\prec$

Step 3 The particular path from bottom to top of this semilattice, which we will use in the calculation of $\boxed{Q}$, has been arrowed. The node numbering has already been chosen to meet the requirements of step 3 of algorithm 1.

Step 4 Clearly $\boxed{Q_{44}} = [(a + b)^*]$. (from fig. 5(f)).

Step 5 Repeating step 5 of the algorithm we get successively

$$\boxed{Q_{43}} = \begin{bmatrix} e+(a+b)^*a & (a+b)^* \\ (a+b)^*a & \boxed{Q_{44}} \end{bmatrix} \quad \text{(from fig. 5(d))}$$

$$\boxed{Q_{23}} = \begin{bmatrix} a^*+a^*b(a+b)^*a^* & a^*b(a+b)^*a & a^*b(a+b)^* \\ (a+b)^*a^* & \boxed{Q_{43}} & \\ (a+b)^*a^* & & \end{bmatrix}$$

(from fig.5(a))

$$\boxed{Q} = \begin{bmatrix} e+a(a^*+a^*b(a+b)^*a^*) & a(a^*+a^*b(a+b)^*a^*) & aa^*b(a+b)^*a & aa^*b(a+b)^* \\ a^*+a^*b(a+b)^*a^* & & & \\ (a+b)^*a^* & \boxed{Q_{23}} & & \\ (a+b)^*a^* & & & \end{bmatrix}$$

(From Fig.3).

In each of these matrices we have only shown the new entries in the matrix. The final expression for Q is $Q_{13} = aa^*b(a+b)^*a$.

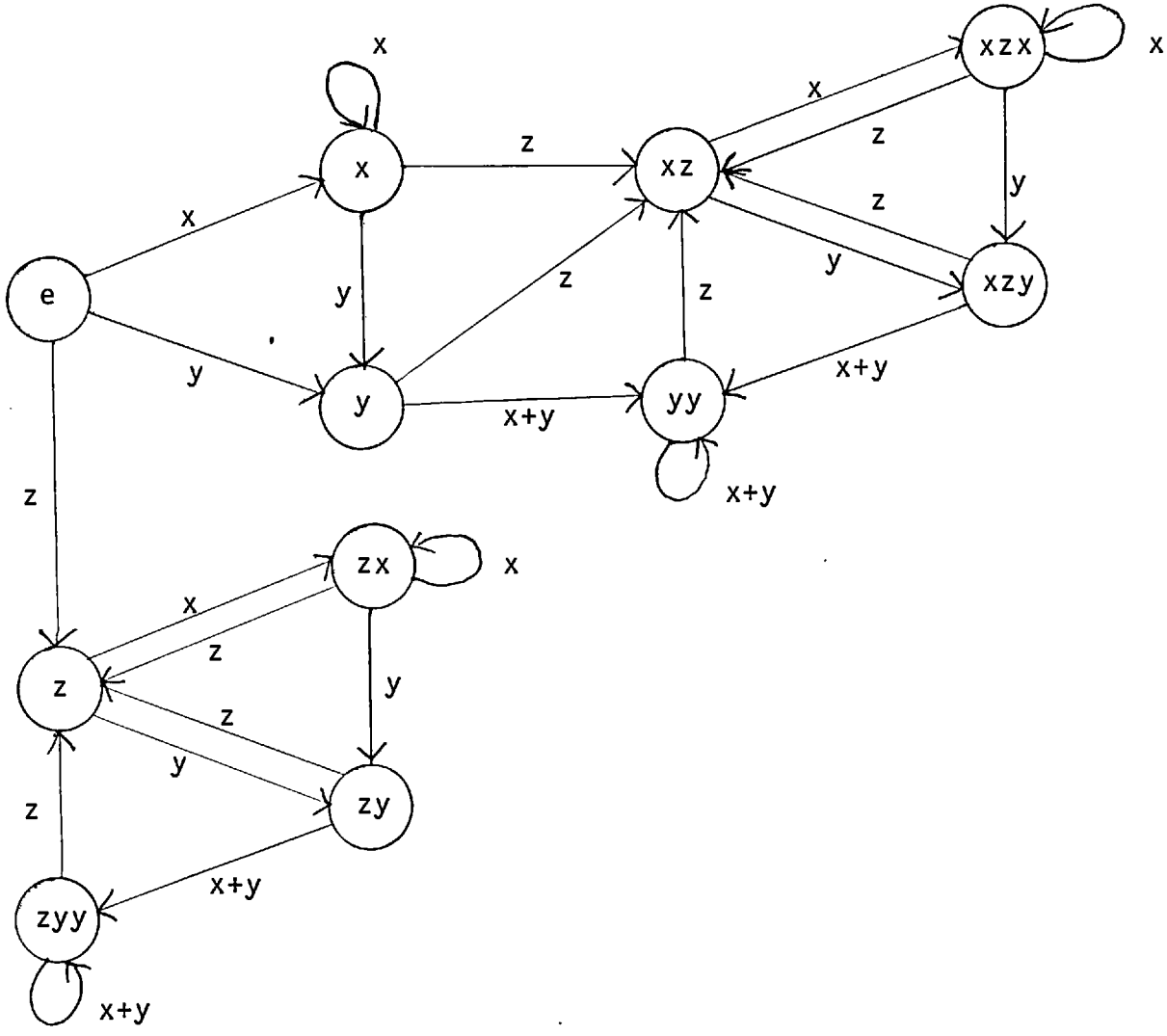
Remarks. This example is instructive for two reasons. Firstly it illustrates that in general one has a choice of path through the semilattice. Different paths will usually give different regular expressions for the language Q, although in this case all paths yield expressions of the same star height.

Whether there are examples where two different paths through the semi-lattice yield different star-height expressions I do not know. In any case one can always determine the star height that a particular path will give and choose one which is optimal. Secondly, all regular expressions appearing in $\boxed{Q}$ are of star height one, yet the rank of the factor graph is two! Indeed we shall show later that the algorithm always yields regular expressions having star-height less than or equal to the rank of the factor graph of Q.

Example 2 (continued)

Let us return to example 2 (see pages 111-114). The language considered is $Q = [(x+y)^*zx^*(x+y)]^*$, and its factor graph is reproduced in fig. 8(b) below. As we are only interested in deriving a regular expression for the language Q, which is the (2,2)th entry of the factor matrix, we shall not calculate the whole factor matrix using algorithm 2 but only Q_{22}. To do this we apply steps 1 to 3 of algorithm 2 as before, but then apply steps 4 and 5 in the reverse order. This results in a system of equations for Q_{22} which can then be solved to deduce a regular expression for the language $Q = Q_{22}$.

The semigroup of Q is shown in Fig. 7, and in table 1 we show the factor matrix as a union of congruence classes of Q.



Fig, 7. Semigroup of Q.

e +zy+zx +xzx+xzy	zx+xzx	z+zx +xz+xzx	M-xz	M+z
e+x +zx+xzx +y+zy+xzy	e+x +zx+xzx	e+x +zx+xzx +z+xz	M-xz	M+z
y+x +xzx+xzy	x +xzx	e+x +xz+xzx	N-e-xz	N
zy+zx +xzx+xzy	zx+xzx	z +xz+xzx	M-xz	M+z
xzx+xzy	xzx	xz+xzx	N-e-xz	N

where

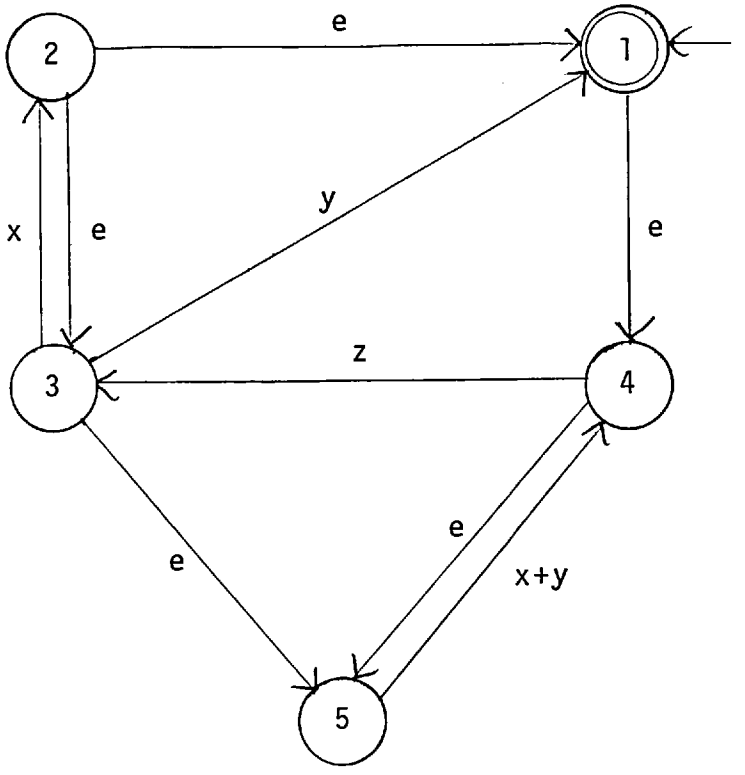
$M = e + x + y + xz + xzx + xzy + zx + zy + zyy + yy$

$N = e + x + y + xz + xzx + xzy + yy$.

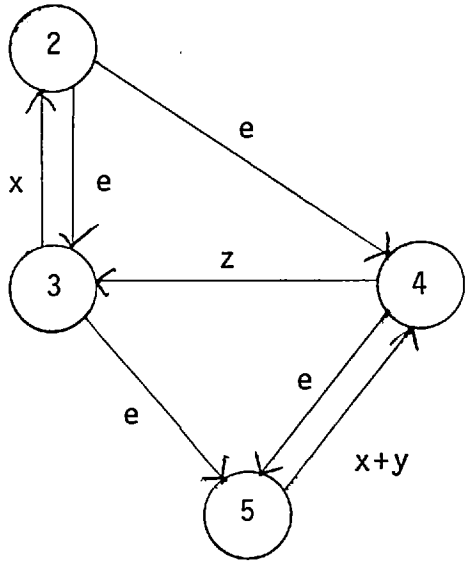
Table 1. The Matrix $\lceil C(Q) \rceil$

We now find that there are three factor graphs associated with the language Q. The first is the factor graph of Q, which for convenience we have reproduced below, and the second and third are factor graphs for the languages $\{Q_{33}, Q_{34}, Q_{43}\}$ and $\{Q_{44}, Q_{54}, Q_{45}, Q_{55}\}$, respectively.

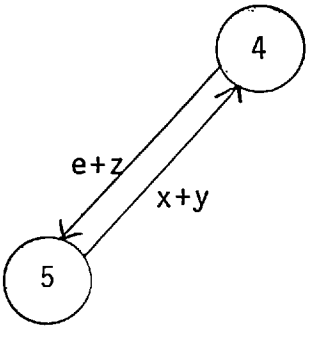
The ordering $\lessdot$ on the factor graphs is total and so there is no question of a choice of path through the semi-lattice.



(a) Factor Graph of Q



(b) Factor Graph of Q_{33}, Q_{34}, Q_{43}



(c) Factor Graph of Q_{44}, Q_{45}, Q_{54} and Q_{55}

Fig. 8

Applying the last step of algorithm 2 requires us to split G_Q in the manner indicated below:

$$G_Q = \left[\begin{array}{c|ccc} & & & e \\ \hline e & & e & \\ y & x & & e \\ & & z & e \\ & & & x+y \end{array}\right] = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right]$$

(Cf. figs. 8(a) and (b)).

$Q = [G_Q^*]_{11}$ is then calculated as

$$Q = A^*_{11} + [A^* \cdot B \cdot G_H^* \cdot C \cdot A^*]_{11}$$

where G_H is the second factor graph, i.e.

$$G_H = \begin{bmatrix} & e & e & \\ x & & & e \\ & z & & e \\ & & x+y & \end{bmatrix} .$$

A^*_{11} is just e, and so we can write down an equation for Q, viz:

$$\begin{aligned} Q = & \underbrace{e}_{A^*_{11}} + \underbrace{e}_{A^*_{11}} \cdot \underbrace{e}_{B_{14}} \cdot \underbrace{Q_{43}}_{[G_H^*]_{43}} \cdot \underbrace{y}_{C_{31}} \cdot \underbrace{e}_{A^*_{11}} \\ & + \underbrace{e}_{A^*_{11}} \cdot \underbrace{e}_{B_{14}} \cdot \underbrace{Q_{42}}_{[G_H^*]_{42}} \cdot \underbrace{e}_{C_{21}} \cdot \underbrace{e}_{A^*_{11}} \\ = & \; e + Q_{43} y + Q_{42} . \end{aligned} \qquad (1)$$

In the above equation we have indicated how each term arises. All other terms in the product $A^* \cdot B \cdot G_H^* \cdot C \cdot A^*$ are null. In order to calculate Q_{43} and Q_{42} we apply the same procedure to G_H. (See figs. 8(b) and (c)). First we write

$$G_H = \left[\begin{array}{c|c} \begin{matrix} & e \\ x & \end{matrix} & \begin{matrix} e & \\ & e \end{matrix} \\ \hline \begin{matrix} & z \\ & \end{matrix} & \begin{matrix} & e \\ x+y & \end{matrix} \end{array}\right] = \left[\begin{array}{c|c} K & L \\ \hline M & N \end{array}\right], \text{ say.}$$

Q_{43} and Q_{42} are then calculated using

$$Q_{43} = [G_F^* \cdot M \cdot K^*]_{43}$$

$$Q_{42} = [G_F^* \cdot M \cdot K^*]_{42}$$

where G_F is the minimal factor graph (fig. 8(c)):

$$G_F = \left[\begin{array}{cc} & e+z \\ x+y & \end{array}\right] .$$

By inspection $K^* = \left[\begin{array}{cc} x^* & x^* \\ x^*x & x^* \end{array}\right]$

$$\text{thus } Q_{43} = \underbrace{Q_{44}}_{[G_F^*]_{44}} \cdot \underbrace{z}_{M_{43}} \cdot \underbrace{x^*}_{K^*_{33}} = Q_{44}zx^* \qquad (2)$$

$$\text{and } Q_{42} = \underbrace{Q_{44}}_{[G_F^*]_{44}} \cdot \underbrace{z}_{M_{43}} \cdot \underbrace{x^*x}_{K^*_{32}} = Q_{44}zxx^* \; . \qquad (3)$$

Finally Q_{44} is calculated directly from the factor graph G_F (Fig. 8(c)). One easily obtains

$$Q_{44} = (x+y+zx+zy)^* \quad . \tag{4}$$

Using back-substitution, equations (1), (2), (3) and (4) are solved to give

$$\begin{aligned} Q &= e+(x+y+zx+zy)^*zx^*y \\ &\quad +(x+y+zx+zy)^*zx^*x \\ &= e+(x+y+zx+zy)^*zx^*(x+y) \quad . \end{aligned}$$

Note that once again we obtain an expression for Q which has star-height strictly less than the rank of the factor graph.

7. Final Theorem

We have observed in the previous examples that the algorithm for determining the closure G_Q^* yields expressions for Q which are of star-height strictly less than the rank of the factor graph G_Q. The algorithm requires that one use an elimination method to determine certain closures A_{FF}^* and the closure G_H^* of a factor graph G_H which is minimal with respect to the ordering $\prec$. We shall now prove that, provided the order of elimination of nodes used in the determination of the various matrices A_{FF}^* and G_H^* is optimal with respect to the star-height of the resulting regular expressions, the algorithm always yields expressions for Q of star-height less than or equal to the rank of the factor graph G_Q of Q. The proof follows rather simply from the following theorem.

Theorem 7.1 Let Q be a regular language with factor graph G_Q, and let H be a factor of Q, with factor graph G_H. Then $\text{rank}(G_H) \leq \text{rank}(G_Q)$.

We shall in fact prove more than this, namely that for any factor H of Q there is some graph G'_H such that $G_H \subseteq G'_H \subseteq C^H_{max} + L^H_{max}$, and G_Q is pathwise homomorphic to G'_H.

Suppose that the graph G_Q has nodes N_Q, that $N_H \subseteq N_Q$ is the set of nodes of G_H, and $H = Q_{ij}$ where $i,j \in N_H$ (i.e. $i = s_H$ and $j = t_H$). The next lemma is a necessary preliminary to defining a mapping $\gamma: N_Q \to N_H$.

Lemma 7.2 Let $p \in N_Q$. Then $\exists$ a unique node $m_p \in N_H$ such that (i) $Q_{im_p} \cdot Q_{m_pj} \subseteq Q_{ij}$ dominates $Q_{ip} \cdot Q_{pj} \subseteq Q_{ij}$ and

(ii) if $m' \in N_H$ also has the property that

$Q_{im'} \cdot Q_{m'j} \subseteq Q_{ij}$ dominates $Q_{ip} \cdot Q_{pj} \subseteq Q_{ij}$

then

(a) $Q_{im_p} \supseteq Q_{im'}$ and (b) $Q_{m'm_p} \supseteq e$.

Proof Let $\{n_1, n_2, \ldots n_r\} \subseteq N_H$ be all those nodes in N_H such that $Q_{in_k} \cdot Q_{n_kj} \subseteq Q_{ij}$ dominates $Q_{ip} \cdot Q_{pj} \subseteq Q_{ij}$, $k = 1,2,\ldots r$. (Obviously the set is non-empty.)

Then $(Q_{in_1} + Q_{in_2} + \ldots + Q_{in_r}) \cdot (Q_{n_1j} \cap Q_{n_2j} \cap \ldots \cap Q_{n_rj}) \subseteq Q_{ij}$ dominates $Q_{ip} \cdot Q_{pj} \subseteq Q_{ij}$, and is itself dominated by $Q_{im_p} \cdot Q_{m_pj} \subseteq Q_{ij}$ for some $m_p \in N_F$. But m_p clearly satisfies (i) and (ii) (a). Part (ii) (b), follows directly from Theorem III 4.1(ii), since

$Q_{im'}$ and Q_{im_p} are both left factors of $H = Q_{ij}$ and $Q_{im_p} \geq Q_{im'} \cdot e$ by (ii) (a). Finally uniqueness of m_p follows from (ii) (b) and the acyclicity of $C^H_{max} \backslash E$ (Lemma III 5.4).

Now we may define a mapping $\gamma: N_Q \to N_H$ by: $\gamma(p) = m_p$ where m_p for any $p \varepsilon N_Q$ is the unique node of N_H defined by lemma 7.2 above.

We now extend γ to be a pathwise homomorphism in a rather trivial manner. We define the graph G'_H to have nodes N_H, and an arc labelled a from node k to node m $(k, m \varepsilon N_H)$ if and only if there is an arc a from some node $p \varepsilon \gamma^{-1}(k)$ to some node $r \varepsilon \gamma^{-1}(m)$ in the graph G_Q. Finally γ is extended to be a mapping from arcs of G_Q into arcs of G'_H by : if a labels an arc from node p to node r in G_Q then γ maps it into the arc labelled a from $\gamma(p)$ to $\gamma(r)$ in G'_H.

By the construction of G'_H, γ is a pathwise homormorphism and so we deduce from McNaughton's pathwise homormorphism theorem that:

Lemma 7.3 $\text{rank}(G^r_H) \leq \text{rank}(G_Q)$.

Lemma 7.5 will state that $G_H \trianglelefteq G'_H$, from which Theorem 7.1 follows immediately.. In order to prove this we prove the following lemma.

Lemma 7.4 Let p and r be any two nodes of G_Q and let $\gamma(p) = k$ and $\gamma(r) = m$. Then $Q_{km} \geq Q_{pr}$.

Proof Consider the subfactorization $Q_{ip} \cdot Q_{pj} \leq Q_{ij}$ which is dominated by $Q_{ik} \cdot Q_{kj} \leq Q_{ij}$ (by the definition of γ and lemma 7.2(i)).

Now $Q_{kj} \supseteq Q_{pj} \supseteq Q_{pr} \cdot Q_{rj}$.

Hence $\exists$ $m' \epsilon N_H$ such that

$$Q_{pr} \subseteq Q_{km'} \tag{1}$$

and

$$Q_{rj} \subseteq Q_{m'j} \cdot \tag{2}$$

But $Q_{rj} \subseteq Q_{m'j} \Rightarrow (Q_{ir}+Q_{im'}) \cdot Q_{rj} \subseteq Q_{ij}$.

Suppose this subfactorization is dominated by $Q_{im"} \cdot Q_{m"j} \subseteq Q_{ij}$ where $m" \epsilon N_H$. Then $Q_{im"} \supseteq Q_{im'}$ and as these are both left factors of $H = Q_{ij}$, by III4.1(ii),

$$Q_{m'm"} \supseteq e \ . \tag{3}$$

But, also, $Q_{im"} \cdot Q_{m"j} \subseteq Q_{ij}$ dominates $Q_{ir} \cdot Q_{rj} \subseteq Q_{ij}$.

$\therefore$, by lemma 7.2 (ii) (b),

$$Q_{m"m} \supseteq e \ . \tag{4}$$

Now, by (3) and (4), $Q_{m'm} \supseteq e$. (5)

Hence $Q_{km} \supseteq Q_{km'} \cdot Q_{m'm} \supseteq Q_{km'}$, by (5)

$$\supseteq Q_{pr}, \text{ by (1).}$$

Lemma 7.5 $G_H \subseteq G'_H$

Proof In order to prove this lemma, we first prove

(a) $G'_H \subseteq C^H_{max} + L^H_{max}$, and

(b) $G^*_H \subseteq G'^*_H$.

(a) Suppose $a \in [G'_H]_{km}$. Then by construction of G'_H, $\exists$ nodes p and r $\in N_Q$ such that $\gamma(p)=k, \gamma(r)=m$ $(k,m \epsilon N_H)$, and $a \in [G_Q]_{pr}$.

Thus, by lemma 7.4, $a \in Q_{km}$. Hence $a \in [C^H_{max} + L^H_{max}]_{km}$.

I.e. $G'_H \leq C^H_{max} + L^H_{max}$.

(b) Now let us prove $G^*_H \leq G'^*_H$. Since G^*_H is a submatrix of $\overline{|Q|} = G^*_Q$, a word $w \in [G^*_H]_{km}$ if and only if there is a sequence of nodes $k = p_0, p_1, \ldots p_n = m$ with each $p_r \in N_Q$ and such that $w = a_1 a_2 \ldots a_n$, where $a_r \in [G_Q]_{p_{r-1}p_r}$.

But then $a_r \in [G'_H]_{\gamma(p_{r-1})\gamma(p_r)}$, and $\gamma(p_0) = k$, and $\gamma(p_n) = m$.

I.e. $w \in [G'^*_H]_{km}$. Thus $G^*_H \leq G'^*_H$.

Now from (a) and (b) and Theorems III 5.6 and 5.2, $\overline{|H|} = G^*_H \leq G'^*_H \leq (C^H_{max} + L^H_{max})^* = \overline{|H|}$. Hence $G^*_H = G'^*_H = \overline{|H|}$. Finally, since G_H is the unique minimal starth root of $\overline{|H|}$ (Theorem III 5.6), $G_H \leq G'_H$ and the lemma is proved.

We may now deduce Theorem 7.1 directly from lemmas 7.3 and 7.5 since $G_H \leq G'_H$ trivially implies $\text{rank}(G_H) \leq \text{rank}(G'_H)$, which by lemma 7.3, $\leq \text{rank}(G_Q)$.

Corollary (to Theorem 7.1) With a suitable ordering of the nodes of the factor graph G_Q, algorithm 2 yields a regular expression for the language Q which is of star-height less than or equal to the rank of the factor graph G_Q.

Proof The algorithm requires that one determine G^*_H for a factor graph G_H which is minimal with respect to the ordering $\preccurlyeq$. By the above corollary the rank of G_H does not exceed the rank of G_Q and so with a suitable ordering of its nodes the escalator method yields regular expressions for the entries of

G_H^* all of which have star height not exceeding the rank of G_Q. Also required is that one determine A_{FF}^* for a number of graphs A_{FF}. Each such graph is a subgraph of a factor graph G_F and hence also has rank not exceeding the rank of G_Q. Thus once again one can obtain regular expressions for the entries in A_{FF}^* of star-height less than or equal to the rank of G_F. Since these are the only two cases where starred expressions are introduced by the algorithm the corollary holds.

As we have already demonstrated in examples 1, 2 and 3 the regular expressions obtained by the algorithm may well have star height <u>strictly</u> less than the rank of G_Q.

The proof of Theorem 7.1 is very useful in hand calculations to search for factor graphs of factors of Q which are separable from Q. The idea is to endeavour to eliminate nodes from the factor graph G_Q by "coalescing" them with other nodes of the factor graph. To eliminate node p by "coalescing" it with node k, one simply converts any arc a from some node i to node p into an arc a from node i to node k, and any arc a from node p to some node j into an arc a from node k to node j. Suppose after eliminating a number of nodes from G_Q, this results in a graph G' having nodes $N' \subseteq N_Q$.

The next step is to "prune off" arcs of G', i.e. in effect construct $G = (G'\backslash E)\backslash(G'\backslash E)^{2+*}$. This graph G will then be a factor graph only if $G \subseteq (C^Q_{max}+L^Q_{max})$, which can easily be checked by inspection. Note, however, that G is not necessarily a factor graph of Q (see example 7, section 8), although all factor graphs can be obtained in this way.

Thus the method is not fail-safe and ultimately resort must be made to Algorithm 3. Nevertheless it is undoubtedly a useful aid to rough calculations.

Example 3 illustrates the process quite well. In order to obtain the factor graph of Q_{23} one eliminates node 1 by coalescing it with node 2, the graph for Q_{14} is obtained by coalescing node 3 with node 4, the graph for Q_{24} by coalescing node 3 with node 4 and node 1 with node 2, and so on.

8. Empirical Results

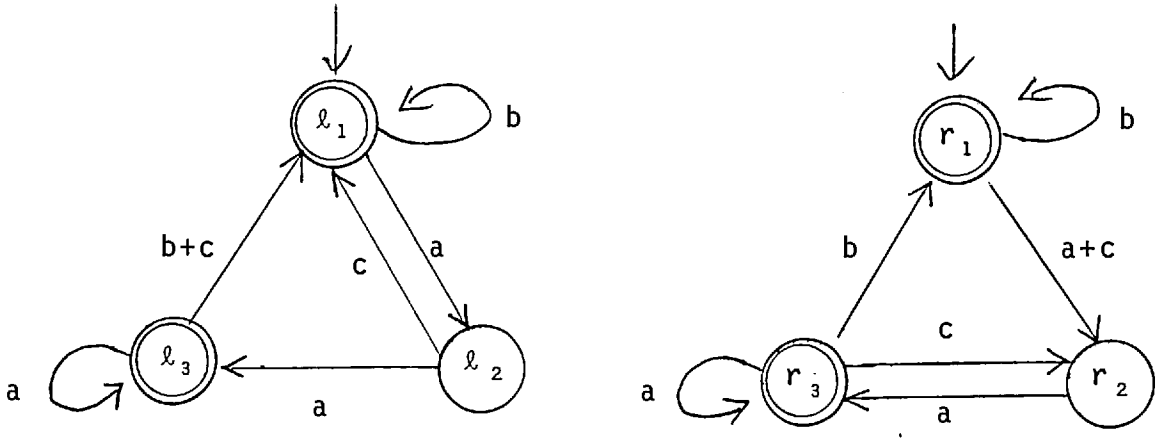
We would have liked, of course, to end this chapter with a theorem to the effect that algorithm 2 always yields a minimal star-height expression for the language Q. Indeed for a long time we thought that this could well be true. Just by taking a very large selection of regular languages which have appeared in the literature, and laboriously calculating factor graphs we achieved almost 100% success in arriving at minimal star-height forms for these languages.

If algorithm 2 did always yield a minimal star-height expression for any regular language Q, a necessary condition would be that, for those languages Q all of whose factors are inseparable from Q, the star-height of Q would equal the rank of the factor graph of Q. In our empirical investigation we eventually found an example which appeared in McNaughton's paper [29] which refuted this condition, thus showing that algorithm 2 does not always give the minimal star-height

expression for a language Q. The example follows.

Example 4 (Refutation of conjecture that algorithm 2 always yields a minimal star-height expression.)

Consider Q = (b + aa + ac + aaa + aac)*. The machine and anti-machine for this language are shown below.



Machine Anti-machine

Fig. 9

If we use algorithm 1 we find that

$$D_{\ell_1}Q = r_1 + r_3$$

$$D_{\ell_2}Q = r_2 + r_3$$

and $$D_{\ell_3}Q = r_1 + r_2 + r_3 .$$

Thus the L.R factorizations of Q are

$$\begin{aligned} L_1 \times R_1 &= (\ell_1 + \ell_2 + \ell_3) \times r_3 \\ L_t = L_2 \times R_2 = R_s &= (\ell_1 + \ell_3) \times (r_1 + r_3) \\ L_3 \times R_3 &= (\ell_2 + \ell_3) \times (r_2 + r_3) \\ L_4 \times R_4 &= \ell_3 \times (r_1 + r_2 + r_3) \quad . \end{aligned}$$

The factor graph of Q can now be determined:

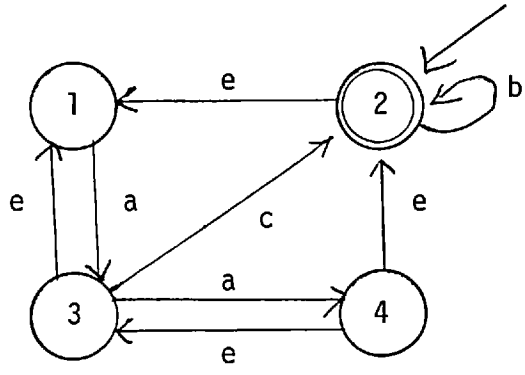


Fig. 10

In order to determine whether Q has any factors whose factor matrix is a submatrix of $\overline{|Q|}$ we calculate the semigroup of Q. The semigroup machine is shown in the next diagram in which nodes, or equivalently c-classes of Q, are labelled by a representative element of the corresponding c-class.

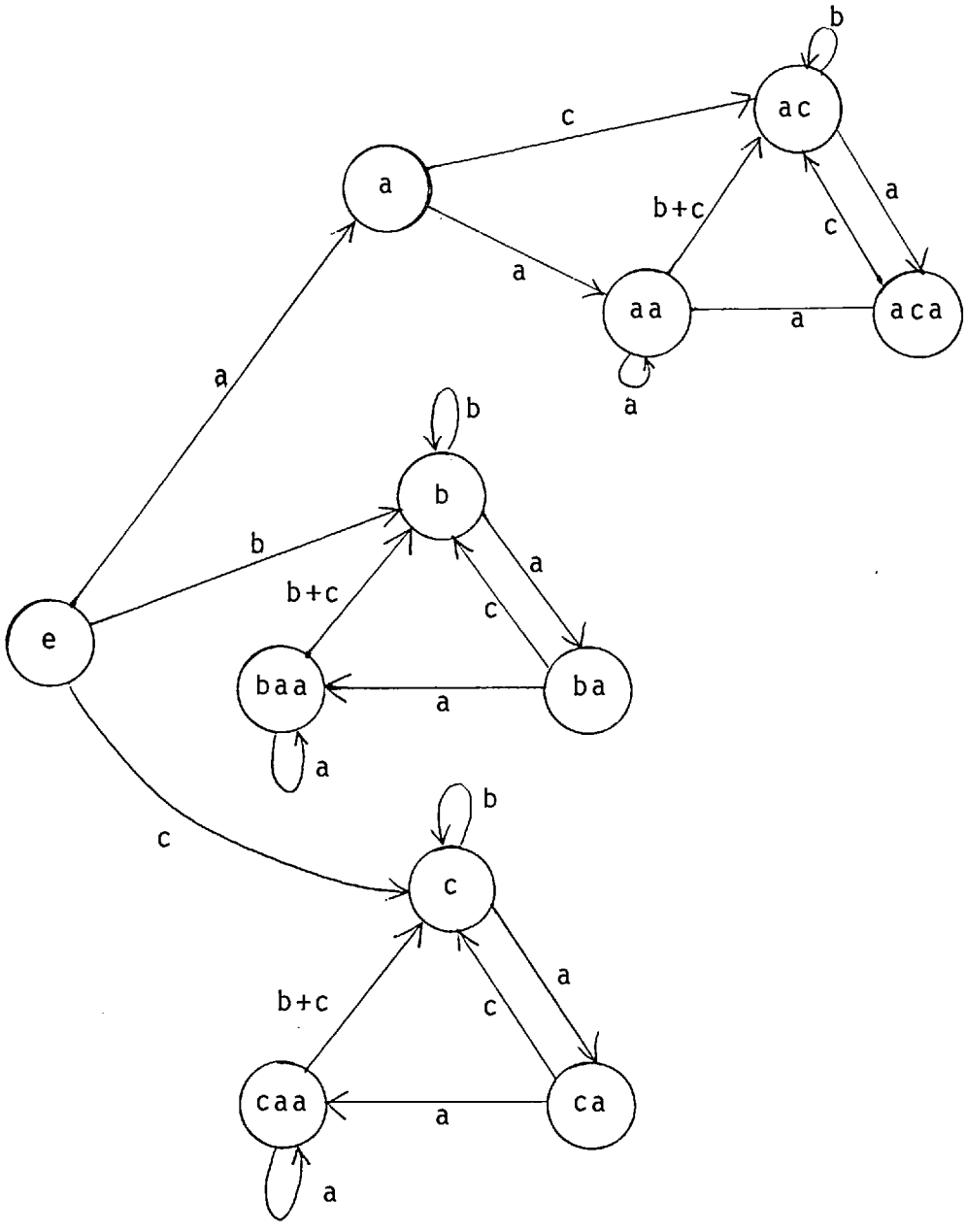


Fig. 11

Using the semigroup multiplication we can now determine each entry in $G^*_{\overline{Q}} = \overline{|Q|}$ as a union of c-classes of Q. Thus we get:

$$\overline{|Q|} = \begin{bmatrix}
e+a+aa & ac+aa & a+aa & aa \\
+ac+aca & & +aca & \\
\\
e+a+aa & e+aa & a+aa & aa \\
+ac+aca & +ac & +aca & \\
+b+ba+baa & +b+baa & +ba+baa & +baa \\
\\
e+a+aa & a+aa & e+a+aa & a+aa \\
+ac+aca & +ac & +aca & \\
+c+ca+caa & +c+caa & +ca+caa & +caa \\
\\
e+a+aa & e+a+aa & e+a+aa & e+a+aa \\
+ac+aca & +ac & +aca & \\
+b+ba+baa & +b+baa & +ba+baa & +baa \\
+c+ca+caa & +c+caa & +ca+caa & +caa
\end{bmatrix}$$

Finally by inspection of the above matrix we can verify that there are no indices i,j,p and k such that

$$Q_{ip} \subseteq Q_{ik} \quad \text{and} \quad Q_{pj} \subseteq Q_{kj} \ ,$$

and hence all factors are inseparable from Q.

The rank of G_Q is two whereas we already have an expression for Q which has star-height one. Thus for this example algorithm 2 fails to give a minimal star-height expression.

We conclude this section with a brief discussion of some of the "more interesting" examples studied by other authors.

Example 5 This example appears as example 6.5 in the paper by Cohen and Brzozowski [12].

Consider the language Q defined by the following machine.

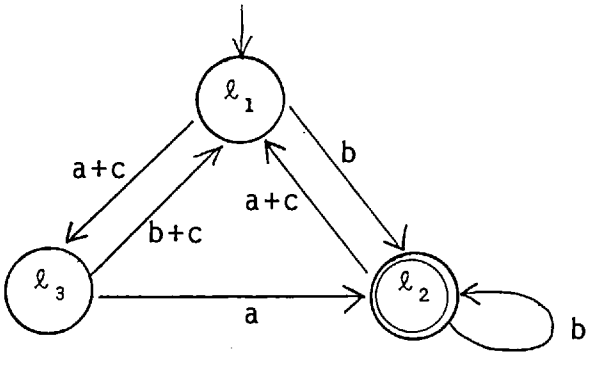


Fig. 12 Machine

Its anti-machine is given in the next diagram, following which we show a sequence of factor graphs of the language Q and some of its factors.

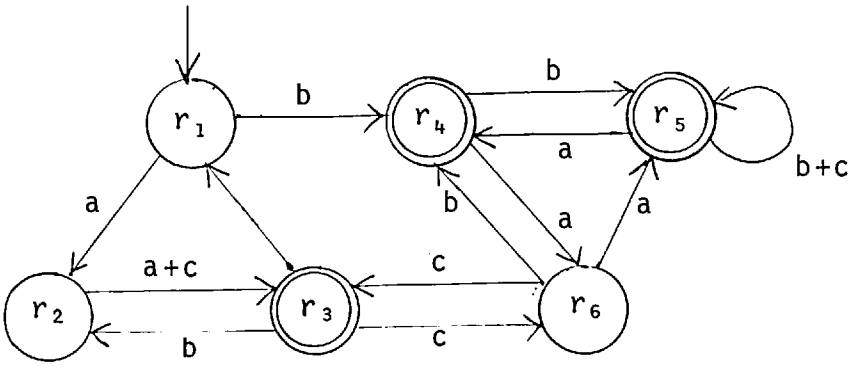


Fig. 13 Anti-machine

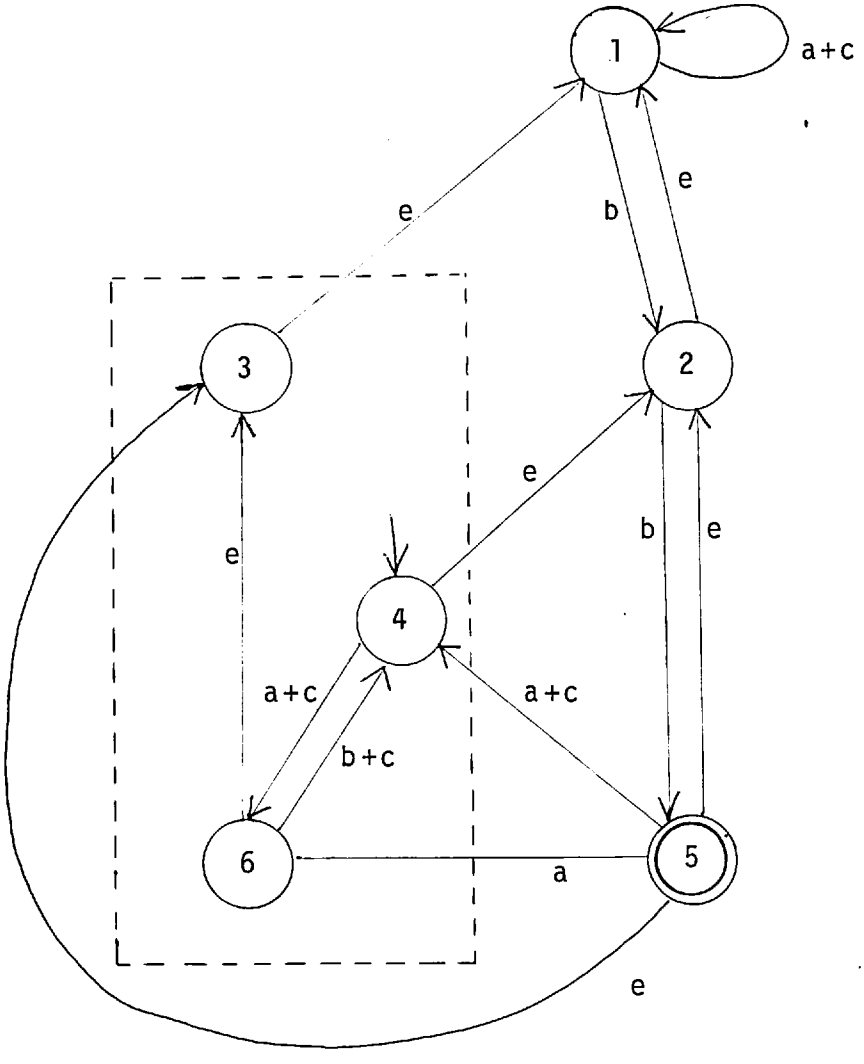


Fig. 14 Factor Graph of $Q = Q_{45}$

Matrix indicated by dotted lines = A, say.

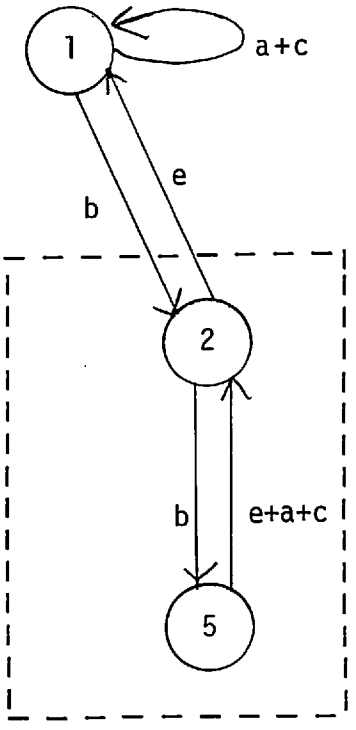


Fig. 15 Factor Graph of Q_{22}

Matrix indicated by
dotted lines = B, say.

Fig. 16 Factor Graph of Q_{11}

If we apply algorithm 2 to the above sequence of factor graphs we would have to calculate Q_{11} (which is trivially (a+b+c)* - see fig. 16) and the closures B* and A*, where we have indicated the matrices A and B by dotted lines (figs. 14 and 15). Both of these matrices have rank 1 and so we deduce that Q has star-height 1.

This example was introduced by Cohen and Brzozowski to illustrate the difficulties inherent in a method they propose for finding the star-height of a regular language, their method being an enumerative one viz. calculate all subgraphs of a sequence of graphs of a specific type and determine whether each such graph is a recogniser for Q. In contrast using factor theory we are able to determine the star-height of this language directly, without any enumeration being involved.

Note also that neither the machine nor anti-machine have rank one - astute readers may have criticised our earlier examples on this point.

Example 6 (McNaughton [29],p314). The language

$$Q = \{x^*(x + z)\ x^*(y + z)\}^*$$

was proved by McNaughton to have star-height 2. His proof technique is to consider subfactors of a language and show that the complexity of their interconnections in any recogniser of Q must be above some value. In this case let G be any recogniser of Q, and consider those nodes N of G such that a word w in the subfactor x^*zyz takes node N to some node N' in G. Let A be the set of all such nodes. Consider also the set B of all nodes M of G such that a word v in the subfactor $zyzx^*$ takes some node M' to node M in G. Then one easily proves that the sets A and B are disjoint but are strongly connected, and that they each define some strongly connected component of G of rank at least one. This then allows one to

prove that the language has star-height at least two.

Examination of the machine or anti-machine for this language yields no insight which would lead to the above determination of the language's star-height. However if we study the factor graph (shown below - fig. 17)

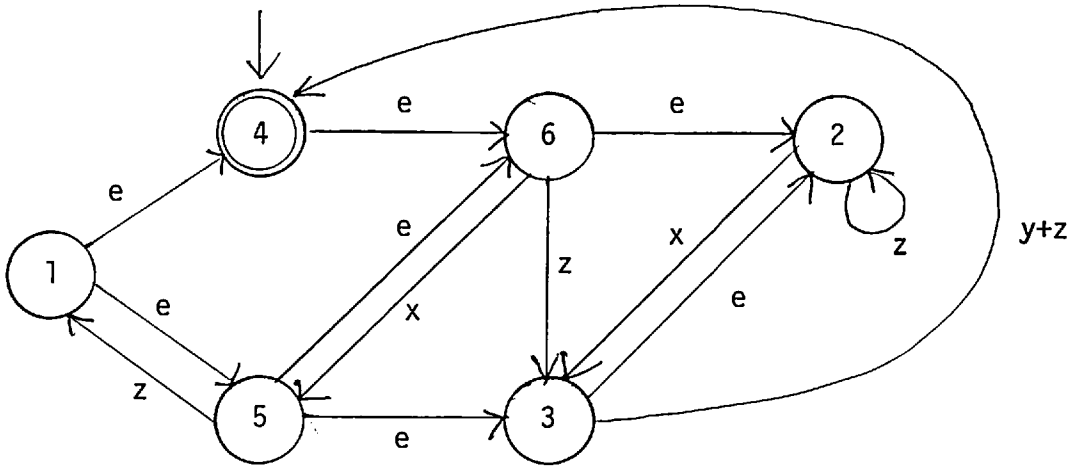


Fig. 17

we notice that we can eliminate nodes 1 and 4 to obtain the following factor graph in which all factors are inseparable.

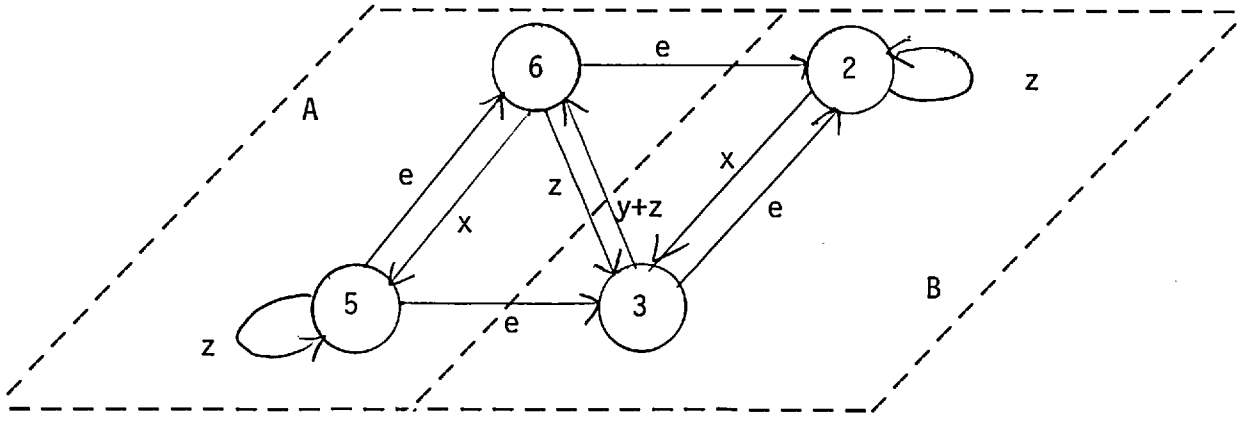


Fig. 18

A and B define two (almost) symmetrical halves of this graph (see fig. 18), but are distinguished by the asymmetry of the graph.

Example 7 Let Q = $(a + (a + b)c^*(c + d))^*$.

This example was also proved by McNaughton [29,p315] to have star-height two. The above expression is identical to what one would obtain from the factor graph (fig. 19).

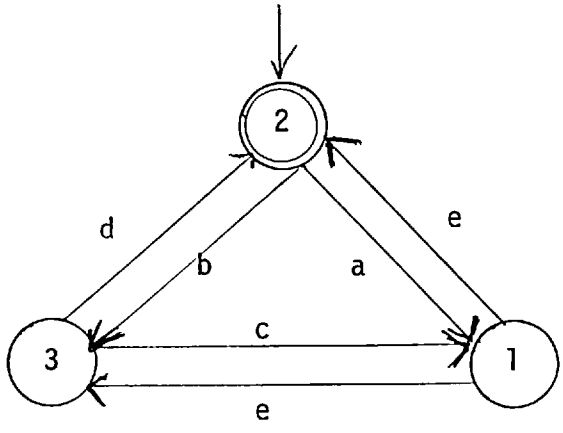


Fig. 19

All factors of Q are inseparable from Q. Note that the factor graph is pathwise homomorphic to the following graph, but which is not a factor graph for any language.

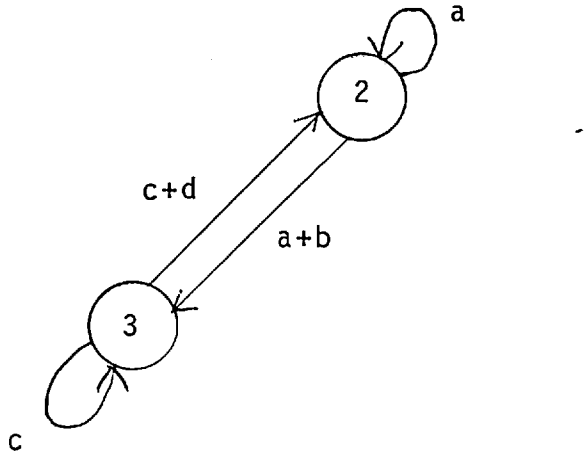
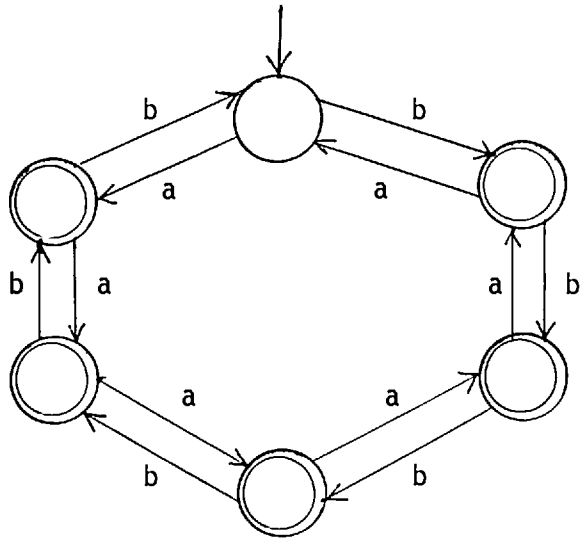


Fig. 20

CONCLUSIONS

In his book [13,p123] Conway advises readers to avoid calculating the factor matrix of any language Q. While disagreeing with this advice, for the simple reason that one cannot expect to make any advances in the theory of factors without doing such calculations, we should nevertheless mention two drawbacks to any practical use of factor theory.

The first is that calculations with factors inevitably tend to be rather long. With practice the method given in Chapter III to calculate the factor matrix $|\overline{Q}|$ is quite simple and straightforward, most of the effort in fact going into calculating the machine and anti-machine for Q. But, by hand, the work is tedious and rather prone to error, as well as involving quite a large amount of computation. Moreover, as soon as one begins calculating factor graphs of factors, the effort involved really does become quite daunting. (An obvious case for programming on a computer!) The second and much more significant reason, is that the factor graph may well have an extremely large number of nodes, even for languages having quite simple finite-state machines. For instance the language recognised by the finite-state machine below has a factor graph having 64 ($=2^6$) nodes.



Indeed, for any $n \geq 3$, the subset of (a+b)* consisting of all words such that the number of a's minus the number of b's is not congruent to 0 mod n has 2^n nodes in its factor graph (the above example is the case n = 6). Also if the language Q has a factor graph of a manageable size, the factor graph of $\sim Q$ will often be quite unmanageable. However these problems are not peculiar to factor theory, and would appear to be a fact of life when handling regular languages. If anything, they illustrate just how much we don't know about regular languages!

In retrospect, the algorithm given in chapter IV for calculating the closure G_Q^* of G_Q is based on very simple ideas. The essential ingredients of the algorithm are the following:

(a) there is a transitive relation "is a factor of" on the entries in G_Q^* .

(b) the relation "is a factor of" can be reduced to an equivalence relation "is inseparable from" on the entries in G_Q^* .

(c) each equivalence class modulo "inseparability" defines a graph G_H such that G_H^* is a submatrix of G_Q^* .

By way of comparison, consider any graph G and consider the following sequence of ideas.

(a)' there is a transitive relation "is connected to" on the nodes of the graph G.

(b)' the relation "is connected to" can be reduced to the equivalence relation "is strongly connected to" on nodes of G.

(c)' each equivalence class modulo "is strongly connected to" defines a subgraph of the graph G (in fact a section of G).

The process of deducing an algorithm to calculate G_Q^* is thus a very familiar one, and one inevitably seeks to extend it to other recognisers of Q. For example entries in M*, where M is the machine of Q, are derivatives of Q and derivatives of derivatives are derivatives. Thus "is a derivative of" is a transitive relation on the derivatives of Q. Unfortunately in this case no additional benefit is gained by considering a symmetric closure of "is a derivative of" since if S and T are derivatives of Q, S is a derivative of T and T is a derivative of S if and only if the two nodes to which they correspond (cf Theorem III 2.2) are in the same

section of the machine of Q. Similarly analysis of the anti-machine or the semigroup machine yields no improvement on the usual elimination methods for finding their closure.

One possibility remains. We have observed that factors of a language Q are unions of c-classes of Q which are maximal in some subfactorization of Q. Instead of considering maximal unions of c-classes we could consider arbitrary unions of c-classes. Using similar techniques to those given in Chapter III, we could construct "c-class graphs" for Q. In a c-class graph, G, entries in G* are unions of c-classes of Q and each node can be labelled $(\ell_{i_1}+\ldots+\ell_{i_k}) \times (r_{j_1}+\ldots+r_{j_m})$ where the ℓ_i's are ℓ-classes and the r_j's are r-classes of Q. The only requirement is that

$$(\ell_{i_1}+\ldots+\ell_{i_k})\cdot(r_{j_1}+\ldots+r_{j_m}) \subseteq Q$$

is a subfactorization of Q. Using some results of McNaughton and Papert [31], it is very easy to prove that unions of c-classes of unions of c-classes of Q are themselves unions of c-classes of Q. We thus have the beginnings of an analysis similar to the one given here for the factor graph. Moreover, in studying c-class graphs one can benefit much more from previous work on the star-height problem than we have been able to. For the "pure-group events" studied by McNaughton [28] a c-class graph is clearly what he would call a "μ-graph". Cohen and Brzozowski [12] also study "subset automata" which are particular cases of c-class graphs.

Very severe practical problems arise however, as there is usually not just one, but many c-class graphs for a particular language Q (one of which is the factor graph). Problems which we mentioned above in studying factor graphs are thus accentuated tremendously. Needless to say, we have no empirical results to date as to how successful this could be, and there is still a lot to be done before the star-height problem is solved.

An intermediate problem which is probably worth tackling before embarking on an investigation of c-class graphs is the following. Let F and H be two regular languages, and suppose F is a union of c-classes of H and H is a union of c-classes of F. Is the star-height of F equal to the star-height of H? An answer of no to this question would make us very sceptical of pursuing this line of investigation, but we would guess that the answer is more likely to be yes.

It is important to note that algorithm 2 cannot have an analogue in linear algebra. Considering yet again example 1; from the factor graph of the language Q (fig. 1,p.123) we get the following system of equations defining Q:

$$Q = (e+a)P + e$$

$$P = bQ + eT$$

$$T = aT + bP \quad ,$$

where $P = Q_{21}$ and $T = Q_{31}$. Substituting $e = 1$, $a = -\frac{1}{2}$, $b = 1$ in these equations and solving (in linear algebra) for Q, we get

$$Q = -2 \quad .$$

However if we replace m* by $(1-m)^{-1}$ and substitute the same values for e,a and b in

$$Q = (b+ab)^*(e+a)(a+b)^*b(b+ba)^*b + (b+ab)^*$$

(IV § 4) we get

$$Q = 6 \ .$$

If the method did have an analogue in linear algebra, the two values would agree, which they evidently do not.

The main conclusion to be drawn from the work presented here is that one should not be content with the simple elimination methods of Chapter II, and more effort should be put into developing new closure algorithms. The approach we would suggest for doing this would be the same as that used here. That is, investigate one particular class of graphs, develop closure algorithms using properties particular to this class and finally seek to extend the algorithms to have more general applicability. Moreover, let us not see any hypothetical solutions to the star-height problem which simply involve "enumeration" until all other possible hypotheses have definitely been exhausted.

We have had little to say in this thesis about other applications of Conway's factor theory, but we would anticipate that it will become much more important as a theoretical tool in the study of regular languages. Conway introduces it as a method of studying approximations to regular languages, and goes on to use it extensively in his study of the biregulators. (Note: biregulators are usually called "generalised sequential machines".) Some connections between factors and the semigroup

of a language have been pointed out, and we feel that the direction of future research on factors would be to investigate such connections more fully. Note that it is always possible to discover the semigroup multiplication directly from the matrix $C^Q_{max} + L^Q_{max}$, and so in studying the factor matrix one loses no information about the semigroup. Indeed much more information is contained in the factor matrix. The semigroup is often too coarse a description of a language, since it is invariant under relabelling of the start and terminal nodes of the machine. However under such relabelling the factor matrix changes quite substantially. For these reasons we would expect factor theory to yield more insight into those problems which have traditionally been tackled by studying the semigroup of the language [30].

But these are just a few suggestions for further work on regular languages. The study of regular languages is a particularly attractive area for further research since it offers quite a large number of unsolved or inadequately solved problems. (See [34] for further discussion.) Moreover problems in regular algebra are usually expected to be solvable - but they are clearly very difficult and very challenging.

REFERENCES

1. Aho, A.V. and Ullman, J.D.

"The Theory of Parsing, Translation and Compiling", vol.I.

Prentice Hall, Englewood Cliffs, N.J. 1972.

2. Backhouse, R.C. and Carré, B.A.

"Regular algebra applied to path-finding problems"

J. Inst. Maths. Applics. 15 (1975) pp.161-186.

3. Brzozowski, J.A.

"Derivatives of regular expressions"

J.A.C.M. 11, 4 (Oct. 1964) pp.481-494.

4. Brzozowski, J.A.

"Roots of star events"

J.A.C.M. 14, 3 (July 1967) pp.466-477.

5. Bunch, J.R.

"Complexity of sparse elimination"

In

"Complexity of sequential and parallel numerical algorithms"

Traub, J.F. (Ed.), Academic Press, New York and London (1973).

6. Carré, B.A.

"An algebra for network routing problems"

J. Inst. Maths. Applics. 7 (1971), pp.273-294.

7. Carré, B.A.

"A matrix factorization method for finding optimal paths through networks"

I.E.E. Conf. Publ. No.51 (Computer Aided Design) 1969, pp.388-397.

8. Carré, B.A.

"An elimination method for minimal-cost network flow problems"

In

"Large sparse sets of linear equns."

(Proc. IMA Conf., Oxford, 1970) Ed. J.K. Reid, Academic Press, London.

9. Cohen, R.S.

"Rank non-increasing transformations on transition graphs"

Inf. and Control 20 (1972), pp.93-113.

10. Cohen, R.S.

"Star-height of certain families of regular events"

J. Comp. Syst. Scs. 4, 3 (1970), pp.281-297.

11. Cohen, R.S.

"Techniques for establishing star-height of regular sets"

Math. Systs. Th. 5, 2 (1971) pp.97-114.

12. Cohen, R.S. and Brzozowski, J.A.

"General properties of star-height of regular events"

J. Comp. Syst. Scs. 4, 3 (1970), pp.260-280.

13. Conway, J.H.

"Regular algebra and finite machines"

Chapman and Hall, London 1971.

14. Dantzig, G.B.

"All shortest routes in a graph"

Theory of Graphs (International Symposium, Rome 1966), Gordon and Breach, New York, pp.91-92.

15. Dejean, F. and Schützenberger, M.P.

"On a question of Eggan"

Info. and Control 9, (1966) pp.23-25.

16. Dijkstra, E.W.

"A note on two problems in connexion with graphs"

Numerische Mathematik, 1 (1959), pp.269-271.

17. Dreyfus, S.E.

"An appraisal of some shortest path algorithms"

Operat. Res. 17, 3 (May 1969), pp.395-412.

18. Eggan, L.C.

"Transition graphs and the star-height of regular events"

Michigan Math. J. 10 (1964) pp.385-397.

19. Floyd, R.W.

"Algorithm 97, shortest·path"

Comm. Assoc. Comp. Mach. 5, 6 (June 1962).

20. Fontan, G.

"Sur les performances d'algorithmes de recherche de chemins minimaux dans les graphes clairsemés"

Rep. Lab. d'Auto. et d'Analyse des Systèmes, Toulouse (1974).

21. Fox, L.

"An introduction to numerical linear algebra"

Clarendon Press, Oxford (1964).

22. Ginzburg, A.

"Algebraic theory of automata"

Academic Press, New York 1968.

23. Grassin, J. and Minoux, M.

"Variations sur un algorithme de Dantzig avec application à la recherche des plus courts chemins dans les grands reseaux"

Rev. Fr. d'Auto. Info. & Rech. Oper. 1 (March 1973).

24. Hartmanis, J. and Stearns, R.E.

"Algebraic structure theory of sequential machines"

Prentice Hall, Englewood Cliffs, N.J. (1966).

25. Hoffman, A.J. and Winograd, S.

"Finding all shortest distances in a network"

IBM J. Res. & Devel. 16, 4 (July 1972), pp.412-414.

26. Householder, A.S.

"Principles of numerical analysis"

McGraw Hill, New York 1953.

27. Johnson, D.B.

"Algorithms for shortest paths"

Technical Report 73-169, Dept. of Computer Science, Cornell University (May 1973).

28. McNaughton, R.

"The loop complexity of pure-group events"

Info. and Control, 11 (1967), pp.167-176.

29. McNaughton, R.

"The loop complexity of regular events"

Infor. Sciences 1 (1969), pp.305-328.

30. McNaughton, R. and Papert, S.

"Counter-free automata"

Res. Monograph no.65, M.I.T. Press, Cambridge, Mass. and London.

31. McNaughton, R. and Papert, S.

"The syntactic monoid of a regular event"

In Arbib, M.A. (Ed.)

"Algebraic theory of machines, languages and semigroups"

Academic Press, New York (1968).

32. Munro, I.

"Efficient determination of the transitive closure of a directed graph"

Info. Proc. Letters (1971) pp.56-58, North Holland Publ. Co.

33. Murchland, J.D.

London School of Economics Report LSE-TNT-26 (1967).

34. Rabin, M.O.

"Mathematical theory of automata"

In J.J.T. Schwartz (Ed.)

"Mathematical aspects of computer science"

Amer. Math. Soc. Proc. in Applied Maths. 19, (1967).

35. Rabin, M.O. and Scott, D.

"Finite automata and their decision problems"

IBM J. Res. Dev. 3, 114-125 (1959).

36. Rodionov, V.V.

USSR Comp. Math. and Math. Phys. 8, (1968), pp.336-343.

37. Rose, D.J. and Bunch, J.R.

"The role of partitioning in the numerical solution of sparse systems"

In

"Sparse matrices and their applications"

Rose, D.J., Willoughby, R.A. (Eds.) Plenum Press, New York and London (1972), pp.177-187.

38. Salomaa, A.

"Two complete axiom systems for the algebra of regular events"

J.A.C.M. 13, 1 (Jan. 1966), pp.158-169.

39. Salomaa, A.

"Theory of automata"

Pergamon Press, Oxford (1969).

40. Scott, D.

"The lattice of flow diagrams"

In

"Symposium on semantics of algorithmic languages"

Ed. E. Engeler: Springer-Verlag, Berlin (1971).

41. Strassen, V.

"Gaussian elimination is not optimal"

Numer. Math. 13 (1969), pp.354-356.

42. Tewarson, R.P.

"Sparse matrices"

Academic Press, New York and London (1973).

43. Warshall, S.

"A theorem on boolean matrices"

J.A.C.M., 9 (1962), pp.11-12.

44. Yen, J.Y.

"Finding the lengths of all shortest paths in N-node non-negative - distance complete networks using $\frac{1}{2}N^3$ additions and N^3 comparisons"

J.A.C.M., 19, 3 (July 1972), pp.423-424.

45. Yen, J.Y.

"An algorithm for finding shortest routes from all source nodes to a given destination in general networks"

Quart. Appl. Maths. 27 (1970), pp.526-530.

APPENDIX A - PROOF THAT $M_p(R)$ IS A REGULAR ALGEBRA

Let R be a regular algebra, and consider all p×p matrices with entries in R. Let this set be denoted by $M_p(R)$. In $M_p(R)$ define operators $\cdot$ and + by:

If $A = [a_{ij}]$, $B = [b_{ij}]$

$$A \cdot B = \left[\sum_{k=1}^{p} a_{ik} b_{kj} \right], \qquad A+B = [a_{ij} + b_{ij}] .$$

We define A* by induction on p. For p=1 A* is already defined, since $M_1(R)$ may be identified with R. For p>1 split A into four matrices

$$A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

where A_{11} is 1×1, A_{21} is (p-1)×1, A_{12} is 1×(p-1) and A_{22} is (p-1)×(p-1).

Define

$$A^* = \begin{bmatrix} A_{11}^* + A_{11}^* A_{12} C_{22} A_{21} A_{11}^* & A_{11}^* A_{12} C_{22} \\ C_{22} A_{21} A_{11}^* & C_{22} \end{bmatrix}$$

where $C_{22} = (A_{22} + A_{21} A_{11}^* A_{12})^*$.

We shall now prove that, with the above definitions of +, $\cdot$ and *, $M_p(R)$ is a regular algebra.

In fact this is quite simple to do. Axioms A1-A9 are straightforward. A10 and A11 are also trivially verified by induction on p. The only non-trivial aspect of the proof is to show that the condition for uniqueness of solution of equations carries over to matrices.

To prove this we proceed by induction on p. For

p=1, $M_1(R)$ can be identified with R and so is, by assumption, a regular algebra. Consider then a square matrix A of order p>1, and suppose $M_{p-1}(R)$ is a regular algebra (and hence the condition for uniqueness of solutions of equations holds in $M_{p-1}(R)$). Let n = p-1 .

Split A into four submatrices:

$$A = \left[\begin{array}{c|c} A_{11} & A_{12} \\ \hline A_{21} & A_{22} \end{array}\right]$$

where A_{11} and A_{22} are square matrices of orders 1 and n respectively. Let $N^{(q)}$ denote the null matrix of order q.

We now prove an alternative condition for the definiteness of A in terms of A_{11}, A_{12}, A_{21} and A_{22}, and then use this alternative condition in lemma 2 to prove uniqueness of solution of equations if A is definite.

Lemma 1 A is definite $\Rightarrow A_{11}$ and $A_{22} + A_{21}A_{11}^{*}A_{12}$ are definite.

Proof

(a) Assume A_{11} is not definite. Then, by definition, $\exists\ T_{11}$ such that $T_{11} \subseteq A_{11}T_{11}$ and $T_{11} \neq \phi$.

Take

$$T = \left[\begin{array}{c|c} T_{11} & \phi \\ \hline \phi & N^{(n)} \end{array}\right]$$

Then $T \subseteq AT$ and so A is not definite.

(b) Assume $A_{22} + A_{21}A_{11}^{*}A_{12}$ is not definite. Then $\exists\ T_{22} \neq N^{(n)}$ such that $T_{22} \subseteq (A_{22} + A_{21}A_{11}^{*}A_{12})T_{22}$.

Take

$$T = \left[\begin{array}{c|c} \phi & A_{11}^* A_{12} T_{22} \\ \hline \phi & T_{22} \end{array}\right]$$

Then

$$A.T = \left[\begin{array}{c|c} \phi & A_{11}A_{11}^* A_{12}T_{22} + A_{12}T_{22} \\ \hline \phi & A_{21}A_{11}^* A_{12}T_{22} + A_{22}T_{22} \end{array}\right]$$

$$= \left[\begin{array}{c|c} \phi & A_{11}^* A_{12} T_{22} \\ \hline \phi & (A_{22} + A_{21}A_{11}^* A_{12})\, T_{22} \end{array}\right]$$

$$\geq T .$$

Hence A is not definite.

(a) and (b) together imply A_{11} or $A_{22} + A_{21}A_{11}^* A_{12}$ not definite $\Rightarrow$ A is not definite. The contrapositive of this is the lemma.

Consider now the equation $Y = A \cdot Y + B$, and split Y and B into submatrices Y_{11}, Y_{12}, Y_{21} and Y_{22} and B_{11}, B_{12}, B_{21} and B_{22} exactly as in A.

Lemma 2 The equation $Y = AY + B$ has solution $Y \geq A^*B$ with equality if A_{11} and $A_{22} + A_{21}A_{11}^* A_{12}$ are both definite.

Proof Multiplying out the equation $Y = A \cdot Y + B$, we get four equations:

$$Y_{11} = A_{11}Y_{11} + A_{12}Y_{21} + B_{11} \qquad (1)$$

$$Y_{12} = A_{11}Y_{12} + A_{12}Y_{22} + B_{12} \qquad (2)$$

$$Y_{21} = A_{21}Y_{11} + A_{22}Y_{21} + B_{21} \qquad (3)$$

$$Y_{22} = A_{21}Y_{12} + A_{22}Y_{22} + B_{22} \qquad (4)$$

Now A_{11} has order 1, so we may apply R1 to (1) and (2) giving

$$Y_{11} \geq A_{11}^* A_{12} Y_{21} + A_{11}^* B_{11} \tag{5}$$

$$Y_{12} \geq A_{11}^* A_{12} Y_{22} + A_{11}^* B_{12} \tag{6}$$

with equality if A_{11} is definite.

(5) and (6) are now substituted in (3) and (4), giving:

$$Y_{21} \geq (A_{22} + A_{21} A_{11}^* A_{12}) Y_{21} + B_{21} + A_{21} A_{11}^* B_{11} \tag{7}$$

$$Y_{22} \geq (A_{22} + A_{21} A_{11}^* A_{12}) Y_{22} + B_{22} + A_{21} A_{11}^* B_{12} \tag{8}$$

$A_{22} + A_{21} A_{11}^* A_{12}$ has order p-1, so by the induction hypothesis we can apply R1, giving

$$Y_{21} \geq (A_{22} + A_{21} A_{11}^* A_{12})^* (B_{21} + A_{21} A_{11}^* B_{11}) \tag{9}$$

$$Y_{22} \geq (A_{22} + A_{21} A_{11}^* A_{12})^* (B_{22} + A_{21} A_{11}^* B_{12}) \tag{10}$$

with equality if both A_{11} is definite and $A_{22} + A_{21} A_{11}^* A_{12}$ is definite.

Finally (9) and (10) are substituted in (5) and (6), giving

$$Y_{11} \geq A_{11}^* A_{12} C_{22} (B_{21} + A_{21} A_{11}^* B_{11}) + A_{11}^* B_{11} \tag{11}$$

$$Y_{12} \geq A_{11}^* A_{12} C_{22} (B_{22} + A_{21} A_{11}^* B_{12}) + A_{11}^* B_{12} \tag{12}$$

with the same condition for equality.

Combining (9),(10),(11) and (12), the definition of A* and lemma 1, we clearly get the rule of inference R1 for $M_p(R)$. So by induction $M_p(R)$ is a regular algebra for all p.

APPENDIX B - INFORMAL PROOF OF THEOREM B4

Here we shall demonstrate that the algorithms for finding G* which we have loosely called "elimination methods" invariably result in regular expressions having star-height at least equal to the rank of G. We begin by abstracting the salient features of an "elimination method" and then show how any elimination method defines an "order of elimination of arcs". We then investigate in detail the effect of eliminating arcs of the graph, and finally show how we can build an "analysis" of the graph from the order of elimination of the arcs.

Before doing so we need some definitions and a fundamental theorem from McNaughton [29]. By an analysis of a graph G we shall mean a partial ordering of ordered pairs $\langle N,G_1\rangle$ where N is a node of G_1, a strongly connected component of G, having the following properties:

1. For each section S_i of G there is a node N of S_i such that $\langle N,S_i\rangle$ is maximal in the partial ordering.
2. For no subgraph G_1 are there two nodes N,N' such that both $\langle N,G_1\rangle$ and $\langle N',G_1\rangle$ occur in the partial ordering.
3. If $\langle N,G_1\rangle$ occurs and G_2 is a section of that subgraph that has all nodes of G_1 except N then, for some N' of G_2, $\langle N',G_2\rangle$ is an immediate inferior of $\langle N,G_1\rangle$.
4. All of the immediate inferiors of $\langle N,G_1\rangle$ are of the kind mentioned in 3. (and hence $\langle N,G_1\rangle$ is minimal if all loops of G_1 contain N).

An analysis of a graph will always be a forest of as many trees as the graph has sections.

The height of an analysis is the length of the maximal length chain in the partial ordering.

Theorem B1 (McNaughton [29]). The rank of a graph is the minimum height of all analyses of the graph.

Our aim is to show how an "elimination" method defines an analysis of G such that the height of the analysis is less than or equal to the maximum star-height of regular expressions of G*. Firstly we state more precisely what we mean by an elimination method.

Definition B2 An elementary matrix is a matrix whose non-null elements all lie in the same row, or in the same column.

Note that it is "elementary" to find the closure of a row or column matrix using II (3.1) or (3.2) - hence the definition.

Definition B3 An elementary elimination step is a step in an algorithm which involves solely the computation of the closure of an elementary matrix.

We can abstract three essential features of the methods of Chapter II.

1. The tautologies II (2.3) and (2.4) are applied exclusively to derive an expression for A* as a product

$$A^* = J_1^* J_2^* \ldots J_m^* \qquad (1.1)$$

of elementary matrices.

If at some stage in the derivation of (1.1) the matrix B, say, is split into matrices C and D such that B=C+D, and either (2.3) is used to express B* as

$$B^* = C^*(DC^*)^* \tag{1.2}$$

or (2.4) is used giving

$$B^* = (C^*D)^* C^* \tag{1.3}$$

then C and D are always chosen so that

2. C is null wherever D is non-null and vice-versa; and

3. if (1.2) is used, DC* is null wherever C is non-null and vice-versa, and if (1.3) is used C*D is null wherever C is non-null and vice-versa.

Some terminology is useful here. We shall refer to calculating the closure C* of C as <u>eliminating</u> C. Evaluating the product DC* or C*D, as the case may be, is called <u>forward substitution</u> of C in D and finally finding the product C*(DC*)* (or (C*D)*C*) is called <u>back substitution</u> of D in C. An <u>elimination method</u> consists of expressing the computation of A* as a sequence of forward and back substitutions and elementary elimination steps, in which subsequences of these steps may be interpreted collectively as eliminating C for some matrix C.

The definition of an elimination method has a number of implications which we now consider. Firstly an elimination method always defines an ordering on the arcs of G which we shall call the <u>order of elimination</u> of the arcs. Specifically if at some stage (1.2) (or (1.3)) is used we say that the arcs of C are eliminated before the arcs of D. By virtue of properties 2. and 3. this ordering is clearly well-defined and

total, except that the arcs of each elementary matrix J_i are incommensurate - these are eliminated simultaneously.

Secondly, conditions 2. and 3. imply that we can always evaluate the non-null elements of $J_1, J_2, \ldots J_m$ by successive transformations of G. Specifically we can set $M^{(0)}=G$, then perform in-situ modifications of the elements of M to transform it to a matrix $M^{(f)}$ which contains the non-null elements of the matrices $J_1^*, J_2^*, \ldots J_m^*$ (other than e's on the diagonal) in their appropriate positions. If for instance at some stage in the derivation of (1.1) we use the formula (1.2) then the appropriate action would be to evaluate DC* (possibly using additional storage) and store the non-null elements of this matrix in the appropriate positions of M. The remaining elements of M are left unchanged. Once $M^{(f)}$ has been calculated G* can be calculated using (1.1). (This may not be the most efficient way of evaluating G* by the particular elimination method but our concern here is solely with the star-height of the resulting regular expressions.)

To investigate what actually happens when we perform an elimination step, let us suppose that at some stage the matrix M has the form

$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}$$

where M_{11}, M_{22} and M_{33} are square matrices.

We make no assumptions about the size of the various submatrices. Without loss of generality we may assume that some subset of the matrices M_{11}, M_{12}, M_{13}, M_{21}, M_{31} are to be eliminated. This elimination process will in general be itself expressed in terms of elementary eliminations and forward and back substitutions, but here we wish to consider its composite effect on M. A number of remarks are in order.

Remark 1 If the submatrix M_{ij} is next to be eliminated then the result of all previous eliminations will have been forward substituted into M_{ij}.

(To see this consider the context-free grammar

$$E \to EfEb \; ; \quad E \to \ell$$

f represents forward, b back substitution; ℓ represents elementary elimination. An algorithm for finding G* may be regarded as constructing a left-to-right bottom-up parse of a sentence of this grammar. If an E has just been recognised an f must follow (possibly preceeded by b's) before a new E may be recognised. The remark may now be proved by induction on the length of the derivation of the current state of the parse.)

Remark 2 No submatrix M_{ij} for $i \neq j$ may be eliminated before either M_{ii} or M_{jj} is eliminated without violating condition 3.

By remark 1 the result of eliminating M_{ij} must be forward substituted into M_{ii} and M_{jj} before they are eliminated. But this will result in a modification of M_{ij} itself, thus violating condition 3 - either it is premultiplied by M_{ii} or post-multiplied by M_{jj} depending on which of (1.2) or (1.3) is used at the time.

Remark 3 If M_{ii} and M_{ij} have been eliminated then M_{ji} may not be eliminated after M_{jj}.

If M_{ii} and M_{ij} have all been eliminated, then by remark 1 the results must be forwarded substituted into M_{ji} before this is eliminated. This involves pre- or post-multiplying M_{ji} by $M_{ii}^{*}M_{ij}$ (depending on whether (1.2) or (1.3) was used). However post-multiplication changes M_{ii}, violating condition 3. Pre-multiplication changes M_{jj} and thus will also violate condition 3 unless M_{jj} has not already been eliminated. Hence the remark.

Remark 4 M_{ik} and M_{ji}, where $k \neq i$ and $j \neq i$, may not both be eliminated using a single application of (1.2) or (1.3) without violating condition 3 since this would in general affect M_{jk}.

Remark 5 The star-height of elements of M is increased if and only if an elementary elimination step is executed in which the arc (k,k) is eliminated for some k.

This is obvious, because substitutions only involve multiplications of submatrices of M.

Now let us suppose that the next step is to eliminate M_{11} and M_{12}. (The case of eliminating M_{21} and M_{11} can be considered similarly.) After elimination of $C = M_{11} + M_{12}$,

$$M \leftarrow \begin{bmatrix} M_{11}^{*} & M_{11}^{*}M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}$$

In view of remark 5 we should like to see what effect this elimination has on M_{22} and M_{33}. Suppose therefore that

at some later stage we wish to eliminate the contents of M_{22}. By remark 3 we must already have eliminated M_{21}, or must do it simultaneously with the elimination of M_{22}. Hence, by remark 1, M_{11}^* and $M_{11}^* M_{12}$ must also have been forward substituted into M_{21} and M_{22}, and, in order not to violate condition 3, this can only be done by post-multiplying by M_{11}^* and $M_{11}^* M_{12}$. Thus after the forward substitution

$$M \supseteq \begin{bmatrix} M_{11}^* & M_{11}^* M_{12} & M_{13} \\ M_{21} M_{11}^* & M_{22} + M_{21} M_{11}^* M_{12} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}$$

($\supseteq$ is used here, because the elements of M may also have been changed between the elimination of M_{11} and M_{12} and their forward substitution into M_{22} and M_{21}). Note that the same result will be obtained if we consider M_{21} and M_{22} eliminated separately or simultaneously.

Thus we see that the star-height of expressions obtained by eliminating submatrices of M_{22} will now depend on the star-height of elements in M_{11} provided $M_{21} M_{11}^* M_{12}$ is non-null. In contrast, if we eliminate submatrices of M_{33} before eliminating M_{13} or M_{31} the star-height of the resulting regular expressions will not depend on the star-height of elements of M_{11} since forward substitution of M_{21} and M_{11} clearly does not affect M_{33}.

We restate this in the form of a lemma on the elements of the matrix M.

Lemma B2 At some stage in the elimination process let $M=[m_{ij}]$ and let the set of all arcs which have currently been eliminated be G_1. Suppose the next step in the elimination process involves the elimination of arc (i,i), and suppose after this step the set of eliminated arcs is G_2. Then,

(a) $m_{ii} = u + vm_{rr}w$

if

(b)(i) $\exists$ nodes k and j s.t. arcs (k,k) and (j,j) have been eliminated,

(ii) nodes k and j are strongly connected to node r in G_1,

(iii) arcs (i,k) and (j,i) are non-null,

and

(iv) arc (r,r) was the last diagonal arc to be eliminated in the section of G_1 containing {r,j,k},

and, moreover, (b) is true if

(c) node i is strongly connected to node r in G_2 and arc (r,r) was the last diagonal arc to be eliminated in the section of G_1 containing {r,j,k}.

(To see how this corresponds to our previous discussion, consider j,k and r as nodes of M_{11} and i as a node of M_{22}. Condition (iii) implies that $M_{21}M_{11}^{*}M_{12}$ is non-null, condition (iv) implies that since eliminating (r,r) the only changes made to M_{11} have been by back-substitution (and hence none have been made to m_{rr}), and finally conditions (i) and (ii) imply that $m_{kj} = u'm_{rr}v' + w'$ for some u',v' and w'.)

From the order of elimination of the arcs of G we can now construct an analysis of G; lemma B2 then enables us to relate the height of the analysis to the star-height of expressions in G*. We begin with the arcs eliminated first and proceed in order to the arcs eliminated last. Suppose at some stage we have considered the arcs of graph G_1 and have constructed an analysis of G_1. Suppose the constituent trees of this analysis have roots $r_1, r_2, \ldots r_p$. Suppose also that the next set J of arcs to be eliminated are all in the row/column i and that the union of these arcs and the arcs of G_1 form the graph G_2. If the arc (i,i) is not in J then do not alter the analysis. Otherwise consider the largest subset $r_{k_1}, \ldots, r_{k_x}$ of the roots $r_1, \ldots r_p$ such that each r_{k_j} is strongly connected to node i in G_2. If this subset is non-empty add a new root labelled i to the forest and connect it by branches to each of $r_{k_1}, \ldots, r_{k_x}$. If however the subset is empty then if $m_{ii} \neq \phi$ before elimination add a new root labelled i to the forest (and do not connect it to any others); if $m_{ii} = \phi$ before elimination do not alter the analysis.

Lemma B3 The star-height of m_{ii} after elimination of (i,i) is greater than or equal to the height of the tree with root i.

Proof This follows easily from lemma B2 by induction on the height of the tree. If the height is 0 or 1 the lemma is obvious from II 3.1 and 3.2. Otherwise, we note that each r_{k_y} satisfies the properties of r in lemma B2, hence after

elimination of (i,i), $m_{ii} \leftarrow (u+vm_{r_{k_y} r_{k_y}} w)^*$. By induction $m_{r_{k_y} r_{k_y}}$ has star-height $\geq$ to the height of the tree with root r_{k_y} and hence m_{ii} has star-height at least 1 greater than this. Hence the lemma.

We could strengthen the lemma to an equality as did Eggan [18] for the escalator method, but this is not relevant here.

Combining Theorem B1 and lemma B3 we have our theorem:

Theorem B4 If an elimination method is used to find G* for a graph G, then G* will contain regular expressions having star-height at least equal to the rank of G.