GRAPHS

FACTOR ~~THEORY~~, FAILURE FUNCTIONS AND BI-TREES

Roland C. Backhouse and Rüdiger K. Lutz,
Heriot-Watt University, Edinburgh, Scotland.

Abstract

The factors and factor matrix of a regular language are defined and their properties stated. It is then shown that the factor matrix of a language Q has a unique starth root - called the factor graph of Q - which is a recogniser of Q. An algorithm to calculate the factor graph is presented. The Knuth, Morris and Pratt pattern matching algorithm, its extensions and Weiner's substring identifier algorithm are outlined and are all shown to be equivalent to finding the factor graph of some regular language.

Key words and phrases: regular language, factor, factor matrix, pattern matching, string-matching.

Authors' address: Department of Computer Science,
Heriot-Watt University,
79 Grassmarket,
Edinburgh, EH1 2HJ,
SCOTLAND.

Tech. Rep. No. 4, October 1976.

1. Introduction

In order to be of any value a theory must do three things: it must correlate a body of known facts, it must explain these facts and (most importantly) it must predict a number of related facts not already observed by the practitioners of the subject. Automata theory has certainly proved its value in correlating and explaining a large variety of techniques in language recognition and related areas. Yet it has rarely been used to predict results of practical value to computer programmers which have not already been obtained by other means. One exception is the remarkable result due to Knuth, Morris and Pratt [7] that pattern-matching (i.e. finding all occurrences of a given string as a consecutive substring of a given text) can be performed in linear time.

In this paper we do not predict any significant new algorithms, nor, indeed, do we explain any known algorithms. Our contribution is merely to correlate the Knuth, Morris, Pratt [7] pattern-matching algorithm and Weiner's [11] substring identifier algorithm with a little-known area of automata theory initially developed by Conway [6] - the study of factors of a regular language. The correlation is, however, quite startling and we therefore feel that it offers a significant challenge to automata theorists to explain the correlation and to exploit it by developing new algorithms for the solution of practical problems.

In section 2 we define the factors of a language and state a number of properties of factors due to Conway [6]. We then prove that the factors of a regular language Q define a (non-deterministic) recogniser of Q which we call the factor graph of Q. Sections 3 and 4 then show that the failure function method of solving the pattern matching problem

is equivalent to finding the factor graph of a regular language. Section 5 shows that, after a minor modification, Weiner's algorithm [11] is also equivalent to finding the factor graph of a regular language.

We shall assume familiarity of the reader with the terminology of graph theory and language theory [2]. There is a well-known correspondence between labelled p-node graphs and p×p matrices, and hence we use the terms graph and matrix synonomously. e is used to denote the empty word and V is used to denote a finite alphabet. Following Conway [6] we call a matrix all of whose non-null entries are e a constant matrix and a matrix all of whose entries are subsets $\{a_1,a_2,\ldots,a_k\}$ of V a linear matrix. A constant + linear matrix is, as the terminology suggests, one which is the union of a constant and a linear matrix. A recogniser (G,S,T) consists of a constant + linear matrix G and two subsets S and T of the nodes of G which are designated as start and terminal nodes, respectively. The language recognised by (G,S,T) is $\bigcup_{\substack{s\in S\\ t\in T}} G^*_{st}$. A recogniser is all-admissible if for all nodes x of the graph there is a path in the graph from some start node s to x and a path from x to some terminal node t. Finally if X is a finite set 2^X denotes the set of all subsets of X and $|X|$ denotes the size of X.

2. Factory Theory

The concept of a factor of a language was first introduced by Conway [6]. Conway also introduced the concept of the factor matrix of a regular language. In this section we introduce the concept of a factor graph and present an algorithm for computing the factor graph of a given regular language. Firstly, however, we need to summarize a few well-known properties of regular languages as well as the fundamental properties of factors due to Conway.

2.1 $\mathcal{L}$-, c- and r- classes

This section defines the $\mathcal{L}$-, c- and r- classes of a language Q and relates them to the derivatives and anti-derivatives of Q. The reader familiar with the work of Rabin and Scott [10] should have no difficulty understanding our definitions.

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

$$xQ_{\mathcal{L}}y \iff (\forall z \in V^*, \quad zx \in Q \iff zy \in Q)$$

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

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

These are, of course, the usual left-invariant equivalence relation, right-invariant equivalence relation and congruence relation introduced by Rabin and Scott [10].

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

Theorem 2.1

A language $Q \subseteq V^*$ is regular <=> the relation Q_l is of finite index <=> the relation Q_r is of finite index <=> 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 $l(x)$ 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 semi-group of Q is the quotient of the free semi-group 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 $\{l_1, \ldots, l_m\}$ and the states of the anti-machine be $\{r_1, \ldots, r_{am}\}$. 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, $l(x)$, 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 $c_i.c(a) = c_j$. 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, ...).

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

$$D_xQ = \{y | xy \varepsilon Q\}$$

$$\sim D_xQ = \{y | xy \notin Q\} .$$

D_xQ is called the derivative of Q with respect to x.

We then have:

Lemma 2.3

$$x\ Q_r\ y \iff D_xQ = D_yQ$$

This is the basis of the method of derivatives for calculating the machine of a language Q [4].

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

$$\mathfrak{a}_xQ = \{y \mid xy\varepsilon\overleftarrow{Q}\} = \{y \mid \overleftarrow{yx}\varepsilon Q\} .$$

Lemma 2.4

$$x\ Q_l\ y \iff \overleftarrow{\mathfrak{a}_xQ} = \overleftarrow{\mathfrak{a}_yQ} .$$

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

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

Lemma 2.5

$$x\ Q_c\ y \iff C_xQ = C_yQ .$$

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

Theorem 2.6

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

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

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

Proof

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

Then $y \in D_xQ \iff xy \in Q \iff \overleftarrow{x} \in \mathfrak{C}_{\overleftarrow{y}}Q$.

But by lemma 2.4, $\mathfrak{C}_{\overleftarrow{y}}Q = \mathfrak{C}_{\overleftarrow{y^1}}Q$ for all y^1 such that $y^1 Q_l y$.

$\therefore\ y \in D_xQ \iff y^1 \in D_xQ$ for all y^1 such that $y^1\ Q_l\ y$.

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

Part (b) is proved similarly.

Consider now C_xQ. The pair $(u,v) \in C_xQ \iff uxv \in Q \iff v \in D_{ux}Q$ and $\overleftarrow{u} \in \mathfrak{C}_{\overleftarrow{xv}}Q$.

But then, by an identical argument to that above, this implies that $u^1xv^1 \in Q$ for all $u^1 \in l(u)$ and $v^1 \in r(v)$.

i.e. $C_xQ \supseteq l(u) \times r(v)$.

Whence $C_xQ = \sum_{(u,v)\in C_xQ} l(u) \times r(v)$, 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.

2.2 The Fundamentals of Factor Theory

The following definitions are taken from Conway [6].

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 [6].

Lemma 2.7

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

Lemma 2.8

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 (2.1) 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 2.9

The left factors of any language Q are either ϕ (the empty set) or are unions of $\mathcal{l}$-classes of Q. The right factors of Q are either ϕ or are unions of r-classes of Q and the factors are ϕ or are unions 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 union of $\mathcal{l}$-classes of Q. Similarly any non-empty right factor is a union of r-classes of Q.

If H is any factor of Q it is the central term in a factorization $LHR \subseteq Q$ (lemma 2.8). If $H \neq \phi$, let $x \in H$. Then the set $C_x Q = \{(u,v) \mid uxv \in Q\} \supseteq L \times R = \{(u,v) \mid u \in L, v \in R\}$. 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 characterization 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 where Q is regular.

2.3 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 2.8) 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 2.7 and 2.8, 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 × q matrix which is called the factor matrix of Q and is denoted $\lceil Q \rceil$.

Various properties of the factor matrix may be observed [6], some of which are summarised below.

Theorem 2.10

(i) H is a factor of Q <=> H is some entry Q_{ij} in the factor matrix $\lceil Q \rceil$.

(ii) Q_{ij} is maximal in the subfactorizations $L_i.Q_{ij} \subseteq L_j$ and $Q_{ij}.R_j \subseteq R_i$. 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) $\lceil Q \rceil = \lceil Q \rceil^*$.

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

The reader is referred to [3] or [6] for the proof of theorem 2.10 which is quite straightforward.

Theorem 2.10 is an extremely interesting and powerful theorem, from which most results on factors can be deduced immediately. Particularly useful is 2.10 (iv), which we shall often apply in its alternative form (a) $Q_{ii} \supseteq e$ and (b) $Q_{ij} = \sum_k Q_{ik}.Q_{kj}$.

Part (iii) tells us that the s th column of $\lceil Q \rceil$ 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), $\lceil Q \rceil = \lceil Q \rceil^*$, 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.

2.4 The Factor Graph

In this section 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 is not straightforward 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 $\lceil Q \rceil$. If α is a regular language it is quite easy to prove that there is always a unique minimal starth root of α^* (see Brzozowski [5]). 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

$$A_1 = \begin{bmatrix} \phi & e & \phi \\ \phi & \phi & e \\ e & \phi & \phi \end{bmatrix} \quad \text{and} \quad A_2 = \begin{bmatrix} \phi & \phi & e \\ e & \phi & \phi \\ \phi & e & \phi \end{bmatrix}$$

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.

Theorem 2.11 (Conway)

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

Proof

C_{max} and L_{max} are defined to be the unique maximal constant and linear matrices (respectively) such that $\lceil Q \rceil \supseteq C_{max}$ and $\lceil Q \rceil \supseteq L_{max}$. Once again we refer the reader to [3] or [6] for the remainder of the proof.

Theorem 2.12

Let A be an element of $M_p(RL)$. Let $M_p(RL)$ 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^*, where $X^{2+*} = X^2 . X^*$.

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

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

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

$\Rightarrow \quad n = 1$

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

I.e. $\quad X \subseteq Y$ 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 2.11 and 2.12 enable us to proceed to the proof of our main theorem.

Lemma 2.13

$C_{max} \backslash E$ 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 $\lceil Q \rceil$. We have $\lceil Q \rceil = \lceil Q \rceil *$, (2.10 (iv)), and $\lceil Q \rceil \supseteq C_{max}$, by Theorem 2.11. Therefore $\lceil Q \rceil = \lceil Q \rceil \cdot \lceil Q \rceil \supseteq \lceil Q \rceil \cdot C_{max}$.

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

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

Therefore $Q_{si} = Q_{sj}$ i.e. $L_i = L_j$ (by 2.10 (iii)).

Similarly, using $\lceil Q \rceil \geq C_{max} \cdot \lceil Q \rceil$, 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 2.14

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

Finally we come to the main theorem of this section.

Theorem 2.15

Let Q be a regular language, and let C_{max} and L_{max} be as defined in Theorem 2.11. 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 2.10 (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 2.12, 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 $G_Q^* \geq (C_{max} + L_{max}) \backslash E$, since then $G_Q^* = (G_Q^*)^* \geq (C_{max} + L_{max})^*$ $= \lceil Q \rceil$, by Theorem 2.11.

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 2.15. Hence the initial assumption that property p3 holds for w must be false. Hence $G^*_Q \supseteq (C_{max} + L_{max})\backslash E$, and by our earlier argument $G^*_Q = \boxed{Q}$. Finally, applying Theorem 2.12, G_Q is the minimal starth root

of $\overline{|Q|}$. The last part of the theorem follows immediately from Theorem 2.10 (iii), $Q = Q_{st} = [G^*_Q]_{st}$.

2.5 An Algorithm to calculate the factor graph

The previous results embody an algorithm to calculate the factor graph of Q, which we shall develop with the aid of an example.

Step 1

Calculate the machine and anti-machine for the language Q. Label the states of the machine (anti-machine) $l_1, l_2, \ldots, l_m$ ($r_1, r_2, \ldots, r_{am}$) 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 $x_i y_j \in Q$.

Example 1

Let Q = (b + a(aa*b)*b)*

The machine and anti-machine of Q, constructed in step 1, are shown below.

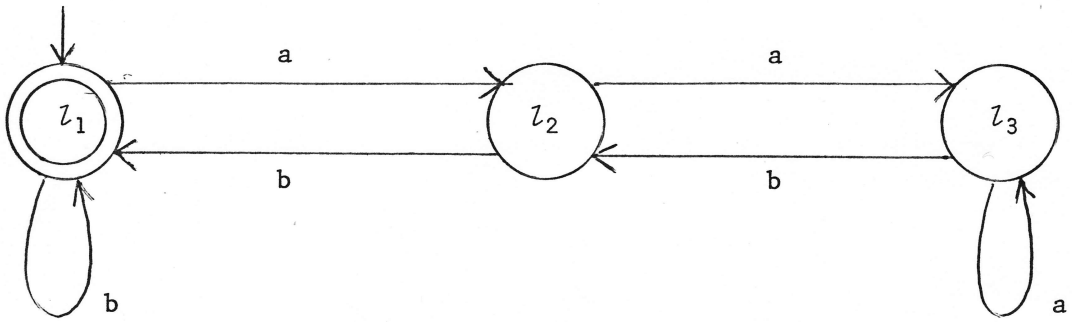


Fig. 1 (a) Machine of Q.

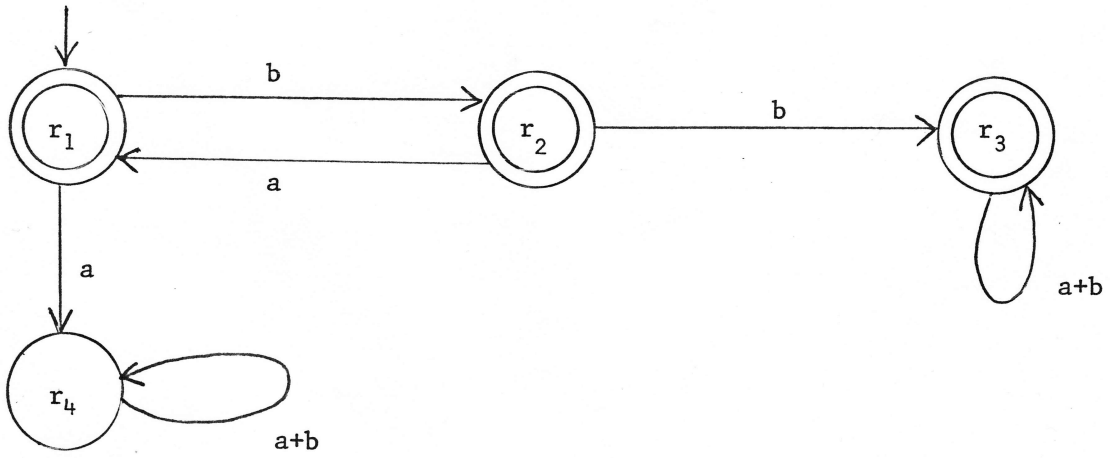


Fig. 1 (b) Anti-machine of Q.

The tables constructed in step 2 are shown below.

Node/ $\mathcal{L}$-class	Representative element	Derivative
$\mathcal{L}_1$	e	$r_1 + r_2 + r_3$
$\mathcal{L}_2$	a	$r_2 + r_3$
$\mathcal{L}_3$	aa	r_3

Table 1(a) Machine

Node/ r-class	Representative element	Reverse of anti-derivative
r_1	e	$\mathcal{L}_1$
r_2	b	$\mathcal{L}_1 + \mathcal{L}_2$
r_3	bb	$\mathcal{L}_1 + \mathcal{L}_2 + \mathcal{L}_3$
r_4	a	ϕ

Table 1(b) Anti-machine

The correctness of step 2 is based on theorem 2.6.

For instance the class $\mathcal{l}_2$ has representative a

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

and a.a $\notin$ Q (a ——— " ——— r_4),

but a.b $\in$ Q (b ——— " ——— r_2),

and a.bb $\in$ Q (bb ——— " ——— r_3),

the derivative corresponding to $\mathcal{l}_2$ is $r_2 + r_3$.

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 $\{\mathcal{l}_{i_1}, \ldots, \mathcal{l}_{i_k}\}$ (including the empty subset) of the $\mathcal{l}$-classes of Q and finds for each subset those classes $r_{j_1}, \ldots r_{j_n}$ common to the $\mathcal{l}_{i_1}$ th, $\mathcal{l}_{i_2}$ th, ..., $\mathcal{l}_{i_k}$ th entries in the third column of Table 1. One then has $(\mathcal{l}_{i_1} + \ldots + \mathcal{l}_{i_k}).(r_{j_1} + \ldots + r_{j_n}) \subseteq Q$ 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. $L_t = \mathcal{l}_{t_1} + \mathcal{l}_{t_2} + \ldots + \mathcal{l}_{t_k}$ is that left factor such that the $\mathcal{l}$-class $\mathcal{l}_{t_i} \subseteq L_t$ 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.

Applying step 3 to our example we would get the following subfactorizations:

$$(l_1+l_2+l_3).r_3 \quad , \quad (l_2+l_3).r_3 \ , \ (l_1+l_3).r_3, \ l_3.r_3,$$
$$(l_1+l_2).(r_2+r_3) \quad , \quad l_2.(r_2+r_3),$$
$$l_1.(r_1+r_2+r_3) \ ,$$
$$\phi \ .(r_1+r_2+r_3+r_4).$$

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 2, 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	l_1+l_2	R_2	r_2+r_3
	L_3	$l_1+l_2+l_3$	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)), $Q = l_1$, and from the anti-machine (Fig. 1(b)), $Q = r_1+r_2+r_3$; but $L_1 = l_1$ and $R_1 = r_1+r_2+r_3$.

Step 4

Calculate C_{max}. Whence deduce

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

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

Step 5

Calculate L_{max}. Whence deduce

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

$[L_{max}]_{ij} \supseteq a$ 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 $\mathcal{l}$-classes of Q and the knowledge that $\mathcal{l}_k.a \subseteq \mathcal{l}_j$ if and only if under input a the $\mathcal{l}_k$th state of the machine for Q goes to state $\mathcal{l}_j$.

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

Steps 4 and 5, applied to our example, are summarised in figs. 2-5, below.

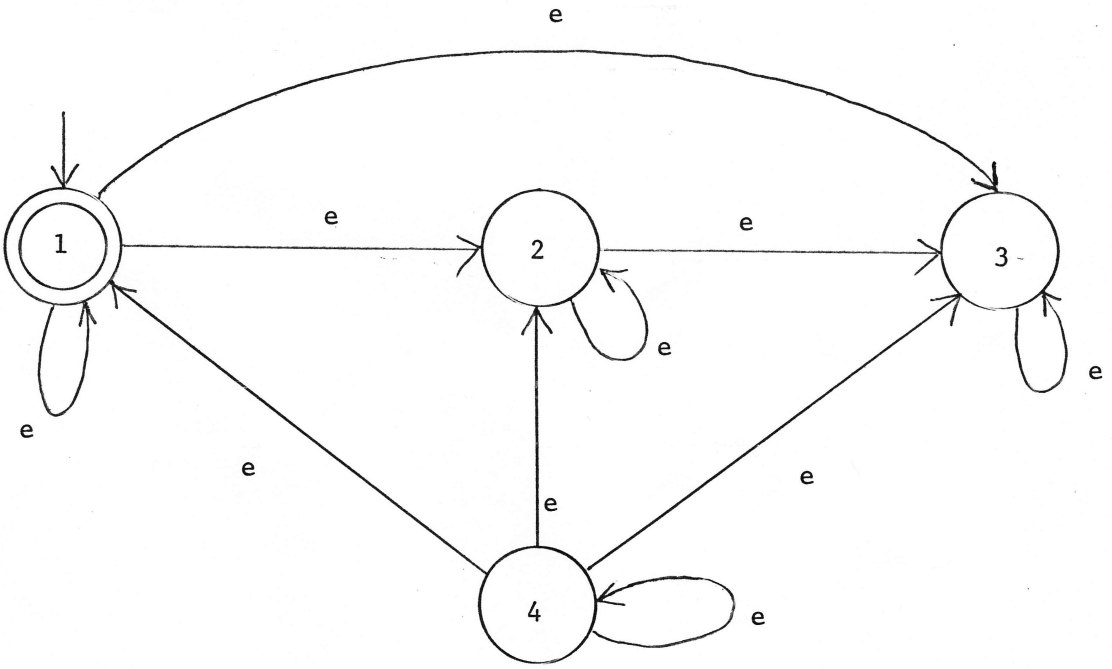


Fig. 2 - C_{max}

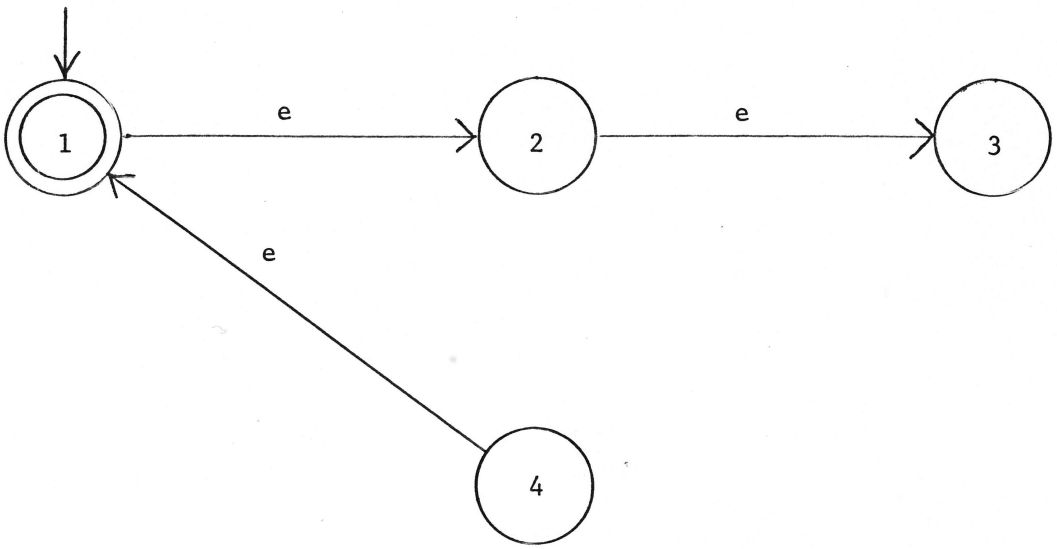


Fig. 3 - C_{min}

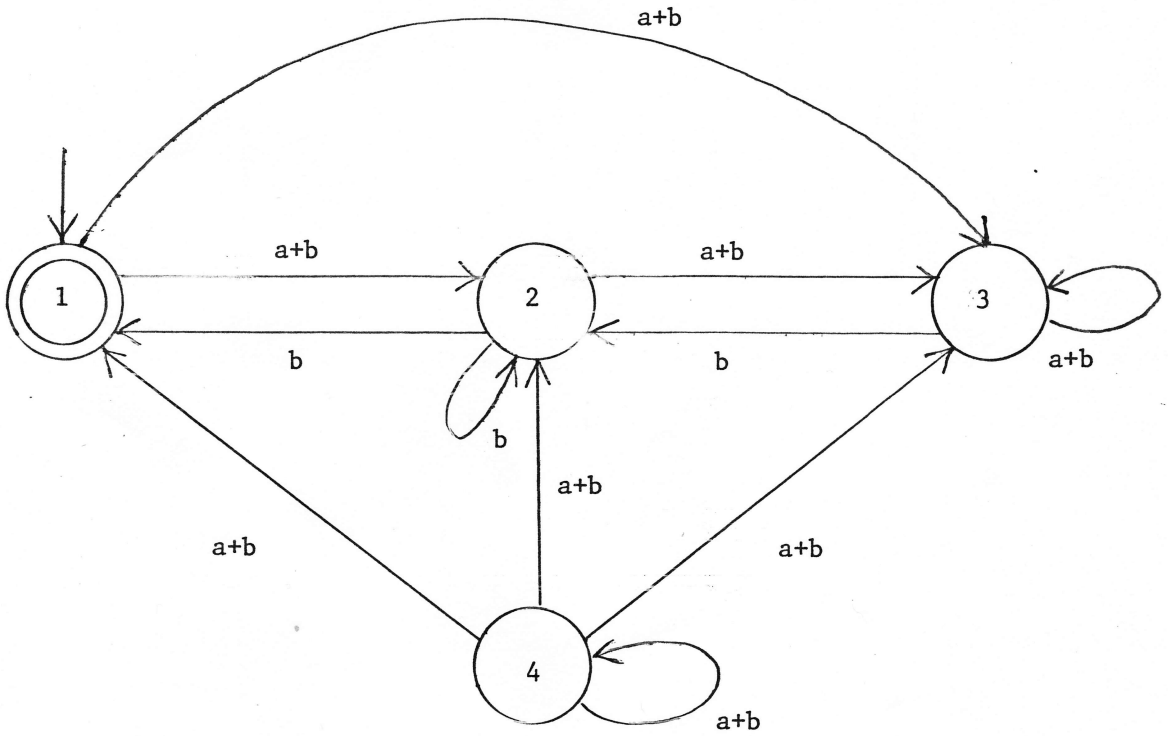


Fig. 4 - L_{max}

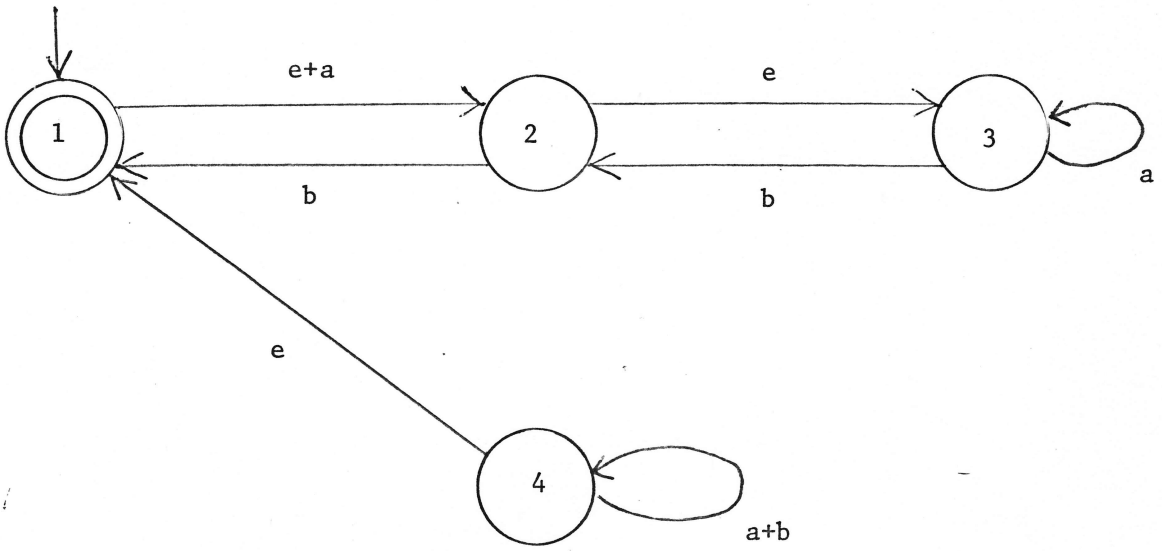


Fig. 5 The Factor Graph G_Q

The construction of C_{max} and C_{min} should be clear. For instance there is an arc from node 1 to nodes 2 and 3 in C_{max} since $L_1 = \mathcal{l}_1 \subseteq L_2 = \mathcal{l}_1+\mathcal{l}_2 \subseteq L_3 = \mathcal{l}_1 + \mathcal{l}_2 + \mathcal{l}_3$. There is no arc from 1 to 3 in C_{min}, since there are arcs from 1 to 2 and from 2 to 3. The construction of L_{max} is slightly more complicated. As an example, there is an arc labelled b from node 2 to nodes 2 and 1 since $L_2.b = (\mathcal{l}_1 + \mathcal{l}_2).b$ $= \mathcal{l}_1.b + \mathcal{l}_2.b = \mathcal{l}_1 + \mathcal{l}_1$ (from the machine in fig. 1(a)), and $\mathcal{l}_1 \subseteq L_2 \subseteq L_1$.

2.6 Remarks on the Algorithm

Conway [6] advises his readers against calculating the factor matrix of a regular language. Whilst disagreeing with this advice - the following sections would not have been written had we not done so - it is necessary to observe that the above algorithm has exponential time complexity (in the size of a regular expression denoting Q). Calculating the machine or anti-machine may have exponential time-complexity and so also may the comparisons of subfactorizations in step 3. Indeed the number of nodes in the factor graph of Q may be exponential in the size of a regular expression denoting Q. For example, if Q is the set of all words such that the number of a's minus the number of b's is not congruent to 0 modulo n, where $n \geq 3$, the machine of Q has n nodes but its factor graph has 2^n nodes [3].

It is difficult for us to compare the above algorithm with Conway's method of calculating the factor matrix for the simple reason that we have never completely understood Conway's method! Our method is quite straightforward to use (on small expressions) and one particularly important merit is its systematic use of representatives of the $\mathcal{L}$- and r-classes. There is some redundancy in the algorithm - the third column of table 1(b) can be deduced from the third column of table 1(a), or vice-versa - but this redundancy is probably worth retaining as a check on hand calculations.

The above remarks militate strongly against the possibility of applying factor theory in any practical language recognition problems. Nevertheless the next three sections demonstrate one area where the factor graph has found a practical application. The algorithms used bear no resemblance to the algorithm of section 2.5, thus suggesting that more

efficient (specifically and generally applicable) algorithms can be designed. In summary, therefore, the algorithm of section 2.5, is proposed as a stop-gap measure to enable us to become more familiar with the properties of factor graphs.

A minor technical nuisance in the calculation of factor graphs is that ϕ may be a factor and the factor graph can have up to two "useless" nodes, i.e. nodes such there is no path from node s to the node, or no path from the node to t. We shall call the factor graph obtained by eliminating these useless nodes the all-admissible factor graph of Q and, in future, all factor graphs we display will be all-admissible factor graphs.

3. Failure Functions

The problem of relevance to the next three sections is the string-matching problem. That is, given a (long) symbol string $X = x_1\ x_2 \ldots\ x_m$, the "text", and another (short) string $Y = y_1\ y_2 \ldots\ y_n$, the "pattern", over the same alphabet V, find all occurrences of the pattern as a consecutive substring in the text.

Two methods for solving this problem are available, both of which have time-complexity which is linear in the combined length of the pattern and text strings. In the next two sections we shall relate the first method, the use of failure functions [7], to factor theory and in section 5 we relate the second method, Weiner's bi-trees [11] to factor theory.

Definition

Given a string $Z = z_1\ z_2 \ldots z_r$, the function $f_Z^*: \{1 \ldots r\} \to 2^{\{0 \ldots r\}}$ is defined by:

$$f_Z^*(i) = \{j \mid j \leq i \text{ and } z_1 \ldots z_j = z_{i-j+1} \ldots z_i\}.$$

The failure function $f_Z: \{1 \ldots r\} \to \{0 \ldots r-1\}$ is defined by:

$$f_Z(i) = \max \{j \mid j \in f_Z^*(i) \text{ and } j \neq i\} .$$

Where no confusion is likely to arise we shall omit the subscript Z from f_Z and f_Z^*.

Example 2

The functions f and f* defined for Z = aabbaab are given in table 2.

i	1	2	3	4	5	6	7
z_i	a	a	b	b	a	a	b
f*(i)	{0,1}	{0,1,2}	{0,3}	{0,4}	{0,1,5}	{0,1,2,6}	{0,3,7}
f(i)	0	1	0	0	1	2	3

TABLE 2

The importance of the failure function to the string-matching problem lies in the fact that it may be computed in time O(r)[7]. To solve the string-matching problem concatenate Y, a new symbol $ and X in that order, and compute the failure function for Z = Y$X. The values of i for which f(i) = n mark the positions where Y matches a substring of X, or more precisely:

$$f_Z(n+1+i) = n \iff z_1 \ldots z_n = z_{i+2} \ldots z_{n+1+i}$$

$$\iff y_1 \ldots y_n = x_{i-n+1} \ldots x_i$$

We shall not describe the computation of the failure function in any detail. In outline the failure function is calculated for the successive strings $z_1, z_1z_2, z_1z_2z_3, \ldots, z_1z_2z_3 \ldots z_r$. In calculating f(i), the failure function for $z_1z_2 \ldots z_{i-1}$ is used to "recognise" (non-deterministically) the string $z_1z_2 \ldots z_i$. The construction of a recogniser from the failure function is as follows.

Construction 1

Given a string Z over alphabet V and the failure function f for Z this algorithm constructs a transition diagram G(Z). (Later we show that G(Z) is the all-admissible factor graph of V*Z.)

1. G(Z) has r+1 nodes labelled 0,1,...,r.
2. For each $a \in V$, $a \neq z_1$, add an arc labelled a from node 0 to itself.
3. For each i, $1 \leq i \leq r$ add an arc labelled z_i from node i-1 to node i.
4. For each i, $1 \leq i \leq r$, add an arc labelled e (the empty word) from node i to f(i).
5. Node 0 is the start state and node r is the final state.

Example 2 (again)

The transition diagram constructed as above for the string aabbaab is shown in fig. 6.

If we append to Z an (r+1) th symbol z_{r+1} we can "recognise" f(r+1) by finding the largest value i such that there is a path from node 0 to node i which spells out $z_1 z_2 \ldots z_{r+1}$. We then have f(r+1) = i.

Theorem 3.4 states that G(Z) is the all-admissible factor graph of $V^* z_1 z_2 \ldots z_r$. First, however, we present a number of lemmas.

In all the following lemmas the string $Z = z_1 z_2 \ldots z_r$, over the alphabet V, is understood.

Lemma 3.1

$$f^*(i) = \{j \mid z_1 \ldots z_i \in V^* z_1 \ldots z_j\}$$
$$= \{j \mid V^* z_1 \ldots z_i \subseteq V^* z_1 \ldots z_j\} .$$

Proof

Clearly $\{j \mid z_1 \ldots z_i \in V^* z_1 \ldots z_j\} = \{j \mid V^* z_1 \ldots z_i \subseteq V^* z_1 \ldots z_j\}$.

Now $z_1 \ldots z_i \in V^* z_1 \ldots z_j \Rightarrow i \geq j$ and $z_1 \ldots z_i = w z_1 \ldots z_j$ for some $w \in V^*$.

I.e. $z_{i-j+1} \ldots z_i = z_1 \ldots z_j$. Also $i \geq j$ and $z_{i-j+1} \ldots z_i = z_1 \ldots z_j$

$\Rightarrow V^* z_{i-j+1} \ldots z_i = V^* z_1 \ldots z_j \supseteq V^* z_1 \ldots z_i$. Hence

$\{j \mid V^* z_1 \ldots z_i \subseteq V^* z_1 \ldots z_j\} = \{j \mid z_{i-j+1} \ldots z_i = z_1 \ldots z_j\}$.

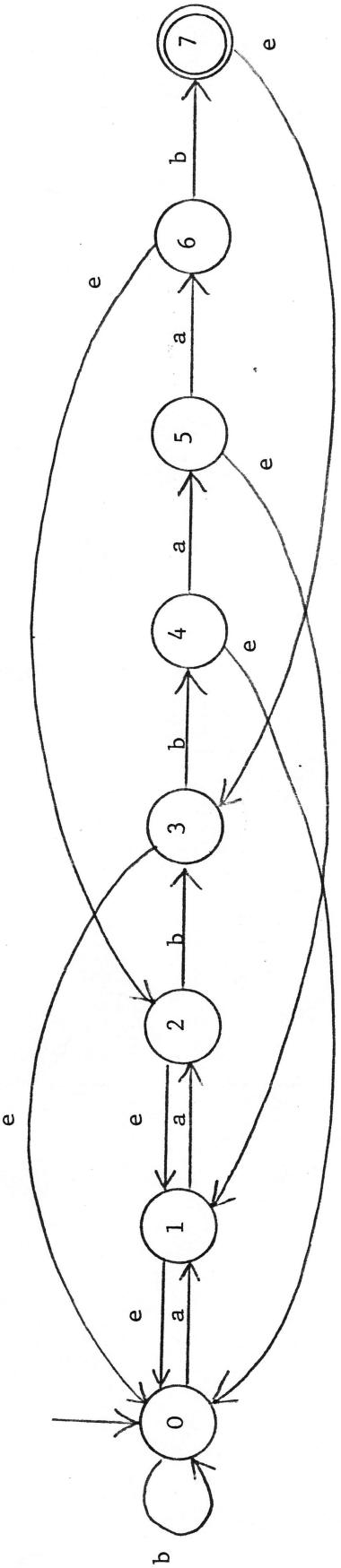


FIG. 6 FACTOR GRAPH OF V*aabbaab

The notation f* is intended to suggest the following lemma:

Lemma 3.2

$f^*(i) = \{i\} \cup f^*(f(i))$.

Proof

Clearly $i \in f^*(i)$. Now suppose $j \neq i$.

$j \in f^*(i) \iff j<i$ and $z_1 \ldots z_j = z_{i-j+1} \ldots z_i$

$\iff j \leq f(i)$ and $z_1 \ldots z_j = z_{i-j+1} \ldots z_i$

and $z_1 \ldots z_{f(i)} = z_{i-f(i)+1} \ldots z_i$

(by the definition of f(i))

$\iff j \leq f(i)$ and $z_1 \ldots z_j = z_{f(i)-j+1} \ldots z_{f(i)}$.

$\iff j \in f^*(f(i))$.

Let $Q = V^* z_1 \ldots z_r$.

Lemma 3.3

Q has r+2 left factors, namely $L_o = V^*$, $L_1 = V^* z_1$, $L_2 = V^* z_1 z_2, \ldots,$ $L_r = V^* z_1 z_2 \ldots z_r$ and $L_{r+1} = \phi$.

Proof

ϕ is maximal in $\phi . V^* \subseteq Q$ (assuming $r \geq 1$) and so is a left factor of Q. V^* is also clearly a left factor. Finally $V^* z_1 \ldots z_i$ is maximal in the subfactorization $V^* z_1 \ldots z_i . z_{i+1} \ldots z_r \subseteq Q$ and so is also a left factor. Thus Q has at least r+2 left factors.

Let $L.R \subseteq Q$ be a factorization of Q. $R \neq \phi$ since $V^* . z_1 z_2 \ldots z_r \subseteq Q$. Let w be a word of shortest length in R. There are two cases to consider.

Case 1

length(w) $\geq$ r. In this case $R \subseteq V^* z_1 z_2 \ldots z_r$ and $V^* . V^* z_1 \ldots z_r$ dominates $L.R \subseteq Q$. $\therefore L = V^*$.

Case 2

length(w) < r. Let length(w) = r-i, where $1 \leq i \leq r$. Then $w = z_{i+1} \ldots z_r$ and $L \subseteq V^* z_1 z_2 \ldots z_i$.

Suppose $L \neq \phi$. Let $v \in R$ and $u \in L$. $u = u^1 z_1 \ldots z_i$ for some $u^1 \in V^*$. Also length(v) $\geq$ r-i (by the choice of i). Hence $u.v \in Q$ implies $z_1 \ldots z_i . v \in Q$. I.e. $z_1 \ldots z_i \in L$ by maximality of L. But then $z_1 \ldots z_i . R \subseteq Q$, hence $V^* z_1 \ldots z_i . R \subseteq V^* . Q = Q$. I.e. $V^* z_1 \ldots z_i \subseteq L$ (by maximality of L). Thus $L = V^* z_1 \ldots z_i$.

We note that ϕ is a factor of Q and therefore the factor graph of Q has one "useless" node. In the proof of theorm 3.4 we shall, as anticipated earlier, ignore this useless node and its associated entries in the factor graph and factor matrix. Theorem 3.4 and its proof will therefore refer always to the all-admissible factor graph.

Theorem 3.4

The transition diagram G(Z) of construction 1 is the (all-admissible) factor graph of $Q = V^* Z$.

Proof

By lemma 3.3 the factor graph of Q has r+1 nodes. Let C_{max} and L_{max} be the maximal constant and linear matrices such that $\boxed{Q} = (C_{max} + L_{max})^*$. Since $V^* z_1 \ldots z_{i-1} . z_i . z_{i+1} \ldots z_r \subseteq Q$ and $V^* z_1 \ldots z_{f(i)} \supseteq V^* z_1 \ldots z_i$ (lemma 3.1) we have $C_{max} + L_{max} \supseteq G(Z)$. Let C_{min} and L_{min} be the constant and linear parts of G(Z). We must prove that (a) $C_{max} = C_{min}^*$, (b) $L_{max} \subseteq (C_{min} + L_{min})^*$, (c) $C_{min} \subseteq (C_{max} \setminus E) \setminus (C_{max} \setminus E)^{2+*}$ and (d) $L_{min} \subseteq L_{max} \setminus ((C_{max} + L_{max}) \setminus E)^{2+*}$.

(a) and (b) establish that $\boxed{Q} = G(Z)^*$; (c) and (d) establish the minimality of G(Z).

Let $L_i = V^*z_1 \ldots z_i$, $1 \le i \le r$ and $L_o = V^*$.

(a) is immediate from lemma 3.1 and the identity $C^*_{min} = E + C_{min}.C^*_{min}$. (For, $e \in Q_{ij} \iff L_j \supseteq L_i$ (Theorem 2.10) $\iff j \in f^*(i)$ (lemma 3.2). $e \in [C_{min}]_{ij} \iff j = f(i)$.)

To prove (b), suppose $a \in Q_{ij}$, where $a \in V$. Then $z_1 \ldots z_i.a.z_{j+1} \ldots z_r \subseteq L_i.a.R_j \subseteq Q$ (Theorem 2.10) $\Rightarrow i \ge j-1$, $a = z_j$ and $z_1 z_2 \ldots z_{j-1} = z_{i-j+2} \ldots z_i$. Thus $j-1 \in f^*(i)$, whence $[C^*_{min}]_{i,j-1} = e$ (by (a)), and $[L_{min}]_{j-1,j} = a$. $\therefore a \in [C^*_{min}.L_{min}]_{ij}$. I.e. $L_{max} \subseteq (C_{min}+L_{min})^*$.

To prove (c), suppose $e \in [C_{min}]_{ij}$ and $e \in [(C_{max} \setminus E)^{2+*}]_{ij}$. Then $j = f(i)$. But also $\exists$ k such that $[C_{max} \setminus E]_{ik} = e$ and $[C_{max} \setminus E]_{kj} = e$. I.e. $i \ne k \ne j$, $k \in f^*(i)$ and $j \in f^*(k)$. But then $i > k > j$ and $k \in f^*(i)$ contradicting the maximality of $f(i) = j$.

(d) is proved similarly. Suppose $z_j \in [(C_{max}+L_{max}) \setminus E]^{2+*}_{j-1,j}$. Then $\exists$ indices k, m such that $e \in [C_{max} \setminus E]_{j-1,k}$, $z_j \in [L_{max}]_{km}$ and $e \in [C_{max} \setminus E]_{mj}$ $\therefore$ $k < j-1$ and $j < m$. I.e. $k < m-1$. But also $z_1 \ldots z_k.z_j.z_{m+1} \ldots z_r \subseteq L_k.z_j.R_m \subseteq Q$ (Theorem 2.10) $\Rightarrow k \ge m-1$. This is a contradiction, so (d) is proved.

4. Generalised Failure Functions

An obvious generalisation of the string-matching problem is the following: Given a text X and p patterns $Y_1, Y_2, \ldots, Y_p$ find all occurrences of each pattern in the text. One method of solving this problem would be to apply the method outlined in the last section to each of the strings $Y_1, Y_2, \ldots, Y_p$ in turn. The time required would then be proportional to $p.m + n_1 + n_2 + \ldots + n_p$, where m is the length of the text and n_i is the length of the pattern Y_i. This is clearly unacceptable and we are thus lead to consider generalising the failure function to sets of strings or alternatively (in fact, equivalently) finding the factor graph of V*W where W is a set of strings.

We begin by generalising the failure function to a set of strings.

Definition

Let $W = \{w_1 = a_{11}\ldots a_{1n_1}, w_2 = a_{21}\ldots a_{2n_2}, \ldots, w_k = a_{k1}\ldots a_{kn_k}\}$ be a set of words over the alphabet V. The function $f^*_W: V^* \to 2^{V^*}$ is defined by:

$$f^*_W(b_1b_2\ldots b_m) = \{v = c_1c_2\ldots c_r \mid r \geq 0,\ v = b_{m-r+1}\ldots b_m \text{ and } \exists\ w_i \in W \text{ with } v = a_{i1}\ldots a_{ir}\}$$

Thus $f^*_W(u)$ is the set of all prefixes of words in W which are also suffices of u.

The function $f_W: V^* \to V^*$ is defined by

$$f_W(u) = v_{max}$$

where $v_{max} \in f^*_W(u)$, $v_{max} \neq u$ and $\text{length}(v_{max}) \geq \text{length}(v)$ for all $v \neq u$, $v \in f^*_W(u)$. (Clearly $f_W(u)$ is well-defined.)

If $W = \{Z = z_1 z_2 \ldots z_r\}$ is a set containing exactly one word then $j \geq 1$ and $z_1 z_2 \ldots z_j \in f_W^*(u)$

$\Leftrightarrow$ $u = z_1 z_2 \ldots z_i$ for some i, $1 \leq j \leq i \leq r$

and $j \in f_Z^*(u)$.

In this way f_W^* generalises the previous definition of f_Z^* .

Again we shall choose to omit the subscript W when there is no danger of confusion.

Lemmas 4.1 and 4.2 are analogous to lemmas 3.1 and 3.2 and can be proved using the same techniques.

Let P(W) be the set of all words v (including the empty word) which prefix a word in W.

Lemma 4.1

$$f_W^*(u) = \{v \mid v \in P(W) \text{ and } u \in V^*v\}$$
$$= \{v \mid v \in P(W) \text{ and } V^*u \subseteq V^*v\} .$$

Lemma 4.2

$$f_W^*(u) = \{u\} \cap P(W) \cup f_W^*(f_W(u)) .$$

Given a set of strings W there is a unique rooted labelled tree T_W associated with W having the following properties.

1. Each node n has at most $|V|$ sons, the branch from n to each of its sons being labelled with a unique letter $a \in V$.

2. For each word $w_i \in W$ there is a node n of the tree such that the path from the root to n spells out w_i.

3. If n is a leaf of the tree, the path from the root to n spells out some $w_i \in W$.

Fig.7(a) shows the tree associated with W = {abc,bc,bda}. The construction of T_W from W is elementary [1] and we shall not discuss it further.

We shall call any tree with the properties 1., 2. and 3. a V-tree.

If T is a V-tree we can identify T with the function

$$T: N \times V \rightarrow N \cup \{NIL\}$$

where N is the set of nodes of T, n_o is the root of T and, for each $n \in N$ and $a \in V$,

$$T(n,a) = \begin{cases} n^1 & \text{where } n^1 \text{ is the a-son of n, if it exists} \\ NIL & \text{otherwise} \end{cases}$$

It is useful to extend the domain of T to $(N \cup \{NIL\}) \times V^*$ by defining

$$T(NIL,u) = NIL \quad \text{for all } u \in V^*$$

$$T(n,e) = n \quad \text{for all } n \in N$$

$$T(n,ua) = T(T(n,u),a) \quad \text{for all } n \in N, u \in V^*, a \in V.$$

T also defines a (1-1) function $w_T: N \rightarrow V^*$ where, for each $n \in N$,

$$w_T(n) = u \iff T(n_o,u) = n.$$

As was the case with a single string Z, given a set of strings W we can use the failure function for W to construct a recogniser of V*W. The main components of this recogniser are the V-tree associated with W and a set of e-arcs from nodes of the V-tree to nodes of the V-tree. Construction 2 below defines a slightly augmented form of this recogniser.

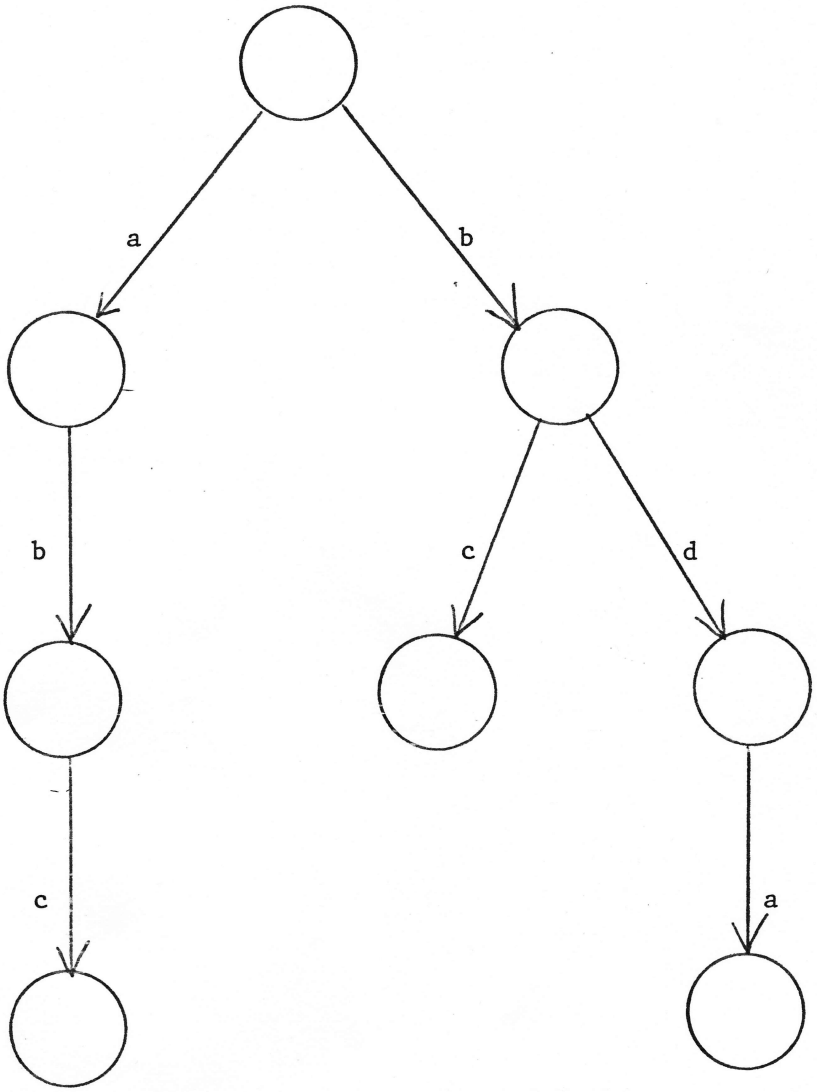


FIG. 7(a)

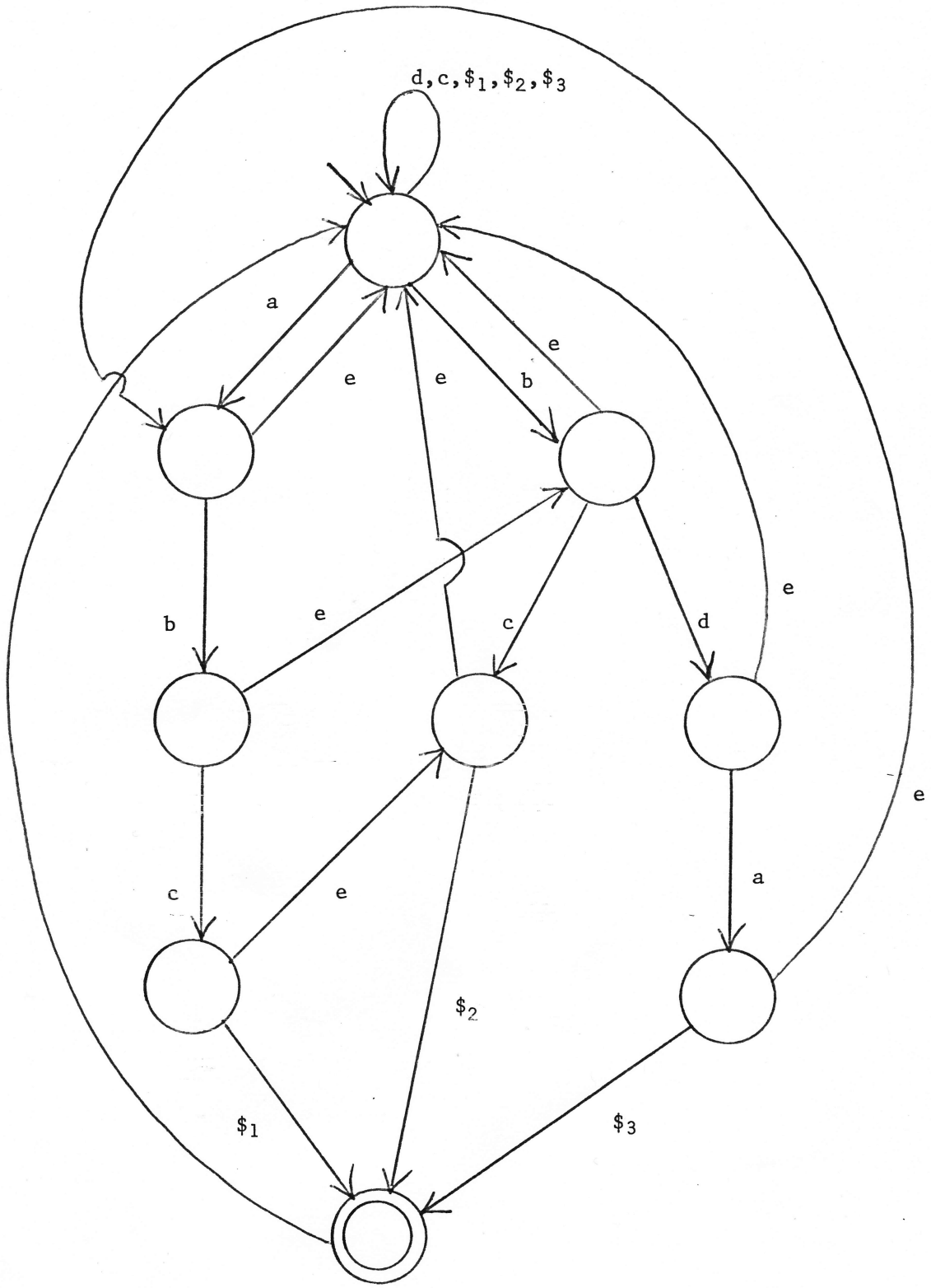


FIG. 7(b) Factor Graph of I*(abc $1 ∪ bc $2 ∪ bda $3).

Construction 2

Let $W = \{w_1, w_2, \ldots, w_k\}$ be a set of strings over the alphabet V. Let $\$_1, \$_2, \ldots \$_k$ be new symbols not in V. Let $I = V \cup \{\$_1, \$_2, \ldots, \$_k\}$. The following algorithm constructs a transition diagram G(W) which we shall show is the factor graph of $I^*(w_1\$_1 \cup w_2\$_2 \cup \ldots \cup w_k\$_k)$.

Step 1

Construct the V-tree T associated with W. (The arcs of T are directed from father to son).

Step 2

For each $u \in V^*$ such that $w_T(n) = u$ for some node n of T, calculate f(u) [1]. Add an arc labelled e from n to n^1 where $w_T(n^1) = f(u)$.

Step 3

Add a new node m to the transition diagram. Add an arc labelled $\$_i$ from the node n, such that $w_T(n) = w_i$, to the new node m.

Step 4

Add an arc labelled e from the new node m to the root n_o of T. For each $a \in I$ such that n_o does not have an a-son add an arc labelled a from n_o to itself.

Fig.7(b) shows the transition diagram so constructed for W = {abc,bc,bda}.

Theorem 4.3

The transition diagram G(W) constructed as above is the (all-admissible) factor graph of $I^*(w_1\$_1 \cup w_2\$_2 \cup \ldots \cup w_k\$_k)$.

The proof of theorem 4.3 proceeds along much the same lines as the proof of theorem 3.4 and so we shall not present it in its entirety.

The generalisation of lemma 3.3 is the only non-straightforward aspect of the proof - the appropriate lemma and its proof are as follows.

Lemma 4.4

Let $Q_i = I^*w_i$, $1 \leq i \leq k$, and let $Q = Q_1\$_1 \cup Q_2\$_2 \cup \ldots \cup Q_k\$_k$. Then L is a left factor of Q <=> L is a left factor of at least one Q_i or L=Q.

Proof

<=. Q is a left factor of Q. Q_i is also a left factor of Q since Q_i is maximal in the subfactorization $Q_i\$_i \subseteq Q$. If L is a left factor of Q_i then L is maximal in some factorization $L.R \subseteq Q_i$ of Q_i. But then L is also maximal in the factorization $L.R\$_i \subseteq Q$. I.e. L is a left factor of Q. (Note: this last result is a particular case of the important observation due to Conway [6] that factors of factors of Q are factors of Q, left factors of left factors of Q are left factors of Q and right factors of right factors of Q are right factors of Q).

=>. $L = \phi$ is a left factor of Q and also of each Q_i, $1 \leq i \leq k$ (lemma 3.3). Let $L \neq \phi$ be a left factor of Q and suppose R is the corresponding right factor.

Suppose, firstly, that $e \in R$. Then $L.e \subseteq Q$, i.e. $L \subseteq Q$. Let L_i, $1 \leq i \leq k$, be defined by $L_i.\$_i = L \cap I^*\$_i$. Clearly $L = L_1\$_1 \cup L_2\$_2 \cup \ldots \cup L_k\$_k$. Suppose $w \neq e$ and $w \in R$. Then $w = v\$_j$ for some j, $1 \leq j \leq k$. Choose any i such that $L_i \neq \phi$. Then $L_i\$_i v\$_j \subseteq Q$. Hence $w = v\$_j \in Q$. I.e. $R \subseteq Q \cup \{e\}$. But $Q.(Q \cup \{e\})$ is a subfactorization of Q which dominates L.R. Thus L = Q.

Now suppose $e \notin R$. $R \neq \phi$ since $I^*.Q \subseteq Q$ dominates $L.\phi \subseteq Q$. Let $R_i = R \cap I^*\$_i$. Then $R = R_1 \cup R_2 \cup \ldots \cup R_k$ and $L.R_i \subseteq Q_i$ is a subfactorization of Q_i in which R_i is maximal. I.e. R_i is a right factor of Q_i and $L \subseteq L_i$ where L_i is the left factor of Q_i corresponding to R_i. Therefore $L \subseteq \bigcap_{1 \leq i \leq k} L_i$. But $(\bigcap_{1 \leq i \leq k} L_i).(R_1 \cup \ldots \cup R_k) \subseteq Q$. Therefore $L = \bigcap_{1 \leq i \leq k} L_i$ by the maximality of L. Now define two subsets P_1 and P_2 of $\{1,\ldots,k\}$ by

$$P_1 = \{i \mid L_i = I^*\}$$

$$P_2 = \{i \mid L_i \neq I^*\}$$

We have two cases to consider: (1) $P_2 = \phi$, (2) $P_2 \neq \phi$.

(1) In this case $L = I^*$ which is a left factor of Q_i for all i, and we have no more to prove.

(2) $P_2 \neq \phi$. Let $j \in P_2$. By lemma 3.3, $L_j = I^*u_j$ for some $u_j \in V^+$. Let $m \in P_2$ be defined by

$$\text{length}(w_m) = \max_{j \in P_2} \text{length}(w_j)$$

Let $v \in L$ and $j \in P_2$. Then $v \in L_j \cap L_m$ and hence $v = \alpha.w_j = \beta w._m$ for some $\alpha, \beta \in I^*$. $\text{Length}(w_j) \leq \text{length}(w_m)$ by definition. Therefore w_j is a suffix of w_m and consequently $L_j \supseteq L_m \therefore L = \bigcap_{\substack{1 \leq j \leq k, \\ j \in P_2}} L_j = L_m$.

I.e. L is a left factor of Q_m.

Corollary

The left factors of Q are $\{\phi, Q\} \cup \{I^*w_T(n) \mid n \text{ is a node of the V-tree of W}\}$.

As mentioned earlier, the remainder of the proof of theorem 4.3 is a straightforward application of the techniques used in the proof of theorem 3.4.

5. Bi-trees

The failure function method of solving the string-matching problem is quite easy to understand. Weiner's bi-tree method [8] is quite difficult to understand - that is, we found it so. The following observations may help the reader to understand Weiner's [8] and McCreight's [8] algorithms for constructing prefix trees. Firstly a number of definitions are in order.

Definitions

Let $Z = z_1 z_2 \ldots z_r \in V^*$. We shall assume (without loss of generality) that the symbol z_r does not occur elsewhere in Z, i.e. $z_i \neq z_r$ for all $i \neq r$. The string $u \in V^*$ identifies position i of Z if $Z = yuv$, $\text{length}(y) = i-1$ and Z cannot be written as y^1uv^1 unless $y^1=y$. The shortest string which identifies position i of Z is called the position identifier of position i of Z and is denoted $p_Z(i)$.

Let $S = \{p_Z(i) \mid 1 \leq i \leq \text{length}(Z)\}$. The V-tree P of S is called the prefix-tree of Z. Weiner [8] has shown that a second tree can be associated with Z. We shall call this tree the auxiliary prefix tree of Z and denote it by A. Some properties of A are summarised in the following theorem.

Theorem 5.1

(i) A is a V-tree having the same nodes as P.

(ii) For all nodes n and n^1 of A and all $a \in V$, $A(n,a) = n^1 \iff w_p(n^1) = au$ and $w_p(n) = u$ for some $u \in V^*$.

The following lemma is immediate from the definition of the failure function and theorem 5.1 (ii).

Lemma 5.2

$w_P(n) = u$ and $A(n,a) = n^1$

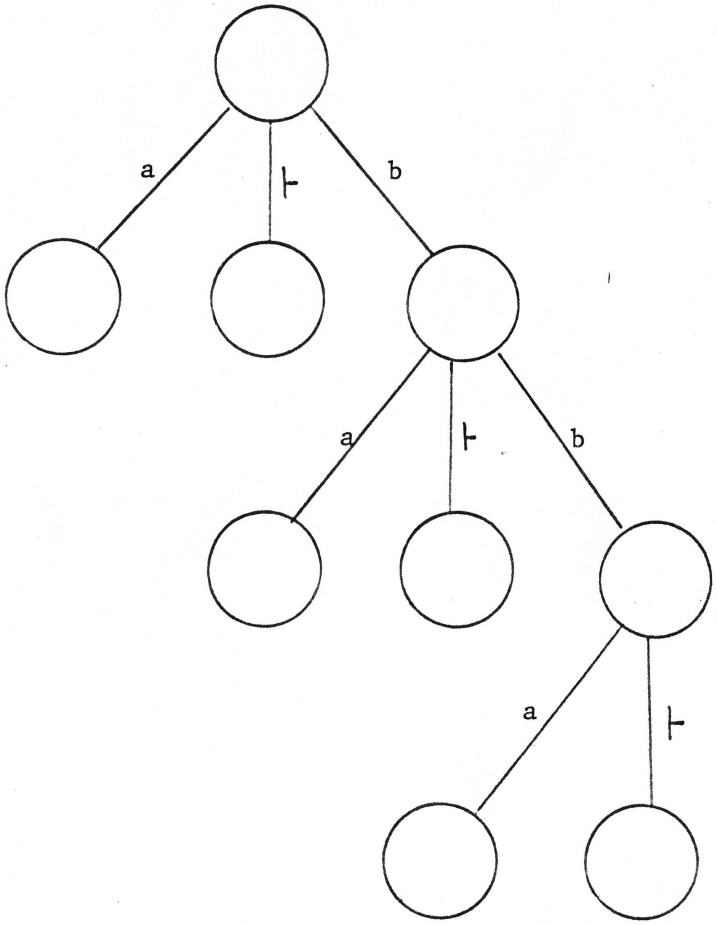
<=> $f_S(w_P(n^1)) = u$.

In other words the auxiliary prefix-tree is a form of "inverse" failure function.

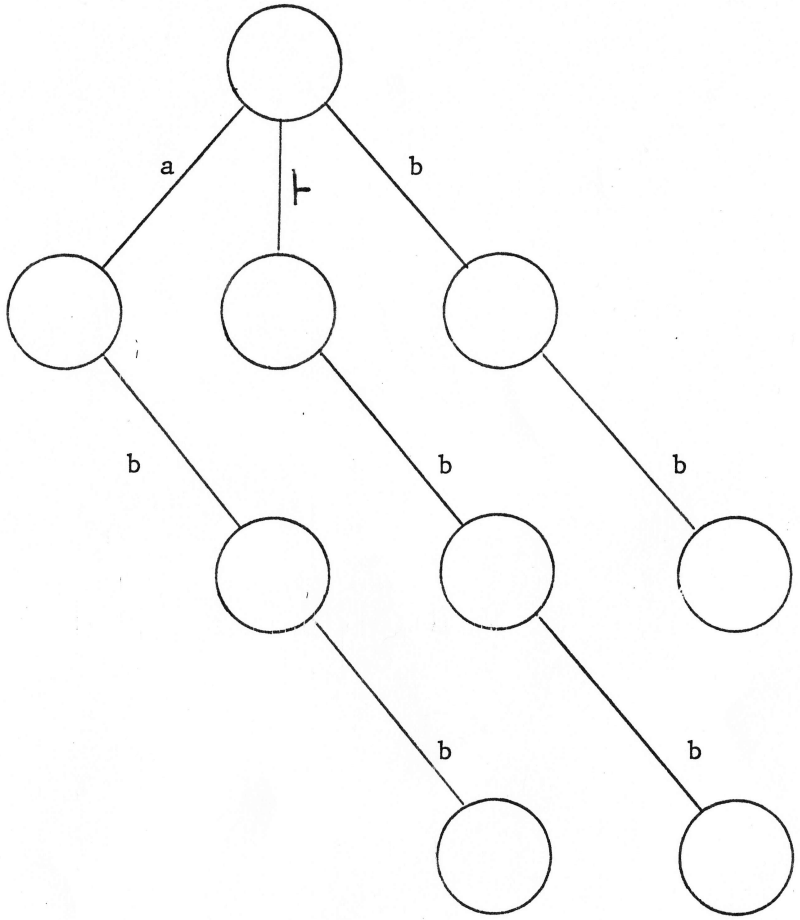
Example 3

An example should suffice to illustrate the relationship between the bi-tree and factor graphs. Figs. 8(a) and (b) show the prefix tree and auxiliary prefix tree for the string bbabb⊢. This string has prefix identifiers {bba, ba, a, bb⊢, b⊢,⊢}. The factor graph of $\{a,b,\vdash,\$_1,\$_2,\$_3,\$_4,\$_5,\$_6\}^*.\{bba\$_1,ba\$_2,a\$_3,bb\vdash\$_4,b\vdash\$_5,\vdash\$_6\}$ is shown in fig. 9. Fig. 9(a) shows the linear part and fig. 9(b) shows the constant part of the factor graph.

FIG. 8 Z = bbabb⊢

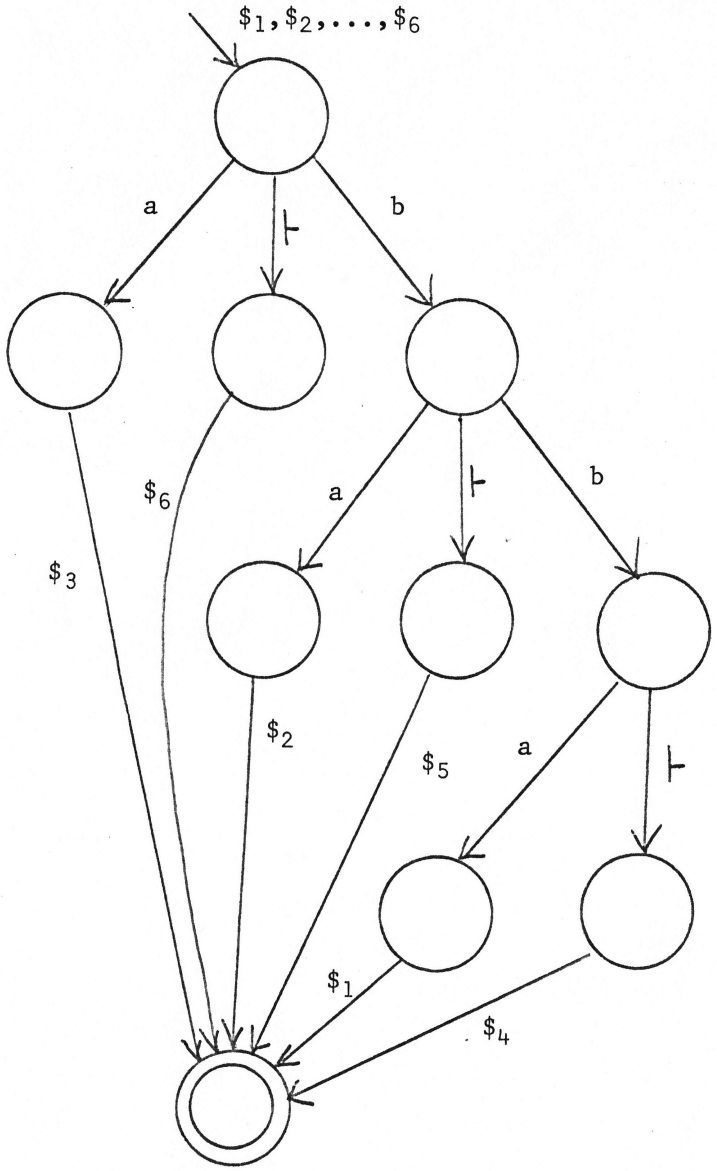


(a) Prefix tree

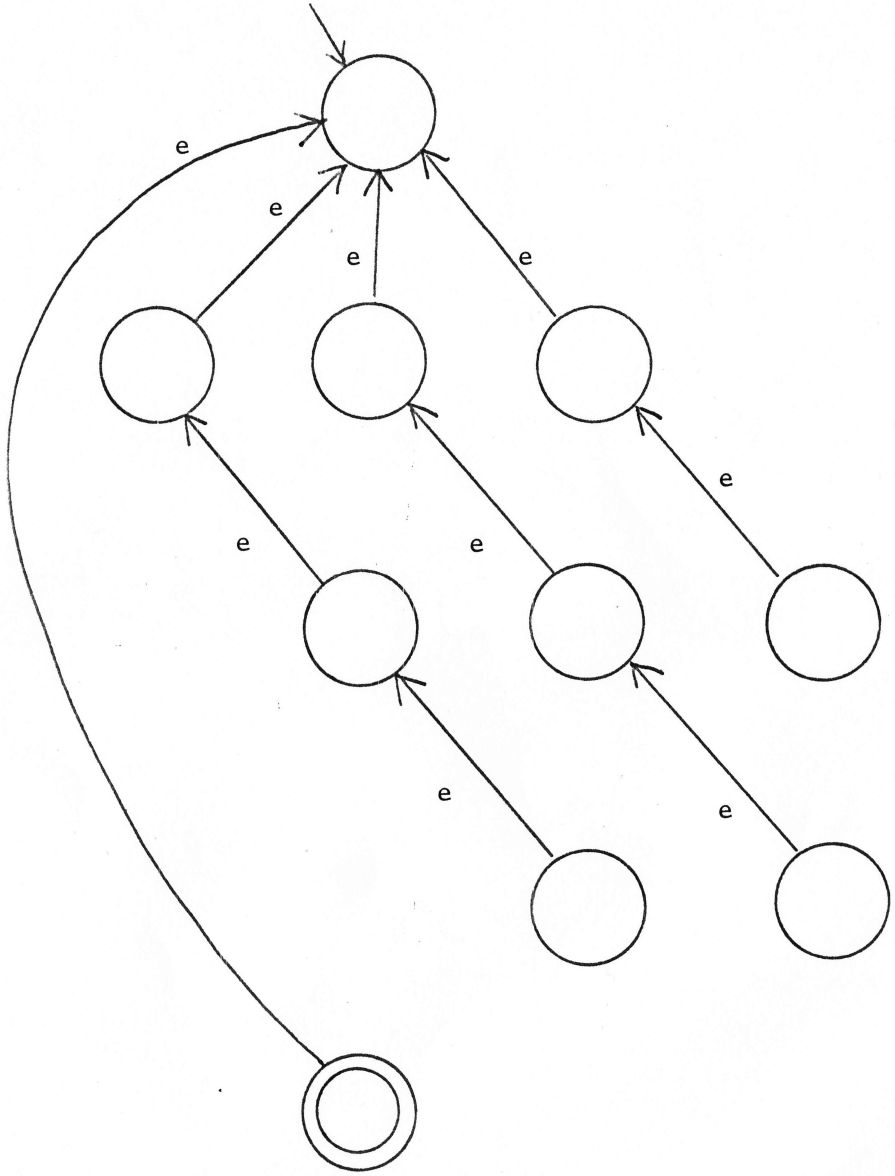


(b) Auxiliary prefix tree

FIG. 9 Factor graph of $Q = \{a,b,\vdash,\$_1,\$_2,\$_3,\$_4,\$_5,\$_6\}^*\{bba\$_1, ba\$_2, a\$_3, bb\vdash\$_4, b\vdash\$_5, \vdash\$_6\}$



(a) L_{min}



(b) C_{min}

6. Conclusions

The previous sections show an unequivacable correlation between the factor graph and various pattern-matching algorithms. As yet we have been unable to provide a satisfactory explanation for the correlation. Conway [6] introduced the concept of a factor as an extremely elegant technique for finding approximations to regular languages and for establishing whether an operator is regularity perserving. Conway's work appears not to be well-known, so if we have brought it to the attention of other automata theorests we will have achieved a lot.

At the time of writing (October, 1976) we have spent only a short time exploring the potential of a factor graph. Preliminary results on pattern-matching with don't cares [9] are both positive and negative. The size of the factor graph of a language of the form $V^*w_1Vw_2V...Vw_k$ can explode exponentially with the size of the regular expression, yet it can always be "condensed" to a graph of linear size and this graph yields the correct value of the failure function (unlike the method used in the absence of don't cares [9]). We therefore hope to be able to report on this work in the near future.

References

1. Aho, A.V. and Corasick, M.J.
"Efficient string matching: An aid to bibliographic search"
C.A.C.M. (June 1975) 18, 6, Pp. 333-340.

2. Aho, A.V., Hopcroft J.E. and Ullman, J.D.
"The design and analysis of computer algorithms"
Addison-Wesley: Reading, Mass. (1974).

3. Backhouse, R.C.
"Closure algorithms and the star-height problem of regular languages"
Ph.D. Thesis, Univ. of London, Sept. 1975.

4. Brzozowski,J.
"Derivatives of regular expressions"
J.A.C.M. 11, 4 (Oct. 1964) Pp. 481-494.

5. Brzozowski, J.
"Roots of start events"
J.A.C.M. 14, 3 (July 1967) Pp. 466-477.

6. Conway, J.H.
"Regular algebra and finite machines"
Chapman and Hall: London (1971).

7. Knuth, D.E., Morris J.H. and Pratt, V.R.
"Fast pattern matching in strings"
TR CS-74-440, Stanford Univ. Stanford, California, 1974.

8. McCreight, E.M.
"A space-economical suffix tree construction algorithm"
J.A.C.M. 23, 2 (April 1976) Pp. 262-272.

9. Fischer, M.J. and Paterson, M.S.
"String-matching and other products"
SIAM-AMS Proc. 7 (1974) Pp. 113-125.

10. Rabin, M.D. and Scott, D.
"Finite automata and their decision problems"
IBM J. Research and Development 3 (1959) Pp. 114-125.

11. Weiner, P.
"Linear pattern matching algorithms"
Conf. Record IEEE 14th Annual Symposium on Switching and Automata Theory (1973) Pp. 1-11.