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 starting 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.
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. ε 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 ε a **constant matrix** and a matrix all of whose entries are subsets \{a_1, a_2, ..., 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 \[ \bigcup_{s \in S} G_s^* \bigcup_{t \in T} G_t^* \]. 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. **Factroy 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 \( L, c \) and \( r \) classes

This section defines the \( L \), \( c \) 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 \( V^* \) - \( Q_L \), \( Q_r \) and \( Q_c \) - given by:

\[
xQ_Ly \iff (\forall z \in V^*, zxQ \iff zyQ)
\]

\[
xQ_r y \iff (\forall z \in V^*, zxQ \iff yzQ)
\]

\[
xQ_c y \iff (\forall u, v \in V^*, uxvQ \iff uyvQ).
\]

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 $\iff$ the relation $Q_L$ is of finite index $\iff$ the relation $Q_R$ is of finite index $\iff$ 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 $\hat{Q}$, where $\hat{Q}$ 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 $L_1$ 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 $L_1$.

(b) If $x$ 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 $a \in V$, 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 $c_i$ 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 $c_i$.

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 $x \in V^*$ partitions $V^*$ into two sets, denoted $D_xQ$ and $\sim D_xQ$, where

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

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

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

6.

We then have:

**Lemma 2.3**

\[ x \not{=} y \Leftrightarrow D_x^Q = D_y^Q \]

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

Similarly the relation \( Q \) leads one to define anti-derivatives:

The anti-derivative of \( Q \) with respect to \( x \), denoted \( a_x^Q \) is

\[ a_x^Q = \{ y \mid xy \in Q \} = \{ y \mid yx \in Q \} \]

**Lemma 2.4**

\[ x \not{=} y \Leftrightarrow a_x^Q = a_y^Q \]

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

\[ C_x^Q = \{ (u,v) \mid uv \in Q \} \]

**Lemma 2.5**

\[ x \not{=} y \Leftrightarrow C_x^Q = C_y^Q \]

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 \( D_x^Q \supseteq (y) \) if and only if \( xyeQ \).

(b) The reverse of anti-derivatives of \( Q \), i.e. languages of the form \( a_y^Q \), are unions of \( r \)-classes of \( Q \), where \( a_y^Q \supseteq (x) \) if and only if \( xyeQ \).
(c) The contexts $C_xQ$ of a language $Q$ are unions of subsets of $V^* \times V^*$ of the form $L(x) \times r$, 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 $y \in D_xQ \iff xy \in Q \iff \tilde{x} \in Q_y$.

But by lemma 2.4, $D_xQ = \bigcup_{y \in Q} y$ for all $y \in Q$. Thus $y \in D_xQ \iff y^1 \in D_xQ$ for all $y^1 \in Q$. Hence $D_xQ = \bigcup_{y \in D_xQ} r(y)$, and part (a) is proved.

Part (b) is proved similarly.

Consider now $C_xQ$. The pair $(u,v) \in C_xQ \iff u^1xv \in Q$.

Let $v \in D_uQ$ and $\tilde{u} \in Q \leftarrow x$. Then $u^1xv \in Q$. 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 $v^1 \in r(v)$.

Hence $C_xQ = \bigcup_{(u,v) \in C_xQ} L(u) \times r(v)$, 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 $F, G, H, \ldots, K, Q$ 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 ⊆ Q.  

(*)

\( F.G...H...J.K \) dominates it if it is also a subfactorization of Q and \( F \subseteq F, G \subseteq G, \ldots, K \subseteq K \). 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 \( \phi \) (the empty set) or are unions of \( \mathcal{L} \)-classes of Q. The right factors of Q are either \( \phi \) or are unions of r-classes of Q and the factors are \( \phi \) 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 $L \neq \emptyset$, 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 $L = \bigcup_{x \in L} \mathcal{L}(x)$, 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 $H \neq \emptyset$, let $x \in H$. Then the set $C_x Q = \{ (u,v) | u x v \in Q \}$ $\supseteq L x R = \{ (u,v) | u \in L, v \in R \}$. But if $y \in c(x)$, $C_y Q = C_x Q \supseteq L x R$. Thus, as above, $y \in H$ and $H = \bigcup_{x \in H} c(x)$.

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.
10.

(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 $q \times q$ matrix which is called the factor matrix of $Q$ and is denoted $[Q]$.

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 $Q$ if and only if $H$ is some entry $Q_{ij}$ in the factor matrix $[Q]$.

(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) $[Q] = [Q]^*$.

(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 $A_1 \subseteq Q_{i k_1}$, $A_2 \subseteq Q_{k_1 k_2}$, $\ldots$, $A_m \subseteq Q_{k_{m-1} j}$.

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) $Q_{ii} \triangleright e$ and (b) $Q_{ij} = \sum_k Q_{ik} Q_{kj}$.

Part (iii) tells us that the $s$th column of $[Q]$ 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), $[Q] = [Q]^*$, 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.

2.4 The Factor Graph

In this section we shall prove that there is a unique minimal matrix $G_Q$ such that $\overline{Q} = G_Q^*$. $G_Q$ 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 start root of $A^*$. Our aim is to prove that there is a unique minimal start root of $\overline{Q}$. If $\alpha$ is a regular language it is quite easy to prove that there is always a unique minimal start root of $\alpha^*$ (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 start 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 start root of $M$. 


In order to prove that \( Q \) nevertheless does have a unique minimal starth root we first prove that \( Q \) 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)**

\[
\exists \text{ unique maximal constant and linear matrices } C_{\text{max}} \text{ and } L_{\text{max}}
\]
such that \( Q = (C_{\text{max}} + L_{\text{max}})^* \).

**Proof**

\( C_{\text{max}} \) and \( L_{\text{max}} \) are defined to be the unique maximal constant and linear matrices (respectively) such that \( Q \supset C_{\text{max}} \) and \( Q \supset L_{\text{max}} \).

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 \( M_n(R) \). Let \( M_n(R) \) have unit element \( E \).

Let \( [B\setminus C]_{ij} = [b_{ij} \setminus c_{ij}] \) where \( \setminus \) denotes set difference.

If \( A^* = \{(A \setminus E) \setminus (A \setminus E)^{2+*}\}^* \) then \( (A \setminus E) \setminus (A \setminus E)^{2+*} \) is the unique minimal starth root of \( A^* \), where \( X^{2+*} = X^2.X^* \).

**Proof**

Let \( X = (A \setminus E) \setminus (A \setminus E)^{2+*} \). By assumption \( X \) is a starth root of \( A^* \).

Suppose \( Y \) is also a starth root. We must show that \( X \subset Y \).

Suppose \( w \in X_{ij} \).

Clearly \( w \in Y_{ij} = [(Y \setminus E)^*]_{ij} \), because \( Y \) is a starth root of \( A^* \) and \( A^* \supset X \).

Hence \( w \in [(Y \setminus E)^{n}]_{ij} \) for some \( n \) where, by definition of \( X \), \( n \geq 1 \).

Now \( Y \subset A^* = (A \setminus E)^* \). Hence \( Y \setminus E \subset (A \setminus E)^+ \).
\[ w \in [(\lambda E)^n]_{ij} \subseteq [(\lambda E^+)^n]_{ij}. \]

But
\[ w \in X_{ij} = [(\lambda E)(\lambda E)^{2+}]_{ij} \]
\[ \Rightarrow n = 1 \]
\[ \Rightarrow w \in (Y \setminus E)_{ij} \subseteq Y_{ij}. \]

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

Considering the matrix \( M \) mentioned at the beginning of this section, we find that
\[
(\lambda E)(\lambda E)^{2+} = \begin{bmatrix}
\phi & \phi & \phi \\
\phi & \phi & \phi \\
\phi & \phi & \phi
\end{bmatrix}
\]

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**

\( C_{\text{max}} \setminus E \) is acyclic.

**Proof**

Suppose \( C_{\text{max}} \setminus E \) is cyclic. Then there must be two distinct nodes \( i \) and \( j \) such that \( [C_{\text{max}}]_{ij} = e = [C_{\text{max}}]_{ji} \). Now consider the matrix \( [Q] \).

We have \( [Q] = [Q]^* \), (2.10 (iv)), and \( [Q] \geq [C_{\text{max}}] \), by Theorem 2.11.

Therefore \( [Q] = [Q] \cdot [Q] \geq [Q] \cdot C_{\text{max}} \).

Hence \( Q_{si} \geq Q_{sj} \cdot [C_{\text{max}}]_{ji} = Q_{si} \)
and \( Q_{sj} \geq Q_{si} \cdot [C_{\text{max}}]_{ij} = Q_{sj} \).
Therefore \( Q_{si} = Q_{sj} \), i.e. \( L_i = L_j \) (by 2.10 (iii)).

Similarly, using \( |Q| \geq C_{\text{max}} \cdot |Q| \), 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 \( C_{\text{max}} \setminus E \) must be acyclic.

**Corollary 2.14**

Let \( |Q| \) be a \( q \times q \) matrix. Then \( (C_{\text{max}} \setminus E)^P = N \) 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 \( C_{\text{max}} \) and \( L_{\text{max}} \) be as defined in Theorem 2.11. Then there is a unique minimal matrix \( G_Q \) such that \( G_Q^* = |Q| \), given by \( G_Q = ((C_{\text{max}} + L_{\text{max}}) \setminus E)^* ((C_{\text{max}} + L_{\text{max}}) \setminus E)^{2**} \). 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 \( G_Q^* = |Q| \) where

\[
G_Q = ((C_{\text{max}} + L_{\text{max}}) \setminus E)^* ((C_{\text{max}} + L_{\text{max}}) \setminus E)^{2**}.
\]

In turn this only requires proving that \( G_Q^* \geq (C_{\text{max}} + L_{\text{max}}) \setminus E \), since then \( G_Q^* = (G_Q^*)^* \geq (C_{\text{max}} + L_{\text{max}})^* = |Q| \), by Theorem 2.11.

Let \( p_1 : w \in [(C_{\text{max}} + L_{\text{max}}) \setminus E]_{ij} \).

Then \( p_2 : w \) has length 0 or 1.

Suppose \( p_3 : w \notin [G_Q^*]_{ij} \).

Then \( p_4 : w \notin [G_Q]_{ij} \)

and so \( p_5 : w \in [(C_{\text{max}} + L_{\text{max}})^{2**}]_{ij} \).
Hence indices \( i = k_1, k_2, \ldots, k_{m+1} = j \) and words \( a_1, a_2, \ldots, a_m \) s.t. \( m \geq 2, w = a_1a_2 \ldots a_m \).

and 

\[ p_1 : \quad a_h \in [(C_{\text{max}} + L_{\text{max}}) \setminus E] k_h k_{h+1} \]

and hence 

\[ p_2 : \quad a_h \text{ has length } 0 \text{ or } 1, \text{ for all } h=1,\ldots,m \] .

Now for some \( h \) we must have 

\[ p_3 : \quad a_h \notin [G^*]_{Q} k_h k_{h+1} \]

(otherwise \( w \in [G^*]_{Q} \).)

Hence for this \( h \)

\[ p_4 : \quad a_h \notin [G^*]_{Q} k_h k_{h+1} \]

and so, for this \( h \)

\[ p_5 : \quad a_h \in \left( (C_{\text{max}} + L_{\text{max}}) \setminus E \right)^{2\times} k_h k_{h+1} . \]

Let this \( a_h \) be \( \overline{v} \). \( v \) has the same properties as \( w \) and so can in turn be expressed as the product of two or more words \( b_1b_2 \ldots b_n \), where by the same argument one of the \( b \)'s = \( u \), say, also has the same properties as \( w \). In this way we can express \( w \) as a product.

\[ w = y_1y_2 \ldots y_x \]

of an unbounded number \( x \) of words \( y_f \), where

\[ y_f \in [(C_{\text{max}} + L_{\text{max}}) \setminus E] n_f n_{f+1} \]

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 \( (C_{\text{max}} \setminus E)^P \) is non-null for all \( p \). But this contradicts corollary 2.15. Hence the initial assumption that property \( p_3 \) holds for \( w \) must be false. Hence \( G^* \geq (C_{\text{max}} + L_{\text{max}}) \setminus E \), and by our earlier argument \( G^*_Q = [Q] \). Finally, applying Theorem 2.12, \( G_Q \) is the minimal starth root
of $|Q|$. The last part of the theorem follows immediately from Theorem 2.10 (iii), $Q = Q_{st}^6 = [G^6]^st$.

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) $l_1, l_2, \ldots, l_m$ ($r_1, r_2, \ldots, r_m$) 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 $l_i$th row of Table 1 if and only if $x_i y_j \in Q$, and similarly $l_i$ appears as a term in the $r_j$th 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.

### Table 1(a) Machine

<table>
<thead>
<tr>
<th>Node/(l)-class</th>
<th>Representative element</th>
<th>Derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>(l_1)</td>
<td>e</td>
<td>(r_1 + r_2 + r_3)</td>
</tr>
<tr>
<td>(l_2)</td>
<td>a</td>
<td>(r_2 + r_3)</td>
</tr>
<tr>
<td>(l_3)</td>
<td>aa</td>
<td>(r_3)</td>
</tr>
</tbody>
</table>

### Table 1(b) Anti-machine

<table>
<thead>
<tr>
<th>Node/(r)-class</th>
<th>Representative element</th>
<th>Reverse of anti-derivative</th>
</tr>
</thead>
<tbody>
<tr>
<td>(r_1)</td>
<td>e</td>
<td>(l_1)</td>
</tr>
<tr>
<td>(r_2)</td>
<td>b</td>
<td>(l_1 + l_2)</td>
</tr>
<tr>
<td>(r_3)</td>
<td>bb</td>
<td>(l_1 + l_2 + l_3)</td>
</tr>
<tr>
<td>(r_4)</td>
<td>a</td>
<td>(\phi)</td>
</tr>
</tbody>
</table>
The correctness of step 2 is based on theorem 2.6.

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

the derivative corresponding to \( L_2 \) is \( r_2 + r_3 \).

**Step 3**

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

To do this, one considers all subsets \( \{ L_{i_1}, ..., L_{i_k} \} \) (including the empty subset) of the \( L \)-classes of \( Q \) and finds for each subset those classes \( r_{j_1}, ..., r_{j_n} \) 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 \((L_{i_1} + ... + L_{i_k})(r_{j_1} + ... + r_{j_n}) \leq Q \) is a subfactorization of \( Q \) in which \((r_{j_1} + ... + r_{j_n}) \) is maximal. By inspecting all such subfactorizations one may deduce the left and right factors. \( L_t = L_{t_1} + L_{t_2} + \ldots + L_{t_k} \) is that left factor such that the \( L \)-class \( L_{i_1} \subseteq L_t \) 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 \( r_{s_1} \subseteq R_s \) if and only if it corresponds to a terminal node of the anti-machine for \( Q \).
20.

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

\[
(\ell_1 + \ell_2 + \ell_3) \cdot r_3, \quad (\ell_2 + \ell_3) \cdot r_3, \quad (\ell_1 + \ell_3) \cdot r_3, \quad \ell_3 \cdot r_3,
\]

\[
(\ell_1 + \ell_2) \cdot (r_2 + r_3), \quad \ell_2 \cdot (r_2 + r_3), \quad \ell_1 \cdot (r_1 + r_2 + r_3), \quad \phi \cdot (r_1 + r_2 + r_3 + r_4).
\]

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.

<table>
<thead>
<tr>
<th>Left Factors</th>
<th>Right Factors</th>
</tr>
</thead>
<tbody>
<tr>
<td>L_t = L_1 \ell_1</td>
<td>R_s = R_1 r_1 + r_2 + r_3</td>
</tr>
<tr>
<td>L_2 \ell_1 + \ell_2</td>
<td>R_2 r_2 + r_3</td>
</tr>
<tr>
<td>L_3 \ell_1 + \ell_2 + \ell_3</td>
<td>R_3 r_3</td>
</tr>
<tr>
<td>L_4 \phi</td>
<td>R_4 r_1 + r_2 + r_3 + r_4</td>
</tr>
</tbody>
</table>

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.

Step 4

Calculate \( C_{\text{max}} \). Whence deduce

\[
C_{\text{min}} = (C_{\text{max}} \backslash \text{E}) \backslash (C_{\text{max}} \backslash \text{E})^{2+K}.
\]
\[ [C_{\text{max}}]_{ij} \geq e \text{ if and only if } L_j \supseteq L_i, \text{ and this can easily be deduced from the representation of } L_i \text{ and } L_j \text{ as unions of } L\text{-classes of } Q. \]

**Step 5**

Calculate \( L_{\text{max}} \). Whence deduce

\[ L_{\text{min}} = L_{\text{max}} \setminus ((C_{\text{max}} + L_{\text{max}}) \setminus E)^{2+}. \]

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

Finally \( G_Q = C_{\text{min}} + L_{\text{min}} \).

Steps 4 and 5, applied to our example, are summarised in figs. 2-5, below.
Fig. 3 - $C_{\text{min}}$

Fig. 4 - $L_{\text{max}}$
Fig. 5 The Factor Graph $G_Q$

The construction of $C_{\text{max}}$ and $C_{\text{min}}$ should be clear. For instance there is an arc from node 1 to nodes 2 and 3 in $C_{\text{max}}$ since

$L_1 = L_1 \leq L_2 = L_1 + L_2 \leq L_3 = L_1 + L_2 + L_3$. There is no arc from 1 to 3 in $C_{\text{min}}$, since there are arcs from 1 to 2 and from 2 to 3. The construction of $L_{\text{max}}$ is slightly more complicated. As an example, there is an arc labelled $b$ from node 2 to nodes 2 and 1 since $L_2.b = (L_1 + L_2).b$ 

$= L_1.b + L_2.b = L_1 + L_1$ (from the machine in fig. 1(a)), and $L_1 \leq L_2 \leq L_1$. 
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 ≥ 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 $L$- 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 $\phi$ 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 $f_Z^*: \{1 \ldots r\} \rightarrow 2^{\{0 \ldots r\}}$ is defined by:

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

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

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

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 \( Z = \text{aabbaab} \) are given in Table 2.

<table>
<thead>
<tr>
<th>( i )</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>( z_i )</td>
<td>a</td>
<td>a</td>
<td>b</td>
<td>b</td>
<td>a</td>
<td>a</td>
<td>b</td>
</tr>
<tr>
<td>( f^*(i) )</td>
<td>{0,1}</td>
<td>{0,1,2}</td>
<td>{0,3}</td>
<td>{0,4}</td>
<td>{0,1,5}</td>
<td>{0,1,2,6}</td>
<td>{0,3,7}</td>
</tr>
<tr>
<td>( f(i) )</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Table 2

The importance of the failure function to the string-matching problem lies in the fact that it may be computed in time \( O(r) \). 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, \ldots, \(r\).

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 \(z_1z_2\ldots z_{r+1}\). 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**

\[
\begin{align*}
\mathbf{f}^*(i) &= \{j \mid z_1 \ldots z_i \in V^* z_1 \ldots z_j\} \\
&= \{j \mid V^* z_1 \ldots z_i \subseteq V^* z_1 \ldots z_j\}.
\end{align*}
\]

**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 \(z_1 \ldots z_i = wz_1 \ldots z_j\) for some \(w \in V^*\).

I.e. \(z_{i-j+1} \ldots z_i = z_1 \ldots z_j\). Also \(i \geq j\) and \(z_{i-j+1} \ldots z_i = z_1 \ldots z_j\) \(\Rightarrow V^* z_{i-j+1} \ldots z_i = V^* z_1 \ldots z_j \subseteq V^* z_1 \ldots z_i\). Hence

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

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

\[ \iff j \in f(i) \quad \text{and} \quad z_1 \ldots z_j = z_{i-f(i)+1} \ldots z_i \]

(by the definition of $f(i)$)

\[ \iff j \in f(i) \quad \text{and} \quad z_1 \ldots z_j = z_{f(i)-j+1} \ldots z_{f(i)} \]

\[ \iff j \in f^*(f(i)). \]

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

**Lemma 3.3**

$Q$ has $r+2$ left factors, namely $L_0 = V^*$, $L_1 = V^*z_1$, $L_2 = V^*z_1z_2$, $\ldots$, $L_r = V^*z_1z_2 \ldots z_r$ and $L_{r+1} = \phi$.

**Proof**

$\phi$ 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 $V^*.z_1z_2 \ldots z_r \subseteq Q$.

Let $w$ be a word of shortest length in $R$. There are two cases to consider.

**Case 1**

$\text{length}(w) \geq r$. In this case $R \subseteq V^*z_1z_2 \ldots z_r$ and $V^*.V^*z_1 \ldots z_r$ dominates $L.R \subseteq Q$. \( \therefore L = V^*. \)
Case 2

\[ \text{length}(w) < r. \] Let \( \text{length}(w) = r - i, \) where \( 1 \leq i \leq r. \) Then

\[ w = z_{i+1} \ldots z_r \text{ and } L \subseteq V^*z_1z_2 \ldots z_i. \]

Suppose \( L \neq \emptyset. \) Let \( v \in R \) and \( u \in L. \) \( u = u^1z_1 \ldots z_i \) for some \( u^1 \in V^*. \) Also \( \text{length}(v) \geq r - i \) (by the choice of \( i). \) Hence \( u \cdot v \in Q \) implies \( z_1 \ldots z_i \cdot v \in Q. \) I.e. \( z_1 \ldots z_i \in L \) by maximality of \( L. \) But then

\[ z_1 \ldots z_i \cdot R \subseteq Q, \] hence \( V^*z_1 \ldots z_i \cdot R \subseteq V^*Q = Q. \) 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 \( \emptyset \) is a factor of \( Q \) and therefore the factor graph of \( Q \) has one "useless" node. In the proof of theorem 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 \( \|Q\| = (C_{\max} + L_{\max})^*. \)

Since \( V^*z_1 \ldots z_{i-1} \cdot z_i \cdot z_{i+1} \ldots z_r \subseteq Q \) 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

\[ (a) \ C_{\max} = C_{\min}, \quad (b) \ L_{\max} \subseteq (C_{\min} + L_{\min})^*, \]

\[ (c) \ C_{\min} \subseteq (C_{\max} \setminus E) \cap (C_{\max} \setminus E)^{2+*} \quad \text{and} \quad (d) \ L_{\min} \subseteq L_{\max} \setminus ((C_{\max} + L_{\max}) \setminus E)^{2+*}. \]

(a) and (b) establish that \( \|Q\| = G(Z)^*; \) (c) and (d) establish the minimality of \( G(Z). \)
Let $L_i = V^*z_1 \ldots z_i, 1 \leq i \leq r$ and $L_0 = V^*.$

(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 $z_1 \ldots z_i a z_{j+1} \ldots z_r \subseteq L_i \cdot a \cdot R_j \subseteq Q$ (Theorem 2.10) $\Rightarrow i \geq j-1$, $a = z_j$ and $z_1 z_2 \ldots z_{j-1} = z_{i-j+2} \ldots z_i$.
Thus $j-1 \in f^*(i)$, whence $[C^*_{\min}]_{ij-1} = e$ (by (a)), and $[L_{\min}]_{j-1,j} = a$.

$\therefore a \in [C^*_{\min}]_{L_i}$. I.e. $L_{\max} \subseteq (C^*_{\min} L_{\min}^*)^*.$

To prove (c), suppose $e \in [C^*_{\min}]_{ij}$ and $e \in [(C^*_{\min} \setminus E)^{2\#}]_{ij}$. 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. $i \neq k \neq j$, $k \in f^*(i)$ 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 $e \in [C^*_{\max} \setminus E]_{mj} \ldots k < j-1$ and $j < m$. I.e. $k < m-1$. But also $z_1 \ldots z_k z_j z_{m+1} \ldots z_r \subseteq L_k \cdot z_j \cdot R_m \subseteq Q$ (Theorem 2.10) $\Rightarrow k \geq m-1$. 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 $p \cdot m + n_1 + n_2 + \ldots + n_p$, 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 $f^*_W : V^* \rightarrow 2^{V^*}$ 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 suffixes of $u$.

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

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

where $v_{\max} \in f^*_W(u)$, $v_{\max} \neq u$ and length$(v_{\max}) \geq$ length$(v)$ for all $v \neq u$, $v \in f^*_W(u)$. (Clearly $f_W(u)$ is well-defined.)
If \( W = \{ Z = z_1z_2...z_r \} \) is a set containing exactly one word then \( j \geq 1 \) and \( z_1z_2...z_j \in f^*_W(u) \)

\[ \Leftrightarrow u = z_1z_2...z_i \text{ for some } i, 1 \leq j \leq r \]

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**

\[ f^*_W(u) = \{ v : v \in P(W) \text{ and } u \in V^*v \} \]

\[ = \{ v : v \in P(W) \text{ and } V^*u \subseteq V^*v \} \].

**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

\[
T : N \times V \to N \cup \{\text{NIL}\}
\]

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

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

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

\[
T(\text{NIL}, u) = \text{NIL} \text{ for all } u \in V^* \\
T(n, e) = n \text{ for all } n \in N \\
T(n, ua) = T(T(n, u), a) \text{ for all } n \in N, u \in V^*, a \in V.
\]

\( T \) also defines a (1-1) function \( w_T : N \to V^* \) 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(b) Factor Graph of $I^*(abc \$_1 \_bc \$_2 ubda \$_3)$. 
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 $I^*(w_1\$1w_2\$2u\ldots u w_k\$k)$.  

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 $n'$ where $w_T(n') = f(u)$.  

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 $n_0$ of $T$. For each $a \in I$ such that $n_0$ does not have an $a$-son add an arc labelled $a$ from $n_0$ 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 $I^*(w_1\$1w_2\$2u\ldots u w_k\$k)$.  

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 $Q_i = I^*w_i^r$, $1 \leq i \leq k$, and let $Q = Q_1^* \cup Q_2^* \cup \ldots \cup Q_k^*$. Then $L$ is a left factor of $Q$ if and only if $L$ is a left factor of at least one $Q_i$, or $L = Q$.

**Proof**

$\Rightarrow$. $Q$ is a left factor of $Q$. $Q_i$ is also a left factor of $Q$ since $Q_i$ is maximal in the subfactorization $Q_i^* \subseteq Q$. 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 $L.R^* \subseteq Q_i$. 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 $Q$).

$\Rightarrow$. $L = \emptyset$ is a left factor of $Q$ and also of each $Q_i$, $1 \leq i \leq k$ (lemma 3.3).

Let $L \neq \emptyset$ 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 $L_i^* = L \cap I^*$. Clearly $L = L_1^* \cup L_2^* \cup \ldots \cup L_k^*$. 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 $L_i \neq \emptyset$. Then $L_i^* v_j^* \subseteq Q$. 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 $e \notin R$. $R \neq \emptyset$ since $I^*.Q \subseteq Q$ dominates $L.\emptyset \subseteq Q$.

Let $R_i = R \cap I^*$. 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 $L \subseteq L_1 \cap \ldots \cap L_k$. But $(L_1 \cap \ldots \cap L_k). (R_1 \cup \ldots \cup R_k) \subseteq Q$. Therefore $L = L_1 \cap \ldots \cap L_k$ by the maximality of $L$. Now define two subsets $P_1$ and $P_2$ of $\{1, \ldots, k\}$ by
40.

\[ P_1 = \{ i | L_i = I* \} \]
\[ P_2 = \{ i | L_i \neq I* \} \]

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

(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) \( P_2 \neq \emptyset \). 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 \( \alpha, \beta \in I^* \). Length\((w_j) \leq \text{length}(w_m) \) by definition. Therefore \( w_j \) is a suffix of \( w_m \) and consequently \( L_j \supseteq L_m : L = \cap_{1 \leq j \leq k, j \in P_2} L_j = I_m \).

I.e. \( L \) is a left factor of \( Q_m \).

Corollary

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

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 \( Z = z_1z_2\ldots z_r \in V^* \). 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 \( S = \{ p_Z(i) | 1 \leq i \leq \text{length}(Z) \} \). 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

\[ w_p(n) = u \text{ and } A(n, a) = n^1 \]

\[ \iff \]

\[ f_S(w_p(n^1)) = u. \]

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\textasciitilde. This string has prefix identifiers \{bba, ba, a, bb\textasciitilde, b\textasciitilde\}. The factor graph of \{a,b,\textasciitilde,\$1,\$2,\$3,\$4,\$5,\$6 \}^* \{bba$1, ba$2, a$3, bb$4, b$5, \textasciitilde$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.
6. **Conclusions**

The previous sections show an unequivocal 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 preserving. Conway's work appears not to be well-known, so if we have brought it to the attention of other automata theorists 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 $V^*w_1Vw_2V\ldots Vw_k$ 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"

2. Aho, A.V., Hopcroft J.E. and Ullman, J.D.
"The design and analysis of computer algorithms"

3. Backhouse, R.C.
"Closure algorithms and the star-height problem of regular languages"

4. Brzozowski, J.
"Derivatives of regular expressions"

5. Brzozowski, J.
"Roots of star events"

6. Conway, J.H.
"Regular algebra and finite machines"

"Fast pattern matching in strings"

8. McCreight, E.M.
"A space-economical suffix tree construction algorithm"

9. Fischer, M.J. and Paterson, M.S.
"String-matching and other products"

10. Rabin, M.D. and Scott, D.
"Finite automata and their decision problems"

11. Weiner, P.
"Linear pattern matching algorithms"