THEORY OF WORD SPACES AND COMPUTER SCIENCE

Dissertation submitted to
Jawaharlal Nehru University
in partial fulfilment of the requirements
for the award of the Degree of
MASTER OF TECHNOLOGY
in
COMPUTER SCIENCE AND TECHNOLOGY

by
MANOJ JAIN

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI-110 067,
JANUARY 1992

CERTIFICATE

This is to certify that the thesis entitled "THEORY OF WORD SPACES AND COMPUTER SCIENCE", being submitted by me to Jawaharlal Nehru University in partial fulfilment of the requirements for the award of the degree of Master of Technology is a record of original work done by me under the supervision of Prof. K. K. Nambiar, School of Computer and Systems Sciences during the Monsoon semester, 1991.

The results reported in this thesis have not been submitted in part or full to any other University or Institution for the award of any degree etc..

[illegible]

(MANOJ JAIN)

[illegible] 3/1/92

Dr. R. G. Gupta
Professor & Dean,
SCSS, J.N.U.,
New Delhi - 110 067.

[illegible]
3.1.92

Dr. K. K. Nambiar
Professor, SCSS,
J.N.U. ,
New Delhi - 110 067.

To my dear

Mother & Father

ACKNOWLEDGEMENTS

I express my sincere gratitude to my guide Prof. K.K. Nambiar who provided me the necessary insight into the subject and was always ready with his helping hand in case of difficulties. But for his able guidance and constant support, it would not have been possible for me to complete the project. He had the patience of listening to my problems and providing constructive suggestions.

I am thankful to Prof. R. G. Gupta for his inspiration and help. I also thank all my teachers who shared their valuable skills, expertise and time.

Besides this, I can not forget to thank my friends and colleagues for their encouragement, suggestions and worthy discussions. They helped in creating an atmosphere in which I was able to keep up my spirits. Among them, I am very grateful to Vandana Saroch and Pramod Varma for their cooperation and assistance which they rendered to me at the time when I needed it most.

Thanks are also due to everyone who directly or indirectly helped me in the due course of my work. Last but not the least I convey my gratitude to the nonteaching staff of the school.

Finally I dedicate this thesis to my parents whose unbounded afflatus had been a constant source of power, I owe every happy moment of my life to them.

MANOJ JAIN

CONTENTS

CHAPTER	1	INTRODUCTION	1
CHAPTER	2	RECURSIVE FUNCTIONS	3
CHAPTER	3	NU MACHINE	13
CHAPTER	4	NON LINEAR ALGEBRA	29
CHAPTER	5	FINITE AUTOMATA AND WORDSPACE	34
CHAPTER	6	CAYLEY GRAPHS AND THEIR WORDSPACE	40
CHAPTER	7	GRAPHICAL EVALUATION OF DETERMINANTS	44
CHAPTER	8	CONCLUSION	51
		BIBLIOGRAPHY	52

CHAPTER 1

INTRODUCTION

INTRODUCTION

Elegant and extensive theory is available for matrices with elements from a field. Much of the study of engineering and technology is either a direct or indirect investigation of the theory of vector spaces. In the early stages of the development of computers, they were widely used in these applications. However, the situation has considerably changed over a period of time, and today computers are mostly used for business applications.

The data that is used in the business is in the form of tables. These tables may be considered as matrices with elements from an arbitrary set. The study of matrices have paid rich dividends in the last two decades and researches are still going on.

In this thesis we are making an attempt to study the matrices with elements from a loose algebraic structure not as strong as a field. The matrices with regular expressions as their elements are taken up in our work..

We are also introducing Nu Machine which can compute all recursive functions in a mechanical manner. This machine is as powerful as a Turing machine. The algebra for Nu Machine, called non linear algebra, is studied in detail.

An algorithm to find the wordspace of a finite automaton is given. This may be of use in the field of formal languages and other related areas.

We have also investigated Cayley graphs which are seen to be having properties which are closely related to groups.

Finally, we have introduced a graphical method for evaluation of determinants with an example.

CHAPTER 2

RECURSIVE FUNCTIONS

RECURSIVE FUNCTIONS

In this chapter we study an important class of functions and show that any such function can be evaluated in a purely mechanical manner. This type of mechanical procedures are needed if an end result is to be obtained by using a machine.Functions that can be evaluated by mechanical means are said to be "effectively computable" or "simply computable".

This class of functions called **recursive functions** is now taken up here.We restrict ourselves to only those functions whose arguements and values are natural numbers.Such functions are called number-theoretic.In general the number-theoretic functions of n variables which are denoted by $f<x_1,x_2,\ldots\ldots,x_n>$ shall be considered.Any function $f: \mathbf{N}^n \dashrightarrow \mathbf{N}$ is called total because it is defined for every n-tuple in $\mathbf{N}^n$.On the other hand if $f:D \dashrightarrow \mathbf{N}$ where $D \subseteq \mathbf{N}^n$,then f is called partial. Examples these functions are

1 $f<x,y> = x + y$,which is defined for all $x,y \in \mathbf{N}$ and hence is a total function.

2 $g<x,y> = x - y$,which is defined only for those $x,y \in \mathbf{N}$ which satisfy $x \geq y$.Hence $g<x,y>$ is partial.

ELEMENTARY FUNCTIONS

We now give a set of three functions called the **elementary functions** or sometimes **initial functions**, which are used in defining other functions by induction.

$Z:Z(x) = 0$	**zero function**
$S:S(x) = x + 1$	**successor function**
$U_i^n:U_i^n<x_1,x_2,....,x_n>=x_i$	**projection function**

The projetion function is also called the generalized identity function. As examples, we have $U_1^2<x,y> = x$, $U_2^3<2,4,6> = 4$, etc..

ELEMENTARY PROCEDURES

The following are the three elementary procedures.

(i) **Composition :**

The operation of composition is used to generate other functions.

Given a set of m functions $f_1, f_2, ..., f_m$ each of n variables and let g be a function of m variables. Then the composition of g with $f_1, f_2, ..., f_m$ produces a composition function h given by

$$h<x_1,x_2,.....,x_n>=g<f_1<x_1,x_2,.....,x_n>,....,f_m<x_1,x_2,.....,x_n>>$$

As an example, let

$f_1<x,y> = x + y$, $f_2<x,y> = xy + y^2$, $g<x,y> = xy$

Then

$$h<x,y> = g< f_1<x,y>, f_2<x,y> >$$
$$= g< x + y,\ xy + y^2 >$$
$$= (x + y)\ (xy + y^2)$$

(ii) **Recursion :**

The following operation which defines a function $f<x_1, x_2, \ldots\ldots, x_n, y>$ of n +1 variables by using two other functions $g<x_1, x_2, \ldots\ldots, x_n>$ and $h<x_1, x_2, \ldots\ldots, x_n, y, z>$ of n and n + 2 variables, respectively, is called **recursion** .

$$f <x_1, x_2, \ldots\ldots, x_n, 0> = g <x_1, x_2, \ldots\ldots, x_n>$$
$$f<x_1, x_2, \ldots\ldots, x_n, y +1> = h<x_1, x_2, \ldots\ldots, x_n, y , f<x_1, x_2, \ldots\ldots, x_n, y>>$$

In this definition , the variable y is assumed to be the inductive variable in the sense that the value of f at y + 1 is expressed in terms of the value of f at y. The variables $x_1, x_2, \ldots\ldots, x_n$ are treated as parameters and are assumed to remain fixed throughout the definition. Also it is assumed that both the function g and h are known.

Primitive Recursive Function : A function is called primitive recursive iff it can be obtained from the initial or elementary functions by a finite number of operations of composition and recursion. Of course all,primitive recursive functions are total.

From this definition it follows that it is not always necessary to use only the initial functions in the construction of a particular primitive recursive function.For example, if we already have a set of functions $f_1, f_2, \dots, f_k$ which are primitive recursive, then we could use any of these functions along with the initial functions to obtain another primitive recursive function , provided we restrict ourselves to the operations of composition and recursion only.

Example 1 : Show that the function $f\langle x,y\rangle = x + y$ is primitive recursive .

Solution : Notice that $x + (y + 1) = (x + y) + 1$, so that

$$f\langle x, y + 1\rangle = f\langle x, y\rangle + 1 = S(f\langle x , y\rangle)$$

also $f\langle x, 0\rangle = x$

We can now formally define $f\langle x , y\rangle$ as

$$f\langle x , 0\rangle = x = U_1^1\langle x\rangle$$

$$f\langle x, y + 1\rangle = S(U_3^3\langle x , y, f\langle x ,y\rangle\rangle)$$

Here the base function is $g(x) = U_1^1(x)$, and the inductive -step function is $h\langle x,y, z\rangle = S(U_3^3\langle x,y,z\rangle)$. In order to see how we can actually compute the value of $f\langle 2,4\rangle$,for example we have

$f\langle 2,0\rangle = 2$

$f\langle 2,4\rangle = S(f\langle 2,3\rangle) = S(Sf\langle 2,2\rangle)) = S(S(S(f\langle 2,1\rangle)))$

$= S(S(S(S(f\langle 2,0\rangle)))) = S(S(S(S(2)))) = S(S(S(3)))$

$= S(S(4)) = S(5) = 6$ ////

Example 2 : Using recursion, define the multiplication function " * " given by $g\langle x,y\rangle = x * y$

Solution : Since $g\langle x,0\rangle = 0$ and $g\langle x, y + 1\rangle = g\langle x,y\rangle + x$, we write $g<x,0> = Z(x)$

$g<x,y+1> = f< U_3^3<x,y,g<x,y>>, U_1^3<x,y,g<x,y>> >$

where f is the addition given in Example 1. ////

(iii) **Minimalization :**

Let $g<x_1,x_2,\ldots x_n,y>$ be a total function. If there exists atleast one value of $y \in \mathbf{N}$, such that the function $g<x_1,x_2,\ldots x_n,y> = 0$ for all n-tuples $<x_1,x_2,\ldots x_n> \in \mathbf{N}^n$, then g is called a regular function.

Not all functions are regular, as can be seen from $g<x,y> = | y^2 - x |$. Obviously $g<x,y>$ is total, but $| y^2-x | = 0$ for only those values of x which are perfect squares and not all values of x. This fact shows that there is no value of $y \in \mathbf{N}$ such that $|y^2 - x| = 0$ for all x .On the other hand, the function y - x is regular because for y = 0, y-x is zero for all x.

A function $f<x_1,x_2,\ldots,x_n>$ is said to be defined from a total function $g<x_1,x_2,\ldots,x_n>$ by minimization (minimalization) or μ operation if

$$f<x_1,x_2,\ldots,x_n> = \begin{cases} \mu_y(g<x_1,x_2,\ldots,x_n,y> = 0) & \text{if there is such a } y \\ \text{undefined} & \text{otherwise} \end{cases}$$

where μ_y means the least y greater that or equal to zero.

From the definition it follows that $f<x_1,x_2,\ldots,x_n>$ is well defined and total if g is regular.

If g is not regular , then the operation of minimalization may produce a partial function.

TYPES OF RECURSIVE FUNCTIONS

A function is said to be **recursive** iff it can be obtained from initial functions by a finite number of applications of the operations of composition , recursion, minimalization over regular functions.

It is clear from the definition that the set of recursive functions properly includes the set of primitive recursive functions.Also the set of recursive functions is closed under the operations of composition , recursion, minimalization over regular functions.If we remove the restriction for the operation of minimalization, so that it can be performed over any total function and not necessarily over just regular ones, we get a still larger class of functions defined as partial recursive functions.

A function is said to be **partial recursive** iff it can be obtained from the initial functions by a finite number of applications of the operations of composition, recursion, and minimalization.

As was done in the case of primitive recursive functions, it is not necessary to always start from the initial functions in order to construct other functions in the class. In fact, if a set of recursive functions is known, they can be used along with the admissible operations to generate other recursive functions.

EXAMPLES

In addition to previous examples we have some more functions which are used frequently. The following are some of them --

1 Sign function or nonzero test function , sg :

$sg(0) = 0$ $sg(y + 1) = 1$

$sg(0) = Z(0)$ $sg(y + 1) = S(Z(U_2^2<y, sg(y)>))$

2 Zero test function, $\overline{sg}$:

$\overline{sg}(0) = 1$ $\overline{sg}(y + 1)$

3 Predecessor function, P:

$P(0) = 0$ $P(y + 1) = y = U_1^2(y , P(y))$

Note that

$P(0) = 0$ $P(1) = 0$ $P(2) = 1$ $P(3) = 2$ ......

4 Odd and even parity function , Pr :

$Pr(0) = 0$ $Pr(y + 1) = \overline{sg}(U_2^2(y, Pr(y)))$

$Pr(0) = 0$ $Pr(1) = 1$ $Pr(2) = 0$ $Pr(3) = 1$ ......

5 Proper subtraction function, $\dot{-}$:

$x \dot{-} 0 = x$ $x \dot{-} (y + 1) = P(x \dot{-} y)$

Note that $x \dot{-} y = 0$ for $x < y$ and $x - y$ for $x \geq y$.

6 Absolute value function , $| \ |$:

$$| x - y | = (x \dot{-} y) + (y \dot{-} x)$$

7 Minimum function , min<x,y> minimum of x and y :

$$\min\langle x,y\rangle = x \dot{-} (x \dot{-} y)$$

8 maximum function , max<x,y> maximum of x and y

$$\max\langle x,y\rangle = y + (x \dot{-} y)$$

9 The square function , $f(y) = y^2$:

$$f(y) = y^2 = U_1^{\,1} * U_1^{\,1}$$

10 The exponentiation function , $f\langle x , y\rangle = x^y$:

$f\langle x,0\rangle = sg(x)$

$f\langle x,y + 1\rangle = x * f\langle x,y\rangle = U_1^{\,3}\langle x,y,f\langle x,y\rangle\rangle * U_3^{\,3}\langle x,y,f\langle x,y\rangle\rangle$

11 <u>Ackermann's function</u>

In our discussion so far we considered only one induction variable in the definition of recursion. It is possible to consider two or more induction variables. Note that in the definition of $f\langle x_1,x_2,\ldots x_n,y\rangle$ using recursion,$x_1,x_2,\ldots x_n$ were treated as parameters and only y was treated as the induction variable. Now we define a function in which we have two induction variables and no parameters.This function will be used in the following section and is known
as Ackermann's function.The function A <x,y> is defined by

$$A\langle 0,y\rangle = y + 1$$

$$A\langle x + 1,Y + 1\rangle = A\langle x,1\rangle$$

$$A\langle x + 1,y + 1\rangle = A\langle x, A\langle x + 1,y\rangle \rangle$$

Observe that one can construct the value of A<x,y> for fixed values of x and y by using the above definition. Therefore, A<x,Y> is well defined and total.It is known that

A<x,y> is not primitive recursive, but recursive.We now demonstrate how the above definitions can be used in finding the value of A<2,2>.

```
A<2,2> = A <1,A <2,1>>

A<2,1> = A <1,A <2,0>>
       = A <1,A <1,1>>

A<1,1> = A <0,A <1,0>>
       = A <0,A <0,1>>
       = A <0,2> = 3

A<2,1> = A <1,3>
       = A <0,A <1,2>>

A<1,2> = A <0,A <1,1>>
       = A <0,3> = 4

A<2,1> = A <0,4> = 5

A<2,2> = A <1,5>
       = A <0,A <1,4>>

A<1,4> = A <0,A <1,3>>

A<1,3> = A <0,A <1,2>>
       = A <0,4> = 5

A<1,4> = A <0,5>= 6

A<2,2> = A <0,6> = 7                ////
```

There are a few more definitions which are related to recursive functions and so should be understood.

Recursively enumerable set : A set is said to be recursively enumerable if it is the range of a total recursive function . Given an element z which is in the set, in a finite number of computations of the recursive function z will be generated.

Recursive set : A recursive set is a recursively enumerable set whose complement also is a recursively enumerable set.

The set of natural numbers, set of even numbers,set of odd numbers etc. are recursive sets which are recursively eneumerable also.To give explicit examples of a recursively enumerable set which is not recursive is difficult if not impossible.

The above definitions and examples will be sufficient to understand Nu Machine and Non-linear Algebra which are taken up in the following chapters. The next chapter deals with the Nu Machine. ////

CHAPTER 3

NU MACHINE

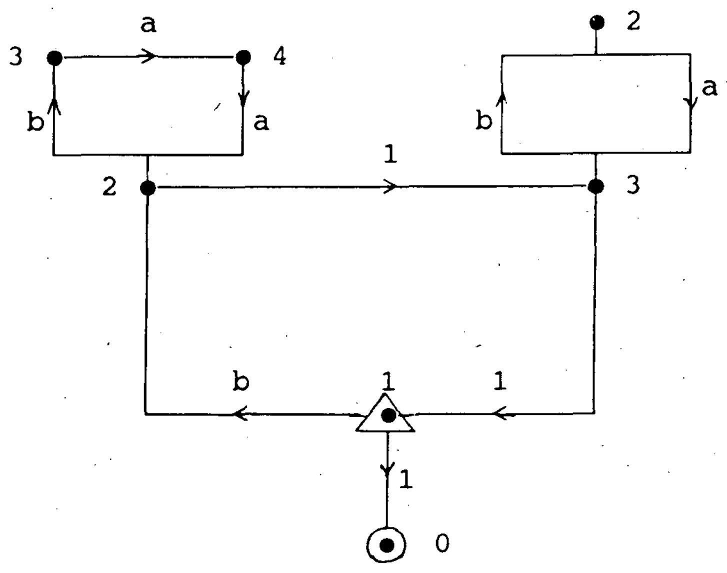
NU MACHINE

In this chapter, an analysis of Nu Machine is made. It is a mathematical model which assays to compute all recursive functions (i.e Turing computable functions). It is named as **Nu Machine** (pronounced as "New machine") and is abbreviated as **NM**. From what follows, it can be observed that this model is much simpler than TM mainly because of its graphical representation. Note that this model is an extended version of ABACUS machine mentioned by SHEPHERDSON. Before going into further details, consider one example of NM for recursive functions and see how it functions.

Every NM is a digraph with labelled nodes and edges. Nodes are labelled 0,1,2,...etc and edges are labelled a, b or e. If there is an 'a' or a 'b' from node A to node B then add an 'a' or a 'b' to the string in node A and move to node B respectively. If the number of a's minus the number of b's is zero in node A then move through the edge labelled 'e' to node B without doing anything. The initial node is represented by a triangle around that node and the final node is represented by a circle around that node. (More precise definition will be given later).

EXAMPLE

(i) **NM for multiplication :**



Initially node 1, node 2, node 3 & node 4 have values 'x', 'y', '0' & '0' respectively. (Value of a node means, the number of a's minus the number of b's in that node. Initially there will be a string of a's of length x, if the value is x). At the end of the calculation, node 4 will have the answer 'x.y'. The description of the states for the computation of '2 * 3' is as follows.

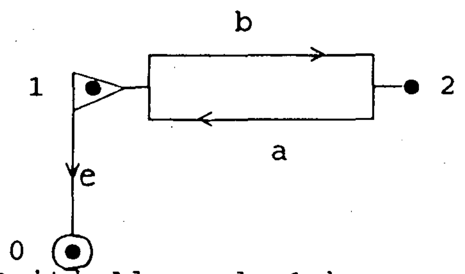
value of state				present state
1	2	3	4	
2	3	0	0	1
1	3	0	0	2
1	2	0	0	3

value of state				present state
1	2	3	4	
1	2	1	0	4
1	2	1	1	2
1	1	1	1	3
1	1	2	1	4
1	1	2	2	2
1	0	2	2	3
1	0	3	2	4
1	0	3	3	2
1	0	3	3	3
1	0	2	3	2
1	1	2	3	3
1	1	1	3	2
1	2	1	3	3
1	2	0	3	2
1	3	0	3	3
1	3	0	3	1
0	3	0	3	2
0	2	0	3	3
0	2	1	3	4
0	2	1	4	2
0	1	1	4	3
0	1	2	4	4
0	1	2	5	2
0	0	2	5	3
0	0	3	5	4
0	0	3	6	2
0	0	3	6	3
0	0	2	6	2
0	1	2	6	3
0	1	1	6	2
0	2	1	6	3
0	2	0	6	2
0	3	0	6	3
0	3	0	6	1
0	3	0	6	0

Before procceding a little further let us look at one more example which will be useful for our understanding the Nu Machine clearly,

Examples :

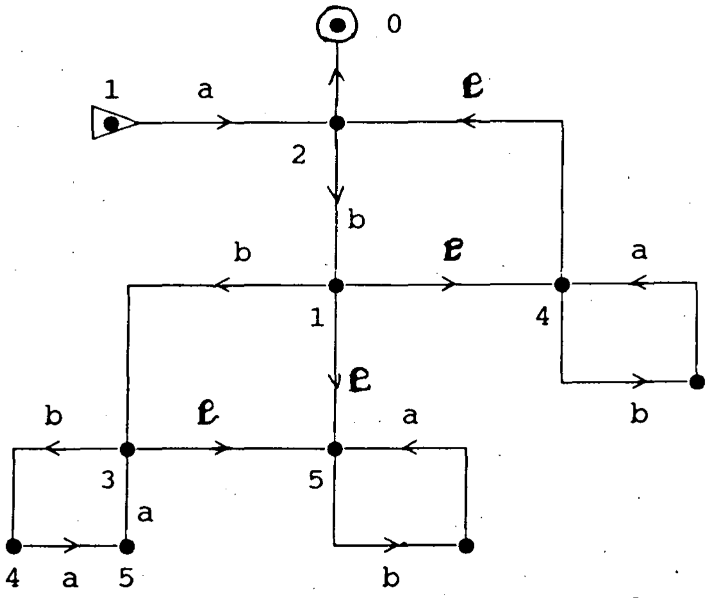
(i) **NM for addition :**



Initially node 1 has a value X (remember that value of a node means, a string of a's of length X) and node 2 has the value Y. At the end of the computation node 2 will have the answer X+Y. As mentioned earlier, the value of a node is, the number of a's minus the number of b's in that node. The following table illustrates the transitions of the machine when, X = 7 & Y = 5. The answer 12 (string of a's of length 12) appears in node 2 when the machine halts in node 0, at the end of the calculation .

1	2	present node
aaaaaaa	aaaaa	1
aaaaaaab	aaaaa	2
aaaaaaab	aaaaaa	1
aaaaaaabb	aaaaaa	2
aaaaaaabb	aaaaaaa	1
aaaaaaabbb	aaaaaaa	2
aaaaaaabbb	aaaaaaaa	1
aaaaaaabbbb	aaaaaaaa	2
aaaaaaabbbb	aaaaaaaaa	1
aaaaaaabbbbb	aaaaaaaaa	2
aaaaaaabbbbb	aaaaaaaaaa	1
aaaaaaabbbbbb	aaaaaaaaaa	2
aaaaaaabbbbbb	aaaaaaaaaaa	1
aaaaaaabbbbbbb	aaaaaaaaaaa	2
aaaaaaabbbbbbb	aaaaaaaaaaaa	1
aaaaaaabbbbbbb	aaaaaaaaaaaa	0 (end)

(ii) **NM for exponentiation :**



Initially nodes 1, 2, 3, 4 & 5 have values 0, Y, X, 0 & 0 respectively. At the end of the calculation the answer X^Y will appear in node 1. The steps involved are exactly as in the above example. Now, it is time to define NM and all the basic machines more precisely.

DEFINITION OF NM

An NM is a labelled digraph with the following properties.

(i) There are two types of edges - dotted edges and line edges (i.e. ------- & ———————).

(ii) Edges are labelled in the form a / α , where a ϵ (Σ U {e}) and α ϵ Σ^*. (Here Σ is the set of symbols in the alphabet and α is the string with letters form the alphabet.)

(iii) Two different edges can have the same label.

(iv) Nodes are labelled 0,1,2.....

(v) Two different nodes can have the same label.

(vi) There is one initial and one final node.

(vii) Out degree of the final node is zero.

In any node of a machine which accepts a language, there is a string of symbols taken from an alphabet. There is a pointer which reads the symbols according to the rules specified. Initially the pointer will be at the left end of the string. Meanings of the notations are as follows. The symbol 'e' represents the zero length character i.e. a null character. If there is no symbol from the specified alphabet on the right of the pointer it means the machine reads 'e'.

$\overset{a/\alpha}{\overline{\hphantom{xxxxxx}}}$: If the symbol on the right of the pointer in the node is 'a' then replace 'a' with the string 'α' and move the pointer to the right of 'α'.

$\overset{a/\alpha}{\text{-------}}$: If the symbol on the right of the pointer in the node is 'a' then replace 'a' with the string 'α' and move the pointer to one position left of 'α'.

$\overset{a}{\overline{\hphantom{xxxxxx}}}$: If the symbol on the right of the pointer in the node is 'a' then replace 'a' with 'a' and move the pointer to one position right.

$\overset{a}{\text{--------}}$: If the symbol on the right of the pointer in the node is 'a' then replace 'a' with 'a' and move the pointer to one position left.

We say NM is **halted normally** if the machine ends in its final node which is normally Node 0. Otherwise, NM is **halted abruptly.**

NM FOR RECURSIVE FUNCTIONS

An NM is a labelled digraph with the following properties.

(i) Edges have labels 'a', 'b' or 'e'.

(ii) Two different edges can have the same label.

(iii) Nodes are labelled 0, 1, 2,....

(iv) Two different nodes can have the same label.

(v) There is one initial and one final node.

(vi) Outdegree of the final node is zero.

(vii) Outdegree of every node (other than final node) is either one or two.

It can be observed that, the examples which have been seen before completely agree with the definition. Meanings of the symbols a, b and e were mentioned in this chapter. Change in the value of a node will automatically affect the other nodes with the same label. Final node is usually numbered as 0. There is no limit for the number of nodes, each node taking values from the set of natural numbers. Let the function to be calculated be

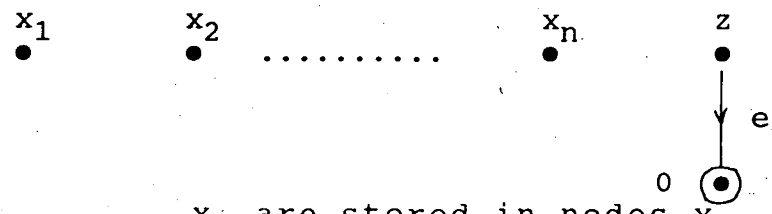
$f(x_1,x_2,...,x_n)$. Initially the variable values are kept in the first n nodes (except 0). Answer can be in any of the nodes except input nodes which is specified before. If the output node is one of the input nodes then transfer the answer into a node which is not an input node at the end of computation. This makes it easier to reuse those nodes. Answer is given by the absolute difference between the number of a's and the number of b's in the output node.

Now the existence of NM's for each of the elementary functions and procedures is shown. The theory will be clear if it can be demonstrated how these basic machines can be interconnected to obtain a machine for a given function. And then it will be obvious that there exists an NM for any recursive function.

NM's FOR ELEMENTARY FUNCTIONS

(i) **Zero function :**

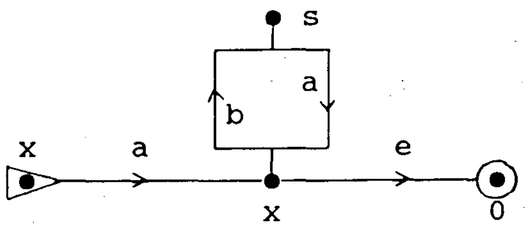
$$Z\ (x_1,x_2,.........,x_n) = 0.$$



x_1, x_2,......., x_n are stored in nodes x_1, x_2, x_3,....,x_n respectively. (Recall that the value x_1 means a string of a's of length x_1). Answer appears in node z (Note that all non-input nodes have values zero initially).

(ii) **Successor function :**

$$S(x) = x'$$

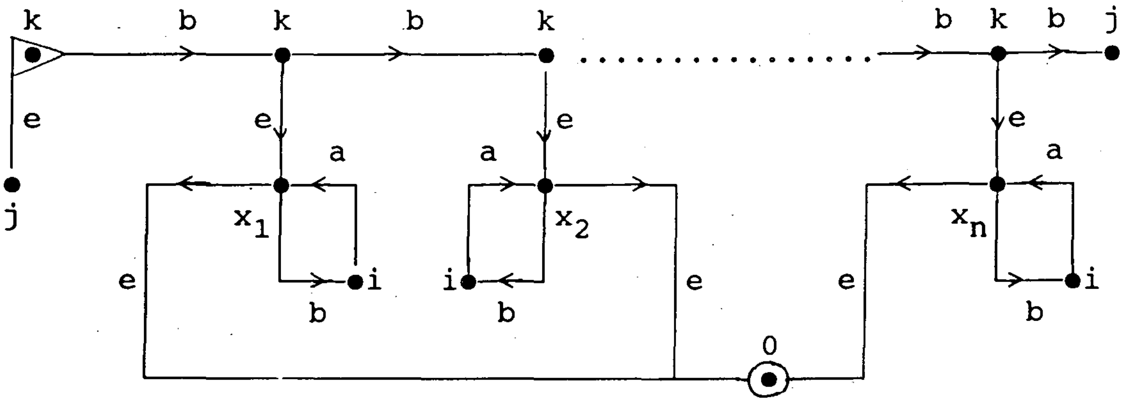


The input value is put in node x. Then an 'a' is added to the string in node x and transferred to the output node s. The new value will be x+1 in node x itself.

TH-3925

(iii) **Identity function :**

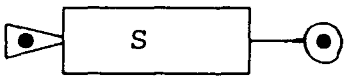
$$U^n (x_1, x_2, \ldots\ldots, x_n) = x_k$$



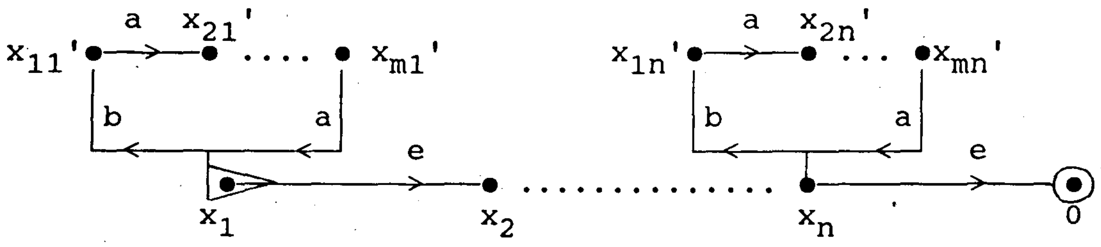
The value k is put in node k. In each step of the computation the value gets decreased until the value of node k becomes zero. When it happens the machine will be in node x_k where the value x_k is kept. Then it transforms the value x_k into the output node i. (Note that all the nodes other than input nodes have to be initialised i.e. value of those nodes have to be made zero before the computation starts).

NM's FOR ELEMENTARY PROCEDURES

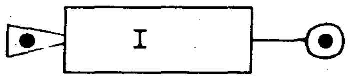
Before going into the details, make following machines which will be useful in defining machines for the elementary procedures. Call them **S(x,x_1',x_2',...,x_m')** and **I(x)**. Here, each of the variable in the above functions is a set of n nodes. S stores the value of each node in x, in the corresponding node in x_1',x_2'...x_m'. The machine,



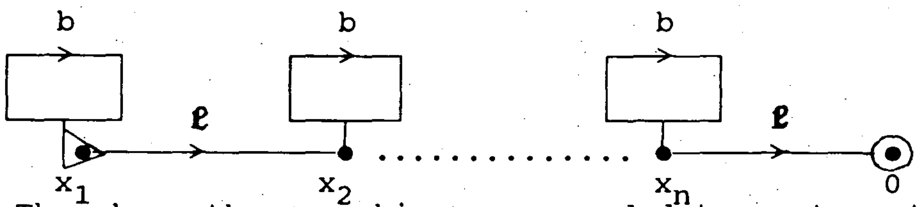
is given by,



The machine I is used for initialisation. It reduces the value of each node in x to zero. The machine,



is given by,



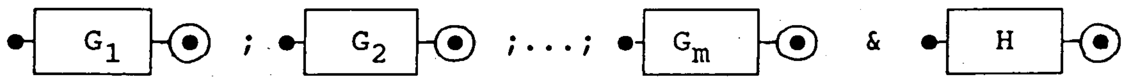
The above three machines are needed to restore the input values at the end of computation when the input nodes are being used. Here onwards only the names of the above machines will be used instead of the entire machine. This will be clear when we study machines for elementary procedures which are used in constructing a machine for a given function. The method of construction will be demonstrated later.

(i) **Composition :**

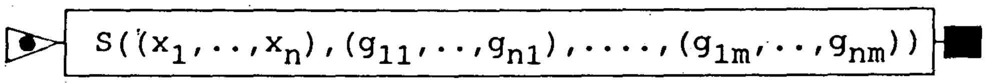
Given a set of functions G_k ($x_1, x_2, \ldots, x_n$), $k=1,2,\ldots,m$ and H ($y_1, y_2, \ldots, y_m$), define

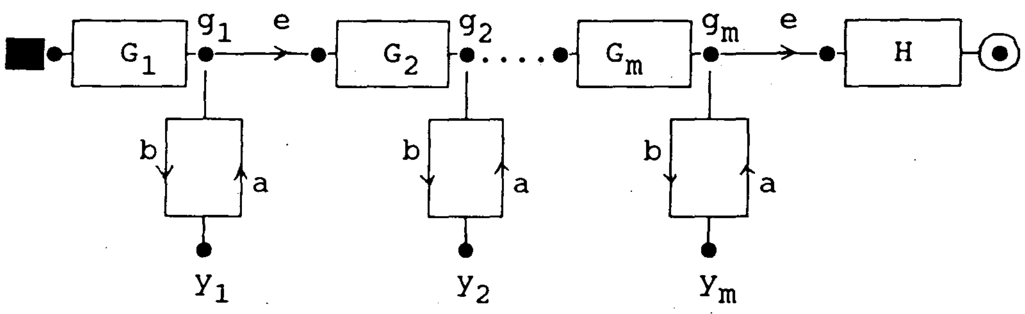
$$F(x_1, x_2, \ldots, x_n) = H(G_1, G_2, \ldots, G_m).$$

NM for composition is given below. Every G has its own input nodes. So the inputs of F i.e. x_1 to x_n are first transformed into the input nodes of G_i for all i, $1 \le i \le n$. Input nodes of G_i are denoted by $x_{1i}, x_{2i}, \ldots, x_{ni}$. g_i's are the output nodes of G_i's and y_i's are the input nodes of H. The given machines are,



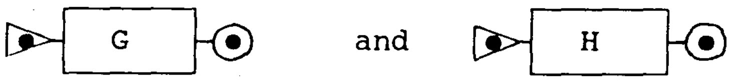
The following is the machine for composition. First block initializes the input nodes of G_i's. Then in the second block each G_i is calculated and the result is transferred to y_i as the input of H. Final node of F will be the final node of H and the value of the function F will be in the output node of H.


$S((x_1,..,x_n),(g_{11},..,g_{n1}),....,(g_{1m},..,g_{nm}))$

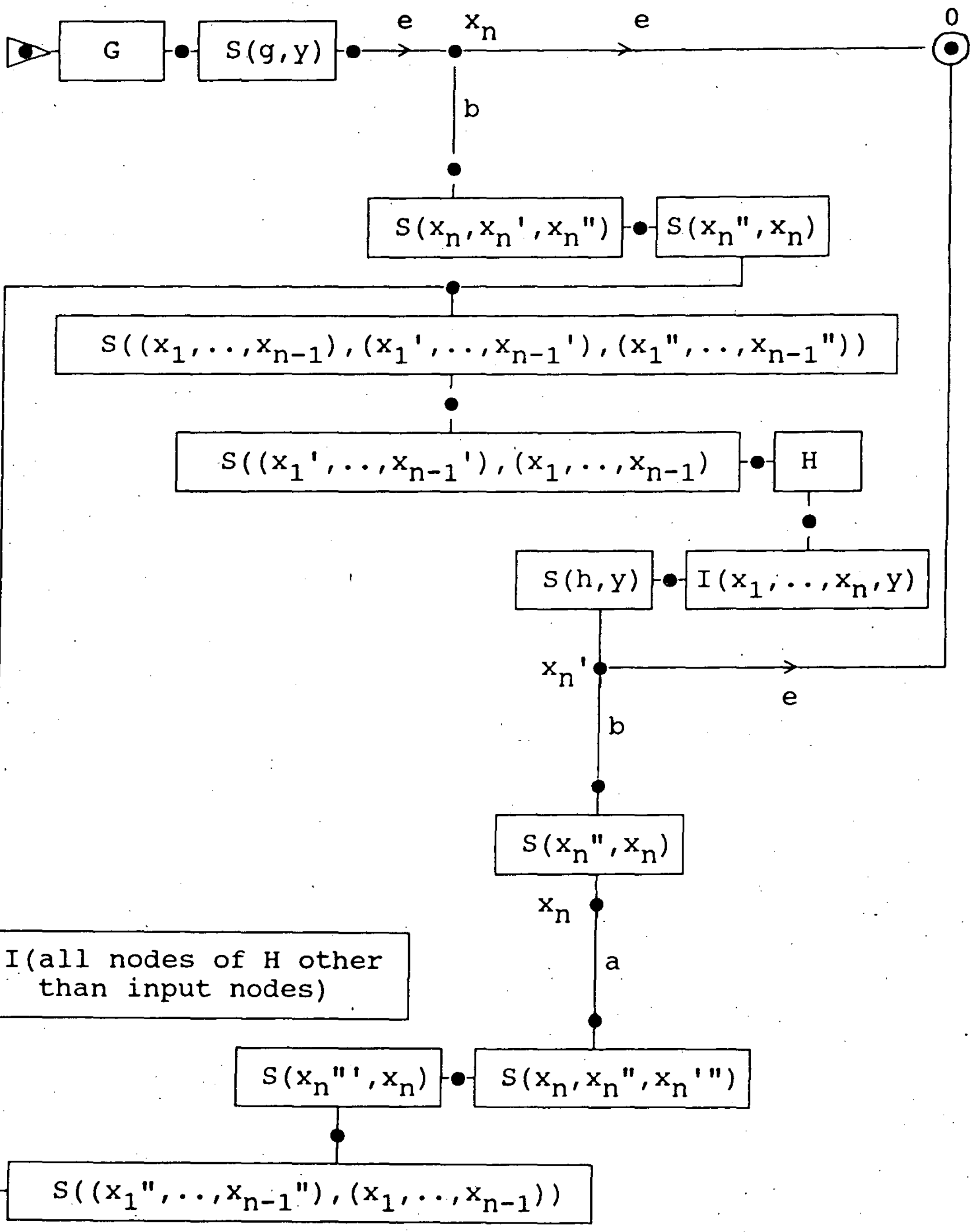


(ii) **Primitive recursion** :

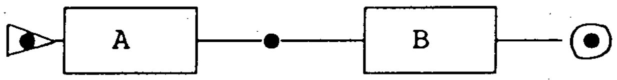
Given two functions $G(x_1,x_2,......,x_{n-1})$ and $H(x_1,x_2,...,x_n,y)$, define $F(x_1,x_2,...,x_n)$ as in the second chapter. The given functions are,



NM for this procedure is given below.


G
S(g,y)
e
x_n
e
0
b
S(x_n,x_n',x_n")
S(x_n",x_n)
S((x_1,..,x_{n-1}),(x_1',..,x_{n-1}'),(x_1",..,x_{n-1}"))
S((x_1',..,x_{n-1}'),(x_1,..,x_{n-1})
H
S(h,y)
I(x_1,..,x_n,y)
x_n'
e
b
S(x_n",x_n)
x_n
a
I(all nodes of H other than input nodes)
S(x_n"',x_n)
S(x_n,x_n",x_n'")
S((x_1",..,x_{n-1}"),(x_1,..,x_{n-1}))

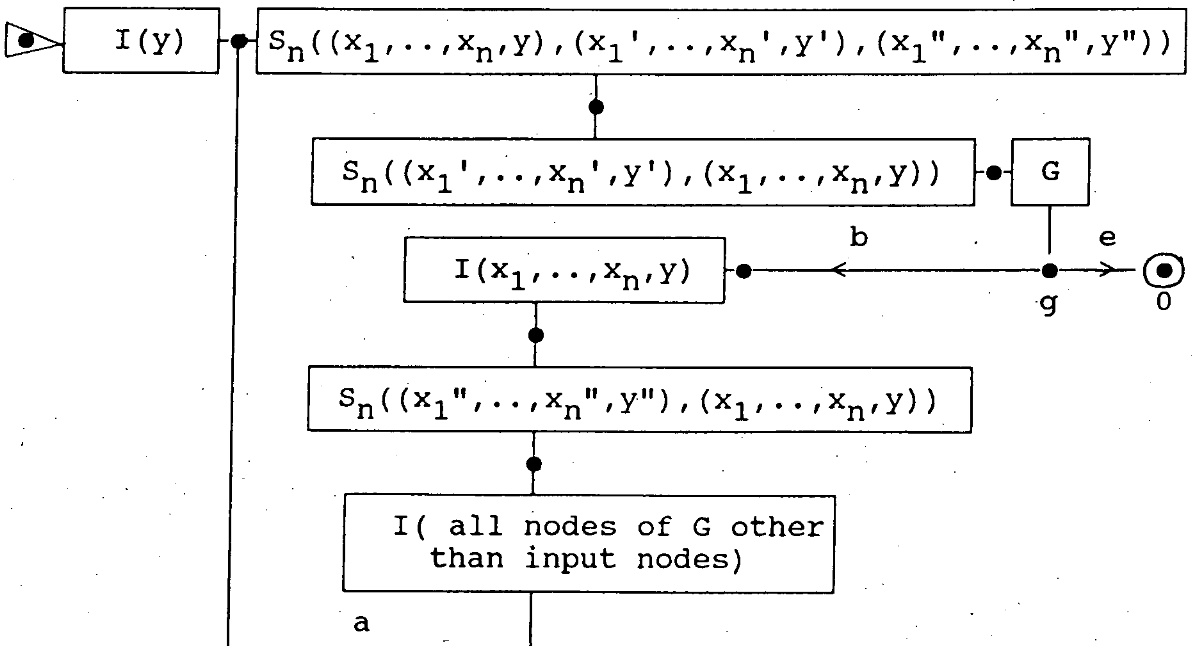
In the above machine, node 'g' is the output node of the machine G and node 'h' is the output node of H. Two machines A and B are interconnected to get C in the following fashion.



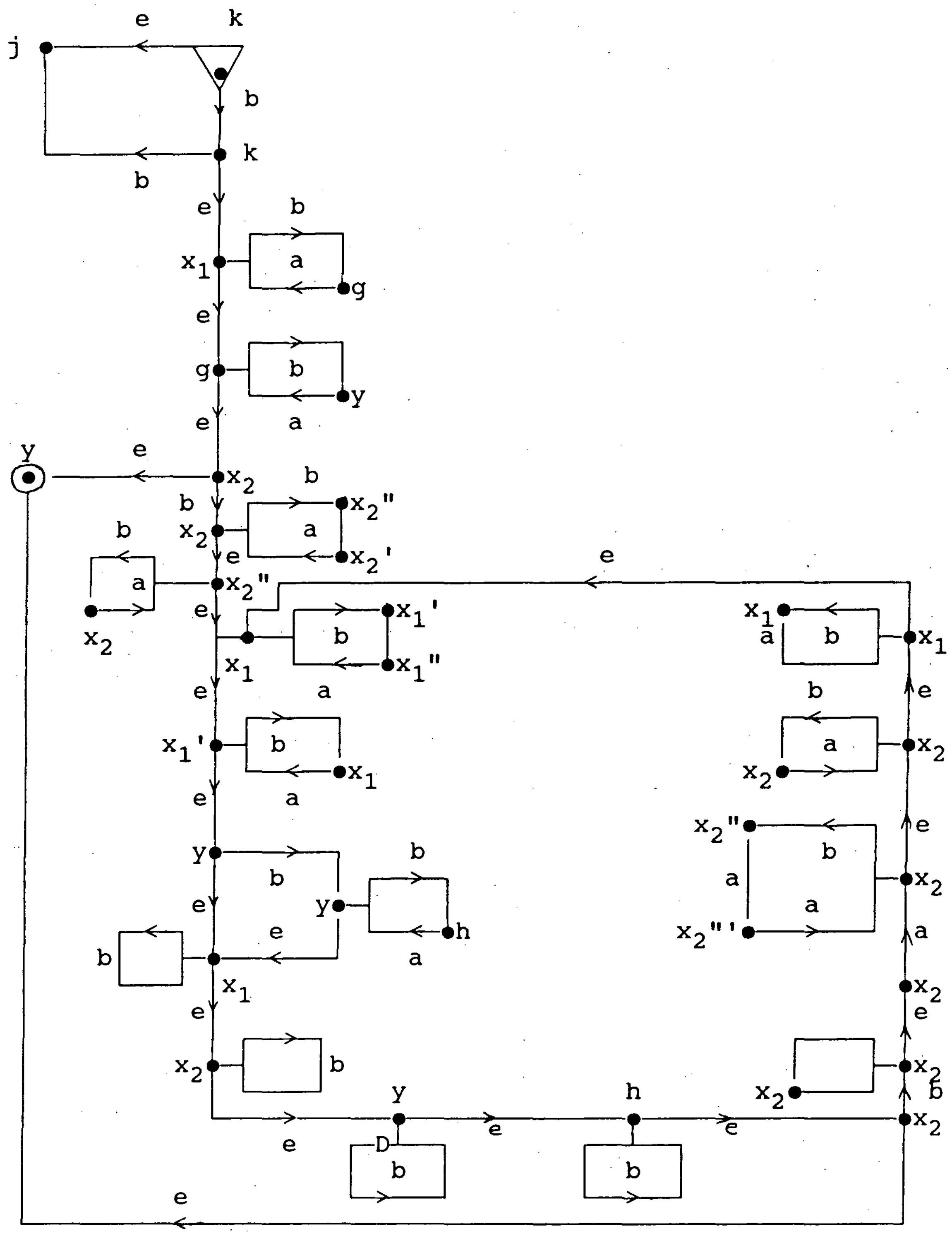
Initial node of C is the initial node of A and final node of C is the final node of B. The connection is done by replacing the final node of A by the initial node of B.

(iii) **Minimalisation** :

Define $F(x_1,\ldots,x_n) = \mathrm{Min}\ \{\ y \mid G(x_1,\ldots,x_n,y)\ \}$ where G is a given function. NM for this procedure is given below.

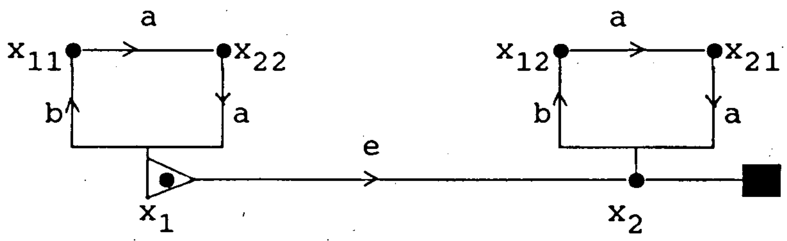

I(y)
$S_n((x_1,..,x_n,y),(x_1',..,x_n',y'),(x_1'',..,x_n'',y''))$
$S_n((x_1',..,x_n',y'),(x_1,..,x_n,y))$
G
b
e
$I(x_1,..,x_n,y)$
g
0
$S_n((x_1'',..,x_n'',y''),(x_1,..,x_n,y))$
I(all nodes of G other than input nodes)
a
y

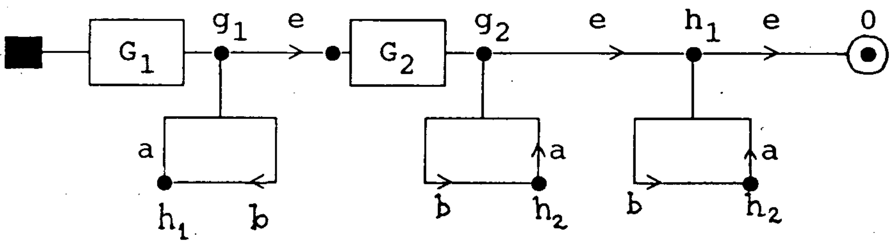
The following example illustrates how to make machines using above defined machines and how to combine them. The machine given on next page is for the proper subtraction defined in Chapter 2. This function is used for calculating $|x_1-x_2|$ (see chapter 2).



For defining the function 'absolute difference' we need the above machine. As explained in Sec.2.3, $F(x_1, x_2) = | x_1 - x_2 | = H(G_1, G_2)$ where, G_1 and G_2 are proper subtractions and $H(x_1, x_2) = x_1+x_2$. Now, for getting the machine for F connect the machines as follows.

Let G_1 and G_2 be the machine given above with a totally different set of nodes. Let the input nodes of G_i be x_{i1} and x_{i2}. Let the output nodes be g_1 and g_2. After changing the nodes as above mentioned, design a machine as follows.





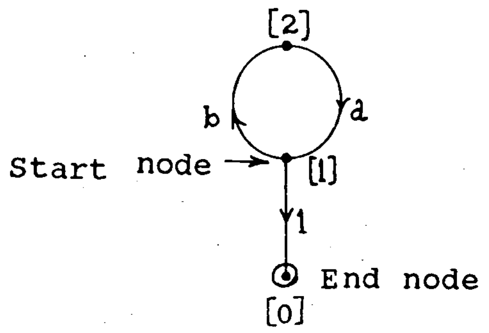
where, h_1 and h_2 are the input nodes of H and the output $|x_1 - x_2 |$ will appear in node h_2.

CHAPTER 4

NON LINEAR ALGEBRA

NON LINEAR ALGEBRA

This chapter deals with 'Non Linear Algebra' which is developed for Nu Machines. A given Nu Machine can be represented in terms of a square matrix. For example, let us consider the following Nu Machine which is used for addition.



The inputs to the nodes are strings containing a's only. If two strings of lengths 'x' and 'y' are kept at nodes 1 and 2 respectively then, in the end node 2 shall have a string of length 'x + y' and node 2 will have a null string at the end. This Nu Machine can be represented in the form of a matrix as follows.

$$A = \begin{bmatrix} 0 & b & 1 \\ a & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \qquad \text{NU MATRIX FOR ADDITION}$$

We shall call the above matrix by the name 'Nu Matrix' for addition.

Here, we first define the state of the Nu Machine by a column matrix in which the element C_{j1} is the string present at node 'j'.

If we initially have a^3, a^2 at nodes 1, 2 respectively and a null string at node 0, then the state vector is nothing but

$$\begin{bmatrix} a^3 \\ a^2 \\ 1 \end{bmatrix}$$

In order to carry out operations we now introduce 'Non linear Algebra' for Nu Machine. The non linear algebra makes use of the above two matrices viz., Nu Martix and State vector. In order to do computation using the non linear algebra we proceed as follows.

1. Take the Nu Matrix.
2. Corresponding to initial inputs write the State vector.
3. Start with the first row of the Nu Matrix. Change the first row of the state vector. For changing the row use the following rules.

If the row contains an 'a' then add one 'a' to the same row element of the vector and if it has a 'b', remove one 'a' from the element of the corresponding row (if possible). Note the column number in the row where 'a' or 'b' (as the case may be) is present. Say the column number is 'i', then go to the i^{th} row in the Nu Matrix. Again repeat the same procedure of changing the state vector but this time the i^{th} row of the vector is changed using the same rules.

Repeat this process until you reach the last row of the

Nu Matrix is reached at which time the execution ceases.

Note : If there is a 'b' in a row but the same row in the vector has an empty string (i.e., 1) then we cannot remove 'a' from this element. In that case we do nothing and go to that column which has got 1 in that row. Next time we will go to the row which corresponds to this column.

Now we shall use this for our example. The initial state vector in our case is,

$$\begin{bmatrix} a^3 \\ a^2 \\ 1 \end{bmatrix}$$

We start with the first row of the Nu Matrix. This row has got a 'b', therefore the first row element of the vector is positioned at 2nd column, so we will now move to 2nd row. The state vector becomes,

$$\begin{bmatrix} a^2 \\ a^2 \\ 1 \end{bmatrix}$$

Now we shall use the 2nd row of the Nu Matrix and change the 2nd row of the state vector. The 2nd row has an 'a' at column '1'. Since 'a' is present in the second row so, we shall add one 'a' to the string already present at the second row of the state vector. This changes the 2nd element from a^2 to a^3. Next we have to go to 1st row of the Nu Matrix. The present vector, is

$$\begin{bmatrix} a^2 \\ a^3 \\ 1 \end{bmatrix}$$

Proceeding in the same fashion we shall get,

$$\begin{bmatrix} a \\ a^3 \\ 1 \end{bmatrix}, \begin{bmatrix} a \\ a^4 \\ 1 \end{bmatrix}, \begin{bmatrix} 1 \\ a^5 \\ 1 \end{bmatrix}$$

and now we have to use first row of Nu Matrix. This row contains a 'b', but the corresponding element is already a null string therefore we cannot delete one 'a' from this. So, in this case we shall do nothing and shall look for column containing '1' in the same row. In our case '1' is at 3rd column, next we have to go to 3rd row which is the last row. As we have now reached the last row, our execution is over.

It is clear from the final state vector that the 2nd row is the addition of initial 1st row and initial 2nd row.

Thus we have seen how Non linear Algebra can be used for addition of two strings. This is similar to adding two numbers. Using this 'Nu Matrix for addition' we can find thesum in the manner described above.

In additon to this, we can write a Nu Matrix for a Nu Machine which performs multiplication. This Nu Matrix can be used and by applying previous rules of Non linear Algebra we can find product of two numbers. This Non linear Algebra can perform the same task which a Nu Machine can accomplish. It was shown in the last chapter that the Nu Machine can compute all recursive functions so this Non linear Algebra also can evaluate recursive functions.

CHAPTER 5

FINITE AUTOMATA AND WORDSPACE

FINITE AUTOMATA AND WORDSPACE

In this chapter we shall study an algorithm for finding out wordspace for a given finite automaton. Given a finite automaton its adjacency matrix can be found from which its wordspace is obtained. At first the algorithm is given, after that it is explained with a few examples.

ALGORITHM

1. For the given finite automaton of 'n' states. Find its adjacency matrix which is a 'n x n' square matrix.
2. In the same matrix change its diagonal elements by their respective Kleene closures. The resulting matrix is called A_1.
3. Now do the following.

$$A_2 = A_1 + C_1 \, a_{11}^* \, R_1$$
$$A_3 = A_2 + C_2 \, a_{22}^* \, R_2$$
$$A_4 = A_3 + C_3 \, a_{33}^* \, R_3$$

....

....

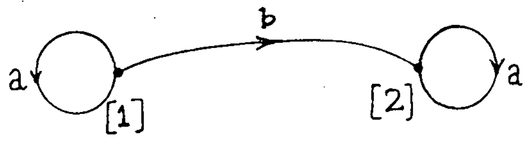
$$A_{n+1} = A_n + C_n \, a_{nn}^* \, R_n$$

In the above equations C_i is the i^{th} column, R_i is the i^{th} row of the matrix A_i calculated at the previous stage and a_{ii} is its i^{th} diagonal element.

The matrix A_{n+1} is nothing but A^*, the wordspace for the given finite automaton.

EXAMPLE (i)

Suppose we have the following automaton with two states and whose alphabet is {a,b}



Its adjacency matrix is

$$A = \begin{bmatrix} a & b \\ 0 & a \end{bmatrix}$$

The Kleene closure of diagonal element a_{11} is a^* and the diagonal element a_{22} is also a^*. Therefore,

$$A_1 = \begin{bmatrix} a^* & b \\ 0 & a^* \end{bmatrix}$$

Now we have,

$$A_2 = A_1 + C_1\, a_{11}{}^* \, R_1$$

i.e.,

$$A_2 = \begin{bmatrix} a^* & b \\ 0 & a^* \end{bmatrix} + \begin{bmatrix} a^* \\ 0 \end{bmatrix} [\ (a^*)^*\ ]\ [\ a^* \quad b\ ]$$

i.e.,

$$= \begin{bmatrix} a^* & b \\ 0 & a^* \end{bmatrix} + \begin{bmatrix} a^* \\ 0 \end{bmatrix} [\ a^* \quad a^*b\ ]$$

$$= \begin{bmatrix} a^* & b \\ 0 & a^* \end{bmatrix} + \begin{bmatrix} a^* & a^*b \\ 0 & 0 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} a^* & a^*b \\ 0 & a^* \end{bmatrix}$$

Similarly,

$$A_3 = A_2 + C_2\, a_{22}^{*}\, R_2$$

$$A_3 = \begin{bmatrix} a^* & a^*b \\ 0 & a^* \end{bmatrix} + \begin{bmatrix} a^*b \\ a^* \end{bmatrix} \left[\ (a^*)^*\ \right] \left[\ 0 \quad a^*\ \right]$$

$$= \begin{bmatrix} a^* & a^*b \\ 0 & a^* \end{bmatrix} + \begin{bmatrix} a^*b \\ a^* \end{bmatrix} \left[\ 0 \quad a^*\ \right]$$

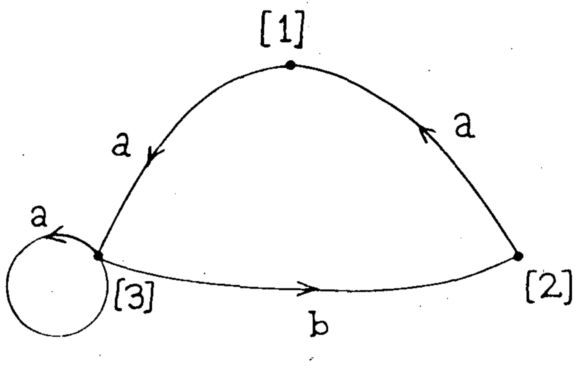
$$= \begin{bmatrix} a^* & a^*b \\ 0 & a^* \end{bmatrix} + \begin{bmatrix} 0 & a^*ba^* \\ 0 & a^* \end{bmatrix}$$

$$= \begin{bmatrix} a^* & a^*ba^* \\ 0 & a^* \end{bmatrix}$$

The above matrix A_3 is nothing but A^* or wordspace for the given finite automaton. In the same way, we can find A^* for other finite automata using this algorithm.

Now we shall consider another example in which the finite automaton has got three states and the alphabet of the automaton is { a,b }.

EXAMPLE (ii)



The adjacency matrix of this automaton is,

$$A = \begin{bmatrix} 1 & 0 & a \\ a & 1 & 0 \\ 0 & b & a \end{bmatrix}$$

The Kleene closures of diagonal elements a_{11}, a_{22} and a_{33} are 1^*, 1^* and a^* i.e., 1, 1 and a^* respectively.

Therefore,

$$A_1 = \begin{bmatrix} 1 & 0 & a \\ a & 1 & 0 \\ 0 & b & a^* \end{bmatrix}$$

Now we have,

$$A_2 = A_1 + C_1 \, a_{11}^* \, R_1$$

So,

$$A_2 = \begin{bmatrix} 1 & 0 & a \\ a & 1 & 0 \\ 0 & b & a^* \end{bmatrix} + \begin{bmatrix} 1 \\ a \\ 0 \end{bmatrix} [\, 1^* \,] \, [\, 1 \quad 0 \quad a \,]$$

i.e.,

$$A_2 = \begin{bmatrix} 1 & 0 & a \\ a & 1 & 0 \\ 0 & b & a^* \end{bmatrix} + \begin{bmatrix} 1 \\ a \\ 0 \end{bmatrix} [\, 1^* \,] \, [\, 1 \quad 0 \quad a \,]$$

or

$$= \begin{bmatrix} 1 & 0 & a \\ a & 1 & 0 \\ 0 & b & a^* \end{bmatrix} + \begin{bmatrix} 1 \\ a \\ 0 \end{bmatrix} [\, 1 \quad 0 \quad a \,]$$

$$= \begin{bmatrix} 1 & 0 & a \\ a & 1 & 0 \\ 0 & b & a^* \end{bmatrix} + \begin{bmatrix} 1 & 0 & a \\ a & 0 & a^2 \\ 0 & 0 & 0 \end{bmatrix}$$

i.e.,

$$A_2 = \begin{bmatrix} 1 & 0 & a \\ a & 1 & a^2 \\ 0 & b & a^* \end{bmatrix}$$

Similarly,

$$A_3 = A_2 + C_2\, a_{22}^*\, R_2$$

i.e.,

$$A_3 = \begin{bmatrix} 1 & 0 & a \\ a & 1 & a^2 \\ 0 & b & a^* \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ b \end{bmatrix} [\, 1^* \,]\ [\, a \quad 1 \quad a^2 \,]$$

$$= \begin{bmatrix} 1 & 0 & a \\ a & 1 & a^2 \\ 0 & b & a^* \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ b \end{bmatrix} [\, a \quad 1 \quad a^2 \,]$$

$$= \begin{bmatrix} 1 & 0 & a \\ a & 1 & a^2 \\ 0 & b & a^* \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \\ b \end{bmatrix} [\, a \quad 1 \quad a^2 \,]$$

$$= \begin{bmatrix} 1 & 0 & a \\ a & 1 & a^2 \\ 0 & b & a^* \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ a & 1 & a^2 \\ ba & b & ba^2 \end{bmatrix}$$

or

$$A_3 = \begin{bmatrix} 1 & 0 & a \\ a & 1 & a^2 \\ ba & b & a^*+ba^2 \end{bmatrix}$$

Now we have,

$$A_4 = A_3 + C_3\, a_{33}^*\, R_3$$

$$A_4 = A_3 + \begin{bmatrix} a \\ a^2 \\ a^*+ba^2 \end{bmatrix} [\ (a^*+ba^2)^*\ ]\ [\ ba \quad b \quad a^*+ba^2\ ]$$

$$= A_3 + \begin{bmatrix} a \\ a^2 \\ a^*+ba^2 \end{bmatrix} [\ (a^*+ba^2)^*ba \quad (a^*+ba^2)^*b \quad (a^*+ba^2)^*\ ]$$

Thus,

$$= A_3 + \begin{bmatrix} a(a^*+ba^2)^*ba & a(a^*+ba^2)^*b & a(a^*+ba^2)^* \\ a^2(a^*+ba^2)^*ba & a^2(a^*+ba^2)^*b & a^2(a^*+ba^2)^* \\ (a^*+ba^2)^*ba & (a^*+ba^2)^*b & (a^*+ba^2)^* \end{bmatrix}$$

i.e.,

$$A_4 = \begin{bmatrix} 1+a(a^*+ba^2)^*ba & a(a^*+ba^2)^*b & a(a^*+ba^2)^* \\ a+a^2(a^*+ba^2)^*ba & 1+a^2(a^*+ba^2)^*b & a^2(a^*+ba^2)^* \\ (a^*+ba^2)^*ba & (a^*+ba^2)^*b & (a^*+ba^2)^* \end{bmatrix}$$

The matrix A_4 evaluated above is the A^* or wordspace of the given finite automaton. Thus we have seen how this method can be applied to a finite automaton to reach its A^* or in other words, its wordspace. ////

CHAPTER 6

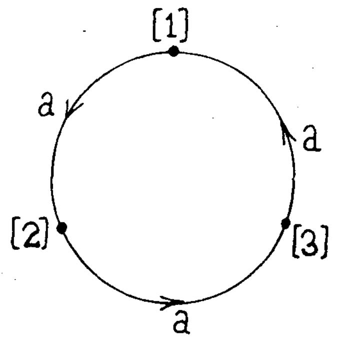
CAYLEY GRAPHS AND THEIR WORDSPACE

CAYLEY GRAPHS AND THEIR WORDSPACE

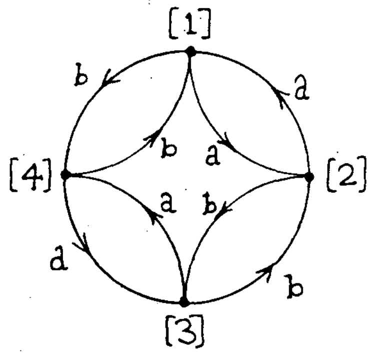
The Cayley graph is a highly symmetrical graph. All its nodes are exactly symmetric with respect to each other. It has got the following properties.

1. The label of the edge is a symbol from an alphabet.
2. All the symbols of the alphabet are associated with each of the nodes.
3. If the cardinality of the alphabet is "n" then indegree as well as outdegree of every node is "n".
4. Each symbol of the alphabet comes exactly once to a node.
5. Each symbol goes out from a node exactly once.

Now, let us look into some examples of Cayley graphs.



GRAPH - A



GRAPH - B

These can be represented in terms of square matrices as shown below,

$$A = \begin{bmatrix} 0 & a & 0 \\ 0 & 0 & a \\ a & 0 & 0 \end{bmatrix}$$

$$B = \begin{bmatrix} 0 & a & 0 & b \\ a & 0 & b & 0 \\ 0 & b & 0 & a \\ b & 0 & a & 0 \end{bmatrix}$$

The graph A comes from alphabet { a } and graph B from alphabet { a,b }. Graph B forms a quadratic dihedral group. The A^* and B^* could be obtained for graph A and graph B respectively.The A^* gives the wordspace for graph A. Here we shall first find A^*, in a similar fashion B^* could be reached.

As we know for the following automaton,

the a^* is

$$a^* = e + a + a^2 + a^3 + \ldots\ldots..$$

where 'e' is the null string.

Similarly for our graph we have,

$$A^* = E + A + A^2 + A^3 + \ldots\ldots...$$

where 'E' is a diagonal matix of same size as A with all the diagonal elements having value 'e' (i.e. the null string). So,

$$A^* = \begin{bmatrix} e & 0 & 0 \\ 0 & e & 0 \\ 0 & 0 & e \end{bmatrix} + \begin{bmatrix} 0 & a & 0 \\ 0 & 0 & a \\ a & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & a^2 \\ a^2 & 0 & 0 \\ 0 & a^2 & 0 \end{bmatrix} + \begin{bmatrix} a^3 & 0 & 0 \\ 0 & a^3 & 0 \\ 0 & 0 & a^3 \end{bmatrix}$$

Therefore,

$$A^* = \begin{bmatrix} (a^3)^* & (a^3)^*a & (a^3)^*a^2 \\ (a^3)^*a^2 & (a^3)^* & (a^3)^*a \\ (a^3)^*a & (a^3)^*a^2 & (a^3)^* \end{bmatrix}$$

$$= (a^3)^* \begin{bmatrix} e & a & a^2 \\ a^2 & e & a \\ a & a^2 & e \end{bmatrix} = (a^3)^* \mathbf{C}$$

where C is called Cayley matrix and C is given by

$$\mathbf{C} = \begin{bmatrix} e & a & a^2 \\ a^2 & e & a \\ a & a^2 & e \end{bmatrix}$$

This Cayley graph forms a group. The elements of the first row i.e., **e,a** and **a^2** give rise to group with generator relation $\underline{a^3 = e}$. The multiplication table for this group is,

•	e	a	a^2
e	e	a	a^2
a^2	a^2	e	a
a	a	a^2	e

Multiplication Table

From above it can be seen that multiplication table is same as the Cayley matrix **C**. Cayley matrix has got another interesting properties.

1. For a Cayley matrix **C** the following property holds,

$$\mathbf{C} \cdot \mathbf{C} = \mathbf{C}$$

2. In addition to this the determinant of a Cayley matrix is equal to '**e**'.
3. Also for any Cayley matrix the inverse is same as the matrix itself, i.e. $\mathbf{C}^{-1} = \mathbf{C}$.

For graph A we have found its A^* and also its Cayley matrix, similar to this for graph B we can find its B^* and the Cayley matrix. The B^* and Cayley matrix **C'** for graph B is

$$B^* = (a^2 + b^2 + abab)^* \begin{bmatrix} e & a & ab & b \\ a & e & b & ab \\ ab & b & e & a \\ b & ab & a & e \end{bmatrix}$$

and Cayley matrix is

$$C' = \begin{bmatrix} e & a & ab & b \\ a & e & b & ab \\ ab & b & e & a \\ b & ab & a & e \end{bmatrix}$$

The generator relations for this graph are $\underline{a^2 = b^2 = (ab)^2 = e}$

In this chapter we have studied Cayley graphs in details with two examples. The Cayley graphs have got applications in group theory and formal language theory.

CHAPTER 7

GRAPHICAL EVALUATION OF DETERMINANTS

GRAPHICAL EVALUATION OF DETERMINANTS

We now investigate a new way of evaluating the determinant of a square matrix using graphical method. For a given square matrix we can draw its corresponding digraph and vice versa. There is one to one correspondence between a square matrix and its digraph. We give below a method which will be used for drawing a digraph for a given square matrix.

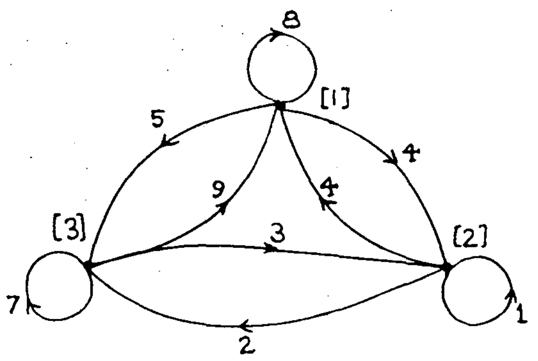
METHOD : If the size of the square matrix A is n x n, then draw n points (or nodes), numbering them from 1 to n. Now for each element a_{ij} of the matrix we draw a corresponding edge on the graph by joining node 'i' and node 'j' with a directed line segment from 'i' to 'j'. The directed edge from node 'i' to node 'j' is assigned a value equal to element a_{ij} of the determinant.

EXAMPLE :

Suppose we have the following square matrix :

$$A = \begin{bmatrix} 8 & 4 & 5 \\ 4 & 1 & 2 \\ 9 & 3 & 7 \end{bmatrix}$$

Then, using the above method its corresponding graph will be



Graph of square matrix A

The algorithm for evaluating the determinant by graphical method follows with an example.

ALGORITHM :

STEP 1. Draw the graph of the square matrix whose determinant is to be calculated.

STEP 2. Select one node in the graph.

STEP 3. List all the outgoing edges of this node. (or all incoming edges of this node).

STEP 4. For each edge E_{ij} in the above set of edges do the following :

4.1 Every edge E_{ij} is assigned a "**weight**" defined by the following equation

$$\text{Weight}(E_{ij}) = \begin{cases} + a_{ij} & \text{if } i = j \quad \text{(i.e. for a self loop)} \\ - a_{ij} & \text{if } i \neq j \quad \text{(otherwise)} \end{cases}$$

4.2 Remove all the outgoing edges of node 'i' from the graph and also all the incoming edges of node 'j'.

4.3 In the remaining graph thus obtained now, club node 'i' and 'j' by putting them into a single node.

4.4 The resulting graph after execution of STEP 4.3 differs from the original graph. This reduced graph can be evaluated using the same algorithm. The determinant of the resulting graph is given name **Sub-determinant** corresponding to the edge E_{ij}.

Note that when the graph after STEP 4.3 gets reduced to a graph of only one node then its determinant value is nothing but the value of self-loop edge associated with it.

STEP 5. Corresponding to each edge E_{ij} from the set of edges of STEP 3, there is a contribution to the original determinant which is given the name of **"contribution"**. This contribution is given by the following equation.

$$\text{Contribution}(E_{ij}) = \text{Weight}(E_{ij}) * \text{Sub-determinant}(E_{ij})$$

STEP 6. The value of the original determinant is given by

$$\text{Determinant} = \Sigma\ \text{Contribution}(E_{ij})$$

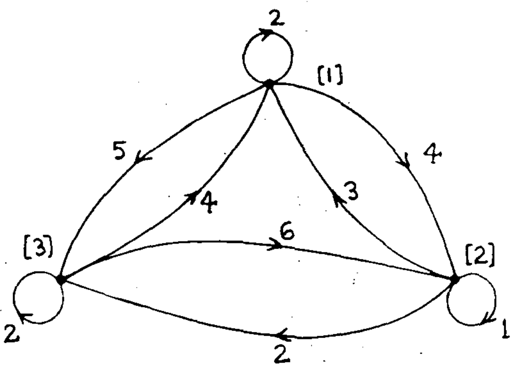
The above algorithm is a recursive algorithm for determinant evaluation. After STEP 4.3 of the above algorithm we are again using the same algorithm for evaluating Sub-determinant(E_{ij}) corresponding to edge E_{ij} . Thus the control goes back to STEP 2 of the algorithm until the graph is reduced to a single node graph at which time Sub-determinant gets the value of self-loop edge and then this process terminates. Hence it is clear how this algorithm works recursively.

To clarify this algorithm consider an example,

Suppose we have the following determinant :

$$\mathbf{A} = \begin{bmatrix} 2 & 4 & 5 \\ 3 & 1 & 2 \\ 4 & 6 & 2 \end{bmatrix}$$

STEP 1. The graph corresponding to above matrix is



GRAPH A

Graph of square matrix A

STEP 2. We have chosen node 1.

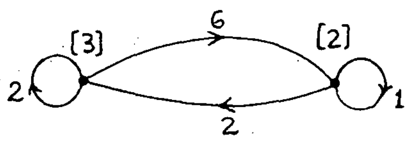
STEP 3. The edges going out from this node are E_{11}, E_{12} and E_{13}.

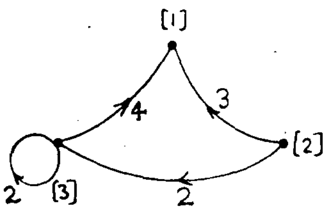
STEP 4.

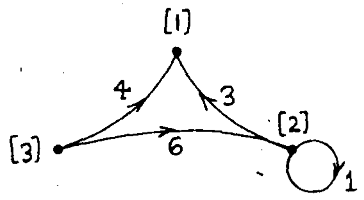
STEP 4.1

Weight(E_{11})= +2 | Weight(E_{12}) = -4 | Weight(E_{13})= -5

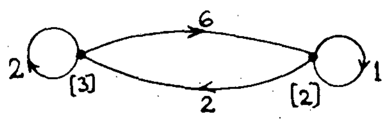
STEP 4.2



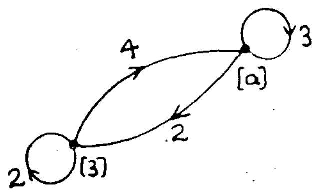




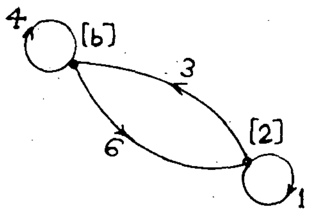
STEP 4.3



GRAPH B



GRAPH C



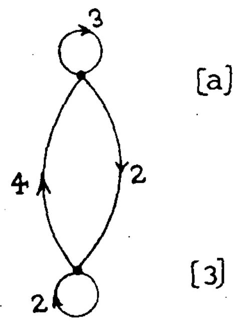
GRAPH D

STEP 4.4
Sub-determinants

E_{11}	E_{12}	E_{13}

These can be evaluated by using the same algorithm once again i.e. we have to go back to STEP 2 and start the process of evaluating the determinant of reduced graph corresponding to edge E_{ij} . The value of this determinant is nothing but Sub-determinant(E_{ij}) for the original determinant.Here in this example we shall evaluate only one Sub-determinant say, E_{12}. The other two can be calculated in a similar fashion.

The reduced graph corresponding to edge E_{12} is the following



Graph C

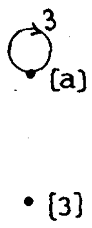
STEP 2 Node "3" is chosen here.

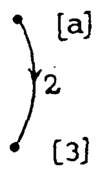
STEP 3 The edges going out from this node are E_{33},E_{3a}.

STEP 4.1

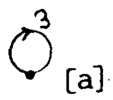
Weight(E_{33}) = +2 | Weight(E_{3a}) = -4

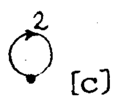
STEP 4.2





STEP 4.3





STEP 4.4 Each of the above reduced graphs contains only one node with a self loop associated with it. Thus we have,

Sub-determinant(E_{33}) = 3 | Sub-determinanant(E_{3a}) = 2

STEP 5

Contribution(E_{33}) = (+2)*(3) = +6 | Contribution(E_{3a}) = (-4)*(2) = -8

STEP 6 Determinant(Graph C) = Σ Contribution

= (+6) + (-8) = -2

Sub-determinant(E_{12}) = Determinant(Graph C) = -2

Similarly Sub-determinant(E_{11}), Sub-determinant(E_{13}) could also be obtained by evaluating Determinant(Graph B) and Determinant(Graph D) respectively . For our example it could be seen that,

Sub-determinant(E_{11}) = -10 and Sub-determinant(E_{13})= -14

Therefore from STEP 5 now we have,

Contribution(E_{11}) = Weight(E_{11}) * Sub-determinant(E_{11})

= (2) * (-10) = -20

Contribution(E_{12}) = (-4) * (-2) = 8

Contribution(E_{13}) = (-5) * (-14) = 70

On moving to STEP 6 we get,

Determinant(A) = Σ Contribution = (-20) + (8) + (70) = 58

SOME USEFUL RESULTS

1. We need not draw an edge E_{ij} in the digraph in case the value of element a_{ij} is zero, because such an edge has zero weight making its contribution equal to zero.

2. In the graph if indegree (or outdegree) of one of the nodes is zero, then the determinant corresponding to that graph vanishes.

3. Take a n x n square matrix and draw its graph. Now from this graph if we remove the node numbers, then we can renumber the "n" nodes of this graph in n! ways. There is one particular square matrix corresponding to every numbering. Each of these n! determinants has the same value. This is because the algorithm evaluates the determinant without taking node numbers into consideration.

4. The "transpose" of a square matrix in terms of graphs can be obtained merely by changing the directions of every edge of the graph.

CHAPTER 8

CONCLUSION

CONCLUSION

An attempt has been made in this thesis to study the matrices with the elements from a loose algebraic structure not as strong as a field. In particular, the matrices with regular expressions as their elements were seen in detail. We have given an algorithm for finding out the wordspace of a finite automaton. In this method the matrices with regular expressions as elements were used. Study of this method may be of help to computer scientists who are working in the area of formal languages, language development, and compiler design.

The recursive functions were discussed in detail, we have studied the Nu Machine which can compute any recursive function in a mechanical manner. In additon, the non linear algebra was studied in the same context.

We have made a study of Cayley graph which has applications in group theory. Some of the results are listed in the chapter related to Cayley graphs. This graph is intimately related to groups.

While we were studying matrices during our project work, a useful result of finding determinant using graphical method was discovered. The method has been discussed in detail in the last chapter. We feel that a lot more can be done in this area.

BIBLIOGRAPHY

BIBLIOGRAPHY

1. Boolos George, Jeffery Richard : Computability and Logic. Cambridge University Press, 1974.

2. Tremblay J.P., Manohar R. : Discrete Mathematical Structures with Applications to Computer Science. McGraw-Hill Computer Science Series, 1975.

3. Carroll John, Long Darrell : Theory of Finite Automata. Prentice-Hall International Editions, 1989.

4. McEliece Robert J., Ash Robert B. : Introduction to Discrete Mathematics. McGraw-Hill International Editions, Computer Science Series, 1989.

5. Salomaa Arto : Jewels of Formal Language Theory. Computer Science Press, 1981.

6. Hopcroft John E., Ullman Jeffery D. : Introduction to Automata Theory Languages and Computation. Narosa Publishing House, 1989.