

(898)
602

A UNIFIED APPROACH TO THEORY OF COMPUTATION

Dissertation submitted
in partial fulfilment of the requirements
for the award of the Degree of

MASTER OF TECHNOLOGY

in

COMPUTER SCIENCE & TECHNOLOGY

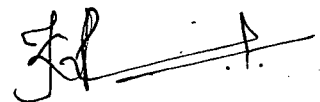
PRAMOD VARMA K.

**SCHOOL OF COMPUTER AND SYSTEMS SCIENCES
JAWAHARLAL NEHRU UNIVERSITY
NEW DELHI - 110067
JANUARY 1991**

CERTIFICATE

This is to certify that the thesis entitled "A UNIFIED APPROACH TO THEORY OF COMPUTATION", 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, 1990.

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..



(PRAMOD VARMA K.)



Prof. N. P. Mukherjee
Dean, SCSS,
J. N. U., New Delhi.



Prof. K. K. Nambiar
Professor, SCSS,
J. N. U., New Delhi.

ACKNOWLEDGEMENTS

I take this opportunity to express my sincere indebtedness to my guide Prof. K. K. Nambiar who, throughout the course, especially project period, enlightened me with his vast knowledge and experience. During this period, I experienced the true knowledge and in him I saw a real teacher. I am really glad that I could gain, albeit very little, from the flowing knowledge he poured on me during our discussions. I wish I could be his student forever.

I extend my thanks to Prof. N.P. Mukherjee, Dean, School of Computer and Systems Sciences, JNU for providing me the opportunity to undertake this project. I would also like to thank the authorities of our school for providing me the necessary facilities to complete my project.

This project would not have been completed without the help of many people. Of all these, I am particularly obliged to two of my close friends, 'Moti' and 'Chum', who helped me all through my project.

Finally, I dedicate this thesis to my parents whose unbounded afflatus had been a constant source of power, and to my younger brother whose affectionate criticism pushed me more towards my goal. I owe every happy moment of my life to them.

Pramod Varma K.

To my dear

achan, amma

&

aniyan

CONTENTS

<i>Ch. 1</i>	INTRODUCTION	1
<i>Ch. 2</i>	RECURSIVE FUNCTIONS	5
	2.1 Elementary functions	5
	2.2 Elementary procedures	6
	2.3 Types of recursive functions	7
<i>Ch. 3</i>	CURRENT MODELS OF COMPUTATION	12
	3.1 Finite automata	12
	3.2 Push down automata	15
	3.3 Turing machines	18
<i>Ch. 4</i>	NU MACHINE	23
	4.1 Definition of NM	25
	4.2 NM for recursive functions	27
	4.3 NM's for elementary functions	28
	4.4 NM's for elementary procedures	29
	4.5 NM's corresponding to PDA and FA	34
	CONCLUSION	39
	BIBLIOGRAPHY	40
	APPENDICES	
	A. NM for addition	
	B. NM for multiplication	

Chapter 1

INTRODUCTION

At present THEORY OF COMPUTATION is taught making use of different models of computation. The following are the three models in use.

Finite Automata (FA) is a machine capable of recognizing sets of strings of a particular type called **Regular Languages**, **Push Down Automata (PDA)** is one which recognizes sets of strings called **Context Free Languages** and **Turing Machine (TM)** is the third one which is capable of computing all **Recursively Enumerable Languages**.

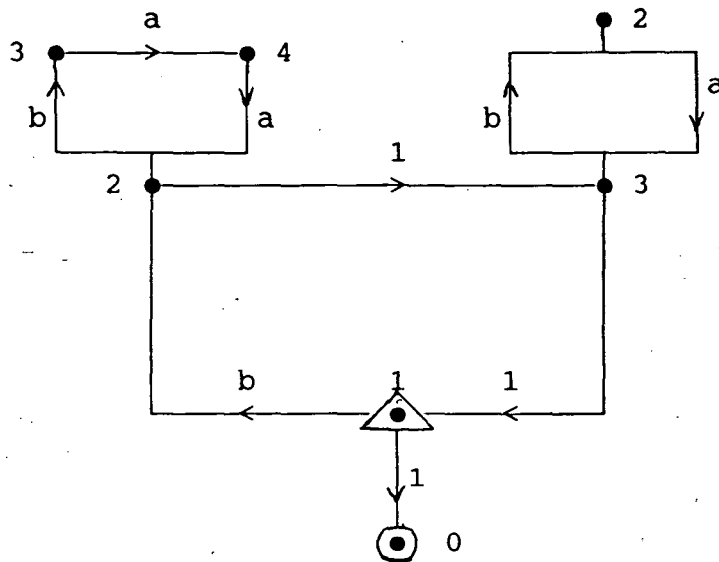
But it is clear that all these three models differ substantially from each other. (For more details of FA, PDA and TM see chapter III). So it will be conceptually very advantageous for both students and instructors if a uniform model can be obtained for all computations. In this project an attempt is made to obtain such a model.

The mathematical model which is assayed in this work can 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 labeled nodes and edges. Nodes are labeled 0,1,2,...etc and edges are labeled 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 labeled '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
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 concluding this chapter we will have an overall look at the distribution of chapters in this thesis. In the next chapter Recursive Functions and related definitions are introduced to avoid the ambiguity in notations and to have a ready reference for the reader. For easy understanding several examples are included. In chapter III, FA, PDA & Turing Machine are introduced along with some examples in a very concise manner. It will be helpful to compare FA, PDA, and TM with NM - the model which is studied in very detail in chapter IV along with further possibilities, like restricting NM to form PDA and then FA .



RECURSIVE FUNCTIONS

The word 'Recursive Function' albeit it is used in the first chapter, the precise definition and more details are furnished in this chapter. The investigation of these functions is very important for the proper understanding of the remaining part of the work. Moreover, the notations and definitions used in this chapter may not be the same as in the literature. Because of the constant use of these words in the later part of the work, all of them are defined precisely to avoid confusion. The family of recursive functions can be further divided into three subfamilies namely, **Primitive Recursive Functions, Total Recursive Functions, and Partial Recursive Functions**. Before going into the details of the above families, the following basic definitions have to be looked into. (Note that the domain and range of all functions are the set of natural numbers.)

2.1 ELEMENTARY FUNCTIONS

The following are the three elementary functions.

(i) **Zero Function :**

$$Z(x_1, x_2, \dots, x_n) = 0 \quad \forall x_i \text{'s.}$$

(ii) **Successor Function :**

$S(x) = x'$, where x' is the integer next to x in the natural sequence.

(iii) **Identity Function :**

$$U_k^n(x_1, x_2, \dots, x_{k-1}, x_k, x_{k+1}, \dots, x_n) = x_k.$$

2.2 ELEMENTARY PROCEDURES

The following are the three elementary procedures.

(i) **Composition :**

Given a set of functions $G_k (x_1, x_2, \dots, x_n)$,
 $k=1, 2, \dots, m$ and $H(x_1, x_2, \dots, x_m)$, define
 $F (x_1, x_2, \dots, x_n) = H (G_1, G_2, \dots, G_m)$.

(ii) **Primitive Recursion :**

Given two functions $G (x_1, x_2, \dots, x_{n-1})$ and
 $H (x_1, x_2, \dots, x_n, y)$ define $F (x_1, x_2, \dots, x_n)$
as follows.

$$F(x_1, \dots, x_{n-1}, 0) = G(x_1, \dots, x_{n-1}) = G_0$$

$$F(x_1, \dots, x_{n-1}, 1) = H(x_1, \dots, x_{n-1}, 0, G_0) = G_1$$

⋮

$$F(x_1, \dots, x_{n-1}, x_n) = H(x_1, \dots, x_{n-1}, x_n, G_{x_n-1}) .$$

(iii) **Minimalisation :**

Given $G(x_1, \dots, x_n, y)$ we can construct

$$F(x_1, \dots, x_n) = \text{Min} \{ y / G(x_1, \dots, x_n, y) = 0 \} .$$

If $G = 0$ is guaranteed then it is **Total**

Minimalisation else it is **Partial Minimalisation**.

■

With these definitions, the above mentioned families of functions can be defined as follows.

2.3 TYPES OF RECURSIVE FUNCTIONS

All functions which are defined using elementary functions , composition and primitive recursion are categorised as **Primitive Recursive Functions**.

If we are using total minimalisation in the derivation of a function then that function is included in the set of **Total Recursive Functions**.

If we are using partial minimalisation instead of total minimalisation then that function is a **Partial Recursive Function**. ■

The following examples will help the reader to understand the above defined functions without any ambiguity.

EXAMPLES

(i) For defining $F(x) = 1$: **Constant function**

Given $G(x) = Z(x)$ and $H(y) = S(y)$.

Using composition we can get

$$F(x) = H \cdot G = S(Z(x)) = S(0) = 1 .$$

(ii) $F(x) = x^* = x \div 1$: **Proper subtraction**.

Given $G = 0$ and $H(x,y) = U_1(x,y) = x$.

We can get $F(x)$ using primitive recursion .

$$F(0) = G = 0$$

$$F(1) = H(0,0) = 0$$

$$F(2) = H(1,0) = 1$$

$$\begin{array}{c} \vdots \\ \vdots \\ \vdots \end{array}$$

$$F(x) = H(x-1,x-2) = x-1 .$$

(iii) $F(x_1, x_2) = x_1 + x_2$: **Addition** .

Given $G(x_1) = U_1(x_1) = x_1$ and

$H(x_1, x_2, y) = S(y) = y+1$.

Using primitive recursion we get $F(x_1, x_2)$.

$$F(x_1, 0) = G(x_1) = x_1$$

$$F(x_1, 1) = H(x_1, 0, x_1) = x_1 + 1$$

⋮
⋮
⋮

$$F(x_1, x_2) = H(x_1, x_2 - 1, x_1 + (x_2 - 1)) = x_1 + x_2 .$$

(iv) $F(x_1, x_2) = x_1 * x_2$: **Multiplication** .

Given $G(x_1) = 0$ and $H(x_1, x_2, y) = x_1 + y$.

Define $F(x_1, x_2)$ as follows using primitive recursion.

$$F(x_1, 0) = G(x_1) = 0$$

$$F(x_1, 1) = H(x_1, 0, 0) = x_1$$

$$F(x_1, 2) = H(x_1, 1, x_1) = 2 * x_1$$

⋮
⋮
⋮

$$F(x_1, x_2) = H(x_1, (x_2 - 1), (x_2 - 1) * x_1) = x_1 * x_2$$

(v) $F(x) = x!$: **Factorial** .

Given $G = 1$ (constant function) and $H(x, y) = (x+1) * y$.

Define $F(x)$ as follows .

$$F(0) = G = 1$$

$$F(1) = H(0, 1) = 1$$

$$F(2) = H(1, 1) = 2$$

⋮
⋮
⋮

$$F(x) = H(x-1, (x-1)!) = x!$$

(vi) $F(x_1, x_2) = x_1^{x_2}$: **Exponentiation** .

Given $G = 1$ and $H(x_1, x_2, Y) = x_1 * Y$.

Define $F(x_1, x_2)$ as follows.

$$\begin{aligned} F(x_1, 0) &= G &= 1 \\ F(x_1, 1) &= H(x_1, 0, 1) &= x_1 \\ F(x_1, 2) &= H(x_1, 1, x_1) &= x_1^2 \\ &\vdots &\vdots \\ &\vdots &\vdots \\ F(x_1, x_2) &= H(x_1, (x_2-1), x_1^{x_2-1}) &= x_1^{x_2} . \end{aligned}$$

(vii) $F(x_1, x_2) = x_1 \dot{-} x_2$: **Proper subtraction**.

Define $G(x_1) = x_1$ and $H(x_1, x_2, Y) = Y^*$ (predecessor of Y) for deriving $F(x_1, x_2)$ using primitive recursion.

$$\begin{aligned} F(x_1, 0) &= G(x_1) &= x_1 \\ F(x_1, 1) &= H(x_1, 0, x_1) &= x_1 - 1 \\ &\vdots &\vdots \\ &\vdots &\vdots \\ F(x_1, x_2) &= H(x_1, x_2-1, (x_1 - x_2 + 1)) &= x_1 - x_2 \end{aligned}$$

(viii) $F(x_1, x_2) = |x_1 - x_2|$: **Absolute difference** .

Given three functions $G_1(x_1, x_2) = x_1 \dot{-} x_2$, $G_2(x_1, x_2) = x_2 \dot{-} x_1$ and $H(x_1, x_2) = x_1 + x_2$, define $F(x_1, x_2) = H(G_1, G_2)$ using composition as follows.

$$\begin{aligned} F(x_1, x_2) &= H((x_1 \dot{-} x_2), (x_2 \dot{-} x_1)) \\ &= \begin{cases} x_1 - x_2 & \text{if } x_1 > x_2 \\ 0 & \text{if } x_1 = x_2 \\ x_2 - x_1 & \text{if } x_1 < x_2 \end{cases} \end{aligned}$$

(ix) $F(x_1, x_2) = \text{Min}(x_1, x_2)$: Minimum of two numbers.

Define $G_1(x_1, x_2) = x_1$ and $G_2(x_1, x_2) = x_1 \dot{-} x_2$. And then $H(x_1, x_2) = x_1 \dot{-} x_2$ for obtaining $F(x_1, x_2)$ using composition.

$$\begin{aligned} F(x_1, x_2) &= H(x_1, (x_1 \dot{-} x_2)) \\ &= x_1 \dot{-} (x_1 \dot{-} x_2) \\ &= \begin{cases} x_2 & \text{if } x_1 > x_2 \\ x_1 & \text{if } x_1 \geq x_2 \end{cases} \end{aligned}$$

All of the above defined functions are examples of primitive recursive functions. Before seeing few more final definitions of this chapter, consider two examples for total recursive functions.

(i) $F(x) = \lceil x^{1/2} \rceil$: Roof of $x^{1/2}$.

$G(x, y) = x \dot{-} y^2$ is given. Define $F(x)$ using minimalisation as follows.

$F(x) = \text{Min} \{ y / x \dot{-} y^2 = 0 \}$ (Total minimalisation).

(i.e. start putting values for y from 0 and take the first value of y which makes $G(x, y) = 0$).

(ii) Ackermann function.

It is defined in a recursive fashion as follows.

$$\begin{cases} A(0, n) = n+1 \\ A(m, 0) = A(m-1, 1) \\ A(m, n) = A(m-1, A(m, n-1)) \end{cases}$$

For example,

$$\begin{aligned}
 A(2,n) &= A(1,A(2,n-1)) \\
 &= A(1,A(1,A(2,n-2))) \\
 &\quad \vdots \\
 &\quad \vdots \\
 &= A(1,(1,\dots, A(2,0))\dots) \text{ and so on until} \\
 &\quad \text{we have,} \\
 A(2,n) &= 2n + 3. \quad \blacksquare
 \end{aligned}$$

The following definitions also, are related to recursive functions and so should be understood.

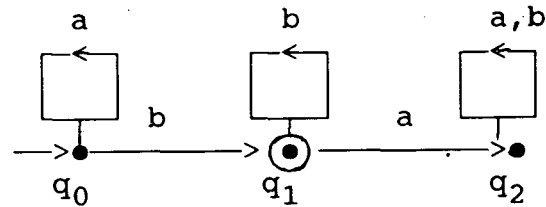
A **recursively enumerable set** is the range of a total function.

A **recursive set** is a recursively enumerable set whose complement also is a recursively enumerable set. \blacksquare

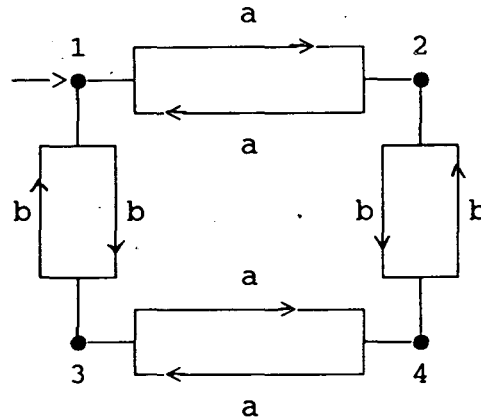
Set of natural numbers, set of even numbers etc. are recursively enumerable sets. Set of even numbers, set of odd numbers etc. are recursive sets.

The above definitions will be enough to understand NM which is given in chapter IV. The next chapter recalls the definitions of the current models. \blacksquare

CURRENT MODELS OF COMPUTATION



(ii)



The above DFA accepts the language consisting of all strings over $\{a,b\}$ that contain an

- (i) even number of a's and an even number of b's if 1 is the final state.
- (ii) odd number of a's and an even number of b's if 2 is the final state.
- (iii) even number of a's and an odd number of b's if the final state is 3.
- (iv) odd number of a's and an odd number of b's if 4 is the final state. ■

There are three models of FA namely, DFA, Non-Deterministic Finite Automata (NFA), NFA with ϵ -transition (ϵ -NFA). It can be shown that all these models are equivalent to each other i.e. all of them accept exactly

As mentioned in the first chapter, FA, PDA, TM are the three models which are generally in use. In this chapter these models are studied briefly with examples. It will be very helpful for the reader to compare these models with NU-Machine. Moreover, the examples given for these current models are the same for NM, which makes the comparison easy for the reader.

A language recognizer or acceptor determines whether an input string is in a language or not. All these machines process inputs and generate outputs. For example, a vending machine takes coins as inputs and returns food as output, a combination lock expects a sequence of numbers and opens the lock if the input is correct and so on. These machines can be either **Deterministic** or **Non-Deterministic**. In this section, only deterministic models will be studied.

3.1 FINITE AUTOMATA

These are the machines which accept languages of a special type called **Regular Languages/Regular sets** over a set of symbols called the **Alphabet Σ** . Regular sets are defined as follows.

- (i) ϕ , $\{\epsilon\}$, $\{a\}$ for every $a \in \Sigma$ are regular sets.
- (ii) If X and Y are regular sets, $X \cup Y$, $X \cdot Y$, X^* are regular sets.
- (iii) Nothing else is a regular set. ■

Examples of regular sets are $\{a, b\}^*$, $\{aa, b, ab\}$ and so on. (For more details of closure(*) of a set, alphabets, regular expressions...etc see ref[1]).

Deterministic Finite Automaton (DFA) is a quintuple $M=(Q, \Sigma, \delta, q_0, F)$ where,

- (i) Q is a finite set of states,
- (ii) Σ is the alphabet,
- (iii) $q_0 \in Q$ is a distinguished state known as the 'initial state',
- (iv) $F \subseteq Q$ is called the set of 'final or accepting states', and
- (v) $\delta : Q \times \Sigma \rightarrow Q$ is a total function known as 'transition function'. ■

The language accepted by M , denoted by $L(M)$ is the set of strings in Σ^* accepted by M .

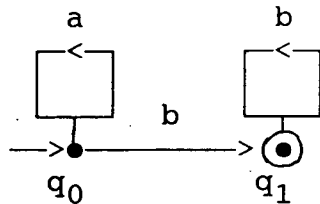
State Diagram of a DFA is a labeled digraph G defined by the following conditions.

- (i) The nodes of G are the elements of Q .
- (ii) The labels on the arcs of G are elements of Σ .
- (iii) q_0 is the initial node $\rightarrow \bigcirc$.
- (iv) F is the set of final or accepting nodes: each of them is depicted \bigcirc .
- (v) There is an arc from node q_i to q_j labeled 'a' if $\delta(q_i, a) = q_j$.
- (vi) For every node q_i and symbol a , there is exactly one arc labeled 'a' leaving q_i .

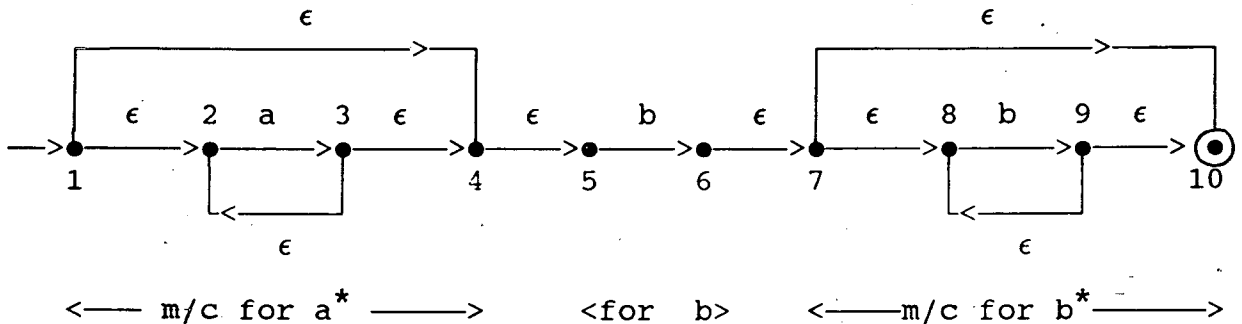
Examples :

- (i) The regular expression for the following example is a^*bb^* .

the same set of languages. The main purpose of ϵ -NFA is to model a machine in an algorithmic way when a regular expression is given. The following are the NFA and ϵ -NFA respectively for the first example given above.



(Note that the redundant arcs and node are removed without changing the language).



See the ref.[1] for a method of making ϵ -NFA from the given regular expression.

3.2 PUSH DOWN AUTOMATA

Regular languages have been characterized as the languages generated by regular grammars and accepted by finite automata. This section presents a class of machines, **The Push Down Automata**, that accepts the **The Context Free Languages (CFL)**. A CFL is the set corresponding to a **Context**

Free Grammar (CFG). (see ref[1] for more details about grammars). A PDA is a finite state machine augmented with external memory, called a **stack** which provides the PDA with first-in, last-out memory management capability.

A **Push Down Automaton** is a septuple $M = (Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ where,

- (i) Q, Σ, q_0 & F are same as in the definition of DFA,
- (ii) Γ is a finite set called the 'stack alphabet',
- (iii) $Z_0 \in \Gamma$ is a particular stack symbol called 'start symbol' and
- (iv) $\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow$ subsets of $Q \times \Gamma^*$, is the 'transition function'.

Moves of the PDA are defined as follows. The interpretation of $\delta(q, a, z) = \{(p_1, \alpha_1), \dots, (p_m, \alpha_m)\}$ where q and $p_i, 1 \leq i \leq m$ are states, $a \in \Sigma, z \in \Gamma$ and α_i is in $\Gamma^*, 1 \leq i \leq m$ is that, the PDA in q with input symbol a and stack symbol z enter state p_i for any i , replace z by the string α_i and advance the read head (or input head) by one symbol. Note that the leftmost symbol of α_i will be the top symbol of the stack. Similarly, $\delta(q, \epsilon, z) = \{(p_1, \alpha_1), \dots, (p_m, \alpha_m)\}$ means that the PDA in state q , independent of the input symbol being scanned and with z as the top symbol on the stack, can enter state p_i and replace z by α_i for any $i, 1 \leq i \leq m$. In this case the read head is not advanced.

PDA $M=(Q, \Sigma, \Gamma, \delta, q_0, Z_0, F)$ is **Deterministic** if,

- (i) for each q in Q and z in Γ , whenever $\delta(q, \epsilon, z)$ is nonempty, then $\delta(q, a, z)$ is empty for all $a \in \Sigma$;
- (ii) for no q in Q , z in Γ and a in $(\Sigma \cup \{\epsilon\})$ does $\delta(q, a, z)$ contain more than one element.

Examples :

- (i) The following are the transitions for a PDA (deterministic) which accepts $\{wcw^R / w \in (0+1)^*\}$ by empty stack (see ref[1]). Note that the rule $\delta(q_2, \epsilon, R) = (q_2, \epsilon)$ means that the PDA in state q_2 with R the top stack symbol, can erase the R independently of the input symbol. In this case, the read head is not advanced, and infact, there need not be any remaining input.

$M = (\{q_1, q_2\}, \{0, 1, c\}, \{R, B, G\}, \delta, q, R, \Phi)$ where,

$\delta(q_1, 0, R) = (q_1, BR) : \delta(q_1, 1, R) = (q_1, GR) :$

$\delta(q_1, 0, B) = (q_1, BB) : \delta(q_1, 1, B) = (q_1, GB) :$

$\delta(q_1, 0, G) = (q_1, BG) : \delta(q_1, 1, G) = (q_1, GG) :$

$\delta(q_1, c, R) = (q_2, R) : \delta(q_1, c, B) = (q_2, B) :$

$\delta(q_1, c, G) = (q_2, G) :$

$\delta(q_2, 0, B) = (q_2, \epsilon) : \delta(q_2, 1, G) = (q_2, \epsilon) :$

$\delta(q_2, \epsilon, R) = (q_2, \epsilon) .$ ■

An **Instantaneous Description (ID)** is used to describe the configuration of a PDA at any instant formally. If M is a PDA then $(q, aw, z\alpha) \vdash (p, w, \beta\alpha)$ if $\delta(q, a, z)$

contains $(p, \beta)(\text{ref}[1])$. Define $L(M)$, the language accepted by 'final state' to be $\{ w / (q_0, w, z_0) \vdash (p, \epsilon, \alpha) \text{ for some } p \in F \text{ and } \alpha \in \Gamma^* \}$. Define $N(M)$, the language accepted by 'empty stack (or null stack)' to be $\{ w / (q_0, w, z_0) \vdash (p, \epsilon, \epsilon) \text{ for some } p \in Q \}$.

For FA, the deterministic and nondeterministic models were equivalent with respect to the language accepted. The same is not true for PDA. Infact, for $\{ ww^R / w \in (0+1)^* \}$ we have the following nondeterministic PDA and there is no equivalent deterministic PDA.

$$\begin{aligned}
 M = (\{q_1, q_2\}, \{0, 1\}, \{R, G, B\}, \delta, q_1, R, \Phi) \text{ where,} \\
 \delta(q_1, 0, R) = (q_1, BR) : & \quad \delta(q_1, 1, G) = \{(q_1, GG), (q_2, \epsilon)\} : \\
 \delta(q_1, 1, R) = (q_1, GR) : & \quad \delta(q_2, 0, B) = (q_2, \epsilon) : \\
 \delta(q_2, 1, G) = (q_2, \epsilon) : & \quad \delta(q_1, 0, B) = \{(q_1, BB), (q_2, \epsilon)\} : \\
 \delta(q_1, 0, G) = (q_1, BG) : & \quad \delta(q_1, \epsilon, R) = (q_2, \epsilon) : \\
 \delta(q_1, 1, B) = (q_1, GB) : & \quad \delta(q_2, \epsilon, R) = (q_2, \epsilon) : \quad \blacksquare
 \end{aligned}$$

Equivalence of, acceptance by final state and empty stack and, PDA's and CFL's can be proved. (See ref[1] for proof and other details). Note that there are other variations of PDA such as two-stack PDA, which accepts a larger set of languages.

3.3 TURING MACHINES

The Turing machine, introduced by Alan Turing, exhibits many of the features commonly associated with a modern computer. Its significance for the theory of

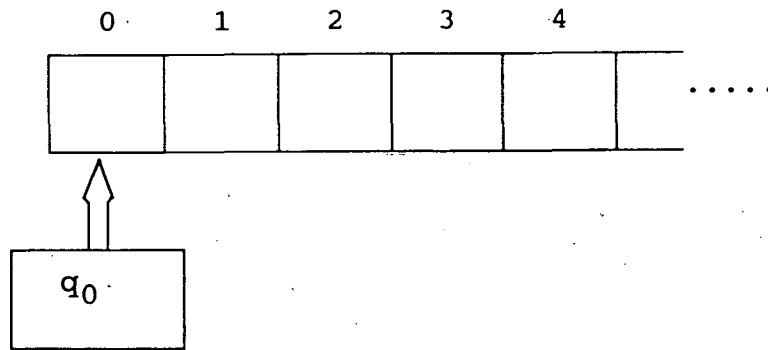
computing is fundamental: given a large but finite amount of time, the Turing machine is capable of any computation that can be done by any modern digital computer, no matter how powerful. A Turing machine is a finite-state machine in which a transition prints a symbol on the tape. The tape head may move in either direction, allowing the machine to manipulate the input as many times as desired. The structure of a Turing machine is similar to that of a finite automaton with the transition function incorporating these additional features.

A **Turing Machine(TM)** is a quintuple $M = (Q, \Sigma, \Gamma, \delta, q_0)$ where

- (i) Q is a finite set of states, Γ is a finite set called the 'tape alphabet', Γ contains a special symbol B that represents a 'blank'.
- (iii) Σ is a subset of $\Gamma - \{B\}$ called the 'input alphabet',
- (iv) $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L,R\}$ is a partial function,
- (v) $q_0 \in Q$ is a distinguished state called the 'start state'. ■

The tape of a Turing machine extends indefinitely in one direction. The tape positions are numbered by the natural numbers with the leftmost position numbered zero as shown below.

A computation begins with the tape head in state q_0 scanning the leftmost position. The input, a string from Σ^* , is written on the tape beginning at position one. Position zero and remainder of the tape are assumed to be



blank. The tape alphabet provides additional symbols that may be used during computation. A transition consists of three actions: changing the state, writing the symbol in the square scanned by the tape head, and moving the tape head. The transition $\delta(q_i, x) = (q_j, y, L)$ means that, the machine in state q_i with an input x , changes the state to q_j replacing x by y and moves the read head to the left square of the tape.

A Turing machine **halts** when it encounters a state, symbol pair for which no transition is defined. A transition from tape position zero may specify a move to the left of the boundary of the tape. When this occurs, the computation is to **terminate abnormally**. When we say that a computation halts, we mean that it terminates in a normal fashion. Turing machines are designed to perform computations on strings from the input alphabet. A computation begins with the tape head scanning the leftmost tape square with the input string beginning at position one. All tape squares to the right of the input string are assumed to be blank. The above defined Turing machine is called the **standard Turing machine**.

A machine configuration denoted $\alpha q_i \beta B$ where $\alpha \beta$ is the string spelled by the symbols on the tape from the left-hand boundary to the rightmost nonblank symbol. The notation $\alpha q_i \beta B \vdash \tau q_j \sigma B$ indicates that the configuration $\tau q_j \sigma B$ is obtained from $\alpha q_i \beta B$ by a single transition. Turing machines may be used as language acceptors; a computation accepts or rejects the input string. Initially, acceptance is defined by the final state of the computation. A Turing machine augmented with final states can be defined as follows.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0, F)$ be a Turing machine. A string $\alpha \in \Sigma^*$ is **accepted by final state** if the computation of M with input α halts in a final state. The language of M , $L(M)$, is the set of all strings accepted by M . ■

A language accepted by a Turing machine is called a **recursively enumerable language**. If the Turing machine halts for all input strings, the language is said to be **recursive**. The computations of a Turing machine provide a decision procedure for membership in a recursive language.

Examples :

(i) $M = (\{q_1, q_2, q_3, q_4, q_5\}, \{0, 1\}, \{0, 1, x, y, B\}, \delta, q_0, \{q_5\})$

where,

$\delta(q_1, 0) = (q_2, x, R) : \quad \delta(q_1, y) = (q_4, y, R) :$
 $\delta(q_2, y) = (q_2, y, R) : \quad \delta(q_2, 0) = (q_2, 0, R) :$
 $\delta(q_2, 1) = (q_3, y, L) : \quad \delta(q_3, 0) = (q_3, 0, L) :$
 $\delta(q_3, y) = (q_3, y, L) : \quad \delta(q_3, x) = (q_1, x, R) :$
 $\delta(q_4, B) = (q_5, B, R) :$



Dissertation
681.3.06
v59
un

This TM accepts the language $\{ 0^n 1^n / n \geq 1 \}$. The acceptance of the string 0011 is as follows.

$$\begin{array}{l}
 q_1 0011B \quad | \quad xq_2 011B \\
 \quad \quad \quad | \quad x0q_2 11B \quad | \quad xq_3 0y1B \\
 \quad \quad \quad | \quad q_3 x0y1B \quad | \quad xq_1 0y1B \\
 \quad \quad \quad | \quad xxq_2 y1B \quad | \quad xxyq_2 1B \\
 \quad \quad \quad | \quad xxq_3 yyB \quad | \quad xq_3 xyyB \\
 \quad \quad \quad | \quad xxq_1 yyB \quad | \quad xxyq_4 yB \\
 \quad \quad \quad | \quad xxyyq_4 B \quad | \quad xxyyBq_5
 \end{array}$$

Languages can be recognized by Turing machines without requiring the addition of final states. The alternative approach accepts a string if computation generated by the string causes the Turing machine to halt.

Let $M = (Q, \Sigma, \Gamma, \delta, q_0)$ be a Turing machine. A string $\alpha \in \Sigma^*$ is **accepted by halting** if the computation of M with input α halts.

There are various versions of TM such as Multitrack machines, NonDeterministic TM, Two way tape machines, Multitape machines, Atomic TM, Context sensitive TM. It can be proved that all these machines accept precisely the recursively enumerable languages.

References : [1]

Hopcroft, John E. and Ullman, Jeffrey D. : *Introduction to Automata Theory, Languages and Computation*. Narosa Publishing House, 1988, Ed. 2.

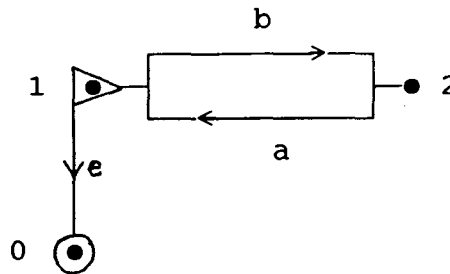
Chapter 4

NU MACHINE

In this chapter, a profound analysis of NM is made. Before going into details, studying an example will be useful. All the notations and its meanings are given in detail to avoid ambiguity. At the end, possibilities of restricting NM to PDA and FA are analysed with examples.

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 .

4.1 DEFINITION OF NM

An NM is a labeled digraph with the following properties.

- (i) There are two sorts of edges - dotted edges and line edges (i.e. ----- & _____).
- (ii) Edges are labeled in the form a / α , where $a \in (\Sigma \cup \{e\})$ and $\alpha \in \Sigma^*$.
- (iii) Two different edges can have the same label.
- (iv) Nodes are labeled $0, 1, 2, \dots$.
- (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 blank. If there is no symbol from the specified alphabet on the right of the pointer it means the machine reads 'e'.

$\frac{a/\alpha}{\text{-----}}$: 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 the right of ' α '.

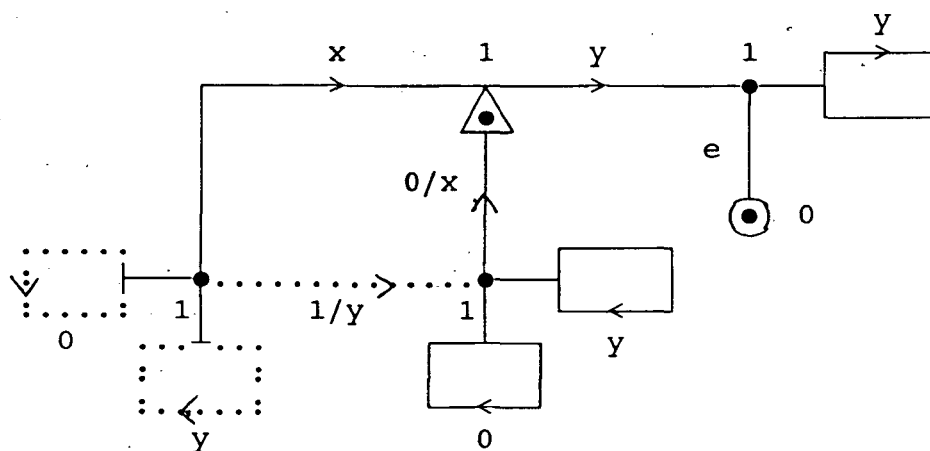
$\frac{a/\alpha}{\text{-----}}$: If the symbol on the right of the pointer in the node is 'a' then replace 'a' with the string ' α ' and move the pointer to one position left of ' α '.

$\underline{\quad\quad\quad}^a$: 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 .

$\underline{\quad\quad\quad}^a$: 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**.

Note that the example given in the first chapter is an example of NM which computes recursive functions. NM for recursive functions will be defined later with some restrictions in the above definition. Albeit the definition of NM for recursive functions is sufficient for designing an NM for accepting a language, more complicated definition is needed for easy understanding. NM for the language $\{ 0^n 1^n / n \geq 1, \Sigma = \{0,1\} \}$ is given below.



On further restrictions in the above definition, PDA and FA can be designed for corresponding languages. This is given at the end of this chapter with an example each. Now the definition of NM for recursive functions is defined. Note that the alphabet is restricted to $\{a, b\}$, there are no dotted edges and no a/α labels, and out degree of every node (other than final node) is either one or two.

4.2 NM FOR RECURSIVE FUNCTIONS

An NM is a labeled 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 labeled 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. Meaning of the symbols a, b and e are mentioned in the first 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

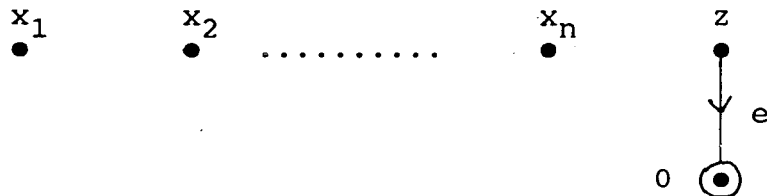
number of nodes, each node taking values from the set of natural numbers. Let the function to be calculated be $f(x_1, x_2, \dots, x_n)$. Initially the variable values are kept in the first n nodes (except 0). Answer can be in any of the nodes except input nodes which is specified before. If the output 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.

4.3 NM'S FOR ELEMENTARY FUNCTIONS

(i) **Zero function :**

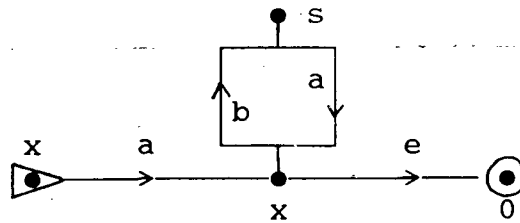
$$Z(x_1, x_2, \dots, x_n) = 0.$$



x_1, x_2, \dots, x_n are stored in nodes $x_1, x_2, x_3, \dots, x_n$ 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 :**

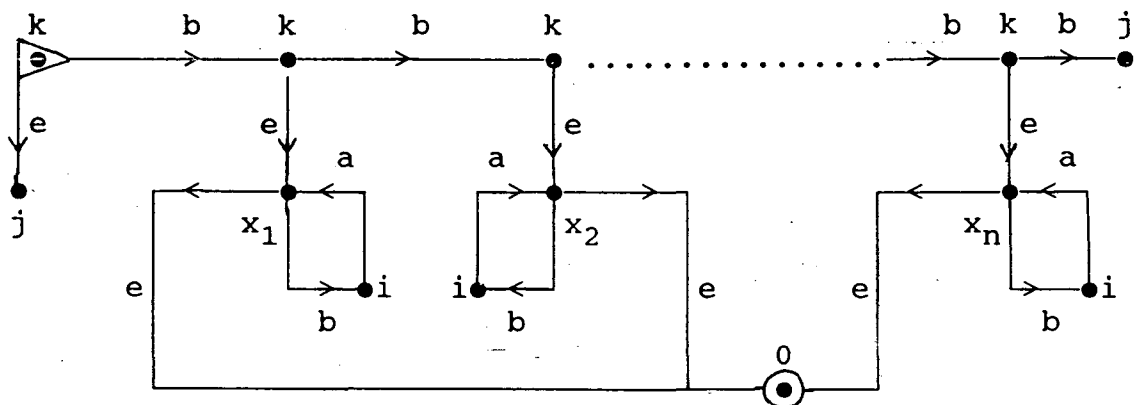
$$S(x) = x'$$



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.

(iii) **Identity function :**

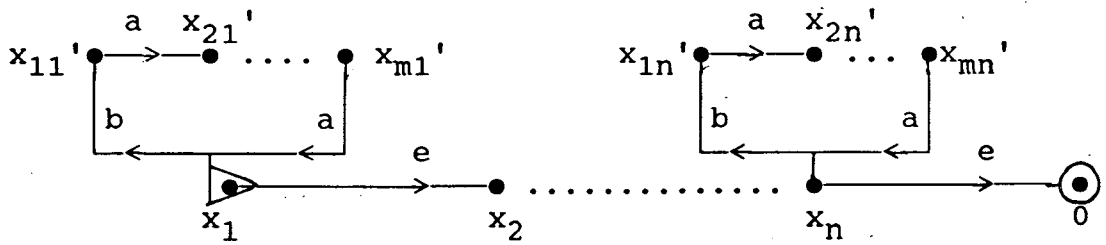
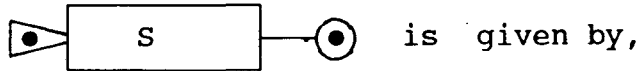
$$U_k^n(x_1, x_2, \dots, x_n) = x_k$$



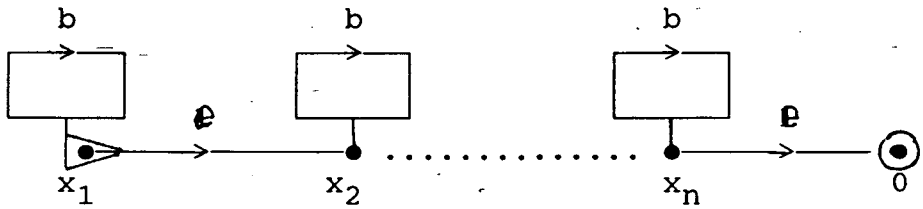
The value k is put in node k. In each step of the computation the value gets decreased until the value of node k becomes zero. When it happens the machine will be in node x_k where the value x_k is kept. Then it transforms the value x_k into the output node i. (Note that all the nodes other than input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

4.4 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 $S(x, x_1', x_2', \dots, x_m')$ 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 $x_1', x_2' \dots x_m'$. The machine,



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



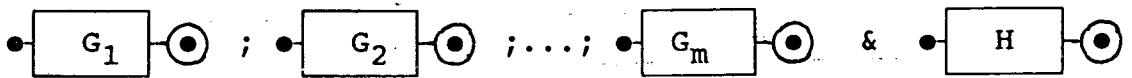
The above three machines are needed to restore the input values at the end of computation when the input nodes are being used. Here onwards only the names of the above machines will be used instead of the entire machine. This will be clear when we study machines for elementary procedures which are used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

(i) **Composition :**

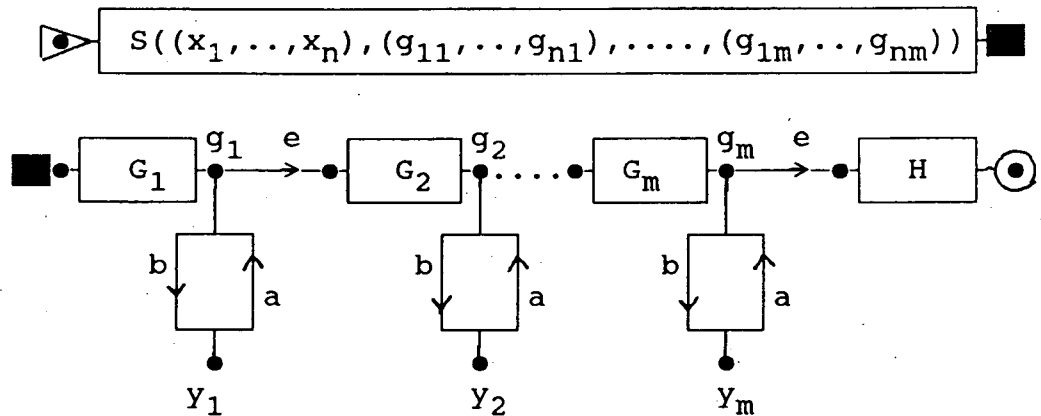
Given a set of functions $G_k (x_1, x_2, \dots, x_n)$, $k=1,2,\dots,m$ and $H (Y_1, Y_2, \dots, Y_m)$, define

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

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 , $1 \leq i \leq m$. Input nodes of G_i are denoted by $x_{1i}, x_{2i}, \dots, x_{ni}$. 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,

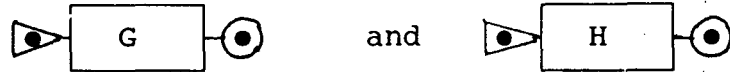


The following is the machine for composition. First block initialises the input nodes of G_i 's. Then in the second block each G_i is calculated and the result is transferred to y_i as the input of H . Final node of F will be the final node of H and the value of the function F will be in the output node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

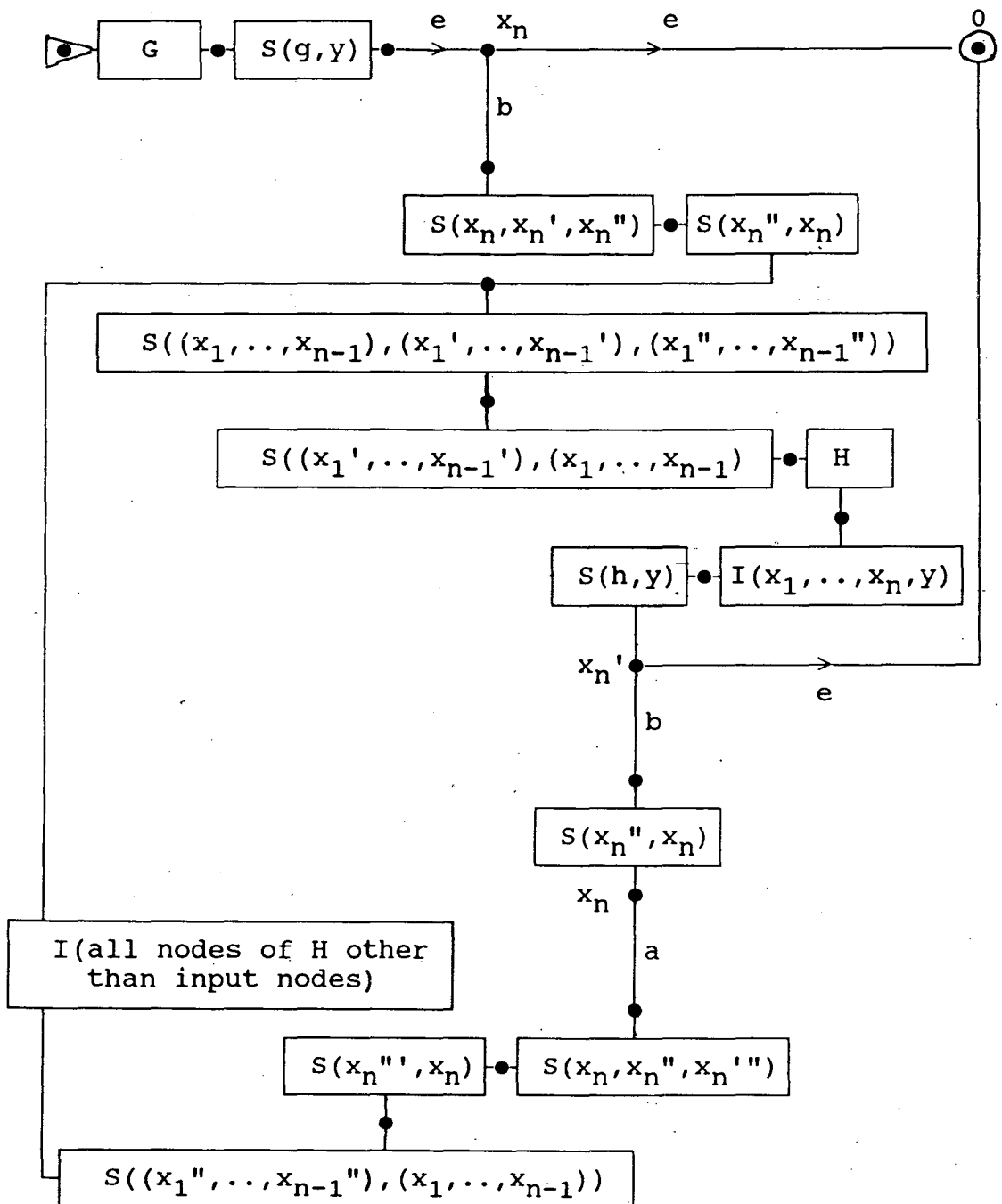


(ii) Primitive recursion :

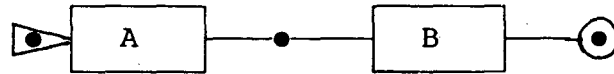
Given two functions $G(x_1, x_2, \dots, x_{n-1})$ and $H(x_1, x_2, \dots, x_n, y)$, define $F(x_1, x_2, \dots, x_n)$ as in the second chapter (section 2.2). The given functions are,



NM for this procedure is given below.



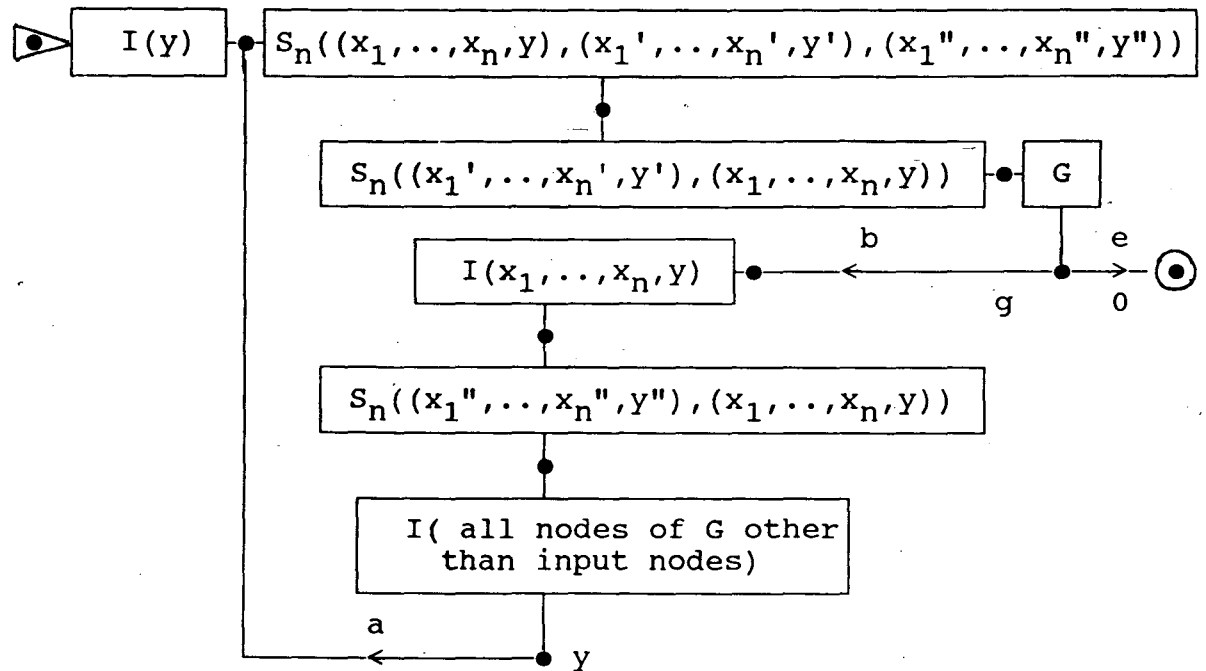
In the above machine, node 'g' is the output node of the machine G and node 'h' is the output node of H. Two machines A and B are interconnected to get C in the following fashion.



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.

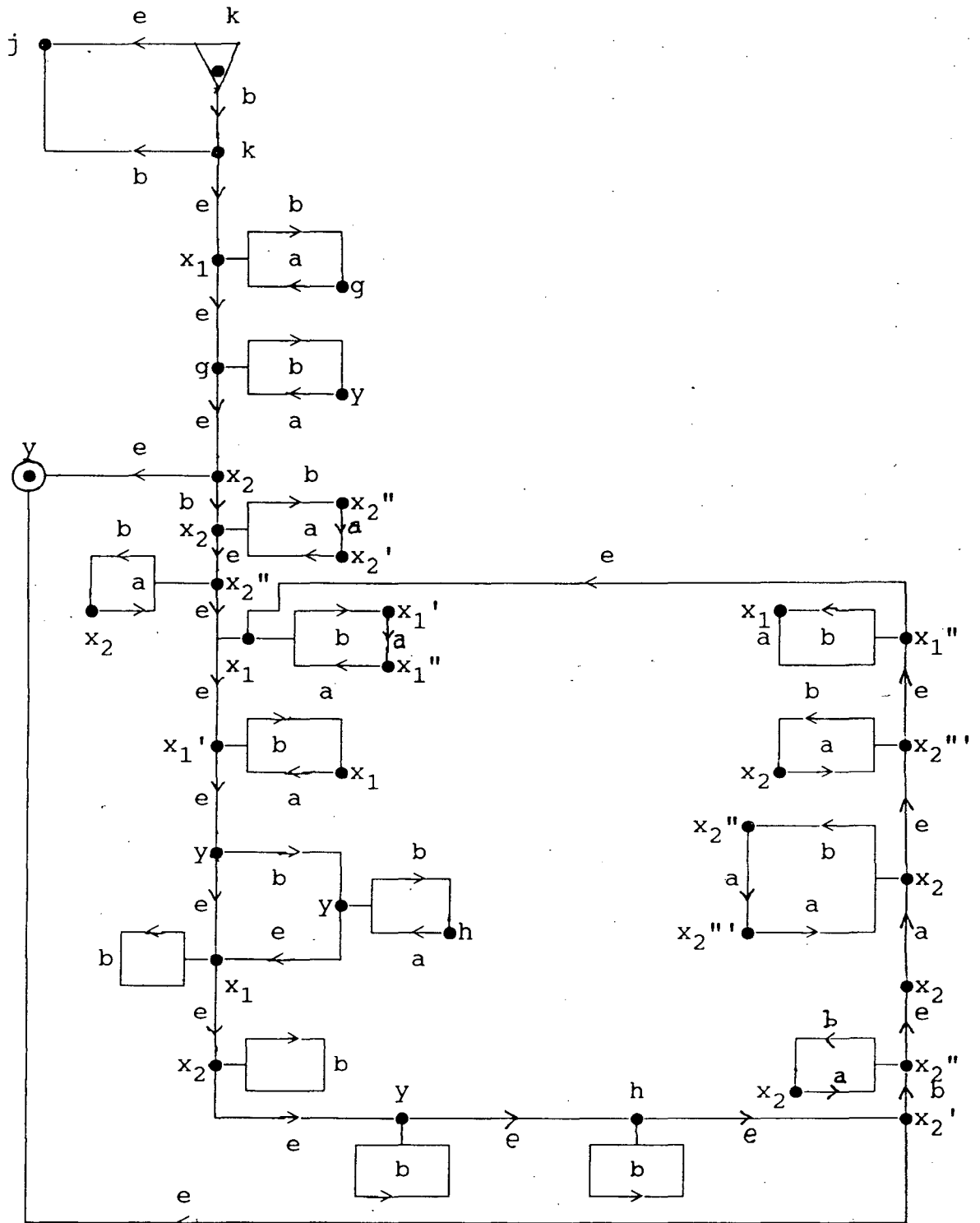
(iii) **Minimalisation :**

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

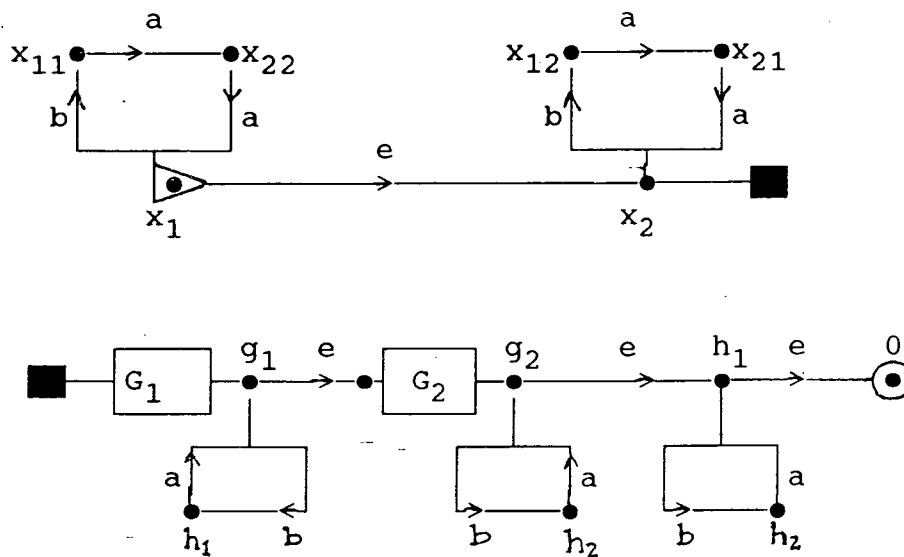


The following example illustrates how to make machines using above defined machines and how to combine

them. The following machine is for the proper subtraction defined in Chapter II (see example (vii) in Sec. 2.3). This function is used for calculating $|x_1 - x_2|$ which is the seventh example of the above mentioned section.



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 .

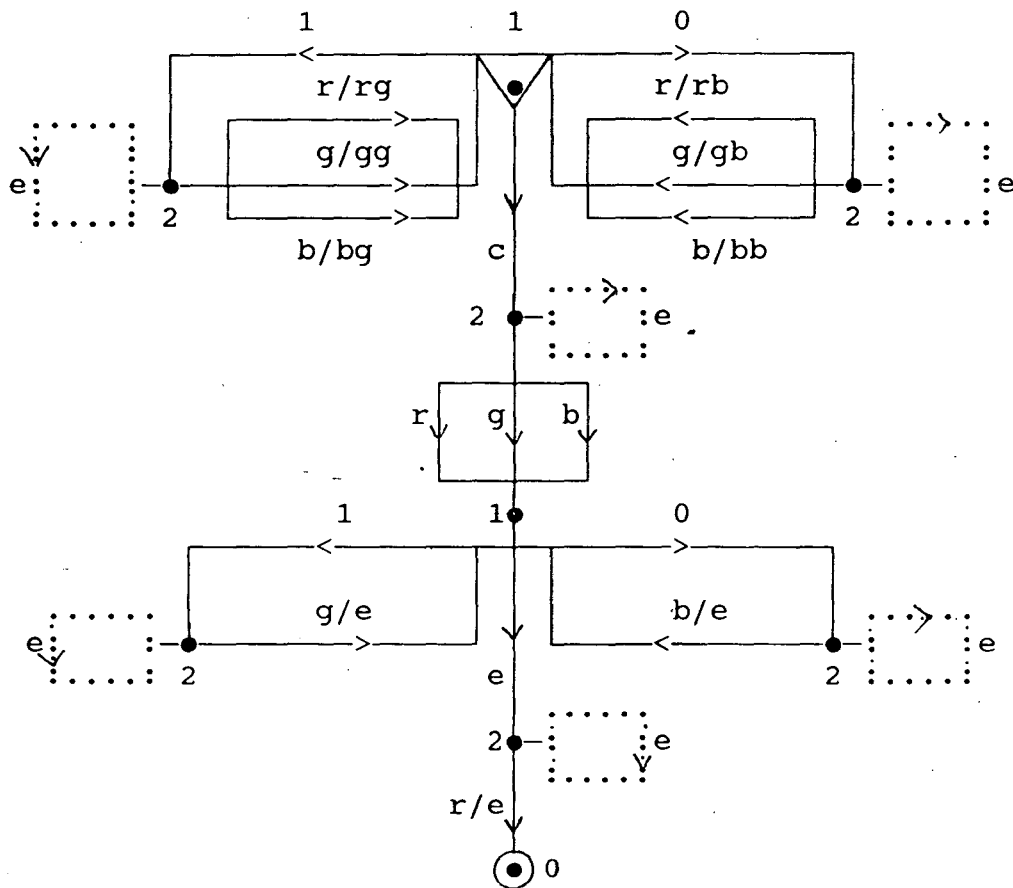
4.5 NM's CORRESPONDING TO PDA & FA

The machines corresponding to PDA and FA can be obtained by restricting the definition given in Sec.4.1 as follows.

NM for PDA is defined as NM with the following restrictions.

- (i) There are only two sorts of nodes. Node 1 represents the tape and the Node 2 represents the stack.
- (ii) There are no dotted edges with 'a/α' labels. All the dotted edges have a label 'e' and
- (iii) There are no dotted edges from Node 1.

The following machine illustrates the above definition, which accepts the CFL $\{ w c w^R / w \in (a+b)^* \}$.



The following table provides the steps involved in accepting the string 01c10.

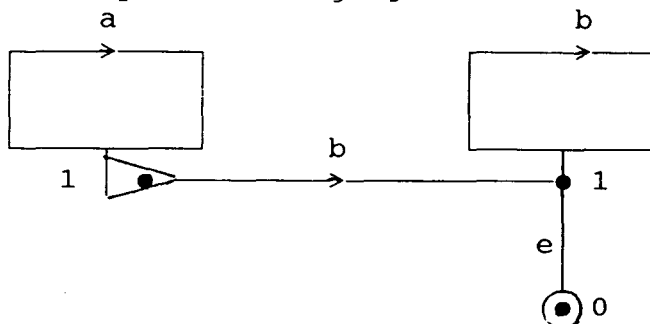
string in		present node
node 1	node 2	
$\hat{0}1c10$	\hat{r}	1
$0\hat{1}c10$	\hat{r}	2
$0\hat{1}c10$	$rb\hat{}$	1
$01\hat{c}10$	$rb\hat{}$	2
$01\hat{c}10$	$r\hat{b}$	2
$01\hat{c}10$	$rbg\hat{}$	1
$01c\hat{1}0$	$rbg\hat{}$	2
$01c\hat{1}0$	$rb\hat{g}$	2
$01c\hat{1}0$	$rbg\hat{}$	1
$00c1\hat{0}$	$rbg\hat{}$	2
$01c1\hat{0}$	$rb\hat{g}$	2
$01c1\hat{0}$	$rbe\hat{}$	1
$01c10\hat{}$	$rbe\hat{}$	2
$01c10\hat{}$	$rb\hat{}$	2
$01c10\hat{}$	$r\hat{b}$	2
$01c10\hat{}$	$re\hat{}$	1
$01c10\hat{}$	$re\hat{}$	2
$01c10\hat{}$	$r\hat{}$	2
$01c10\hat{}$	\hat{r}	2
$01c10\hat{}$	$e\hat{}$	0 (end).

On further restrictions NM for FA can be defined as follows.

NM for FA is a labeled digraph which have,

- (i) no dotted edges,
- (ii) no edges with a label of the form 'a/ α ',
- (iii) only labels of the form 'a' and
- (iv) only one sort of nodes.

The following example illustrates this. This machine accepts the language a^*bb^* .



It can be observed that the above machine is almost same as the DFA given in chapter III. Thus, a uniform model can be used instead of the current models of computation.



CONCLUSION

The purpose of this project was to design a uniform model for all computations instead of having three different models. It has been proved by showing the machines for the elementary functions and procedures, and how to connect them, that, the NU MACHINE is equally powerful as Turing machine and is simple to design. Hopcroft says in an article in Scientific American

'' If one is inclined to try building, say, the Turing machine that multiplies, one soon begins to appreciate the difficulties that must be faced in the design of a useful computer program. Most small Turing machines, namely the ones with only a few possible states, do not carry out any useful or even sensible task. ''

Moreover, we could see that NU MACHINE can be restricted to PDA and FA to have a uniform model for all computations. Thus, we have,

NM for FA	:	with one type of nodes,
NM for PDA	:	with two types of nodes and
NM	:	with many types of nodes.

Even though, the purpose of the project is fulfilled, there are much more things to do like restricting NM to obtain a model which accepts Context Sensitive Languages.

To get more clear idea of NM, demonstration programs - one for addition and one for multiplication - are included in the Appendices A and B at the end of this thesis.

BIBLIOGRAPHY

1. Boolos, George and Jeffrey, Richard : *Computability and Logic*. Cambridge University Press, 1974, Ed. 1.
2. Hergert, Douglas : *Mastering Turbo Pascal 5.0*. BPB Publications, 1989, Ed. 1.
3. Hopcroft, John E. and Ullman, Jeffrey D. : *Introduction to Automata Theory, Languages and Computation*. Narosa Publishing House, 1988, Ed. 2.
4. Rogers, David F. and Adams, Alan J. : *Mathematical Elements of Computer Graphics*. McGraw-Hill Publishing Company, 1990, Ed. 2.
5. Sudkamp, Thomas A. : *Languages and Machines*. Addison-Wesley Publishing Company, Inc., 1988.

APPENDICES

A.

PROGRAM FOR ADDITION

```

program ADDITION;
uses crt,graph,grworld;
var
    driver,mode:integer;
    ch,ch1:char;
    size1,size2,x,y,i,temp,i1,i2,ans:integer;
    ptr1,ptr2:pointer;
    n1,n2:array[1..100] of char;
    i2str,b1str,a1str,a2str:string;
{-----}
procedure CALCULATE;
var
    j,na,nb:integer;
begin
    na:=0;
    nb:=0;
    if x=0 then ans:=0
    else
    begin
        for j:=1 to i1 do
            if n1[j]='a' then na:=na+1
            else if n1[j]='b' then nb:=nb+1;
        ans:= na-nb;
    end;
end;
{-----}
function CHECK:boolean;
begin
    calculate;
    if ans>0 then check:=true
    else check:= false;
end;
{-----}
procedure DEFINEBALL;
begin
    setcolor(1);
    circle(10,20,5);
    setfillstyle(1,2);
    floodfill(10,20,1);
    size1:=imagesize(2,12,18,28 );
    getmem(ptr1,size1);
    getimage(2,12,18,28 ,ptr1^);
    size2:=imagesize(2,30,18,46);
    getmem(ptr2,size2);
    getimage(2 ,30,18,46,ptr2^);
    putimage(2,12,ptr2^,normalput);
end;
{-----}

```

```

procedure MOVELINE;
var
  y1,k:integer;
begin
  k:=0;
  y1:=107;
  repeat
    putimage(153,y1,ptr2^,normalput);
    if k=6 then
      begin
        line(152,117,168,117);
        line(168,117,160,105);
        line(160,105,152,117);
        circle(160,112,3);
      end;
    line(160,117,160,y1+1);
    y1:=y1+2;
    putimage(153,y1,ptr1^,normalput);
    k:=k+1;
  until y1=173;
end;
{-----}
procedure MOVECIRCLE;
var
  k:real;
  x1,y1,k1:integer;
begin
  putimage(155,108,ptr1^,normalput);
  delay(25);
  putimage(155,108,ptr2^,normalput);
  line(152,117,168,117);
  line(168,117,160,105);
  line(160,105,152,117);
  circle(160,112,3);
  y1:=108;
  k1:=0;
  repeat
    putimage(205,120,ptr2^,normalput);
    str(i1-x,b1str);
    outtextxy(205,120,b1str);
    k:=(1/2)*ln(1600-(y1-80)*(y1-80));
    x1:=round(155-exp(k));
    putimage(x1,y1,ptr1^,normalput);
    y1:=y1-5;
    circle(160,80,40);
    putimage(x1,y1+5,ptr2^,normalput);
    if k1=2 then
      line(160,117,160,130);
    k1:= k1+1;
  until y1<42;

```

```

putimage(155,42,ptr1^,normalput);
delay(25);
putimage(155,42,ptr2^,normalput);
circle(160,46,3);
circle(160,80,40);
y1:=44;
repeat
    putimage(172,35 ,ptr2^,normalput);
    str(i2,a2str);
    outtextxy(172,35 ,a2str);
    k:=(1/2)*ln(1600-(y1-80)*(y1-80));
    x1:=round(155+exp(k));
    putimage(x1,y1,ptr1^,normalput);
    y1:=y1+5;
    circle(160,80,40);
    putimage(x1,y1-5,ptr2^,normalput);
until y1>103;
circle(160,80,40);
end;
{-----}
begin
    repeat
        writeln('Enter the numbers to be added');
        readln(x);
        readln(y);
        if x>y then
            begin
                temp:=x;
                x:=y;
                y:=temp
            end;
        driver:=1;
        mode:=4;
        initgraph(driver,mode,'b:');
        setgraphmode(cgac2);
        setcolor(1);
        defineball;
        rectangle(0,0,319,199);
        circle(160,80,40);
        circle(160,46,3 );
        circle(160,112,3 );
        circle(160,180,3 );
        circle(160,180,7 );
        line(160,112,160,180);
        line(152,117,168,117);
        line(168,117,160,112);
        b1str:=' ';
        a2str:=' ';
        i2str:=' ';
        outtextxy(143,120,'1');
    
```



```

outtextxy(155,35 , '2');
outtextxy(140,180, '0');
outtextxy(145,140, '1');
outtextxy(105,80 , 'b');
outtextxy(215,80 , 'a');
outtextxy(173,120, '[ , ]');
outtextxy(165,35 , '[ , ]');
outtextxy(205,120, '0');
outtextxy(195,35 , '0');
str(x, a1str);
outtextxy(180,120, a1str);
str(y, a2str);
outtextxy(172,35 , a2str);
ans:=0;
for i:=1 to 100 do
begin
    n1[i]:=' ';
    n2[i]:=' ';
end;
for i1:= 1 to x do
    n1[i1]:='a';
for i2:= 1 to y do
    n2[i2]:='a';
while check do
begin
    i1:=i1+1;
    n1[i1]:='b';
    i2:=i2+1;
    n2[i2]:='a';
    movecircle;

end;
moveline;
circle(160,180,10);
freemem(ptr1, size1);
freemem(ptr2, size2);
str(i2, i2str);
outtextxy(5,190, 'The answer is');
outtextxy(120,190, i2str);
readln;
closegraph;
writeln('Do you want to try more?[y/n]');
readln(ch);
until ch='n';
end.

```

B.

PROGRAM FOR MULTIPLICATION

```

program MULTIPLICATION(input,output);
uses graph,crt,grworld;
{-----}
type
  ch = array [1..210] of char;
{-----}
var
  n1,n2,n3,n4 : ch;
  driver,mode,x,y,temp,na,nb,size1,size2,ans,
  j1,j2,j3,j4,i: integer;
  ch1,a,b: char;
  ansstr,a1str,b1str,a2str,b2str,a3str,b3str,a4str:string;
  ptr1,ptr2: pointer;
{-----}
procedure INITIALISE;
var
  i:integer;
begin
  for i:= 1 to 210 do
    begin
      n1[i] := ' ';
      n2[i] := ' ';
      n3[i] := ' ';
      n4[i] := ' ';
      ansstr:= ' ';
      a1str := ' ';
      b1str := ' ';
      a2str := ' ';
      b2str := ' ';
      a3str := ' ';
      b3str := ' ';
      a4str := ' ';
    end;
end;
{-----}
procedure CALCULATE(arr1: ch; jj1:integer);
var
  i:integer;
begin
  na:=0;
  nb:=0;
  for i:= 1 to jj1 do
    if arr1[i]='a' then na:=na+1
    else if arr1[i]='b' then nb:=nb+1 ;
  ans:=na-nb
end;
{-----}

```

```

function CHECK(arr2:ch;jj2:integer):boolean;
begin
    calculate(arr2,jj2);
    if ans > 0 then check:=true
    else check:=false;
end;
{-----}
procedure DEFINEBALL;
begin
    setcolor(1);
    circle(10,150,5);
    setfillstyle(1,2);
    floodfill(10,150,1);
    size1:= imagesize(3,143,17,157);
    getmem(ptr1,size1);
    getimage(3,143,17,157,ptr1^);
    size2:=imagesize(3,159,17,173);
    getmem(ptr2,size2);
    getimage(3,159,17,173,ptr2^);
    putimage(3,143,ptr2^,normalput);
end;
{-----}
procedure MOVLIN(x1,y1,x2,y2:integer);
var
    i:integer;
begin
    i:=1;
    if( x1=x2) or (x1=250) then
    begin
        putimage(x1,y1,ptr2^,normalput);
        x1:=x1-5;
        x2:=x2-5;
        putimage(x1,y1,ptr1^,normalput)
    end;
    while (x1<>x2) or (y1<>y2) do
    begin
        if y1=y2 then
        begin
            putimage(x1,y1,ptr2^,normalput);
            x1:=x1+2;
            putimage(x1,y1,ptr1^,normalput);
            if i=7 then line(70,70,50,50);
            if i>3 then
            begin
                setcolor(1);
                putpixel(x1-4,y1,1)
            end
        end
    end
end

```

```

else if x1=x2 then
begin
    putimage(x1,y1,ptr2^,normalput);
    y1:=y1+1;
    putimage(x1,y1,ptr1^,normalput);
    if i>3 then
begin
        setcolor(1);
        putpixel(x1+5,y1-5,1)
end;
    if i=7 then
begin
        circle(150,150,3);
        line(140,155,150,140);
        line(150,140,160,155);
        line(160,155,140,155)
end
end
else if ((x1-x2)/(y1-y2))=1 then
begin
    putimage(x1,y1,ptr2^,normalput);
    y1:=y1-1;
    x1:=x1-1;
    putimage(x1,y1,ptr1^,normalput);
    if i>7 then
        line(150,150,x1 ,y1 );
    if i=20 then
begin
        line(150,150,150,185);
        circle(150,150,3);
        line(140,155,150,140);
        line(150,140,160,155);
        line(160,155,140,155)
end
end
else if ((x1-x2)/(y1-y2))=-1 then
begin
    putimage(x1,y1,ptr2^,normalput);
    y1:=y1+1;
    x1:=x1-1;
    putimage(x1,y1,ptr1^,normalput);
    if i>3 then
        line(250,50,x1+5 ,y1 )
end;
    i:=i+1
end
end;
{-----}

```

```

procedure MOVCIR(x5,y5,cx,cy:integer);
var
  xx,yy,ix:integer;
begin
  xx:=x5;
  yy:=y5;
  repeat
    begin
      ix:=0;
      putimage(xx,yy,ptr2^,normalput);
      yy:=yy-3;
      ix:=400-(yy-30)*(yy-30);
      xx:=cx-round(exp(ln(ix)/2));
      putimage(xx,yy,ptr1^,normalput);
      setcolor(1);
      circle(cx,30,20);
      if( yy<35) and (yy>30 ) then
        begin
          circle(cx,50,3);
          line(50,50,250,50);
          line(70,70,50,50);
          line(250,50,230,70)
        end;
      if (yy<20) and (yy>16) then
        circle(30,30,3)
    end
  until yy<=11;
  putimage(xx,yy,ptr2^,normalput);
  putimage(cx,10,ptr1^,normalput);
  if x5=50 then
    begin
      putimage(268,65,ptr2^,normalput);
      calculate(n3,j3);
      str(na,a3str);
      outtextxy(267,70,a3str)
    end
  else if x5=250 then
    begin
      putimage(292,65,ptr2^,normalput);
      calculate(n3,j3);
      str(nb,b3str);
      outtextxy(291,70,b3str)
    end;
  xx:=cx;
  yy:=10;
  repeat
    begin
      ix:=0;
      putimage(xx,yy,ptr2^,normalput);
      yy:=yy+3;

```

```

        ix:=400-(yy-30)*(yy-30);
        xx:=cx+round(exp(ln(ix)/2));
        putimage(xx,yy,ptr1^,normalput);
        setcolor(1);
        circle(cx,30,20);
        if (yy>15) and (yy<19) then
        begin
            circle(250,15,3)
        end;
        if (yy>40) and (yy<44) then
            circle(70,30,3)
        end
    until yy>=49;
    putimage(xx,yy,ptr2^,normalput);
    putimage(cx,50,ptr1^,normalput);
    if x5=50 then
    begin
        str(j4,a4str);
        putimage(100,21,ptr2^,normalput);
        outtextxy(99,25,a4str)
    end
    else if x5=250 then
    begin
        calculate(n2,j2);
        str(na,a2str);
        putimage(256,0,ptr2^,normalput);
        outtextxy(255,3,a2str)
    end;
    circle(cx,30,20)
end;
{-----}
begin
    repeat
        writeln('Enter the numbers to be multiplied with an ENTER in be
        readln(x);
        readln(y);
        if y>x then
        begin
            temp:=y;
            y:=x;
            x:=temp
        end;
        clrscr;
        driver:=1;
        mode:=4;
        initgraph(driver,mode,'b:');
        setgraphmode(CGAC2);
        setviewport(0,0,318,199,true);
        rectangle(0,0,317,198);
        setcolor(1);

```

```

initialise;
outtextxy( 120 , 5 , 'NU-Machine');
defineball;
circle(150,185,3);
circle(50 ,50 ,3);
circle(30 ,30 ,3);
circle(70 ,30 ,3);
circle(250,50 ,3);
circle(250,15 ,3);
circle(150,150,3);
setfillstyle(1,1);
floodfill(150,185,1);
floodfill(50,50,1);
floodfill(30,30,1);
floodfill(70,30,1);
floodfill(250,50,1);
floodfill(250,15,1);
floodfill(150,150,1);
circle(150,185,7);
circle(50,30,20);
circle(250,30,20);
line(150,150,150,185);
line(150,150,50 ,50 );
line(50,50,250,50 );
line(250,50,150,150);
line(140,155,150,140);
line(150,140,160,155);
line(160,155,140,155);
outtextxy(173,150,'1[ ,0 ]');
putimage(188,147,ptr2^,normalput);
str(x,a1str);
outtextxy(188,150,a1str);
outtextxy(35 ,63,'2');
outtextxy(3,30,'3');
outtextxy(84,25,'4[0 ,0 ]');
outtextxy(252,70,'3[0 ,0 ]');
outtextxy(240,3,'2[ ,0 ]');
putimage(255,0,ptr2^,normalput);
str(y,a2str);
outtextxy(255,3,a2str);
outtextxy(165,185,'0');
outtextxy(80,100,'b');
outtextxy(180,100,'1');
outtextxy(140,40,'1');
outtextxy(285,30,'a');
outtextxy(215,30,'b');
outtextxy(20,42,'b');
outtextxy(55,33,'a');
outtextxy(55,1,'a');
outtextxy(160,165,'1');

```



```

j3:=0;      j4:=0;
for j1:= 1 to x do
  n1[j1]:='a';
for j2:= 1 to y do
  n2[j2]:='a';
readln;
while check(n1,j1) do
begin
  j1:=j1+1;
  n1[j1]:='b';
  putimage(211,150,ptr2^,normalput);
  str(j1-x,b1str);
  outtextxy(210,150,b1str);
  movlin(150,150 ,50 ,50);
  for i:=1 to 4 do
    putpixel(154-i,146+i,1);
  while check(n2,j2) do
  begin
    j2:=j2+1;
    n2[j2]:='b';
    j3:=j3+1;
    n3[j3]:='a';
    j4:=j4+1;
    n4[j4]:='a';
    putimage(279,3,ptr2^,normalput);
    calculate(n2,j2);
    str(nb,b2str);
    outtextxy(278,3,b2str);
    movcir(50,50,50,30);
  end;
  movlin(50,50 ,250 ,50);
  circle(50 ,50 ,3);
  setcolor(1);
  for i:=1 to 7 do
    putpixel(57-i,57-i,1);
  while check(n3,j3) do
  begin
    j3:=j3+1;
    n3[j3]:='b';
    j2:=j2+1;
    n2[j2]:='a';
    movcir(250,50,250,30);
  end;
  line(50,50,250,50);
  movlin(250,50 ,150 ,150);
  circle(250,50 ,3);
  setcolor(1);
  for i:=1 to 7 do
    putpixel(243+i,50,1);
end;
end;

```

```
putimage(140,178,ptr2^,normalput);
movlin(150,150 ,150 ,180);
line(150,150,150,180);
circle(153,187,7);
calculate(n4,j4);
str(ans,ansstr);
outtextxy(5,190, 'The answer is ');
outtextxy(120,190,ansstr);
freemem(ptr1,size1);
freemem(ptr2,size2);
readln;
closegraph ;
write('Do you want to try more ?[y/n] ');
readln(ch1);
until ch1='n';
end.
```