In this chapter, we give a mathematical model of computation called NuMachine which can compute all recursive functions. It will be seen that this model is very simple to visualize because of its graphical representation. An example of this model has been given in chapter 1. We begin with a discussion on recursive functions, first defining the elementary functions and elementary procedures, then going on to define primitive recursive functions. Then we present our model of computation and the algebra that goes along with it.

3.1 Recursive Functions

This section deals with one of the important concepts in the theory of computation. We use the definitions given in this section to show that a NuMachine can be constructed to compute a recursive function. All the functions defined here have the set of natural numbers as their domain.
A \textbf{zero function} is defined as $Z(x) = 0$. A \textbf{successor function} is defined as $S(x) = x'$ where $x'$ is the natural number next to $x$ in the sequence of natural numbers. An \textbf{identity function} is defined as $U_k(x_1, \ldots, x_k, \ldots, x_n) = x_k$ where $1 \leq k \leq n$. The above defined three functions are called \textit{elementary functions}.

It can be seen that $Z(0) = 0$, $Z(1) = 0$, and so on, i.e., for any input we have output 0. Successor function applied to a value $x$ will give $x + 1$ as the output. For example, $S(0) = 1$, $S(1) = 2$, and so on. Identity function (also called \textit{projection function}) projects the value of the $k^{\text{th}}$ variable as the output. $U_1(2) = 2$, $U_1(2, 3) = 2$, $U_2(2, 3) = 3$, and so on. For one variable, we simply write $U(x)$ instead of $U_1(x)$. Now we will define procedures to obtain a new function from a given set of functions.

\begin{itemize}
  \item \textit{Composition} of the functions $h(x)$ and $g(x)$ is defined as $f(x) = h \circ g = h(g(x))$. In general, for an $m$-ary function $h(x_1, \ldots, x_m)$, we should have $m$ $n$-ary functions $g_1(x_1, \ldots, x_n)$, $g_2(x_1, \ldots, x_n)$, $\ldots$, $g_m(x_1, \ldots, x_n)$, to obtain $f(x_1, \ldots, x_n)$ as $f(x_1, \ldots, x_n) = h(g_1, \ldots, g_m)$.

For example, let $g(x) = x + 1$ and $h(x) = 2x$. Then we have $f(x) = h(g(x)) = h(x + 1) = 2(x + 1)$.

\item Given two functions $g(x_1, \ldots, x_{n-1})$ and $h(x_1, \ldots, x_n, y)$, define a function $f(x_1, \ldots, x_n)$ as follows.

\begin{align*}
  f(x_1, \ldots, x_{n-1}, 0) &= g(x_1, \ldots, x_{n-1}) = g_0 \\
  f(x_1, \ldots, x_{n-1}, 1) &= h(x_1, \ldots, x_{n-1}, 0, g_0) = g_1 \\
  &\vdots \\
  f(x_1, \ldots, x_{n-1}, x_n) &= h(x_1, \ldots, x_{n-1}, x_n - 1, g_{x_{n-1}})
\end{align*}
\end{itemize}
The above procedure is called *primitive recursion*.

Consider \( g(x) = U(x) = x \) and \( h(x, y, z) = S(z) = z + 1 \) for defining \( f(x, y) = x + y \).

Using primitive recursion we get

\[
\begin{align*}
f(x, 0) &= g(x) = x = g_0 \\
f(x, 1) &= h(x, 0, g_0) = g_0 + 1 = x + 1 = g_1 \\
f(x, 2) &= h(x, 1, g_1) = g_1 + 1 = x + 2 = g_2 \\
\vdots & \quad \vdots \\
f(x, y) &= h(x, y - 1, g_{y-1}) = g_{y-1} + 1 = (x + y - 1) + 1 = x + y
\end{align*}
\]

- Given \( g(x_1, \ldots, x_n, y) \) we can construct

\[ f(x_1, \ldots, x_n) = \min \{ y | g(x_1, \ldots, x_n, y) = 0 \}. \]

If \( g = 0 \) is guaranteed, then it is *total minimalization*, otherwise it is *partial minimalization*.

For defining \( f(x) = \lfloor x^{1/2} \rfloor \), consider \( g(x, y) = x \div y^2 \) where \( \div \) stands for proper subtraction. Define \( f(x) = \min \{ y | x \div y^2 = 0 \} \). Since in this case we always have \( g = 0 \) at some point, it is total minimalization. But for \( f(x) = x^{1/2} \), we will not have a \( y \) for every \( x \) such that \( g(x, y) = |x \div y^2| = 0 \), hence we have a case of partial minimalization.

- All functions which are defined using elementary functions, composition, and primitive recursion are categorised as *primitive recursive functions*.

When we use total minimalization to define the function we get a *total recursive function*. Instead of total minimalization, if we are using only partial minimalization we obtain a *partial recursive function*. 
All functions like \( f(x, y) = x + y \), \( f(x, y) = x \cdot y \), \( f(x) = x' \), \( f(x, y) = x^y \), etc. are primitive recursive functions. Functions like \( f(x) = \lfloor x^{1/2} \rfloor \), \( f(x, y) = \lfloor x + y^{1/2} \rfloor \) are total recursive functions where as \( f(x) = x^{1/2} \), \( f(x) = x + y^{1/4} \) etc. are partial recursive functions\[48\]. Terms like recursive sets and recursively enumerable sets are defined in chapter 5.

### 3.2 NuMachine

This section discusses NuMachine in detail. It is shown that we can construct a NuMachine corresponding to every recursive function. Before going into the details, let us look at a few examples. The machine given in Fig. 3.1 computes the product of two numbers.

![Fig. 3.1](image)

Initially, nodes 1, 2, 3, and 4 have values \( x \), \( y \), 0, and 0 respectively. At the end of the computation when the machine halts at node 0, node 4 will have the answer \( x \cdot y \). The states of the above machine for inputs 2 and 2 is given below.
In the above table, △ indicates the present state of the machine. At the end of the computation the machine halts at node 0. Consider another example of NuMachine which is for exponentiation.

![Diagram of NuMachine and Recursive Functions](image-url)
Initially, nodes 1, 2, 3, 4, and 5 have values 0, y, x, 0, and 0 respectively. At the end of the computation the answer $x^y$ will appear in node 1. The steps of the machine can be easily verified.

- A *NuMachine* (NM) can be defined as a 5-tuple $(Q, \Sigma, q_0, I, q_u)$ along with a labeled digraph where $Q$ is the set of nodes, $\Sigma \subseteq \{a, b, e\}$ is the edge labels, $q_0$ is the initial node, $I$ is the set of input nodes and $q_u$ is the output node. The moves of NM are defined as follows. If the edge from node $i$ to node $j$ has the label $a$, then add an $a$ to the string at $i$ and move to $j$. If the label is $b$, then delete an $a$ from the string at $i$ and move to $j$. If the string at $i$ has zero length, i.e., no $a$'s, then move through the edge labeled $e$ without any action.

The initial node is placed inside a circle. The final nodes, where the machine halts at the end of computation, are large disks (filled circles) and are always numbered 0. No other node has the label 0. Node labels need not be distinct. A node has at the most two outgoing edges. No node can have two outgoing edges labeled $a$ and $b$. The machine can be visualized if we consider each node as a basket and each move as either adding a pebble (we assume to have an infinite number of pebbles) or removing a pebble. Computation halts if a wrong move is defined or the machine has reached one of the final states.

To see if NM can compute all recursive functions, we need to design NMs corresponding to elementary functions and procedures. The following sections discuss this in detail.

### 3.2.1 NM for Elementary Functions

The three elementary functions are zero function, successor function and identity function. In this section we define NMs for each of these functions.
NM for Zero Function

Zero function is defined as $Z(x) = 0$. The corresponding machine can be defined as $(\{0,1\}, \{b,e\}, 1, \{1\}, 1)$ and is given in Fig. 3.3.

\[ \begin{array}{c}
    b \quad 0 \\
    \downarrow \\
    1 \\
    \end{array} \quad \quad \begin{array}{c}
    e \\
    \downarrow \\
    0 \\
    \end{array} \]

Fig. 3.3

For any value $x$ put at node 1, we get the output 0 at node 1 itself when the computation halts at node 0. The machine simply throws all the pebbles into the infinite heap and halts at node 0. Since the output node is 1 itself, at the end of the computation we have the output 0.

NM for Successor Function

Successor function is defined as $S(x) = x'$ where $x'$ is the natural number succeeding $x$. Define the machine as $(\{0,1,2\}, \{a,b,e\}, 1, \{1\}, 2)$. The corresponding diagram is shown in Fig. 3.4.

\[ \begin{array}{c}
    1 \quad a \quad 1 \quad b \\
    \downarrow \\
    e \\
    \downarrow \\
    0 \\
    \end{array} \quad \quad \begin{array}{c}
    2 \\
    \end{array} \]

Fig. 3.4

For any value $x$ put in node 1, the machine first increases the value by 1 and then transfers it to the output node 2. When the length of the string becomes zero the machine moves to the final node and halts.
NM for Identity Function

Identity function is defined as $U_k(x_1, \ldots, x_n) = x_k$. First consider $U_k(x_1, x_2)$. The corresponding machine can be defined as $\langle \{0, 1, 2, 3, 4\}, \{a, b, e\}, 3, \{1, 2, 3\}, 4 \rangle$ and is given in Fig. 3.5.

Nodes 1 and 2 are reserved for values of $x_1$ and $x_2$. Node 3 takes the value of $k$. If the value of $k$ is 1, the machine goes to the node 1 and transfers the value of $x_1$ to the output node 4. If $k = 2$, the machine moves to node 2 and outputs the value $x_2$ at node 4. Any other value of $k$ makes the machine halt abruptly. In general, we have the following machine.

Note that, in general, while defining a function we specify the value of $k$ and so we should only keep the part corresponding to the $k^{th}$ node.
3.2.2 NM for Elementary Procedures

In this section we give procedures for constructing machines for each of the procedures, composition, primitive recursion and minimalization. We need three basic machines, \( I, T, S \) to make the construction easier. Machine \( I \) initializes the values of \( n \) nodes. It is given in Fig. 3.7.

Machine \( T \) transfers values of \( n \) nodes into another set of \( n \) nodes.

Machine \( S \) stores values of \( n \) nodes into another set of \( n \) nodes without altering the values of the first set of nodes. The machine is given in Fig. 3.9.

Notice the use of temporary nodes \( q_1', \ldots, q_n' \) to accomplish the storage. In Fig. 3.9 \( T \) transfers the values at \( q_1', \ldots, q_n' \) to nodes \( q_1, \ldots, q_n \).
NM for Composition

We are given \( m \) \( n \)-ary functions \( g_1, \ldots, g_m \) and \( h(x_1, \ldots, x_m) \). We can construct \( f(x_1, \ldots, x_m) = h \circ g = h(g_1, \ldots, g_m) \) using composition. For constructing NM for \( f \) we assume that we have machines corresponding to \( g_i \)s and \( h \). Let us call them \( G_1, \ldots, G_m \) and \( H \). Let the output node \( q_{ki} \)s of \( G_i \) be the \( i^{th} \) input node of \( H \). The output node of \( H \) gives the final output. Let the initial node of \( G_i \) be \( q_{1i} \) and that of \( H \) be \( q_{1h} \). The machine can be drawn as in Fig. 3.10.

\[
\begin{array}{c}
\text{Fig. 3.10} \\
\includegraphics[width=0.8\textwidth]{fig3_10.png}
\end{array}
\]

In Fig. 3.10, \( q_{gi} \) is the node just previous to the final node of \( G_i \) where \( G_i \) halts. Instead of the edge labeled \( e \) from \( q_{gi} \) to 0, we now have the edge from \( q_{gi} \) to the initial node of \( G_{i+1} \). From \( q_{gm} \) we now have an edge to \( q_{1h} \). The initial node of \( G_1 \), i.e., \( q_{11} \), will be the initial node of the whole machine. Note that the input values \( x_1, \ldots, x_n \) has to be put in input nodes of \( G_1, \ldots, G_n \) separately. To make the design procedure clear, let us look at the following example. Suppose \( g(x) = Z(x) \) and \( h(x) = x' \), then we can define the function \( f(x) = 1 \) (constant function) as \( h \circ g \), i.e., \( f(x) = h(g(x)) = 1 \). The corresponding NM can be constructed as follows.
machines \( G \) and \( H \) are given as

\[
G = (\{0, 1\}, \{b, e\}, 1, \{1\}, 1) \quad \text{and} \quad H = (\{0, 2, 3\}, \{a, b, e\}, 2, \{2\}, 3)
\]
as in Fig. 3.11.a and Fig. 3.11.b.

Then the machine for \( f \) will be as follows.

Notice that output node 1 of \( G \) will be the input node of \( H \) and we have connected the prefinal node 1 to the initial node of \( H \). It can be easily verified that the above machine will give an output 1 for any input value \( x \).

Let us consider another example. Let \( g_1(x) = Z(x) = 0, g_2(x) = x' \) and \( h(x, y) = U_k(x, y) \). Then we can construct

\[
f(x) = h \circ g
\]

\[
= U_k(0, x')
\]

\[
= \begin{cases} 
0, & \text{if } k = 1; \\
x', & \text{if } k = 2.
\end{cases}
\]
Machines $G_1$ and $G_2$ will be the same as the machines given in Fig. 3.11.a and Fig. 3.11.b. The machine $H$ is given in Fig. 3.13.

From $G_1$, $G_2$, and $H$ we can construct $f$ as given in Fig. 3.14.

In the above machine, input value $x$ will be put in node 1 and node 2 and the value of $k$ in node 4. Depending on the value of $k$ the machine will transfer the output of $G_1$ or $G_2$ which are in node 1 and node 3 respectively, into the output node 5.
NM for Primitive Recursion

Using the machines $I$, $T$, and $S$ we can construct the machine for primitive recursion as follows.

In the machine in Fig. 3.15, $I$ initialises all node values of $H$ to 0 except output node and the $n^{th}$ input node $q_{xn}$. 
S stores values at the first \( n - 1 \) input nodes of \( H \) to another set of \( n - 1 \) nodes, i.e., from \( q_{xi} \) to \( q'_{x'i} \), \( 1 \leq i \leq n - 1 \)

\( T_1 \) transfers value at the output node of \( G \) to the \( n + 1 \)th input node of \( H \)

\( T_2 \) transfers value at the output node of \( G \) to output node of \( H \)

\( T_3 \) transfers values from \( q'_{x'i} \) to \( q_{xi} \), \( 1 \leq i \leq n - 1 \)

\( T_4 \) transfers value at the output node of \( H \) to \( n + 1 \)th input node of \( H \).

In the beginning, we input \( x_1, \ldots, x_{n-1} \) to corresponding input nodes of \( G \) and input nodes of \( H \), value \( x_n \) to \( q'_{x_n} \), and 0 to all other nodes. At the end of the computation we will have the output at \( q_{xH} \) -- the output node of \( H \). Note that we can omit the storage machine and some of the transfer machines if we are not damaging the value at the input nodes of \( H \) in between the computation. Let us look at an example. Define \( f(x) = x - 1 \) (proper subtraction) using \( g = 0 \) and \( h(x, y) = U(x) = x \). We have

\[
\begin{align*}
f(0) &= g = 0 = g_0 \\
f(1) &= h(0, g_0) = 0 = g_1 \\
f(2) &= h(1, g_1) = 1 = g_2 \\
&\vdots \\
f(x) &= h(x - 1, g_{x-1}) = x - 1
\end{align*}
\]

The machine \( (\{0, 1, 2, 3, 4, 5, 6, 7\}, \{a, b, e\}, 1, \{6\}, 4) \) corresponding to \( f \) is given in Fig. 3.16.
Fig. 3.16

Fig. 3.17
The machine in Fig. 3.17 is a simplified version of the machine in Fig. 3.16. It is obtained by removing redundant nodes 1, 3, 5, 3, and 6 and some redundant edges. When computing \(3 \div 1\) the moves of the above machine are as follows.

<table>
<thead>
<tr>
<th>Input Nodes</th>
<th>Output Node</th>
<th>(\bullet)</th>
<th>Final State</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>6</td>
<td>4</td>
</tr>
</tbody>
</table>

**Initial State**

- Node \(1\) \(\uparrow a^3\) \(\rightarrow 1\)
- Node \(1\) \(\uparrow a^2\) \(\rightarrow 1\)
- Node \(1\) \(\uparrow a\) \(\rightarrow 1\)
- Node \(1\) \(\uparrow a\) \(\rightarrow 1\)
- Node \(1\) \(\uparrow a\) \(\rightarrow 1\)
- Node \(1\) \(\uparrow a\) \(\rightarrow 1\)
- Node \(1\) \(\uparrow a\) \(\rightarrow 1\)
- Node \(1\) \(\uparrow a\) \(\rightarrow 1\)
- Node \(1\) \(\uparrow a\) \(\rightarrow 1\)

**Final State**

- Node \(1\) \(\uparrow 1\) \(\rightarrow a^2\)
- Node \(1\) \(\uparrow 1\) \(\rightarrow a^2\)

**NM for Minimalization**

In this section, we discuss how to construct a NM if the minimalization procedure is used. We again make use of the machines \(I\), \(T\), and \(S\).
We are given $g(x_1, \ldots, x_n, y)$ and the corresponding machine $G$. Construct a machine corresponding to $f(x_1, \ldots, x_n) = \min \{y | g(x_1, \ldots, x_n, y) = 0\}$ as given in Fig. 3.18. Input $x_i$ is placed in node $q_{x_i}, 1 \leq i \leq n$ and all other node values are made 0 before computation starts. In the figure,

$I$ initializes all node values of $G$

$S_1$ stores the values $x_1, \ldots, x_n$ in nodes $q'_{x1}, \ldots, q'_{xn}$ and value of $y$ at $q'_y$

$T_1$ transfers node values at $q'_{x1}, \ldots, q'_{xn}$ to $q_{x1}, \ldots, q_{xn}$

$T_2$ transfers value from $q'_y$ to $q_y$

$S_2$ stores value from $q_y$ to $q'_y$

The output of the machine will be at $q_{ag}$ (the output node of $G$) and for each $y$ starting from 0 we get an output at $q_{ag}$. If the sequence of outputs decreases and reaches 0, the machine stops. But if the output sequence for $y$ starting at 0 is something like 4,3,1,4,11,\ldots then the machine never stops. We have to see
where it changes from a decreasing sequence to an increasing one. That means for total minimalization the machine in Fig. 3.18 is sufficient, but not for partial minimalization. To control the infinite loop we can insert a machine which computes $U_i - U_j$ where $U_i$ is the output for $y$ and $U_j$ is the output for $y + 1$. When the sequence changes from decreasing to increasing, the output of $U_i - U_j$ becomes 0 and the machine halts.

We now discuss the algebra which goes along with NM and show how the moves of NM can be calculated in terms of new matrix operations.

### 3.3 NuAlgebra

In this section we will define three matrices, $A$, $b$, and $c$ which are related by the formula $Ab = c$ and are useful in calculating the moves of an NM. First let us look at the example of NM for addition given in chapter 1.

\[ \begin{align*}
1 & \quad \quad b \\
\downarrow & \quad \quad c \\
0 & \quad \quad a
\end{align*} \]

Fig. 3.19

The moves of the machine for input $x = 2$, $y = 1$ will be as follows.

<table>
<thead>
<tr>
<th>Input Nodes</th>
<th>Output Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Label</td>
<td>0</td>
</tr>
<tr>
<td>Initial State</td>
<td>$\triangleright a^2$</td>
</tr>
<tr>
<td>Final State</td>
<td>$\triangleright 1$</td>
</tr>
</tbody>
</table>
Define $A$ as the adjacency matrix of NM. For the above example we have

$$A = \begin{pmatrix}
1 & 2 & 0 \\
1 & 0 & b & e \\
2 & a & 0 & 0 \\
0 & 0 & 0 & 0
\end{pmatrix}$$

and input matrix $b$ as

$$b = \begin{pmatrix}
1 \\
1 & a^2 \\
2 & a \\
0 & 1
\end{pmatrix}$$

In $b$ each row corresponds to the input string of that node. 'Multiplying' (procedure to multiply $A$ and $b$ is given later) these two matrices, we get the output matrix $c$, i.e.,

$$\begin{pmatrix}
0 & b & c \\
a & 0 & 0 \\
0 & 0 & 0
\end{pmatrix} \begin{pmatrix}
a^2 \\
a \\
1
\end{pmatrix} = \begin{pmatrix}
1 \\
a \\
1
\end{pmatrix}.$$

The iterative steps involved in multiplication are given below:

$$\begin{pmatrix}
0 & b & c \\
a & 0 & 0 \\
0 & 0 & 0
\end{pmatrix} \begin{pmatrix}
a^2 \\
a \\
1
\end{pmatrix} \rightarrow \begin{pmatrix}
a \\
a \\
1
\end{pmatrix}$$

$$\begin{pmatrix}
0 & b & c \\
a & 0 & 0 \\
0 & 0 & 0
\end{pmatrix} \begin{pmatrix}
a \\
a \\
1
\end{pmatrix} \rightarrow \begin{pmatrix}
a \\
a^2 \\
1
\end{pmatrix}$$

$$\begin{pmatrix}
0 & b & c \\
a & 0 & 0 \\
0 & 0 & 0
\end{pmatrix} \begin{pmatrix}
a^2 \\
1
\end{pmatrix} \rightarrow \begin{pmatrix}
1 \\
a^2 \\
1
\end{pmatrix}$$

$$\begin{pmatrix}
0 & b & c \\
a & 0 & 0 \\
0 & 0 & 0
\end{pmatrix} \begin{pmatrix}
1 \\
a^2 \\
1
\end{pmatrix} \rightarrow \begin{pmatrix}
1 \\
a^3 \\
1
\end{pmatrix}$$

$$\begin{pmatrix}
0 & b & c \\
a & 0 & 0 \\
0 & 0 & 0
\end{pmatrix} \begin{pmatrix}
1 \\
a^3 \\
1
\end{pmatrix} \rightarrow \begin{pmatrix}
1 \\
a^3 \\
1
\end{pmatrix}$$
and we get $Ab = c$. Let us look at the example of NM for multiplication of two numbers. For writing the adjacency matrix we need to distinguish every node of the graph. For that, label all duplicate nodes of $q_i$ as $q_i', q_i''$, etc. The machine is shown in Fig. 3.20.

![Diagram](image)

From the diagram we can write the adjacency matrix $A$ as

$$A = \begin{pmatrix}
1 & 2 & 2' & 3 & 3' & 4 & 0 \\
1 & 0 & b & 0 & 0 & 0 & 0 \\
2 & 0 & 0 & 0 & b & e & 0 & 0 \\
2' & 0 & 0 & 0 & 0 & a & 0 & 0 \\
3 & 0 & 0 & 0 & 0 & 0 & a & 0 \\
3' & e & 0 & b & 0 & 0 & 0 & 0 \\
4 & 0 & a & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}$$
and the input matrix $b$ will be

$$
b = \begin{pmatrix}
1 & a_x \\
2 & a_y \\
2' & a_y \\
3 & 1 \\
3' & 1 \\
4 & 1 \\
0 & 1
\end{pmatrix}
$$

Multiplication $Ab$ will be as follows:

* In the first step we go along row 1 and find a $b$ at position 2. So we reduce $a^x$ to $a^{x-1}$ and then move to row 2.

* In step 2 we go along row 2 and find a $b$ at position 3. So we reduce the value $a^y$ to $a^{y-1}$ at both positions of $b$ corresponding to 2 and 2' since these two labels represent the same node.

* Proceeding in the same fashion, we end up at row 0 and have the resultant $b$ as the matrix $c$. These steps are shown below.

\[
\begin{pmatrix}
0 & b & 0 & 0 & 0 & 0 & e \\
0 & 0 & 0 & b & e & 0 & 0 \\
0 & 0 & 0 & 0 & a & 0 & 0 \\
e & 0 & b & 0 & 0 & 0 & 0 \\
0 & a & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
a^x \\
a^y \\
a^y \\
1 \\
1 \\
1
\end{pmatrix}
\rightarrow
\begin{pmatrix}
a^{x-1} \\
a^y \\
a^y \\
1 \\
1 \\
1
\end{pmatrix}
\]

\[
\begin{pmatrix}
0 & b & 0 & 0 & 0 & 0 & e \\
0 & 0 & 0 & b & e & 0 & 0 \\
0 & 0 & 0 & 0 & a & 0 & 0 \\
e & 0 & b & 0 & 0 & 0 & 0 \\
0 & a & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0
\end{pmatrix}
\begin{pmatrix}
a^{x-1} \\
a^y \\
a^y \\
1 \\
1 \\
1
\end{pmatrix}
\rightarrow
\begin{pmatrix}
a^{x-1} \\
a^y \\
a^y \\
1 \\
1 \\
1
\end{pmatrix}
\]

\vdots
\vdots
Note that no row of $A$ will have both $a$ and $b$. The algorithm can be stated as follows:

Start from the row corresponding to the initial node. Let it be row $k$

Repeat

If $b(k) \neq 1$ then

find the position in row $k$ at which an $a$ or a $b$ occurs. Let that position be $i$

If $A(k, i) = a$ then

add an $a$ to the string at $b(i)$ and all duplicate rows of $i$

else if $A(k, i) = b$ then

delete an $a$ from string at $b(i)$ and all duplicate rows of $i$

$k = i$

else if $b(k) = 1$ then

find the position in row $k$ at which an $a$ or an $c$ occurs. Let that position be $i$

If $A(k, i) = a$ then add an $a$ to the string at $b(i)$ and all duplicate rows of $i$

$k = i$

Until $A(k, j) = 0 \ \forall j = 1$ to $n$.

It should be clear that for every NM there is a corresponding $A$ and on multiplying $A$ with the input matrix $b$, we can obtain the output matrix $c$. Thus, every NM can be equivalently represented by its adjacency matrix $A$. 