A Study of Flow Languages, Vector Machines, Petri Nets and Convex Languages

Shyam Kishore Bajpai
Western Michigan University

Follow this and additional works at: https://scholarworks.wmich.edu/masters_theses

Part of the Computer Sciences Commons

Recommended Citation
https://scholarworks.wmich.edu/masters_theses/1635

This Masters Thesis-Open Access is brought to you for free and open access by the Graduate College at ScholarWorks at WMU. It has been accepted for inclusion in Master’s Theses by an authorized administrator of ScholarWorks at WMU. For more information, please contact maira.bundza@wmich.edu.
A STUDY OF FLOW LANGUAGES, VECTOR MACHINES, PETRI NETS AND CONVEX LANGUAGES

by

Shyam Kishore Bajpai

A Thesis
Submitted to the
Faculty of The Graduate College
in partial fulfillment of the
requirements for the
Degree of Master of Science
Department of Computer Science

Western Michigan University
Kalamazoo, Michigan
December, 1982
A STUDY OF FLOW LANGUAGES, VECTOR MACHINES, PETRI NETS AND CONVEX LANGUAGES

Shyam Kishore Bajpai, M.S.
Western Michigan University, 1982

In this work, relations have been established among vector machines and augmented vector machines, vector machines and Petri nets, and augmented vector machines and generalized Petri nets. At each stage, number of examples are given to illustrate the theory of Petri nets, vector machines and augmented vector machines.
ACKNOWLEDGEMENTS

I wish to express my deep sense of appreciation and gratitude to my supervisor Professor (Dr.) Monysios I. Kountanis for his valuable guidance, comments, suggestions and inspiring discussions during the course of this study and throughout my Computer Science career at Western Michigan University. I am also very grateful to Professor (Dr.) Alden Wright for his valuable, constructive and critical comments and also for his enthusiasm in correcting my English.

A special thanks goes to Mr. Gary Strader, Mr. Arif Azeem and to the Center of Educational Opportunities for reading a draft copy of my thesis and pointing out some corrections in my thesis.

My wife, Shobha, deserves a special recognition for her words of encouragement during the rough months of implementation.

Shyam Kishore Bajpai
INFORMATION TO USERS

This reproduction was made from a copy of a document sent to us for microfilming. While the most advanced technology has been used to photograph and reproduce this document, the quality of the reproduction is heavily dependent upon the quality of the material submitted.

The following explanation of techniques is provided to help clarify markings or notations which may appear on this reproduction.

1. The sign or “target” for pages apparently lacking from the document photographed is “Missing Page(s)”. If it was possible to obtain the missing page(s) or section, they are spliced into the film along with adjacent pages. This may have necessitated cutting through an image and duplicating adjacent pages to assure complete continuity.

2. When an image on the film is obliterated with a round black mark, it is an indication of either blurred copy because of movement during exposure, duplicate copy, or copyrighted materials that should not have been filmed. For blurred pages, a good image of the page can be found in the adjacent frame. If copyrighted materials were deleted, a target note will appear listing the pages in the adjacent frame.

3. When a map, drawing or chart, etc., is part of the material being photographed, a definite method of “sectioning” the material has been followed. It is customary to begin filming at the upper left hand corner of a large sheet and to continue from left to right in equal sections with small overlaps. If necessary, sectioning is continued again—beginning below the first row and continuing on until complete.

4. For illustrations that cannot be satisfactorily reproduced by xerographic means, photographic prints can be purchased at additional cost and inserted into your xerographic copy. These prints are available upon request from the Dissertations Customer Services Department.

5. Some pages in any document may have indistinct print. In all cases the best available copy has been filmed.

University Microfilms International
300 N. Zeeb Road
Ann Arbor, MI 48106

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
BAJPAI, SHYAM KISHORE

A STUDY OF FLOW LANGUAGES, VECTOR MACHINES, PETRI NETS AND CONVEX LANGUAGES

WESTERN MICHIGAN UNIVERSITY M.S. 1982

University Microfilms International 300 N. Zeeb Road, Ann Arbor, MI 48106
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTRODUCTION</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>II. THE PROBLEM — HISTORY AND SCOPE</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>Flow Expressions and Flow Languages</td>
<td></td>
<td>5</td>
</tr>
<tr>
<td>Vector Machine Ingredients</td>
<td></td>
<td>16</td>
</tr>
<tr>
<td>n-tuple Machine or Vector Machine</td>
<td></td>
<td>18</td>
</tr>
<tr>
<td>Extension Machine</td>
<td></td>
<td>25</td>
</tr>
<tr>
<td>III. PETRI NETS</td>
<td></td>
<td>28</td>
</tr>
<tr>
<td>Modelling With Petri Nets</td>
<td></td>
<td>35</td>
</tr>
<tr>
<td>Marked Petri Nets</td>
<td></td>
<td>42</td>
</tr>
<tr>
<td>Analysis of Petri Nets</td>
<td></td>
<td>46</td>
</tr>
<tr>
<td>Characteristics of Petri Nets</td>
<td></td>
<td>50</td>
</tr>
<tr>
<td>Matrix Approach for a Petri Net</td>
<td></td>
<td>54</td>
</tr>
</tbody>
</table>

ACKNOWLEDGEMENTS

TABLE OF CONTENTS
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Petri Net Language</td>
<td>59</td>
</tr>
<tr>
<td>Petri Net Languages by Means of Final Markings</td>
<td>64</td>
</tr>
<tr>
<td>Generalized Petri Nets</td>
<td>67</td>
</tr>
<tr>
<td>Convex Languages</td>
<td>71</td>
</tr>
<tr>
<td>BIBLIOGRAPHY</td>
<td>78</td>
</tr>
</tbody>
</table>
CHAPTER I

INTRODUCTION

Recently, Gischer (1981) introduced the notion of $L_{VM}$, the class of all languages $L$ such that $L = L(M)$ for some vector machine $M$. The notion of a vector machine is then introduced and extended to an augmented vector machine and $L_{VM}$ to $L_{VM+}$ the class of languages accepted by augmented vector machines. Gischer further proved that $L_{VM} \subseteq L_{VM+}$. However, the question of equality was left open. In this thesis, it is shown that $L_{VM}$ is strictly contained in $L_{VM+}$.

In order to complete the study of vector machines, a necessity developed for studying of Petri nets and generalized Petri nets in detail. Some theory of Petri nets is revealed in a paper by Peterson (1977). One should be cautious in reading this paper as some of the examples given in this paper are unclear and incomplete. A completion and correction of such examples is embodied in this thesis under Petri nets. Also a physical example of a computer program is translated into Petri
net language to indicate the power of Petri nets. This allowed me to make a remark concerning a good paper due to Shaw (1978) that shuffle languages can be translated in terms of Petri net languages which may shed more light on the structure of Petri nets.

Finally, convex languages are brought into notice and a question is posed of characterizing convex languages in terms of Petri nets.
CHAPTER II

THE PROBLEM -- HISTORY AND SCOPE

In this chapter we shall introduce the theory of flow languages and flow expressions which are generalizations of regular languages and regular expressions. In order to examine these concepts a simplified version is first introduced. This simplified version of flow expression is called shuffle expression and the corresponding language is called a shuffle languages. Recently, Gischer (1981) introduced the notion of vector machines which recognize the shuffle languages. In order to advance this theory, we shall first examine the vector machines. First of all, we shall show that the vector machines recognizing shuffle languages are not unique; then we shall consider extended vector machines and show that the languages accepted by vector machines are strictly contained in the class of languages accepted by extended vector machines. This provides a solution to the problem of Gischer. A complete discussion of this will follow by considering the following sections:
Flow expressions and flow languages,

Vector machine ingredients,

n-tuple machine or vector machine,

Extended vector machine.

Since the notion of vector machines is a new one, the research scope in this direction is unlimited.
In this section we shall give the definition of flow expression and flow language as introduced by Shaw (1978). Then towards the end of this section a simplified version of these flows is considered.

Shaw has considered the flow expression as an extension of the regular expression to describe sequential and concurrent flows. The idea of this extension is really based on the study of the flow of a single entity through many components, such as the flow of a job through several processes, and also of studying the flow of several entities through a single component, such as flow of a resource through an allocation mechanism to several processes.

In order to define the flow expression, the following are needed:

Sequential Flow: It is defined by the following recursive definition over an alphabet E:

(i) Any atomic element, $a$, $a \in E$, is a flow expression.

(ii) $\lambda$, the symbol for null string is a flow expression.

(iii) If $S_1$ and $S_2$ are two flow expressions then their
concatenation $S_1 S_2$ is a flow expression.

(iv) If $S$ is a flow expression then its sequential closure $S^*$ is also a flow expression.

(v) If $S_1$ and $S_2$ are flow expressions then their union, $S_1 \cup S_2$, is a flow expression.

Also we indicate by $L(a) = \{a\}$, $L(\lambda) = \{\lambda\}$, $L(S_1 S_2) = \{xy : x \in L(S_1) \text{ and } y \in L(S_2)\}$, $L(S_1 \cup S_2) = \{x : x \in L(S_1) \text{ or } x \in L(S_2)\}$, $L(S^*) = L(S^0) \cup L(S^1) \cup \ldots \cup L(S^n) \cup \ldots = \bigcup_{i=0}^{\infty} L(S_i)$; $S_i = S S^{i-1}$, $i > 0$, the languages generated by (i) through (v).

Nonterminating Flow: A cyclic operation denoted by $\infty$, is a nonterminating repetition of its constituents. In this sense, if $S$ is a flow expression, then $S^\infty$ is also a flow expression.

Also $L(S^\infty) = L(SSS\ldots) = \{x_1 x_2 x_3 \ldots : x_i \in L(S) \text{ and } i = 1, 2, 3, \ldots\}$.

Concurrent and Interleaved Flow: Based on the idea of the equivalence of concurrent execution of two components to some sequential interleaved flow through both components, we may assume an interleaved model of parallelism and define the meaning of concurrent operators $\parallel$ and $\circ$ in terms of sequential flows,
which are also called shuffle and interleaved operators.

**Shuffle Operator** :- If $S_1$ and $S_2$ are flow expressions then their shuffle $S_1 \circ S_2$ is also a flow expression and is defined as follows:

(i) If $S_1$ and $S_2$ are atomic then $S_1 \circ S_2$ denotes the flow expression for which $L(S_1 \circ S_2) = \{S_1 S_2, S_2 S_1\}$.

(ii) If $S_1$ and $S_2$ are any two general flows then $S_1 \circ S_2$ denotes the flow expression for which $L(S_1 \circ S_2) = \{x_1 y_1 \ldots x_k y_k \ldots, x_i \in E^*, x_1 \ldots x_k \in L(S_1) \text{ and } y_1 \ldots y_k \in L(S_2)\}$.

Note that in the above definition, by convention we let two dots after a string denote both finite and infinite length strings, as in $x_1 \ldots x_k \ldots$ and by convention ..., i.e., three dots, after a string denote just a continuous string, e.g., $x_1 \ldots x_k$ means continue the appearance of $x_2$ after $x_1$, $x_3$ after $x_2$, and so on until $x_k$.

**Example** :- If $S_1 = (ab)$ and $S_2 = (cd)$ then

$L(S_1 \circ S_2) = \{abcd, acbd, acdb, cabd, cdbd, cdab\}$.

**Interleaved Operator** :- The closure of a shuffle operator $\circ$
is denoted by the interleaved operator \( \oplus \), and is defined as follows:

\[
L(S \oplus i) = \bigoplus_{i=0}^{\infty} L(S \oplus i)
\]

where

\[
S \oplus i = \begin{cases} 
\lambda & \text{if } i = 0, \\
S \oplus i-1 \circ S & \text{if } i > 0.
\end{cases}
\]

Example: \((abc) \oplus\) with \(a = \text{logon}\), \(b = \text{edit}\) and \(c = \text{logoff}\) may describe the simultaneous logon, edit, and logoff activities of an arbitrary number of terminal users connected to CPU.

Restrictions: Locks and signals are used to restrict the set of described flows. Locks are used to handle critical sections whereas signals are used to provide a simple synchronization mechanism. In the following we shall describe formally both of these notions.

Locks: If a flow expression is enclosed within square brackets labelled by a name then such a flow or set of flows is called locked. Such a locked flow with the label is then
treated as atomic when interacting with other flows. If $\Gamma = \{ \{i, j\}, \ldots \}$ then $\Gamma$ is said to be a set of paired lock symbols. In this notation of $\Gamma$, if $\Gamma \cap E$ is $\{ \}$, then the definition of flow expression is extended as follows:

If $S$ is a flow expression then so is $L_k[S_k]$ for any pair $k \in \Gamma$. To define it precisely, we need the following:

1. The shuffle of $(k[S][S]_k)^* \cup (k[S][S]_k)$ represents $S_{\text{lock}}$ over all locks $k$.

2. Suppose $E$ has been enlarged to include all non operator symbols to include the member of $\Gamma$. Then with this convention, we denote by $\hat{L}(S)$, the unrestricted set of flows represented by flow expressions of $S$. If $S$ does not contain any lock symbols, then

$$
\hat{L}(k[S]_k) = \{ k[x]_k : x \in L(S) \}.
$$

In general

$$
\hat{L}(S) = \{ x = x_1x_2 \ldots x_k \ldots : x_i \in E^*, \text{ } z = x_1y_1 \ldots x_ky_k \ldots \in \hat{L}(S), \text{ } y_i \in \Gamma^* \text{ and } y_1 \ldots y_k \ldots \in \hat{L}(S_{\text{lock}}) \}.
$$
Example: 

(i) \( L(\bigcup_k [ab]_k \cap [cd]_k) = \{abcd, cdab\} \).

(ii) \( L(\bigcup_i [ab]_i \cap \bigcup_j [cd]_j) = \{abcd, acdb, acbd, cdab, cabd, cadb\} \).

(iii) \( L(\bigcup_i [a_j [bc]_d]_i \cap \bigcup_j [ef]_i \cup [gh]_j) = \{abcdef, efabcd, abcdgh, abcghd, abcghd, aghbcd, gahbcd, ghabcd\} \).

(iv) \( L(\bigcup_i [a_i [bc]_d]_i) = \{\} \).

The last example (iv) corresponds to an empty flow set because

\[ L(\bigcup_i [a_i [bc]_d]_i) = \{\} \]

and

\[ [i[ ]]_i \notin L(S_{lock}). \]

If a situation corresponding to (iv) appears then it is called a deadlock. Thus, \( L(S) = \{\} \) is a deadlocked flow.

Waits and Signals: Synchronization of flows are discovered by wait \( W \) and signal \( Q \). Consequently, \( W_i \) means wait on signal \( i \) and \( Q_i \) means send signal \( i \). A wait/signal set is defined by

\[ \Omega = \{W_i, Q_i, W_j, Q_j, \ldots\} \]

and

\[ E \cap \Omega = \{\} \].
Let us represent shuffle of \((Q_k^* W_k^*)^* Q_k^* \bigcup (Q_k^* W_k^* \bigcup Q_k^*)^\infty\)
over all signals \(k\) by \(S_{\text{signal}}\). Precisely, it is defined by

\[ S_{\text{signal}} = (((Q_i^* W_i^*)^* Q_i^* \bigcup (Q_i^* W_i^* \bigcup Q_i^*)^\infty) \circ \((Q_j^* W_j^*)^* Q_j^* \bigcup (Q_j^* W_j^* \bigcup Q_j^*)^\infty\) \circ \ldots \]  

and finally, \(L(S)\) is defined by

\[ L(S) = \{ x = x_1 \ldots x_k \ldots : x_i \in E^*, z = x_1 y_1 \ldots x_k y_k \ldots, \]
\[ z \in \overset{\wedge}{L}(S), y_i \in (\bigcup \bigcup \Gamma)^* \text{ and} \]
\[ y_1 \ldots y_k \ldots \in L(S_{\text{lock}} \circ S_{\text{signal}}) \]  

Examples:  
(i) \(L((aw_ib) \circ (dQ_i)) = \{ adb, dab \} \).

(ii) \(L(Q_2((C1W_1I1Q_1)^\infty \circ (C2W_1R2Q_2)^\infty)) \)

= \(\{ C1I1C2R1 \ldots , C2C1I1R2 \ldots , \ldots \} \).

(iii) \(L(Q_k((W_kabQ_k) \circ (W_kcdQ_k))) = \{ abcd, cdab \} \).

(iv) \(L(aW_ibQ_i) = \{ \} \).

Now, we are set to define flow expressions and flow languages.

This is done as follows:

Let

\(E = \) alphabet or entity symbols,

\(\Gamma = \{ i[,], j[,], \ldots \} = \) set of lock symbols,

\(\mathcal{W} = \{ W_i, Q_i, W_j, Q_j, \ldots \} = \) set of wait/signal symbols.
Flow expressions are defined by:

(i) For all a ∈ E, a is a flow expression.

(ii) For all a ∈ Ω, a is a flow expression.

(iii) If S₁, S₂, and S₃ are flow expressions then S₂ S₃, S₂ ∪ S₃, S₁*, S₁⁻, S₂ ∩ S₃ and S₁ ⊕ are flow expressions.

(iv) If S is a flow expression and k [·]ₖ is a pair of lock symbols in Γ then k[S]ₖ is also a flow expression.

Flow languages are defined by the following two steps:

First Step: Here, we define the language L(S) which imposes no restrictions on the lock and signal symbols.

CASE S OF

a ∈ E ∪ Γ ∪ Ω : ^L(S) = {a};

λ : ^L(S) = {λ};

φ : ^L(S) = {};

S₁ S₂ : {xy : x ∈ ^L(S₁) and y ∈ ^L(S₂)} = ^L(S);

S₁ ∪ S₂ : {x : x ∈ ^L(S₁) or x ∈ ^L(S₂)} = ^L(S);

S₁* : ^L(S) = ∞ \bigcup_{i=0}^{∞} ^L(S₁ⁱ);
$S_1 \ominus S_2 : \mathcal{L}(S) = \{x_1 y_1 \ldots x_k y_k \ldots : x_1 \ldots x_k \in \mathcal{L}(S_1) \text{ and } y_1 \ldots y_k \ldots \} \in \mathcal{L}(S_2)\}$;

$S_1 \ominus : \mathcal{L}(S) = \bigcup_{i=0}^{\infty} \mathcal{L}(S \ominus i)$;

$S_1 \ominus \infty : \mathcal{L}(S) = \{x_1 x_2 x_3 \ldots : x_i \in \mathcal{L}(S_1)\}$;

END.

Second Step := $L(S)$ is obtained by $L(S)$ by imposing locks and signals. This is done by

$L(S) = \{x = x_1 \ldots x_k \ldots : x_i \in E^*, z = x_1 y_1 \ldots x_k y_k \ldots, z \in \mathcal{L}(S), y_i \in (\bigcup \Upsilon \Gamma)^* \text{ and } y_1 \ldots y_k \ldots \} \in \mathcal{L}(S_{\text{lock}} \ominus S_{\text{signal}})\}$.

This completes the definition of flow languages and flow expressions. Now we shall define shuffle expressions and shuffle languages along with a simplification of the flow expressions and flow languages as introduced above.

Shuffle Expression := We say $S$ is a shuffle expression over $E$, if and only if the following is true:

(i) $S = a$ if $a \in E$.

or
(ii) If $S_1$ and $S_2$ are shuffle expressions over $E$ then $S$ is given by either one of (1) through (6).

(1) $S = S_1S_2$.

(2) $S = S_1 \cup S_2$.

(3) $S = S_1^*$.

(4) $S = S_1 \bigcirc S_2$.

(5) $S = S_1 \boxtimes S_2$.

(6) $S = (S_1)$.

Shuffle Language: For each shuffle expression $S$, we define an associated shuffle language $L(S)$ as follows:

(1) If $S = a$ and $a \in E$ then $L(S) = \{a\}$.

(2) If $S_1$ and $S_2$ are shuffle expressions then $x \in L(S)$ if and only if either one of the following is true:

(i) $S = S_1S_2$ and $x = yz$ for some $y \in L(S_1)$ and $z \in L(S_2)$.

(ii) $S = S_1 \cup S_2$ and $x \in L(S_1)$ or $x \in L(S_2)$.

(iii) $S = S_1^*$ and $x = \lambda$ or $x = y_1y_2\ldots y_n$, for some $n \geq 1$ where each $y_i \in L(S_1)$, $1 \leq i \leq n$.

(iv) $S = S_1 \bigcirc S_2$ and $x = y_1z_1y_2z_2\ldots y_nz_n$, $n \geq 1$, $y_1y_2\ldots y_n \in L(S_1)$ and $z_1z_2\ldots z_n \in L(S_2)$. 

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
\[(v) \quad S = S_1 \otimes \text{ and } x = \lambda \text{ or } x \in L(S_1 \otimes^n) \text{ for some } n \geq 1 \]

where \(S \otimes^n\) is defined by \(S \otimes^1 = S\) and \(S \otimes^j = S \cup S \otimes^{j-1}\) for all \(j \geq 2\).

\[(vi) \quad S = (S_1) \text{ and } x \in L(S_1)\).

This completes the definitions for shuffle expressions and shuffle languages.
This section contains the required ingredients for defining the vector machines, which are also called n-tuple machines.

A transformation $t$ of a n-tuple consists of an ordered pair of multisets of functions $(P_{i_1}, P_{i_2}, ..., P_{i_k}, v_{j_1}, v_{j_2}, ..., v_{j_l})$: $1 \leq i_r \leq n$, $1 \leq j_r \leq n$ and $r = 1, 2, 3, ...$). The primitives of these multisets are defined by

$$P_i(b) = \begin{cases} (b_1, ..., b_{i-1}, b_{i-1}, b_{i+1}, ..., b_n) & \text{if } b_i > 0 \\ \text{undefined if } b_i = 0 \end{cases}$$

and

$$v_i(b) = (b_1, ..., b_{i-1}, b_{i+1}, b_{i+1}, ..., b_n).$$

Obviously, we have

$$P_{i j} P_{j i} = P_{i j} P_{i j}$$

and

$$v_{i j} v_{j i} = v_{i j} v_{j i}.$$
Thus if

\[ t : \mathbb{N}^n \rightarrow \mathbb{N}^n \]

and

\[ t(b) = \left\{ v_{j_1}, v_{j_2}, \ldots, v_{j_k} \right\} \left[ \left\{ p_{i_1}, p_{i_2}, \ldots, p_{i_k} \right\}(b) \right] \]

then this function is well defined.

If \( T^n \) denotes the set of all transformations defined on \( n \)-tuples, then \( T^n \subseteq T^{n+1} \) in the sense that if \( t \in T^n \) then \( t' \in T^{n+1} \) where \( t'(b, n) = (t(b), n) \).
This section contains a complete definition of a vector machine. A proof follows that shows that vector machines are not unique.

A n-tuple machine is defined by a four tuple \((E, R, S, H)\) where

- \(E\) is a finite set,
- \(R\) is a finite subset of \((E \cup \{\lambda\})^n\)
- \(S\) is in \(\mathbb{N}^n\)
- \(H \subseteq \mathbb{T}^n\).

We denote this machine by \(M\), and list the following definitions:

1. A sequence \(r_1, r_2, \ldots, r_p\) from \(R\) is called a computational sequence of \(M\).

2. If \(r_i = (a_i, t_i), 1 \leq i \leq p\), and there exists \(h_j\) in \(\mathbb{N}\), \(1 \leq j \leq q; q \geq 0\) such that \((h_q \ldots h_1, t_p \ldots t_1(S))\) is well defined and is equal to \(0 = (0, \ldots, 0) \in \mathbb{N}^n\); then \(r_1r_2, \ldots, r_p\) is called an accepting sequence and \(\{h_j\}\) is called a clearing sequence.
3. If \( x = a_1 a_2 \ldots a_p \) then \( r_1, r_2, \ldots, r_p \) is called an accepting sequence if condition 2 is satisfied for \( x \in M \). In that case \( t_p \ldots t_1 \) is called an accepting function for \( x \) and \( h_q \ldots h_1 \) is called a clearing function of \( r_1 \ldots r_p \).

4. If \( L(M) \) is contained in \( E^* \) and consists of precisely of those elements of \( E^* \) which have accepting sequences then it is called a set of languages accepted by \( M \) and \( L(M) \) is reserved for such a sequence, i.e., \( L(M) \) indicates the languages accepted by \( M \).

Example 1: Let \( M = (E, R, S, H) \) where

\[
E = \{a, b, c\}
\]

\[
S = (0, 0, 0, 1, 0, 0)
\]

\[
R = \langle \langle \{a, \{p_4\}, \{v_1, v_4\}\} \rangle, \langle b, \{p_1, p_5\}, \{v_2, v_5\}\rangle, \\
\langle c, \{p_2, p_6\}, \{v_3, v_6\}\rangle, \langle \lambda, \{p_4\}, \{v_5\}\rangle, \\
\langle \lambda, \{p_5\}, \{v_6\}\rangle \rangle
\]

\[
= (r_1, r_2, r_3, r_4, r_5)
\]

and

\[
H = \langle \langle \{p_3\}, \{\}\rangle, \langle \{p_6\}, \{\}\rangle \rangle
\]

\[
= (h_1, h_2)
\]
where \( \{ \} \) for \( p \) or \( v \) denotes the identity function. Then
\[
L(M) = \{ a^k b^k c^k : k \geq 0 \}.
\]

Proof: First suppose that any sequence begins with

(a) \( r_2 \), then
\[
(v_2, v_5) \left[ \{ p_1, p_5 \} \right](S) = (v_2, v_5)(-1, 0, 0, 1, -1, 0)
\]
which is undefined.

(b) \( r_3 \), then
\[
(v_3, v_6) \left[ \{ p_2, p_6 \} \right](0, 0, 0, 1, 0, 0)
\]
is undefined.

(c) \( r_5 \), then
\[
(v_6) \left[ \{ p_3 \} \right](0, 0, 0, 1, 0, 0)
\]
is undefined.

(d) \( r_4 \), then
\[
(v_5) \left[ \{ p_4 \} \right](S) = (0, 0, 0, 0, 1, 0).
\]

Clearly, after \( r_4 \), \( r_1 \), \( r_2 \), \( r_3 \) cannot be used as they will lead
to undefined sequences. But \( r_5 \) may occur since
\[
\begin{align*}
 r_5 r_4(S) &= r_5(0, 0, 0, 0, 1, 0) \\
 &= (0, 0, 0, 0, 0, 1).
\end{align*}
\]
This implies that a possible string starting with $r_4$ corresponds to $h_2t_5t_4(S)$ and is $\lambda$. Hence, $\lambda \in L(M)$.

Now, if any sequence begins with $r_1$ then

$$r_1(S) = (1, 0, 0, 1, 0, 0)$$

and so

$$r_1^k(S) = (k, 0, 0, 1, 0, 0).$$

No clearing sequence exists at this stage. Also, after $r_1$, only $r_4$ is defined. Hence, we have

$$r_4r_1^k(S) = (k, 0, 0, 0, 1, 0).$$

At this stage there are two possibilities:

1. $t_5$ may be applied then $b_1 = k$ and $b_i = 0$ for $1 < i < 4$

and $b_5 = 1$, and an application of $h_2$ implies $b_1 = k$ and $b_i = 0$ for $1 < i < 5$. Now there are no sequences which can be applied on it to make $b_1 = 0$. Thus, this sequence is not acceptable.

2. $t_2$ may be applied then

$$r_2r_4r_1^k(S) = (k-1, 1, 0, 0, 1, 0).$$

Also

$$r_2^kr_4r_1^k(S) \notin (0, k, 0, 0, 1, 0),$$
An application of \( t_5 \) implies

\[
r_6 r_2 k r_4 r_1 k(S) = (0, k, 0, 0, 0, 1).
\]

An application of \( t_5 \) now gives that

\[
r_6 r_2 k r_4 r_1 k(S) = (0, k, 0, 0, 0, 0).
\]

An application of \( h_2 \) leads to an undefined sequence, however an application of \( r_3 \) leads to

\[
r_3 k r_6 r_2 k r_4 r_1 k(S) = (0, 0, 0, 0, 0, 1)
\]

and so

\[
h_1 r_3 k r_6 r_2 k r_4 r_1 k(S) = (0, 0, 0, 0, 0, 0).
\]

Hence we conclude that the accepting sequence leads to

\[
L(M) = \{a^k b^k c^k : k \geq 0\}.
\]

Example 2:- Let \( N = (E, R, S, H) \) where

\[
E = \{a, b, c\}
\]

\[
S = (0, 0, 0, 1, 0, 0)
\]

\[
R = (\{a, (\{p_4\}, \{(v_2, v_4)\})\},
\]

\[
(b, (\{p_2, p_5\}, \{(v_3, v_5)\}))\},
\]

\[
(c, (\{p_5, p_3\}, \{(v_5)\}))\},
\]

\[
(\lambda, (\{p_4\}, \{v_5\})),
\]

\[
= (r_1, r_2, r_3, r_4)
\]
and

\[ H = (\Lambda, \{p, q\}, \{r, s\}) \]

\[ = \{h_1\} \]

where \( \{\} \) for \( p \) or \( v \) denotes the identity function. Then

\[ L(M) = \left\{ a^kb^kc^k : k \geq 0 \right\}. \]

Proof: - No sequence can start with \( r_2 \) and \( r_3 \). If a sequence starts with \( r_4 \) then the possible next sequence is \( h_1 \) leading to \( \Lambda \in L(M) \).

If the sequence starts with \( r_1 \) then we have

\[ t_1^k(S) = (0, k, 0, 1, 0, 0). \]

At this stage the next possible starting for a sequence is \( r_4 \).

We then get

\[ t_4t_1^k(S) = (0, k, 0, 0, 1, 0). \]

At this stage the next possible sequences are \( h_1 \) and \( r_2 \).

If clearing sequence \( h_1 \) is applied then it leads to

\[ h_1t_4t_1^k(S) = (0, k, 0, 0, 0, 0) \]

which leaves the next sequence undefined, and therefore no clearing sequences are possible. Hence, this sequence is not an accepting sequence. With \( r_2 \) we have
\[ t_2^k t_4^k t_1^k (S) = (0, 0, k, 0, 1, 0). \]

Again the next possible sequences are \( h_1 \) and \( r_3 \). With \( h_1 \), all possible sequences are undefined. Hence there is no accepting sequence with \( h_1 \). But with \( r_3 \) we have

\[ t_3^k t_2^k t_4^k t_1^k (S) = (0, 0, 0, 0, 0, 0). \]

Further note that there are no more accepting sequences that are possible. Thus we get

\[ L(M) = \left\{ a^k b^k c^k : k \geq 0 \right\}. \]

This completes the proof of example 2.

As a consequence of examples 1 and 2 we have:

**Theorem 1** :- For given \( E, S \), the n-tuple machine, \((E, R, S, H)\) where \( E \) is finite, \( R \) is a finite subset of \((E \cup \{ \lambda \}) \times T^N, S \in N^n \) and \( H \subseteq T^N \), which generate \( L(M) \) is not unique.
Extension Machine

This section contains an extended definition of vector machines as introduced in the previous section. In this section it is shown that the class of languages accepted by an extended machine contains the class of languages accepted by a vector machine.

Let
\[ L_{VM} = \{ L : L = L(M) \text{ for some vector machine } M \} \]

Also let us define
\[ \overline{p}_i : N^n \to N^n ; 1 \leq i \leq n \]
by
\[ \overline{p}_i(b) = b \text{ if and only if } b_i = 0 \text{ else undefined.} \]

Now augmented transformations
\[ t : N^n \to N^n \]
are the same as ordinary transformations introduced earlier except that \( p \) is augmented with \( \overline{p} \). Clearly
\[ p_ip_i \neq \overline{p}_i \overline{p}_i \text{ for all } i, 1 \leq i \leq n. \]
Let us define $T^{n^+}$ by

$$T^{n^+} = \{ p : p \text{ may not contain both } p_i \text{ and } \overline{p_i} \text{ for all } i, 1 \leq i \leq n \text{ defined on } n\text{-tuple} \}.$$

Finally, define an augmented vector machine to be a four-tuple $M = (E, R, S, H)$ where $E$ and $S$ are defined as before and $R \subseteq (E \cup \{ \lambda \}) \times T^{n^+}$ and $H \subseteq T^{n^+}$.

Also let $L_{VM^+} = \{ L : L(M) \text{ for some augmented vector machine} \}$.

In [78] the following question is posed:

Problem:—Prove or disprove $L_{VM} = L_{VM^+}$.

In the following we shall prove that $L_{VM} \neq L_{VM^+}$.

Theorem 2:—$L_{VM}$ is strictly contained in $L_{VM^+}$.

Proof:—It is obvious that $L_{VM}$ is contained in $L_{VM^+}$.

In order to show the strict containment, the following example is produced.

Example:—Let $M = (E, R, S, H)$ where

$$E = \{ a \}$$

$$S = (0, 0, 0, 1, 0, 0)$$

$$H = \{ \{ p_4 \}, \{ \} \} = \{ h_1 \}$$
and

\[ R = ((a, \{ p_4 \}, \{ \alpha \}), (a, \{ \overline{p}_4 \}, \{ \beta \})). \]

Then

\[ L_{VM} = \{ a^2 \} \quad \text{and} \quad L_{VM^+} = \{ a^k : k \geq 1 \}. \]

Proof: In this case we obtain

\[ \overline{p}_4(S) = (0, 0, 0, 1, 0, 0) \]

and therefore

\[ t_4^k(S) = (0, 0, 0, 1, 0, 0) \]

and by \( h_1 \) we get

\[ h_1 \overline{t}_4^k(S) = (0, 0, 0, 0, 0, 0). \]

Hence \( L_{VM^+} = \{ a^k : k \geq 1 \} \). But \( L_{VM} \) is obviously equal to \( \{ a^2 \} \).

This completes the proof of the example and also that of theorem 2.
CHAPTER III

PETRI NETS

A Petri net is an abstract, formal model of information flow. Shortly, a simple and explanatory example of this net will be considered. The example will be represented by a graphical picture. There are two types of properties involved with the Petri nets. The first type is called static and is similar to the flow chart of a computer program. The second type resembles the execution of the computer program and is called a dynamic property.

The graph representing the static property of a Petri net contains two types of nodes. These nodes are represented by circles and bars. The circles are used to represent places and bars are used to represent transitions. Places and transitions are connection by direct arcs in either direction. If the arc is directed from node $i$ to node $j$ then node $i$ is called the input and node $j$ is called an output of $i$. 

28
The execution of a Petri net is controlled by the position and movement of markers, called tokens. Graphically, tokens are indicated by black dots and are placed within the circles to indicate their place in the Petri net.

We distinguish static and dynamic Petri nets by naming them Petri nets and marked Petri nets. Following are the rules for movements in Petri nets:

1. Tokens are moved by the firing of transitions.

2. A transition is said to be enabled if all of its input places have a token in them.

3. If transitions are enabled then they have the right to be fired.

4. When transitions are fired, each enabling token is moved from its input place to the output places of the transitions. If there are more than one output places, a token goes into each output place.

5. The distribution of tokens in a marked Petri net defines the state of the net and is called its marking.
6. There may be more than one marking and these are obtained as long as enabled transitions are available.

7. If $t_1$ and $t_2$ are enabled transitions such that, when $t_1$ is fired $t_2$ becomes disabled and when $t_2$ is fired then $t_1$ becomes disabled. Then $t_1$ and $t_2$ are said to be in conflict.

Example 1: The following facts can be drawn from Figure 1.

1. $P_1$ is an input to transition $t_2$.

2. $P_2$ and $P_3$ are outputs of transition $t_2$.

3. Other transitions can be described similarly.

4. Transition $t_2$ is enabled since it has a token in its input place $P_1$. Transition $t_9$, on the other hand, is not enabled since one of its inputs $P_9$ does not have a token.

5. If $t_2$ fires then $t_2$ removes the enabling token from place $P_1$ and places tokens in $P_2$ and $P_3$, which are the two outputs of $t_2$. See Figure 2.

6. In the Petri net of Figure 2, if we fire $t_1$, $t_3$, $t_4$, $t_5$, and $t_7$, we obtain Figure 3. This shows that some of the transitions which were not enabled are now enabled and some become disabled. Furthermore, if $t_1$, $t_2$, $t_3$, $t_4$, $t_5$, and $t_7$,
are fired in a sequence, we will have two tokens at $P_8$ and $P_6$.

See Figure 3.

Even though Petri nets are simple, they can be considered and utilized in a large number of ways. Petri nets can be considered as formal automata and investigated either as automata or as generators of formal languages. Even though Petri nets have been utilized by Honeywell for investigating the fault tolerant properties of design, for instance, see Jack (1976), but the literature involved is obscure and often is not available.

Petri nets were originated in Germany and there are two distinct emphases in the study of these nets. European research workers have emphasized the study of the fundamental concept in relation with the system, whereas at MIT and, in general, research workers in the United States have emphasized the mathematical aspects of the study of these nets and their relation to automata theory.
Figure 1.
Figure 2.
Figure 3.
A model is the representation of some important facts or features of the objects under study. The advantage of the study of a model is that it reduces the cost and practical inconvenience. Petri nets are a modelling tool and are designed to study the discrete event systems with concurrent or parallel events. Petri nets especially model two aspects of systems events and conditions. A day-to-day example may be the following:

The two conditions that "A TTY is needed" and "A TTY is available" may cause the event "Allocation of TTY" to occur. The occurrence of this event leads that the two conditions are no more true, i.e., "A TTY is not available" and "A TTY is not needed." In Petri net form it is represented by the following Figure 4.

Figure 4.
Let us consider some more specific examples.

Example 2 :- Let us consider the following:

* Jobs appear and are put on an input list. When the processor is free and a job is on the input list, the processor starts to process the job.

* When a job is completed, it is placed on an output list. If there are more jobs on the input list, the processor continues with another job; otherwise, it waits for another job.

Solution :- Figure 5 and Figure 6 describe the example 2 in terms of Petri net. This is the simplest description of modelling a simple computer system by means of a Petri net.

![Diagram](image-url)
New job enters

A job on the input list

Processor is free

Job processing is started

Job is being processed

Job processing is completed

A job is on the output list

Job leaves

Figure 6.
Example 3: The following Petri net describes the model for someone who is looking for a job. This net explains a static similarity among the computer flow charts.

Figure 7.
Formal Definition and Structure of a Petri Net

In this section we shall define Petri nets on a sound basis. As mentioned earlier, Petri nets are composed of two basic components:

1. Set of places which will be denoted by \( P \).
2. Set of transitions which will be denoted by \( T \).

In order to establish relationship between \( P \) and \( T \) we need some rules. For this purpose we need two mappings called input mapping and output mapping which are denoted by \( I \) and \( O \) respectively. Both of these mappings are defined on the set of transitions \( T \) to the collection of subsets of places. Then \( I(T) \) is called the set of input places and \( O(T) \) is called the set of output places. Now we can define a Petri net as a four tuple \( (P, T, I, O) \) where \( P, T, I, O \) have already been explained.

Example 8:- Let \( C = (P, T, I, O) \) where

\[
P = \{ p_1, p_2, p_3, p_4, p_5 \}
\]

\[
T = \{ t_1, t_2, t_3, t_4 \}
\]
The set \( \text{I}(T) \) is defined by the following:

\[
\text{I}(t_1) = \{p_1\}, \quad \text{I}(t_2) = \{p_2, p_3, p_5\}, \\
\text{I}(t_3) = \{p_3\}, \quad \text{I}(t_4) = \{p_4\}
\]

and the set \( \text{O}(T) \) is defined by the following:

\[
\text{O}(t_1) = \{p_2, p_3, p_5\}, \quad \text{O}(t_2) = \{p_5\}, \\
\text{O}(t_3) = \{p_4\}, \quad \text{O}(t_4) = \{p_2, p_3, p_4\}.
\]

Then \( C \) is a Petri net and its graph is shown below.

Figure 8.
It is to be noted that a Petri net is always a directed graph. Sometimes the formal definition of a Petri net given above is called the net structure whereas its graph is called Petri net graph. Since each can be described by the other, we shall call either of them a Petri net.
Marked Petri Nets

Let us assume "token" to be a primitive in the Petri net.

A marking \( m \) is an assignment of tokens to the places in the net.

Tokens are inserted in the places of the Petri net. If \( n \) indicates the number of places in the Petri net then we indicate by the vector \( m = (m_1, m_2, \ldots, m_n) \) the assignments of tokens into places \( P_1, P_2, \ldots, P_n \) respectively. For example \( m_1 \) indicates the number of tokens inside the place \( P_1 \), \( m_2 \) denotes the number of tokens inside the place \( P_2 \), and so on. Formally we define \( m \) by the following:

\[
m : P \rightarrow N = \{0, 1, 2, \ldots, \#\}
\]

where

\[
m(p_i) = m_i \text{ for } p_i \text{ in } P \text{ and } m_i \text{ in } N.
\]

Now, we can define a marked Petri net as a five tuple \((P, T, I, O, m)\) where \( P, T, I, O \) have the same meaning as in Petri nets and \( m \) is defined as above.

In a Petri net graph tokens are indicated by dots within the places. If a Petri net is
then

\[ m = (1, 2, 1, 0) \]

and the Petri net is

\[ C = (P, T, I, 0) \]

and the marked Petri net is

\[ M = (P, T, I, 0, m) \]

where

\[ I(t_1) = \{p_1\}, I(t_2) = \{p_2\}, I(t_3) = \{p_2\}, O(t_1) = \{p_2\}, \]
\[ O(t_2) = \{p_3\}, O(t_3) = \{p_4\}. \]

Having defined Petri nets and marked Petri nets, we may define state space of a Petri net as follows:

The set of all markings \( (N^n) \) with \( n \) places, is called the state space of a Petri net.
The next state function \( d \) defines the changes in the state of a Petri net. Thus,

\[
\begin{align*}
    d(m, t_j) &= \begin{cases} 
        \text{undefined if } t_j \text{ is not enabled for firing,} \\
        m' \text{ if } t_j \text{ is enabled for firing, where } m' \text{ is} \\
        \text{the marking that results from removing tokens} \\
        \text{from the inputs of } t_j \text{ and adding tokens to} \\
        \text{output of } t_j. 
    \end{cases}
\end{align*}
\]

By \( m^0 \) we denote the initial marking of a Petri net. If \( t_j \) is a transition which is enabled for firing, leading to the marking \( m' = d(m^0, t_j) \) and enabling some \( t_k \) for firing, which leads to \( m^2 = d(m^0, t_k) \) and so on, then this is called an execution of a Petri net.

The marking \( m' \) is said to be immediately reachable from \( m \) if it can be reached by firing some transition in the marking \( m \), whereas \( m' \) is said to be reachable if either it is immediately reachable from \( m \) or it is reachable from any marking which is immediately reachable from \( m \).
The reachability set of a marked Petri net is the set of all states reachable from the initial state.

If an execution of a Petri net leads to a marking \( m^t \) for which no transition is capable of firing, then the execution stops.

If a Petri net has no more than one token at any time then this net is called a safe net.

If a Petri net has no more than \( k \) tokens at any time then it is called a \( k \)-bounded net.

If the number of inputs of each fireable transition is equal to the number of outputs of that transition then such a net is called conservative.

The reachability tree is used to determine a finite representation for the reachability set of a Petri net. It consists of a tree whose nodes are markings of the Petri net and whose directed arcs represent the firing of transitions.
Analysis of Petri Nets

The basic technique for analysing a Petri net involves finding a finite representation for the reachability set of a Petri net. It consists of a tree whose nodes represent markings of the Petri net and whose arcs represent the possible changes in the state resulting from the firing of transitions. A path from the initial marking to a node in the tree corresponds to an execution sequence of a Petri net. Since the state space may be infinite, following steps are used to define a finite reachability tree:

1. If a new marking is generated which is equal to an existing marking on the path from the initial marking to the new marking, the new marking becomes a terminal node.

2. If the reachability set of a marked Petri net is infinite then finite representation of this set is obtained by mapping many markings into the same node of the tree. This many to one mapping is obtained by collapsing a set of states into a node by ignoring the number of tokens in a place of the net when this number
becomes large. This is represented by using a special symbol, 

\(w\), for the number of tokens in this place.

3. A marking \(x = (x_1, x_2, \ldots, x_n)\) is said to be greater than \(y = (y_1, y_2, \ldots, y_n)\) if 

\[x_i = y_i \text{ for } i \neq j; 1 \leq i, j \leq n \text{ and } x_j \neq y_j\]

for some j's.

4. If any new marking \(x\) is generated which is greater than a marking \(y\) on the path from the root to the marking \(x\), then those components of marking \(x\) which are strictly greater than the corresponding components of marking \(y\) are replaced by the symbol \(w\).

5. If equal markings are generated and are not on the same path from the initial marking then all but one markings become terminal node.

The technique mentioned above for finding the reachability tree is referred to as a solution technique.
Example 9: Suppose we are given the following Petri net with an initial marking (1, 0, 1, 1).

![Petri Net Diagram](https://example.com/petri_net.png)

**Figure 10.**
From this net we shall obtain the following reachability tree:

Figure 11.
Characteristics of Petri Nets

There are three main characteristics of Petri nets:

1. Reachability,

2. Liveness,

and

3. Safeness.

Out of these three characteristics, we have already discussed about the reachability. Now we shall discuss the liveness and safeness of a Petri net.

A marking $M_0$ is said to be live for a Petri net if it is possible ultimately to fire any transition of the net by progressing through some further firing sequence.

Example: -- The following net is a live Petri net:
A marking $M_0$ is said to be safe if there exists an integer $n$ such that each place of the net has at most $n$ tokens for every marking reachable from $M_0$. 
Example: The following is a safe net:
since it leads to the following net where the definition of safeness is always met.

Note that the above Petri net is not a live Petri net.

It is easy to construct Petri nets which are both live and safe.
Matrix Approach for a Petri Net

Let \( P \) and \( T \) denote the number of places and transitions in a Petri net respectively.

A marking vector \( M_k \) is a \( P \times 1 \) column vector of non-negative integers. The \( k \)th entry of this vector denotes the number of tokens on place \( k \). In particular, \( M_0 \) denotes the initial marking vector.

The \( k \)th firing vector \( V_k \) is a \( T \times 1 \) column vector of o's and l's. In \( V_k \), the \( i \)th entry is 1 if transition \( i \) is to be fired at the \( k \)th firing and is 0 otherwise. Let

\[
Q^- = \begin{bmatrix} A_{ij}^- \end{bmatrix}
\]

be a \( T \times P \) matrix having \( A_{ij}^- = 1 \) if place \( j \) is an input place for transition \( i \) and 0 otherwise.

Similarly, let

\[
Q^+ = \begin{bmatrix} A_{ij}^+ \end{bmatrix}
\]

be a \( P \times T \) matrix having \( A_{ij}^+ = 1 \) if place \( j \) is an output place for transition \( i \) and 0 otherwise.
In these terms, therefore, a transition $i$ is fireable at marking $M_k$ if each entry of $M_k$ is greater than or equal to the corresponding entry of the $i^{th}$ row of $Q^-$. 

Let us define $Q$ by

$$Q = Q^+ - Q^-.$$ 

Clearly, the $i^{th}$ row of $Q$ represents the token change in each of the $p$ places when transition $i$ fires once.

If $M_{k+1}$ denotes the marking resulting from the marking $M_k$ by the $k^{th}$ firing $V_k$, then

$$M_{k+1} = M_k + Q^T V_k \quad (1)$$

for $k = 0, 1, 2, \ldots$ and $Q^T$ denotes the transpose matrix of $Q$.

Clearly, $M_{k+1} \geq 0$. The equation (1) may be called a state equation.

Example: Let us consider the following Petri net:
In this case, the matrices $Q^{-}$ and $Q^{+}$ are given by

$$Q^{-} = \begin{bmatrix}
    t_1 & 1 & 0 & 0 & 0 \\
    t_2 & 0 & 1 & 1 & 0 \\
    t_3 & 0 & 1 & 1 & 1
\end{bmatrix}$$

and

$$Q^{+} = \begin{bmatrix}
    t_1 & 0 & 1 & 1 & 0 \\
    t_2 & 0 & 1 & 1 & 1 \\
    t_3 & 1 & 0 & 0 & 0
\end{bmatrix}$$

Now, if $k = 1$ then

$$M_2 = \begin{bmatrix}
    0 \\
    1 \\
    1 \\
    1
\end{bmatrix}$$

Also, from the right hand side of equation (1), we get
\[ Q = Q^+ - Q^- \]
\[ = \begin{bmatrix}
  -1 & 1 & 1 & 0 \\
  0 & 0 & 0 & 1 \\
  1 & -1 & -1 & -1 \\
\end{bmatrix} \]

and

\[ M_1 + A^{TV}_1 \]
\[ = \begin{bmatrix}
  0 \\
  1 \\
  1 \\
  0 \\
\end{bmatrix} + \begin{bmatrix}
  -1 & 0 & 1 \\
  1 & 0 & -1 \\
  1 & 0 & -1 \\
  0 & 1 & -1 \\
\end{bmatrix} \begin{bmatrix}
  0 \\
  1 \\
  1 \\
\end{bmatrix} \]
Petri Net Language

First we need the following:

Labelled Transitions: We shall introduce labellings for the transitions in a Petri net in order to study formal languages.

The labelling is done by defining a labelling function \( e \) by

\[
e : T \rightarrow E
\]

where

- \( T \) is the set of transitions
- \( E \) is a finite alphabet.

A labelled marked Petri net defines a set of strings over \( E \) where each string corresponds to a possible execution of the net.

The set of all possible strings corresponding to the possible execution of a marked labelled Petri net is called Petri net language.

By imposing restrictions on the labelling function \( e \) we can generate different classes of Petri net languages. We shall describe some of them in the following:
1. **Free languages**: If \( e(t_i) \neq e(t_j) \) for \( t_i \neq t_j \) then the resulting Petri net languages are called free languages.

2. **Many Linked Languages**: If some transitions under \( e \) map to the same symbol then such languages are termed as many linked languages and are usually far away from natural languages.

3. \( \lambda \)-**Free Languages**: If a class of labelling functions allows transitions to be labelled with the null label \( \lambda \), then the resulting Petri net languages are called \( \lambda \)-free languages. A null label is defined as a label which does not show up in the string resulting from an execution of the Petri net.

In the following we shall give an example of a free language, a non-free language and also give an example of a \( \lambda \)-free language. Finally, we shall give an example where we show that if \( t_1 \) is assigned a null label then the language may contain a regular language. Also these examples provide a correction to the examples referred in Chapter 1.

**Example 10**: Let us consider the following Petri net with initial marking \((1, 0, 0, 0)\).
Figure 11.
Suppose the language under consideration is the set of sequences whose net effect is to move the token in place \( p_1 \) to \( p_4 \). Then the Petri net language consists of \( A \cup B \) where

\[
A = \left\{ a^n c b^m d : n \geq 0, m \geq 0 \right\}
\]

and

\[
B = \left\{ a^n b^m d : n \geq 1, m \geq 1 \right\}.
\]

Solution:- Since \( e_i(t_j) = k_i \) for \( i = 1, 2, 3, 4 \), where

\[
k_1 = a, k_2 = c, k_3 = b, \text{ and } k_4 = d;
\]

the solution technique yields that

\[
\text{cd, } a^n cb^md, a^n b^md, a^n cb^md
\]

for \( n \geq 0, m \geq 0 \) belong to the Petri net language. Hence, the Petri net language consists of \( A \cup B \).

Example 11:- Suppose the Petri net is defined as in the example 10 but with the new labellings

\[
e_2(t_1) = a = e_2(t_2)
\]

and

\[
e_2(t_3) = b = e_2(t_4)
\]

then find the Petri net language.
Solution:— Again by the solution technique we find that

Petri net language is given by \( \sum a^n b^m : n, m \geq 1 \).

Example 12:— If in example 10, we let \( e_1(t_1) = \lambda \), then

the language generated by the Petri net consists of

\[ \{ b^n d : n \geq 1 \} \cup \{ c b^n d : n \geq 0 \} . \]

Our next examples are concerned with the classification

of different languages. Hence, we shall deal with them in the

next section.
Petri Net Languages by Means of Final Markings

A Petri net language is the set of all possible sequences resulting from the execution of a labelled Petri net starting from the initial marking or markings and terminating in any element of a set of final markings. Hack (1975) defined the following four types of Petri languages.

**L-type** :- In this case the set of final markings is defined by a finite final marking set $F$.

**G-type** :- If $F$ is a finite set of markings then a final marking is any marking which is greater than or equal to any element of $F$.

**T-type** :- A final marking is any terminal marking, i.e., a marking from which no transition is enabled.

**P-type** :- All reachable markings are final markings.

Example 13 :- Let us consider the Petri net of example 10.

In this case, if $F = \{(0, 0, 1, 0)\}$, then the corresponding
L-language is given by the set
\[ \{ a^n b^m : n \geq 0, m \geq 0 \} \cup \{ a^n b^m : n \geq 1, m \geq 1 \} . \]

G-language is given by the set
\[ \{ a^n c^m b^n : m \geq n \geq 0 \} \cup \{ a^n b^m : n \geq m \geq 1 \} . \]

T-language is given by the set
\[ \{ a^m c^m b^n d^n : m, n \geq 0 \} \cup \{ a^m b^n d^n : m, n \geq 1 \} , \]
and

P-language is given by the set
\[ \{ a^m : m \geq 0 \} \cup \{ a^m c^m b^n : m, n \geq 0 \} \cup \{ a^m c^m b^n d^n : m \geq n \geq 0 \} \cup \{ a^m b^n d^n : m, n \geq 1 \} \cup \{ a^n b^m : n \geq 1, m \geq 1 \} . \]

It is clear from the above relations that the P-language set contains L, G and T-languages. In general, L-type and P-type languages have been investigated in depth. It is well known that L-type languages are closed under union, intersection, concatenation, concurrency, reversal,  \( \lambda \)-free homomorphism; whereas P-type languages are closed under union, intersection, concatenation, and concurrency. Also it is known that all regular languages are Petri net languages. Further, it is also

Reproduced with permission of the copyright owner. Further reproduction prohibited without permission.
known that there exist context free languages that are not
Petri net languages and vice versa. See Crespi-Reghizzi and
Mandrioli (1975).
Generalized Petri Nets

A Petri net with multiple arcs is called a generalized Petri net. It is shown by Hack (1976) that this does not increase the modelling power of a Petri net. Thus, generalized Petri nets are at our convenience. Formally, we define a generalized Petri net as follows:

A generalized Petri net is defined as a five tuple $\text{GN} = (P, T, F, B, M_0)$ where

- $P$ is a finite set of places $\{p_1, p_2, \ldots, p_r\}$,
- $T$ is a finite set of transitions $\{t_1, t_2, \ldots, t_s\}$,
- $F$ is a forward incidence function defined by $F : P \times T \rightarrow N$ and
- $B$ is a backward incidence function defined by $B : P \times T \rightarrow N$

where $N$ is a set of natural numbers, and finally

$M_0$ is an initial marking defined by $M_0 : P \times T \rightarrow N$. 


In order to work with a generalized Petri net we need some more definitions.

**Fireable transitions** :- Given a marking $M$, a transition $t$ is said to be fireable if for every place $p$ we have

$$M(p) \geq F(p, t).$$

Also when $t$ has been defined then the new marking $M'$ of $M$ is calculated by

$$M'(p) = M(p) - F(p, t) + B(p, t).$$

**Firing Sequence** :- This is a sequence $e$ of transitions such that each prefix of $e$ leads to a marking such that the following transition is fireable.

**Labelled Petri Net** :- A labelled Petri net is a four tuple $A = (N, E, \oplus, M_0)$ where

- $N$ is a generalized Petri net defined by $N = (P, T, F, B, M_0)$ and $P, T, F, B, M_0$ have the same meaning as described before,
- $E$ is a finite alphabet,
- $\oplus$ is a labelling function defined by
  $$\oplus : T \rightarrow E,$$
  and
M_\text{f} is a final marking defined by

\[ M_\text{f} : P \rightarrow N. \]

Remark :- If \( \phi \) is a partial function, then a labelled Petri net is said to be a \( \zeta \)-transition labelled Petri net and contains only those transitions for which \( \phi(t) \) is well defined.

A partial function defined on \( D = D_1 \cup D_2 \) is a function such that it is well defined on \( D_1 \) and is undefined on \( D_2 \). A symbol \( \zeta \) is used for empty.

In view of the above remark, we can extend the definition of \( \phi \) to take firing sequences into \( E^* \) by saying that if \( t \in T \), \( u \in T^* \), then \( \phi(ut) = \phi(u)\phi(t) \). In addition let \( \phi(\zeta) = \zeta \), where \( \zeta \) stands for the empty string. Thus,

\[
\phi(ut) = \begin{cases} 
\phi(u)\phi(t) & \text{if and only if } \phi(t) \text{ is defined} \\
\phi(u) & \text{otherwise.}
\end{cases}
\]

Extended L-type Languages :- A language \( L \) is said to be extended L-type language if there exists a labelled Petri net \( A = (N, E, \phi, M_\text{f}) \) where \( N \) is a generalized Petri net.
such that if for every $x \in L$, then there exists a firing sequence $u$ beginning at $M_0$ and leading to $M_f$ for which $\mathcal{R}(u) = x$ and conversely.

$L_0^c$-Languages: The set of $L$-type languages with respect to a generalized Petri net is denoted by $L_0^c$.
Convex Languages

Let $E$ be alphabet and $E^*$ be a free monoid with the null word $. Let $\leq$ be a partial ordering defined on $E^*$ by embedding, i.e., $x \leq y$ if and only if $x = x_1 x_2 \ldots x_n$ and $y = y_1 y_2 x_2 \ldots x_n y_{n+1}$ for some integer $n$ where $x_i, y_j \in E$ for $1 \leq i \leq n+1$.

The following result is due to Haines, whose proof is somewhat expanded here in order to make this theorem simpler to read.

**Theorem A** [Haines]: Each set of pairwise incomparable elements of $E^*$ is finite.

**Proof**: We prove this theorem by using induction. Clearly, the theorem is trivially true if $E$ is a 1-letter alphabet. Suppose the theorem is true for all $n$-letter alphabet and fails for $(n+1)$-letter alphabet $E$.

By assumption for a $(n+1)$-letter alphabet $E$ the theorem is false implies that for every infinite set $Y = \{y_0, y_1, y_2, \ldots \}$ of $E^*$ of incomparable elements, there is $Y = \{y_1, y_2, \ldots \}$ of $E^*$ of incomparable elements such that there is an $x = y_0$ in $E^*$ for which $x \neq y_i$ for all $i$ and is smallest in length.
Therefore, without loss of generality, we may let $Y$ be chosen such that $x$ is of smallest length with $x \preceq y_1$ for all $i$.

Obviously, $x \notin E$. Let $x = x_1 x_2 \ldots x_{k-1} x_k$, $x_j$ in $E$, $1 \leq j \leq k$. If $k=1$, then $y_j$ is in $(E - x_1)^*$ for all $i$ and contradicts the hypothesis. Clearly, by the choice of $x$, we have

$$x_1 x_2 \ldots x_{k-1} \preceq y_1, \quad k > 1 \quad (1)$$

for all but finitely many $i$. Hence for $i \geq 1$ we may assume the truth of (1), just by trivial manipulation. Also, for $i \geq 1$ there exist unique words $y_{i1}, y_{i2}, \ldots, y_{ik}$ such that

$$y_i = y_{i1} x_1 y_{i2} x_2 \ldots y_{ik-1} x_{k-1} y_{ik}$$

and $x_j \notin y_{ij}$ for $1 \leq j \leq k-1$. The latter statement follows because of the previously stated property of $Y$. Also, the choice of $x$ guaranties that $x_k \notin y_{ik}$. Hence $x_j \notin y_{ij}$ for $1 \leq j \leq k$ and for all $i \geq 1$.

Now in order to produce a contradiction, we show that there exist infinite index sets $N_1, N_2, \ldots, N_k$ such that $N_j$
contains $N_{j+1}$ for $1 \leq j \leq k$ and $y_{pj} \leq y_{qj}$ for $p < q$ and $p, q \in N_j$ for $1 \leq j \leq k$.

The idea clearly should be based on showing the existence of $N_j$ by the existence of $N_{j-1}$. Let

$$N_0 = \left\{ i : i \geq 1 \right\}$$

and $Y_j = \left\{ y_{ij} : i \in N_{j-1} \right\}$.

There are two possibilities regarding $Y_j$.

(a) $Y_j$ is finite and (b) $Y_j$ is infinite.

We shall discuss both the cases in the following:

(a) If $Y_j$ is finite then at least one of the sets

$$\left\{ i \in N_{j-1} : y_{ij} = w \right\}$$

is infinite for some fixed word $w$. However, this set by convention is $N_j$. Thus, let $N_j$ be any such infinite set.

(b) If $Y_j$ is infinite then by hypothesis we know that the theorem is true for up to an $n$-letter alphabet. The induction hypothesis therefore applies for $Y_j$ contained in $(E - x_j)^*$. Thus, lemma 3 yields that $Y_j$ possess an infinite
chain \( y_{s_1^j} y_{s_2^j} \cdots y_{s_k^j} \cdots \). Let \( \{t_i^j\} \) be a strictly increasing subsequence of \( S_1, S_2, \ldots \). Now, we may choose

\[
N_j = \{t_i^j : i \geq 1\}.
\]

Thus, the existence of \( N \) is guaranteed in either of the cases (a) or (b).

Having shown the existence, we are set for a final assault.

Now if \( p < q \) and \( p, q \in N_k \) then \( p, q \in N_j \) for \( 1 \leq j \leq k \) and

\[
y_{p_j} < y_{q_j} \text{ for } 1 \leq j \leq k.
\]

Therefore,

\[
y_p = y_{p_1} x_1 y_{p_2} x_2 \cdots y_{p_{k-1}} x_{k-1} y_{p_k} \\
\leq y_{q_1} x_1 y_{q_2} x_2 \cdots y_{q_{k-1}} x_{k-1} y_{q_k} \\
= y_q
\]

which is a contradiction.

Lemma 3: If every set of pairwise incomparable elements of \( E^* \) is finite then every infinite subset of \( E^* \) possesses an infinite chain.

Proof: Let \( A \) be an infinite subset of \( E^* \) and that every chain in \( A \) is identical with the maximum element of \( A \) and is,
therefore, by hypothesis, finite. Since A is infinite, infinitely many distinct chains have the same maximum element u which contradicts the definition of ≤. Hence the lemma is proved.

Now we turn back to convex languages. Convex languages are defined as follows:

A Language L over the alphabet X is said to be

(a) Convex if x ≤ z ≤ y and x, y ∈ L ⇒ z ∈ L.

(b) Right convex if x ≤ y and x ∈ L ⇒ y ∈ L.

(c) Left convex if y ≤ x and x ∈ L ⇒ y ∈ L.

(d) Strongly convex if x ≤ y and y ∈ L ⇒ x ∈ L.

All languages (a) to (d) are convex. Every x ∈ X* is a convex. Convex languages are also regular. L is a left convex if and only if L is a right convex.

The following is a very interesting result:

Lemma 4 :- L is a left convex if and only if L = L and L + is a right convex if and only if L = L where
\[ L^+ = \{ x \in X^* : y \leq x \text{ for some } y \in A \subseteq X^* \} \]

and

\[ L = \{ x \in X^* : x \leq y \text{ for some } y \in A \subseteq X^* \}. \]

Proof:- Let \( L \) be a left convex language. Then, by definition \( x \leq y \) and \( y \subseteq L \implies x \in L \). Thus if \( x \in L \) and

\[ x \in L \text{ then it follows that for some } y \text{ and } x \leq y \neq x \in L \]

which is a contradiction. Thus, if \( x \in L \) then \( x \) must belong to \( L \). Conversely, if \( x \) is in \( L \) then it is immediate that

\[ x \in L. \]

Similarly other parts of the theorem follow. See Haines (1969). Now as a consequence of theorem A it follows that:

Theorem B :- If \( A \) is contained in \( X^* \) then there exist

finite subsets \( F \) and \( G \) in \( X^* \) such that \( A = F \) and \( A = X^* - G. \)

Also \( A \) and \( A \) are regular sets.

Since \( A \) and \( A \) are regular sets, there exists a Petri net

which as \( A \) and \( A \) as a subset of its languages. At this stage

a most fundamental question arises of characterizing those

Petri nets which accept \( A \) and \( A \) only. In general, the construction

of a Petri net which accepts all convex languages is also open.
BIBLIOGRAPHY


