To my Supervisors

and

My Wife

ACKNOWLEDGEMENTS

I am greatly indebted to my supervisors Prof. J.W. Atwood and Prof. T. Radhakrishnan for their guidance, encouragement, many fruitful suggestions and comments during the course of this research. It is Professor Atwood's untiring helping tendency and kindness both financially and otherwise which enabled me to join the university. It is the unforgetful and invaluably friendly advice and help of Prof. Radhakrishnan which enabled me to sustain many hardships in life. The best way to express my thanks, I felt, is to dedicate this thesis to my supervisors.

I wish to express my sincere gratitude to many of my friends in Canada and India, in particular, Mr. Pervez Ahmad, Mr. Ramon, Mr. Murali, Mr. Roy, Mr. Selvaraj, Mr. V. C. Bhavsar, Mr. R. D. Kuamr and Mr. P. Chinnuswamy for their assistance and companionship.

The grants from the Godrej Foundation and the Wadia Trust, and the educational loan from the Sethna Trust were very helpful towards my travel and initial expenditures. Their timely help is sincerely acknowledged.

I also wish to thank Prof. A. K. De (Director), Prof. J. R. Isaac, Prof. H. B. Kekre and other colleagues of the Indian Institute of Technology, Bombay, India for granting me study leave to pursue graduate studies at Concordia University.

I am grateful to Concordia University for awarding me the 'Concordia University Graduate Fellowship' during 1979-'81. I also wish to thank the members of the faculty of the Department of Computer Science and the Dean of Engineering, Concordia University for admitting me as a member of part-time faculty which helped me to support my family while studying. Finally, the secretaries of the department need to be thanked for their help.

This study was partially supported by the Quebec Ministry of Education's Formation des Chercheurs et Action Concertee (FCAC) program, and by a Concordia Fellowship.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1. INTRODUCTION

1.1 Numerals, Scripts and Languages

Computers are used in linguistically different parts of the world, in areas as different as medicine, agriculture, word processing, etc. The data to be processed for these applications are generated and maintained in different natural languages depending on the region. To get a linguistic picture of the world, let us examine the major world languages and numerals that are used by more than 90% of the world population. Based on their script, the languages could be classified into the following categories:

1. Latin script languages
2. Cryllic script languages
3. Indo-Asian languages
4. Arabic languages
5. Ideographic languages
6. Pictorial languages
7. Scripts for blind persons.

A brief description of each of these categories is given in the following paragraphs.

Latin Script Languages: Most of the European languages use the Latin script. The well-known languages such as English,

French and German fall into this category. The characters of these languages include A to Z (also a to z) and a few additional characters. Some additional characters of the languages of this category are listed in figure 1.1 [1]. A careful consideration reveals that they consist of the symbols a to z with diacritical marks such as ¨, ·, ^, ˇ, ~, ´. The writing is from left to right and top to bottom. English is the mother tongue of more than 300 million people. Number of people who speak English with atleast some degree of proficiency totals many million more.

Cryllic Script Languages: More than fifty different languages use the Cryllic script or an extended version, as with the Latin script. The Russian language comes under this category. Most nations of the Soviet Union and some other nations of foreign countries use scripts based on the Russian alphabet. Certain languages use, apart from the Russian characters, some additional characters. The set of additional characters of a few Cryllic script languages is listed in figure 1.2 [1]. The writing is from left to right and top to bottom. The Cryllic script is used by more than 250 million people.

It is interesting to note that the languages using Latin and Cryllic scripts have a small number of characters of the order of a few tens. These characters have simple graphemes of almost uniform size.

Albanian	ç, ë
Aymara	ñ
Basque	ē, ñ, ŕ
Breton	é, è, ė, ī, ï, ñ, ò, ū
Bui	Б, ʁ, ɘ, ɒ, ш, ᵊг, ч, ø, ь, з, ʒ
Catalan	à, ç, é, è, ï, ó, ò, ü, ú
Choctaw	ą, į, ǫ, ų
Chuana	è, ė, ō, ś
Cree	ā, ī
Czech	á, č, d' (ď, ǎ, Ď), é, ě, í, ň, ó, ř, š, t' (ť, Ť), ú, ů, ý, ž
Danish	å, æ, ø
Delaware	á, é, í, ī, ó, ō
Eskimo	â, ê, î, ō, r'
Esperanto	ĉ; ĝ, ĥ; ĵ, ŝ, ŭ
Estonian	ä, õ, ō, ü
Ewe	d(Ɖ), ɛ, f(Ƒ), ɣ, ɑ, ɔ, ʊ; ã, ẽ, ĩ, ɔ̃, õ, (d, ġ, h, ṅ, ɜ)
Faroese	æ, ø, ꝺ
Fiji	ā, ē, ī, o, u
Finnish	ä, ö
French	à, â, ç, ē, é, è, ê, ī, ï, œ, ū, ù, û
Frisian	â, è, í, ô, ú, û
Fulbe	ā, ḅ, ḍ, ē, ɛ, ġ, ń, ō, ɔ, ẇ
German	ä, ö, ß, ü
Guarani	á, à, e, é, è, í, ì, ñ, ó, ò, ú, ũ, ý, ỹ
Hausa	ɓ(Ɓ), ɗ(Ɗ), ƙ(Ƙ)
Hungarian	á, é, í, ő, ó, ö, ű, ú, ü
Icelandic	á, æ, ð (Ð), é, í, ō, ó, þ, ú, ý
Irish	á, é, í, ó, ú
Italian	à, é, è, í, ì, ï, ó, ò, ú, ù
Javanese	ḍ, é, è, ṭ
Juang	Б, ɘ, ɲ, ɵ, ъ, з, ь, ƶ, ƨ, ч, ƥ, ш
Kasubian	ą, ć, ė, ē, ę, ł, ń, ó, ȯ, ö, ś, ų, ż, ź
Kurdish	ê, ō (in Iraq);
Lahu	ç, ê, l, ş, û, x̌ (in Syria)
Latin	ā, ă, æ, ē, ĕ, ī, ĭ, ō, ŏ, œ, ū, ŭ
Lettish	ā, č, ē, ģ, (ǧ), ī, ķ, ļ, ņ, š, ū, ž
Lingala	ɛ, ɛ́, ɪ, ɔ, ɔ, ɔ́, ʊ
Lithuanian	ą, č, ę, ė, į, š, ų, ū, ž
Lisu	Б, Д, ɜ, Г, ŋ, з, ч, ш, ʍ, ʒ, л, я, ф, ь, ц
Luba	ñ, (š), ž
Madura	ḍ, è, ṭ
Miao	ɑ, ɜ, æ, ɥ, ɯ, ŋ, ɵ, ʔ, ɾ, ʒ, и, Б, ƨ, з, ж
Malagash	á, à, é, è, í, ì, ó, ò, ý, ỳ
Malay	ĕ
Mandingo	ã, ã, ç, é, è, ĕ, ę, g, ī, į, ṅ, ȯ, ȯ̃, õ, ȯ, ó, ȯ́,

Fig. 1.1 Additional Characters of the Latin Script

Minankabaw	ḟ, ū, ṻ, ṯ, u̲, ē, ī
Mohawk	ȧ, ę̇
Mossi	ã, ã, ã̄, ẽ, ẽ̄, ẽ, (ɛ̃), ɛ̃, g', ĩ, ĩ̄, ń, õ, (o̲), ȭ, p', t'
Navaho	ǧ, k', ł, ṅ, ṅ, t'
Norwegian	å, æ, ø
Occidental	l', n', á, é, í, ó, ú (or à, è, ì, ò, ù)
Ojibwe	a;e; ?, i; o; š
Polish	ą, ć, ę, ł, ń, ó, ś, ź, ż
Portuguese	á, à; ã, â, ç, é, ê, è, ẽ, ĩ, í, ó, ò, ô, õ, ũ, ú
Quechua	á, ñ, ó, ú
Rhaeto-Romanic	à, á, è, é, ò, ö, ü
Rumanian	ă, î, ş, ţ
Samoan	ā, ō, ū
Seneca	ę, œ, ų̃
Serbo-Croatian	č, ć, đ (Đ), š, ž
Sioux	ą, c', ć, ç, ǧ, ȟ, į, k, k', ŋ, p., p', ś, š, ţ, ų, ź, ż
Slovak	ä, á, č, d' (ď, ḋ, Ď), ė, í, l' (ĺ, L'), ň, ó, ô, ŕ, š, t' (ť, Ť), ú, ý, ž
Slovene	č, š, ž
Spanish	á, é, í, ñ, ó, ú, ü
Suto	è, ò, š
Sundanese	ě
Swahili	ng'
Swedish	ä, å, ö
Tagalog	(ñg)
Turkish	â, ç, ğ, ı, I', İ, ö, ş, ü, û.
Ublio	b, d, ñ
Vietnamese	ă, ă, đ (Đ), ė, ó, o', n'; vowels in combination with diacritical signs to denote the tones:
Volapük	ä, ē, ö, ü
Welsh	á, â, ê, é, ó, ô, ŷ, ŷ, ŵ
Wendish	ć, ė, ě, ł, ń, ó, ŕ, ż, ž
Wolof	ɔ, ñ, ŋ, ŕ, ŭ
Y	ą, (ə), ə, ų, ŋ, ɵ, ж, ʒ, ɔ
Yoruba	ọ, ḥ, ṅ, ḳ, ũ
Zulu	ɓ(ƃ),

Fig. 1.1 (contd.)

Abazin I	— I
Abkhasian	— ԥ, ҧ, ҿ, ҽ, ҙ (ӡ), ҟ, ҡ (қ), ө, ҩ, ҭ, х, ҵ, ҷ, ɔ
Adighe I	— I
Altaic	— j, ң, ӧ, ӱ
Avar I	— I
Azerbaijani	— г, ɔ, j, ҝ, ə, ɣ, ӌ, ч
Bashkir	— г, ҙ, ҡ, ң, ө, ҫ, ү, һ, ә
Buryat	— ө, ү, һ
Byelorussian	— і, ў
Chechen	— I
Chukcha	— ӷ, ӈ
Chuvash	— ӑ, ӗ, ҫ, ӳ
Dargwa	— I
Dungan	— ɔ, ӄ, ң, ӱ, ү
Eskimo	— ӱ
Even (Lamut)	— ӈ, ө, ӧ
Evenki (Tungus)	— ӈ
Gagauzi	— ӓ, ӧ, ӱ
Ingush	— I
Kabardian (Circassian)	— I
Kalmyck	— ɔ, һ, ӄ, ң, ə, ү, (ӑ, ӧ, ӱ)
Karachai	— ӱ
Kara-Kalpak	— қ, г, х, ӈ, ө, ү, ӳ, ә
Kazakh	— ɔ, г, қ, ң, ө, ұ, ү, һ, і
Khakass	— г, і, ӧ, ӱ, ҷ, ӌ
Khanty (the language of the Vakh Khanty)	— ӑ, ҕ, ӈ, ӧ, ө, ө, ӱ, ɔ, ӫ
Khanty (the language of the Kazym Khanty)	— ӑ, ə, ӫ, ӈ, ө, ӓ
Khanty (the language of the Surgut Khanty)	— ӑ, ҕ, ӈ, ө, ӧ, ӱ, ə
Khanty (the language of the Shuryshkar Khanty)	ӈ
Kirghiz	— ң, ө, ү
Komi-Permian	— і, ӧ
Komi (Zyryan)	— і, ӧ
Kurdish	— ɔ, ӧ, һ; ҁ(Q), ш
Lak	— I
Lesghin	— I
Macedonian	— ѓ, ќ, љ, њ, ѕ
Mansi (Vogul)	— ӈ
Mari (High)	— ӑ, ӧ, ӱ, ҷ
Mari (Low)	— ӈ, ӧ, ӱ
Mongolian	— ө, ү
Osselic	— ӕ(æ)
Serbo-Croatian	— ђ(Ђ), ј, љ, њ, ћ(Ћ), џ
Tajik	— г, ӣ, қ, ӯ, х, ҷ
Tatar	— ɔ, ң, ү, җ, ҫ, һ
Touvinian	— ң, ө, ү
Turkmen	— ӂ, ң, ө, ү, ɔ
Udmurt	— ӝ, ӟ, ӵ, ӧ, ӥ
Uigur	— қ, ң, г, ү, ө, ӂ, ɔ, һ
Ukrainian	— є, і, ї
Uzbek	— ў, қ, г, х
Yakut	— ҕ, ҥ, ө, һ, ү

Fig. 1.2 Additipnal Characters of the Cryllic Script

Indo-Asian Languages: Many different Indian languages and Asian languages such as Sinhalese, Burmese, Thai and so on use a variety of scripts. Devanagari, Dravidian and Pali are the most commonly used. The number of characters in each of these languages vary from a few hundreds to a few thousands. In fact, some of these languages, such as the Indian language Telugu, can have a theoretically infinite number of characters. In practice the number is limited to about two thousand. The characters of these languages are more complicated than the characters of the Latin and Cryllic scripts. They also vary in size. The writing is from left to right and top to bottom. These languages are spoken by more than 500 million people. Figure 1.3 gives the basic characters of a few Indian languages. More information about Indian languages is given in later sections.

Arabic Languages: The Arabic, Urdu, and Farsi languages belong to this group. These languages are wide-spread in the United Arab Republic, Iran, Kuwait, India and other Arabic countries. The complexity of the characters (refer to figure 1.4) of these languages falls between the Indo-Asian languages and the Ideographic languages. The writing is from right to left and top to bottom. These scripts are used by more than 300 million people.

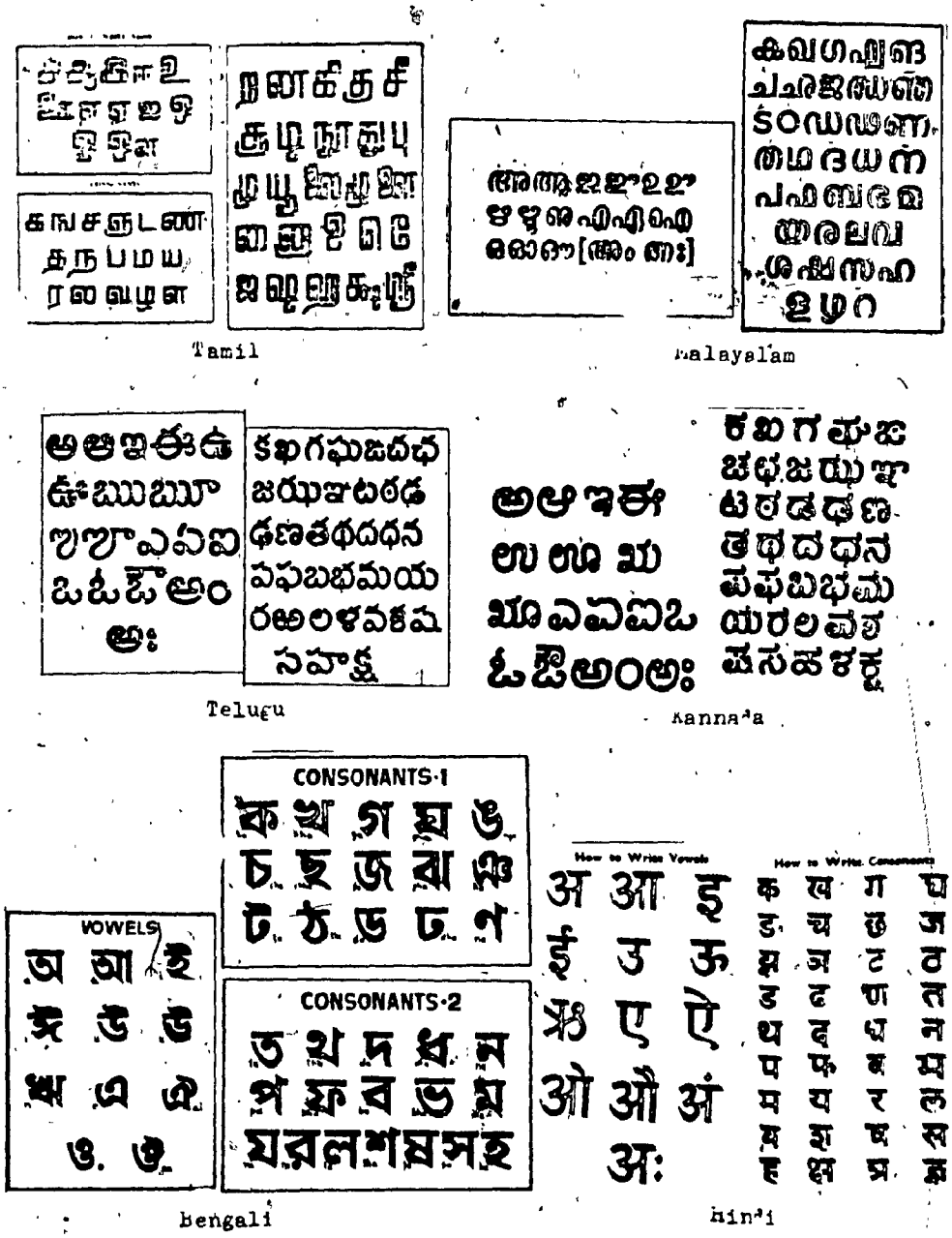


Fig. 1.3 Basic Characters of a Few Indian Languages

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ ﴿١﴾

ٱلْحَمْدُ لِلَّهِ رَبِّ ٱلْعَٰلَمِينَ ﴿٢﴾ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ ﴿٣﴾

مَٰلِكِ يَوْمِ ٱلدِّينِ ﴿٤﴾ إِيَّاكَ نَعْبُدُ وَإِيَّاكَ

نَسْتَعِينُ ﴿٥﴾ ٱهْدِنَا ٱلصِّرَٰطَ ٱلْمُسْتَقِيمَ ﴿٦﴾

صِرَٰطَ ٱلَّذِينَ أَنْعَمْتَ عَلَيْهِمْ غَيْرِ ٱلْمَغْضُوبِ

عَلَيْهِمْ وَلَا ٱلضَّآلِّينَ ﴿٧﴾

Fig. 1.4 Sample Arabic Text

Ideographic Languages: Chinese, Japanese, and to a certain extent Korean, are examples of ideographic languages. The Chinese language has an amazing number of characters, about 48,000; around 5000 characters are used in writing and about 3000 in printing [2]. The presence of Katakana and Hiragana syllabic alphabets distinguishes Japanese from Chinese. Japanese characters are somewhat simpler than Chinese characters, still the graphemes of these languages are extremely complicated. To illustrate, a few Chinese characters are shown in figure 1.5 [1]. The Chinese characters cannot be accommodated on a keyboard of normal size. The direction of writing is from top downward, from right to left. Since 1956 the direction of writing in most publications has been from left to right (as in Indo-European languages). Repeated attempts have been made to simplify the script of Chinese (hierographs). Chinese, Japanese and Korean are used by approximately 600, 100 and 30 million people respectively.

Pictorial Languages: Languages and scripts like Nasi, Maya, old Babylonian, the hieroglyphic form of the Egyptian script, the Creto-Mycenaean script, and the Hittite Hieroglyphic script belong to the pictorial class of scripts. They use pictures (refer to figure 1.6) to convey information rather than characters as in other languages. These scripts are extremely complicated both to write and for computerization. Also they are used by a very small

CHINESE

他是三十歲左右的青年人，有着闊大人一付英明而樸質的面孔，
下頦稍長，鼻子像是並架起的梁，兩眼的眉毛豎了起來，顯
示出少年軍人特有的英俊；眼角下的眉毛下沉的，長長的，掩飾着他
那雙放銀白色的眼光，使他的面部充滿了一種英武的氣概；但並不能
造成使別人不願親和他親近的印象。他對於朋友是誠懇的，對於工作
是積極的，對於人民是熱愛的。無論是當他在大學校裏求學的時候，
在地下工作的時候，以及帶領部隊打仗的時代，他都擔當着重要付出
重大的代價，甚至不顧惜自己的犧牲。幾年來，他運用一種靈活部隊
的巧妙的戰術，戰鬥的經驗這樣的豐富了他，而且成為自然了。這次
他從平內軍區領了他們的任務，率領着兩千名的戰士，越過了重山山
脈所附的半閒。在這地中行走，過在荒僻的山谷裏宿營，在石洞裏飲
居。我們完成上級的任務，他終於帶着部隊向着察哈爾的邊界轉
來了。

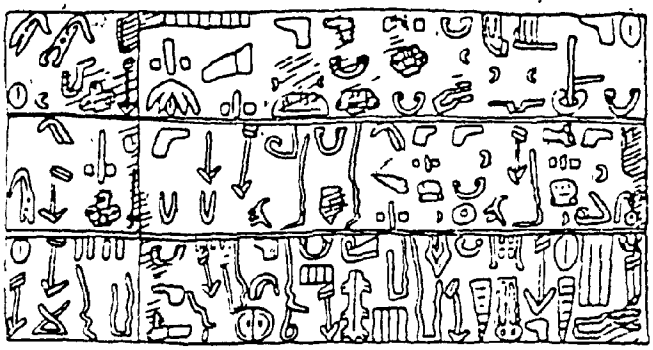
Chinese, Korean, Japanese

Hieroglyphic Script

他是三十歲左右的青年人，有
下頦稍長，鼻子像是並架起的梁
示出少年軍人特有的英俊；眼角下
那雙放銀白色的眼光，使他的面部
造成使別人不願親和他親近的印象
是積極的，對於人民是熱愛的。無
在地下工作的時候，以及帶領軍隊
重大的代價，甚至不顧惜自己的犧
的巧妙的戰術，戰鬥的經驗混熟的

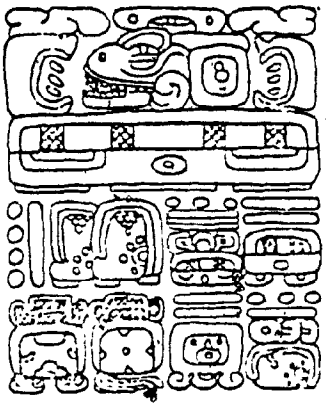
Fig. 1.5 Sample Chinese Characters

Hittite Hieroglyphic Script



Hittite

Maya Script



Creto-Mycenaean Script

Pictorial Script

Egyptian Script

Hieroglyphic

Nasi

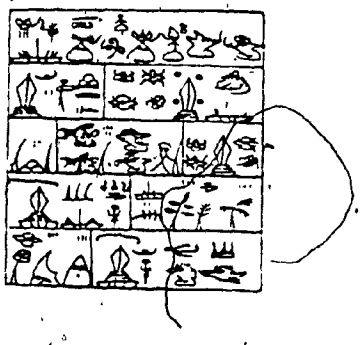


Fig. 1.6 Sample Text of a Few Pictorial Languages

fraction of the population of the world, and many of them are becoming extinct.

Scripts for the Blind: Keyboards suitable for blind people to communicate with computers are available. The Braille script is used for outputting text. Braille is a system of raised characters formed by using combinations of six dots in three rows and two columns as shown in figure 1.7. The dots represent the projections on the output paper.

Numerals: In most parts of the world, the graphics used for the numerals are 0 to 9. From figure 1.8 [1] it is interesting to note that there are languages using different graphics to represent the decimal digits. Some of these languages, such as Tamil, Telugu, Malayalam, etc., do not use these graphics any more in practice. However, languages like Hindi, Marathi and so on, predominantly use the graphics shown in figure 1.8, and not 0 to 9.

1.2 The Need for A Multiscript System

Text processing and text editing with computers have created a need for input, output and internal representation of the characters of natural languages. The output from the computing systems should be in a form comprehensible to users with minimal computer knowledge and training. The

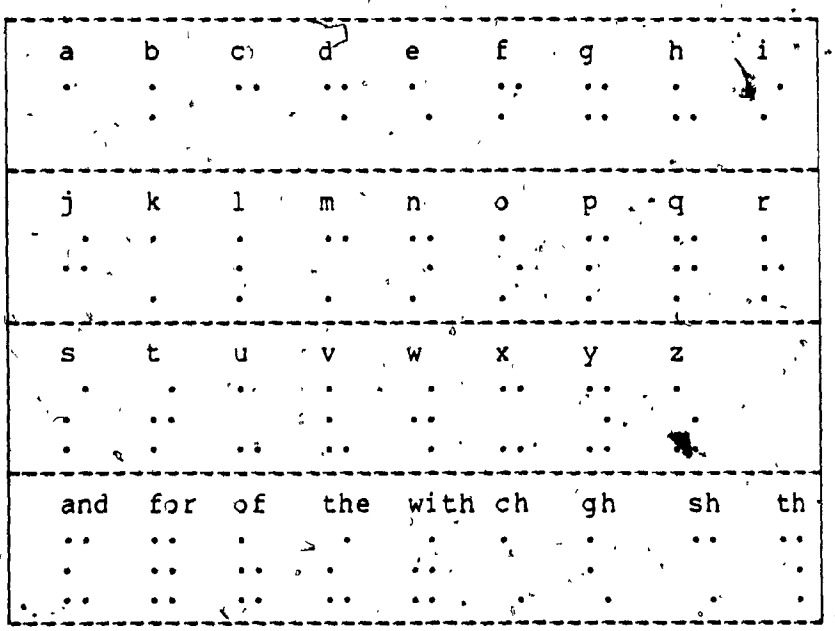


Fig. 1.7 Code for the Braille System

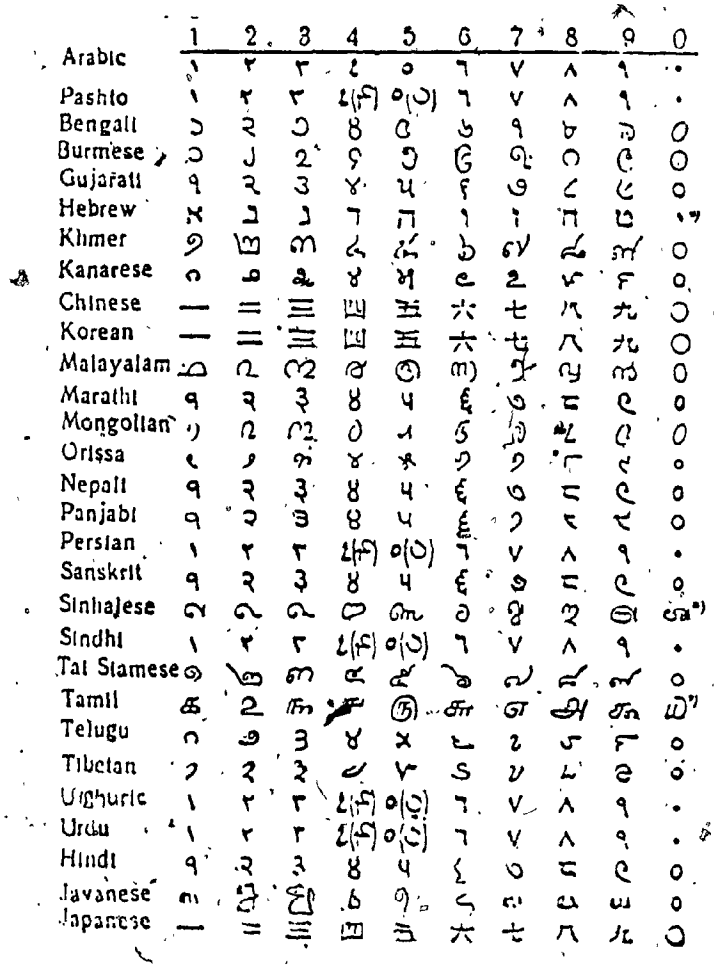


Fig. 1.8 Numerals in Different Languages

symposium on 'Linguistic Implications of Computer Based Information Systems' held in India has considered some of these problems. For the past decade or so the Chinese and Japanese have been actively involved in computerized information handling in their languages [3, 4]. Much of the work for Arabic and Farsi languages has been carried out by Hyder and Parhami et al. [5, 6, 7]. Computers capable of handling data presented in the Russian language already exist [8]. The emergence of word processors in French, German and English is an example of this awareness among other nations.

Unlike other nations, India faces a unique situation. There are about 800 languages and dialects prevalant in India [9], of which fifteen regional languages are well-developed and recognized by the national constitution. There are about ten different scripts, and each script has hundreds of characters and complex orthographics. Of the six hundred million people only 40 million know English. In the public and private sectors much of the information is generated and maintained in Indian languages.

Two computer-related operations require such dataprocessing if their usage is to be beneficial. The sector of the public involved with newspaper publishing needs to avoid the tedious job of translating information supplied by central bureaus in English. Performing the

translation into regional languages once at the source itself would no doubt be advantageous, but transmission would require the telex system to handle information in all those languages. The present-day telex systems in India, except for a few major post offices having separate telex systems in Hindi, operates only in English. To facilitate multilingual information handling, a multilingual telex system would be imperative. The feasibility of designing such a system is examined in a later chapter in this thesis. In another area of computer application, photocomposing, the present computer facilities accept only English text; the advantages of this process cannot be enjoyed by typesetters for other languages even if they possess a computer. Here again we need a system to handle multiple scripts. The list of such problems could be easily extended.

With the introduction of microprocessors and the development of microcomputers at ever-declining cost, the day is not far off when, even in underdeveloped countries, primary schools and many small businessmen may possess computers. Microcomputers may be in as wide use as the transistor radios of today. In such circumstances it will even be necessary to develop programming languages that will have the flavour of the users' native languages [10].

As with India, there are countries in the world where more than one language (with possibly different scripts) is

in use. Textual information exists in those different scripts. Sometimes the same information is written and rewritten in different languages and/or different scripts. Attempts have been made to process texts in a single script at a time. We are interested in a unified facility to handle texts written in multiple scripts and multiple languages.

1.3 Present Computing Systems

Right from the inception of computing systems, there were continuous efforts to generalize one facet or the other of a computing system. There are four important facets for any computing system.

1. Data representation.
2. Computational languages used for communication between users and computing system.
3. Algorithms.
4. Natural language aspects of data.

Computing systems emerged initially with the capabilitiy of handling 'numerical' data and were later generalized to handle non-numeric data as well.

Languages for using computers grew from machine language to assembly language to higher level languages, and are now heading towards the natural-language-like query

languages. The large volume of data to be handled in enterprises and the government, and the non-availability of appropriate information at the right time for decision making, has resulted in the development of data base systems.

1.4 Universal System

An ideal computing system suitable for the user communities of the world could be depicted as in figure 1.9. In this system, the user may write programs in any natural language of his/her choice. The programming language could be any of the existing languages such as PASCAL, PL/I, COBOL, SNOBOL, FORTRAN, etc. The program may contain non-numeric literals (character strings) of more than one language. Finally, the program may handle data (both input and output) in one or more languages. The programs so written could be fed either from terminals or through batch. In short, this is a multilingual computing system offering no linguistic barrier to its users.

In order to realise the goals of a universal system, the following would be required:

1. Multilingual input devices.
2. Multilingual output devices.
3. Identification of suitable internal coding schemes.

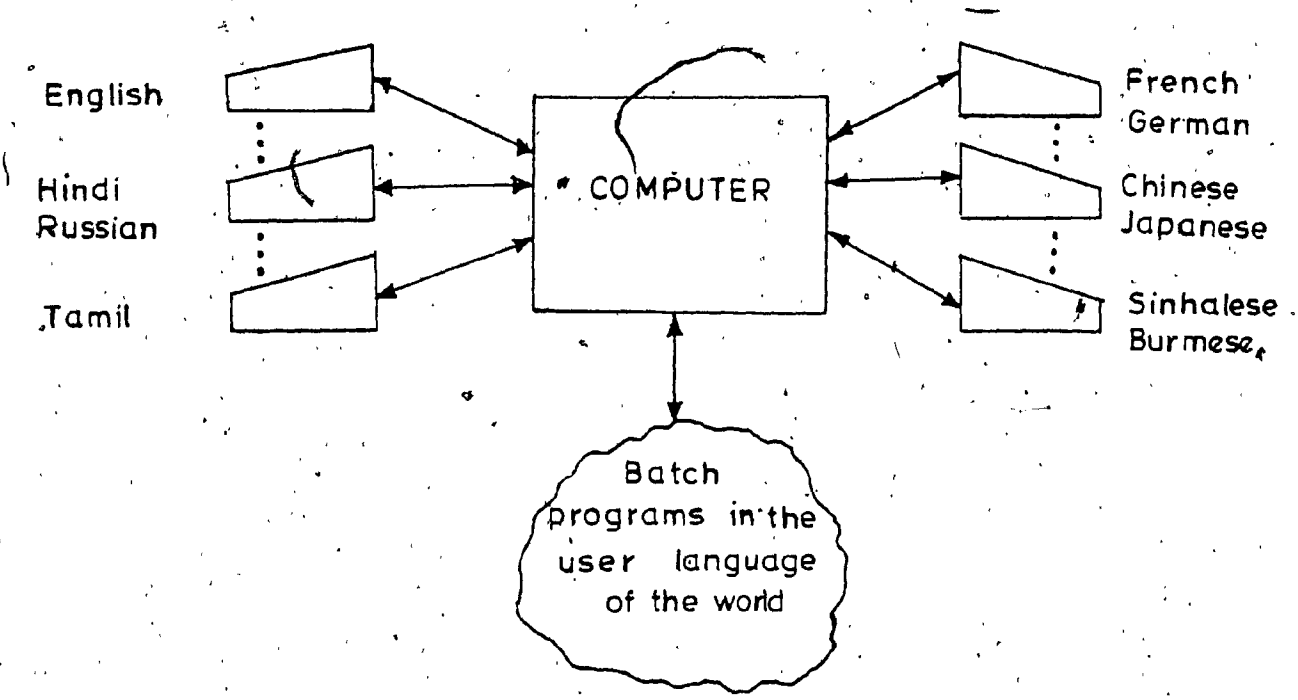


Fig. 1.9 Universal Computing System

4. Programming languages with the flavour of the natural languages.

It is a well known fact that developing language processors and other system software is very expensive. The cost will be exorbitant if the same must be developed for each and every language of the world. No doubt, this system is very expensive to realize. Instead, modularities could be introduced into language processors in such a way that only the lexical analyser is language dependent. Diagramatically it could be represented as in figure 1.10.

Another alternative to reduce cost and yet achieve the goal is to make use of the existing software in English and only to develop language dependent pre- and post- processors for the programs. The pre-processor will accept programs written in a particular language and convert them into equivalent programs in English and then feed them to the existing language processors. The compiled-and-run output will form the input to the post-processor which will reproduce the program and the result in the original language. This will reduce the cost considerably.

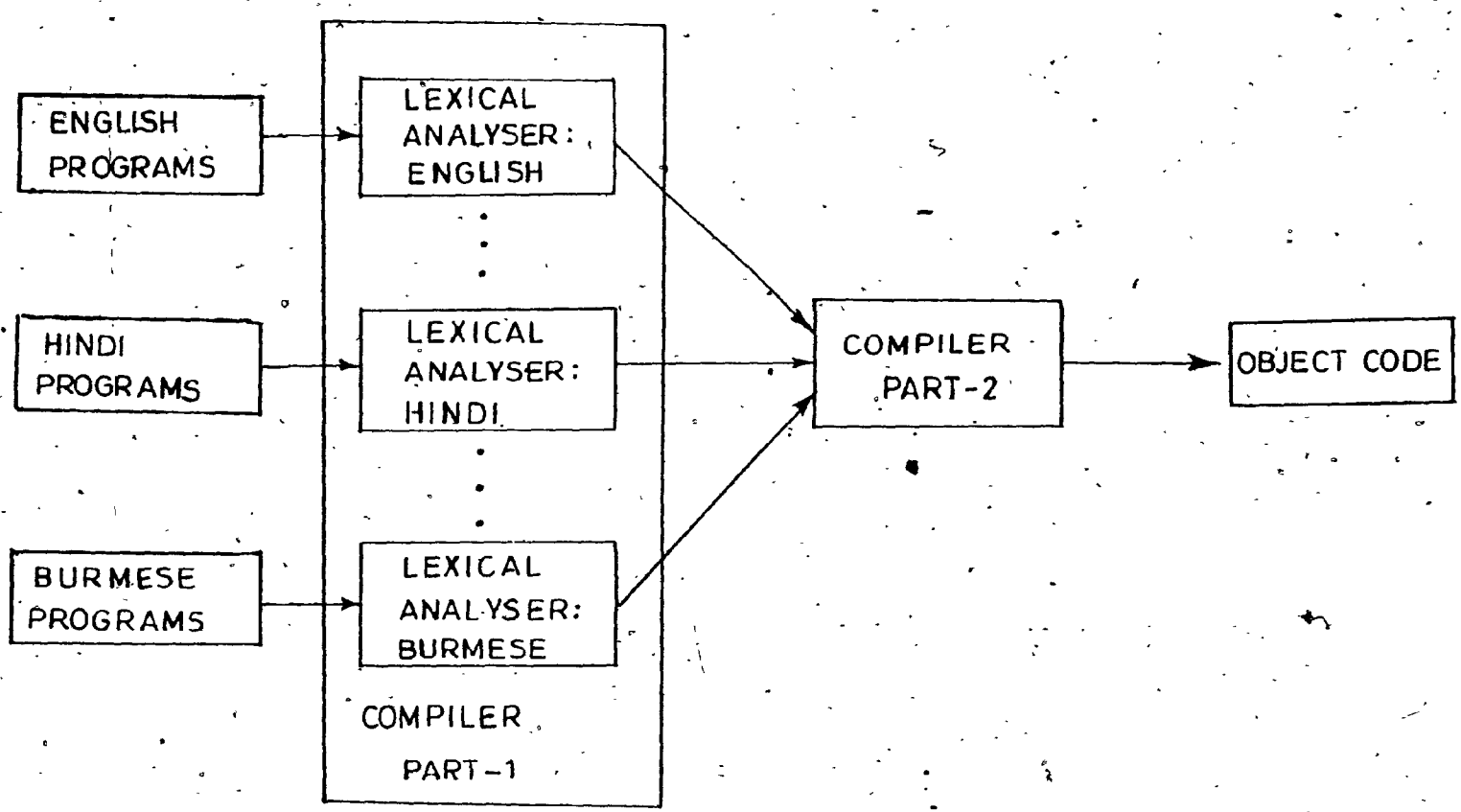


Fig. 1.10 Modular Language Processor

1.5 Scope of the Thesis

The objective of this thesis is to study the feasibility and limitations of designing a facility to input, output and process texts written in more than one script. We call such a facility a 'textwriter'. The studies are restricted to two sets of scripts as they are used in some of the languages of India, to show the viability of the textwriter proposed in the thesis.

The design of a textwriter requires the following:

1. Character set analysis
2. Character decomposition
3. Character generation
4. Keyboard design for text input
5. Output device design
6. Multilingual text representation

In the past attempts have been made to handle scripts in isolation, that is, only one language such as Hindi, Telugu or any other [11, 12]. This thesis examines a multiscript system.

Regarding the work in character set analysis, the characters of the selected languages are analysed and the feasibility of accommodating them in a keyboard of reasonable size is studied.

Indian languages have hundreds of characters. Therefore, to accommodate them on a reasonable sized keyboard decomposition is necessary. The decomposition should be done in such a way as to simplify character generation, representation and manipulation. Methods are suggested to decompose the characters of the selected languages and to design a multiscript keyboard.

The dot-matrix method is quite popular for generation of English characters, numerals and special characters. Besides the dot-matrix method, linguistic methods based on a set of primitives and syntactic rules have been reported for English alphabets [13]. The suitability of these methods for generation of the characters of Indian languages is studied.

The task of character generation involves many factors such as multiplicity of scripts, large number of characters, complex graphics and varying sizes of characters. It is felt that implementation through high level language will be slow, and hardwiring will be complex and inflexible to script changes. Therefore, a microprocessor/microprogrammed implementation is suggested. Also, some selected applications of the textwriter are considered.

1.6 Organization of the Thesis

The work done in this and related areas is surveyed in Chapter II with an emphasis on Indian languages. In Chapter III, the design of symbolic and BLS (basic-letter-sound) based keyboards are explained for the input/output of text written in multiple scripts. Then the two different keyboards are compared. In Chapter IV, two hierarchically structured approaches for multiscript character generation are proposed. The suitability of linguistic, line segment and dot-matrix techniques for character generation are examined. Manual and automatic means for the determination of optimal dot-matrix size are studied. Methods for organizing and optimizing the Read Only Memory (ROM) storage are proposed. The use of microprocessors and microprogramming for textwriter design is mentioned. Selected applications of the textwriter are studied in Chapter V. Chapter VI provides a conclusion and a list of suggested problems for future research in this area.

2. STATE OF THE ART

In this chapter, we present a brief summary of the work reported to date in computerized information handling in Indian languages. The summary is organized into five sections: (1) Indian scripts and languages, (2) input devices, (3) output devices, (4) coding, processing and programmer's interface in Indian languages and (5) selected application areas such as text composition, language processing, language instruction and word processing. An extensive survey of the work done in computerized information handling in Indian languages can be found in [14]. Due to space limitations, the complete list of references is not given in the survey. A full list of more than 150 references with keyword out of context (KWOC) index is available in [15].

2.1 Indian Scripts and Languages

In India there are many languages with different scripts and each language has many characters with varying size and complex graphemes. Notwithstanding the problems [16] of many scripts, a number of new scripts were developed and proposed as unified scripts for all Indian languages.

The Abasama script [17], Sulipi [18] and Nandaka Alphabets [19] are a few examples. There seems to be no rationale behind creating new scripts for Indian languages. The absence of a formalism for script design, that is, the absence of a scientific tool either to analyse and to evaluate an existing script or to design a new script from a given set of constraints is pointed out in [20].

Since a unified language or script is difficult to achieve, many have advocated Devanagari or enhanced Roman [21] as common scripts. For many reasons [16], adapting either of these as a common script has not been acceptable to the Indian people at large. Instead, script refinements and standardization [11, 20, 21] are taking place in almost all languages. The main reform suggested is linearization. That is, instead of writing the symbols of composite characters vertically one below the other as in (STRĒ), linearization suggests writing the symbols horizontally (side by side).

2.2 Input

Much of the research work done is in the development of input/output devices. Mechanical typewriters which are based on the graphemes of the scripts are commercially available for all major Indian languages. Many questions may be raised about the suitability of and optimal

distribution of symbols on the mechanical keyboards [11]. Also we find that the dead keys in typewriters are not suitable for computer applications. Attaching diacritical marks is difficult for the keyboard operator. The design of a keyboard without dead keys, and with a new symbol set for Devanagari, is suggested by Mudur and Wakankar [11]. New letter frequencies, based on modern prose, were obtained for many Indian languages such as Tamil [22, 23], Telugu [24], Hindi [25], Kannada [26]. New sets of symbols for Hindi [11], and Telugu [27] have been identified which are suitable for typewriters, computerized printing and for machine representation. Based on the letter frequencies, keyboards with new symbol distributions were proposed for Hindi [11], Kannada [26] and Tamil [23]. Much of this work was done by statisticians as early as 1960 for the purpose of telecommunication and has hardly affected the design of mechanical typewriters.

2.3 Output

The varying size of Indian characters has dictated that designers opt for dotmatrix type printers and display devices. Various opinions were expressed about the matrix size needed to display a character of a language and different ad hoc sizes of dot matrices have been tried: 17 x 7 [28], 24 x 48 [29], 11 x 9 [30], 7 x 7 [10]. Where

the matrix size is small, for example 7 x 7, a three adjoining tier system (that is, three rows to generate one line of text) has been proposed [10]. A working model, if not an acceptable model, was developed by Koteswara Rao, et al [10] by changing parts of the hardware of a commercially available CRT display in order to display Hindi and Telegu characters. The characters are distinguishable but are not good aesthetically.

Since no electronic keyboard specially meant for Indian languages is available, the existing Roman keyboards have been used by relabeling the key tops with the primitive symbols of the required Indian languages. Under program control the input string was analysed for character generation. This technique was used for keying-in Malayalam [31, 32] and Hindi [11, 31, 32]. Line segment displays were designed to display Devanagari [33] and Farsi [6] numerals. It should be noted that the graphics of numerals in Indian languages differ from those of the Arabic numerals (0 to 9).

2.4 Coding, Programming and Processing

Little work has been done in devising schemes for computer coding, that is, internal representation of Indian language characters. In fact, more work has been done to develop telecommunication codes [26, 34, 35] for

transmission than for computers. For processing of multilingual text it is necessary to develop a suitable internal representation. Such a code should have the alphabetic property, that is, the internal coding should permit a text to be sorted into the accepted collating sequence. Coding schemes based on the graphemes of a script do not have the alphabetic property. Hence, the mechanical typewriters which are based on the graphemes of a script are not suitable as computer input devices.

There is a flavour of natural language in some programming languages and job control languages (JCL). For example, consider the following COBOL sentences:

```
ADD ALLOWANCES TO BASIC-SALARY GIVING GROSS-SALARY.
IF GROSS-SALARY IS GREATER THAN DEDUCTION
    THEN PERFORM ERROR-IN-SALARY-ROUTINE.
```

Diagnostic messages are other examples. For user communication in Indian languages, these language constructs could be redefined. Preprocessing and postprocessing techniques have been tried to process programs using constructs written in Telugu [27], Tamil [36] and Hindi [37, 38].

2.5 Applications

In addition to the development of I/O devices, programming languages and associated compilers and assemblers, efforts were also made on various application areas such as text composition, wordprocessing, language translation, etc. They are briefly reviewed in the following paragraphs.

Introduction of Indian languages in computer based typesetting and photocomposition poses the greatest difficulty due to very high resolution and superior quality of script composition and generation demanded in various fonts and character sizes. In India, the most common technique for printing is hand composition. However, in large newspaper companies, keyboard activated semi-automatic methods are used. Some electronic keyboards based on conjunct characters are already in use for Devanagari and they employ shift and dead keys. These keyborads have no visual feedback and need concious and tedious keystroking. A CRT based phototypesetting for Indian languages is explained in [39]. The unavailability of suitable I/O devices is a major obstacle to the development of vernacular phototypesetters.

Changes were suggested for message transmission through the telex and telecommunication networks. Krishnaya and Vishwanathan [40] have stressed the need for

telecommunication systems to carry news in the language of its origination as well as in English, thus eliminating unnecessary translation from regional languages to English and back to the regional languages. Codes suitable for telex and telegraph in Indian languages have been developed [24, 26, 34, 35]. Ramakrishna [34] has developed information theoretic models for transcription of Indian languages into English and vice versa. Mahabala and Raman [41] have proposed methods for animated film titles in a regional language with the help of computers. An extensive review of typesetting developments in Indian languages and utilization of computers can be found in [42, 43].

An approach to the design of word processors using the ZILOG Z80 microprocessor is explained in [44]. One of the problems in word processing is hyphenation [44-46]. Because of the phonetic nature of Indian scripts, hyphenation in Indian languages is relatively simple. For instance, the following simple rules could accommodate most of the cases for hyphenation of Tamil texts:

Rule 1: The first character of a line should not be a consonant such as க (K), ங (NG), etc.

Rule 2: The symbol constituting a character should not be broken. For example, the character கெ (KE) should not be broken into the symbol க in one line and the symbol ெ in the next line.

Rule 3: A line should not end with just the first character

of a word except when the word itself contains only one character.

Medical information such as patient history, hospital resources, etc., are processed manually in India. Hence, there is a considerable amount of delay in processing and resource allocation. Automated written communication between the hospital and the patient cannot be done in vernacular languages using existing computers. Researchers [47, 48] have proposed database designs for medical information systems. The textwriter proposed in this thesis could be used for automated written communication in native languages. Since much of the information on health and family planning is generated and maintained in the Indian regional languages, such systems will be very useful.

Machine translation within limited scope has been studied by several authors. Namperumal has developed algorithms for translation of simple English sentences into Tamil [49]. Ganeshsundaram [50] proposed a translation scheme involving human interaction. Chakraborty, et al [51] have experimented with machine translation from Hindi to Bengali. These experiments are at a primitive stage and are far from any practical application.

The lack of suitable I/O devices and computing systems in Indian languages have been felt by linguists for many years. They circumvented the problem by transliterating

[52] into Roman script, doing research in areas such as deciding authorship of poems [53], concordance [54], etc.

Our survey indicates that most of the efforts are directed towards the development of I/O devices. Yet, suitable I/O devices are not yet available in the market. Standardization in aspects such as internal coding and transliteration have not been considered. Microprocessor-based I/O devices for Indian languages is a concept worth exploring in detail. Further references are cited at appropriate places in the following chapters.

3. AN INPUT DEVICE FOR A TEXTWRITER

3.1 Input Devices

There are three methods for the input of textual data: (1) keyboard, (2) document reading and (3) speech. Much of the work for Indian languages is in the area of keyboard design. As far as voice data entry is concerned a substantial work of practical importance is yet to be done. In a document data entry system, the input text may be either handwritten or printed and it could be entered on-line or off-line. Commercial optical character recognition (OCR) devices are yet to be developed for Indian languages. However, experimental techniques to recognize Tamil [55, 56], Telugu [57], Devanagari [58, 59], Bengali [60] have been reported. It is worthwhile to develop one powerful recognition technique for all Indian languages than one technique for each language.

For most applications, the data entry is, and will be for the foreseeable future, through a keyboard. Of the three methods, this is the most successful and reliable method of entering information at this time. The design of keyboards can be broadly classified into two types: (1) unilingual keyboards and (2) multilingual keyboards. With a unilingual keyboard, we can enter text of only one language,

whereas a multilingual keyboard facilitates entering of texts in more than one language from the same keyboard. Existing typewriter keyboards are of unilingual type. New keyboards [11, 23] suggested as modifications of existing mechanical keyboards are also of this type only.

Since there are many languages to be handled in the Indian context, it will be economical and convenient to have one keyboard through which we could enter texts in any language. The design of such multilingual keyboards for a textwriter is dealt with in this chapter.

3.2 Multilingual Keyboard Design

Let, $C = \{c_1, c_2, \ldots, c_n\}$

be the set of characters in a language. The cardinality, $||C||$ of C gives the total number (i.e., n) of characters in the language. For keyboard design, if $||C||$ is small, then we can assign each character in C to a key position on the keyboard. For example, $||C||$ for English is only 52 (26 uppercase letters, A to Z and 26 lowercase letters, a to z). Since this number is small, each of these characters is assigned a key position in the keyboards.

If $||C||$ is high, then it will not be possible to assign each character a key position. This is particularly true for Indian languages. To get an idea, let us briefly

review the structure of Indian language characters.

3.2.1 Structure of Indian Characters

The characters of the Indian languages can be classified into four types. These are (1) vowels, (2) consonants, (3) special characters and (4) composite characters. Characters other than composite characters are basic to a language. We call this set of basic characters, and the sound associated with each of them, the 'Basic-Letter-Sounds' (BLS). Composite characters are obtained by the combination of consonants and vowels. These can be further subdivided into consonant-vowel (c-v) characters and conjunct characters. C-V characters are obtained by the combination of one consonant and one vowel; conjunct characters are obtained by the combination of two or more consonants with a vowel. Conjunct characters are present in all Indian languages except Tamil. Special characters are those which do not combine with any other characters.

As an illustration consider the set of Tamil characters shown in figure 3.1. It has 18 consonants, க (K) to ன (N) shown in column 13, 12 vowels, அ (A) to ஔ (AU) shown in row 1 and one special character ஃ(AKH). The c-v characters are shown within the rectangle and there are 18 x 12 = 216 of them. They are

அ	ஆ	இ	ஈ	உ	ஊ	எ	ஏ	ஐ	ஒ	ஓ	ஔ	
க	கா	கி	கீ	கு	கூ	கெ	கே	கை	கொ	கோ	கௌ	க்
ங	ஙா	ஙி	ஙீ	ஙு	ஙூ	ஙெ	ஙே	ஙை	ஙொ	ஙோ	ஙௌ	ங்
ச	சா	சி	சீ	சு	சூ	செ	சே	சை	சொ	சோ	சௌ	ச்
ஞ	ஞா	ஞி	ஞீ	ஞு	ஞூ	ஞெ	ஞே	ஞை	ஞொ	ஞோ	ஞௌ	ஞ்
ட	டா	டி	டீ	டு	டூ	டெ	டே	டை	டொ	டோ	டௌ	ட்
ண	ணா	ணி	ணீ	ணு	ணூ	ணெ	ணே	ணை	ணொ	ணோ	ணௌ	ண்
த	தா	தி	தீ	து	தூ	தெ	தே	தை	தொ	தோ	தௌ	த்
ந	நா	நி	நீ	நு	நூ	நெ	நே	நை	நொ	நோ	நௌ	ந்
ப	பா	பி	பீ	பு	பூ	பெ	பே	பை	பொ	போ	பௌ	ப்
ம	மா	மி	மீ	மு	மூ	மெ	மே	மை	மொ	மோ	மௌ	ம்
ய	யா	யி	யீ	யு	யூ	யெ	யே	யை	யொ	யோ	யௌ	ய்
ர	ரா	ரி	ரீ	ரு	ரூ	ரெ	ரே	ரை	ரொ	ரோ	ரௌ	ர்
ல	லா	லி	லீ	லு	லூ	லெ	லே	லை	லொ	லோ	லௌ	ல்
வ	வா	வி	வீ	வு	வூ	வெ	வே	வை	வொ	வோ	வௌ	வ்
ழ	ழா	ழி	ழீ	ழு	ழூ	ழெ	ழே	ழை	ழொ	ழோ	ழௌ	ழ்
ள	ளா	ளி	ளீ	ளு	ளூ	ளெ	ளே	ளை	ளொ	ளோ	ளௌ	ள்
ற	றா	றி	றீ	று	றூ	றெ	றே	றை	றொ	றோ	றௌ	ற்
ன	னா	னி	னீ	னு	னூ	னெ	னே	னை	னொ	னோ	னௌ	ன்

Fig. 3.1 Set of Tamil Characters

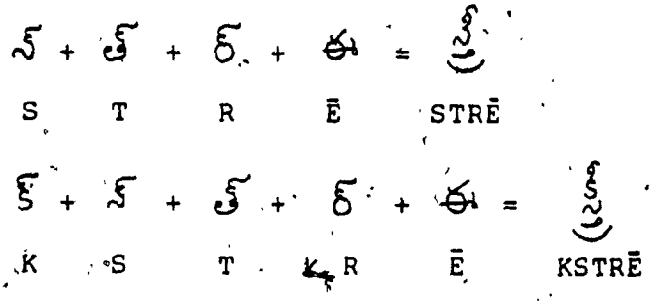
obtained by the combination of the vowels and consonants in their respective columns and rows. For example, the c-v character ெக (KE) is obtained by the combination of the vowel ெ (E) and the consonant க் (K)

That is, க் + ெ = ெக
K E KE

In modern Tamil literature, five additional consonants and their associated c-v characters are also used occasionally. They are listed on the bottom five lines in figure 3.1. In figure 3.1, it could be observed that the composite characters of a row are similar (in appearence or grapheme) to the consonant of that row. The combining vowel, with a few exceptions, assumes a specific graphic symbol, called the sign of the vowel, in the c-v character. For example, ' ெ ' is the sign of the vowel ெ (E); ' ா ' is the sign of the vowel ' ஆ (Ā) with the exception of ண (ṆĀ), ஈ (ṚĀ) and ன (NĀ); and so on.

Just as an illustration, a few conjunct characters from Telugu are shown below with the method of derivation (the characters to the right of the equal sign are the conjunct characters).

స్ + త్ + ఈ = స్తీ
S T Ē STĒ



It should be stated that the derivation procedure for the sounds of the composite characters is systematic. For example, consider all the c-v characters obtained by the combination of the vowel ఎ (E) in Tamil (refer to figure 3.1). Their graphemes are easily obtained by concatenating the vowel sign ె with the graphemes of the c-v characters in column 2, whereas such a simple rule is not valid for the c-v characters of ఉ (U) and ఊ (Ū).

3.2.2 Characteristics of Indian Languages

The important characteristics of Indian languages are summarized below:

(1) There are many languages in use and in each language there are many characters, ranging from a minimum of 247 in Tamil to over 2000 in Telugu.

(2) The characters are not uniform in size and extend widely both horizontally and vertically.

(3) Indian characters are complex to generate due to vowel symbols, half characters, and conjunct characters which lead to complex graphics.

(4) Some of the rules for attaching modifiers and forming conjunct characters are context dependent.

(5) There are exceptions in the rules for the generation of graphemes of the c-v and conjunct characters.

(6) Although the Indian languages have different scripts, they possses common basic-letter-sound characteristics.

Two approaches may be considered for building keyboards:

1. Symbol based approach.

2. Basic-letter-sound based approach.

3.3 Symbol Based Multilingual Keyboard Design:

In this approach, the character set C of a language is analysed, decomposed and a set of primitive symbols are identified. Let,

$$S = \{s_1, s_2, \ldots, s_m\}$$

be the set of symbols. Each symbol is a partial graphic of a character. Therefore, any character in C could be constructed by using one or more symbols in S. The decomposition of characters in C should be done in such a way that $||S||$ is small (as in English). To get an insight into character analysis and symbolic decomposition, let us

analyse the characters of Tamil in figure 3.1, column by column.

3.3.1 Character Analysis

Column 1: The characters in column 1 are basic. It is not only difficult, but also inconvenient to break them into smaller symbols. Hence we will treat them as primitive symbols. There are 19 symbols in this column { அ (A) to ன (NA)}.

Column 2: The characters in this column could be obtained by using the symbols in column 1 plus ◌ா , ஞ (NĀ), ற (RĀ), ந (NĀ) and ஆ (Ā). Hence these five symbols join the set of basic symbols from column 1.

Column 3: The c-v characters in this column could be obtained by appending the symbol ' ◌ி ' to the symbols in column 1, except for the character ட (TI) for which the place of appending is different from the other characters. Hence, three symbols இ (I), டி (TI), and ◌ி from this column join the basic symbols.

Column 4: Same as column 3 except that the appending symbol here is ' ◌ீ '. Therefore, the three new symbols from this column are ஈ (Ī), டீ (TĪ) and ' ◌ீ '.

Column 5: The four c-v characters ஙு (NGU), பு (PU),

யு (YU), and ஙு (VU) are obtained by appending the symbol ' ு ' with their respective symbols in column 1. The remaining are to be retained without change. Hence there are fifteen plus one, sixteen new symbols from this column.

Column 6: The four c-v characters ஙூ (NGŪ), பூ (PŪ), யூ (YŪ) and வூ (VŪ) could be obtained by adding the symbol ' ூ ' to their counterparts in column 1. The characters ஞா (GNŪ), ணா (ṆŪ), தா (ṬŪ), நா (NŪ), ஷா (LŪ), ஸா (RŪ) and ஜா (ṈŪ) could be obtained by appending the symbol ' ா ' to their counterparts in column 5. The characters ஊ (TŪ), மூ (MŪ), ரூ (RŪ), ழூ (ḶŪ) and ளூ (ḺŪ) could be obtained by appending the symbol ' ூ ' to their counterparts in column 5. Thus the additions from this column are ' ா ', ' ூ ', கூ (KŪ), சூ (CŪ) and ஔ (Ū).

Column 7: ெ (E) and ே are the additional symbols required to obtain the characters in this column.

Column 8: ே (Ē) and ை are the additional symbols required to obtain the characters in this column.

Column 9: The symbols from this column are ' ஂ ' and ' ஃ ' and the vowel ஐ (AI).

Column 10-12: Require only the two vowel symbols ஒ (O) and ஓ (Ō). The symbols constituting the vowel

(AU) were already included.

Column 13: ' o ' is the new symbol required for this column.

Therefore the total number of symbols required to generate all the Tamil characters is 61; they are listed in figure 3.2. This is a reasonable number compared to 52 symbols for English and could be easily accommodated on a keyboard of reasonable size. The characters of Malayalam may be analysed by a similar procedure to obtain the primitive set of symbols. They are listed in figure 3.3.

To simplify our problem of decompostion, the mechanical typewriter designers have done the analysis to a certain extent. Symbol-based mechanical typewriters are available for most of the Indian languages. But it may not be possible to adopt them as such for computer applications [11]. Nevertheless they could be used as a starting point and refined further [11].

3.3.2 Keyboard Layout

The primitive set of symbols, S, corresponding to the characters of a language are assigned to the keys on the keyboard according to their frequencies of occurrences. Since symbol based keyboards for major Indo-European languages using the Latin script and the Indian scripts are

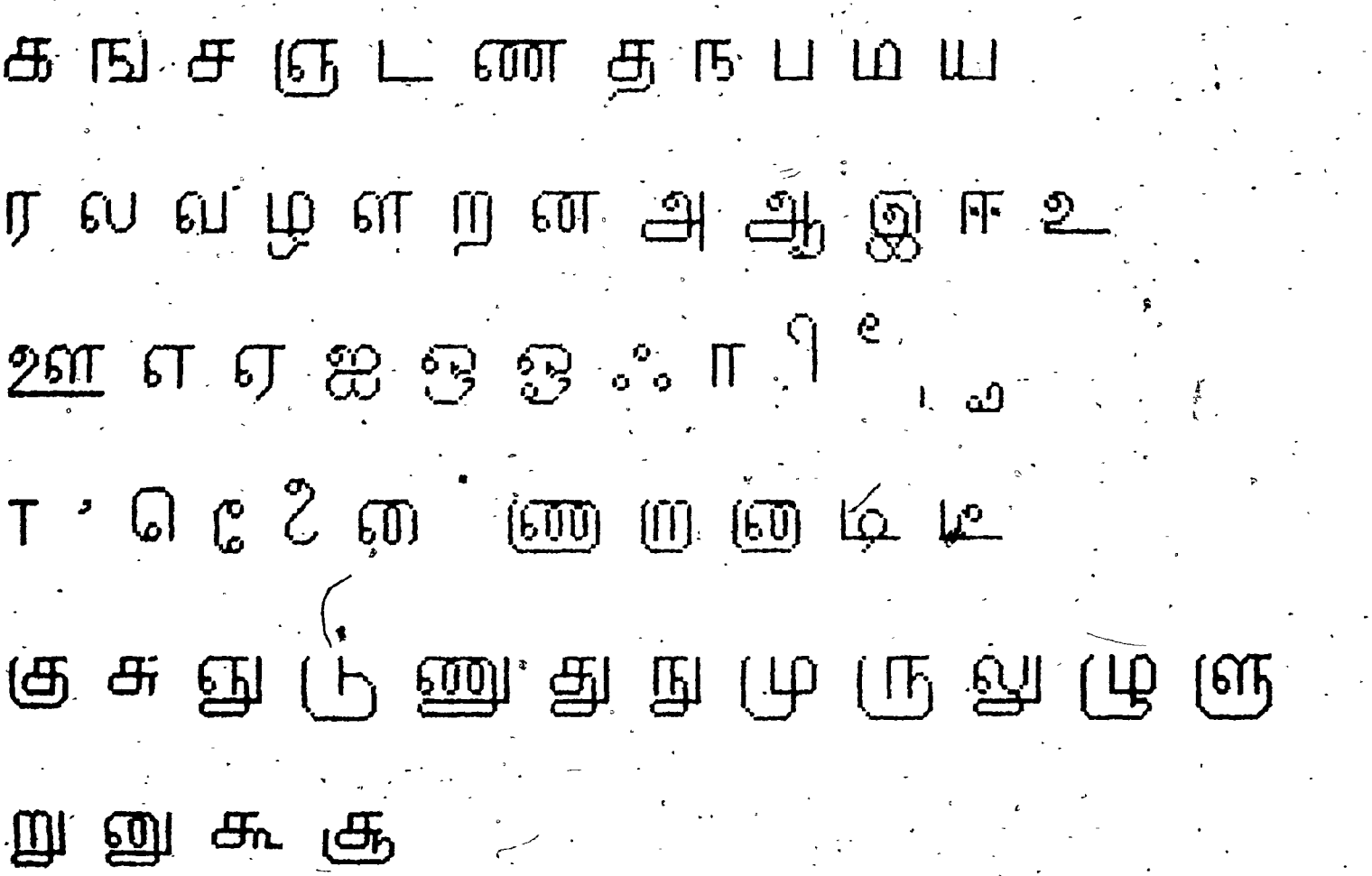


Fig. 3.2 Set of Tamil Symbols

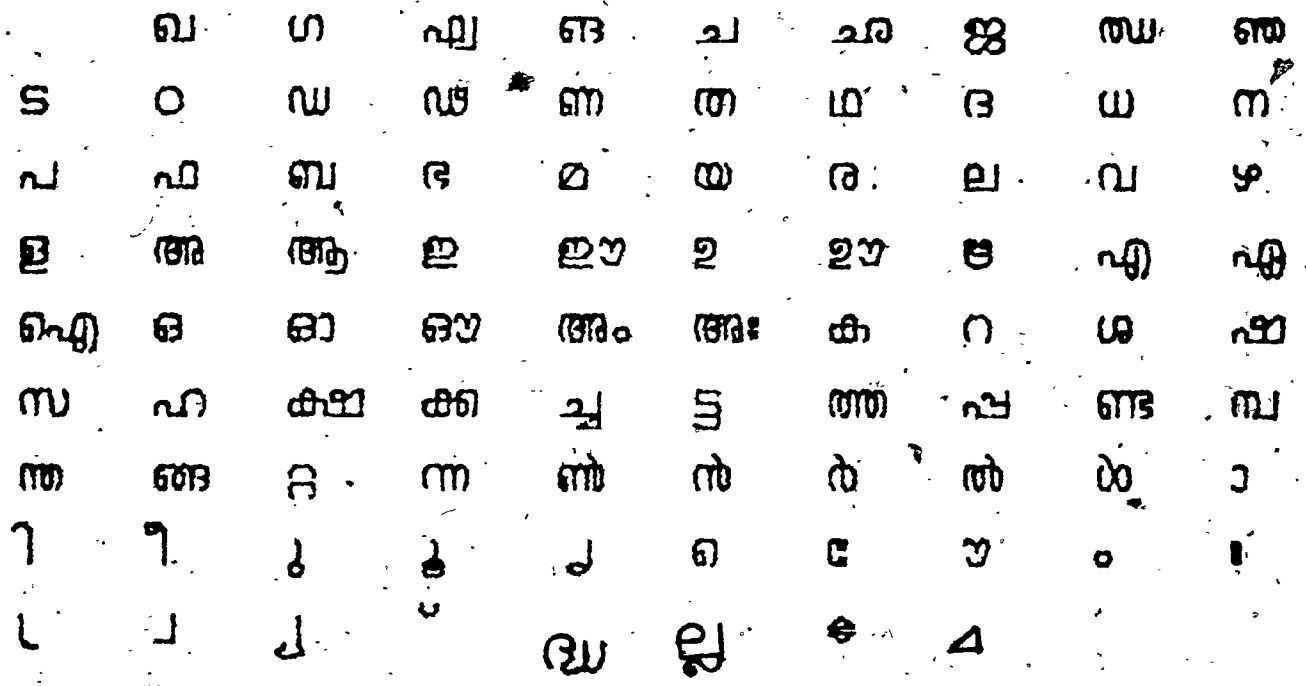


Fig. 3.3 Set of Malayalam Symbols

available, it will be advantageous if the multilingual keyboard layout also does not depart appreciably from those existing. Moreover, we will be growing increasingly towards a multipurpose keyboard, that is, the same keyboard will be used for an electric typewriter, data preparation, telex and computer communication. Therefore, it will be better to adapt the existing typewriter keyboards, perhaps with a few extra keys as indicated in the following paragraphs. Also, this has the advantage that conventional typing schools can train the operators in developing speed.

Figure 3.4 gives the layout of a multilingual keyboard with Tamil symbols. It consists of 72 keys including the space bar. This keyboard differs marginally from the existing typewriter keyboard for Tamil. The differences are the following.

1. The increased number of keys is due to an increase in the number of special symbols and the addition of a language selection code (L.S.C).

2. The character ஔ (NU) is obtained by one key depression in the multilingual keyboard whereas in mechanical typewriters it is obtained by two key depressions. It is designed so in the mechanical typewriter due to larger length of the character and mechanical design constraints which will not be there in an electronic keyboard. Such changes, if ~~any, are~~

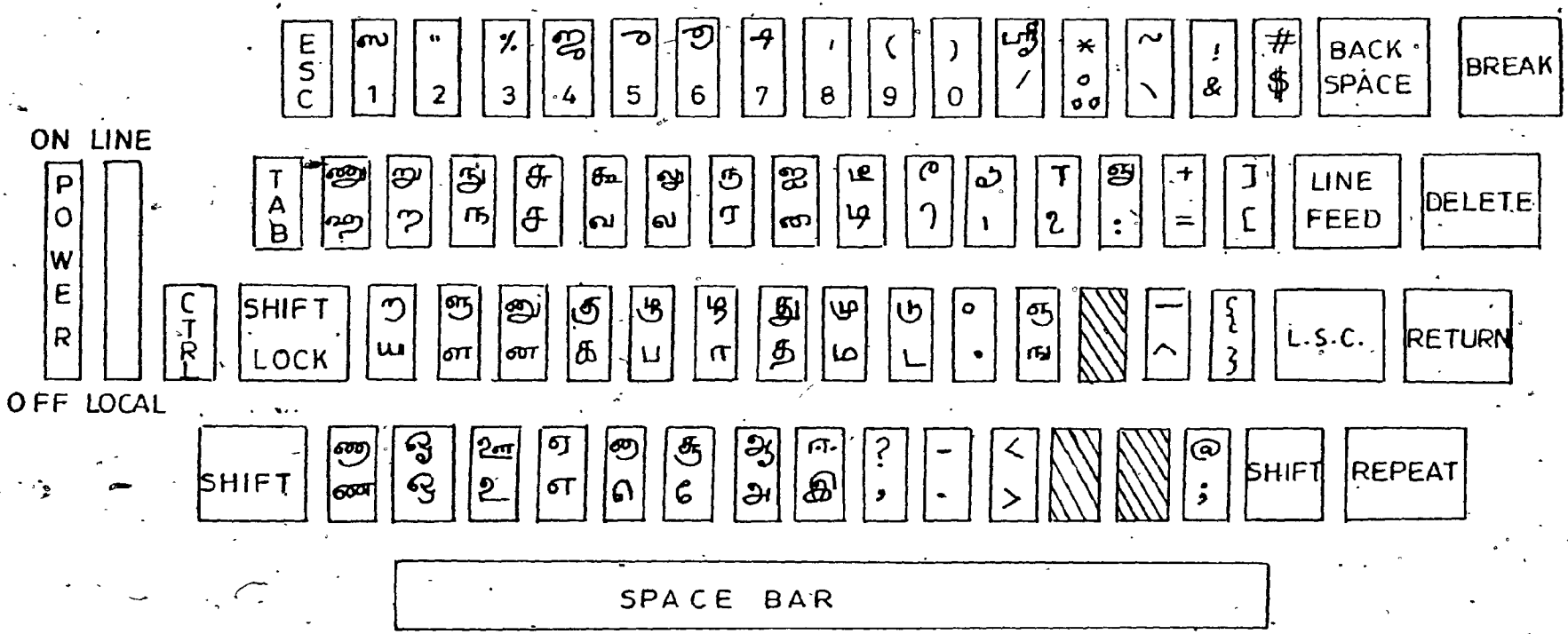


Fig. 3.4 Multilingual Symbolic Keyboard with Tamil Symbols

to be made for other language keyboards also.

3. The character ஹ (HA) is provided as a separate character whereas in mechanical typewriters it is obtained by typing the characters உ (U) and ர (RA) one after another and manually adjusting the space to touch each other. உ (U) and ர (RA) are themselves two independent characters. By typing them consecutively, it will not be possible to decide whether the combination is one character ஹ (HA) or two characters உ (U) and ர (RA). Similar changes are also to be made in other language keyboards wherever applicable.

4. The character ஜ (GNU) is provided. This character is not present in the mechanical typewriter. It is also not used by printers.

5. There is no dead key in the proposed keyboard.

6. The character like கி (KI) must be typed in the order க , and ி whereas it is typed in the mechanical keyboard as ி , and க using the dead key containing the symbol ி .

The meaning and functions of the keys like TAB, SHIFT, SHIFT LOCK, LINE FEED, RETURN, etc., are the same as those of any standard computer terminal keyboard such as in Digital Decwriter II, ADM-3A, Tektronix terminals.

3.3.3 Multilingual Text Entry

The keyboard in figure 3.4 could be used to enter the text of major Indian languages and European languages such as English, German, French and others. Since our interest is to handle (or enter) text of many languages having different scripts, it is necessary to indicate the selected language by means of a code. Such a coding should be suitable for internal representation as well. In other words, given a string of symbols, the set of consecutive symbols belonging to a particular language must be identified. This could be achieved by prefixing a unique language code to the text as shown in figure 3.5. By this scheme the text of any number of languages could be represented.

Implementing the 'language code' on the keyboard may be done in many ways. One possible method is shown in figure 3.6. The 'language Selection Code' is a unique one byte code (constant), the occurrence of which indicates that the text of the current language has ended and a text of another language is going to follow. This language selection code is generated by depressing the key [L.S.C] provided on the multilingual keyboard (figure 3.4). Once we know that there is a change in the language, then the next question is to which language the change is taking place.

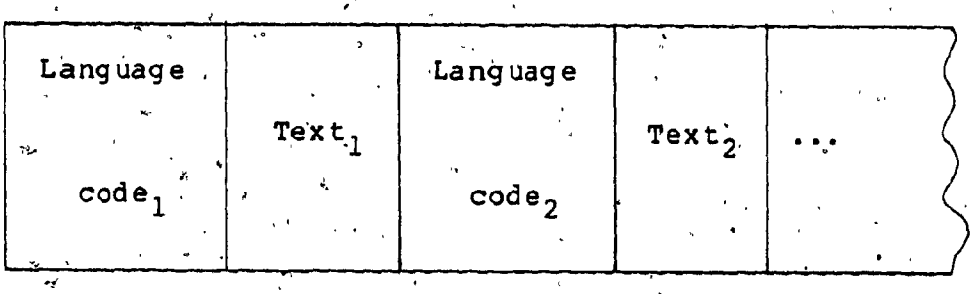


Fig. 3.5 Multilingual Text Representation

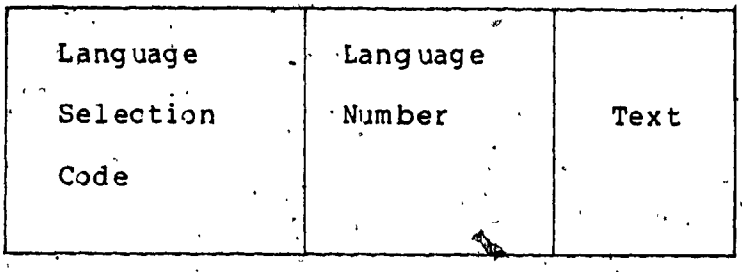


Fig. 3.6 Language Coding

This is indicated by the 'language number'. In other words, the language number tells to which language the succeeding text belongs to. Each language is assigned a number such as English 0, Tamil 1, Malayalam 2 and so on. The number of digits used for language number depends on the total number of languages supported by the computing system: one digit up to 10 languages, two up to 100 languages, and so on. It is necessary to understand the need for separating the language selection code and the language number. The function of the language selection code is to merely indicate that the language change is taking place; it does not say to which language the change is taking place. This is indicated by the language number.

3.3.4 Reduction of Keys

The number of keys could be reduced by associating special symbols such as + - * % & ' " only with the English language version of the keyboard. The overhead in doing so will be that for specifying the special symbols, a language change must be indicated. To avoid frequent changes, the most commonly used characters could be included in the symbol set for all languages. This will, of course, increase the number of keys correspondingly. Another alternative is to use 'CTRL' key (which gives a third shift)

for special symbols and to use the communication code as part of English.

3.3.5 Key-Top Symbol Display Techniques

Since the same keyboard unit is used to enter multilingual text, it is neceesary to display on the key-top the appropriate symbols of the language it currently handles. This could be achieved in many ways of which three are explained below:

Key Replacement: In this method the keys on the keyboard are replaced manually one by one whenever a change of language takes place. This is inexpensive, but suitable only if the change does not take place often. Since there is a definite possibility of misplacing the keys, they should be replaced with the help of a keyboard layout and verified thoroughly before entering new data.

Template Replacement: Here, instead of replacing the keys, all the key symbols are placed on replaceable templates, one for each of the languages supported. Designed in one of several ways, and with appropriate markings on movable keys, it may be inserted on various keyboard models with comparative simplicity. With current technology, the flexibility of "touch-tone" type systems could also be easily adapted for this purpose.

Display Panel: Instead of the mechanical and manually replaceable templates, electronic displays, especially the plasma display type panels, would be suited for our purpose. Each key display could be designed to have a plasma panel and a driver or the entire key display could be driven by one driver. The code for the symbols of each language could be stored in ROMs. As of now, the cost of plasma panels is high for an application of this kind, but as the cost of electronic display components is decreasing, it is likely that suitable displays will be available soon.

The key top symbol display problem could be simplified to a certain extent by grouping the characters of languages having similar scripts and perhaps differing in a few characters. For example, a single keyboard layout could be provided by grouping the diacritical marks for different Latin script languages. The key top display could be retained without change for all the languages in one group. Such a group exist even in Indian languages. For example, Hindi, Marathi and Sanskrit have identical characters except for a few; Assamese, Bengali and Manipuri have similar characteristics. The effect of such a keyboard is that it will have relatively more keys.

3.4 Basic-Letter-Sound Based Keyboard Design

For European languages having very few characters with simple graphemes, the need for alternate designs for keyboards did not arise. Such is not the case with Indian languages. An alternate keyboard design could be based on the common Basic-letter-sound characteristics of Indian languages. Such a design is described in this section.

3.4.1 Basic-Letter-Sounds

The analysis of Indian language characters has revealed that the vowels, consonants and special characters are the basic characters in any Indian language and they are asociated with unique sounds. Let us call these basic characters the set of basic-letter-sounds or BLS. Let

$$BLS = \{ \alpha_1, \alpha_2, \ldots, \alpha_n \}$$

be the set of basic-letter-sounds. Any character in a language could be obtained by a combination of α_is from the set BLS. The interesting properties of the set BLS are the following:

1. The ||BLS|| for any Indian language is small (of the order of 55).
2. A string of αs unambigiously identifies the characters in the language.
3. The α-coded text when sorted gives the text in lexical order (refer to figure 3.7). This is not the

Unsorted	Sorted
அன்பெனும் 11	அன்பெனும் 11
பரசிவம் 12	இரவுறும் 17
தரமிகும் 13	எங்கேமெய் 18
சம்பு 14	கானல்நீர் 19
மரபுறு 15	சம்பு 14
வான 16	சீர்கொண்ட 20
இரவுறும் 17	தரமிகும் 13
எங்கேமெய் 18	பரசிவம் 12
கானல்நீர் 19	மரபுறு 15
சீர்கொண்ட 20	வான 16

Fig. 3.7 Lexical ordering of α-coded Text

case with symbolically coded text.

4. The cardinality of the union of BLS sets for a set of Indian languages is also small (of the order of 70). This is because there are many basic-letter-sounds common among Indian languages.

The set BLS for Tamil is

BLS = { அ to ஔ , ... , க் to ன் }

which has only 31 basic-letter-sounds. The composite characters like க (KA) are obtained by the combination of two basic-letter-sounds க் (K) and அ (A). Therefore, க (KA) could be expressed as க் அ (KA).. Similarly, the other composite characters could be expressed as combinations of ∝s.

3.4.2 BLS Based Keyboard Layout

A multilingual keyboard could be designed based on the basic-letter-sounds by alloting each ∝ to a key position. The BLS based keyboard could be designed in two ways.

Method 1 : The ∝s of a language are assigned to the keys one at a time. It is quite possible to accommodate the ∝s of any language by adding a few keys to the existing keyboards. However, this arrangement of keyboard requires

the αs of different languages to be displayed on the key top whenever there is a change in language.

Method 2 : In order to avoid the language dependent key top display for each language, one could use the union of αs of the set of all Indian languages. To design such a keyboard, the basic-letter-sounds of 12 Indian languages were obtained and are listed in Table I. Only the transliteration into Roman script (according to the Library of Congress Cataloging Bulletin) is shown in the Table to avoid a lengthy presentation.

A keyboard layout with the αs of Table I is given in figure 3.8. It has 62 keys excluding the space bar. In this keyboard, some of the keys will remain unused for a particular language corresponding to those αs which are not present in that language. Figures 3.9 and 3.10 give the keyboard layout with English characters and Tamil αs respectively. The unused key positions are crossed. It could be noticed that the layout for English is exactly the same as that of the present keyboards except for the addition of keys for language selection code to make it a multilingual keyboard.

The layout for Tamil includes only a few special symbols needed very often. Other special symbols could be specified as English symbols by indicating the appropriate language code or by using the 'CTRL' key.

Table I Set of αs of Indian Languages

Vowels and Special Characters									
A	Ā	I	Ī	U	Ū	Ṛ	Ṝ	Ḷ	Ḹ
Ĕ	E	Ê	Ē	ĂI	AI	Ŏ	O	Ô	Ō
ĂU	AU	AṂ	AḲ	AḴH	SHRI				

Consonants									
K	KH	G	GH	Ṅ	C	Ĉ	CH	J	J̆
JH	Ñ	Ṭ	ṬH	Ḍ	Ṛ	ḌH	ṚH	Ṇ	T
T̲	TH	D	DH	N	P	PH	B	BH	M
Y	Y̆	R	L	W	V	L̤	L̤	R̤	N̤
S	Ṣ	S	H	KṢ	F	Z	R	RXH	Q

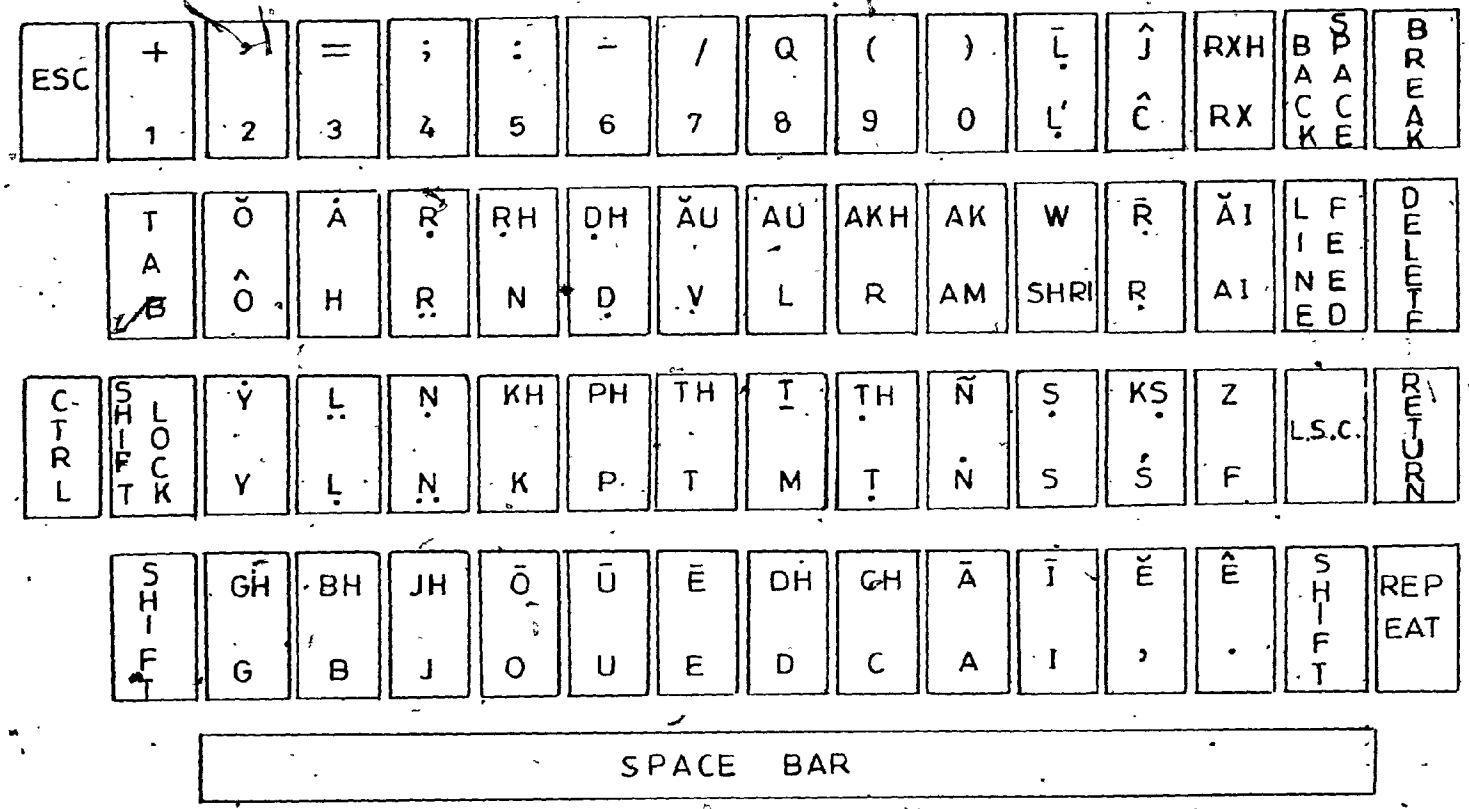


Fig. 3.8 Multilingual BLS Keyboard with ∝s of Indian Languages

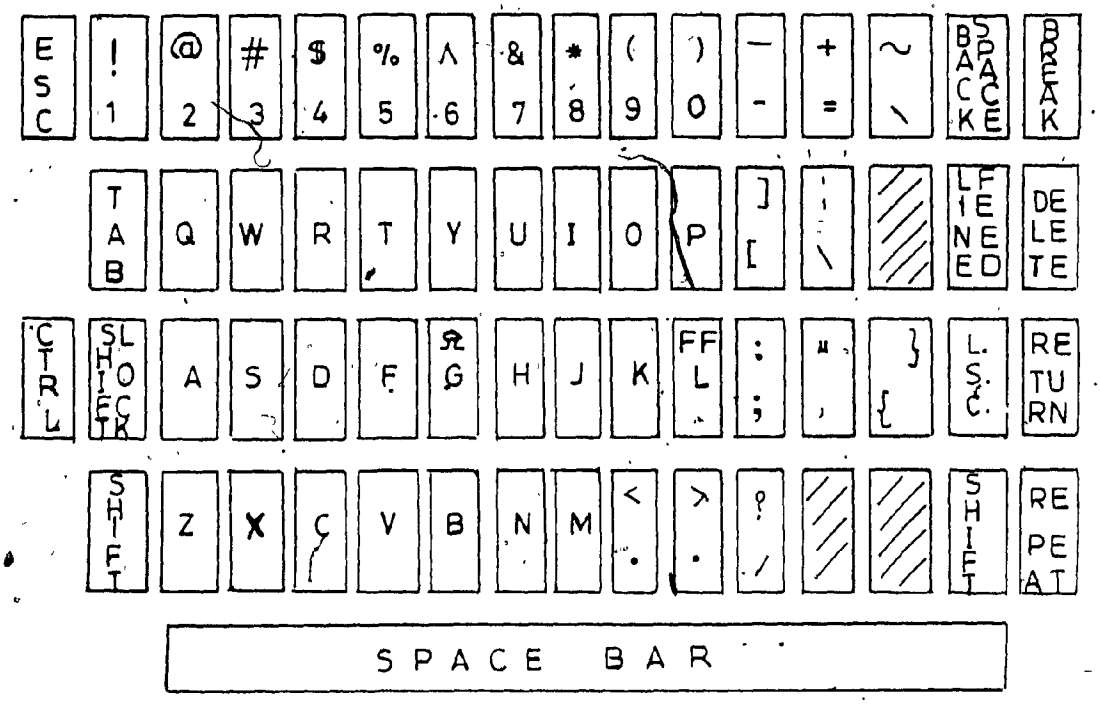


Fig. 3.9 Multilingual BLS Keyboard with English Characters

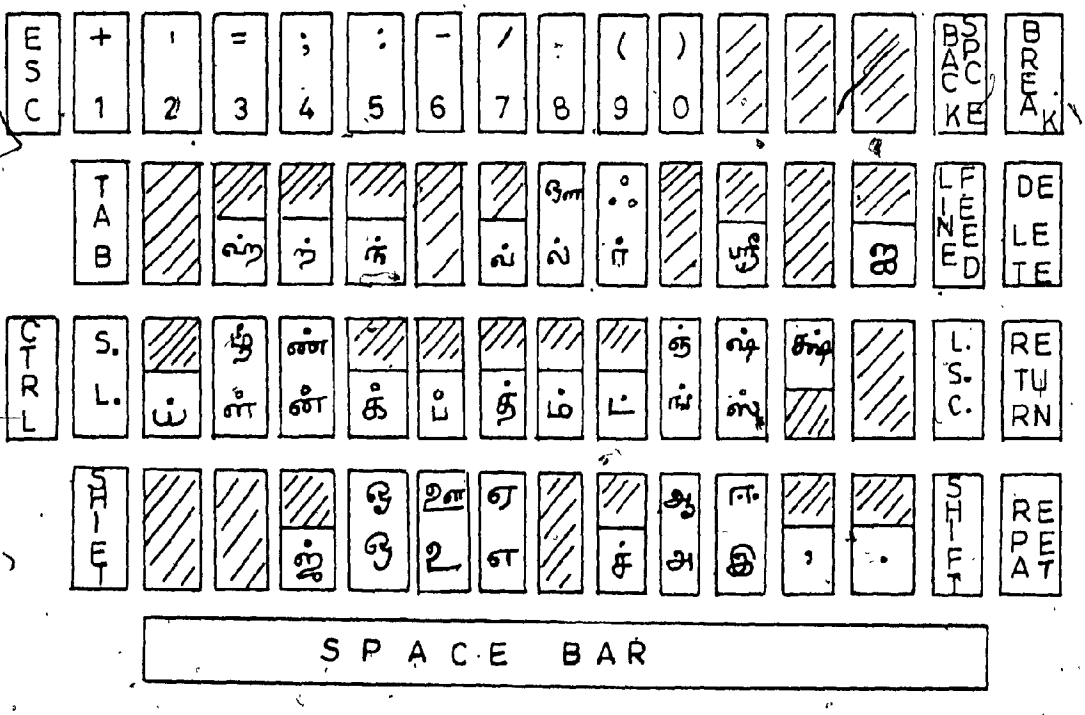


Fig. 3.10 Multilingual BLS Keyboard with ∝s of Tamil

A unilingual keyboard based on the phonetic properties of Hindi and Telugu was proposed by Narasimham and Sinha [29, 62, 63]. Their proposal has the following shortcomings:

1. Apart from giving the correct sequence of αs and diacritical marks, it is also necessary to indicate which of them are to be grouped to form composite characters.
2. The keys are laid in a 'V' shape.
3. A large number of keys (62) are in a single shift which makes touch typing difficult.
4. They are unilingual keyboards.
5. It was designed only for Indian languages.

3.4.3 Placement Criteria

In order to allocate the αs to various keys, we obtain the frequency distribution of the αs that may vary from one language to another. The frequency could be approximated as the average over all languages. A simple expression for estimating the frequency of an α is given below:

Let L : be the number of languages

C_k : the set of characters in language k where k varies from 1 to L

$||C_k||$: total number of characters in k^{th} language

fC_{ki} : the character frequency of i^{th} character in k^{th} language where i varies from 1 to $||C_k||$

BLS : the set of $\propto$s

$||BLS||$: total number of $\propto$s

BLS_j : j^{th} $\propto$ in BLS where j varies from 1 to $||BLS||$

$fBLS_{kj}$: the frequency of j^{th} $\propto$ in k^{th} language

$fBLS_j$: the average frequency of j^{th} $\propto$ over L languages

Then, $$fBLS_{kj} = \sum_{i=1}^{||C||} m_i * fC_{ki}$$

where m_i is the number of times the BLS_{kj} occurs in the character C_{ki}. For example,

$m_i = 0$ when BLS_{kj} is a vowel and C_{kj} is a consonant

$m_i = 1$ when both C_{kj} and BLS_{kj} are the same consonant

$m_i = 2$ for a composite character of type c-c-v (two consonants and one vowel) having both the consonants of the same type as that of BLS_{kj}.

Then, $$fBLS_j = 1/L \sum_{k=1}^{L} fBLS_{kj}$$

As an illustration consider the Tamil characters in figure 3.1. The frequencies of these characters obtained from a sample of 14,000 characters taken from modern prose are given in Table II [23]. The frequency of ு ∝ = அ (A) is obtained by summing the values in column 1 and that of ∝ = க (K) is obtained by summing the values in row 2. The special characters ஃ (AKH) and ஸ்ரீ (STRĒ) have the values as in Table II.

The above expression for $fBLS_j$ is based on the assumption that the probability of occurence of a text of a particular language is the same as that of the other. But this may not be true. Depending on the region where the machine is used, the probability will differ. For example, a machine in Tamil Nadu is likey to handle more Tamil text than text of any other language. Taking this into account, the expression could be modified as:

$$fBLS_j = \sum_{k=1}^{L} p_k * fBLS_{kj}$$

where p_k is the probability of occurence of text of language k. But this will result in non-uniformity of keyboard layout from region to region, and is not desirable from the point of view of manufacturing, portability, etc.

Table II Frequencies of Tamil Characters

	அ	ஆ	இ	ஈ	உ	ஊ	எ	ஏ	ஐ	ஒ	ஓ	ஔ	ஃ	ஸ்
	277	077	224	006	089	009	200	024	010	058	007	000	000	008
க	616	126	168	009	232	041	005	025	052	073	022	000	430	கீ
ஙு	002	000	000	000	000	000	000	000	000	000	000	000	135	ஙூ
ச	115	020	121	010	032	003	048	013	015	031	007	001	087	சீ
ஞ	007	002	000	000	000	000	000	000	001	000	000	000	011	ஞீ
ட	166	044	146	003	176	001	002	016	084	001	000	000	189	டீ
ண	056	005	027	001	002	000	000	000	005	000	000	000	146	ணீ
த	382	118	248	007	332	008	033	027	063	018	020	000	362	தீ
ந	062	066	037	023	002	011	007	007	002	000	010	000	243	நீ
ப	286	082	109	007	065	006	062	022	011	037	061	000	251	பீ
ம	144	096	040	007	110	006	004	014	047	004	004	001	493	மீ
ய	247	018	095	000	070	002	005	025	039	000	011	000	062	யீ
ர	189	062	117	011	278	001	002	006	026	002	005	000	322	ரீ
ல	126	053	061	000	049	002	005	022	076	000	005	000	318	லீ
வ	354	083	150	015	053	001	025	064	019	000	001	000	025	வீ
ழ	032	002	031	000	031	000	000	000	015	000	000	000	041	ழீ
ள	055	017	064	003	043	004	001	002	067	001	004	000	228	ளீ
ற	118	042	080	000	148	000	001	008	042	003	003	000	162	றீ
ன	150	058	060	000	038	002	000	009	047	002	007	000	494	னீ
ஜ	020	003	000	001	000	000	000	004	003	000	000	000	004	ஜீ
ஷ	013	001	000	000	000	000	000	000	000	000	000	000	009	ஷீ
ஸ	002	000	002	000	000	000	000	000	000	000	000	000	029	ஸீ
ஹ	011	001	001	000	000	000	000	000	000	000	000	000	000	ஹீ
க்ஷ	000	002	000	000	000	000	000	000	000	000	000	000	000	க்ஷீ

3.5 Comparison of Symbolic and BLS Based Keyboards

1. There is an inherent but unavoidable drawback in allocating symbols to keys in a symbol-based keyboard. That is, the symbols of characters having the same basic sound are placed on different keys for different languages. For example, அ (A) in Tamil and अ (A) in Hindi will be in different key positions, because the frequencies of two characters having the same basic sound may differ from language to language.

   An effect of this language dependent distribution on the keyboard operator lies in adapting to the changes in language. More errors in typing will occur if language changes are frequent. This could be alleviated to a certain extent by the suboptimal load distribution to fingers. On the other hand the BLS based keyboard does not suffer from this problem because there is only one key for a given α_i.

2. In the symbolic keyboard there will be immediate feedback (echo) for each key depression whereas in a BLS based keyboard, the character will appear only after keying the complete α-code of the concerned character.

3. A key-top symbol display is needed for the symbolic keyboard, whereas it may be eliminated with BLS based

keyboards.

4. With the BLS based keyboard, the operator has to learn, in addition to the graphemes, the BLS-structure of the script.

5. The keyboards could be compared on the basis of the average number of key depressions (AVG) required per character. This could be computed by using the following expression:

$$AVG = \sum_{i=1}^{||C||} p_i * d_i$$

where p_i is the probability of occurrence of character i, d_i is the number of key depressions required for character i and $||C||$ is the total number of characters in the language. d_i will be different for the symbolic keyboard and the BLS based keyboard. Using the character frequency in Table II, the average number of key depressions required using the symbolic keyboard was found to be 1.557, and that for the BLS keyboard 1.627. This shows that both the keyboards are equally almost efficient. It should be noted that the character frequency of those characters requiring three key depressions are mostly zero in Table II. With more samples they will have non-zero entries. This will increase the average key depressions per character in

the case of the symbolic keyboard and decrease the average key depressions in the case of the BLS keyboard, since the BLS keyboard requires only two key depressions for these characters.

6. Internal machine coding based on BLS characteristics gives the text in collating sequence on sorting, whereas symbol based coding does not.

4. AN OUTPUT DEVICE FOR A TEXTWRITER

The text entered from a keyboard must be echoed. As well, output from the computer needs to be either printed or displayed in the selected language. A scheme for displaying text is shown in figure 4.1. The BLS-coded text is first decoded and then the appropriate characters are generated. It is necessary to have one BLS decoder and an associated text generator for each language supported. Because of the non-uniform size of characters, the large number of characters and the many languages involved, it will not be possible to design electromechanical printers with 'typefaces' (for example, mechanical typewriters, drum printers, chain printers). The feasible output devices are dotmatrix type printers, line type plotters and dotmatrix or line type display devices.

4.1 Problems in Character Generation

To display (or print) the information entered, the key code must be interpreted to generate the required characters. To display (print) the information from the computer, the internal code must be decoded. In this context the following complications arise:

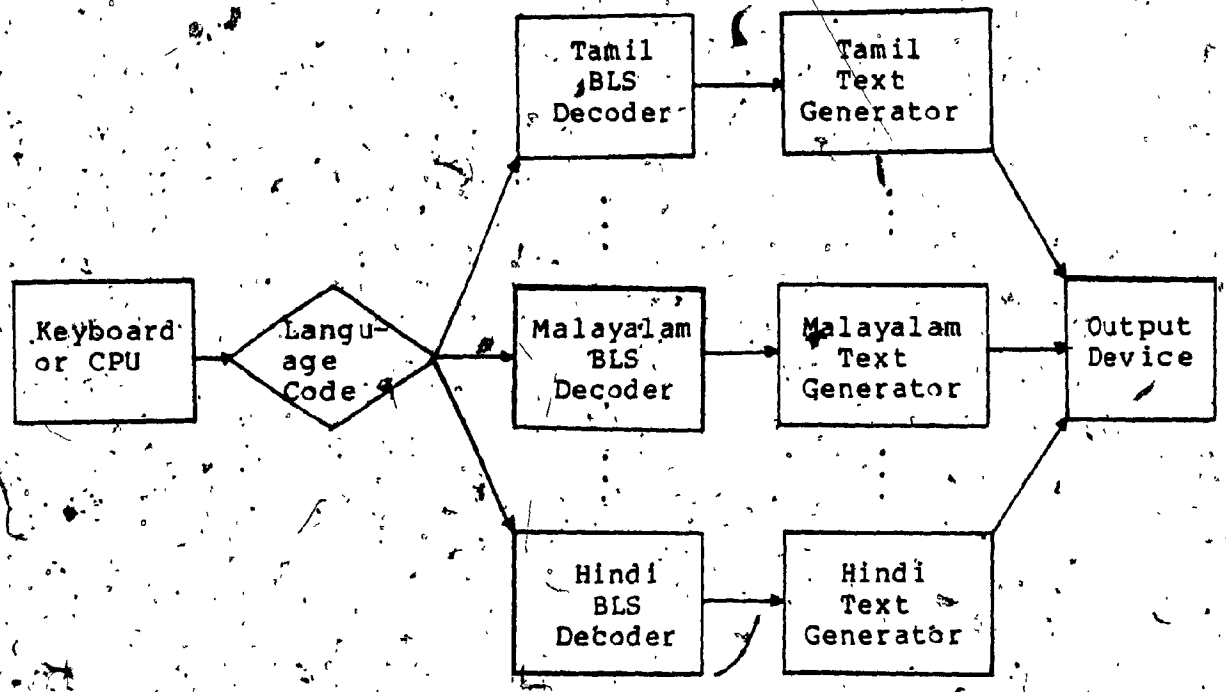


Fig. 4.1 Scheme for Generating Text from BLS Code

1. Characters will have different lengths of ∝-codes. For example, க் (K) is one unit long, but க (KA) is two (க் and அ); க்க (KKA) is three (க் , க் and க), etc.

2. The characters in a combination do not always appear side by side as in English. In some cases, one is below the other, either touching or not { க் (K), க்ஷ (KLU), ஸ்ரீ (STĒ)}; in some cases they extend horizontally as well as vertically { கூ (GŪ)}; and yet in other cases slightly shifted below and sideways as well { கு (GR), ர (RA)}.

3. The characters vary in both height and width. For example ண (NA) and ணா(NĀ) are two combinations in Tamil of ண and ா which are unequal in size.

4. In all Indian languages, there are very many characters, the number varying from 247 in Tamil to over 2000 in Telugu.

Before going into the techniques of generation, we must note an important difference between generating characters in Indian languages and in English. On a European keyboard, each symbol on the key is a complete character. Hence, the character could be displayed without the need to keep track of the preceding of succeeding symbols (characters) entered. Such is not the case with Indian characters. To generate

some of the characters of Indian languages it may be necessary to append (attach) the currently entered symbol to the previous symbol or vice versa, because a character may consist of one or more of the symbols on the keys. In the conventional Tamil typewriter, the combination கி (KI) is obtained by typing the symbols க and ி in the reverse order. To assure that symbols do not appear separately as கி, the key with symbol ி is a dead key. If the same typing order is carried over to computer terminals, the occurrence of the symbol ி must be remembered until the next key depression. But this will deny immediate visual feedback to the operators. To avoid this we can enter the symbol க first and ி second, which is also the natural order of writing.

To show the variations, let us see another example. On the Devanagari keyboard, the symbol ि is the sign of the vowel इ (I). Even though it appears always before the owner consonant in a syllable, it can appear in two different forms as कि and किं.

The sequence of entries are ि and क to get कि; and ि, क and ं to get किं. We see that the symbol ि does not get attached to the immediately succeeding symbol but may extend beyond. These are not the only exceptions; every Indian language has a number of

such special cases. Hence, character generation is not trivial.

4.2 Unified Approaches to Multiscript Character Generation

The output device of a textwriter need not be restricted to one type. In other words many types of output devices such as dotmatrix printer, display device, plotter, etc. could be used. Therefore, a multilingual text generation system should be as device independent as possible in order to be portable between different types of devices and between different makes of the same type. Our analysis of the scripts of major world languages, has revealed that it is possible to develop a unified approach for multilingual text generation. Two different approaches to generate text from α-coded input strings are explained in the following sections.

4.2.1 General Hierarchical Approach

One approach to generate the characters of a language is to store the dotmatrix (or line segments) for every character in the language (as done for English characters in the existing computing systems). For example, we can store the 247 dotmatrices for the Tamil characters and then display the appropriate matrix corresponding to the

character. This approach is simple to implement but will require a large memory.

A close look at the Tamil characters reveals that it is possible to generate the 247 characters by storing the dotmatrices for the basic symbols (refer to figure 3.2.) only. The characters in column 1, 7, 8, and 10 to 12 could be generated by storing the dotmatrices for characters in column 1 and the symbols ெ , ா and ◌ . In other words, the characters கெ (KE) and கொ (KO) could be generated by juxtaposing the symbols க , ◌ and ெ . Therefore, it is sufficient to store the symbols க , ◌ and ெ only. The BLS text generator, suitable for the Indian languages, could be realized as an hierarchical model as shown in figure 4.2.

Device Primitives: Each basic symbol could be realized by using the device primitives such as (1) pen up, pen down and draw a line if the device is a plotter, (2) beam on, beam off and draw a line of point (dot) if the device is a display of (3) activation of appropriate pins if the device is a dotmatrix printer.

Symbol Generator: There could be many methods to generate the symbols of a language. The following three methods are suitable for symbol generation: (1) line segment method, (2) linguistic method and (3) dotmatrix method. These techniques are explained later in this chapter.

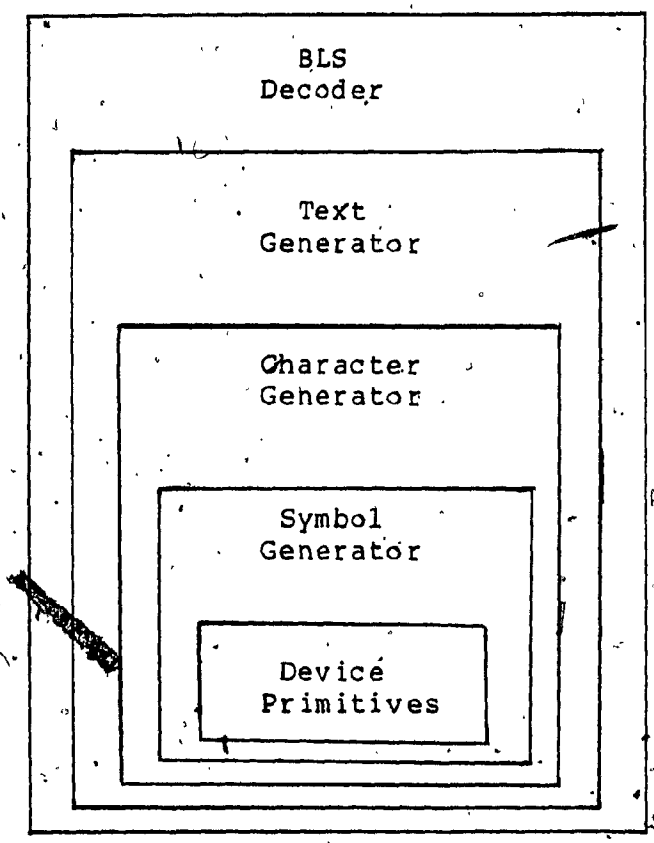


Fig. 4.2 Hierarchical Model of a Text Generator from BLS Code

Character Generator: Appending of symbols, to get the required character, is the task of a character generator. There will be one character generator for each language, more precisely for each script. Sometimes it is possible to construct a single character generator for a set of languages having the same script. For example, Hindi, Sanskrit and Marathi have the same script and hence it is sufficient to have a single character generator for these languages. Sometime it is also possible to have one character generator for a set of languages having a majority of their characters in common. For example, Bengali, Assamese and Manipuri have the same characters except for a difference of six characters, they could be grouped together. Similarly, the Latin script languages discussed earlier could be grouped together when the number of diacritical marks is not large.

The task of a character generator depends on the type of internal coding scheme adopted, whether symbolic or BLS. In the symbolic coding one or more symbols constitutes a character. Therefore, the character generator should generate a character using the set of consecutive symbols constituting it. Here, it is possible to have illegal combinations of symbols which do not constitute a valid character in a language, a unique situation which will not be encountered with the existing computing systems (handling English). Therefore, the character generator should be

capable of detecting such illegal combinations and printing a character indicating the illegal combination. If the BLS coding scheme is adopted, then the BLS decoder should be capable of decoding the text.

Text Generator: A text generator is concerned with such details as language code, intercharacter gap, carriage return, line feed, erase, back space, clear page, etc.

BLS Decoder: The decoder accepts input either from the keyboard or from the central processing unit, analyses it, extracts the characters and calls the text generator to generate them. The valid input sequences could be represented by a set of syntactic rules as follow:

<input> ::= <consonant-vowel> | <consonant> | <vowel> |

<special symbol> | <special characters>

<consonant-vowel> ::= <consonant> <vowel>

<consonant> ::= க் | ங் | ச் | ஞ் | ... | ன்

<vowel> ::= அ | ஆ | இ | ஈ | ... | ஔ

<special characters> ::= ஃ

<special symbol> ::= * | $ | linefeed | ...

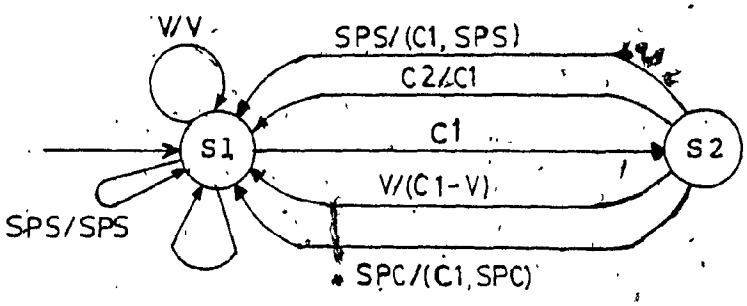
Figure 4.3. gives a two state machine for the above grammar.

A three state machine for the BLS decoder for languages allowing 'consonant-consonat-vowel' type conjunct characters is shown in figure 4.4. The number of states will increase if the language permits conjunct characters with three or more consonants with a vowel.

The decoder could be realized using well-known parsing techniques. Table driven parsing techniques have the particular advantage that by changing the contents of the table, we could generate text of different languages without changing the parser.

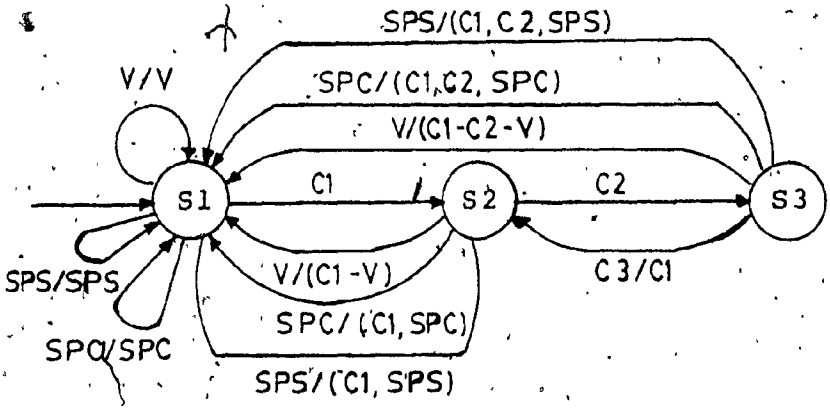
4.2.2 An Approach Based on Orthography of Characters

The character generation method explained in this section is the outcome of our analysis of the orthography of Tamil characters. It has been stated in Chapter 3 that when a vowel combines with a consonant to generate a composite character, the vowel assumes a specific grapheme called the 'sign' of the vowel, in the resulting character. A sign may consists of zero or more symbols. In Tamil, we have found that the sign of a vowel has at most two symbols (example, the sign of ஒ (O)). The symbols constituing a sign may appear before or after the symbol of a consonant (example,



V : Vowel
C1, C2 : Consonant
SPC : Special Character
SPS : Special Symbol

Fig. 4.3 Two State Machine for Generating Tamil Text



V : Vowel
C1, C2, C3 : Consonant
SPC : Special Character
SPS : Special Symbol

Fig. 4.4 Three State Machine for Generating Text for Languages allowing C-C-V Characters

க (KA), கா (KĀ), கெ (KE) and கொ (KO)). It is also possible that the symbol of a sign may get appended with the consonant symbol (example, கி (KI) and கீ (KĪ)).

There are exceptions to the rule of vowel sign. As an illustration consider the composite characters generated by the vowel ஆ (Ā). The sign of this vowel is ா and it appears after the consonant symbol. The exceptions are the characters ண (ṆĀ), ரு (ṚĀ) and ன (NĀ). It is also found that certain vowels (example, உ (U) and ஊ (Ū)), do not have any specific symbol as a sign. Therefore, the generation of the composite characters of உ (U) and ஊ (Ū) is treated as a special case. Figure 4.5 gives the sign of various vowels. It also indicates the position of appearance of the symbols with respect to the consonant symbol. That is, symbols in the first column appear before the consonant symbol and symbols in the second column appear after the consonant symbol. Absence of either the sign or symbol in a position is indicated as blank entry.

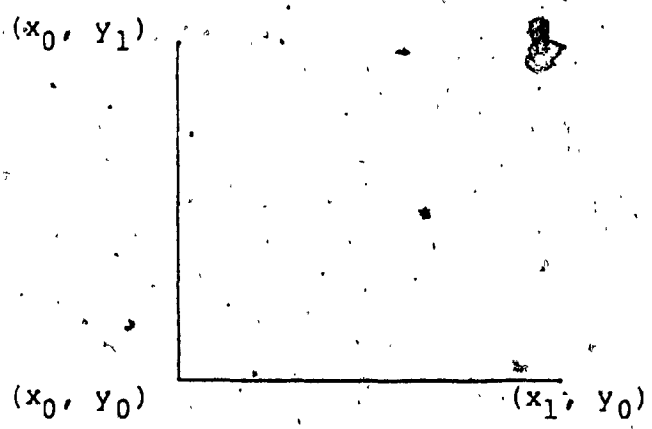
For character generation purposes, the characters are grouped into four categories:

1. special characters such as ஃ (AKH)
2. vowels
3. consonants
4. composite characters.

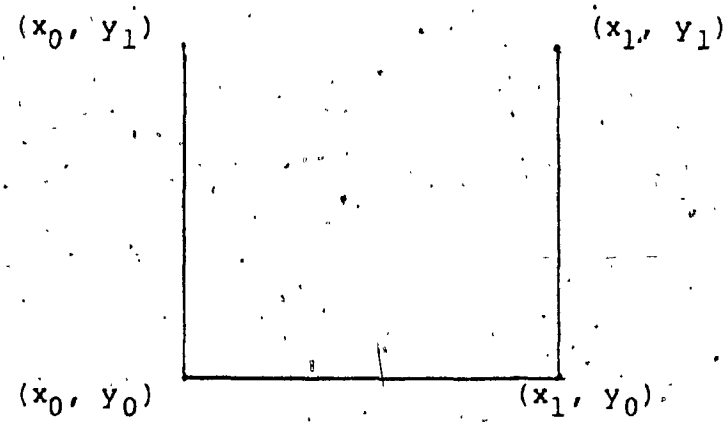
Vowel (Symbols)	Vowel Sign		Consonant	
	Symbol-1	Symbol-2	Symbol-1	Symbol-2
அ			க	•
ஆ		ா ணை, ை *	ங	•
இ		ி ு *	ச	•
ஈ		ீ ூ *	ஞ	•
உ			ட	•
ஊ			ண	•
எ	ெ		த	•
ஏ	ே		ந	•
ஐ	ை	ணை, ுை * ணை, ுவ	ப	•
ஒ	ொ	ா	ம	•
ஓ	ோ	ா	ய	•
ஔ *	ொ	ள	ர	•
			ல	•
			வ	•
			ழ	•
			ள	•
			ற	•
			ன	•

* Exceptional characters

Fig. 4.5 Signs and Symbol Structure for Tamil Characters



(a) Tamil Character ㄴ (TA)



(b) Tamil Character ㅂ (PA)

Fig. 4.8 Line segments for ㄴ (TA) and ㅂ (PA)

sequentially from the first until the last, corresponding to that symbol. Figure 4.9 and figure 3.3 give the 'hard' copy output of the set of Tamil, English and Malayalam symbols generated using this method.

This method is simple and straightforward as compared to the linguistic method (to be discussed next), but requires more memory space. The generative information expressed as above could be easily converted for a plotter by simply interpreting BN as pen down and BF as pen up which are the primitive actions for a plotter.

4.4.2 Linguistic Method

This approach, well suited to a language with a large number of characters, would use a set of picture primitives like straight lines, curves and loops, from which any character of the language could be generated. Primitives could be linked to form intermediate picture patterns, which could be linked with primitives and/or intermediate picture patterns to result in another intermediate picture pattern, until the complete character is generated.

Each primitive would be associated with a set of attributes such as length and thickness. They and the intermediate picture patterns would have in them distinguished picture points called vertices, identifed by

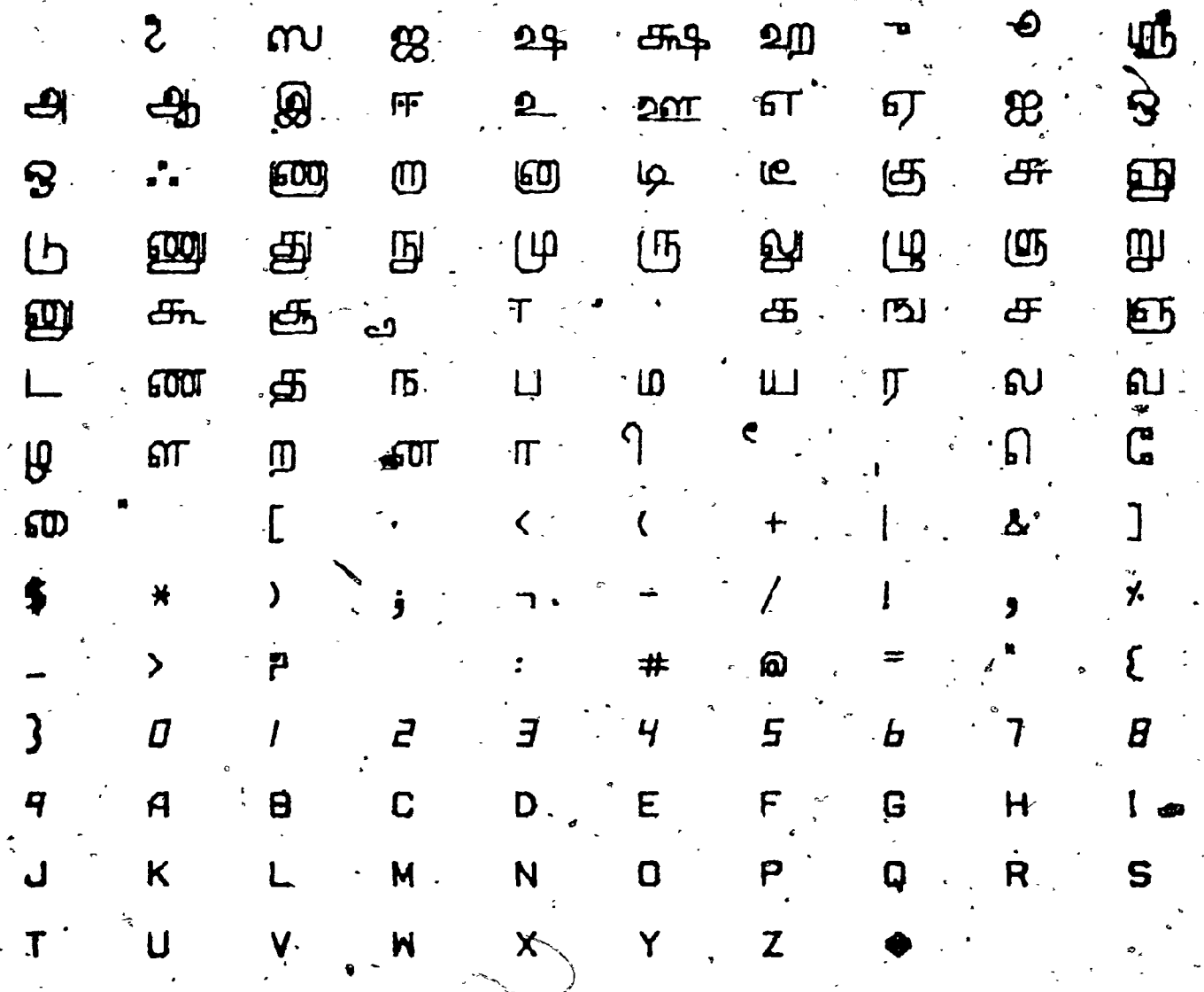


Fig. 4.9 List of Tamil, English, and Special Symbols Generated by the Line Segment Method

ordinals 1, 2, 3, etc., at which the primitive or intermediate patterns can be linked to form intermediate picture patterns. The vertex set of an intermediate pattern is a specific subset of the union of vertex sets of its components. As an illustration consider figure 4.10. By linking the primitives 'v' and 'h' we get the picture ⌊ which is the Tamil character TA; further by linking the primitive 'v', we get the Tamil character ⌴ (PA).

By repeatedly applying rewriting rules, which specify the method of linking the patterns to obtain intermediate patterns, we can generate any character of the language. Just as a generative grammar can generate the sentences of the language, a generative grammar for handprinted English characters has been devised by Narasimhan and Reddy [13]. If this grammar model were directly applied to Indian characters, however, the set of rewriting rules would be very large since phrase names for all characters of the language have to be given, and because of complexity of the characters, the number of intermediate phrases would be large. Further, such a straightforward model would not take into account the similarities in the structural derivation of composite characters.

The grammar model of Narasimhan and Reddy has been modified to overcome the above drawbacks and simulated using a Tektronics display as the output medium. The primitives

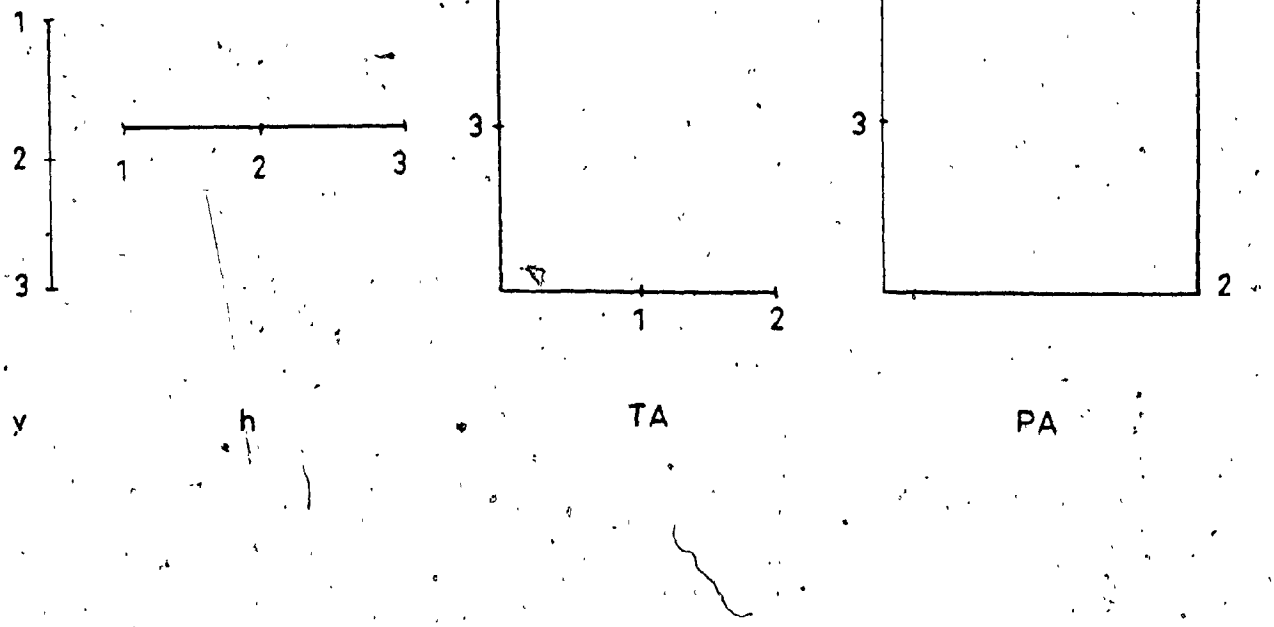


Fig. 4.10 Pattern Interconnection

of this model are realized in terms of the primitive actions of the display device, which in this case are drawing straight line segments between a pair of points with beam on and beam off conditions. The primitives and intermediate patterns are associated with a start point and an end point. Also they have attributes, such as magnification and rotation. The modified rewriting rules of the grammar are of the form:

$$IP \text{ ---> } IP_1(M_1,R_1) + IP_2(M_2,R_2) + \ldots + IP(M_n,R_n)$$

where IP is the name of the intermediate pattern defined. Each IP_j, $j = 1, 2, \ldots n$ is the name of the intermediate picture pattern or primitive. The symbols M_j and R_j correspond to the attribute values of the picture pattern IP_j. M_j is the magnification factor and R_j is the rotation if IP_j is an intermediate picture pattern; if IP_j is a primitive (BN or BF), then M_j and R_j correspond to the relative x and y coordinate values. The symbol '+' separates the various components in the rule. The intermediate picture pattern named IP is generated by generating the intermediate patterns and primitives, specified by the right hand side of the rewriting rule, in sequence from left to right. The primitive or intermediate pattern IP_{j+1} is attached to the pattern derived up to that point, such that the starting point of the pattern

IP_{j+1} coincides with the end point of the intermediate pattern. The end point of the pattern derived up to the (j+1)st component is the same as that of the (j+1)st component. The phrase generator starts with a given phrase and reduces the intermediate phrases and primitives in the rewriting rule, corresponding to the phrase name, to a sequence of primitive actions of the display and generates the phrase.

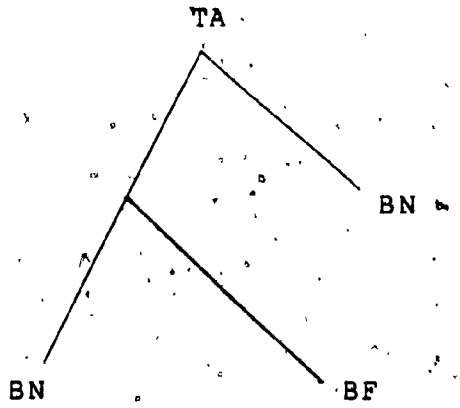
As an illustration, consider the generation of symbols ⌞ (TA) and ⌴ (PA) used in the line segment method. The same two symbols are selected to demonstrate the differences between the line segment method and the linguistic method.

(⌞) TA -----> $BF(x_1,y_1)$ + $BN(x_0,y_0)$ + $BN(x_1,y_0)$

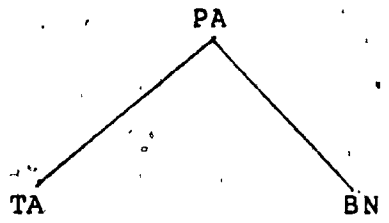
(⌴) PA -----> TA(1,0) + $BN(x_1,y_1)$

where TA(1,0) indicates that TA is the intermediate picture pattern with no magnification (magnification factor is 1) and no rotation (angle of rotation is 0) from the phrase name TA defined in the earlier line. The generation procedure of a phrase can be represented as a tree as shown in figure 4.11 for TA and PA.

A comparison of the generative procedures of ⌞ (TA) and ⌴ (PA) using the line segment method and the linguistic method clearly shows the saving in space by the



(a) Derivation Tree for $\llcorner$(TA)



(b) Derivation Tree for $\sqcup$(PA)

Fig. 4.11 Derivation Tree for $\llcorner$(TA) and $\sqcup$(PA)

later method. Of course, the linguistic method will take more computer time for generation and the program will be more complicated (hence taking more space for the program) than in the line segment method.

4.4.3 Dotmatrix Method

This method differs from the previous two methods in the sense that each symbol is represented as a matrix of dots instead of line segments. Dot matrix type displays and printers are available for Roman characters. In these devices a fixed size say 9 x 7, dot matrix for each Roman character is assumed. Such an assumption is not valid for non-English language characters because they vary in height as well as in width. Also providing a constant size for line and symbol will result in unequal intercharacter gaps and interline spaces. Therefore, in the multilingual display or printer, the intercharacter gap and interline gap must be variable. A desirable display could treat the entire page (screen) as a matrix of dots to allow fixed intercharacter and interline spaces. A set of symbols generated using the dotmatrix method is shown in figure 3.2.

The dot matrix printing could be of three different types.

1. Column matrix in which the symbol is printed column by column taking more than one print stroke per symbol and hence per character but having less complicated hardware. The nine pins (one per dot) per column available with the existing printers are too small to print multilingual texts.

2. The dot matrix could be of one symbol size. Unlike column printers, this could print one symbol in one stroke. The size of the dot matrix should be such that the largest symbol in any language could be accommodated in it. The estimated size of the matrix is 15 x 17.

3. A line matrix, with which we can print the entire line in one stroke, will be expensive. It could be used for high speed line printers.

4.5 ROM Storage for Symbols

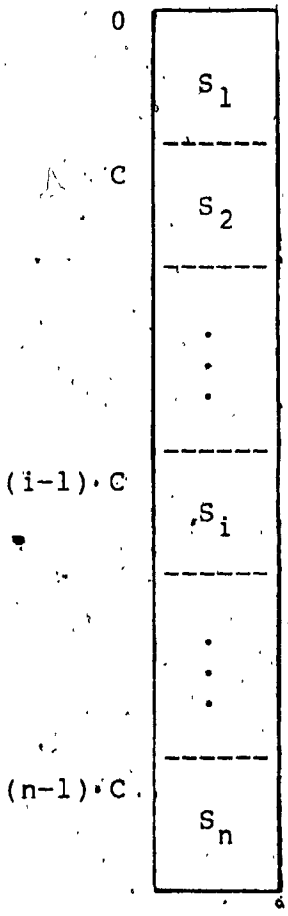
4.5.1 Data Structure for ROM Storage

The dot matrices for various symbols could be stored in a read only memory (ROM) in many different ways. Depending on the data structure chosen, the program to implement the symbol generator will vary. Two possible methods are discussed below:

Data Structure-1: The symbols vary in size. One way to code the symbols is to assume a constant dot matrix size for all symbols equal to the size of the largest symbol and store them. The advantage of this method is that the starting address of a symbol could be obtained easily as shown in figure 4.12.

Data Structure-2: It is possible to reduce the storage requirement by storing the actual dot matrix size of each symbol instead of one large uniform size. Then it is necessary to store the size of a symbol as it varies from symbol to symbol. This organization is shown in figure 4.13. It is possible that the table of pointers to symbols could be avoided by incorporating them as an integral part of the symbol generator program because the starting addresses of the symbols are known once they are stored in the ROM.

Shift Information: As stated earlier, sometimes it is necessary to append a symbol either with the preceding or the succeeding symbol to get a character. For example, the symbol ◌ி is to be appended to the preceding symbol க in order to get the character கி (KI). The symbol ◌ை is to be appended to the succeeding symbol ன , to get னை (NAI). It is also clear from the above example that the number of columns to be shifted to append a symbol (zero for னை (NAI) and non-zero for கி (KI)) varies. Therefore, it



The start address of i^{th} symbol = (i-1)*C .

where C is the dot matrix size of a symbol.

Fig. 4.12 Data Structure-1 for ROM Storage

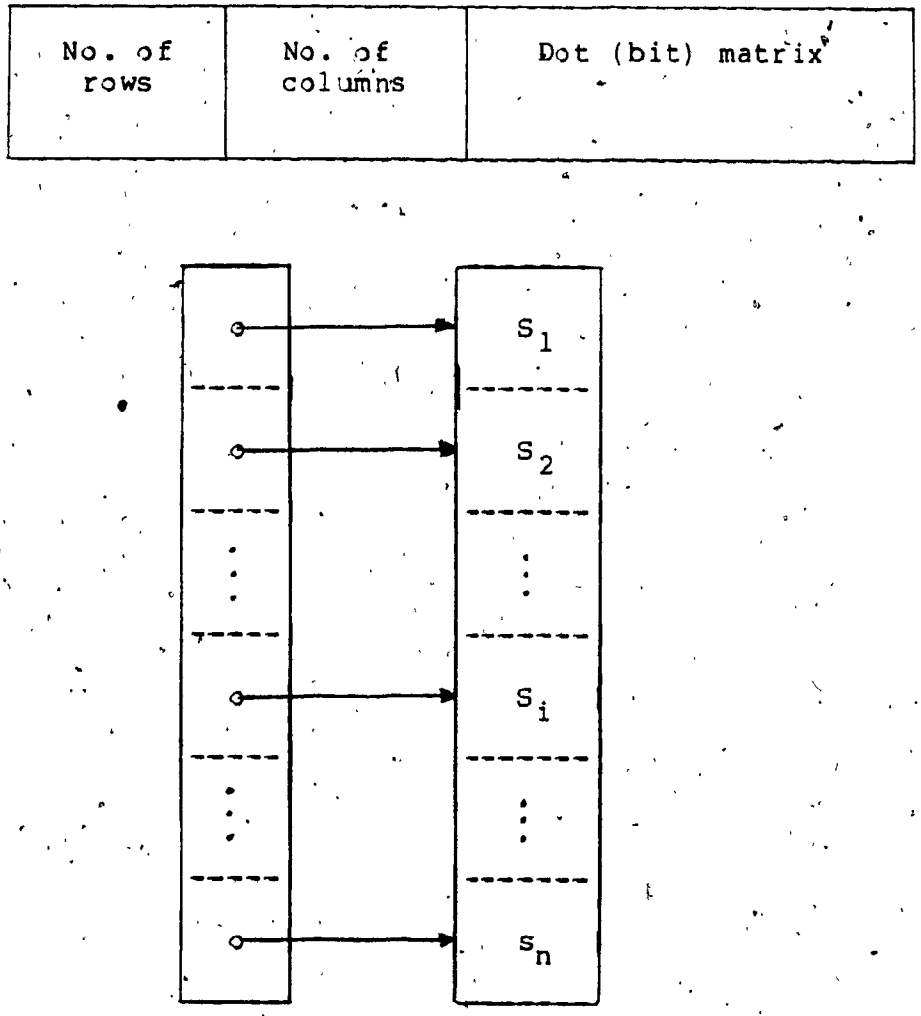


Fig. 4.13 Data Structure-2 for ROM Storage

is necessary to incorporate such shift information for the various symbols. Fortunately, the amount to be shifted is the same for most of the symbols. Therefore, we can make this constant the standard value by default and specify in a table only those requiring the non-standard values as shown in Table III.

4.5.2 Storage Minimization and Selection of Primitives

One of the challenges of the linguistic method is the selection of primitives. There are two extreme cases possible in the selection of picture primitives for generating a class of pictures, namely (1) select a maximum number of primitives and (2) select a minimum number of primitives. In the first case, a large amount of storage would be required to store all the primitives. However, the number of rewriting rules would be small and the computation time for interconnecting the primitives to generate the picture elements would be small. In the second case, the storage required for primitives would be smaller, but the number of rewriting rules would be large. Also the computation time for interconnection of picture primitives would be high. The primitives should be selected such that an acceptable compromise is achieved between the storage and computation time requirements. In the case where the set of primitives is the same as the symbol set to be generated,

Table III Shift Information

Sample character	Symbol	Backward shift before print/display	Forward shift after print/display
கி	ி	7	4
கீ	ீ	9	4
பு	ு	4	4
பூ	ூ	9	4
மூ	ூ	21	18
க்	்	11	9
தா	ா	4	4
ணை	ை	4	0

Note: Intersymbol and intercharacter gap = 4 dots.

the syntactic picture generation reduces to triviality. Unfortunately no formal methods are available to select the set of primitives for a specified class of pictures. Symbols must be carefully studied and then decomposed into components. The decomposition is to be done in such a way that the components are few in number and that they occur in many symbols. This process has to be iterative. A decomposition for Tamil symbols is shown in figure 4.14.

Optimization: Let us consider the dot matrices for the symbols க (KA) and க் (CA) in figure 4.15. Represent these dot matrices as a series of numbers as in figure 4.16, where each number represents a vertical column and is obtained by treating the dots in each column as a binary number of fixed size. Comparison of the two strings reveals that there is a common substring which corresponds to the common graphic pattern between க (KA) and க் (CA). Suppose,

SS_1 = <0036, 0041, 0041, 1741, 1041, 1041, 1041, 1776>

SS_2 = <1041, 1041, 1036>

SS_3 = <1040, 1040, 1040>

now, க (KA) and க் (CA) could be written by the production rules:

க (KA) ---> $SS_1 + SS_2$

க் (CA) ---> $SS_1 + SS_3$

where + is used as the linear concatenation operator. SS_1,

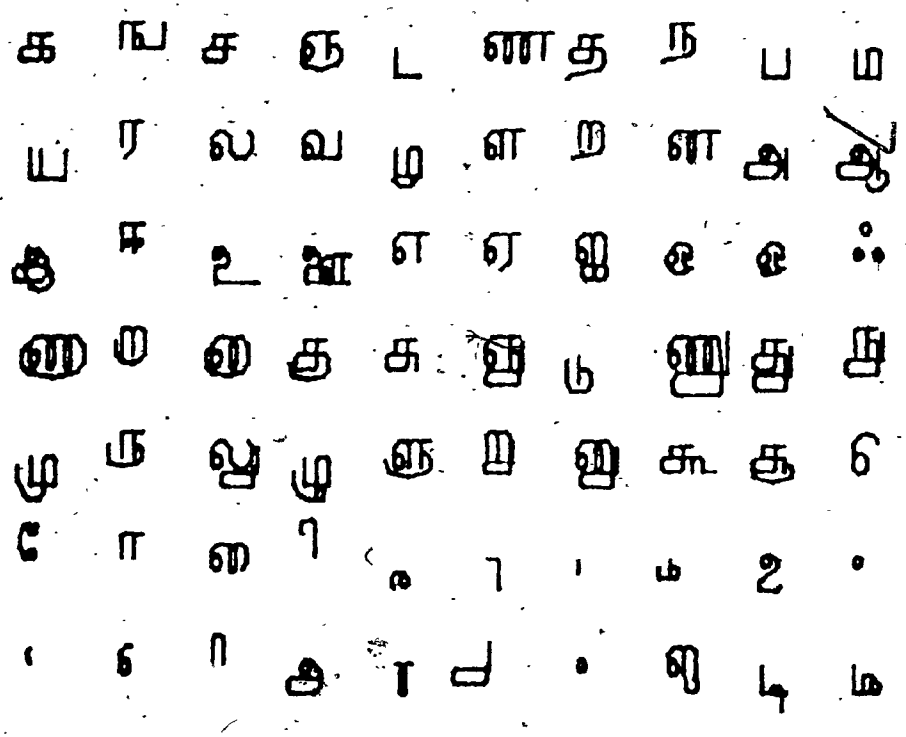
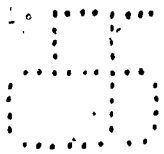
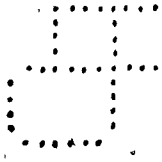


Fig. 4.14 Set of Primitives for generation of Tamil Characters by the Linguistic Method



(a) Dot matrix of ಕ (KA)



(b) Dot Matrix of ಕ್ (CA)

Fig. 4.15 Dot Matrices for ಕ (KA) and ಕ್ (CA)

0036, 0041, 0041, 1741, 1041, 1041, 1041, 1776, 1041, 1041, 1036

(a) Sequence of Numbers for ಕ (KA) in Octal

0036, 0041, 0041, 1741, 1041, 1041, 1041, 1776, 1040, 1041, 1040

(b) Sequence of Numbers for ಕ್ (CA) in Octal

Note: Dot matrix size is 10x10. Coding is column by column considering the bottom-most dot as the least significant digit.

Fig. 4.16 Numerical Representation of ಕ (KA) and ಕ್ (CA)

SS_2, and SS_3 are the picture primitives in the linguistic method and (KA) and (CA) are the production rules.

We could generalize this method for a class of m symbols. Suppose S_i denotes the sequence of integers corresponding to the symbol i. We define the following sets:

$$S = \{S_1, S_2, \ldots, S_m\}$$

$$SS = \{SS_1, SS_2, \ldots, SS_n\}$$

The set SS is constructed such that every SS_i is a subsequence of integers contained in at least one S_j; and every S_i can be constructed as a linear concatenation of one or more SS_js. Let PR_i denote the production rule for the symbol S_i and M(k) denote the storage for the item k. Then,

$$M(S) = \sum_{i=1}^{m} M(S_i)$$

$$M(SS) = \sum_{i=1}^{n} M(SS_i)$$

$$M(PR) = \sum_{i=1}^{m} M(PR_i)$$

If M(S) is much greater than M(SS) + M(PR), we have reduced the storage requirement by an amount equal to the difference between them. Our objective then is to maximize

$$M(S) - [M(SS) + M(PR)]$$

Since M(S) is fixed for given set of symbols, we try to minimize M(SS) without unduly increasing M(PR).

As a further work in optimization, the following analysis has been done using the Tamil symbols shown in figure 3.2.

s = { Tamil symbols shown in figure 3.2 }

SS = { Unique numbers (1-grams) in S sequences }

Represent the production rules, PRs for each symbol as

<k> (<index to SS>)^k

where k is the number of columns in a symbol and

(<index to SS>)^k means that <index to SS> is repeated k times. For Tamil symbols, it has been found that

m = 63

n = 270

M(S) = 2442 bytes

M(SS) = 810 bytes

M(PR) = 916 bytes

M(S) - [M(SS) + M(PR)] = 716 bytes

By this method there is 29% savings in memory.

We have observed that many elements in SS appear only once in S. Therefore, by storing these values directly in the PRs in place of indices as

$$\langle k \rangle\ (\langle type \rangle\ \{ {\langle index\ to\ SS \rangle \atop \langle value \rangle} \})^k$$

where 'type' is a one bit flag which indicates whether the number following is an index to SS or the actual value of a column in the symbol, and { } stands for select either <index to SS> or <value>. In this case, SS contains only those sequence numbers appearing twice or more in S. By this method, there is 34% savings in memory.

4.6 Determination of Dot Matrix Size

For display using the dot matrix method, it is necessary to identify the minimum dot matrix size required to code the characters of different languages. This can be determined either manually or by automatic methods.

4.6.1 Manual Method

In this method, the symbols of each language are drawn on graph sheets assuming a particular dotmatrix size as shown in figure 4.17. Then, they are coded and displayed on a graphic display terminal. The coded symbols may or may not be optimal in size. Therefore, the sizes of various symbols are either reduced or enlarged as necessary and displayed. The aesthetic features and distinguishability are studied for different sizes. The smallest size that

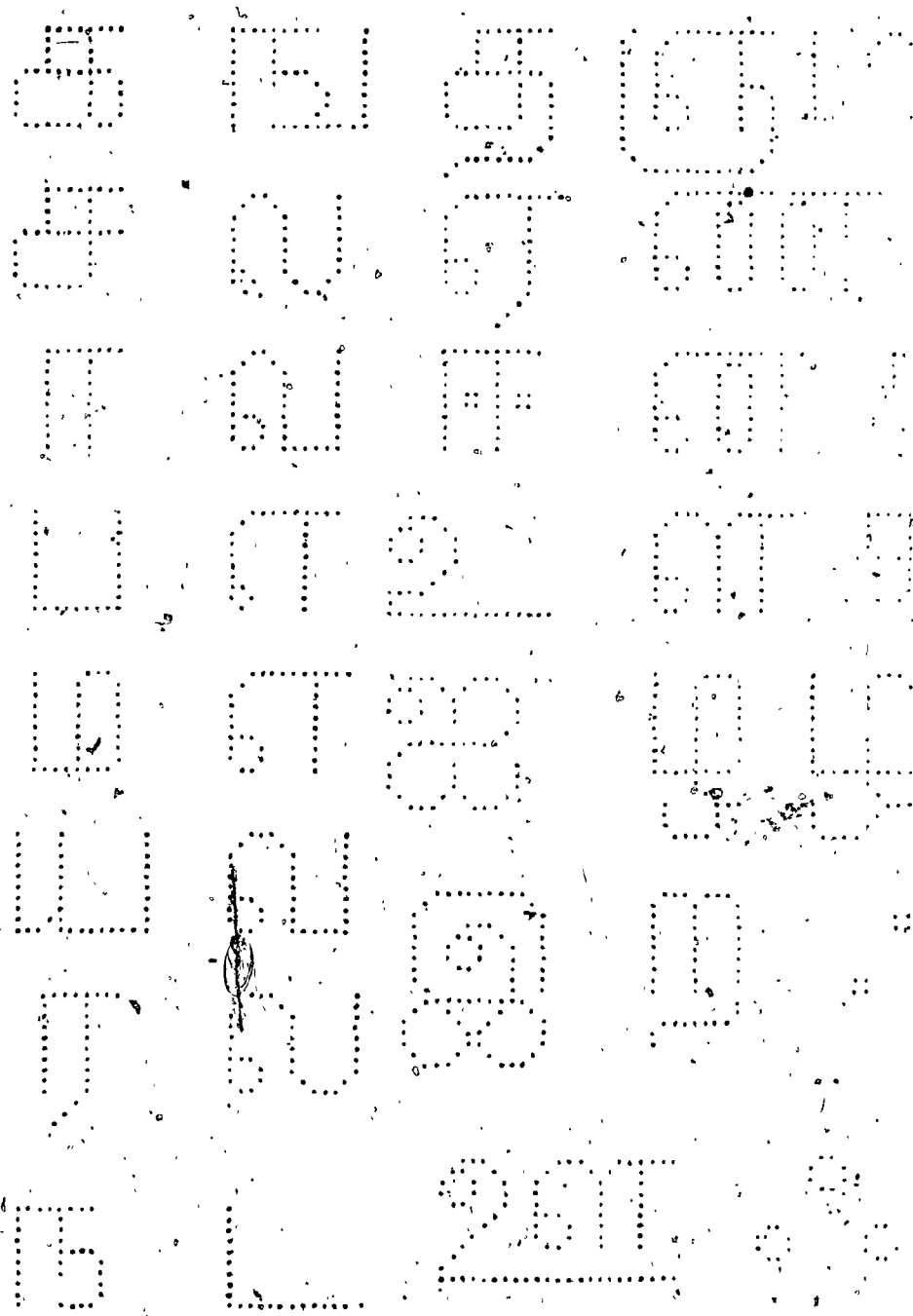


Fig. 4.17 Manually Coded Dot Matrices for Tamil Symbols

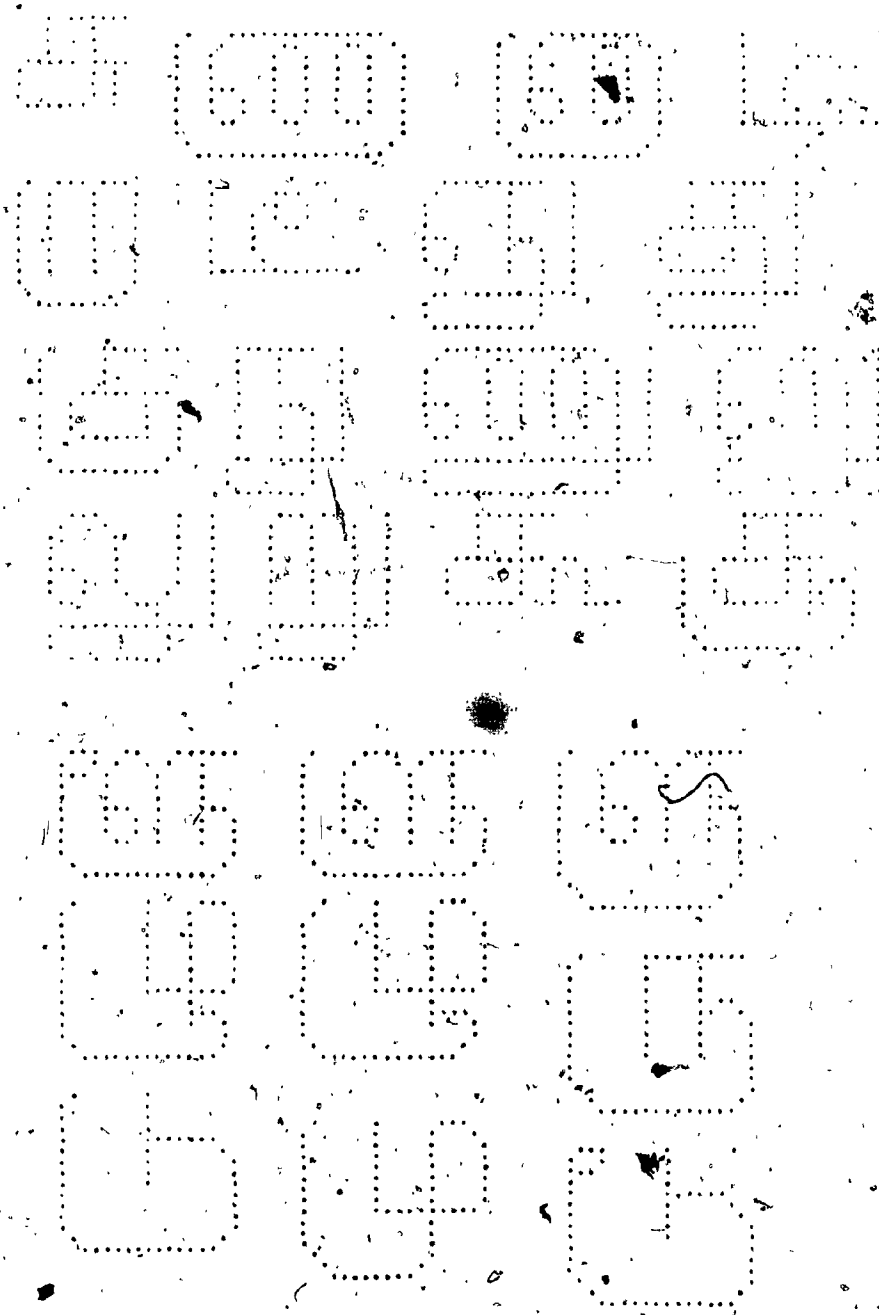


Fig. 4.17 (contd.)

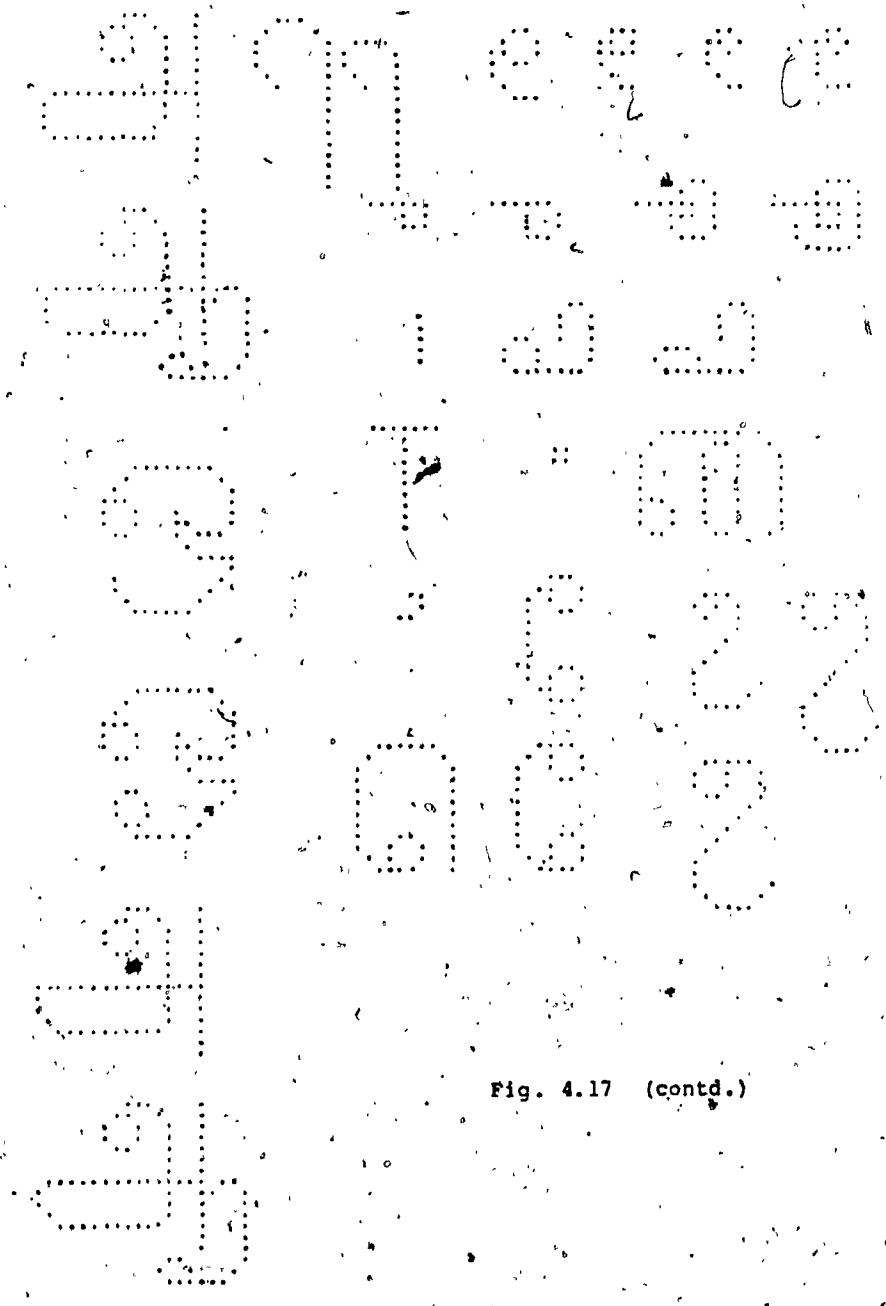


Fig. 4.17 (contd.)

could be clearly identified is selected as the size of the dot matrix.

Our experiments (refer to figure 4.18) with the Tamil symbols shown in figure 3.2 using the Tektronix display device reveal a reasonable size to be 15 x 17.

4.6.2 Automatic Method

An alternative to the manual method is the automatic method shown in figure 4.19. In this method instead of hand coding the symbols, the printed characters of a language under study are digitized (to a binary pattern) using a digital scanner. Then the skeleton of the digitized pattern is derived using a thinning algorithm. The thinned character is then analysed as in the mannual method to determine the optimal dot matrix size. Experiments have been conducted on a set of Tamil characters using this method. The result of the study on character ஓ (Ō) is shown in figures 4.20 through 4.22. The thinning algorithm used is that of Deutsch [64]. The dot matrix size for ஓ (Ō) by this method is found to be 12 x 11. The dot matrix size computed from this experiment to display any symbol is 14 x 16.

The Tektronix graphics terminal used in our experiments has a high resolution of 100 dots(points) per inch. However

DOT MATRIX SIZES

10,11	9,9	8,8	7,7	7,6
14,13	12,11	11,10	10,9	9,7
10,22	9,19	8,17	7,15	7,13

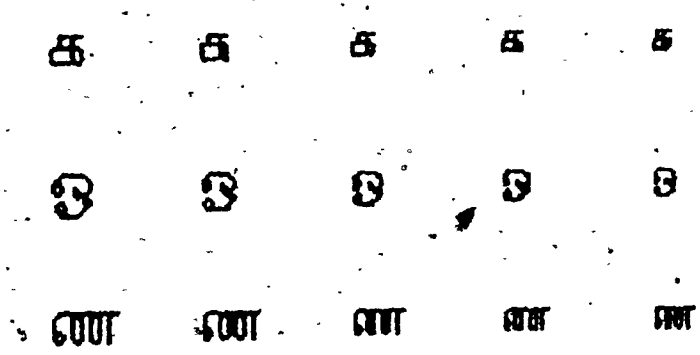


Fig. 4.18 Analysis of a Manually Coded Tamil Symbol

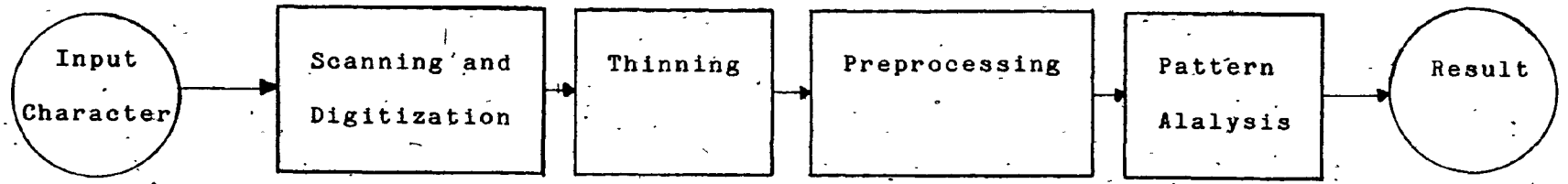


Fig. 4.19 Scheme for Automatic Method of Dot Matrix Size Determination

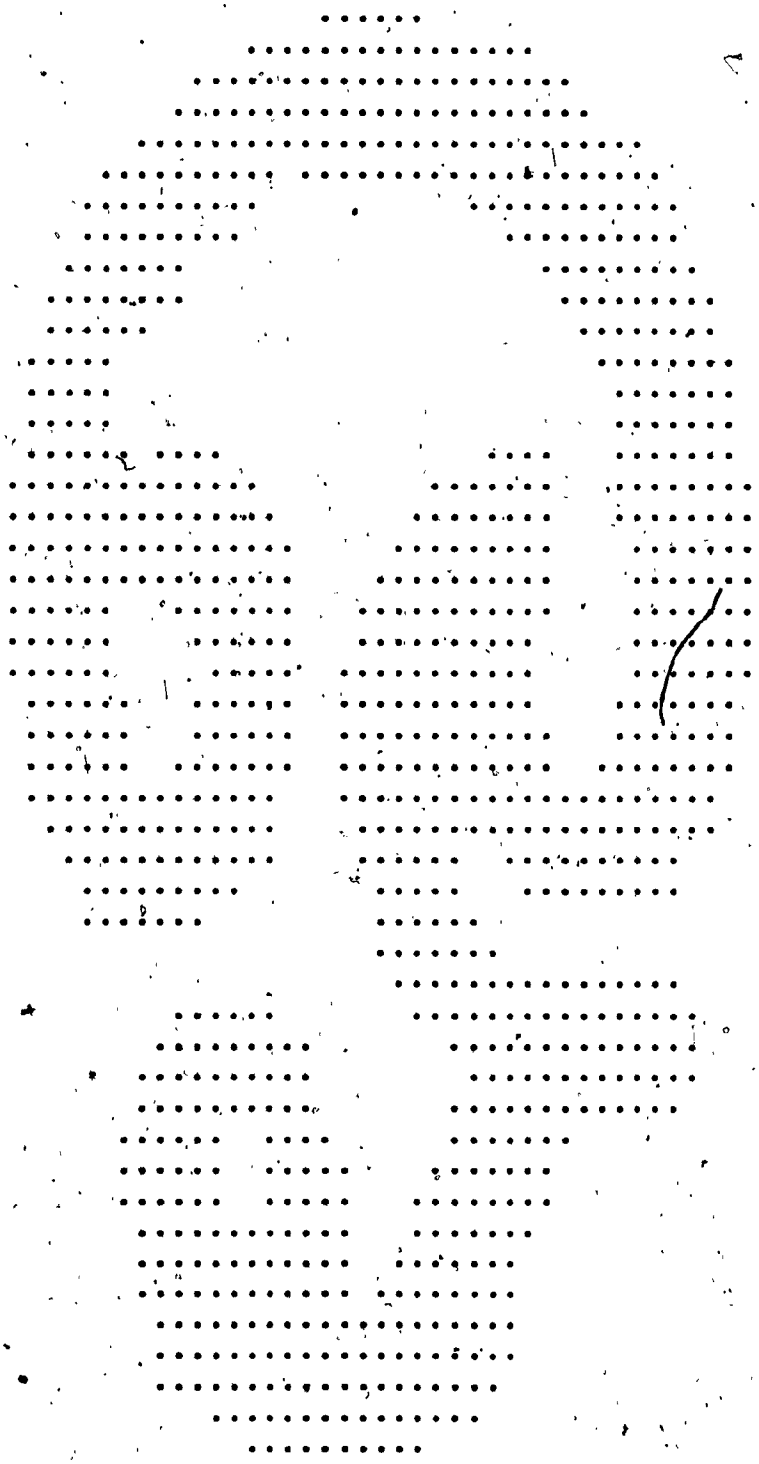


Fig. 4.20 Digitized Tamil Character ஓ (Ō)

Fig. 4.21 Thinned Tamil Character ஓ (ō)

DOT MATRIX SIZES

26,70	7,21	7,20	6,19	6,18	6,18	5,17	5,16	5,15	5,14
47,35	12,11	12,10	11,10	10,9	10,9	9,8	9,8	9,8	8,7
27,30	8,10	7,9	7,8	6,8	6,8	6,7	5,7	5,7	5,6

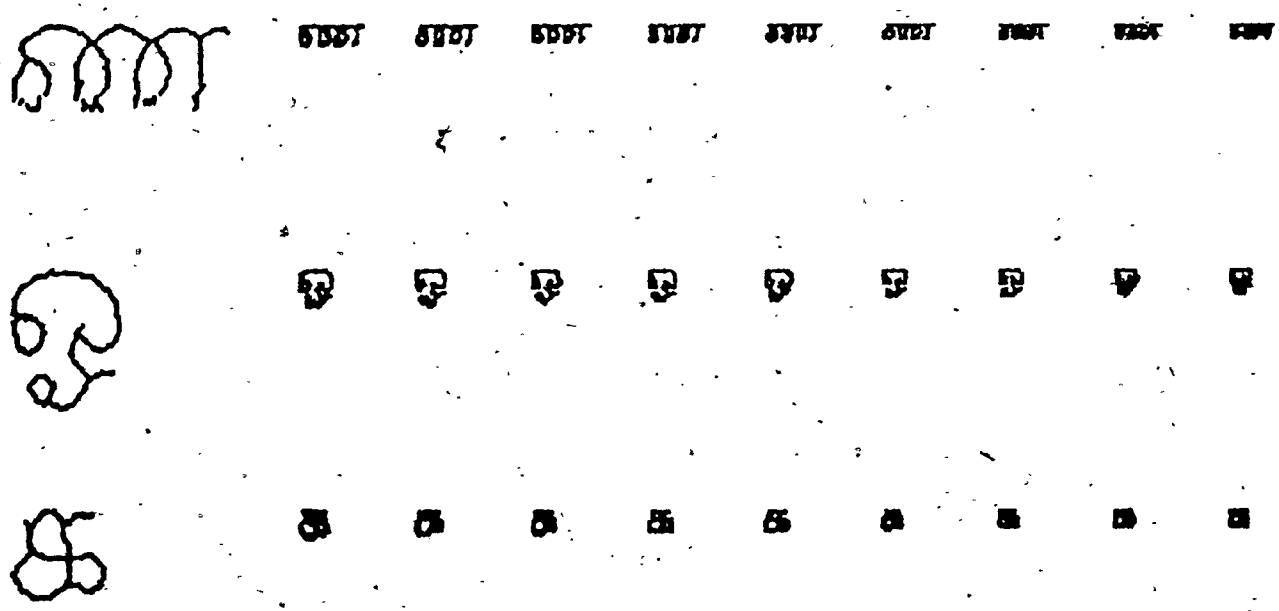


Fig. 4.22 Analysis of Thinned Characters

due to the hallow around dots on the screen, two consecutive dots appear together (touching) on the screen. In our expermiments, one point was left blank between two successive dots. The dot matrix sizes obtained are, thus, influenced by the view screen to a certain extent.

4.6.3 Scaling Algorithm

The following algorithm was used to scale down or scale up the characters studied under both the automatic and manual methods for size determination. The digitized picture pattern (character) is treated as binary matrix of size m rows and n columns.

m x n : the number of picture points in the pattern

s : scale factor (s is less than 1 for scaling down and greater than 1 for scaling up)

p_{ij} : value (dot or blank) of the picture point at the i^{th} row and j^{th} column in the unscaled picture pattern

sp_{kl} : value (dot or blank) of the picture point at the k^{th} row and l^{th} column in the scaled picture pattern.

Then, the picture points in the scaled picture are given by

$$sp_{kl} = p_{ij}$$

where $i = 1 \ldots m$, $j = 1 \ldots n$,

k = TRUNC(i * s) and l = TRUNC(j * s).

'TRUNC' is a function which gives the truncated value (nearest lower integer) given a real value. One may also employ rounding instead of truncation.

4.6.4 Comparison of the Methods

Both methods are suitable only for determination of the dot matrix size of the characters or symbols. The actual dot pattern (font design) needs to be done manually after determining the size. In this respect the thinned character from the automatic method could be easily modified to obtain the final pattern whereas the manual coding is difficult and time consuming. The automatic method will give better results compared to the manual method because the characters taken for analysis are the printed characters. The automatic method is convenient and faster compared to the manual method.

4.7 Character Fonts and Electronic Dot-Pattern Design Board

4.7.1 Font Design and Evaluation

Font Design: The parameters to be considered in designing dot matrix type characters are character size, luminance,

luminance contrast (modulation), aspect ratio, font, percent active area, dot size, dot shape, dot spacing and number of dots. Experimental results [65] available for English characters could be taken into consideration in designing dot matrix fonts for Indian characters.

Font Analysis: Within a given dot matrix size, it is possible to design characters with different fonts. Many fonts are available for English characters. Font styles make a great difference in legibility. Therefore, they need to be evaluated to identify the most suitable one(s). Also, a font design needs to be evaluated for its effectiveness before fabrication. Methodologies were developed for evaluating various fonts for upper case English characters. Questions still remain about optimal fonts for lower case letters, superscripts, subscripts, italics and other symbols [66].

Suen and Shiau [67] designed an iterative method of determining the most distinctive set of 5x7 dotmatrix English characters for computer output display systems. In their investigation eight quantitative measurements (similarity functions, hamming distances, linear correlation functions, cross correlation functions, information content, entropy, nearest-neighbor distance-1 and -2) were used in an iterative process to eliminate undesirable fonts and thus

determine the most distinctive set of fonts. In each stage of the iterative process, all eight quantitative measurements were computed and compared among different fonts of the same symbol.

A two-phase study was conducted by Maddox [68] which related the confusions among dot matrix English characters to the two-dimensional spatial frequency similarities of these characters. The purpose of this study was to evaluate the utility of various dot matrix fonts when displayed in common dot matrix sizes and to ascertain the effectiveness of the spatial frequency analysis model of the visual system in accounting for observed human performance.

These techniques could be successfully applied to evaluate various dot matrix fonts for Indian characters. However, this is beyond the scope of this thesis.

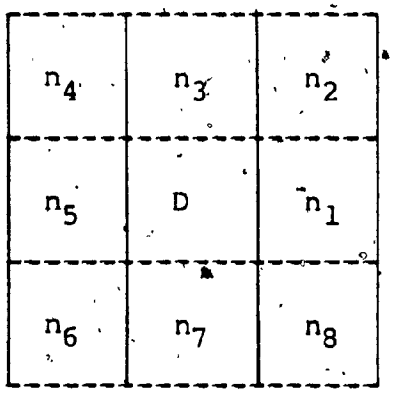
4.7.2 Electronic Dot Pattern Design Board

Font design is to a large extent a manual operation. Developing dot matrices for characters of many languages is a very long process, even for single font, if done manually. Coding hundreds of characters is not only painful but also time consuming. Even then one is not sure that the characters when printed or displayed will look as they do in the drawing. If they do not, then the data is to be changed

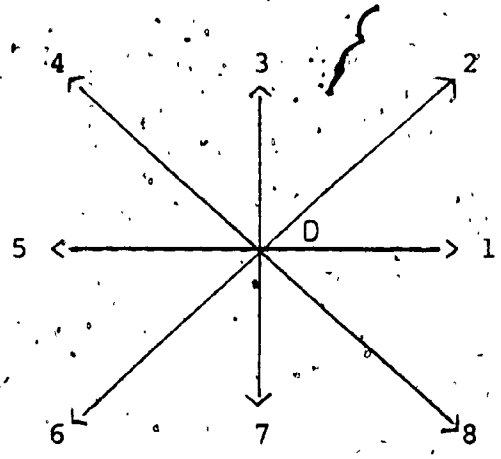
and tried again. This iterative off-line dot matrix design is a slow process. Clearly what is required is an "electronic dot pattern design board" where all trials could be conveniently made electronically (on-line), to arrive at a final font for the symbols. The design of one such drawing board is explained below.

An electronic dot pattern design board should provide the following facilities:

1. Declaration of a dot matrix of required size.
2. An identifiable cursor (single dot or a cross).
3. The possibility of moving (positioning) the cursor to any desired dot.
4. The possibility of ploting/deleting one or more dots in any of the eight principal neighbourhoods of a given dot. The eight neighbours of a dot, D, are shown in figure 4.23.
5. The possibility of storing the final or intermediate dot pattern in secondary memory.
6. The possibility of retrieving any character from the secondary memory and displaying it at any desired position in the dot matrix, with or without affecting the pattern already existing. This is necessary to identify whether or not the symbols get properly appended to another desired symbol. For example, consider the characters கி (KI), பி (PI) and



(a) Neighbours of the dot 'D'



(b) Direction Number for the Neighbours of D'

Fig. 4.23 Neighbours of the Dot 'D'

கி (MI) in Tamil. They are obtained by appending the symbol ி to க (KA), ப (PA) and ம (MA) respectively. In all the three cases it is necessary that the appending symbol ி should properly touch the main symbol. Obviously, the symbols க , ப , ம , ி designed independently, need to be checked for proper appending. This feature will also be useful for comparison of the optical and psychological effects of different fonts.

The following facilities are desirable, but not essential:

7. Automatic display of the command or instruction issued by the designer. This will enable the designer to make sure that the command issued is correct before making any changes to the pattern.
8. A facility to scale a given dot pattern up or down, in order to be able to study the optical and psychological effects of different sizes and the effect of scaling distortion.
9. The ability to store the present pattern as a "previous pattern" before executing a command; abort the present command and restore the pattern to the previous pattern if directed by the designer. This will enable the designer to get back the old pattern in case of an incorrect command or instruction, thus saving the time and effort of reconstruction.

In this design the display screen is divided vertically into two parts -- one for displaying the command/instruction and the other for the dot matrix as shown in figure 4.24. Commands are displayed one after another and one below the other. When a command is typed at the last display line, then the commands are moved up by one line, as with a normal computer display terminal, leaving the dot pattern unaffected (unshifted). The various commands are explained below:

MOVE CURSOR:

< MC, r,c >

The cursor will be positioned to the dot at row 'r' and column 'c'.

DECLARE MATRIX:

< DM, r,c >

This command will create a dot matrix having 'r' rows and 'c' columns. After creation the cursor will automatically be positioned to d11, the dot at row-1 and column-1. An error message will be printed if the size declared cannot be accommodated with the available memory.

DRAW DOTS:

< DD, n,d >

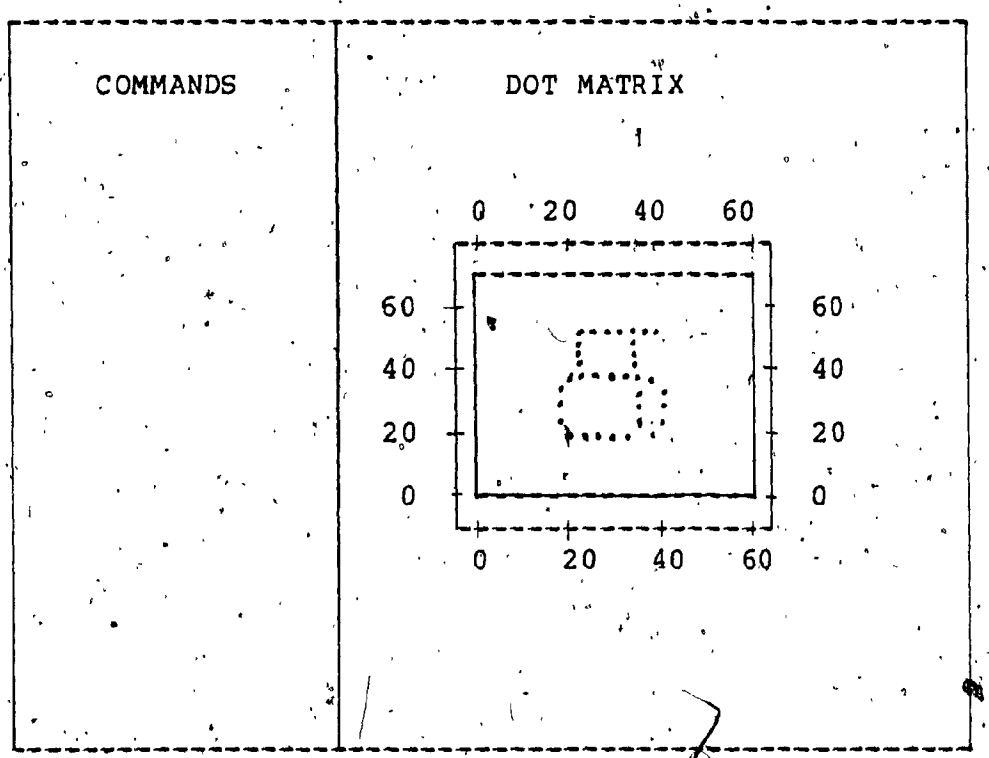


Fig. 4.24 Dot Pattern Design Board

This will draw 'd' dots in the neighbourhood 'n' from the present cursor position. If the cursor reaches the boundary before drawing 'd' dots, then the operation will be terminated and the cursor will be positioned at the boundary. The character 'B' will be displayed at the end of the operation against the command to indicate that the boundary condition has occured.

DRAW SPACES:

< DS, n,d >

This command is identical to 'draw dots' with the difference that spaces will be left instead of dots. If a dot already exists, it will be made into a space.

STORE PATTERN:

Format-1	< SP, id >
Format-2	< SP, id,sr,sc,fr,fc >
Format-3	< SP, id,sr,sc >

Three formats are possible here. Format-1 stores the entire dot matrix in the secondary memory with the identification, 'id'. Format-2 stores a partial matrix starting from row 'sr' and column 'sc' to row 'fr' and column 'fc'. If 'fr' and 'fc' are omitted (Format-3), then they are assumed to be the maximum values declared.

DISPLAY PATTERN:

Format-1	< DP >
Format-2	< DP, id >
Format-3	< DP, id,sr,sc >

Format-1 will clear the dot matrix. Format-2 will cause the pattern 'id' will be retrieved from secondary storage. The entire dot matrix will be cleared, and the pattern will be displayed starting from row-1 and column-1. Format-3 will retrieve the pattern id, and display it starting from row 'sr' and column 'sc'. The rest of the dots will remain unaffected.

SCALE PATTERN:

< S, fc >

This command will scale down (fc is less than 1) or scale up (fc greater than 1) the character under display on the screen, by the amount (fc) specified, and then display the scaled character.

5. APPLICATIONS OF A TEXTWRITER

The textwriter is a system for the generation, production and editing of texts written possibly in more than one script. Input/output, coding, and programming facilities are the basic requirements for such a system. In addition, an individual application may pose its own problems. To get an insight into the problems in a multilingual environment the following applications are considered in detail:

1. A Multilingual Telex System
2. Data Acquisition/Database Applications
3. Automatic Transliteration
4. Teaching Scripts and Languages.

5.1 Multilingual Telex System [69]

With the modern development in science and technology, there are more messages exchanged today than in the past. Also the centers of rural areas have become sources or destinations of such message transfers. Message transfers in India cross the linguistically-oriented state boundaries. Automatic machine translation of messages from one language to another is desirable, but it is still a research problem.

However it may be remarked that machine translation in a limited and well defined context, such as the weather report [70], has been achieved with a good degree of success. In the case of a telex message which consist of three basic components, receiver's name and address, message, and sender's name, the following may be considered:

(a) Translation of a message from language L1 to another language L2 which is limited to a well defined context. For example, words like 'STREET' need to be translated.

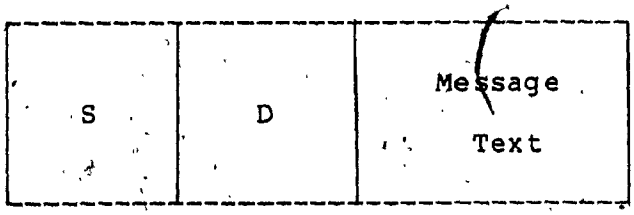
(b) Transliteration or rewriting of the components of a message written in the script of language L1 to that in L2.

Translation of general texts is difficult, but transliteration and translation of bounded scope are possible.

A message format for the proposed multilingual telex system is shown in figure 5.1. Each component of a message may follow the format:

<language code, text>

The text may be encoded using either the symbolic coding or the BLS coding. The selection of either of the coding schemes could be based on the average number of bytes of information to be transmitted per character. Suppose a text in a language L1 is α-coded with the set of its αs (say, P1) and S in figure 5.1 is set to indicate L1. Then, if D is different from S, say L2, either translation or



S : Source Language Indicator

D : Destination Language Indicator

Fig. 5.1 Telex Message Format

transliteration is required. To accomplish such a translation/transliteration, a mapping is required from P1 to P2. Two Indian languages Tamil and Malayalam were considered to study the complexity of this mapping. For example, the Tamil word மணி (MANI) is transliterated into Malayalam word മണി (MANI) by means of the partial mapping shown in Table IV. This process is not trivial when the complete mapping is considered. One has to consider the ways to handle one-to-many and many-to-one mappings between the scripts and their dependencies on context. Automatic transliteration of text from one Indian language to another is discussed in detail in section 5.3.

An architecture to support the multilingual telex operation is given in figure 5.2. It consists of a textwriter, read only memories (ROMs), random access memories (RAMs), a mass storage (back-up storage) and a modem. In addition, there is a code interpreter to perform translation of bounded scope and transliteration of the telex messages.

The keyboard of the teleprinter (textwriter) could be either symbol-based or BLS-based. ROMs are used for the storage of the monitor program, the software for the communication protocol, code interpreter and character generators for the printers. The dot-matrices for the symbols of different languages could be either stored

Table IV Partial Mapping of Tamil and Malayalam Characters

Tamil		Malayalam	
Character	Code	Character	Code
ம ணி	ம் அ ண் இ	മ ണി	മ അ ണ് ഇ

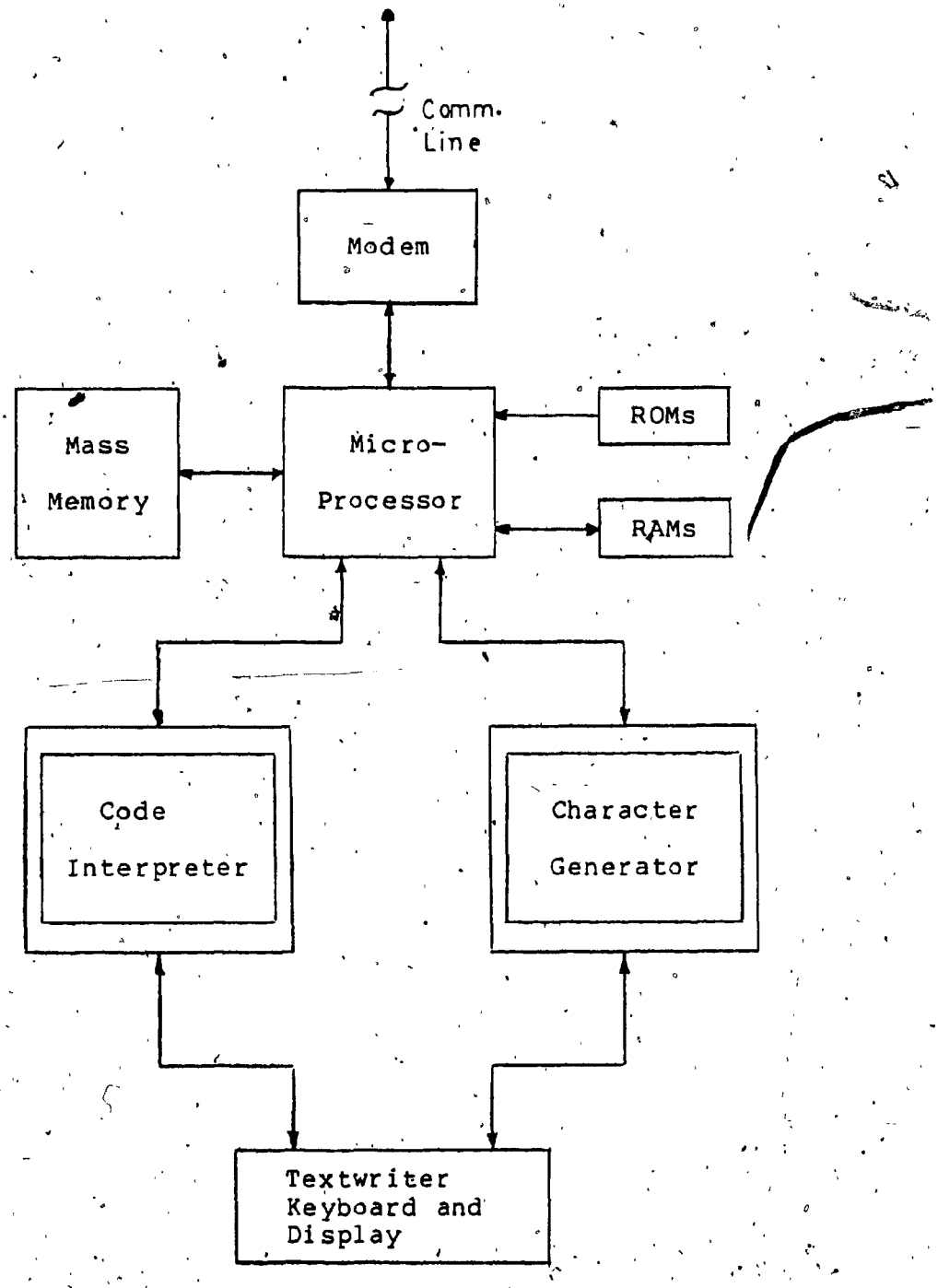


Fig. 5.2 A Microprocessor Based Multilingual Telex System Architecture

completely in ROMs or both in ROMs and mass memory based on their frequency of usage. The RAMs are used for data storage and in some configurations for storage of programs. In areas where the number of languages to be handled is limited to one or two as in rural areas, the character generators could be stored in ROMs and the mass memory may be excluded from the system.

5.2 Data Acquisition/Database Applications

Data on land revenue, family planning, census, etc., are collected nation-wide and maintained in their respective regional languages in India. As an illustration, part of the land revenue record maintained by the Tamil Nadu (the language spoken is Tamil) Government is given in figure 5.3 and figure 5.4. For administrative purposes the Indian subcontinent is divided into States and Union Territories on the basis of the language spoken by the majority of the people in that region. Each State or Union Territory is further divided into Districts, Districts into Taluks and Taluks into Villages. The items in figure 5.3 are listed below in order:

1. Serial number
2. District
3. Taluk
4. Village

தொகுப்பு வரிசை எண் :

மாவட்டம்:

வட்டம்:

கிராமம்:

பட்டா எண்: 127

நில

உடைமையாளரின் பெயர் :

தகப்பனர்/கணவர் பெயர் :

முகவரி :

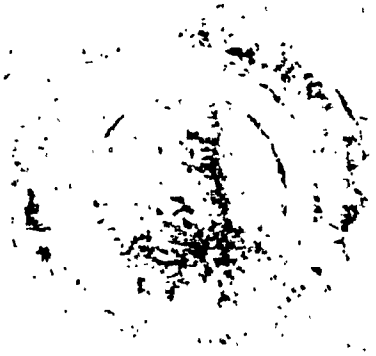


Fig. 5.3. A Sample Land Revenue Record of Tamil Nadu

6 செலுத்த வேண்டிய நிலவரியின் முழு விவரம்

நிர்வை	பசலி.					
	13 [illegible]		13		13	
	ரூ.	வப.	ரூ.	வப.	ரூ.	வப.
1. புன்செய் வரி ..						
2. நன்செய் வரி ..	10	12				
3. கூடுதல் நிலவரி ..	3	04				
4. கூடுதல் நன்னீர் மேல் வரி						
5. பசலி மிகை ..						
6. நிர்வை மிகை ..						
7. ஜாரட்டி மேல் வரி.	4	55				
8. ஜாரட்டி ஒன்றிய மேல் வரி.	15	18	—			
9. பலவகை விதிப்பு ..						
மொத்த வரி ..	32	89				
குறைப்பு.						
1. வரி தள்ளுபடி ..						
2. புன்செய் வரி நீக்கம்	1					
3. நன்செய் வரி நீக்கம்	2	81				
4. பலவகை குறைப்பு.						
மொத்தக் குறைப்பு ..	2	81				
செலுத்த வேண்டிய நிகரத் தொகை.	30	08				
கையொப்பம்	S. Nithyanantham					

Fig. 5.4: A Sample Land Taxes Record of Tamil Nadu

5. Plot number

6. Name of land owner

7. Name of father

8. Address

9. Land type

10. Crop cultivated in the last Fasli (crop season).

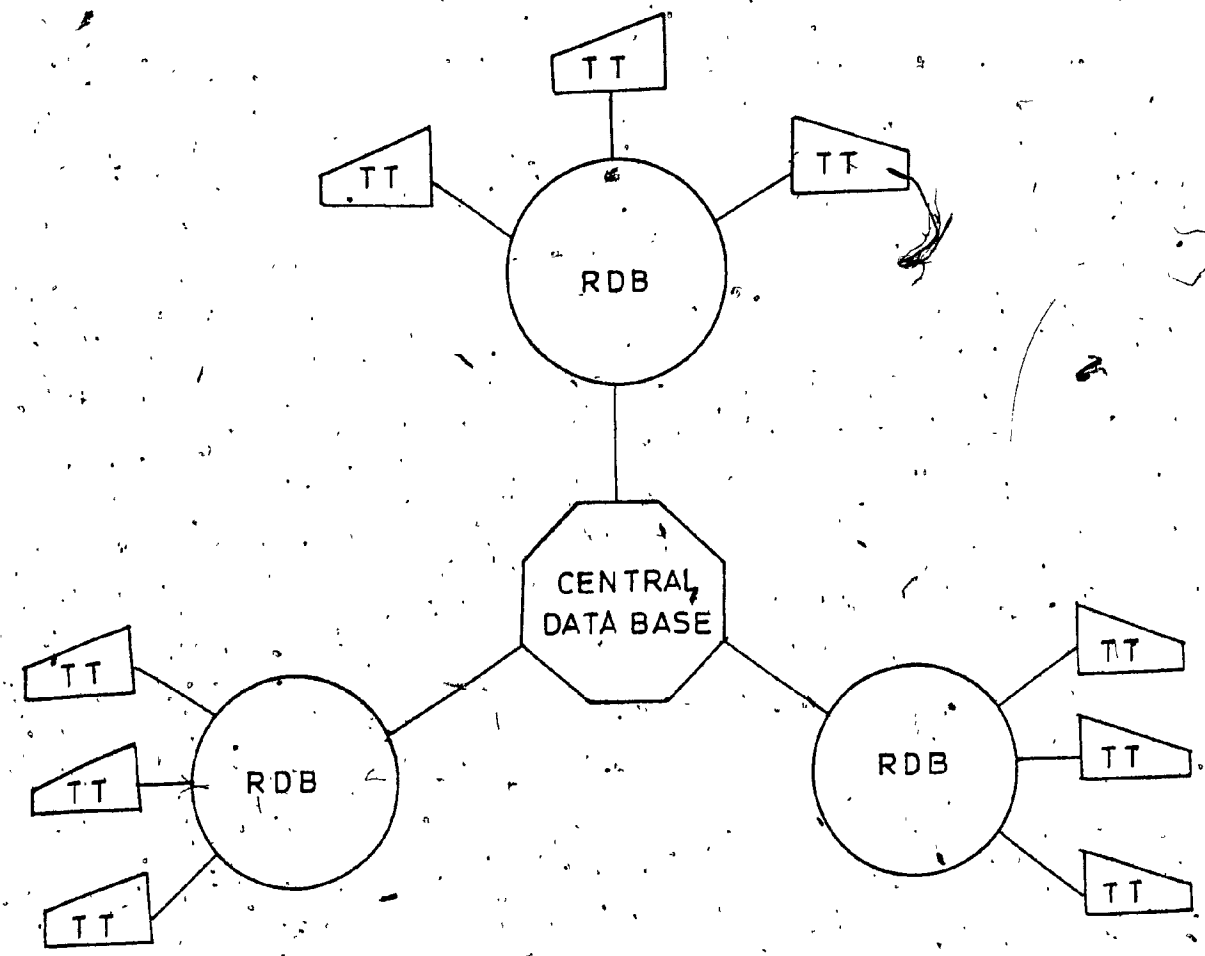
In figure 5.4 various types of taxes to be paid and possible deductions are given.

India having large number of land owners, this agricultural land records yield voluminous but at the same time valuable data. To make this information available in time for decision making, national planning, to extract statistical information, etc., computers could be of great service. Using the architecture of figure 5.2 such data can be collected and maintained.

Since information is collected nation-wide (a village being the lowest administrative level of information generation), a distributed database as shown in figure 5.5 could be created. Each State will maintain a regional database (RDB); terminals or database servers* [71] situated

---

*A possibly portable microcomputer enabling remote access to and use of a subsidiary database which is "refreshed" at intervals by communication with a central data base system.



TT : Terminal at Taluk (or) Data Base Server

RDB : Regional Data Base

Fig. 5.5 Distributed Database for Land Revenue Record

at each taluk will be connected to their respective RDBs through communication networks. The RDBs in turn will be linked to the central database at the headquarters. If one wants, the database servers at taluk level could be connected to their respective districts and districts in turn to RDBs. However the system will become more complex.

Such a data base system should have the following capabilities:

1. Data acquisition in multiple languages.

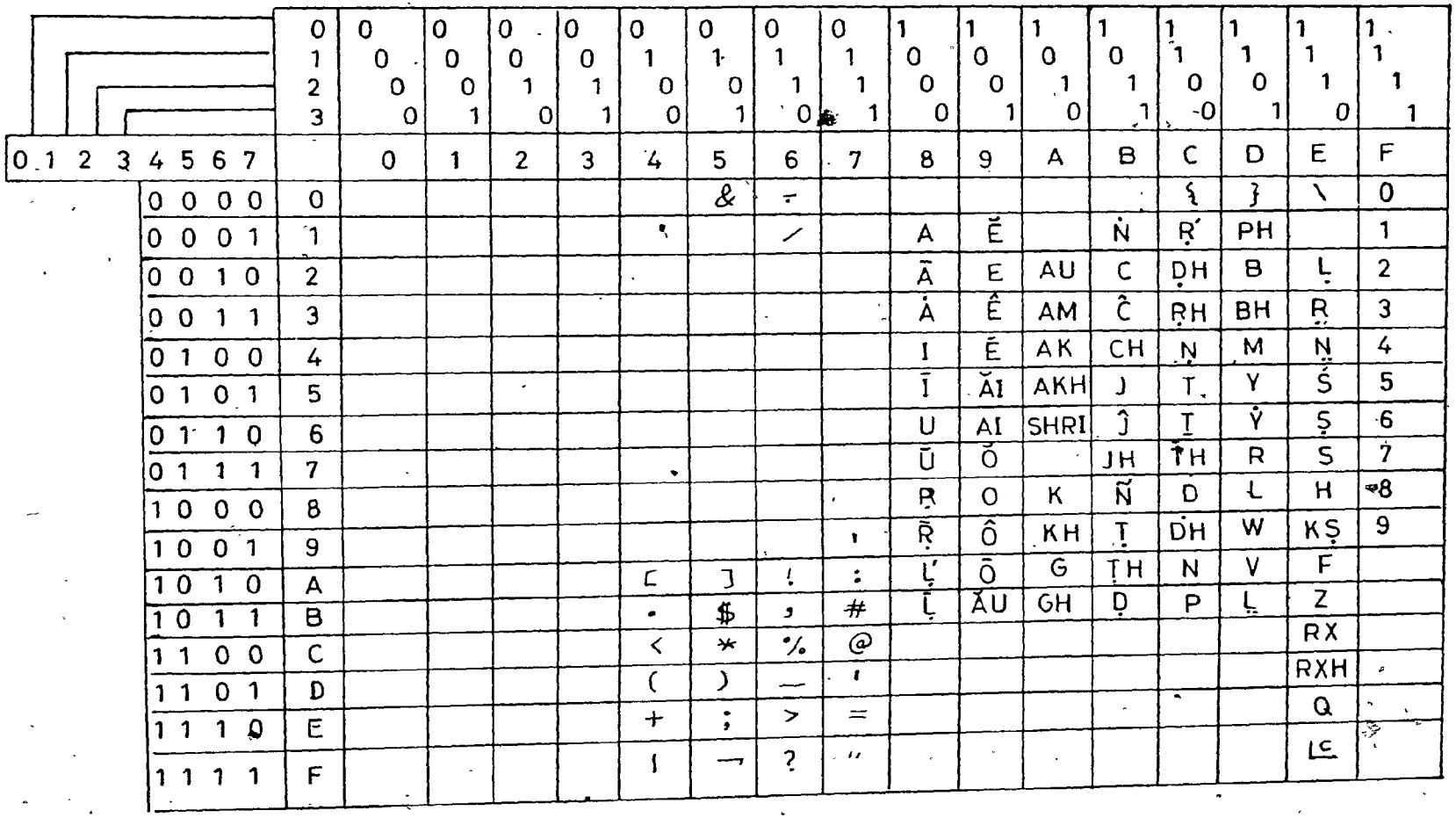
2. Information in any RDB must be made available in any other RDB. The information so furnished by the source RDB may be in its own language or in the language of the receiving RDB. Hence, RDBs (possibly the database servers) should have the capabilities to correctly interpret the information in different languages. To do so it may be necessary to transliterate and or do bounded scope translation of the information.

Capability (1) could be met with the textwriter. For capability (2), the textwriter could be expanded to be a database server with the addition of a floppy disk and the necessary micro-DBMS [71]. With improving technology, the database servers could be made portable in which case they could be carried to remote villages to collect information and refresh the central database at intervals. Thus the database remains up to date.

5.3 Automatic Transliteration of Text

One of the advantages of the BLS code is that it simplifies transliteration of text. A technique for automatic transliteration of ∝-coded text from one Indian langage to another is presented in this section; the associated problems are discussed with respect to the languages Tamil and Malayalam.

While coding the ∝s, the ∝s common among different languages are given the same internal code. One possible code for the set of ∝s of Indian characters in Table I is shown in figure 5.6; in particular the code for the ∝s of Tamil and Malayalam are shown in figure 5.7 and figure 5.8 respectively. It could be observed that from the coding scheme, the internal representation of a text becomes almost language invariant. For example, the ∝-code for the word KAMALAM is the same whether in Tamil, Malayalam or in any other Indian language. Therefore, a text entered in Tamil could be printed in Malayalam or any other Indian language by simply considering it as a text in that language. Thus at first sight, the transliteration looks trivial with BLS coding. However linguistic characteristics make transliteration a non-trivial task. Figure 5.9 gives the result of transliteration of a set of names from Tamil to Malayalam. The following points are taken into

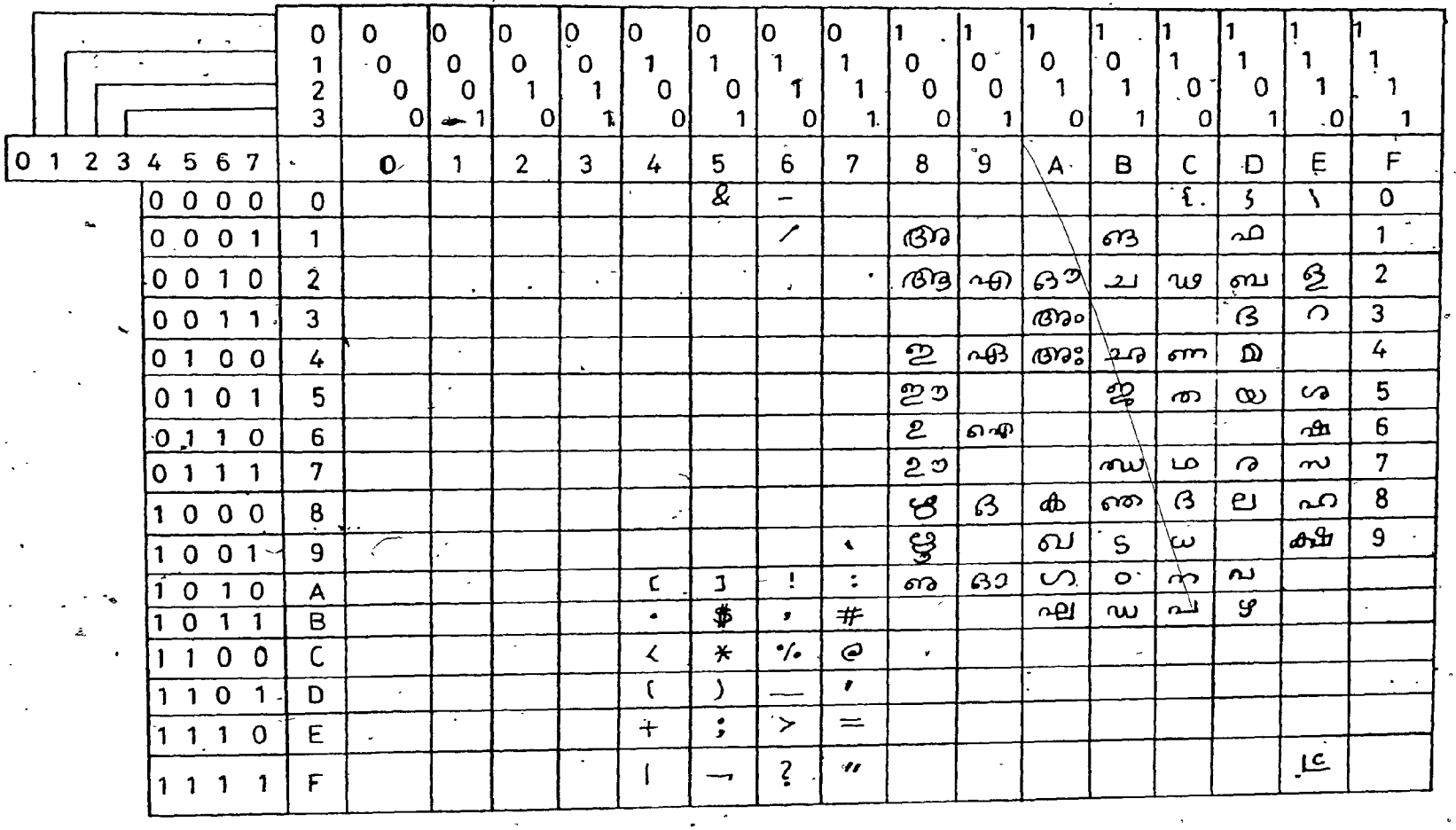


0 1 2 3 4 5 6 7		0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 0 0 0	0						&	÷						ऽ	ॐ	ॲ	0
0 0 0 1	1					'		/		A	Ĕ		Ṅ	Ṛ́	PH		1
0 0 1 0	2									Ā	E	AU	C	ḌH	B	Ḷ	2
0 0 1 1	3									Ȧ	Ê	AM	Ĉ	ṚH	BH	Ṛ	3
0 1 0 0	4									I	Ē	AK	CH	Ṇ	M	Ṉ	4
0 1 0 1	5									Ī	Ă	AKH	J	Ṭ	Y	Ś	5
0 1 1 0	6									U	AI	SHRI	Ĵ	Ṯ	Ẏ	Ṣ	6
0 1 1 1	7									Ū	Ŏ		JH	ṮH	R	S	7
1 0 0 0	8									Ṛ	O	K	Ñ	D	Ḻ	H	8
1 0 0 1	9								'	Ṝ	Ô	KH	Ṯ	ḌH	W	KṢ	9
1 0 1 0	A					[	]	!	:	Ḷ'	Ō	G	ṬH	N	V	F	
1 0 1 1	B					.	$	,	#	Ḻ	Ă U	GH	Ḍ	P	Ḷ	Z	
1 1 0 0	C					<	*	%	@							RX	
1 1 0 1	D					(	)	—	'							RXH	
1 1 1 0	E					+	;	>	=							Q	
1 1 1 1	F					!	¬	?	"							Ḷc	

Fig. 5.6 Code for ∝s of Indian Languages

0 1 2 3 4 5 6 7		0 1 2 3	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 0 0 0	0							&	–								\	0
0 0 0 1	1								/		அ			ங்				1
0 0 1 0	2										அ	எ	ஆ	ச்			ள்	2
0 0 1 1	3																ற்	3
0 1 0 0	4										இ	ஏ			ண்	ம்	ன்	4
0 1 0 1	5										ஈ		ஃ	ஜ்	த்	ய்		5
0 1 1 0	6										உ	ஐ	ஹ்				ஷ்	6
0 1 1 1	7										ஊ					ர்	ஸ்	7
1 0 0 0	8											ஒ	க்	ஞ்		ல்	ஷ	8
1 0 0 1	9									'				ட்			ஹ்ஷ	9
1 0 1 0	A						(	]	!	:		ஓ			ங்	வ்		
1 0 1 1	B						.	$	,	#					ப்	ழ்		
1 1 0 0	C						<	*	%	@								
1 1 0 1	D						(	)	—	'								
1 1 1 0	E						+	;	>	=								
1 1 1 1	F						|	¬	?	"							ட	

Fig. 5.7 Code for ∝s of Tamil



0 1 2 3 4 5 6 7		0 1 2 3	0 0 0 0	0 0 0 1	0 0 1 0	0 0 1 1	0 1 0 0	0 1 0 1	0 1 1 0	0 1 1 1	1 0 0 0	1 0 0 1	1 0 1 0	1 0 1 1	1 1 0 0	1 1 0 1	1 1 1 0	1 1 1 1
			0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 0 0 0	0							&	–						{	}	\	0
0 0 0 1	1								/		ആ			ങ		ഫ		1
0 0 1 0	2										അ	ഏ	ഒ	ച	ഘ	ബ	ള	2
0 0 1 1	3												ാ			ഭ	ഁ	3
0 1 0 0	4										ഇ	ഐ	ി	ഛ	ണ	മ		4
0 1 0 1	5										ഈ			ീ	ത	യ	ശ	5
0 1 1 0	6										ഉ	ഒ					ഷ	6
0 1 1 1	7										ഊ			ു	ഥ	ര	സ	7
1 0 0 0	8										ഋ	ഓ	ക	ൂ	ദ	ല	ഹ	8
1 0 0 1	9									'	ഌ		ഖ	ട	ധ		ക്ഷ	9
1 0 1 0	A						(	]	!	:	ഠ	ഓ	ഗ	ഠ	ന	ഩ		
1 0 1 1	B						•	$	,	#			ഺ	ഡ	പ	ഴ		
1 1 0 0	C						<	*	%	@	'							
1 1 0 1	D						(	)	—	'								
1 1 1 0	E						+	;	>	=								
1 1 1 1	F						|	¬	?	"							ൈ	

Fig. 5.8 Code for ∝s of Malayalam

TAMIL	MALAYALAM
விநாயகம்	വിനായകം
சரவணன்	ചരവണൻ
பூங்கொடி	പൂങ്കൊടി
வேலு	വേലു
வடிவேலு	വടിവേലു
ராமு	രാമു
கமலம்	കമലം
கோகிலா	കോകിലാ
சண்முகம்	ചൺമുകം
சுந்தரம்	ചുന്ദരം
நாராயணன்	നാരായണൻ
ஐசக்	ഐചക്
ராமன்	രാമൻ
மலர்	മലര്

Fig. 5.9 A Sample Transliteration from Tamil to Malayalam

consideration in the transliteration.

1. For each character in the source language, there should be an equivalent representation in the target language. For example, the Tamil characters ன் (N) and ஃ (AKH) are not present in Malayalam. They have to be realized using one or more characters in Malayalam. In our case, the Malayalam characters ന് (N) and ക് (K) are used as equivalent characters.

2. Syllables like க்க (KKA) in Tamil must be transliterated as a conjunct character ക്ക (KKA) in Malayalam and vice versa since there is no conjunct character in Tamil.

3. The orthography of certain characters are 'context' dependent. As an illustration, consider the Tamil character ர் (R). Its Malayalam equivalent is ര് (R). However ര് (R) can appear either as ര് or as ർ depending on the context. Similarly the Malayalam character ന് (N) has to be tranliterated into Tamil as either ந் or ன் depending on the contenxt.

4. Words ending with ம் (M) in Tamil should appear in Malayalam as ം and not as മ് (M). For example, விநாயகம் (VINAYAGAM) in Tamil must be transliterated into Malayalam as വിനായകം (VINAYAGAM) and not as വിനായക‌മ് (VINAYAGAM).

5. Another major problem is the one to many and many to one mapping of characters as shown in figure 5.10. Mapping many characters into a single character does not create any problem. However, the inverse mapping, that is, one to many is difficult because it depends on the context of the word (example, கந்தன் (KANTHAN), கங்கை (GANGKAI)). The contextual information is not explicitly present in the ∝ code of a character. In general, we could say that many to one mapping is trivial whereas one to many mapping is difficult and it could be resolved by context analysis.

5.4 Teaching of Scripts and Languages

The textwriter could be made useful to the public and to school children in areas such as teaching scripts and languages, question-answering, etc. In India, where the illiteracy rate is high and where many different scripts and languages are used (even within a single region), TV broadingcasting through satellites could be used to teach these scripts and languages. At different time different language lessons could be broadcast. In countries like Canada where teachers are scarce, language teaching could be accomplished through media such as 'Cable TV', 'Personal Computer' or 'Videotex Networks' [72]. Using the home TV

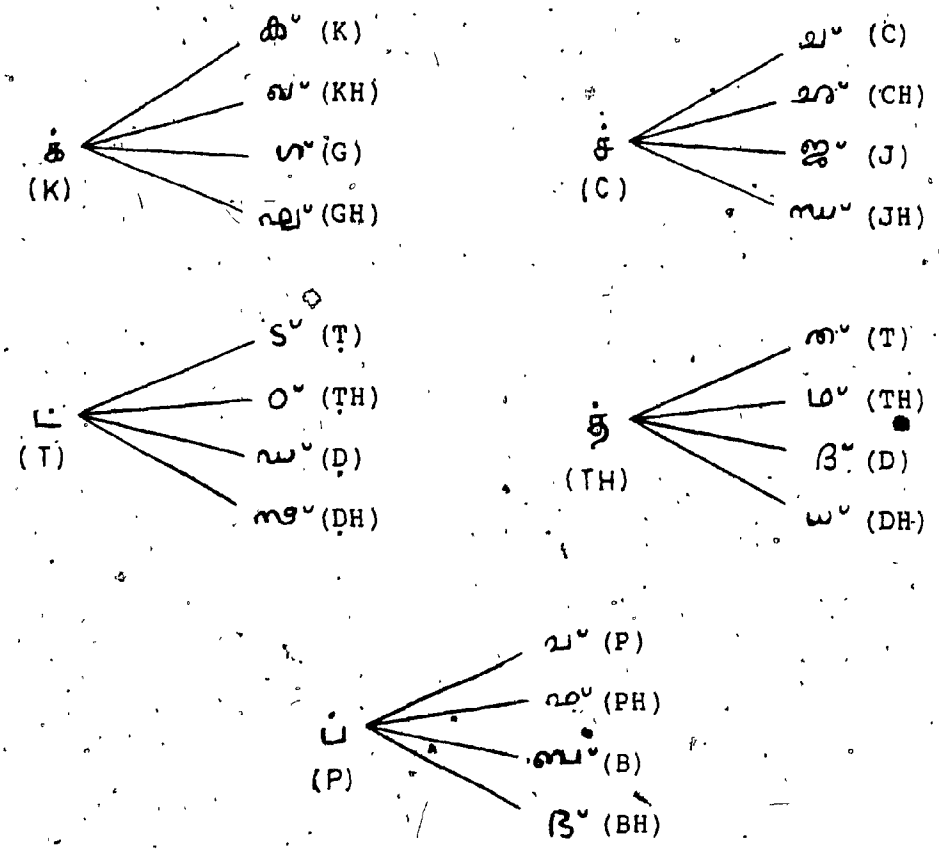


Fig. 5.10 Mapping of Certain Tamil-Malayalam Characters

screen, videotex networks can provide easy, inexpensive access to vast amounts of information.

Teaching requires both audio and visual aids. We need a multilingual database suitable for teaching multiple languages. The content of the database could be divided into two portions: (1) language invariant sub-database and (2) language dependent sub-database. As an illustration consider figure 5.11. Here the aim is to teach the word 'FLOWER' and its physical counterpart. The pictorial representation is language invariant, whereas the word 'FLOWER' will change from language to language. In figure 5.11, the equivalent word for flower is listed in Tamil, Hindi and Malayalam, in that order. Depending on the language to be taught, the appropriate word could be selected and displayed.

Character Generator: The character generator needed for teaching will be different from that of the text generator explained in Chapter IV. The character generator required for teaching should (1) display the characters in large size, preferably one character in the entire screen, and (2) should write the characters 'step-wise' as in normal writing, as shown in figure 5.12. There should be sufficient time delay between each segment written to show the method of writing. If possible, the current segment added could be made to flicker for a while for better

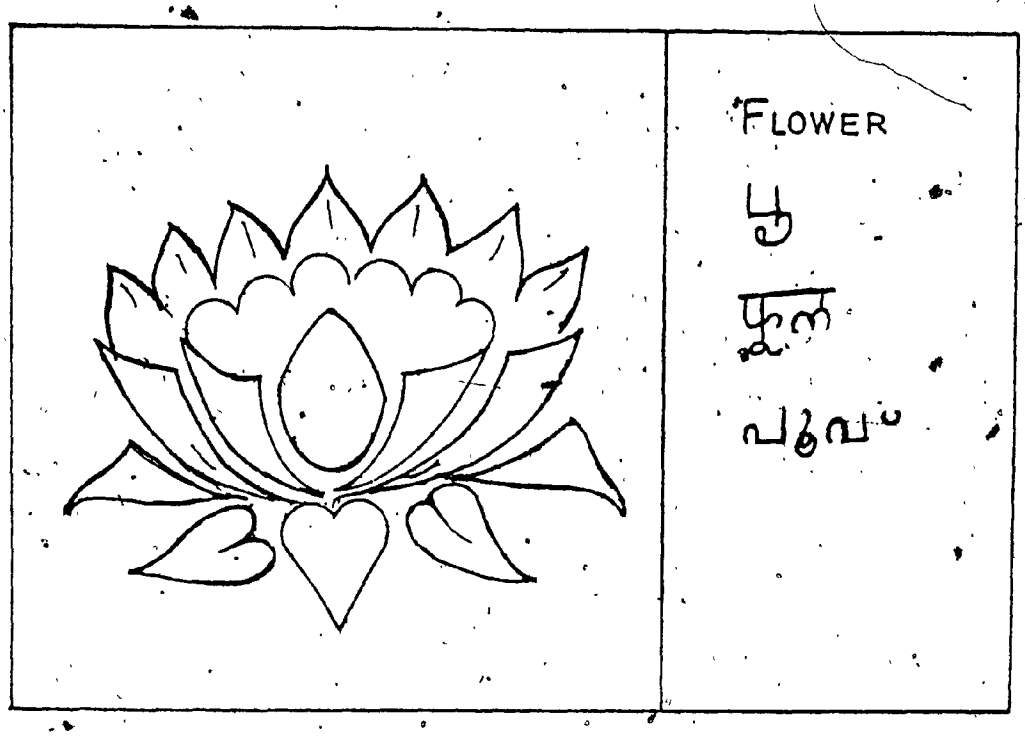


Fig. 5.11 A Sample Text for Computer Aided Teaching

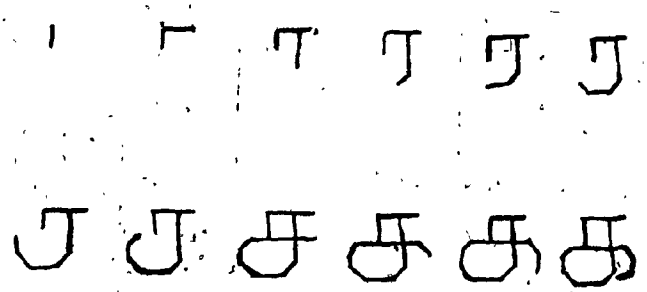


Fig. 5.12 Stepwise Generation of Character க (KA)

attention. Dot matrix type character generators may not be good for this purpose. A superior character generator should be capable of drawing the curves required for Indian characters.

Audio Support: To teach the characters of language, audio support is also needed. While writing the characters, the appropriate sound of the character should also be produced. The quality of the sound should be high and the sound should be repeated at appropriate intervals for the entire duration of the generation of a character. It should not be sounded just once. Such experiments were conducted for Tamil [73] using a dot matrix type character generator. The input to the system was the BLS code. The outcome of the experiment was encouraging. However, the dot matrix type character generator was found unsuitable. The characters were pronounced only once. Due to the limitations of Votrax used, the quality of the sound for some of the Tamil characters was not good. The important outcome of the experiment is that it revealed the feasibility of teaching scripts through simple devices such as the textwriter and a voice synthesizer.

Question-answering System: By having display devices with a light-pen, it is possible to give writing exercise to the learners. The system could pronounce a character and the person could be asked to write the character on the screen

using the light-pen. In this case the system should be able to detect an illegal combination of symbols which do not constitute a legal character [74]. For example, the symbol combination ெ ர ண is not a legal combination in Tamil. An elementary script and language teaching system could be augmented with a sophisticated question-answering system for evaluation and advanced teaching. The suitability of the textwriter for teaching scripts and languages could be easily visualized.

Another impotant application area for the textwriter is text composition for printers. A single textwriter could be used to compose any Indian language text. By incorporating the hypenation rules, the textwriter could as well be used as a stand-alone unit. The textwriter will be economical and convenient. These are hot the only applications of a textwriter. The list could be extended very easily.

6. CONCLUSION

The work reported in this thesis is interdisciplinary in nature, encompassing computer science and linguistics. In fact, the challenge seems to be more from linguistics than from computer science due to the large number of languages and their individual characteristics.

Outlined in the thesis are the various design aspects of a textwriter suitable for input/output of the text of one or more languages and some applications of the textwriter. Problems specific to each of these applications have been studied.

The processor based textwriter is flexible to meet the changes (addition and deletion) in the number of languages supported by it. It could be used for multiple purposes, for example, as an I/O terminal for word processing in offices, as a data entry terminal, a computer terminal or as a telex terminal.

The keyboard of the textwriter could be either symbol based or $\propto$ based. A code suitable for representing multilingual text is given. The $\propto$-coding of text proposed in this thesis has the advantage of giving the text in

collating sequence suitable for sorting whereas the symbolic coding does not.

Two hierarchical models have been proposed for character generation. They have the generality of being applicable to a large number of scripts and languages. Simple device dependent primitives used in these methods enable easy transportability among different devices such as plotter, printer and display.

By combining the proposed scheme for the automatic method of determining dot matrix size with our dot pattern design board, and making the resulting system interactive, we could automate the entire process of font design, font analysis, font evaluation and generation (optimal or non-optimal) of ROM storage for the symbols. This will enable the design of different fonts at much faster rate than a manual method.

Many schemes were proposed for key top symbol display of which the plasma panel display seems to be promising. The decreasing cost of hardware components will ensure the economic viability of such a terminal. Our software testing of the textwriter shows its technical feasibility.

This thesis, apart from its reported contributions, has brought out a number of problems requiring further research: (1) Hardware/firmware realization of the textwriter. (2)

Software implementation of the dot pattern design board. (3) Micro coding (rather than macro coding) of the text generator, in order to improve the performance and the speed of the textwriter. (4) Experiments on the selection of primitives, and on ROM storage optimization. (5) Extension of the character generator for aesthetic appearence. (6) Extension of the facility for transliteration of text, in order to take care of the one-to-many mapping using contextual information [75]. (7) Design of a data base server using the textwriter as an input/out device.

REFERENCES

Note:- LICIS stands for Proceedings of Symposium on Linguistic Implications of Computer Based Information Systems, Electronics Commission, New Delhi, India, Nov.10-12, 1978.

1. Gilyarevsky, R.S., and Grivnin, V.S., "Languages identification guide", 'NAUKA' Publishing House, Central Department of Oriental Literature, Moscow, 1970.

2. Nancarrow, P.H., "48,000 characters in search of a system", LICIS.

3. Stalling, W., "Recognition of printed Chinese characters by automatic pattern analysis", Computer Graphics and Image Processing, Vol. 1, 1972, p.47.

4. Yoshita, M., Iwata, K., Yamamoto, E., Masui, T., and Kabuyama, Y., "The recognition of handprinted characters including 'KATAKANA', 'Alphabets' and 'numeral', Proc. 3rd Int. Conf. Pattern Recognition, 1976, pp. 645-649.

5. Hyder, S.S., "A system for generating Urdu/Farsi/Arabic scripts", Information Processing 77 : Proc. of IFIP Congress, North Holland, Amsterdam, 1977, pp. 1144-1149.

6. Parhami, B., "On the use of Farsi and Arabic languages in computer based information systems", LICIS.

7. Parhami, B., and Mavaddat, F., "Computers and the Farsi language : A survey of problem areas", in Information Processing 77, Proceedings of IFIP Congress 77, Toronto, Aug.8-12, 1977, North Holland, pp673-676.

8. EC-1030 computer series, I.I.T., Bombay, India.

9. Kannaiyan, V., "Scripts in and around India", Government Museum Bulletin, Madras, India, 1960.

10. Koteshwara Rao, N.V., "Computer programming in Indian languages", LICIS.

11. Mudur, S.P., and Wakankar, L.S., "Computer input/output in Devanagari", LICIS.

12. Krishnamoorthy, S.G., Isaac, J.R., Ramanamurthy, S.V., and Rao, P.V.L.N., "Study and coding of Telugu characters for computerized information handling", LICIS.

13. Narasimhan, R., and Reddy, V.S.N., "A generative model for handprinted English letters and its computer implementation", Information and Control (ICC) Bulletin, Vol. 6, 1967, pp. 253-265.

14. Krishnamoorthy, S.G., and Radhakrishnan, T., "A state of the art survey of computerized information handling in Indian languages", Proc. Computers and Office Automation, Calgary, 1980.

15. Krishnamoorthy, S.G., and Radhakrishnan, T., "A KWOC index to the bibliography on computer processing of Farsi and Indian language scripts", Department of Computer Science, Concordia University, June 1980.

16. Pattanayak, D.P., "Problem and planning scripts", LICIS.

17. Abasama, K., "Abasama script for the world languages", Bellare 574 212, South Kanara, India.

18. Sulochanan, "SULIPI : a commom script for most of the Indian languages", Kairaleemandiram, Mezhuvelly (P.O.), Alleppely Dist., Kerala, India.

19. Parameswara Iyer, S., "Nandaka alphabet for English : A feasible common script", Nandaka Ripi Niket, Fort, Trivandrum-1, Kerala, India.

20. Krishnamoorthy, S.G., and Isaac, J.R., "INDUS : A linear script for Indian languages", Proc. of Computer Society of India Annual Convention, Bombay, 1980.

21. Nayak, J., "Evolution of Devanagari script", LICIS.

22. Siromoney, G., "Entropy of Tamil script", Information and Control, Vol. 6, 1963, pp. 297-300.

23. Siromoney, G., "Certain applicaions of information theory", Ph.D Thesis, University of Madras, 1964.

24. Balasubramanyan, P., and Siromoney, G., "A note on entrophy of Telugu prose", Information and Control, Vol. 13, 1968, pp. 281-285.

25. Menon, K.P.S., "1 and 2 letter frequencies in Hindi (directly) by computer", Tech. Report 73-10 (Sept.), Minnepolis, Minn., Dept. of Computer, Information and Control Sciences, Univ. of Minnesota (text 8 pp, supplementary material 97 pp).

26. Rajagopalan, K.R., "Entropy of Kannada prose", Information and Control, Vol. 8, 1965, pp. 640-644.

27. Avadhanulu, R.V.S.S., Chandraprakash, V., and Koteswara Rao, N.V., "A unified approach to the design of Telugu and Hindi FORTRAN compilers", LICIS.

28. Anil Kumar, and Sarna, C.S., "On the generation of Hindi characters on a dot matrix printer through firmware and software", LICIS.

29. Laturkar, K.P., and Sinha, R.M.K., "Devanagari script composition from phonetically coded symbol strings", LICIS.

30. Mahabala, H.N., and Raman, S., "Determination of minimum matrix size of Tamil characters for optimal presentability using an on line teletype as Tamil typewriter", Computer Society of India Annual Convention, Calcutta, 1978.

31. Menon, K.P.S., "Direct input/graphical output of two Indian languages : Hindi and Malayalam", J. Computer Soc. India, Vol. 2, 1971, pp. 14-26.

32. Menon, K.P.S., "High speed visual, direct Indian language data entry", Indian Linguistics, Vol. 35, No. 2, 1974, pp. 97-111.

33. Sinha, R.M.K., "Segment display for Devanagari scripts and numerals", LICIS.

34. Ramakrishna, B.S., "Information theoretical models for translation and transcription", LICIS.

35. Sinha, R.M.K., "Machine oriented Devanagari script (MODS) from information theoretic point of view", LICIS.

36. Krishnamoorthy, S.G., and Isaac, J.R., "A unified approach to information handling in phonetic languages", Tech. Report, Computer Center, I.I.T., Bombay, India, 1979.

37. Dhyani, P., "FORTRAN in Hindi : its implementation on IBM-1130", LICIS.

38. Namperumal, S., "Pre and post processors for FORTRAN jobs in Hindi", LICIS.

39. Narasimham, P.V.H.M.L., and Raman, A., "A CRT based phototypesetter for Indian languages", LICIS.

40. Krishnayya, J.G., and Viswanathan, N., "An English vernacular bridge for news agencies", LICIS.

41. Mahabala, H.N., and Raman, S., "Computer generated regional language animated film titles", LICIS.

42. Mazumdar, A.B., "A review of typesetting developments of Indian languages", LICIS.

43. Schreiber, W.F., "Computer based composition of Indian scripts", LICIS.

44. Ram Vidya, "An approach to the design of a word processor using ZILOG Z80 microprocesor", LICIS.

45. Krishnamoorthy, S.G., and Isaac, J.R., "Multilingual word processor", Proc. of Computer Society of India Annual Convention, Bombay, 1980.

46. Sinha, R.M.K., and Sahasrabuddhe, H.V., "Hyphenation in Indian scripts for computer aided printing", LICIS.

47. Ananthanarayanan, K.R., "Computer based medical information system and use of Indian languages", LICIS.

48. Gupta, J.P., and Prabhakar, A.S., "Need for software modification in computerized information system for health and family welfare", LICIS.

49. Namperumal, S., "Methodology for machine translation", LICIS.

50. Ganeshsundaram, P.C., "Interlinguistic relativity and translation (Computer-aided translation with pre and post editing)", presented in the Vth All India Conference on Linguistics, Delhi, 1974.

51. Chakraborty, A.R., Kundu, S.R., and Raizada, A.S., "An experiment on machine translation from Hindi to Bengali: A step towards translation interlingua for Indian languages", LICIS.

52. Ganeshsundaram, P.C., "Development of a suitable script - an input problem in the analysis of Indian language using the IBM 360/44 computer", J. Indian Inst. Sci. Vol. 58, No. 5, 1976, pp. 225-232.

53. Havanur, S.K., "Computer assistance in deciding the authorship of an Indian poem", Assoc. Lit. and Ling. Computing Bulletin, Vol. 4, No. 1, 1976, pp. 125-130.

54. Rawlinson, A., "A Sanskrit concordance program : printout and explantions - A communication", Dept. of Religious Studies, Lancaster, England.

55. Siromoney, G., Chandrasekaran, R., and Chandrasekaran, M., "Computer recognition of printed Tamil characters", Pattern Recognition, Vol. 10, No. 4, 1978, pp. 243-247.

56. Chinnuswamy, P., and Krishnamoorthy, S.G., "Recognition of handprinted Tamil characters", Pattern Recognition, Vol. 12, 1980, pp. 141-152.

57. Rajasekaran, S.N.S., and Deekshatulu, B.L., "Recognition of printed Telugu characters", Computer Graphics and Image Processing, Vol. 6, 1977, pp. 335-360.

58. Sinha, R.M.K., "Syntactic pattern analysis and its application to Devanagari script recognition", Ph.D. Thesis, Electrical Engg. Dept., I.I.T., Kanpur, India, Dec. 1973.

59. Sethi, I.K., and Chatterjee, B., "Machine recognition of constrained handprinted Devanagari", Pattern Recognition, Vol. 9, No. 2, 1977, pp. 69-75.

60. Som, A., and Nath, A.K., "On some methods of sequential pattern recognition", ISI Symposium on Digital Techniques and Pattern Recognition, Calcutta, 1977.

61. Krishnamoorthy, S.G., Isaac, J.R., and Bhavsar, V.C., "A microproccessor based multilingual terminal for computerized information handling system", Journal of Computers and Humanities, Vol. 14, 1980, pp. 91-104.

62. Narasimham, P.V.H.M.L., Prasad, B., and Rajaraman, V., "Code based keyboard for Indian languages", J. Comput. Soc. of India, Vol. 2, No. 2, 1971, pp. 33-37.

63. Narasimham, P.V.H.M.L., and Sinha, R.M.K., "Phonetically coded keyboarding in Indian languages", LICIS.

64. Deutsch, E.S., "Thinning algorithms on rectangular, hexagonal and triangular arrays", Comm. of ACM, Vol. 15, No. 9, 1972, pp. 827-837.

65. Proc. of the Society for Information Display, Vol. 21, No. 1, 1980

66. Snyder, H.L., and Maddox, M.E., "On the image quality of dot-matrix displays", Proc. of the Society for Information Display, Vol. 21, No. 1, 1980, pp. 3-7.

67. Suen, C.Y., and Shiau, C., "An iterative technique of selecting an optimal 5 x 7 matrix character set for display in computer output systems", Proc. of the Society for Information Display, Vol. 21, No. 1, 1980, pp. 9-15.

68. Maddox, M.E., "Two-dimensional spatial frequency context and confusions among dot-matrix characters", Proc. of the Society for Information Display, Vol. 21, No. 1, 1980, pp. 31-40.

69. Krishnamoorthy, S.G., "Microprocessor based multilingual telex system with phonetically coded keyboarding", Proc. of Pacific Telecommunication Conference, Honolulu, Jan. 1980.

70. Colmerauer, Alain, "Les systems-Q ou un formalisme pour analyser et synthetiser des phrases sur ordinateur," Interne, No.43, Universite de Montreal, Sept. 1970.

71. Ting, D.P., and Tsichritzis, D.C., "A Micro-DBMS for a distributed data base", Proc. Fourth Int. Conf. on Very Large Data Bases, Sept. 1978, pp. 200-206.

72. Ball, A.J.S., Bochmann, G.V., and Gecsei, J., "Videotex Networks", Computer, Vol. 13, No. 12, 1980, pp. 8-14.

73. Raman, H., "Character and speech generation using a computer", Project Report, Dept. of Computer Science, Concordia University, Montreal, Canada, 1981.