Universal Multilingual Information Interchange System
With Character Reader and Terminal

Suban Krishnamoorthy

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy at
Concordia University
Montreal, Quebec, Canada

November 1990

ISBN 0-315-64750-7

ABSTRACT

Universal Multilingual Information Interchange System
With Character Reader and Terminal

Suban Krishnamoorthy, Ph.D.
Concordia University, 1990

The need for a universal multilingual information interchange system with character reader and terminal has been emphasized. A scheme for recognizing machine-printed and handprinted Indian characters has been developed. In this scheme, the characters are assumed to be composed of symbols which in turn are assumed to be composed of line-like elements, called primitives, satisfying certain structural constraints. Attribute graphs are used to describe the structural composition of symbols in terms of the primitives and the relational constraints satisfying them. In the first stage of the two stage recognition process, the correlation coefficients are computed with the attribute graphs stored in the knowledge base for a set of basic graphic symbols and then maximized to recognize the symbols constituting the input character. In the second stage, the input character is recognized from the graphic symbols using a decision tree. The recognizer is intelligent enough to differentiate invalid combination of graphic symbols that do not constitute a valid character. Some preprocessing techniques are discussed to convert the input image into an attribute graph. A coding scheme has been developed to describe multilingual texts. Tamil and Malayalam characters were used to test the recognizer. The results for Tamil are: 91.33% recognition rate, 6.83% rejection rate and 1.83% substitution rate. The results for Malayalam are: 89.5% recognition rate, 8.3% rejection rate and 2.2% substitution rate.

A design is presented for a keyboard based multilingual terminal system for Indian languages. An interactive, computer aided, pattern recognition based, methodology has been developed to identify the best dot matrix size for a given optimality criteria. The nearly optimal dot matrix size for the Tamil symbols was computed to be 11x14 using a graphics terminal having 60 dots per inch resolution. From the symbol size, the character size has been computed as 15x18. An iterative method of determining the most distinct set of dot matrix characters is described based on the distances and information content of the dot matrix characters. Symbol based keyboard design and character generation methodologies have been developed.

Finally, the design of a multilingual data communication system using the multilingual character reader and terminal is considered as an integrated information interchange system. Various communication aspects such as character coding, multilingual document representation, and protocol changes needed for multilingual information interchange are described in detail. Also, a methodology for handling multilingual texts using the existing programming language tools such as compilers for software development is described.

To

Friends and Foes

Acknowledgements

I am very grateful to my supervisor Professor C. Y. Suen for his excellent guidance, support and encouragement without which I could not have completed this challenging endeavor.

I am thankful to many friends like Dr. Pervez Ahmed, Dr. N. L. Sarda, Dr. V. Bhavsar, Dr. Narayanan, and others who have helped me during the course of the program. In particular, my special thanks to Dr. Pervez Ahmed and his wife Mrs. Shama Ahmed. They have gladly accommodated me to stay with them on several visits to Montreal and rendered help in many ways.

My thanks also to Professor T. Radhakrishnan, Professor J. W. Atwood, the members of the thesis committee and several other faculty members of the department of computer science for their many folded kind assistance. I am thankful to the staff members of the graduate office and the department of computer science for providing assistance during registration and other times.

I would like to take this opportunity to thank my family (Kokila, Saravanan and Mala) for not only tolerating the inconveniences but also providing constant encouragement.

List of Contents

Contents **Page Number**

Chapter 1
Introduction

Chapter 2
Multilingual Character Reader

Chapter 3
Features
Definition, Extraction and Representation

Chapter 4
Character Recognition

Chapter 5
Multilingual Terminal Design

Chapter 6
Determination of Symbol Set

Chapter 7
Multilingual Keyboard Design and Character Generation

Chapter 8
Multilingual Data Communication System

Chapter 9
Conclusion

List of Figures

List of Tables

1. INTRODUCTION

1.1 Trends

Computers were unheard of or were a dream at the beginning of this twentieth century. Five decades ago, computers were for only the elite engineers and scientists and the rich. Until a couple of decades ago computers remained as a standalone system with small amount of memory and costing multimillion dollars. In recent years, computing science and technology have advanced tremendously, the cost has been reducing drastically, the computing power kept increasing many folds, and personal (micro) computers started to proliferate everywhere Today, powerful computing system with a million bytes of main memory has become an affordable household item like a TV, even though computers have not found as much use as TV's in houses. One could buy a computing system for a few hundred dollars to play games or to perform word processing at home.

The proliferation of cheap personal computers have made valuable hardware, software and information resources distributed throughout an organization. Systems have become more and more resource hungry. A couple of decades ago, information was stored in several different disjoint files. The information existed in the system over several files could not be retrieved without reprogramming. This has resulted in the consolidation of information to form a database and a new system called database management system was developed to manage the database. In a similar way geographically deployed computing systems resulted in parallel and rapid growth of corporate information. However, those resources could not be shared and retrieved when needed without the ability to talk to each other. This has created a demand for interconnectivity or networking computing systems. In the last two decades several noncompatible computer local area networks such as carrier sense multiple access with collision detection (CSMA/CD), token bus, token ring, etc., were developed. The advances in computing techniques also helped the rapid development of communication systems which enabled wide area or public networking. Today, networking and computing are becoming an integral part of a system. Some computer manufacturers have been making (multiprocessor) systems in which the user terminals are connected to the network via a network terminal server (NTS) rather than directly connecting them to the system like in the earlier days. Several protocols such as System Network Architecture (SNA), Open System Interconnect (OSI), X.25, DoD TCP/IP family, IEEE 802 family, etc., were developed. To reduce the incompatibilities, several standards organizations such as CCITT, ISO, CSA, ANSI, IEEE, etc., were formed. File transfer and network file systems have been used so much that the protocols File Transfer Protocol (FTP) and Network File System (NFS) have become defacto industry standards.

We should call the current and future systems as **Computing and Communication Systems (CCSs)** or **NetComp Systems (NSs)** to be more appropriate.

In today's technology driven society, computers are widely used throughout the world, in the developed nations as well as developing nations. CCSs are being used in different fields such as humanities and social sciences, engineering, economics, education, etc. The economy of each nation is no longer completely independent; it is becoming global due to multinational companies and international trading using CCSs. More than people mobilization, rapid electronic information mobilization between nations is rapidly increasing day-by-day with high volume and becoming more and more critical. In the near future, the entire CCSs in the world will become a single interdependent distributed computing and communication (information interchange) system.

1.2 Linguistic Limitations of CCSs

The information handled by the CCSs can be divided into (1) **AlphaNumeric** and (2) **Non-AlphaNumeric**. The alphanumeric information can be further subdivided into (i) **Alphabetic** and (ii) **Numeric.**

The alphanumeric information has two important attributes. They are:

1. **Language**
2. **Script**

For example, the alphabetic information generated by the alphabets A to Z, a to z are in the English language. The letters A to Z and a to z are the scripts of the English language. The numeric information generated by the digits 0 to 9 uses Roman script. Even though the digits 0 to 9 are universally used, there are nations, which use different scripts even for these ten digits as shown in Figure 1.1 (Gilyarevsky [1970]).

May be with a few exception, all the existing CCSs are capable of handling information only in English. Almost, none of the existing CCSs is capable of handling multilingual textual information such as the one shown in Figure 1.2 (Lifco [1975]). One should realize that this is an important limitation of the existing systems.

However, there are many languages in the world which are different from English. In fact a major portion of the world population do not use English in their day to day life. These people generate, maintain and assimilate large volumes of information (about their business, literature, culture, etc.) in their native languages (other than English). As an illustration a land revenue bill used in the province of Tamil Nadu, India, is shown in Figure 1.3.

1.3 Universal Need

CCSs are increasingly used in the non-English nations. For example, in India which has almost a billion people, fifteen different languages with ten different scripts, computers are reaching the common public like in utility billing such as electricity bills, land taxation bills, family planning, demography, etc. Currently, in India, computerized

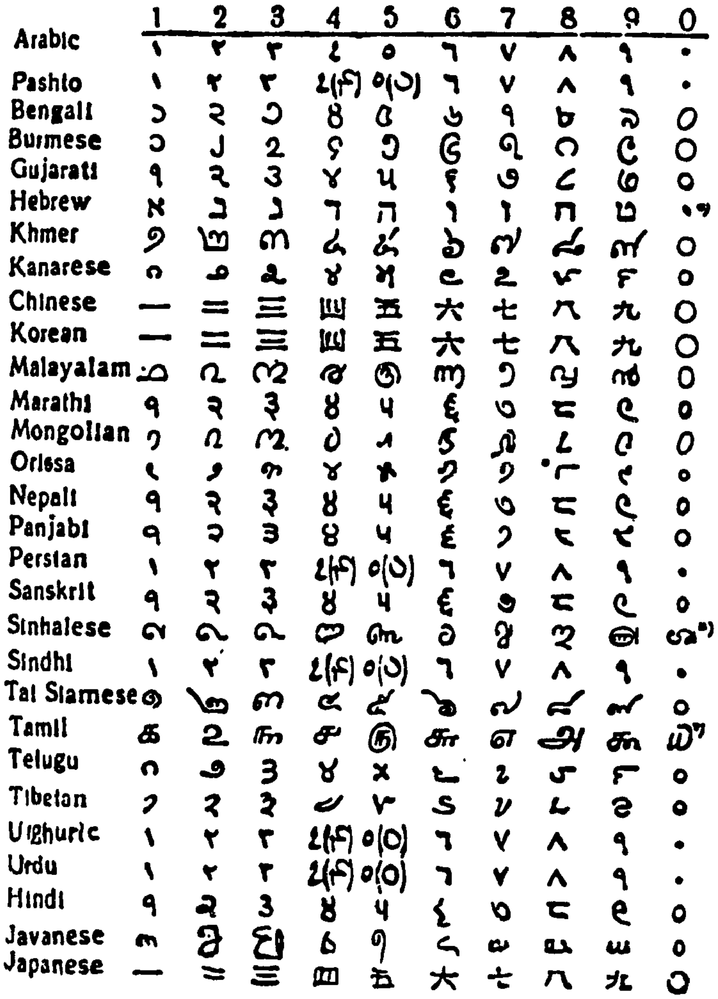
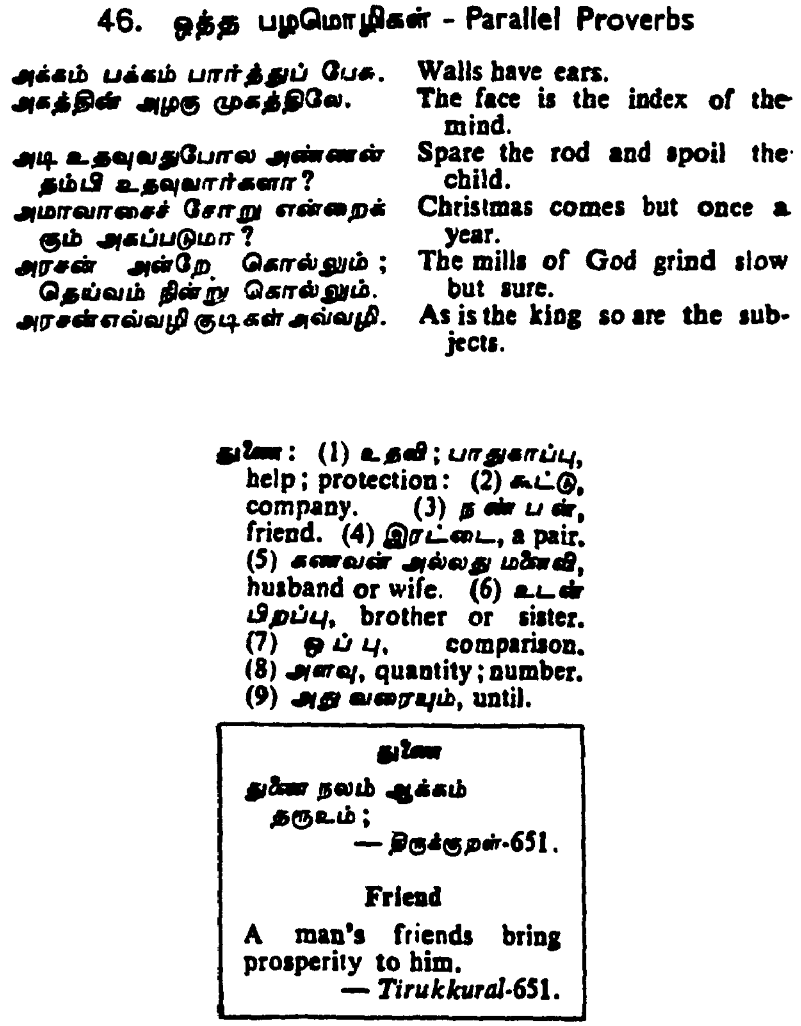

1 2 3 4 5 6 7 8 9 0
Arabic
Pashto
Bengali
Burmese
Gujarati
Hebrew
Khmer
Kanarese
Chinese
Korean
Malayalam
Marathi
Mongolian
Orissa
Nepali
Panjabi
Persian
Sanskrit
Sinhalese
Sindhi
Tai Siamese
Tamil
Telugu
Tibetan
Uighuric
Urdu
Hindi
Javanese
Japanese

Figure 1.1 Numerals in different languages.



46. ஒத்த பழமொழிகள் - Parallel Proverbs

அக்கம் பக்கம் பார்த்துப் பேசு.	Walls have ears.
அகத்தின் அழகு முகத்திலே.	The face is the index of the mind.
அடி உதவுவதுபோல அண்ணன் தம்பி உதவுவார்களா?	Spare the rod and spoil the child.
அமாவாசைச் சோறு என்றைக்கும் அகப்படுமா?	Christmas comes but once a year.
அரசன் அன்றே கொல்லும்; தெய்வம் நின்று கொல்லும்.	The mills of God grind slow but sure.
அரசன்எவ்வழி குடிகள் அவ்வழி.	As is the king so are the subjects.

துணை: (1) உதவி; பாதுகாப்பு, help; protection: (2) கூட்டு, company. (3) நண்பன், friend. (4) இரட்டை, a pair. (5) கணவன் அல்லது மனைவி, husband or wife. (6) உடன் பிறப்பு, brother or sister. (7) ஒப்பு, comparison. (8) அளவு, quantity; number. (9) அது வரையும், until.

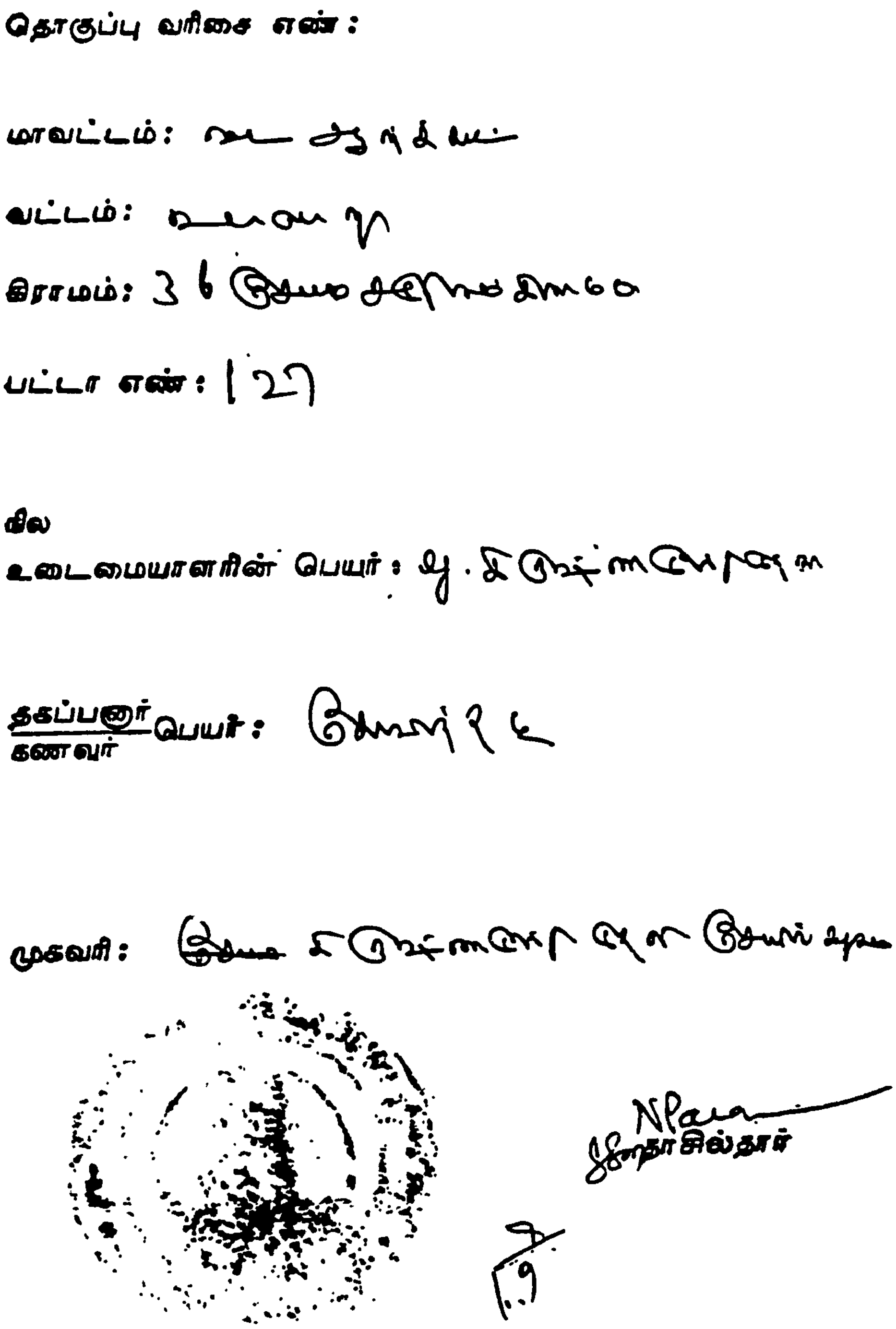
துணை

துணை நலம் ஆக்கம்
தருஉம்;
— திருக்குறள்-651.

Friend

A man's friends bring prosperity to him.
— *Tirukkural*-651.

Figure 1.2 Bilingual texts in Tamil and English.



தொகுப்பு வரிசை எண் :

மாவட்டம்: வட ஆற்காடு

வட்டம்: [illegible]

கிராமம்: 36 [illegible]

பட்டா எண் : 127

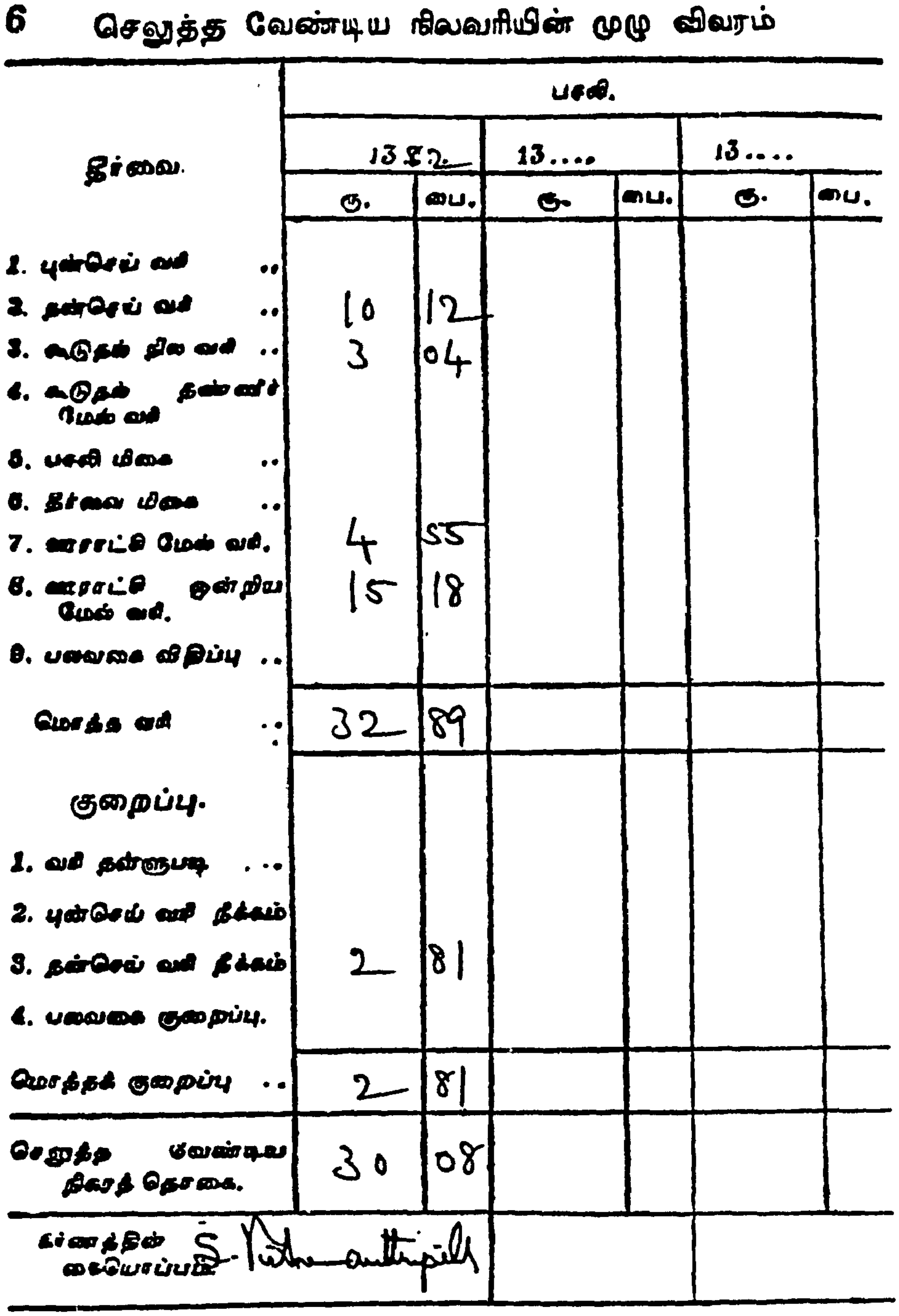
நில
உடைமையாளரின் பெயர் : [illegible]

தகப்பனார்/கணவர் பெயர் : [illegible]

முகவரி : [illegible]

தாசில்தார்

Figure 1.3 A sample land revenue record found in the State of Tamil Nadu.



6 செலுத்த வேண்டிய நிலவரியின் முழு விவரம்

தீர்வை.	பசலி.					
	1352		13....		13....	
	ரூ.	பை.	ரூ.	பை.	ரூ.	பை.
1. புன்செய் வரி ..						
2. நன்செய் வரி ..	10	12				
3. கூடுதல் நில வரி ..	3	04				
4. கூடுதல் தண்ணீர் மேல் வரி						
5. பசலி மிகை ..						
6. தீர்வை மிகை ..						
7. ஊராட்சி மேல் வரி.	4	55				
8. ஊராட்சி ஒன்றிய மேல் வரி.	15	18				
9. பலவகை விதிப்பு ..						
மொத்த வரி ..	32	89				
குறைப்பு.						
1. வரி தள்ளுபடி ...						
2. புன்செய் வரி நீக்கம்						
3. நன்செய் வரி நீக்கம்	2	81				
4. பலவகை குறைப்பு.						
மொத்தக் குறைப்பு ..	2	81				
செலுத்த வேண்டிய நிகரத் தொகை.	30	08				
கர்ணத்தின் கையொப்பம்	S. [illegible]					

Figure 1.3 A sample land revenue record found in the State of Tamil Nadu (contd.).

billings are generated (and can only be generated) in English. The public who are the end users do not understand these bills thus, they tend to become anti-computer or pressure the computing community to change their system to handle their native languages and scripts.

Another example is the telex and telegraph communication systems in India. Right now people in India can send telex or telegrams only in English. Common public who want to send and receive telegrams have to depend on the English speaking minority (less than tenth of the population) of people in that country. The limitations of the existing computing systems have made these literates (in their native language) illiterates.

One should clearly understand that the linguistic limitation of the existing CCSs is a great drawback for non-English speaking countries and people. There is a great pressure on the computer scientists and engineers to eliminate linguistic barriers. Technically advanced nations such as Japan already developed computing systems to handle the Japanese language. Similar situation is now happening to the Chinese language. European people such as French and German enhanced their communication systems to handle their scripts by adding diacritical marks (used in their language) to the international telex and telegraph character set. Developing nations such as India, Middle East, etc., are in the process of developing systems to handle their scripts and languages.

Still, the designs of the existing systems are not general. The solutions proposed so far are partial in the sense they could handle only one or two additional scripts.

"It is the contention of this thesis that at present a unified solution to handle information in any (variable) number of scripts and languages does not exist."

The world is moving rapidly towards global integrated system. As stated before, the entire CCSs in the world will soon become interdependent distributed computing and communication (information interchange) system. For such a world what is needed is a universal multilingual CCS capable of manipulating and interchanging multilingual information between linguistically different nations of the world. Such a system should adopt a coherent and uniform methodology to handle the multilingual information. The design of such a system is the main theme of this thesis.

1.4 Universal CCSs

An universal multilingual information interchange system considered in this thesis is shown in Figure 1.4. Notice that CCSs are composed of several language dependent subsystems to handle information. They are listed below.

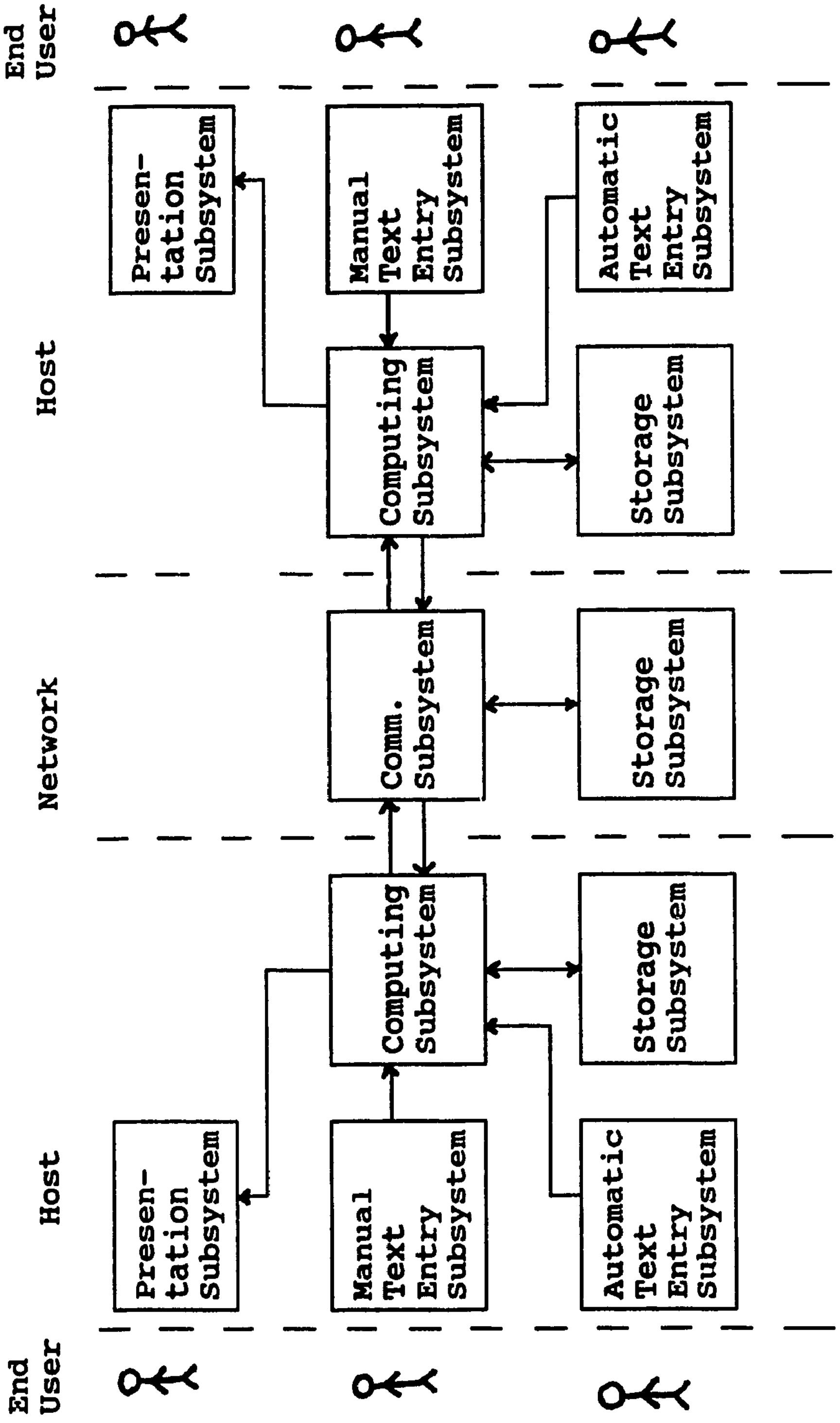


Figure 1.4 Multilingual information interchange system.

1. Text entry subsystems.
2. Presentation subsystems.
3. Computation subsystem.
4. Storage and retrieval subsystem.
5. Communication (transmission and reception) subsystem.

Till today the most popular text entry method is by typing using the keyboard. Even though it is manual, it will continue to remain popular in the future as well. However, cost of manual keyboard based text entry is raising steadily throughout the world (Schantz [1982]). For rapid and large volume text entry an automatic text entry system such as the character reader is the solution. Currently, character readers are increasingly used in the office automation, banking and postal systems in America, Europe and Japan (Suen [1983], Downton, Kabir and Guilevic [1988], Hull, et al [1988], Ciardiello, et al [1988], Nishino and Takao [1988], Banno [1988]). More and more PC based character readers for English are being introduced in the market. Use of character readers will continue to increase in the future as they become more reliable, versatile and economical.

For these reasons multilingual character readers have been selected for the automatic text entry subsystem and multilingual terminals have been selected for the manual text entry subsystem in this thesis. The design of character readers itself poses so much challenge, it has been an independent research topic for the last few decades. The techniques developed to design the various components of a multilingual character reader such as scanning and preprocessing, feature extraction, classification and language identification are described in detail in the subsequent chapters.

Even though voice based text entry and presentation systems are another alternative, voice recognition and synthesis are two separate research topics in their own domain due to their complexities. They are out of scope of this thesis, hence will not be discussed any more.

Printer and display based presentation subsystems are very popular and in wide use and will remain so for a long time. Hence, the terminal is considered for the presentation subsystem. The techniques developed in this thesis to design a multilingual terminal (text entry and presentation subsystems) are explained in detail later in this thesis.

This thesis addresses and makes original contribution towards an integrated multilingual computing and communication system with input, output, facility for manipulating multilingual texts using programming languages, multilingual document representation, storage, retrieval, transmission and reception of such documents with coherent and uniform solution. Methodologies and algorithms developed for such an integrated CCS are explained in the rest of the thesis.

1.5 Theory of Symbolization

The English alphabet has 26 letters and 52 characters (26 uppercase letters and 26 lowercase letters) for representation. A set of fifty two characters is relatively small compared to major world languages such has Indian languages, Chinese and Japanese languages, other Asian languages such as Sinhalese, Burmese, Thai, etc., Cryllic script languages and other Latin script languages. Most of the languages in the world have a large number of characters. For example, the Chinese language has more than 48,000 characters in the literature and at least 5,000 in use; Indian languages like Telugu can have theoretically an infinite number of characters and in practice about two thousand characters. The characters of these languages are more complicated than the characters of English. They also vary in size considerably. These languages are used by more than 2 billion people, a major portion of the world population.

The large number of complex characters in a language makes it impossible to design a keyboard of reasonable size by assigning one character per key position. Character generation requires a large amount of memory to store the graphemes of all the characters. Also, the knowledgebase for character recognition will be huge. In general, the computing and communication systems (CCSs) become complex, expensive and unmanageable. To handle the large number of characters in a manageable way, a new scheme called *symbolization* is proposed in this section. Its universal applicability is demonstrated using the scripts of major world languages.

The word *symbol*, as used in this thesis, is an abstract entity representing a character or part of it in a language. It may or may not have a graphical representation like the character.

1.5.1 Properties of Symbols

Let,

$$C = \{c_1, c_2, \dots c_n\}$$

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

be the set of characters and symbols, respectively, in a language, *L*. The set of symbols, *S* have the following properties:

Property 1.1: Each symbol must represent a partial or complete character in the language.

Property 1.2: There must exist an unique ordered set of symbols $\{s_1, s_2, \dots s_k\}$ for each character in the language which uniquely identifies that character.

Property 1.3: The members of the set of symbols, *S* may or may not have graphical representation. If members of *S* are derived from the graphemes of the script, they will have graphical representation. However, if they are derived from linguistic properties other than graphemes, they may not have graphical representation.

Property 1.4: The cardinality of S is normally much smaller than the cardinality of C, that is, $|S| << |C|$. If $|S| = |C|$, then each symbol uniquely represents a character in the language. In other words, if the mapping $f:S\text{-->}C$ is 1:1, then each symbol stands for one character in the language. This will be the case for languages having small number of characters like English.

Property 1.5: $|S|$ should never be greater than $|C|$.

Property 1.6: The internal or machine representation of the members of S should be such that sorting of texts should give collating sequence. This may or may not be possible for all languages.

Property 1.7: It is possible that one set of symbols may be used to represent the characters of one or more languages.

Definition 1.1: Symbolization

The process of identifying the set of symbols, S for a language is called *symbolization*.

The simplest method of symbolization is to assign one character to one symbol.

However, for languages having a large number of characters, $|S| << |C|$. If $|S|$ is too small, then the unique sequence $\{s_1, s_2, \ldots s_k\}$ identifying the character may become longer. On the other hand if $|S|$ is too large, then the above sequence will be shorter, however, use of such symbols will result in the same problems like large number of characters. Because of the trade off, identifying an optimal symbol set, S, that is symbolization, may become a challenge for certain languages.

1.5.2 Universal Applicability of Theory of Symbolization

The analysis of the scripts of major world languages has revealed that the theory of symbolization is universally applicable to major world languages. To demonstrate the universality of the theory, let us consider the languages of the following scripts:

1. Latin script
2. Cryllic script
3. Indo-Asian scripts
4. Arabic script
5. Ideographic script
6. Braille script

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

Spanish fall under this category. The characters of these languages include A to Z and a to z and a few additional characters. Some of the additional characters of the languages of this category are listed in Figure 1.5 (Gilyarevsky [1970]). A careful consideration will reveal that we could apply symbolization to these languages and identify a set of symbols as:

Ω = { A to Z, a to z and diacritical marks }

where the diacritical marks are .., ., ^, ~, ', etc. Interestingly, the small symbol set, Ω could be used to represent all the Latin script languages shown in Figure 1.5 and many more.

Cryllic Script Languages: More than fifty different languages use the Cryllic script or an extended version. The Russian language comes under this category. Most nations of the Soviet Union and 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 1.6 (Gilyarevsky [1970]). Again, a careful analysis will reveal that it is possible to identify a set of diacritical marks, like in the Latin script, for these additional character and obtain a symbol set as:

Ψ = { Russian alphabets and diacritical marks }

where the diacritical marks are .., ., -, etc. As in Latin script languages, the symbol set Ψ could be used to represent a set of Cryllic script languages, not just one.

Albanian	ç, ë
Aymara	n̈
Basque	ë, ñ, ŕ
Breton	é, è, ê, ï, î, ñ, ô, ü.
Bui	Б, a, ə, ɒ, ш, г, ч, ɵ, ь, з, ʒ
Catalan	à, ç, é, è, ï, ó, ò, ü, ú
Choctaw	ą, į, ǫ, ų
Chuana	ê, è, ô, š
Cree	â, î
Czech	á, č, ď (ď, ď, Ď), é, ě, í, ň, ó, ř, š, ť (ť, Ť), ú, ů, ý, ž
Danish	å, æ, ø
Delaware	á, é, í, ĭ, ó, ō
Eskimo	ā, ĕ, ī, ŏ, r'
Esperanto	ĉ; ĝ, ĥ; ĵ, ŝ, ŭ
Estonian	ä, õ, ö, ü
Ewe	ɖ(Ɖ), ɛ, ƒ(Ƒ), ɣ, ŋ, ɔ, ʋ; ã, ẽ, ĩ, ɔ̃, õ, (ḍ, ġ, ḥ, ṅ, ʒ̣)
Faroese	æ, ø, ð
Fiji	ā, ē, ī, o, u
Finnish	ä, ö
French	à, â, ç, ë, é, è, ê, ï, î, œ, ü, ù, û
Frisian	â, ê, î, ô, ú, û
Fulbe	ā, ɓ, ɗ, ē, ɛ ǵ ṅ, ō, ɔ, ẁ
German	ä, ö, ß, ü
Guarani	á, ã, e, ẽ, í, ĩ, ñ, ó, õ, ú, ũ, ý, ỹ
Hausa	ɓ(Ɓ), ɗ(Ɗ), ƙ(Ƙ)
Hungarian	á, é, í, ö, ó, ő, ü, ú, ű
Icelandic	á, æ, ð (Ð), é, í, ö, ó, þ, ú, ý
Irish	á, é, í, ó, ú
Italian	à, é, è, í, ì, î, ó, ò, ú, ù
Javanese	ḍ, é, è, ṭ
Juang	Б, ə, ɧ, ɵ, ъ, з, ь, ɜ, ƨ, ч, ɦ, ш
Kasubian	ą, ć, é, ē, ę, ł, ń, ó, ô, ö, ś, ų, ż, ź
Kurdish	ê, ö (in Iraq);
Lahu	ç, ê, î, ş, û, ẍ (in Syria)
Latin	ā, ă, æ, ē, ĕ, ī, ĭ, ō, ŏ, œ, ū, ŭ
Lettish	ā, č, ē, ģ, (Ģ), ī, ķ, ļ, ņ, š, ū, ž
Lingala	ɛ, ɛ́., í, ó, ɔ, ɔ́, ɔ̌, ú
Lithuanian	ą, č, ę, ė, į, š, ų, ū, ž
Lisu	Б, Д, э, Г, ŋ, з, ч, ш, ы, ξ, л, я, ф, ь, ц
Luba	ñ, (š), ž
Madura	ḍ, è, ṭ
Miao	я, э, æ, ч, ш, ŋ, ɵ, ʔ, ʃ, ʐ, н, Б, a, з, ж
Malagash	á, à, é, è, í, ì, ó, ò, ý, ỳ
Malay	ĕ
Mandingo	â, ā, ç, é, è, ẽ, ɡ̧, g, ĩ, ɉ, ṅ, ò, ɔ̀, ō, ö, ó, ɔ́,

Figure 1.5 Additional characters of Latin script languages.

Minankabaw	ŕ, ū, ǖ, ũ, u̲ ĕ, ĭ
Mohawk	à, è
Mossi	ä, ȧ, ã, ẻ, ê, ē, (ẹ), ẽ, g', ī, ĩ, ṅ, ō, (ọ), õ, p', t'
Navaho	ğ, k', ł, ń, ň, 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, â, ã, ç, ë, é, è, ê, ĩ, í, ó, ò, ô, õ, ũ, ú
Quechua	á, ñ, ó, ú
Rhaeto-Romanic	à, â, è, ê, ò, ô, û
Rumanian	ă, î, ş, ţ
Samoan	ā, ō, ū
Seneca	ẽ, œ, ũ
Serbo-Croatian	ć, č, đ (Đ), š, ž
Sioux	ą, c', ć, ç, ǧ, ȟ, į, k, k', ŋ, p., p', ś, š, ţ, ų, ź, ž
Slovak	ä, á, č, ď (ď, ď, Ď), é, í, ľ (Ľ), ň, ó, ô, ŕ, š, ť (ť, Ť), ú, ý, ž
Slovene	č, š, ž
Spanish	á, é, í, ñ, ó, ü, ú
Suto	ė, ŏ, š
Sundanese	ĕ
Swahili	ng'
Swedish	ä, å, ö
Tagalog	(ñg)
Turkish	â, ç, ğ, ı, İ', î, ö, ş, ü, û
Uolio	b, d, ñ
Vietnamese	â, ă, đ (Đ), ê, ô, o', n'; vowels in combination with diacritical signs to denote the tones:
Volapük	ä, ë, ö, ü
Welsh	á, â, ê, é, ó, ô, ỳ, ŷ, ŵ
Wendish	ć, č, ě, ł, ń, ó, ř, ź, ž
Wolof	ɔ, ñ, ı, k̃, ɑ
Ỵ	ę, (ə), ə, ч, ŋ, ө, ж, ӡ, з
Yoruba	ọ, ḥ, ị, ḳ, ū
Zulu	ɓ(Ɓ),

Figure 1.5 Additional characters of Latin script languages (contd.).

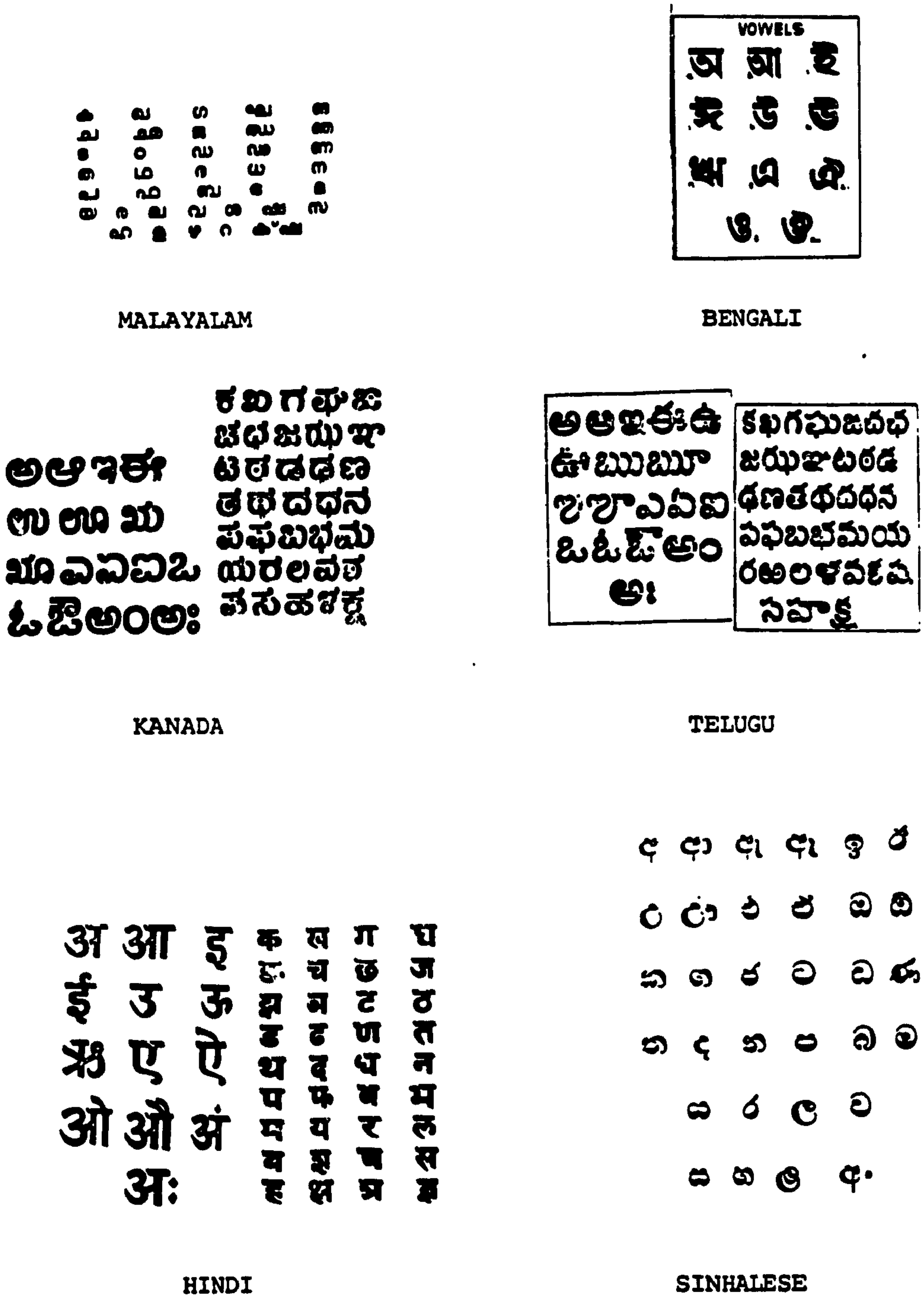
Abazin I	— Ӏ
Abkhasian	— ҕ, ҽ, ҿ, ӡ (Ӡ), қ, ҟ (Ҟ), ҩ, ҧ, ҭ, ҳ, ҵ, ҷ, ә
Adighe I	— Ӏ
Altaic	— ј, ҥ, ӧ, ӱ
Avar I	— Ӏ
Azerbaijani	— ғ, ә, ј, ҝ, ө, ү, һ, ҹ
Bashkir	— ғ, ҙ, ҡ, ң, ө, ҫ, ү, һ, ә
Buryat	— ө, ү, һ
Byelorussian	— і, ў
Chechen	— Ӏ
Chukcha	— ӄ, ӈ
Chuvash	— ӑ, ӗ, ҫ, ӳ
Dargwa	— Ӏ
Dungan	— ә, җ, ң, ў, ү
Eskimo	— ў
Even (Lamut)	— ӈ, ө, ӫ
Fvenki (Tungus)	— ӈ
Gagauzi	— ӓ, ӧ, ӱ
Ingush	— Ӏ
Kabardian (Circassian)	— Ӏ
Kalmyck	— ә, һ, җ, ң, ө, ү, (ӓ, ӧ, ӱ)
Karachai	— ў
Kara-Kalpak	— қ, ғ, ҳ, ң, ө, ү, ў, ә
Kazakh	— ә, ғ, қ, ң, ө, ұ, ү, һ, і
Khakass	— ғ, і, ӧ, ӱ, ҷ, ң
Khanty (the language of the Vakh Khanty)	— ӓ, ӄ, ң, ӧ, ө, ӫ, ӱ, ә, ӛ
Khanty (the language of the Kazym Khanty)	— ӑ, ә, ӛ, ӈ, ө, ӫ
Khanty (the language of the Surgut Khanty)	— ӓ, ӄ, ӈ, ө, ӧ, ӱ, ә
Khanty (the language of the Shuryshkar Khanty)	ӈ
Kirghiz	— ң, ө, ү
Komi-Permian	— і, ӧ
Komi (Zyryan)	— і, ӧ
Kurdish	— ә, ӧ, һ, q(Q), w
Lak	— Ӏ
Lesghin	— Ӏ
Macedonian	— ѓ, ќ, љ, њ, џ
Mansi (Vogul)	— ӈ
Mari (High)	— ӓ, ӧ, ӱ, ӹ
Mari (Low)	— ҥ, ӧ, ӱ
Mongolian	— ө, ү
Ossetic	— ӕ(Ӕ)
Serbo-Croatian	— ђ(Ђ), ј, љ, њ, ћ(Ћ), џ
Tajik	— ғ, ӣ, қ, ӯ, ҳ, ҷ
Tatar	— ә, ө, ү, җ, ң, һ
Touvinian	— ң, ө, ү
Turkmen	— җ, ң, ө, ү, ә
Udmurt	— ӝ, ӟ, ӥ, ӧ, ӵ
Uigur	— қ, ң, ғ, ү, ө, җ, ә, һ
Ukrainian	— є, і, ї
Uzbek	— ў, қ, ғ, ҳ
Yakut	— ҕ, ҥ, ө, һ, ү

Figure 1.6 Additional characters of Cryllic script languages.

Indo-Asian Script Languages: Fifteen different Indian languages and Asian languages such as Sinhalese, Burmese, Thai and so on (except Ideographic and Arabic languages) use more than ten different scripts. Devanagari, Dravidian and Pali are the most commonly used scripts. The number of characters in each of these languages vary from a few hundreds to a few thousands. The characters of these languages are more complicated than the characters of the Latin and Cryllic scripts. Figure 1.7 gives some of the basic characters of a few Indian languages.

Symbolization has been found very useful for Indo-Asian languages. As an illustration consider the complete character set of the Indian language Tamil shown in Figure 1.8. The original character set had 247 characters. Later, an additional 65 characters infiltrated into the language from other languages to accommodate the words derived from those languages. These 65 characters are not shown in the figure and are not considered for analysis. By applying symbolization, it is possible to identify a set consisting of about sixty graphic symbols to represent the 247 Tamil characters. The set is given in the next chapter. As an illustration, notice that all the 126 characters in the rectangle from columns 7 to 13 can be obtained by using the 18 characters in column 1 and by just five additional symbols ெ , ே , ை , ா , ் , a total of 23 symbols.

Such symbolization has been found applicable to other Indian languages and Asian languages such as Sinhalese, Burmese, and Thai.



MALAYALAM

BENGALI

KANADA

TELUGU

HINDI

SINHALESE

Figure 1.7 Samples of basic characters of Indian languages.

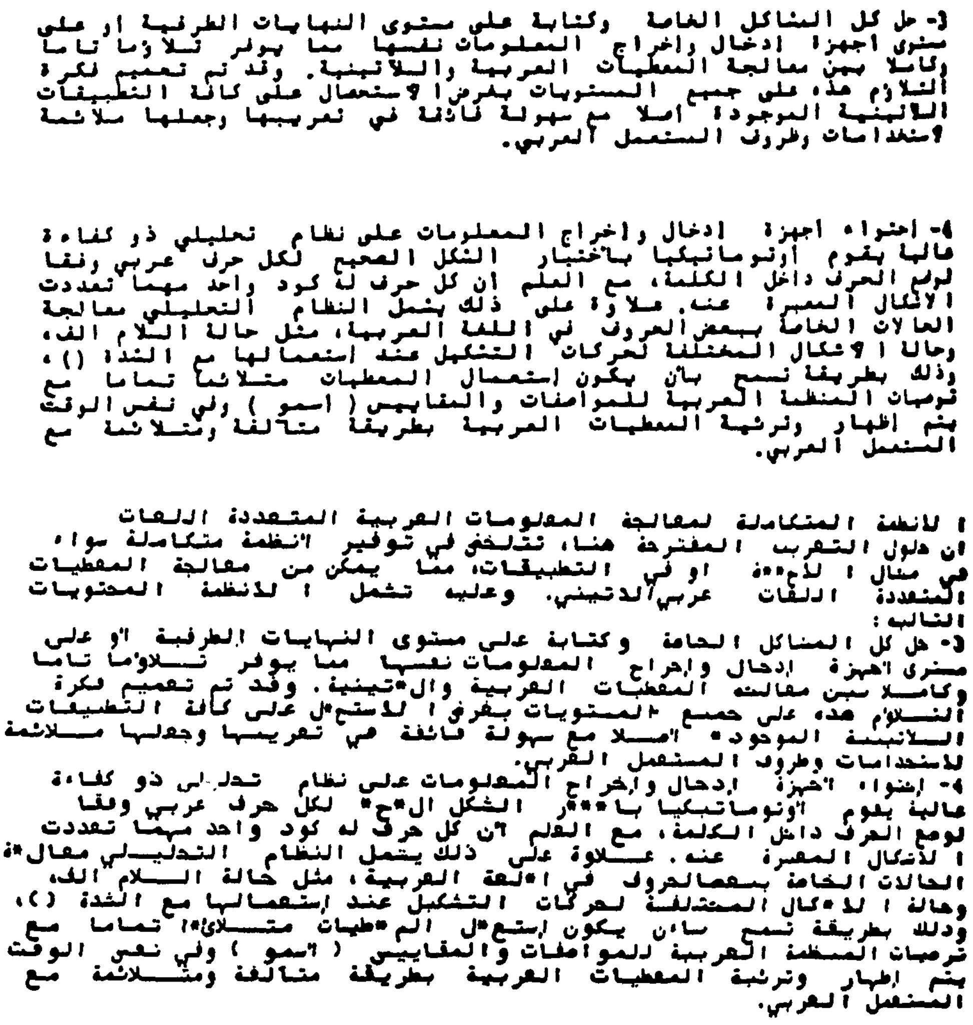
அ	ஆ	இ	ஈ	உ	ஊ	எ	ஏ	ஐ	ஒ	ஓ	ஔ	
க	கா	கி	கீ	கு	கூ	கெ	கே	கை	கொ	கோ	கௌ	க்
ங	ஙா	ஙி	ஙீ	ஙு	ஙூ	ஙெ	ஙே	ஙை	ஙொ	ஙோ	ஙௌ	ங்
ச	சா	சி	சீ	சு	சூ	செ	சே	சை	சொ	சோ	சௌ	ச்
ஞ	ஞா	ஞி	ஞீ	ஞு	ஞூ	ஞெ	ஞே	ஞை	ஞொ	ஞோ	ஞௌ	ஞ்
ட	டா	டி	டீ	டு	டூ	டெ	டே	டை	டொ	டோ	டௌ	ட்
ண	ணா	ணி	ணீ	ணு	ணூ	ணெ	ணே	ணை	ணொ	ணோ	ணௌ	ண்
த	தா	தி	தீ	து	தூ	தெ	தே	தை	தொ	தோ	தௌ	த்
ந	நா	நி	நீ	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

Figure 1.8 Tamil characters.

Arabic Script Languages: The Arabic, Urdu and Farsi languages belong to this group. These languages are wide-spread in the United Arab Republic, Iran, Iraq, Kuwait, India and other Arabic countries. A sample text is shown in Figure 1.9 (Amin [1988]). The complexity of the characters of these languages falls between the Indo-Asian languages and the Ideographic languages. Again, we could see that symbolization could be applied to the characters of these languages even though it is relatively complex.

Ideographic Script Languages: Chinese, Japanese, Korean, etc., are examples of ideographic languages. Chinese language has about 48,000 characters; around 5,000 characters are used in writing and about 3,000 in printing (Shyu, et al [1988]). 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 Chinese text is shown in Figure 1.10.

Keyboarding techniques have been proposed to input Chinese characters based on the phonetic characteristics (Kung [1983], Dong [1986], Cheng and Yu [1987], Yuhang [1987]), and stroke-sequences (Hung [1987]). Such phonetic and/or the stroke entities could be treated as symbols for Chinese characters. Thus, symbolization is applicable to Ideographic languages too.

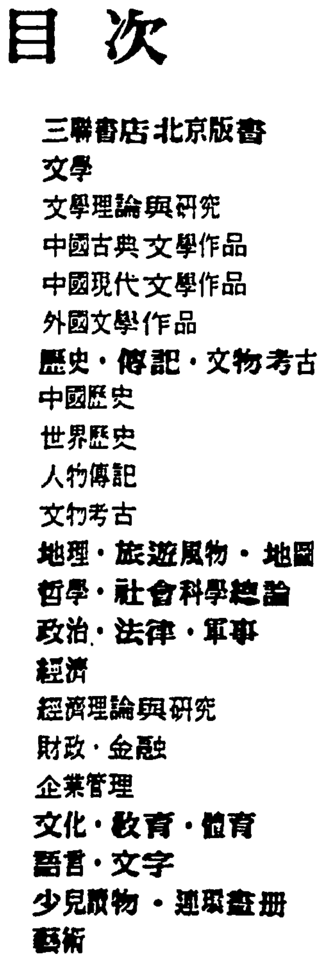


٣- حل كل المشاكل الخاصة وكتابة على مستوى النهايات الطرفية او على
مستوى اجهزة ادخال واخراج المعلومات نفسها مما يوفر تلازما تاما
وكاملا بين معالجة المعطيات العربية واللاتينية. وقد تم تعميم لفكرة
التلازم هذه على جميع المستويات بغرض الاستعمال على كافة التطبيقات
اللاتينية الموجودة اصلا مع سهولة فائقة في تعريبها وجعلها ملائمة
لاستخدامات وظروف المستعمل العربي.

٤- احتواء اجهزة ادخال واخراج المعلومات على نظام تحليلي ذو كفاءة
عالية يقوم اوتوماتيكيا باختيار الشكل الصحيح لكل حرف عربي وفقا
لوضع الحرف داخل الكلمة، مع العلم ان كل حرف له كود واحد مهما تعددت
الاشكال المعبرة عنه. علاوة على ذلك يشمل النظام التحليلي معالجة
الحالات الخاصة ببعض الحروف في اللغة العربية، مثل حالة اللام الف،
وحالة الاشكال المختلفة لحركات التشكيل عند استعمالها مع الشدة ()،
وذلك بطريقة تسمح بان يكون استعمال المعطيات متلائما تماما مع
توصيات المنظمة العربية للمواصفات والمقاييس (اسمو) وفي نفس الوقت
يتم اظهار وترتيبة المعطيات العربية بطريقة متآلفة ومتلائمة مع
المستعمل العربي.

ا لانظمة المتكاملة لمعالجة المعلومات العربية المتعددة اللغات
ان حلول التعريب المقترحة هنا، تتلخص في توفير انظمة متكاملة سواء
في مثال ا لاج**ة او في التطبيقات، مما يمكن من معالجة المعطيات
المتعددة اللغات عربي/لاتيني. وعليه تشمل ا لانظمة المحتويات
التالية :
٣- حل كل المشاكل الحاصة وكتابة على مستوى النهايات الطرفية أو على
مستوى أجهزة إدحال وإحراج المعلومات نفسها مما يوفر تلاؤما تاما
وكاملا سبن مفالنه المعطيات العربية وال*تينية. وقد تم تعميم لكرة
التلاؤم هده على حسع *المستويات بغرض ا لاستع*ل على كافة التطبيقات
اللاتينية الموحود* أصلا مع سهولة فائقة في تعريبها وجعلها ملائمة
للاستخدامات وظروف المستعمل العربي.
٤- إشتواء أجهزة إدحال وإحراج المعلومات على نظام تحليلي ذو كفاءة
عالية يقوم أوتوماتيكيا با**ر الشكل ال*ح* لكل حرف عربي وفقا
لوضع الحرف داخل الكلمة، مع العلم أن كل حرف له كود واحد مهما تعددت
ا لأشكال المعبرة عنه. علاوة على ذلك يشمل النظام التحليلي معال*ة
الحالات الخاصة ببعضالحروف في ا*لغة العربية، مثل حالة اللام الف،
وحالة ا لا*كال المختلفة لحركات التشكيل عند إستعمالها مع الشدة ()،
وذلك بطريقة تسمح بأن يكون إستع*ل الم*طيات متلائ*ا تماما مع
ترصيات المنظمة العربية للمواصفات والمقاييس (أسمو) وفي نفس الوقت
يتم إظهار وترتيبة المعطيات العربية بطريقة متآلفة ومتلائمة مع
المستعمل العربي.

Figure 1.9 Sample Arabic text.



目次

三聯書店北京版書
文學
文學理論與研究
中國古典文學作品
中國現代文學作品
外國文學作品
歷史・傳記・文物考古
中國歷史
世界歷史
人物傳記
文物考古
地理・旅遊風物・地圖
哲學・社會科學總論
政治・法律・軍事
經濟
經濟理論與研究
財政・金融
企業管理
文化・教育・體育
語言・文字
少兒讀物・連環畫册
藝術

Figure 1.10 Sample Chinese text.

Braille Script: Braille is a system of raised characters formed by using combinations of six dots in three rows and two columns as shown in Figure 1.11. The dots represent the projections (so that blind people can read by touching) on the output paper. The Braille script uses the English alphabets and differs only in the presentation of the characters on the output medium. Hence, whatever symbolization scheme that is applicable to English could be applied to Braille script also. In Figure 1.11 the character strings *and, for, of, the, with, ch, gh, sh and th* have more than one English character, each could be treated as a single character in Braille script.

1.6 Outline of the Thesis

The next three chapters describe the design of a multilingual character reader that could be used with the CCSs. Preprocessing techniques, namely binarization and thinning are presented in Chapter two. The structural features used for classification, their extraction and the scheme developed for representation of symbols and characters are explained in Chapter three. In Chapter four, a two stage classification scheme developed to recognize the characters from symbols is described and the results are presented.

Chapters five through seven explain the design of a multilingual terminal that can be attached to a CCS or used as a stand-alone typewriter. The techniques developed for the determination of nearly optimal dot matrix character size and the

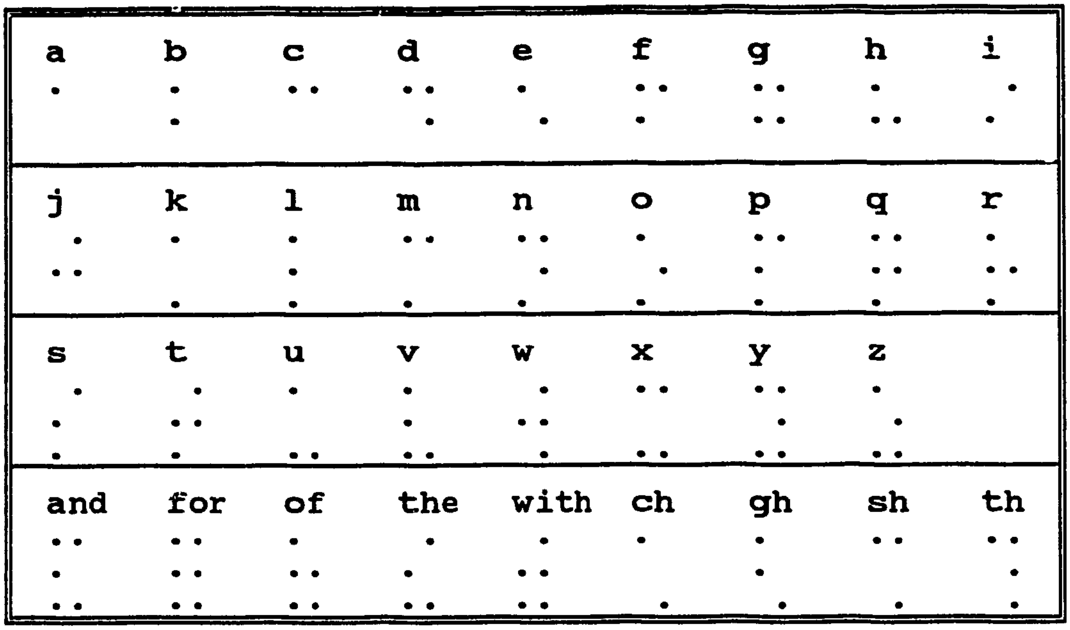


Figure 1.11 Braille script.

determination of optimal dot matrix symbol set are given in Chapters five and six respectively. The multilingual keyboard design and character generation techniques proposed are presented in Chapter seven.

Chapter eight presents the scheme developed for designing a multilingual data communication system that can be configured with the multilingual character reader and the multilingual terminal developed earlier. The techniques developed for multilingual document representation, character coding, storage and retrieval of documents, and the protocol changes needed are described. A scheme for handling multilingual texts using the existing programming tools such as compilers with minimal change is also shown in this chapter. Chapter nine concludes the thesis.

The Indian languages Tamil and Malayalam are used for illustration and experiments throughout the thesis. The symbolization technique is uniformly employed for the design of multilingual character readers, terminal and the computing and communication systems.

1.7 Illustration

Indian scripts and languages (in particular Tamil and Malayalam) are used for illustration wherever needed for the following reasons:

1. the representative nature of the Indian scripts and languages to the

world situation (India has fifteen different languages and ten different scripts recognized by the constitution),

2. a considerable amount of work has to be done to handle Indian scripts,
3. there is a great need and local demand that CCSs be made to handle Indian scripts, and
4. familiarity of the Indian languages to the author.

However, it should be noted that the methodologies and algorithms developed in this thesis are general and could be used for other languages too.

This thesis covers several areas of computer science: pattern recognition, communication, and input/output subsystems. In addition, it includes natural languages and scripts. Each is an independent research area. For example, several dissertations have been submitted just on character recognition alone. A thesis having width may lack depth. However, in this thesis depth is not sacrificed for width of coverage. Each chapter that follows has some new contributions to the knowledge of computer science.

The thinning algorithm of Stefannalli and Rosenfeld [1971] has been extended to eliminate variable line thickness. A new scheme has been proposed to extract features from thinned characters. The recognition methodology and algorithms developed in this thesis will enhance the knowledge of pattern recognition, in particular, the structural approach to character recognition. Also, the thesis contributes to the knowledge of systematic and scientific approach to the design of ergonomic keyboard based multiscript terminal system for Indian languages. A new knowledge based thinning algorithm has been developed for the determination of character size for the design of output devices. The pattern recognition based character size determination and font design methods simplify the task of developing fonts and reduces the time required for font development. An information content based scheme has been proposed for the evaluation of different character fonts. The unified approach to multilingual data communication scheme developed in this thesis adds to the knowledge of designing communication networks for the multilingual nations of today and the world of tomorrow. The scripts and characteristics of several natural languages have been studied and analyzed for these purposes.

The proposed methodologies and algorithms have been tested by conducting experiments using the scripts of Indian languages. The new results obtained have been reported in this thesis.

Note: The usage of the following words should be interpreted only as defined in this thesis. The words *correlation* and *correlation coefficient* are used in this thesis not as in Statistics (mathematics). They are used more in the (American Heritage) dictionary meaning of "*correspondence (closeness) between two comparable entities*". Similarly the words *Hamming distance*, *self-information* and *entropy* should be interpreted only as defined in this thesis and not as in the general use in mathematics or other science and engineering. The definitions of these words, as used in this thesis, are given in the appropriate chapters.

2. MULTILINGUAL CHARACTER READER

2.1 Introduction

More and more organizations, business and non-business, are moving towards automation in networking for interconnectivity and instant access to information using computers to reduce increasing labor cost. The human race is generating more information than ever before not only in one language but in several languages. This has created great demands for faster and less expensive mechanisms to generate, acquire, transmit, store, access and present texts that represent information. The cost of storing and processing texts is declining steadily over the years, however, the cost of manual data entry has been rising and it will continue to rise (Schantz [1982]).

Nowadays fax machines are used for transmitting documents on paper by scanning them on the sending end and transmitting the digitized image. It is reproduced on the receiving end, may be with some loss of quality. However, the digitized image requires several orders of magnitude more storage and transmission time compared to sending the textual document in character codes. This increases the storing, processing and networking cost. Another disadvantage with image transmission of text is that it cannot be used to synthesize voice output, if necessary, from the text. The character readers are an ideal solution to solve this problem.

The origin of character recognition (Anderson [1969]) dated back to 1870 when Varey invented the retina scanner, that is an image transmission system using a mosaic of photocells, and later in 1890 when Nipkow invented the sequential scanner which was a major breakthrough both for modern television and reading machines. However, character recognition first appeared as an aid to the visually handicapped, and the first successful attempts were made by the Russian scientist Tyurin in 1900. The next attempts that have been reported are the Fourier d'Albe's Optophone of 1912 and Thomas' tactile "relief" device of 1926. The historical development of some of these machines, covering the period from 1800 to 1980 are presented in Schantz [1982].

Later versions of OCR (Schantz [1982]) appeared in the mid 1940 with the development of the digital computers. For the first time, OCR was realized as a dataprocessing approach, with particular applications to the business world. Machine simulation of human reading has been the subject of intensive research for more than three decades. Several books on OCR (Gonzalez [1978], Fu [1982], Goos and Hartmanis [1988]), as well as special reports on OCR (Schurmann [1982], Schantz [1982]) have been published. Additionally, extensive bibliographies on OCR (Tappert, Suen and Wakahara [1990], Suen [1978], Suen, et al [1980], Mantas [1986]) have been compiled. An extensive survey covering major databases, recognition techniques and their comparative performances can be found in Suen, et al [1986].

Reasonable amount of success has been obtained and now commercial character reading machines are available. The basic problem confronting a character recognition device is the following: given a set of prototype characters and an input character from the above set, the task is to assign the input character to one of the prototype characters. This approach, called the classificatory approach, treats a picture as a single atomic entity. Such approaches often fail when the input picture is more complex than alphanumeric characters. Inadequacies, like lack of facility to include structural information, etc., of such models have been studied by scientist (Narasimhan and Reddy [1971], Fu [1982], Suen [1986]). To overcome the inadequacies of classificatory approach, the linguistic (or syntactic) approach has been proposed (Narasimhan [1966], Fu [1982], Wang [1988b]).

In the last two decades linguistic (descriptive) method of picture processing has gained popularity. In the linguistic approach, the input picture is assumed to belong to a class of admissible pictures, the admissibility criteria being specified by a generative grammar. Such a grammar is used to analyze the input picture belonging to the admissible class and to generate a structured description of the input picture.

Narasimhan (Narasimhan [1966]) is the pioneer who suggested and applied the linguistic model for solutions of non-trivial problems in picture processing. His model is restricted to the class of pictures containing thin line-like elements. Many attempts have been made by scientists in this area to develop syntax-directed recognition procedures for picture processing languages, analogous to the syntax-directed compilers for programming languages. Narasimhan has given a syntax directed interpretation for a specific class of pictures.

Narasimhan and Reddy [1971] have described syntax-aided techniques for picture processing as applied to the recognition of handprinted English letters. In the syntax-directed techniques the grammatical rules are exclusively used to parse the input picture as in the case of syntax-directed compilers of programming languages. But in the syntax-aided technique, the syntactic information is used as an aid along with other contextual information while analyzing the picture. They have argued that only syntax-aided systems can cope with the real life situations. Kanal and Chandrasekaran [1972], and Fu [1983] have critically examined the various aspects and claims of grammatical approaches. Regarding the suitability of the different approaches to pattern recognition problems, they are of the opinion that a good pattern recognition system has as an integral part, a versatile tool kit and uses statistical, linguistic and heuristic tools in various stages of processing of the patterns, with each tool being applied at the stage for which it seems most suited.

Considerable amount of research and development have been carried out in the recognition of English and numerals (Le Cun, et al [1990], Suen, Nadal, Mai, Legault and Lam [1990], Krzyzak, Dai and Suen [1990], Nadal, Legault and Suen [1990], Bozinivic and Srihari [1989], Stringa [1989], Lam and Suen [1988], Hu [1982], Ahmed [1986], Huang [1986], Shridhar [1986], Mantas [1986]). Syntactic and statistical approaches have been used in the recognition of Chinese characters (Suen [1982], Wang and Suen [1984], Nagy [1988], Xia and Sun [1990]). The extent of work done in the recognition of Chinese characters can be found in the proceedings of the International Conferences on Chinese and Oriental Language Computing from 1982 to 1990 and in the survey by Nagy [1988]. Several researchers (Akamatsu [1983], Arakawa [1983], Hagita [1983], Izaki [1983], Suen [1983], Kurakake [1988], Yamada [1988] have developed methods to recognize Kanji characters both on-line and off-line. A detailed review of the state of the art of on-line handwriting recognition can be found in the paper by Tappert, Suen and Wakahara [1990].

Amin [1984], Amin and Masini [1986], Amin [1988], Parhami and Taraghi [1981], and Taraghi [1978] have developed techniques to recognize Arabic and Farsi texts. Mantas tried to recognize Greek characters. Kushnir has written a dissertation (Kushnir [1983]) and several articles (Kushnir, et al [1982, 1983, 1985]) on the recognition of Hebrew characters.

Very little work has been done on the recognition of Indian language characters which are more complex in structure and larger in number compared to any of the European languages, e.g. Telugu (Rajasekaran [1977]), Devanagari (Sinha [1973, 1984], Sethi and Chatterjee [1977], Bansal and Metha [1988]), Bengali (Som and Nath [1977]) and Tamil (Gift Siromoney, et al [1978]).

The recognition problem may be divided into a number of subproblems. Rajasekaran and Deekshatulu [1977] have proposed a two stage recognition system for Telugu. In the first stage a given character is split into primitives and basic letters using a directed curve tracing method; based on the result of first stage, classification is achieved by means of a decision tree in the second stage. Sethi and Chatterjee [1977] have adopted a multistage decision process where each stage of the decision narrows down the choice regarding the class membership of the input token which is constrained handprinted Devanagari alphabet (vowels and consonants only). The recognition scheme uses loop and line-like primitives. Gift Siromoney and Chandrasekaran [1978] have chosen the less stylized Barathi Antique font type printed Tamil alphabets for recognition study. A given character is hand coded into a 0-1 binary matrix and fed to the recognizer. It is further coded into a small string depending on the frequency runs of "1's" in both columns as well as rows. This string is compared with the stored string patterns for recognition.

Most of the work done in the character recognition area were confined to unilingual characters. Not much effort has been spent in developing multilingual OCR systems. The work reported here is an attempt to develop a multilingual character recognition system using the linguistic approach. It is expected that the following study of the linguistic approach as applied to the recognition of handprinted Tamil and Malayalam characters would contribute to the larger field of multilingual picture processing.

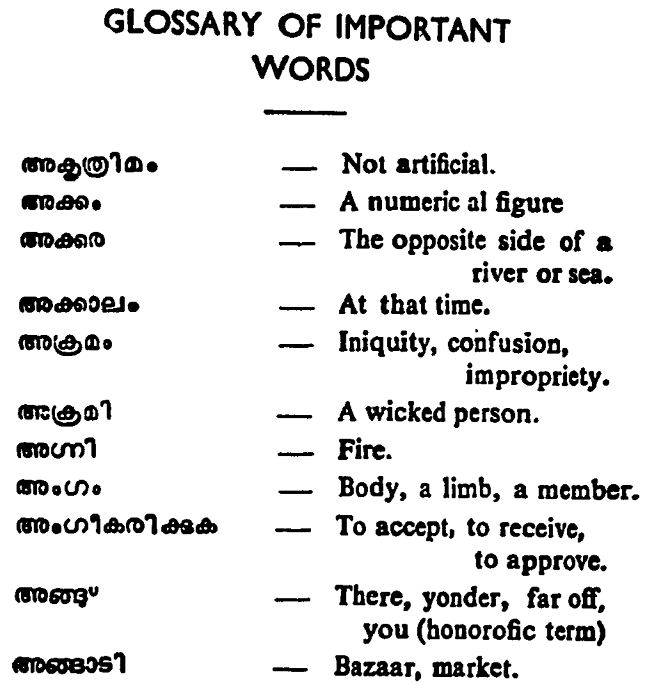
2.2 Representation of Multilingual Texts

For a multilingual OCR system to work properly, it is necessary that the system knows how multilingual texts are presented to it. The language to which a text belongs could be specified either explicitly or implicitly as explained below.

2.2.1 Implicit Specification

One way to represent multilingual texts is to freely use the text of appropriate languages as needed in a multilingual document as shown in Figure 2.1. In this case, the character set for the OCR is the union of the characters of the languages supported by the system. The cardinality of such a set could be very large, particularly, when several languages are supported. In addition, it has the disadvantage of not being able to uniquely determine the language to which a particular text belongs because there are languages that use the same characters. Without the knowledge of the language it is not possible to use the text for voice synthesis. Absence of language code will make a multilingual OCR more complex and less modular; it will also take more time to recognize a character.

Human readers identify the language by apriori knowledge, for example, knowing a dictionary as English-French. Similarly, an OCR system could be instructed about the text to be recognized which will limit the knowledgebases to be searched.



GLOSSARY OF IMPORTANT WORDS

അകൃത്രിമം	— Not artificial.
അങ്കം	— A numeric al figure
അക്കര	— The opposite side of a river or sea.
അക്കാലം	— At that time.
അക്രമം	— Iniquity, confusion, impropriety.
അക്രമി	— A wicked person.
അഗ്നി	— Fire.
അംഗം	— Body, a limb, a member.
അംഗീകരിക്കുക	— To accept, to receive, to approve.
അങ്ങു്	— There, yonder, far off, you (honorofic term)
അങ്ങാടി	— Bazaar, market.

Figure 2.1 Implicit representation of language.

In addition, the following algorithm could be applied:

Algorithm 2.1: Identification of Implicit Script Code

Step 1: Process a character and identify to which script the character belongs, say it belongs to script α. (This may involve searching several knowledgebases.) Set this script as the current script for further recognition.

Step 2: To recognize the subsequent characters search the knowledgebase of script α first. If the system fails to recognize the character, then go back to Step 1.

This algorithm will work with the possibility of increased error rate and computation time.

2.2.2 Explicit Language Specification

The disadvantages of implicit specification can be avoided by explicitly specifying the language code as part of the text as

<lid> <text>

where *<lid>* is the language identifier of *<text>*. In this scheme, the language identifier precedes the text belonging to a particular language. The *lid* consists of two parts:

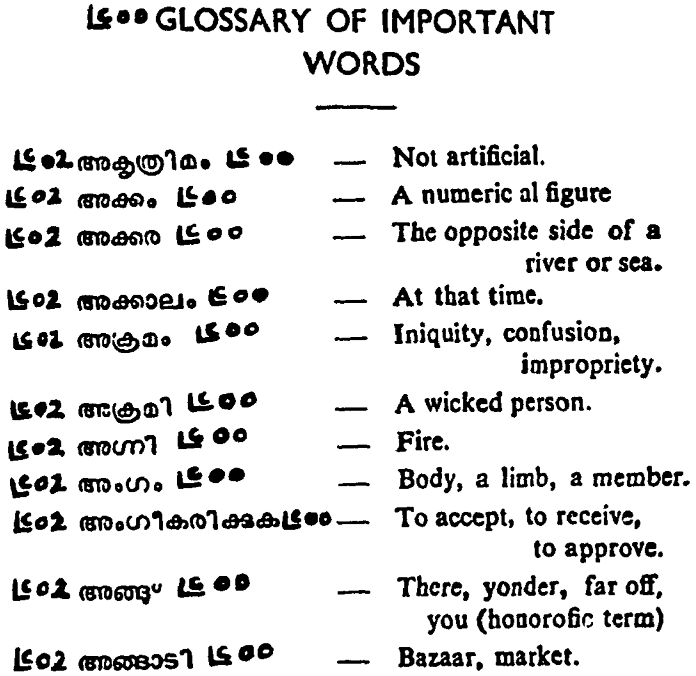
<lc> *<n>*

where *<lc>* is the language code that indicates the occurrence of the change of language and *n* is the language number. Each language is assigned a unique integer number of, say 2 digits. For example, 00 for English, 01 for Tamil, 02 for Malayalam, 03 for French and so on.

To specify the language code in the written form, it is necessary to invent a unique symbol that does not appear as a character in the languages supported by the system. After carefully studying several world languages the symbol LC has been designed to designate the language code. Let us call it *language selection symbol*. The language number follows the language selection symbol. A sample English-Malayalam multilingual text with language code is shown in Figure 2.2.

This scheme will work effectively even if multilingual texts having the same script are specified. The disadvantages are: (1) the existing multilingual texts do not have the explicit language code specified, (2) people have to learn to read with the new convention which should be comparatively easy, and (3) publishers have to accept this scheme.

In this scheme, the recognizer will only search the part of the knowledgebase corresponding to the current language code and not the entire knowledgebase, thus



GLOSSARY OF IMPORTANT WORDS

അകൃത്രിമം	— Not artificial.
അക്കം	— A numeric al figure
അക്കര	— The opposite side of a river or sea.
അക്കാലം	— At that time.
അക്രമം	— Iniquity, confusion, impropriety.
അക്രമി	— A wicked person.
അഗ്നി	— Fire.
അംഗം	— Body, a limb, a member.
അംഗീകരിക്കുക	— To accept, to receive, to approve.
അങ്ങു്	— There, yonder, far off, you (honorofic term)
അങ്ങാടി	— Bazaar, market.

Figure 2.2 Explicit representation of language code.

reducing recognition time. If the current character does not belong to the current language and is not the language selection symbol, the recognizer will reject it. (Misrecognition or rejection of language code will result in increased error rate at which point the system could be programmed to notify the operator for human intervention.)

Due to the generality of the explicit language specification, this scheme has been used in this thesis. (It is not difficult to make an OCR system to work with both explicit and implicit methods of specifying the language code.) The texts of Tamil and Malayalam are used for testing and illustration purposes.

2.3 OCR System Components

The block diagram of the multilingual OCR system is shown in Figure 2.3. The OCR system consists of

1. digitizer
2. pre-processor
3. feature extractor
4. two stage (symbol and character) classifier

The digitizer which is an optical scanner scans and digitizes the input characters into a multilevel gray scale images. The digitized image has 16 gray levels.

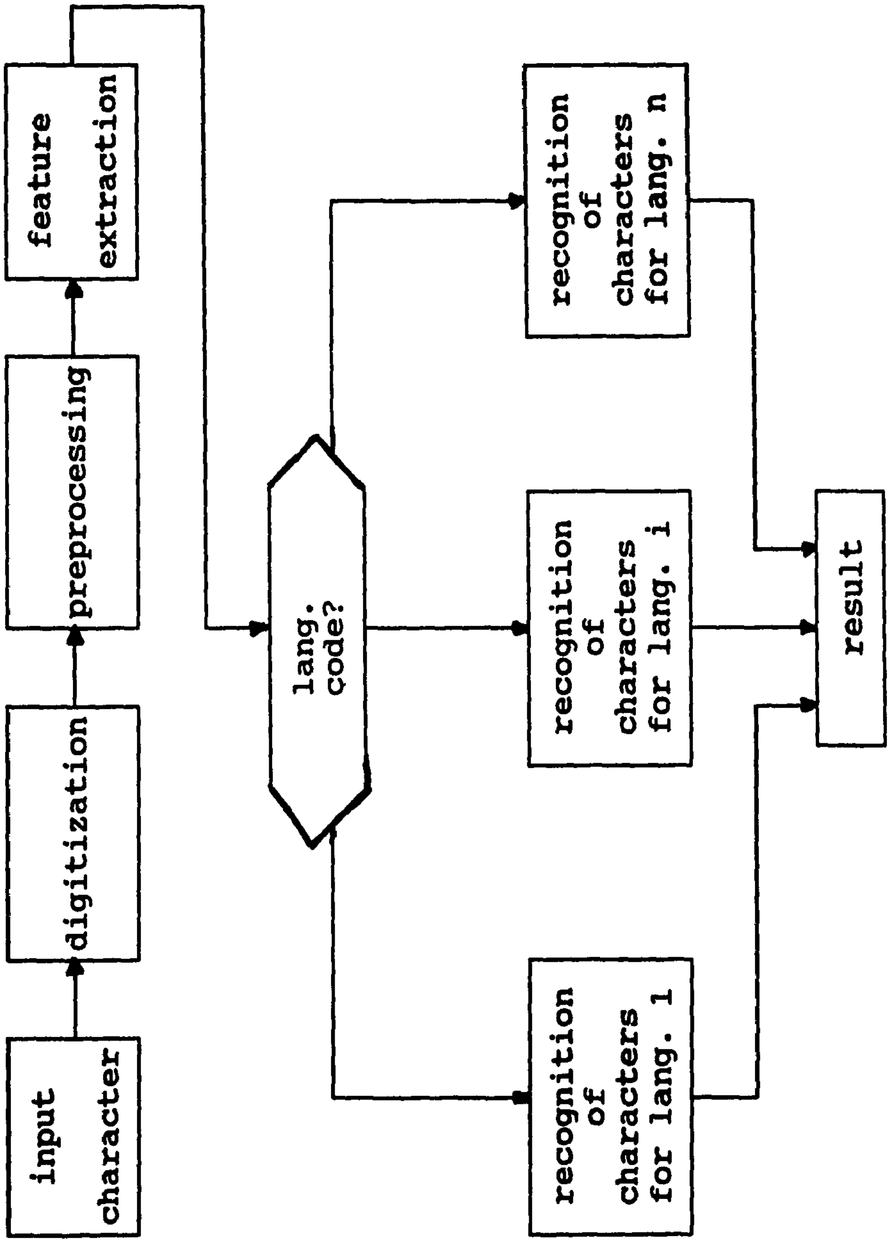

input character
digitization
preprocessing
feature extraction
lang. code?
recognition of characters for lang. 1
recognition of characters for lang. i
recognition of characters for lang. n
result

Figure 2.3 Block diagram of a multilingual OCR.

Both printed and handprinted Tamil and Malayalam characters have been used for digitization. The texts are written/printed on a paper.

In the preprocessing stage, the gray level image is binarized, and some noise such as isolated pixels and single pixel gap are eliminated. It is assumed that the characters and some symbols appear separately without touching as defined in the language. The feature extractor expects the image to be in a line-like form and not thick. Hence, the binary image is thinned by the preprocessor before feeding it to the feature extractor. The preprocessor is explained in detail in the following section.

The set of features, their extraction and the representation of the characters in the knowledgebase are explained in the subsequent chapters.

2.4 Preprocessing

Preprocessing consists of binarization, noise elimination and thinning. They are explained below.

2.4.1 Binarization

The multi-gray level image from the scanner is binarised into a two level (background and object) bit matrix. The background region is represented by 0 and the object (character) region by 1. In binarization techniques a threshold λ is selected; then, all gray levels greater than the threshold λ are assigned to character region and others to the background region. The threshold could be selected based on the gray levels of the pixels in the image (Pal, et al [1983], White and Rohrer [1983], Pal and King [1981]). It is called global technique. It could be based on the gray level of the pixel p_{ij} and the gray levels of a window of certain size around p_{ij} (Rosenfeld and Smith [1981], Ahmed [1986]). This method is called local technique. It could also be based on the location (i,j) of p, the window, and the gray level of p (White and Rohrer [1983]). This method is called dynamic method. A survey on thresholding techniques could be found in (Weszka [1978]).

In this thesis, the global technique has been used with a threshold value of 8. The pixels having a threshold value less than 8 are assigned 0 and the rest 1.

2.4.2 Elimination of Noise

The binarized output obtained by the above method is likely to contain noise. Simple noise cleaning operations such as filling single pixel gaps and removing isolated pixels are performed using Algorithm 2.2 and Algorithm 2.3 respectively. The following notation and definitions are used in explaining the algorithms.

Notation

A pixel p_{ij} has eight neighbors as shown in Figure 2.4a. For simplicity, they are represented by their relative coordinates with respect to the pixel p_{ij} as shown in Figure 2.4b. For example, the pixel, $p_{i,j+1}$ is represented as $(i,j+1)$ and the pixel, p_{ij} is represented as (ij).

Definition 2.1: Isolated Pixel

A pixel p_{ij} is said to be isolated if $(i-1, j-1) = (i-1, j) = (i-1, j+1) = (i, j-1) = (i, j+1) = (i+1, j-1) = (i+1, j) = (i+1, j+1) = 0$ and $(ij) = 1$.

Definition 2.2: Single Pixel Gap

A pixel p_{ij} is said to be single pixel gap if $(i-1, j-1) = (i-1, j) = (i-1, j+1) = (i, j-1) = (i, j+1) = (i+1, j-1) = (i+1, j) = (i+1, j+1) = 1$ and $(ij) = 0$.

Algorithm 2.2: Single Pixel Gap Filling

```
procedure single_pixel_gap_filling;
   declare  P : input binary image;
begin
  $\forall p_{ij} \in P$
      if $p_{ij}$ is single_pixel_gap then
          $p_{ij} = 1$;
end; /* single pixel gap filling */
```

n	n	n
n	p	n
n	n	n

(a) Eight neighbors of a pixel p_{ij}.

(i-1,j-1)	(i-1,j)	(i-1,j+1)
(i,j-1)	(i,j)	(i,j+1)
(i+1,j-1)	(i+1,j)	(i+1,j+1)

(b) Representation of neighbors of p_{ij}.

Figure 2.4 Neighbors of a pixel p_{ij} and their representation.

Algorithm 2.3: Elimination of Isolated Pixels

```
procedure eliminating_isolated_pixel;
    declare  P : input binary image;
begin
    $\forall\ p_{ij} \in P$
        if $p_{ij}$ is isolated_pixel then
            $p_{ij} = 0$;
end; /* eliminating_isolated_pixel */
```

Binary character images with and without the above types of noise are shown in Figures 2.5 and 2.6 respectively.

2.4.3 Thinning

The image of the input symbol will have unspecified line thickness as in Figure 2.7. Features can be extracted by processing all the pixels in the images with varying line thickness (Ahmed [1986]). Features can also be extracted from images having constant line thickness. Both the methods have pros and cons. The later method needs the input image to be thinned to extract the features. However, it is easier and takes less time to extract features than the former method since the number of pixels to be processed by the feature extractor is reduced. The later method is used in this thesis. It has been used by several others (Suzuki and Mori [1990], Lam and Suen [1989], Cheng and Hsu [1987], Zhang and Wang [1988], Wang and Zhang [1989], Xia [1988], Zhang and Suen [1984]) as well.

```
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000011110011111101111111100000000000000000000
000000000000000001111111111111111111111110000000000000000000
000000000000000111111111111111111111111110000000000000000000
000000000000000111111111111111101111111110000000000000000000
000000000000001111111111111111111111111110000000000000000000
000000000000001111111111111111111111111000000000000000000000
000000000000001111111111111111111111000000000000010000000000
000000000000000111111110001111111000000000000000000000000000
000000000000000111111100000111111000000000000000000000000000
000100000000000111111100000111111000000000000000000000000000
000000000000000111111110000111111100000000000000000000000000
000000000000000111111110000111111100000000000000000000000000
000000000000000011111111111111111111111111110000000000000000
000000000000001111111111111111111111111111111000000000000000
000000000000011111111111111111111111111111111000000100000000
000000000000111111111111111111111111111111111100000000000000
000000000001111111111111111111111111111111111100000000000000
000000000011111111110111111111111111111111111100000000000000
000000000011111110000000000011111110000111111100000000000000
000000000011111100000000000011111110000111111100000000000000
000000001011111100000000000011111110000111111100000000000000
000000000011111100000000000011111110000111111100000000000000
000000000011111110000000000111111110000111111110000000000000
000000000011111110000000000111111110001111111100000000000000
000000000001111110000000000111111110011111111100000000000000
000000000001111111111111111111111110011111111100000000000000
000000000001111111111111111111111100011111111100000000000000
000000000001111111111111111111111100011111111000001000000000
000000000000011111111111111011110000111111111000000000000000
000000000000001111111111111111000000111111110000000000000000
000000000000000011111111111100000000000000000000000000000000
000000000000000000011111111000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
000000000000000000000000000000000000000000000000000000000000
```

Figure 2.5 Image with single gap and isolated pixel noises.

```
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000111100111111111111111110000000000000000000000
0000000000000000011111111111111111111111111000000000000000000000
0000000000000001111111111111111111111111111000000000000000000000
0000000000000001111111111111111111111111111000000000000000000000
0000000000000011111111111111111111111111111000000000000000000000
0000000000000011111111111111111111111111000000000000000000000000
0000000000000011111111111111111111111000000000000000000000000000
0000000000000001111111100011111110000000000000000000000000000000
0000000000000001111111000001111110000000000000000000000000000000
0000000000000001111111000001111110000000000000000000000000000000
0000000000000001111111100001111111000000000000000000000000000000
0000000000000001111111100001111111000000000000000000000000000000
0000000000000000111111111111111111111111111100000000000000000000
0000000000000011111111111111111111111111111111100000000000000000
0000000000000111111111111111111111111111111111110000000000000000
0000000000001111111111111111111111111111111111111000000000000000
0000000000011111111111111111111111111111111111111000000000000000
0000000000111111111111111111111111111111111111111000000000000000
0000000000111111111000000000111111110000111111111000000000000000
0000000000111111110000000000111111110000111111111000000000000000
0000000000111111100000000000111111110000111111110000000000000000
0000000000111111100000000000111111110000111111110000000000000000
0000000000111111110000000000111111110000111111111100000000000000
0000000000111111110000000001111111110001111111111000000000000000
0000000000011111111000000001111111110011111111111000000000000000
0000000000011111111111111111111111110011111111111000000000000000
0000000000001111111111111111111111100011111111100000000000000000
0000000000001111111111111111111111000111111111100000000000000000
0000000000000111111111111111111100001111111111000000000000000000
0000000000000111111111111111111000000111111110000000000000000000
0000000000000001111111111111110000000000000000000000000000000000
0000000000000000000111111111000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000
```

Figure 2.6 Image without single gap and isolated pixel noises.

```
              1111111111111111
          1111111111111111111111111111111111111111111
          1111111111111111111111111111111111111111111
         111111            11111111111111111111111111
        1111                        111
       111                          111
      111                           111
    1111                            111
   1111                             111
  1111                              111
  1111                              111
  1111                             1111
 11111                             111
 1111                              111
 1111                              111
 1111                              111
 1111                              111
1111                               111
1111                               111
1111                               111
111                                111
111                                111
111                                111
111      1111                      111
 111       111                      111
 111        111                     111
 111         111                    111
 111          111                   111
 111           111                  111
  111          1111                 111
  111           111                 111
  111           111                 111
  111           111                 111
   111          111                 111
   111          111                 111
   111          111                 111
    111         111                 111
    111        111                  111
    111        111                  111
     111      111                   111
     111      111                   111
     111     111                    111
     111    111                     111
      111  111                      111
      111 111                       111
        1111
```

Figure 2.7 A sample input pattern to thinning algorithm.

A parallel thinning algorithm has been developed to thin the characters. It is an extension of the thinning algorithm by Stefanalli and Rosenfeld [1971]. Application of Stefanalli and Rosenfeld's algorithm, sometimes, gives thinned characters with a line thickness of 2. For example, the result of application of Stefanalli and Rosenfeld's thinning algorithm to the pattern in Figure 2.7 is shown in Figure 2.8. The extended algorithm described below always gives line thickness of 1. The following notation and definitions are used in explaining the thinning algorithm and the feature extractor.

Notation

Very often pictorial conditions in the form of a 3x3 cellular array are required. The pictorial conditions can be explained best with an example. Consider the pictorial condition given in Figure 2.9. The condition states that the pixel (ij) should be "1", the neighbors $(i-1,j-1)$, $(i+1,j-1)$ and $(i+1,j+1)$ should be "1", and the neighbor $(i, j-1)$ should be zero. The pixels in an image satisfying the pictorial condition can be found by applying the boolean function, corresponding to the pictorial condition on the image. The boolean function corresponding to a pictorial condition can be obtained by using the operators '.', '+' and '¬' representing AND, OR, and NOT respectively. The condition in the example can be expressed in terms of the boolean expression as $(i,j).(i-1,j-1).(i+1,j-1).(i+1,j+1).\neg(i,j-1)$. The and operator '.' can be omitted when there is no confusion. For example, the previous condition can be specified as $(i,j)(i-1,j-1)(i+1,j-1)(i+1,j+1)\neg(i,j-1)$.

```
             1111111111111111
          1111              11111111111111111111111
         11                           1
        11                           1
       11                            1
      11                             1
     11                              1
   111                               1
   1                                 1
   1                                 1
  11                                11
  1                                 1
  1                                 1
 11                                 1
 1                                  1
 1                                  1
 1                                  1
11                                  1
1                                   1
1                                   1
1                                   1
1                                   1
1                                   1
11       11                         11
 1        11                         1
 1         11                        1
 1          11                       1
 1           11                      1
 11           11                     1
  1            1                     1
  1            11                    1
  1             1                    1
  11            1                    1
   1            1                    1
   1            1                    1
   11           1                    1
    1           1                    1
    1          1                     1
    11         1                     1
     1        11                     1
     1        1                      1
     1       11                      1
     11     11                       1
      1    11                        1
      1  111                         1
       111
```

Figure 2.8 Output from Stefanalli and Rosenfeld's algorithm.

1		
0	1	
1		1

Figure 2.9 3x3 pictorial condition.

Definition 2.3: Contour Pixel

A pixel p_{ij} satisfying any of the following conditions is called a contour pixel.

- C1: $(i,j)\neg(i+1,j)$
- C2: $(i,j)\neg(i,j+1)$
- C3: $(i,j)\neg(i-1,j)$
- C4: $(i,j)\neg(i,j-1)$

Definition 2.4: Medial Line Pixel

A pixel p_{ij} satisfying any of the following conditions is called a medial line pixel.

- M1: $(i,j)\neg(i,j-1)\neg(i,j+1)((i-1,j-1)+(i-1,j)+(i-1,j+1))((i+1,j-1)+(i+1,j)+(i+1,j+1))$
- M2: $(\neg i-1,j-1)(i,j)\neg(i+1,j+1)((i-1,j)+(i-1,j+1)+(i,j+1))((i,j-1)+(i+1,j-1)+(i+1,j))$
- M3: $(i,j)\neg(i-1,j)\neg(i+1,j)((i-1,j-1)+(i,j-1)+(i+1,j-1))((i-1,j+1)+(i,j+1)+(i+1,j+1))$
- M4: $(i,j)\neg(i+1,j-1)\neg(i-1,j+1)((i-1,j-1)+(i,j-1)+(i-1,j))((i+1,j)+(i+1,j+1)+(i,j+1))$
- M5: $(i,j)(i+1,j)\neg(i+1,j-1)\neg(i,j+1)((i-1,j-1)+(i-1,j)+(i-1,j+1))$
- M6: $(i,j)(i,j+1)\neg(i+1,j)\neg(i-1,j+1)((i-1,j-1)+(i,j-1)+(i+1,j-1))$
- M7: $(i-1,j)\neg(i-1,j+1)\neg(i,j-1)(i,j)((i+1,j-1)+(i+1,j)+(i+1,j+1))$
- M8: $(i,j)(i,j-1)\neg(i-1,j)\neg(i+1,j-1)((i-1,j+1)+(i,j+1)+(i+1,j+1))$
- M9: $(i,j])(i+1,j-1)\neg(i,j-1)\neg(i+1,j)$
- M10: $(i,j)(i+1,j+1)\neg(i,j+1)\neg(i+1,j)$
- M11: $(i,j)(i-1,j+1)\neg(i-1,j)\neg(i,j+1)$
- M12: $(i-1,j-1)\neg(i-1,j)\neg(i,j-1)(i,j)$

Definition 2.5: Redundant Pixel

A pixel p_{ij} is said to be redundant if any of the following conditions are satisfied.

- R1: $(i,j)(i+1,j)(i,j+1)\neg(i-1,j-1)$
- R2: $(i,j)(i,j-1)(i-1,j)\neg(i+1,j+1)$
- R3: $(i,j)(i-11,j)(i,j+1)\neg(i+1,j-1)$
- R4: $(i,j)(i,j-1)(i+1,j)\neg(i-1,j+1)$

The extended parallel thinning algorithm is given below.

Algorithm 2.4: Thinning

```
procedure thinning;
   declare
       $P_{m \times n}$ : input binary image;
       $T_{m \times n}$ : thinned binary image;
       $C_{m \times n}$ : contour points;

begin
   T = 0;
   do forever
     begin
       for i := 1 to 2
         begin
           T = T + $\lambda$(P, i);
           if (T = P) then exit;
           C := $\alpha$(P, i);
           P := P . $\neg$C + T;
         end;
     end;
   $\Psi$(T);
end; /* thinning */

function $\alpha$(P, i)
begin
```

```
    if i = 1 then
       extract contour points satisfying C1 and/or C2;
    else /* i = 2 */
       extract contour points satisfying C3 and/or C4;
    return(contour points);
end;  /* $\alpha(P, i)$ */

function $\lambda(P, i)$
begin
    if i = 1 then
       extract medial line points satisfying M1-M10;
    else
       extract medial line points satisfying M1-M8, M11-M12;
    return(medial line points);
end;  /* $\lambda(P, i)$ */

function $\Psi(T)$
begin
   set redundant points in T satisfying R1-R4 to 0;
end;  /* $\Psi(T)$ */
```

The skeleton of the input image is formed by iteratively removing the contour pixels, except the medial line pixels, from the input image. To avoid nonconnected or empty medial lines for a connected image, each iteration is divided into two subiterations (Stefanalli and Rosenfeld [1971]). In subiteration 1, contour points satisfying conditions C1 and/or C2 are removed. In the next subiteration, contour points satisfying conditions C3 and/or C4 are removed. The medial points detected at each subiteration are accumulated. In order to avoid the elimination of contour points which are also final points, all the contour points are first deleted from P and then the final points are added. The functions $\lambda(P, i)$ and $\alpha(P, i)$ compute the contour points and the medial points respectively for subiteration i.

The line thickness of the image is reduced to unity at all points, by removing the redundant pixels defined in R1 thru R4. The function Ψ (*T*) removes the redundant pixels from *T*. *i* takes the value 1 and 2. Figure 2.10 is the output of Algorithm 2.4 for the input pattern shown in Figure 2.7.

The features, extraction and their representation are described in the next chapter.

```
             1111111111111111
          111                 111111111 1111111111111
         1                             1
        1                             1
       1                              1
      1                               1
     1                                1
    1                                 1
   1                                  1
   1                                  1
  1                                  1
  1                                  1
  1                                  1
 1                                   1
 1                                   1
 1                                   1
 1                                   1
1                                    1
1                                    1
1                                    1
1                                    1
1                                    1
1                                    1
 1       11                           1
 1         1                          1
 1          1                         1
 1           1                        1
 1            1                       1
  1            1                      1
  1            1                      1
  1             1                     1
  1             1                     1
   1            1                     1
   1            1                     1
   1            1                     1
   1            1                     1
    1           1                     1
    1          1                      1
    1          1                      1
     1        1                       1
     1        1                       1
     1       1                        1
      1     1                         1
      1    1                          1
      1  11                           1
       11
```

Figure 2.10 Output from thinning algorithm.

3. FEATURES

DEFINITION, EXTRACTION AND REPRESENTATION

3.1 Introduction

Any pattern which can be recognized possesses a number of discriminatory properties or features. The number of features needed to successfully perform a given recognition task depends on the discriminatory qualities of the chosen features. Feature selection and extraction plays a central role in pattern recognition. One of the important steps in any recognition process is to consider the problem of what discriminatory features to select and how to extract them. During the past several decades considerable research has been done to define and extract the good quality features (Stringa [1990], Ott [1974], Suen [1982a], Zhan [1988]). However, the problem of feature selection is usually complicated by the fact that the most important features are not necessarily easily measurable, or, in many cases, their measurement is inhibited by economic considerations. In fact, the selection of an appropriate set of features which takes into account the difficulties present in the extraction or selection process, and at the same time result in acceptable performance (in terms of speed and reliability), is one of the most difficult tasks in the design of pattern recognition systems (Tou and Gonzalez [1974]). Also, one should keep in mind that recognition techniques vary widely according to the features chosen, the way these features are extracted, and the classification scheme used (Suen et al [1980]).

The features used in solving character recognition problems can be divided into two categories: (1) mathematical (statistical) features, and (2) structural features. Commonly used mathematical features are: transformation and series expansions (Wakahara [1988], Yoshimura [1988], Takahashi [1988], Kushnir [1985], Lai and Suen [1981], Ott [1974]), correlation coefficients based on the pixels in the pattern (Tsukumo and Tanaka [1988], Minneman [1966], McLaughlin and Raviv [1968]), density of pixels in overlapping and nonoverlapping regions in an image (Linquan [1988], Yamada et al [1988], Akamatsu and Kawatani [1983]), moments of character pixels about a chosen point (Tucker and Evans [1974]), crossings such as transitions from background to foreground and vice versa (Holt [1974]), distances of elements or line segments from a given boundary (Fu [1982]), multidirectional loci (Suen [1982a]), and n-tuples (occurrence of zero (background) and one (foreground) pixels) (Leveridge and Leedham [1988]).

Structural features describe the geometrical and topological shapes of a pattern. Commonly used structural features are strokes and bays (cavities) in various directions (Chang et al [1988], Cheng and Hsu [1986], Ding, Wu and Zhu [1988], Hsu and Cheng [1985], Hung [1987], Jian-long and Wenhao [1988], Zhang, He and Ge [1988], Ahmed [1986], Suen and Shillman [1977]), end points, intersection of line segments, loops (Ali and Pavlidis [1977]).

The structural features have high tolerance to distortion and style variations and tolerate a certain degree of translation and rotation whereas mathematical features are highly sensitive to style variations, translation and rotations. On the other hand, it is easy to design and implement machines to extract the mathematical features are easy compared to extraction of structural features. A comparative study and an evaluation of the relative merits of various features based on the sensitivity to deformation of patterns and practical implementation can be found in Suen et al [1980].

3.2 Structural Description of Characters Using DAG

3.2.1 Primitives

Characters and symbols are composed of line-like elements called primitives. The following primitives are used to describe them:

1. Vertical line
2. Horizontal line
3. Left diagonal
4. Right diagonal
5. Left curve
6. Cap
7. Right curve
8. Cup

The pictorial representations of these primitives are given in Figure 3.1. The numbers shown along with the primitives in Figure 3.1 will be used to represent those primitives.

Each primitive, *p* has a set of attributes, A.

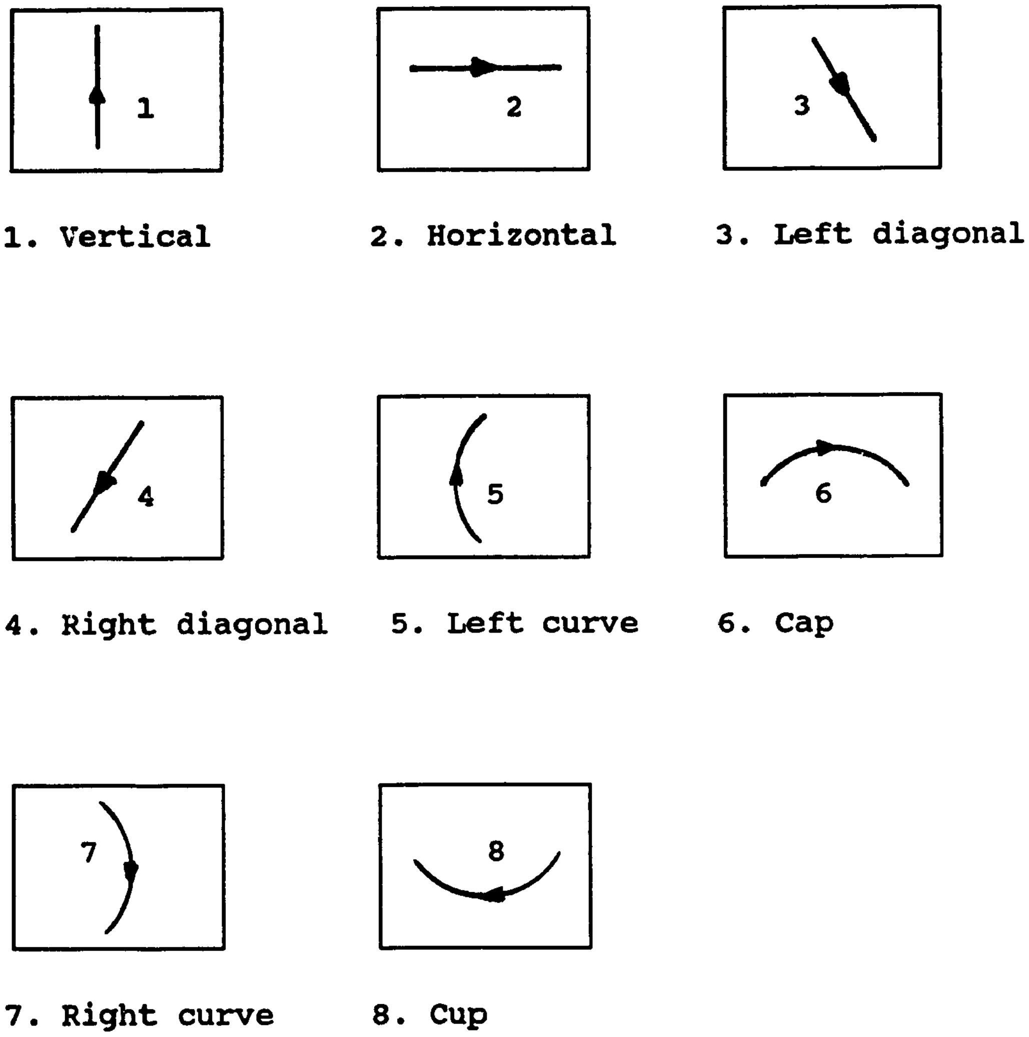
$$A(p) = \{a_1, a_2, \ldots ,a_n\}$$

Several attributes such as type, size, location, etc., could be defined for a primitive. The minimum required attribute for a primitive is its type; others can be left as optional to the specific implementation.

An analysis of the characters of the Latin, the Cryllic and the Indian languages revealed that they have some common geometric characteristics. The characters of these languages can be divided into 3-tiers as shown in Figure 3.2. The size of tier-2 is almost double the size of tier-1 and tier-3; tier-1 and tier-3 are of approximately the same size. This fact can be used to divide the input image into three zones with the size ratio of 1:2:1 instead of three equal sizes. The location attribute can be defined based on these divisions.

3.2.2 Structural Description

The structure of a character/symbol (primitives present in it and the relations satisfied by them) can be described by a directed attribute graph (DAG). The nodes of DAG correspond to the junctions of the character. By junction we mean either the



1. Vertical

2. Horizontal

3. Left diagonal

4. Right diagonal

5. Left curve

6. Cap

7. Right curve

8. Cup

Figure 3.1 Primitives.

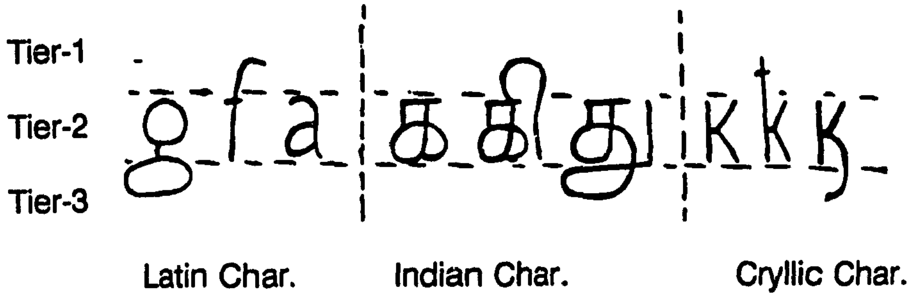


Figure 3.2 Common geometric characteristics of major scripts.

end point of a primitive or a point where two or more primitives meet. The edges and direction of the DAG are the primitives and the direction associated with the primitives respectively. The attribute set A(*p*) of the primitive *p* is assigned as the label of the edge representing the primitive *p* in the DAG. Graphical representation of pictures consisting line-like elements has been used by Narasimhan [1964] in the analysis of bubble chamber photographs and by Hildich [1968] in the analysis of chromosomes.

3.2.3 Representation of Directed Attribute Graph

The DAG of a character can be represented by a connectivity matrix (*c*-matrix). The *c*-matrix of a DAG is a square matrix of size *nxn*, where *n* is the number of nodes in the DAG. Let, c_{ij} represent the element in the i^{th} row and j^{th} column of a *c*-matrix. Then, c_{ij} is non-zero if and only if the nodes *i* and *j* of the DAG are connected by an edge. When non-zero, c_{ij} represents the label (attribute) of the edge (corresponding to the primitives joining the junctions represented by nodes *i* and *j*). c_{ij} is positive $\alpha(p)$ if the direction of the edge is from *i* to *j*, otherwise it is negative $\alpha(p)$. The DAG of the Malayalam letter ഘ (GHA) and its *c*-matrix are given in Figures 3.3, and 3.4 respectively.

3.3 Definition and Extraction of Primitives

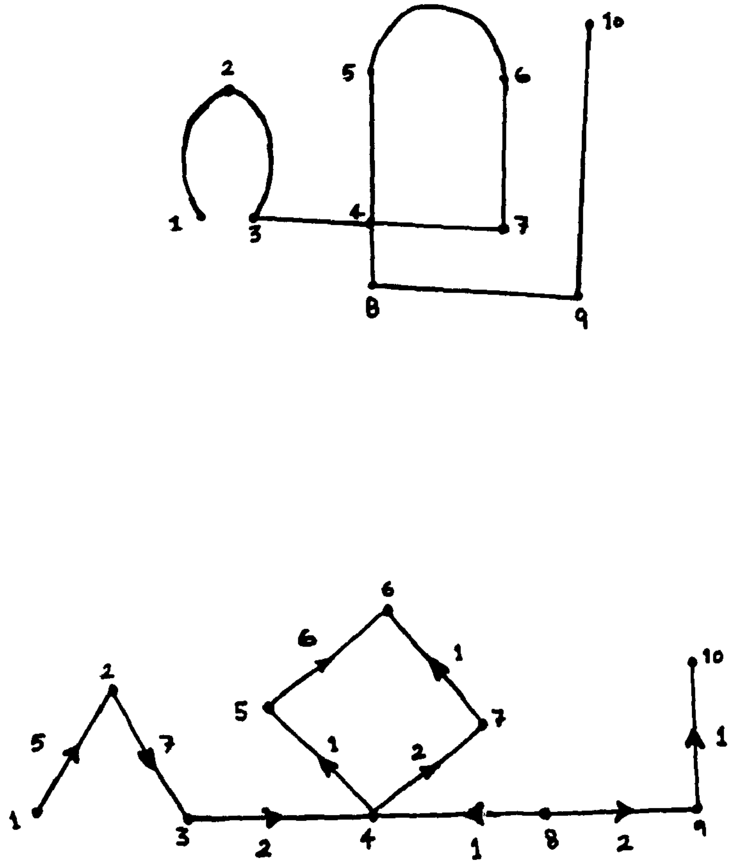


Figure 3.3 Directed attribute graph (DAG) of ഘ(GHA).

	1	2	3	4	5	6	7	8	9	10
1	0	5	0	0	0	0	0	0	0	0
2	-5	0	7	0	0	0	0	0	0	0
3	0	-7	0	2	0	0	0	0	0	0
4	0	0	-2	0	1	0	2	-1	0	0
5	0	0	0	-1	0	6	0	0	0	0
6	0	0	0	0	-6	0	-1	0	0	0
7	0	0	0	-2	0	1	0	0	0	0
8	0	0	0	1	0	0	0	0	2	0
9	0	0	0	0	0	0	0	-2	0	1
10	0	0	0	0	0	0	0	0	-1	0

Figure 3.4 C-Matrix of ഘ(GHA)

3.3.1 Definition of Primitives

Definition 3.1: Vertical Line

The pixels satisfying any of the following conditions are declared as a vertical line.

1. $((i\text{-}1,j\text{-}1)+(i\text{-}1,j)+(i\text{-}1,j+1))(i,j)(i+1,j)$
2. $((i+1,j\text{-}1)+(i+1,j)+(i+1,j+1))(i,j)(i\text{-}1,j)$

Definition 3.2: Horizontal Line

The pixels satisfying any of the following conditions are declared as a horizontal line.

1. $((i\text{-}1,j\text{-}1)+(i,j\text{-}1)+(i+1,j\text{-}1))(i,j)(i,j+1)$
2. $((i\text{-}1,j+1)+(i,j+1)+(i+1,j+1))(i\text{-}1,j\text{-}1)(i,j)$

Definition 3.3: Left Diagonal

The pixels satisfying any of the following conditions are declared as a left diagonal.

1. $((i,j+1)+(i+1,j+1)+(i+1,j+1))(i\text{-}1,j\text{-}1)(i,j)$
2. $((i\text{-}1,j\text{-}1)+(i\text{-}1,j)+(i,j\text{-}1))(i,j)(i+1,j+1)$

Definition 3.4: Right Diagonal

The pixels satisfying any of the following conditions are declared as a right diagonal.

1. $((i,j-1)+(i+1,j-1)+(i+1,j))(i,j)(i-1,j+1)$
2. $((i-1,j)+(i-1,j+1)+(i,j+1))(i,j)(i+1,j-1)$

Definition 3.5: Vertical Junction Point

The pixels satisfying any of the following conditions are declared as a vertical junction point.

1. $((i-1,j-1)+(i-1,j)+(i-1,j+1))(i,j)\neg(i+1,j-1)\neg(i+1,j)\neg(i+1,j+1)$
2. $((i+1,j-1)+(i+1,j)+(i+1,j+1))(i,j)\neg(i-1,j-1)\neg(i-1,j)\neg(i-1,j+1)$

Definition 3.6: Horizontal Junction Point

The pixels satisfying any of the following conditions are declared as horizontal junction points.

1. $((i-1,j-1)+(i,j-1)+(i+1,j-1))(i,j)\neg(i-1,j+1)\neg(i,j+1)\neg(i+1,j+1)$
2. $((i-1,j+1])+(i,j+1)+(i+1,j+1))(i,j)\neg(i-1,j-1)\neg(i,j-1)\neg(i+1,j-1)$

3.3.2 Extraction

The extraction of primitives from the input image is very important because the success of the classifier mostly depends on the correctness of the extracted primitives. The various primitives present in the input image are extracted using the above definition. According to this scheme the pixels belonging to the lines making small angles with any of the principal directions are assigned the primitive corresponding to the principal direction, thus nullifying the effect of small distortions in the input symbol.

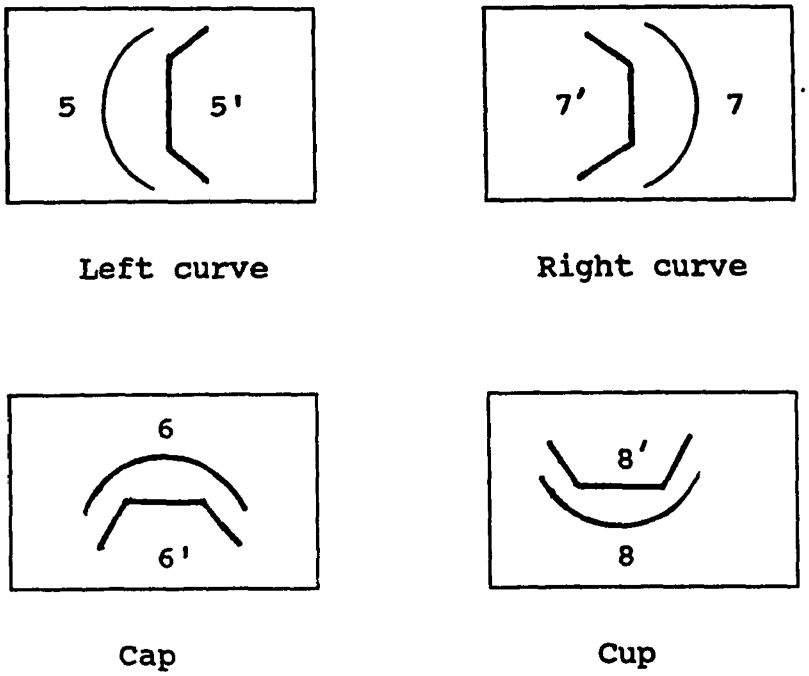
The junctions in the input image are determined, tentatively, as the points which are assigned *v*-primitive or *h*-primitive and satisfy the conditions given in the definitions 3.5 and 3.6 above. The set of junctions so determined will have many redundancies, but the redundancies will be eliminated at a later stage. The computed junctions are numbered. A preliminary connectivity matrix consisting of only straight lines along the principal directions as primitives, is constructed. The primitives are qualified with attributes short and long, depending upon the length of the line segment compared to a threshold value, computed from the skeleton size of the input image. Between junctions *i* and *j* of the image a line segment in a particular direction exists if the junctions are connected by a chain of pixels satisfying the condition given (corresponding to the direction) in the definitions 3.1 to 3.4. The program for constructing the preliminary *c*-matrix of the input symbol is speeded up by testing for only possible line segments between a pair of junctions. For example, it is not necessary to test for a horizontal line between junctions which are almost vertically apart.

The redundancies in the preliminary *c*-matrix constructed by the above procedure should be eliminated. Some of the short line segments should be combined to get curved primitives. The extractions of curved primitives are achieved by approximating them as combinations of straight line segments in the principal directions as shown in Figure 3.5. When a curved primitive is obtained by combining three line segments, the two intermediate junctions in the curve should be deleted from the set of junctions. Also the junctions within a radius of a threshold value (3 in our case) should be treated as one junction. That is, the columns and rows of the preliminary *c*-matrix corresponding to these junctions should be merged. With these modifications on the preliminary *c*-matrix of the input image, the actual *c*-matrix of the input image is obtained. The *c*-matrix of the image in Figure 2.10 is given in Figure 3.6.

3.4 Knowledgebase

The knowledgebase contains several databases, one for each language. It also contains the mapping between languages and language codes. Each database contains the *c*-matrices of the symbols in the language and a decision tree called character recognition tree (CRT). In the first stage of a two stage recognition system, the c-matrices are used to recognize the various symbols in the input image. In the second stage the CRT is used to recognize the characters from the symbols.

One *c*-matrix for each symbol in the language is stored apriori in the database. This set is enhanced further during the training period. The CRT for each language is built apriori in the database after careful analysis of the structures and symbol of the characters in each language.



Left curve

Right curve

Cap

Cup

Figure 3.5 Curved primitives and their approximations.

	1	2	3	4	5	6	7
1	0	2	0	4	0	0	0
2	-2	0	2	0	0	0	-1
3	0	-2	0	0	0	0	0
4	-4	0	0	0	-1	0	0
5	0	0	0	1	0	-7	0
6	0	0	0	0	7	0	0
7	0	1	0	0	0	0	0

Figure 3.6 C-Matrix of input image in Figure 2.10

The techniques (the two stages of the recognition systems, including CRT) developed for the recognition of symbols and characters are discussed in detail in the next chapter.

4. CHARACTER RECOGNITION

4.1 Introduction

The classification or recognition of an unknown character pattern P is a decision making process that assigns the unknown P to a member of a set $K=\{k_1, k_2, \ldots ,k_m\}$ of m known classes or rejects P as a non-member of K via the extraction of significant features or attributes $(p_1, p_2, \ldots ,p_n)$ of P from a background of irrelevant detail.

Sometimes, a classifier may mistakenly recognize P as k_j when it really belongs to k_i. This error is known as substitution error. Ideally, a classifier should have zero substitution error and zero rejection error as long as the input pattern is a member of K. Achieving an ideal classifier is highly impractical due to noise and deformation like unexpected breaks, holes, limbs, etc., in the input pattern.

The development of a good classification method having minimum rejection and substitution errors has been one of the major topics of research in pattern recognition since the inception of this field. Methods from various theoretical and applied fields of science and engineering, such as mathematics, formal languages, graph theory, etc., have been explored and applied for this purpose. Still, a good general classification method applicable to various classes of patterns such as characters of several scripts or characters of a single script with printed, handwritten and cursive characters, supplied either on-line or off-line, is not available. It shows the complexity of the character recognition problem. The different classification methods cited in the literature are reviewed briefly in the following paragraphs.

4.1.1 Overview of Classification Methodologies

The pattern classification methodologies reported in the literature can be loosely grouped into three different categories: decision-theoretic, syntactic and hybrid categories. The decision-theoretic approach is based on the utilization of decision functions for classifying patterns. In a simple way, this approach could be stated as follows:

Let m be the number of classes. Establish m decision functions $d_1(x)$, $d_2(x)$, ... ,$d_m(x)$ with the property that if a pattern x belongs to class ω_i, then

$$d_i > d_j, j=1, 2, \ldots, m, \quad j \neq i$$

This relationship specifies a decision rule in which each pattern to be classified is substituted into all decision functions. The pattern is assigned to the class whose decision function yields the largest numerical value. Ties are resolved arbitrarily.

The decision functions may be expressed in a general linear form

$$d_k(x) = \sum_{l=1}^{K} w_{kl}\, \phi_l(x), \quad k = 1, 2, \dots, m$$

where the $\{\phi(x)\}$ are real, single-valued functions of the pattern x, and $\{w_{ki}\}$ are the coefficients of the decision function corresponding the class ω_k. The usual approach is to specify first the functions $\{\phi_i(x)\}$. The problem then becomes one of determining the coefficients $\{w_{ki}\}$ for each class, so that the foregoing decision rule will hold for as many patterns as possible and will satisfy the given criteria. The decision-theoretic approach to pattern recognition deals with algorithms for estimating these parameters using sample patterns in a training process.

There exist numerous adaptive algorithms that can be used for training a pattern recognition system (Sabourin and Plamondon [1990], Tappert [1990], Simon and Baret [1990]). These algorithms can be divided into (1) deterministic and (2) statistical procedures. The deterministic algorithms deal with the estimation of the decision function coefficients directly from the patterns without resorting to statistical considerations, while statistical algorithms are based on the statistical properties of the various pattern populations under consideration. The use of decision-theoretic approach for classification can be found in (Shyu, et al [1988], Wang and Suen [1984], Dattatreya [1980]).

In the decision-theoretic approach, if the pattern recognition system performs well during training with a well-chosen set of representative patterns, it may be expected to perform satisfactorily when confronted with unknown "field" data during normal operation.

The decision-theoretic approach lacks a suitable formalism for handling pattern structures and their relationships. However, the structure of a pattern carries very useful information that could aid or play an important role in the classification process. The syntactic approach to pattern recognition possesses the structure-handling capability. The terms linguistic, structural and grammatical pattern recognition are also often used in the literature to denote the syntactic approach.

Syntactic pattern recognition is based on concepts from formal language theory, the origins of which may be traced to the middle of 1950s with the development of mathematical models of grammars by Noam Chomsky [1956]. Considerable theoretical as well as applied studies have been reported in the pattern recognition literature (Stringa [1990], Hsieh, et al [1990], Lu and Suen [1990], Morasso [1989], Downton, Kabir and Guillevic [1988], Ding, Wu and Zhu [1988], Hsu and Cheng [1985], Liu and Tai [1988], Xiu-Guang and Jia-Ruo [1988], Leow [1986], Shridhar and Badreldin [1985], Fu [1982], Zhang [1980], Wang [1985]).

Basic to the syntactic pattern recognition approach is the decomposition of patterns into strings of subpatterns or primitives. Each primitive is interpreted as being a symbol permissible in some grammar, where a grammar is a set of rules of syntax for the generation of sentences from the given symbols. Strings generated from a pattern class are used to construct the grammar for the class of patterns. Thus, n grammars (G_1, G_2, ... ,G_n) are constructed for n classes (C_1, C_2, ... ,C_n) of patterns. The language $L(G_i)$ generated by grammar G_i would consist of sentences representing the pattern class C_i where $1 \leq i \leq n$.

Given a sentence $S(P)$ representing an unknown pattern P, the recognition process is one of deciding in which language P represents a valid sentence. Thus, if $S(P) \in L(G_i)$ only, then $P \in C_i$. A unique decision cannot be made if the sentence $S(P)$ belongs to more than one language. In this case, additional features and/or a different approach may be needed to resolve the conflict. If $S(P) \notin L(G_i)$, $1 \leq i \leq n$, then P is rejected, that is, assigned to a rejection class.

Several different types of grammars and parsing techniques are described in the literature (Tsai and Fu [1980], Gonzalez and Thomason [1978]). In general, the various techniques of the syntactic approach could be classified into the deterministic, stochastic and attributed grammar methods. In the deterministic method, an unknown pattern could be successfully parsed by more than one language hindering the decision process. The stochastic grammar method was proposed to help in resolving the uncertainties (Fu [1974]). Still, the stochastic method fails to capture the attributes related to primitives or subpatterns. The attributed grammars provide semantic rules to incorporate attributes associated with each primitive. This approach allows recognition of patterns of greater variability.

Graphs facilitate representation of pattern primitives, their attributes and connectivity. Hence, some researchers used graph-theoretic techniques for patten recognition (Dong, Wu and Ding [1988], Ping and Cheung [1988], Tsai and Fu [1980], Faris [1983], Wang [1984]).

Researchers have found that application of either decision-theoretic or syntactic approach alone is not sufficient where there is a high degree of variability, complexity and distortion in the input patterns. Therefore, hybrid systems empolying both decision-theoretic and syntactic approaches have been suggested by several researchers (Suen [1990], Downton, Kabir and Guillevic [1988], Ahmed [1986], Fu [1983], Tsai and Fu [1980]; Duerr et al [1980]).

Characters can be recognized either on-line (Berthod and Maroy [1979], Berthod [1982]) or off-line. The classifier developed in this thesis recognizes characters off-line. It is explained in detail in the rest of this chapter.

4.1.2 Multistage Hybrid Classifier

The classifier developed in this thesis is a two stage recognition system. In the first stage, the symbols constituting the input pattern are recognized using graph-theoretic approach. In the second stage, the character is recognized from the symbols using a decision tree and the properties of the language. Thus, this approach falls under the hybrid category described earlier.

The two stages are explained in detail in the following sections.

4.2 Stage I: Recognition and Removal of Symbols

Let,

$$S = \{s_1, s_2, \dots s_n\}$$

be the set of symbols constituting the character set of a language L_i. As stated in the previous chapter, the *c*-matrices, that is, DAGs, of the members of S are stored in the knowledgebase.

Let,

$$P = \{p_1, p_2, \dots p_m\}$$

be the set of symbols in the unknown input pattern P. Notice that $P \subset S$.

The first stage of the recognizer has to identify the symbols $p_1, p_2, \dots p_m$ in the unknown pattern P. The a-priori knowledge stored in the knowledgebase is used for this purpose.

The symbol recognition problem can be stated as follows:

"Given the DAG(P), if DAG(s_i) is isomorphic or subisomorphic to DAG(p), then the symbol s_i is in P, where $s_i \epsilon S$ for $1 \leq i \leq n$."

Theorem 4.1: The symbol recognition problem stated above is *np*-complete.

Proof: The normal method of proving a problem to be *np*-complete is by converting the given problem to a known *np*-complete problem. Hence, we will prove the theorem by converting the symbol recognition problem to a known *np*-complete problem.

DAGs for P and S are graphs. In order to recognize a symbol, the recognizer has to compute whether the graph DAG(s_i) is isomorphic or subisomorphic to the graph DAG(P). In other words, this problem is the same as identifying isomorphism or subisomorphism between the two graphs. It has been proved that computing whether a graph G_1 is isomorphic or subisomorphic to graph G_2 is *np*-complete (Garey and Johnson [1979]). Hence, the above symbol recognition problem is an *np*-complete problem.

4.2.1 Correlation Coefficient

Obtaining isomorphism between the unknown pattern and the known symbols is not always possible due to noise and variations in the patterns since isomorphism is a 1:1 correspondence between two graphs. Also, as proved earlier, checking for isomorphism is computationally expensive. Therefore, instead of computing isomorphism, a method of computing correlation coefficients between the known symbols in the knowledgebase and the unknown pattern is proposed and explained in the following paragraphs.

The following definitions are used to describe the methodology and the recognition algorithms.

Definition 4.1: Null Graph

The DAG(P) of a pattern P is said to be null, if DAG(P) has no edge or node in it. The c-matrix of a null graph will have no or zero entries. It is denoted as $\emptyset_P$.

Definition 4.2: Null Pattern

A pattern P is said to be null or has only a background, if DAG(P) is $\emptyset_P$.

Definition 4.3: Null Pattern Function

Given P, the function ϕ determines whether P is a null pattern or not. It is defined as follows:

$$\phi(P) = \begin{cases} 1, & \textit{if DAG(P) is e,} \\ 0, & \textit{otherwise} \end{cases}$$

Definition 4.4: Neighborhood Rule

If junction ψ of pattern P corresponds to junction σ of pattern Q, then the junction connected to ψ by a primitive ρ in P corresponds to the junction connected to σ by the same primitive ρ in Q.

Definition 4.5: Primitive Correspondence Rule

If junctions ψ_i and ψ_j of pattern P correspond to the junctions σ_i and σ_j of pattern Q respectively and the primitive connecting the junctions ψ_i and ψ_j is same as the primitive connecting junctions σ_i and σ_j, then the primitive connecting ψ_i and ψ_j in P corresponds to the primitive connecting junction σ_i and σ_j in Q.

Definition 4.6: Correlation Coefficients

Let, $\{p_1, p_2, \ldots, p_n\}$ be the set of n primitives in the unknown pattern P, $\{q_1, q_2, \ldots, q_m\}$ be the set of m primitives in the known pattern Q and $\{p_1, p_2, \ldots, p_k\}$ be the set of k primitives in P for which correspondences were found with primitives in Q.

We define the following two types of correlation coefficients:

P-Correlation: The P-correlation coefficient, denoted as $\alpha_P(P,Q)$, indicates the extent of correlation with respect to the unknown pattern P. It is defined as:

$$\alpha_p(P,Q) = \frac{k}{n}$$

Q-Correlation: The Q-correlation coefficient, denoted as $\alpha_Q(P,Q)$, indicates the extent of correlation with respect to the known pattern Q. It is defined as:

$$\alpha_Q(P,Q) = \frac{k}{m}$$

α is also called the correlation coefficient function.

Definition 4.7: Junction Complexity

The complexity of a junction β is said to be κ, if κ primitives meet at β. The junction β is a κ-complexity junction.

Definition 4.8: Same Complexity

Two junctions β_i and β_j are said to have the same complexity, if both are κ-complexity junctions, that is, the complexity of β_i is equal to the complexity of β_j.

Definition 4.9: p-neighbor

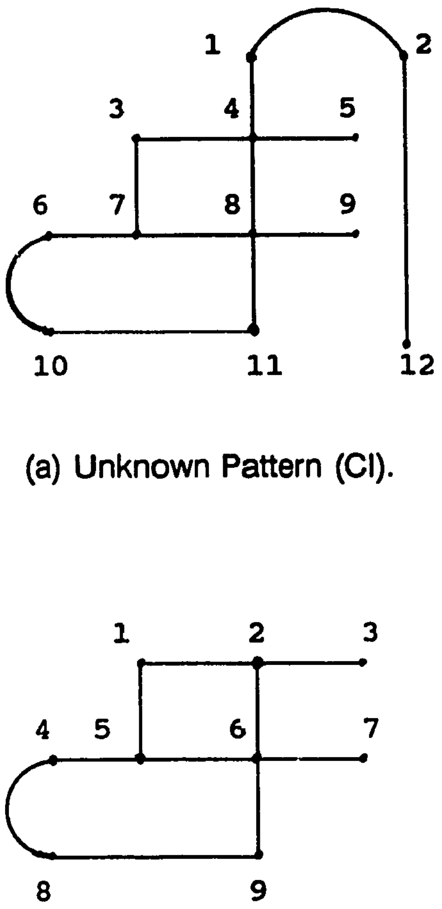
The p-neighbor of a set of junctions X is defined as a set Y of junctions such that the junctions of X are connected to junctions of Y by primitive p.

In order to compute the correlation coefficients, the c-matrix corresponding to the DAG(P) of the unknown input pattern P is compared with the c-matrices corresponding to the DAGs of the member of S in the knowledgebase one by one and correlation coefficients are computed. From the set of correlation coefficients, the maximum correlation value is computed to select the symbol which has the maximum correlation coefficient with the input symbol.

4.2.2 Computation of correlation coefficient

Let us consider an example to illustrate the method of computing correlation coefficient between two patterns. Let the unknown character P be (CI) and the known symbol s_i be (CA). They are shown in Figure 4.1 with junctions numbered starting from 1. The correlation coefficient has to be computed between P and s_i. To compute the correlation coefficient between P and s_i, correspondence between the junctions of P and s_i, must be established based on the structural similarity between P and s_i. Suppose at some stage we conjecture that a junction $p(8)$ of pattern P corresponds to a junction $s_i(6)$ of symbol s_i; the correspondence is denoted as 8<->6. Then, by assuming that P and s_i do not have any disconnected components, we can extend the correspondence to other junctions of P and s_i by applying the neighborhood rule.

In the example, with the correspondence 8<->6 and by applying the



(a) Unknown Pattern (CI).

(b) Known symbol (CA).

Figure 4.1: Unknown pattern (CI) and known symbol (CA).

neighborhood rule we can find correspondence between junctions 9<->7, 7<->5, 4<->2, 11<->9 of *P* and s_i. By repeatedly applying the correspondence and neighborhood rule, we can establish further correspondence to junctions 6<->4, 3<->1, 5<->3, and 10<->8.

Note, the junctions 1, 2 and 12 of *P* do not correspond to any junction of s_i.

From the correspondences between various junctions of *P* and s_i we can compute the correspondences between the primitives of *P* and s_i by applying the primitive correspondence rule. In the example, the unknown pattern *P* has 13 primitives. The known symbol s_i has 10 primitives out of which correspondences could be found for all 10 primitives with *P*. Therefore, the *P*-correlation coefficient is 10/13 and the *Q*-correlation coefficient is 10/10. The *Q*-correlation indicates that the known symbol completely (100%) matched or present in the unknown pattern *P*.

In the above method of computing the correlation coefficient we assumed a junction correspondence to start with. If the starting correspondence is wrong then the correlation coefficient computed will not be correct. So the following heuristic techniques can be used to get the initial correspondence between a junction of *P* and a junction of s_i.

Algorithm 4.1: Computation of Initial Correspondence

Partition the set of junctions of pattern P and the symbol s_i according to the equivalence relation "same complexity". The partitions on the junctions of P and s_i of Figure 4.1 are given in Table 4.1.

If the symbol s_i is to be a derived from pattern P, a junction of s_i of κ-complexity will correspond to a junction of P of the same complexity, except the junction at which the additions or deletions are made.

For example, consider the 4-complexity junctions of symbol s_i and the pattern P of Figure 4.1. P has got 2 junctions {4,8} of complexity 4 and the symbol s_i has got 1 junction {6} of complexity 4. So there are two possible initial correspondences: (1) 4<->6 and (2) 8<->6. Suppose we choose the first correspondence 4<->6 and extend it by the neighborhood rule, we get the correspondence between the junctions as 1<->2, 5<->9, 3<->5, 8<->9 etc. All these correspondences are between junctions of different complexity and so this cannot be a good choice. If we choose the second correspondence 8<->6 the junction correspondences can be found as 9<->7, 7<->5, 4<->2, 11<->9, etc. This time all the junction correspondences except 4<->2 are between junctions of same complexity and so this is a better choice than the first one.

Algorithm 4.2: Computation of Initial Correspondence

An alternative method has been developed by which a set of initial junction

Table 4.1 Partitions of P (கி) and s_i (க)

Junction Complexity	Partitions on junctions of *P*	Partitions on junctions of s_i
1	(5,9,12)	(3,7)
2	(1,2,3,6,10,11)	(1,4,8,9)
3	(7)	(2,5)
4	(4,8)	(6)

correspondences between two patterns P and Q can be found. The method is explained below.

Partition the set of junctions of P and s_i as in Algorithm 4.1. Form the subset pairs (P_j, s_{ij}), where P_j and s_{ij} are the j-complexity partitions of P and s_i respectively, for all possible j. From these initial subset pairs generate additional subset pairs by computing the p-neighbors of the subsets in each pair, for all p.

Then, intersect the generated subset pairs with the initial subset pairs. If the result of intersection is a pair of singleton subsets, a correspondence between the junctions (singleton elements of the pair) is assumed.

The result of the application of the above algorithm for the pattern P and the symbol s_i in Figure 4.1 is shown in Table 4.2. Intersection of the subset pairs in line 3 and line 12 gives one junction correspondence as 7<->5. Similarly the intersection of the subset pairs in line 2 and line 9 gives a correspondence 3<->1. Other correspondences between junctions can be found in a similar manner as follows: 6<->4, and 8<->6.

The second method was used in the implementation. The reason for that is, even if the pattern has disconnected components, the method will succeed. For example in the case of Tamil consonants such as க் , ண், etc.

Table 4.2 Partitions of P (की) and s_i (क)

Line No.	Symbol P	Symbol s_i	
1	(5,9,12)	(3,7)	1-complexity
2	(1,2,3,6,10,11)	(1,4,8,9)	2-complexity
3	(7)	(2,5)	3-complexity
4	(4,8)	(6)	4-complexity
5	(4,8)	(2,6)	2-neighbors of subsets of line 1
6	(4,7,8,12)	(5,6)	1-neighbor of subsets of line 2
7	(4,7,10,11)	(2,5,8,9)	2-neighbors of subsets of line 2
8	(6,10)	(4,8)	5-neighbors of subsets of line 2
9	(3)	(1,6)	1-neighbor of subsets of line 3
10	(6,8)	(1,3,4,6)	2-neighbors of subsets of line 3
11	(1,4,8,11)	(2,9)	1-neighbor of subsets of line 4
12	(3,5,7,9)	(5,7)	2-neighbors of subsets of line 4

4.2.3 Symbol Recognition

Algorithms 4.1 and 4.2 explain the method of computing the initial correlation between the unknown pattern and a known symbol in the knowledgebase. In order to decide which symbol is in *P*, it is necessary to compute the set of correlation coefficients between *P* and each member of the symbol set *S*. Then, maximize the set of correlation coefficients to select the symbol which has the maximum correlation coefficient with the unknown input symbol *P*. The symbol recognition algorithm is given below:

Algorithm 4.3: Symbol Recognition Algorithm

```
procedure recognize_symbols;
declare
    P : unknown pattern;
    S : set of known symbols in the knowledgebase;
    K : set of DAGs for S;
    $\xi$ : set of recognized symbols;
    $\omega$ : array of P-correlation coefficients;
    $\psi$ : array of Q-correlation coefficients;

begin
    compute $\xi = \{\emptyset\}$;
    read DAG(P);
    do
      begin
        for i := 1 to |K|
          begin
            compute k = K[i];
            compute symbol $s \in S$ for k;
            compute $\omega[i] = \alpha_P(P, s)$;
            compute $\psi[i] = \alpha_Q(P, s)$;
          end;
        compute symbol r such that maximize($\omega$, $\psi$);
        if r = $\emptyset$
            compute found_symbol = false;
        else
          begin
            compute $\xi = \xi \cup \{r\}$;
            compute found_symbol = true;
            remove r from $\bar{P}$; /* using Algorithm 4.4 */
          end;
    while((found_symbol = true)&(DAG(P) != $\emptyset_P$));
end;
```

4.2.4 Removal of Symbols

Once the existence of a known symbol s_r in the input pattern P is recognized, it is removed from the DAG(P) using Algorithm 4.4 given below so that the other symbols in P could be recognized. The algorithm removes all the edges and nodes which are in DAG(s_r) and DAG(P) and not in DAG(P') where DAG(P') is the directed attribute graph corresponding to the pattern P' and $P' = P - s_r$. In other words, P' is the pattern obtained after removing the pattern corresponding to symbol s_r from P.

In order to describe the symbol removal algorithm, the nodes and edges in DAG(P) are grouped into three categories as follows: (1) E_s and N_s be the set of edges and nodes, respectively, belongs only to s_r and not to P'. (2) $E_{P'}$ and $N_{P'}$ be the set of edges and nodes, respectively, that belongs only to P' and not to s_r. (3) $E_{sP'}$ and $N_{sP'}$ be the set of edges and nodes, respectively, that are common to both s_r and P'.

Algorithm 4.4: Symbol Removal Algorithm

```
procedure remove_symbol(P, s);

declare
   P : unknown pattern;
   s : known symbol to be removed;
   $E_s$ : set of edges belonging to s;

begin
   compute $E_s$;
   for i := 1 to $|E_s|$
     begin
       compute nodes (i,j) for edge $E_s(i)$;
       compute $DAG(P)_{ij} = 0$;
     end;
   remove $N_s$;
end;
```

Theorem 4.2: Algorithm 4.4 does remove s_r and only s_r from P.

Proof: As defined in Chapter 1, a character is obtained from one symbol or by combining two or more symbols without any overlapping of primitives. Therefore, there is no common edge between s_r and P', that is, $E_{sP'} = \emptyset$. However, $N_{sP'}$ may or may not be empty since the character could be obtained by combining symbols at one or more junctions which form the common nodes. Since the algorithm does not remove any common nodes, all the nodes of P' are preserved. Removal of an entry from the c-matrix of DAG(P) removes only the edge (primitive) connecting two nodes and not the node itself.

After removing all the edges corresponding to s_r, all the nodes in the set N_s will have zero complexity. Removing nodes in N_s that belongs only to s_r should not have any influence on P'. Thus, the algorithm removes s_r and only s_r from P.

4.3 Stage II: Character Recognition

After individual symbols are recognized, some of the adjacent symbols and or multiple symbols in a single pattern are to be combined to recognize the characters in the input text. The symbols ெ , ே , ை , etc., are not characters by themselves. They combine with the succeeding symbols and form a character. The symbol ா combines with preceding symbol or symbols to form a character.

Stage II recognizes characters from the set of symbols recognized from Stage I using a decision tree called Character Recognition Tree (CRT). The organization, construction and searching of the CRT are described below.

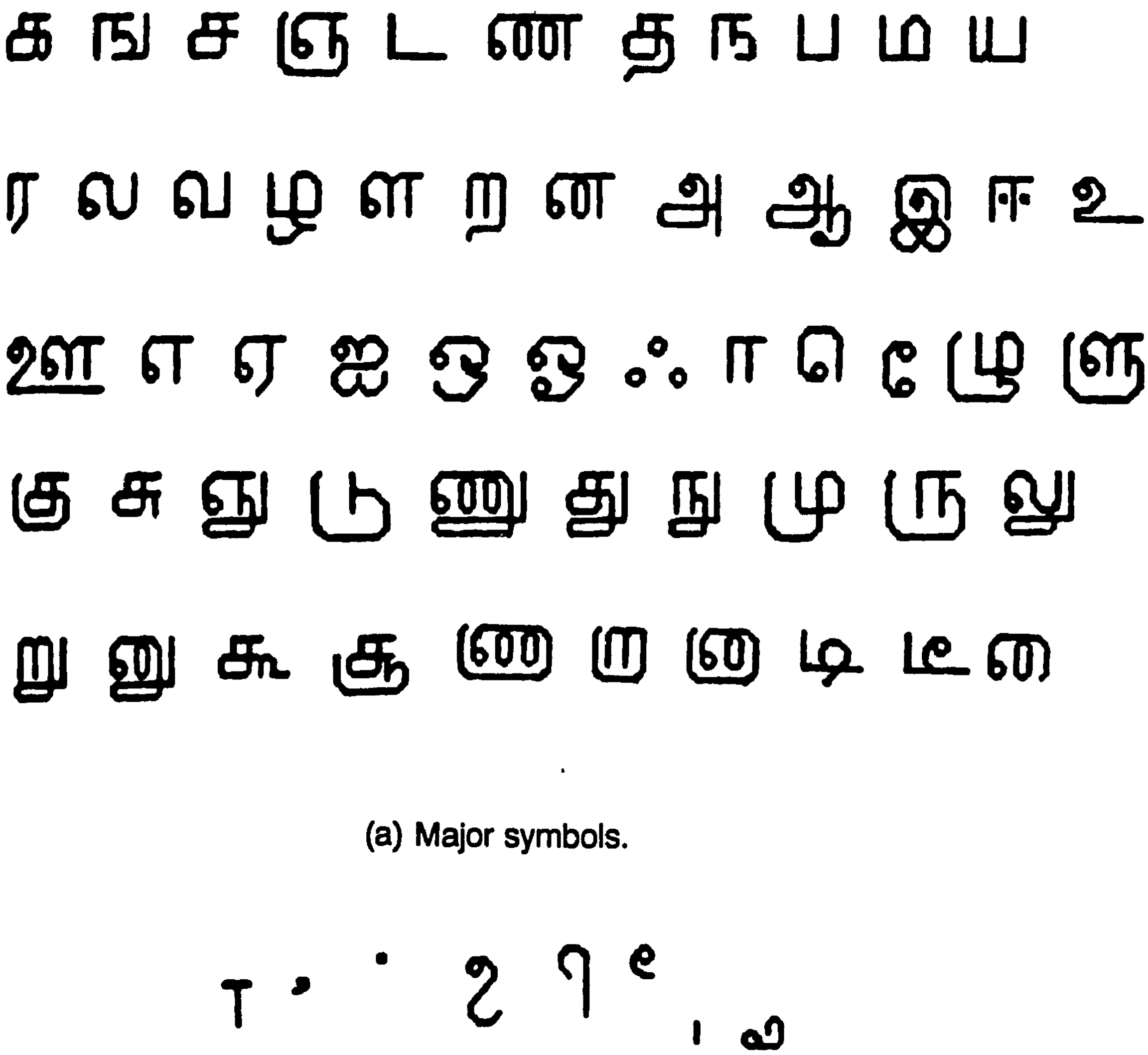
4.3.1 Order of Symbols

For the purpose of construction and searching the decision tree, it is essential that the symbols recognized in Stage I be presented to Stage II as a string in a specific order. Let us illustrate the importance of ordering with an example: In Tamil கீ(K!) is a character constructed from the symbols க and ீ . Stage I, after recognizing the two symbols, could present them to Stage II as either க and ீ or as ி and க. If the decision tree expects the symbols as க followed ி, and Stage I presents them in the reverse order, it is not going to recognize the symbols as a valid character.

The ordering becomes an issue only for those characters that have symbols touching each other (example, கி (KI)) or one below the other (example, க் (K)). Depending on the language, one could order the symbols of such characters in any way one wants since there is no advantage or disadvantage of one ordering over the other.

However, symbols which appear isolated and side by side (example, ெக(KE)) must be presented in the order in which they appear; reordering such symbols may result in wrong character recognition. For example, if a person deliberately presents the symbols of the character ெக (KE) in the order கெ, then it is wrong to reorder and recognize it as a valid character since the sequence கெ is not a valid character.

The Tamil symbols, for the purpose of presentation and construction of the CRT, are divided into two groups: (1) major symbols and (2) minor symbols as shown in Figure 4.2. Stage I will always present the minor symbols first followed by major symbols unless the input pattern dictates otherwise. That is, if the input appears as கி (KI), it will be presented as க followed by ி. On the other hand, if the input appears as ிக, it will be presented to Stage II as ிக.



(a) Major symbols.

(b) Minor symbols.

Figure 4.2 Major and minor symbols of Tamil.

Similar ordering has been done for Malayalam characters as well. The set of Malayalam symbols are shown in Figure 4.3.

4.3.2 CRT: Character Recognition Tree

Let,

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

be the set of symbols in a language.

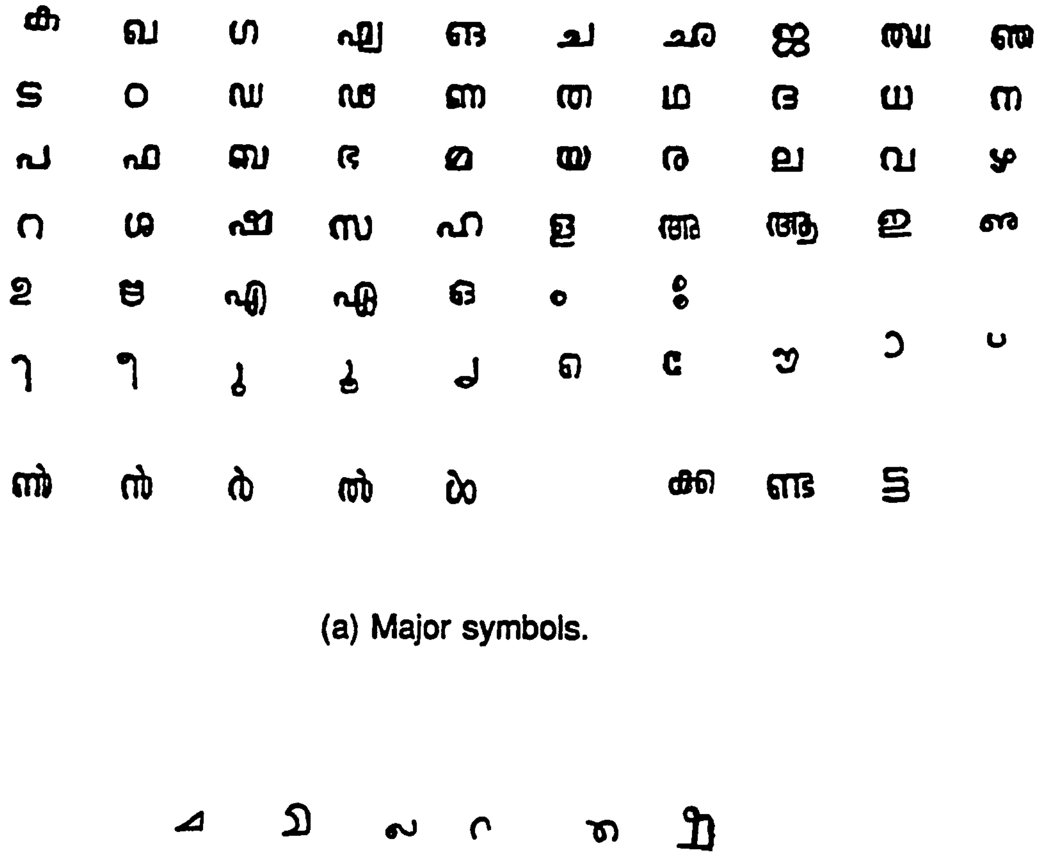
Let,

$$Y = S \cup \{-|\}$$

where -| is the end of text (end of all symbols) indicator. (Let us assume that Stage i will indicate the end of text by -|.)

The CRT is constructed as a multiway tree. In a CRT, each path from the root to a leaf corresponds to one valid character in the language. This way, the nodes of the CRT correspond to the prefixes of the characters in the language.

The data structure for a CRT node is given in Figure 4.4. A CRT node is an array of $n+1$ elements; each element is a CRTnodeItem. The first element of a CRT node corresponds to the first member of Y, the second element corresponds to the second member of Y and so on. Hence, the CRT node is indexed by the members



(a) Major symbols.

(b) Minor symbols.

Figure 4.3 Major and Minor symbols of Malayalam.

```
type
      sym = ($s_1$, $s_2$, ... ,$s_n$, -|);

      structure CRTnodeItem
      {
        char          NodeType;
        short         CharCode;
        CRTnode         *CRTnodePtr;
      };

    CRTnode = array[sym] of CRTnodeItem;
```

Figure 4.4 Data structure for a CRT node.

of Y.

Each CRTnodeItem contains thıee fields: (1) Type, (2) CharCode, and (3) CRTnodePtr.

The 'Type' indicates the status till the current symbol. 'Type' could be any one of the following values:

1. Intermediate
2. Terminal
3. Extendable-terminal
4. Not character

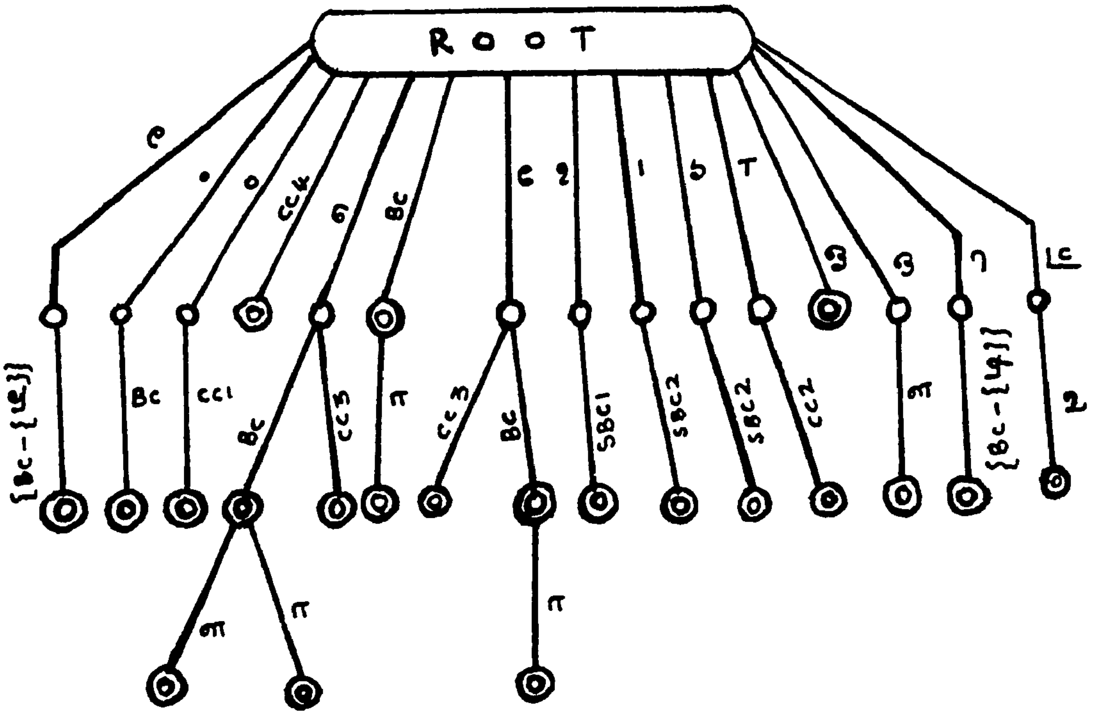
The 'intermediate' type means that the string of symbols encountered so far is not a complete character; however, the string is valid so far. The CRTnodePtr field of the item is a pointer to the next CRT node in the tree to be used for searching. Any value in the 'CharCode' field is not meaningful for this type, hence could be set to a known value, say zero.

The 'terminal' type indicates that the string of symbols including the current symbol constitutes a valid character in the language. The 'CharCode' field of the item corresponding to the current symbol contains the code given to the character for internal representation. The value of CRTnodePtr is not meaningful for this type and hence set to NULL (zero).

The 'extendable-terminal' indicates that the string of symbols encountered so far, including the current symbol, constitutes a valid character. However, the string could be followed by other symbols to make another valid character. For example, கெ(KE) and கொ (KO) are two valid characters in Tamil. In this case, the item corresponding to the symbol க (KA) in the string will be of type 'extendable-terminal'. When the symbol க (KA) is encountered, we cannot immediately declare the character to be KE. The decision can only be made after seeing the next symbol. The CRTnodePtr field of the item points to the next CRT node. The CharCode field of the current item contains the code given to the character for internal representation.

The 'Not-character' type indicates that the string of symbols seen so far, including the current symbol, constitutes an invalid sequence; there is no further path for searching. The CRTnodeptr and CharCode fields for this item are set to NULL and zero respectively. Depending on the input pattern, one or more symbols from the start of the string must be declared invalid and discarded.

The CRT for Tamil characters is given in Figure 4.5. The details of individual items in each node are omitted to accommodate the figure in a single page. It is easy to compute the value of the type fields from the figure. A similar CRT constructed for Malayalam characters is shown in Figure 4.6.



C1 = {அ, ஆ, இ, ஈ, உ, ஊ, எ, ஏ, ஐ, ஒ, ஓ, கூ, கு, சு, ஞ, ச, ஃ, டீ}

CC1 = {பு, மு, ரு, ழு, ளு}

CC2 = {ஞு, ணு, து, நு, ஙு, ஜு, ளு}

CC3 = {ணு, ு, று}

CC4 = {C1 ∪ CC1 ∪ CC2 ∪ CC3}

BC = {க, ங, ச, ஞ, ட, ண, த, ந, ப, ம, ய, ர, ல, வ, ழ, ள, ற, ன}

SBC1 = {ண, ல, ள, ன}

SBC2 = {ங, வ, ப, ய}

Figure 4.5 Character recognition tree for Tamil.

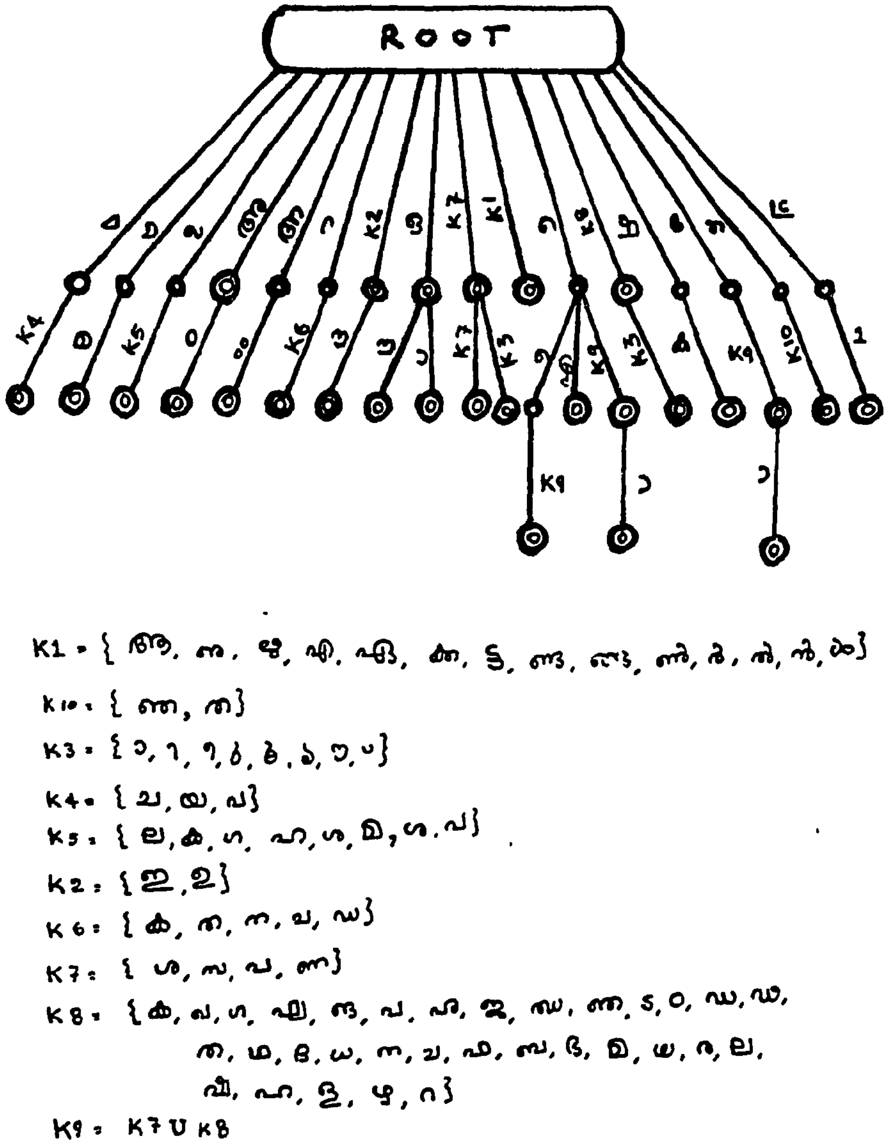


Figure 4.6 Character recognition tree for Malayalam.

Theorem 4.3: The maximum depth of the CRT, for a given language, is κ where κ is the maximum string length of a character in the language. (The length of a string is the number of symbols in the string.)

Proof: Starting from the root, we descend one level down in the tree for every symbol in the string constituting the character. Therefore, the number of levels needed is the same as the length of the string of a character. The maximum number of levels in the CRT will correspond to the character with the longest string which is κ. Hence, the proof.

4.3.3 Operations on CRT

The following operations are defined on the CRT.

- Create new node (CreateNewNode).
- Initialize node (IniNode).
- Set item of a symbol (SetItem).
- Insert the string of a character (InsertCharacter).
- Search for a character (Search).

The CreateNewNode operation creates a new CRT node and returns the pointer to the node. If creation fails, NULL is returned. The IniNode operation initializes all the items in the given CRT node to zero/NULL. The SetItem sets the fields in the CRTNodeItem such as the NodeType, CharCode and CRTnodePtr. The procedures to perform CreateNewNode, IniNode and SetItem are given in Figure 4.7.

4.3.4 Construction of CRT

Given the set of strings corresponding to the character set of a language, the CRT can be constructed by inserting each string into the CRT using the InsertCharacter operation. The procedure for InsertCharacter is given in Figure 4.8. The input parameters to the InsertCharacter procedure are the pointer to the root of the CRT and the string of symbols of the character to be inserted into the CRT. It is assumed that the string is NULL terminated. On the first call to InsertCharacter, the root pointer will be NULL and on return, the root will point to the root CRTnode. In the subsequent calls to InsertCharacter, the root pointer will be non-NULL and it will not be modified. The Insert procedure assumes that the string is valid.

The CRT construction algorithm is given below. Each character string is read and inserted into the CRT. The end of all character strings is indicated by an empty string. After constructing the CRT, that is, when an empty string is read, the algorithm returns the pointer to the root of the CRT. The returned root pointer should be used for searching the CRT.

Algorithm 4.5: CRT Construction Algorithm

```
procedure CRTnode  *CreateNewNode()

declare
   CRTnode   *nodeptr;

begin
   nodeptr = mem_alloc(sizeof(CRTnode));
   return(nodeptr);
end; /* CreateNewNode */

procedure IniNode(nodeptr)

declare
   CRTnode   *nodeptr;
   sym       i;

begin
   for i = $s_1$, to $\dashv$ in sym
      begin
        nodeptr[i].NodeType = Not-character;
        nodeptr[i].CharCode = 0;
        nodeptr[i].CRTnodeptr = NULL;
      end;
end;  /* Ininode */

procedure SetItem(nodeptr, symbol, ntype, ccode, nextnodeptr)

declare
   CRTnode   *nodeptr, *nextnodeptr;
   sym       symbol;
   char      ntype;
   short     ccode;

begin
   nodeptr[symbol].NodeType = ntype;
   nodeptr[symbol].CharCode = ccode;
   nodeptr[symbol].CRTnodeptr = nextnodeptr;
end;  /* SetItem */
```

Figure 4.7 Operations on CRT Nodes.

```
procedure InsertCharacter(crt, string, charcode)

declare
   CRTnode   *crt, *nodeptr, *newnodeptr;
   sym        string[]; /* string is NULL terminated */
   sym        s;
   short      charcode;
   int        i;

begin
   if(sizeof(string = 0))
       return(NOT_OK);  /* error status */
   if(crt = NULL)
     begin
       crt = CreateNewNode();
       IniNode(crt);
     end;
   nodeptr = crt;
   s = string[0];
   for(i=1; i < sizeof(string); i++)
     begin
       if(nodeptr[s].CRTnodeptr = NULL)
         begin
           newnodeptr = CreateNewNode();
           IniNode(newnodeptr);
           SetItem(nodeptr, s, Intermediate, 0, newnodeptr);
         end;
       nodeptr = nodeptr[s].CRTnodeptr;
       s = string[i];
    end;
   nodeptr[s].CharCode = charcode;
   if((nodeptr[s].CRTnodeptr != NULL) and
     (nodeptr[s].NodeType = Intermediate))
       nodeptr[s].NodeType = Extendable-terminal;
   else
       nodeptr[s].NodeType = Terminal;
   return(OK);
end;  /* InsertCharacter */
```

Figure 4.8 The Insert Character procedure.

```
procedure CRTnode   *CRTconstruct()

declare
   CRTnode   *root;
   short     charcode;

begin
   root = NULL;
   read(string, charcode);
   while(string ≠ ∅)
     begin
        InsertCharacter(root, string, charcode);
        read(string, charcode);
     end;
   return(root);
end; /* CRTconstruct */
```

4.3.5 Searching CRT

Given a string of symbols, the task of the search algorithm is to search the CRT from the root and identify whether there is a valid character in the given string. A valid character is a substring of the given string starting from the first symbol. If so, return the character code and the character string. Appropriate status should be returned, if no valid character could be found. The search algorithm is given below.

Algorithm 4.6: CRT Search Algorithm

```
procedure search(crt, string)

declare
   CRTnode   *crt, *nodeptr;
   sym       string[]; /* NULL terminated string */
```

```
  sym     s;
  int     parent_i, i;
  short     parent_CharCode;

begin
  if(sizeof(string = 0)
      return(-2);  /* NULL string status indicator */
  nodeptr = crt;
  parent_i = -1;
  for(i=0; string[i] ≠ NULL; i++)
    begin
      s = string[i];
      switch(nodeptr[s].NodeType)
        begin
          case Terminal:
              return(i, nodeptr[s].CharCode);
          case Extendable-terminal:
              parent_i = i;
              parent_CharCode = nodeptr[s].CharCode;
              nodeptr = nodeptr[s].CRTnodeptr;
              break;
          case Not-character:
              if(parent_i ≠ -1)
                  return(i, parent_CharCode);
              else
                  return(-1); /* Not-character status */
          case Intermediate:
              nodeptr = nodeptr[s].CRTnodeptr;
              break;
        end;
    end;
  if(parent_i ≠ -1)
      return(i, parent_CharCode);
  else
      return(-1);
end;  /* search */
```

Theorem 4.4: The CRT search time complexity is $O(\kappa)$ where κ is the maximum string length of a character in the language.

Proof: For each symbol in the string corresponding to a character, we access one node starting from the root by using the symbol as the index to the array elements in the node. Hence, the number of nodes to be searched is equal to the length of the string of a character. The maximum number of nodes to be searched will correspond to the longest string of symbols of a character which is κ. Therefore, the search time complexity is $O(\kappa)$.

4.3.6 Space for CRT

Theorem 4.5: The CRT space complexity is $O((\delta + 1)(n+1)\tau)$ bytes where δ is the number of distinct suffixes to the first symbols of the set of character strings, n is the number of symbols in the language and τ is the size of the item in bytes.

Proof: The first symbol of each character string is represented by the root node. Every other distinct suffix will be represented by one node in the CRT. Therefore, the number of nodes in the CRT is $(\delta + 1)$. Each node has $(n+1)$ items; the size of each item is τ bytes. Therefore, the size of the CRT is $O((\delta + 1)(n+1)\tau)$.

The amount of space required for the CRT is far more than the total length of the strings of the characters in a language. However, the direct indexing feature of the array representation reduces the search time of the CRT. If minimizing space is more important than search time, the CRT could be represented by a linked list. The linked list representation of the CRT will take more time to search than an array representation.

4.4 Results and Discussion

Various Tamil and Malayalam characters have been tried for recognition. The types of characters used, the recognition results, the analysis of the results, merits and demerits, and future research for possible improvements are discussed in this section.

4.4.1 Test Data

Two test data bases: (1) for Tamil and (2) Malayalam have been used for testing purposes. The Tamil test data contained a mixture of the following types of characters:

1. Printed characters.
2. Handprinted characters
3. Handwritten characters
4. Handprinted bilingual (Tamil and Malayalam) characters.

The Malayalam test data contained printed and handprinted characters only. The authors' limited knowledge and complexity of Malayalam are some of the reasons for the limitations on Malayalam test data. It has been felt that to have true handwritten characters, the subject should have Malayalam has native language. Since the author is not such a subject, the Malayalam test data base did not contain handwritten characters; the omission is not due the recognition methodology or the algorithm.

Most of the composite Malayalam characters were "one consonant and one vowel (*cv*)" type. Only frequently occurring "two consonants and one vowel (*ccv*)" type composite characters were in the data base. The language has provision to generate large number of complex *ccv* characters. It is almost impossible to enumerate all the combinations. Characters were written according to the new script scheme. In this scheme the *cv* type characters obtained by the vowels ഉ (U), ഊ (Ū) and ഋ (Ṛ) are very much simplified like that of the *cv* characters generated by the vowel ഇ (I). Malayalam was primarily included to demonstrate multilingual character recognition. "Not much has been done in the recognition of Malayalam character" is the second reason for selecting Malayalam; any work done in this area will be a contribution to the knowledge of recognition of Malayalam characters.

Bilingual (Tamil and Malayalam) characters were used for multilingual character recognition. Only handprinted characters have been used for this case. The explicit multilingual language coding schemes described earlier was used to distinguish Tamil and Malayalam texts. The digits 1 and 2 were used as the language code for Tamil and Malayalam respectively.

The data bases contained a total of six thousand seven hundred and eighty three symbols of which four thousand one hundred and forty three were Tamil and two thousand six hundred and forty were Malayalam symbols. The Tamil symbols were composed of one thousand six hundred and seventy nine printed, one thousand two hundred and forty four handprinted and one thousand two hundred and twenty handwritten symbols. In the case of Malayalam, one thousand six hundred and twenty six were printed and one thousand and fourteen were handprinted symbols.

The printed characters were taken from different text books. A significant portion comes from the school text books that were in the authors possession. For handprinted and handwritten characters, seven subjects have been used for the Tamil data base and one subject (the author) for the Malayalam data base.

Characters were written/printed on paper. The writing was flexible since there were no line or character position markers on the writing paper. No special restriction was placed on the writing instruments; they were mostly ball and fountain pens. The handprinted characters were constrained in the sense that attention was paid to write clearly, legibly and a bit slower than normal speed. The handwritten Tamil characters were unconstrained. They were written by different subjects. Most of the characters were taken from the letters received by the author from friends and relatives over several years.

A Dest optical scanner has been used to digitize the characters using 4-bit gray scale providing 16 values per pixel. The characters were scanned at a resolution of 300 dots per inch. From the scanned image, characters were segmented manually at appropriate places and the extracted images were written on a file. Using the preprocessing techniques described in Chapter 2, the gray level images were binarized, and noise such as isolated pixels and single pixel gaps have been eliminated. These binary images have been used for feature extraction and classification.

The data pose a variety of problems faced in real life applications like paper quality and color, pen types, ink colors, writing style, etc. The printed characters were of different fonts, shapes and sizes, and so are the handprinted/written characters. The style of writing varied considerably. The quality of some of the writing was very poor. The language has no provision for writing cursive characters which are touching like in English. However, due to variations in writing style, characters sometimes touch and overlap each other. The handwritten test data contained characters which were touching and overlapping. The recognizer assumes that characters and symbols (where applicable) are not touching. Hence, the characters were manually filtered and only fairly good quality, nontouching characters were taken for feature extraction and classification purposes. Some of the characters were distorted in shape. Due to smudging, a few characters have lost their loops. Some of the characters were broken, and a few had unexpected/extra loops or open loops and islands.

Various samples of Tamil and Malayalam characters from the data bases are shown in Figures 4.9 and 4.10 respectively.

4.4.2 Recognition Results and Analysis

The recognition results of Tamil and Malayalam characters are given in Tables 4.3 and 4.4 respectively according to various categories of characters mentioned above. Considering the complexity and versatility of input characters, the recognizer performed very well. The language code has been recognized 100% (may be due to careful writing). As one would expect the percentage of successful recognition decreased from printed characters to handwritten characters with the corresponding increase in the percentage of rejection and substitution errors.

Characters were rejected due to several reasons. They are listed below:

1. Distortion in the shape of the characters
2. Broken characters
3. Unexpected loops
4. Unwanted limbs from the thinning algorithm due to the distortion in the shape

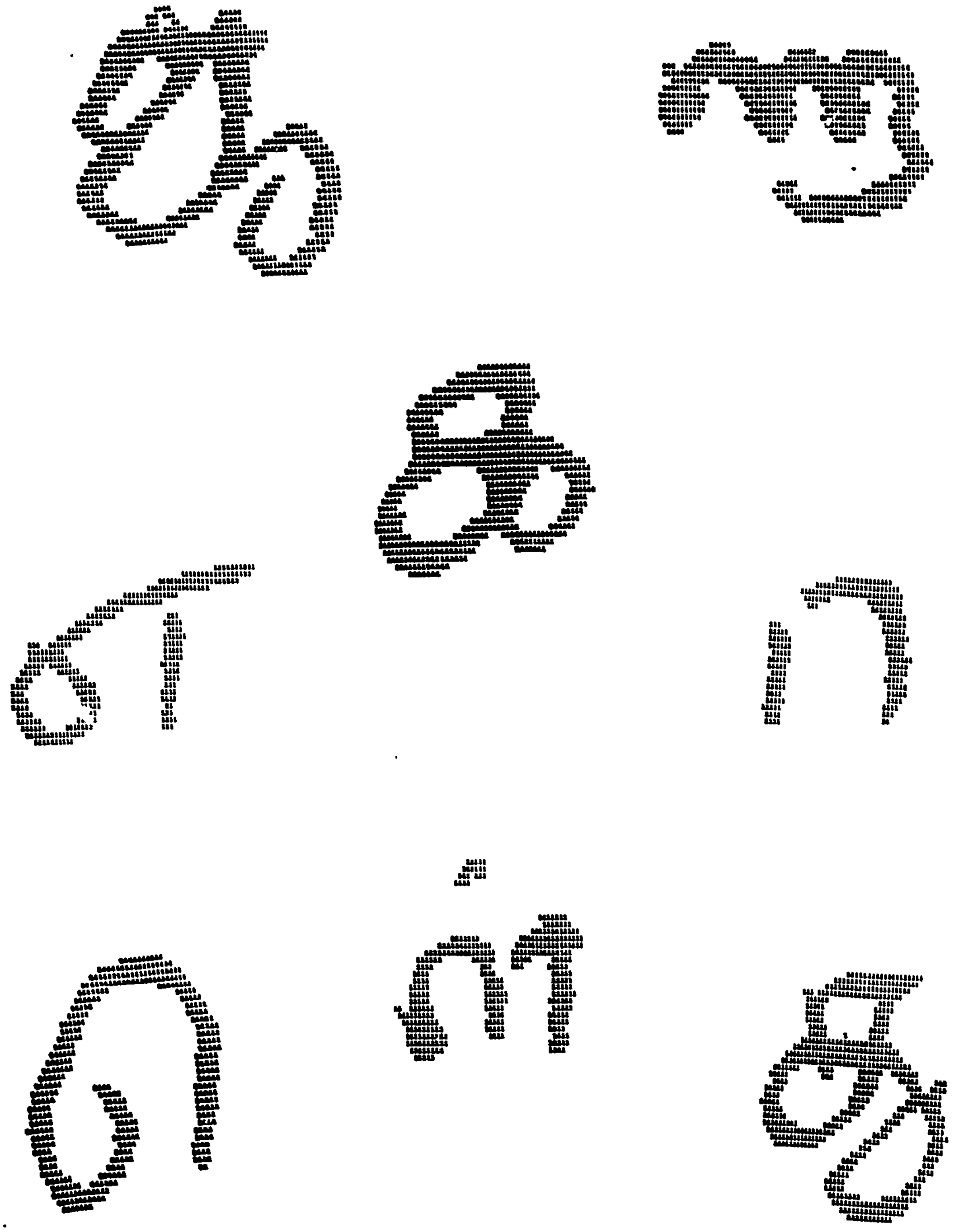


Figure 4.9 Samples from Tamil test data base.

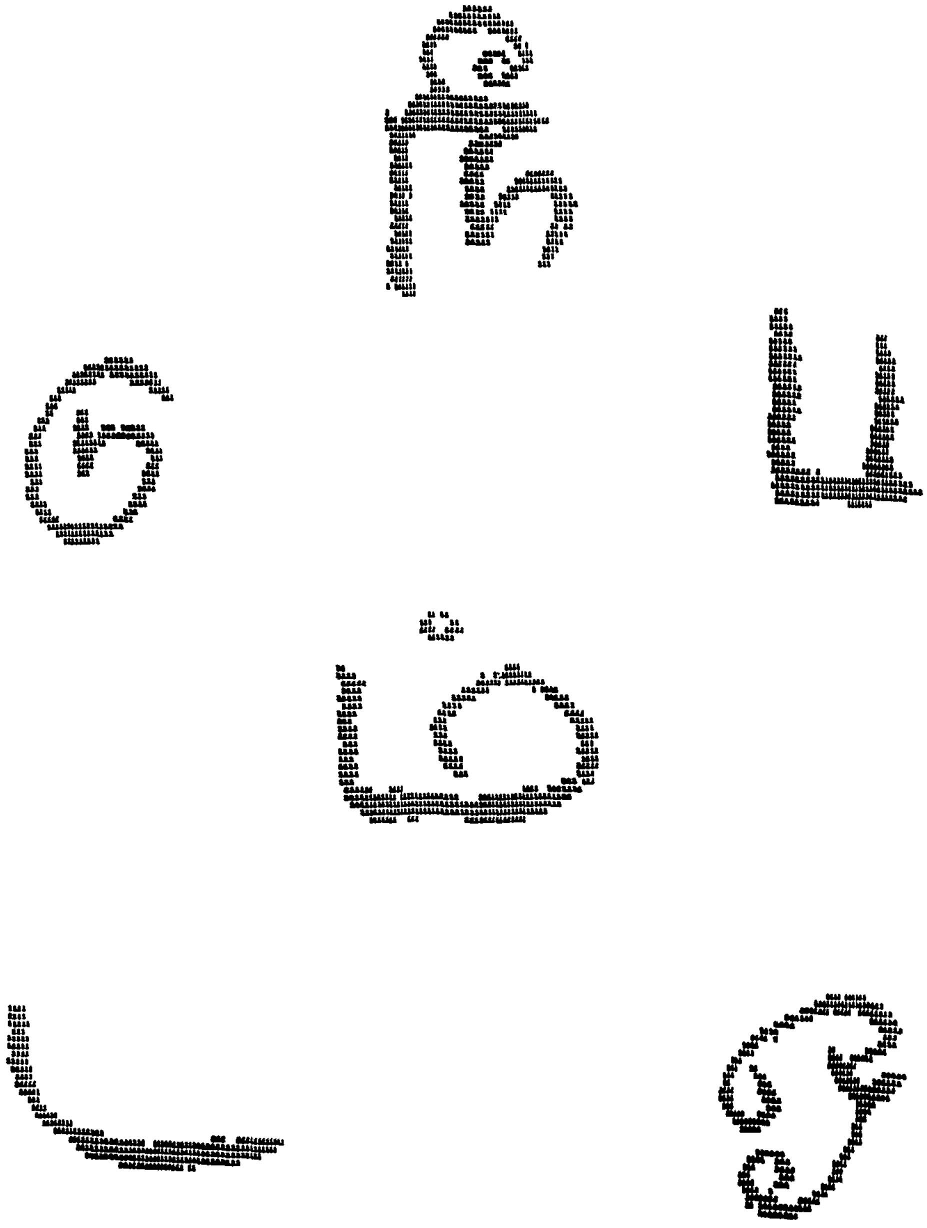


Figure 4.9 Samples from Tamil test data base (contd.).

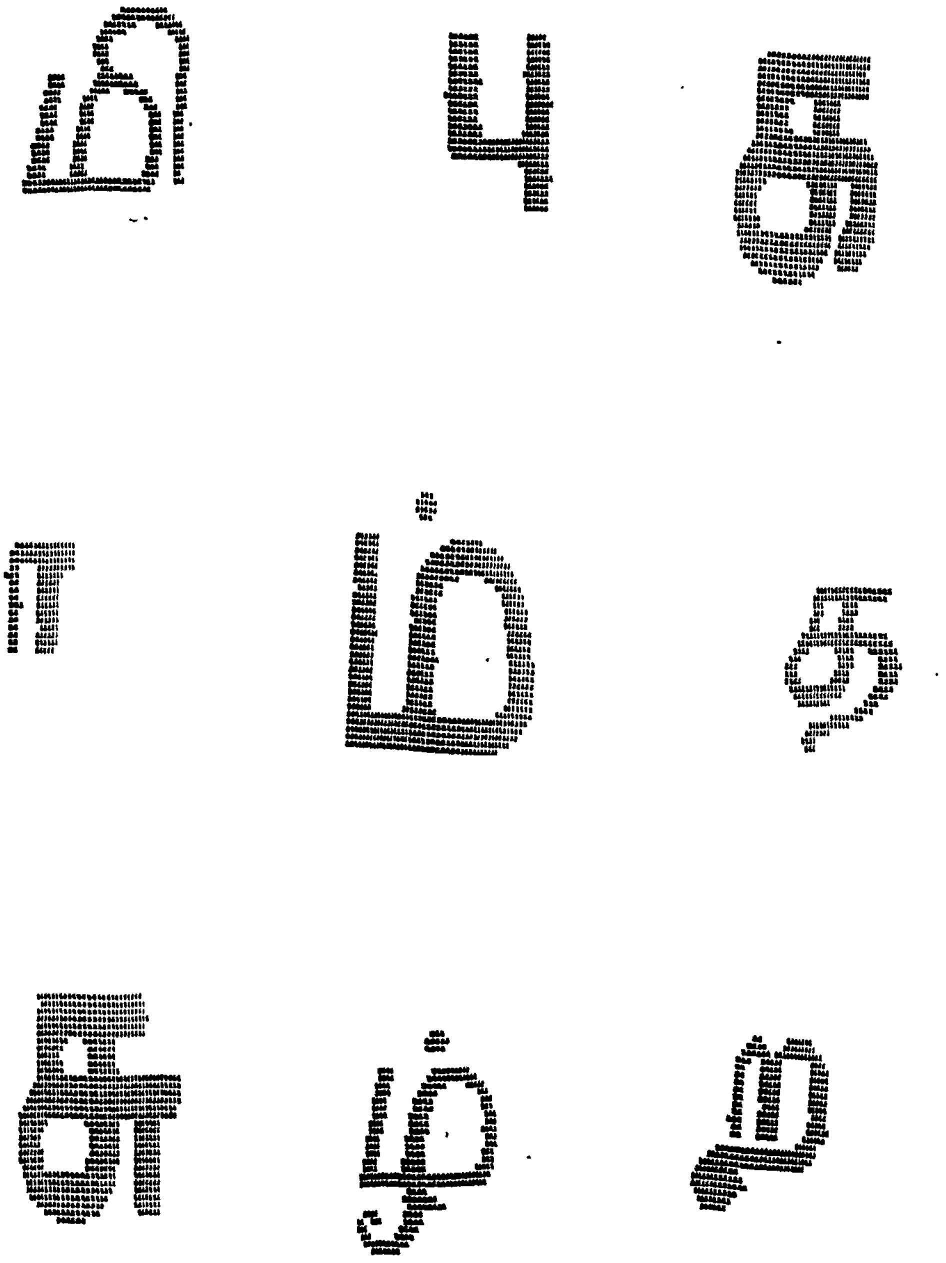


Figure 4.9 Samples from Tamil test data base (contd.).

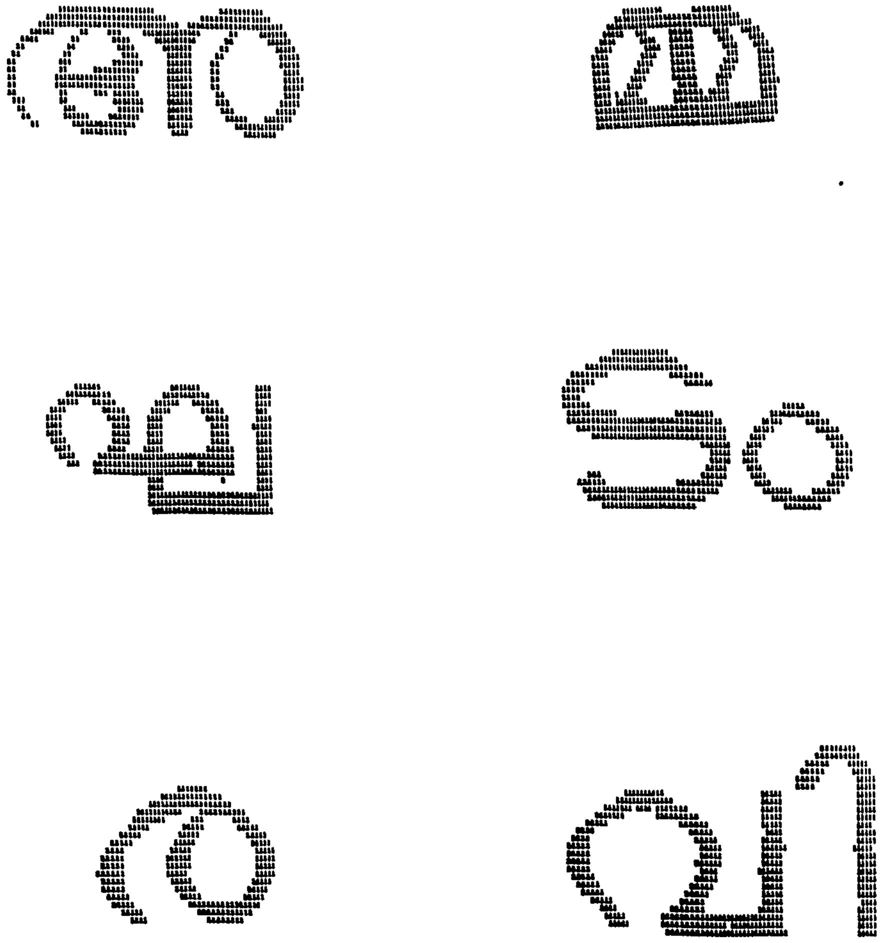


Figure 4.10 Samples from Malayalam test data base.

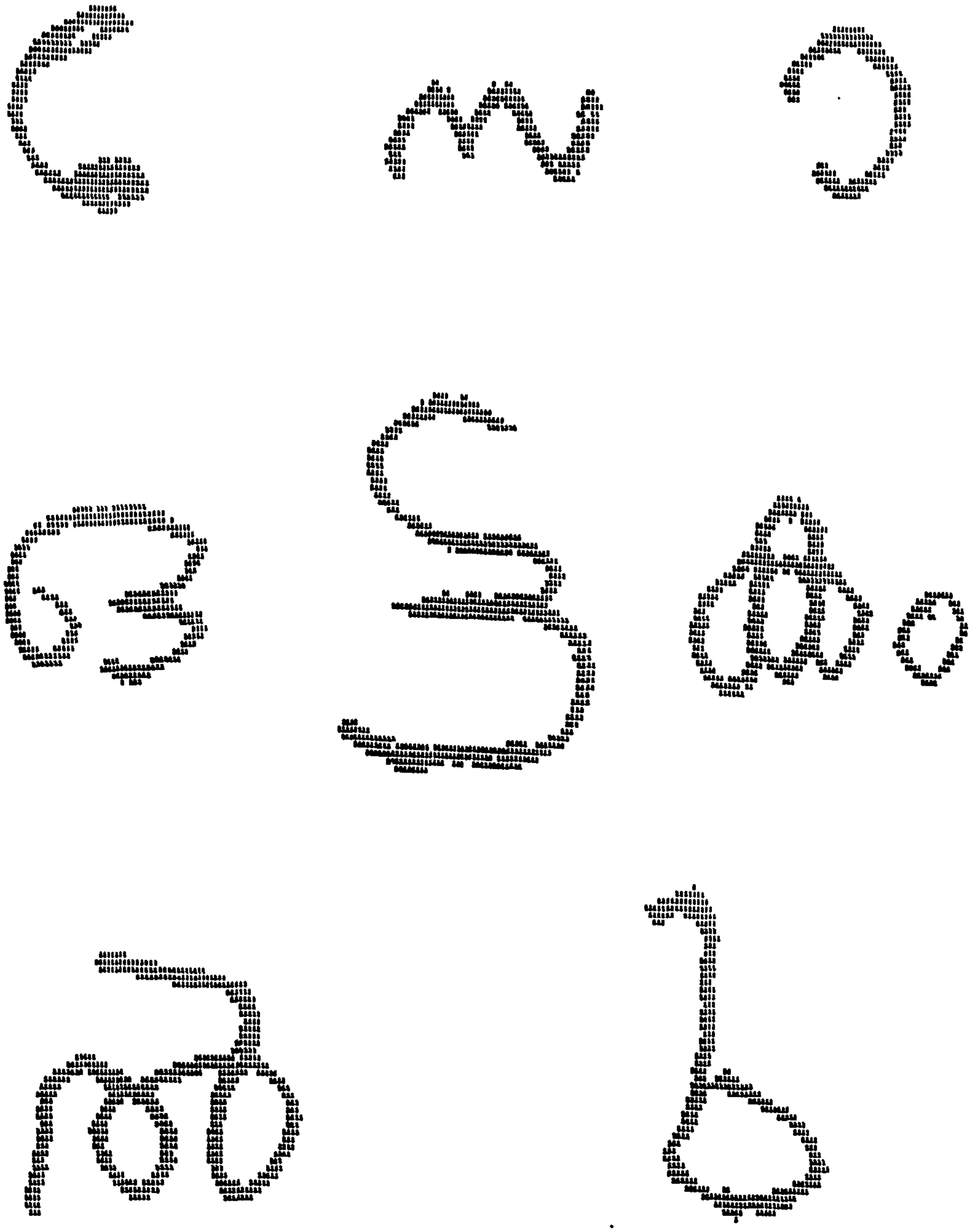


Figure 4.10 Samples from Malayalam test data base (contd.).

Table 4.3 Recognition result for Tamil.

Char. Type	Successful Recognition	Rejection	Substitution
Printed	97.0%	2.5%	0.5%
Handprinted	91.0%	7.0%	2.0%
Handwritten	86.0%	11.0%	3.0%
Average	91.33%	6.83%	1.83%

Table 4.4 Recognition result for Malayalam and language code.

Char. Type	Successful Recognition	Rejection	Substitution
Printed	95.0%	4.0%	1.0%
Handprinted	84.0%	12.6%	3.4%
Average	89.5%	8.3%	2.2%
Lang. Code	100.0%	0.0%	0.0%

5. Deficiency in extracting the curve primitives effectively
6. Rejection of a symbol in a multisymbol character.

Substitution errors occurred due to several reasons. It occurred due to part of a broken character being recognized as a valid character. Rejection and/or substitution of a symbol in a multisymbol character resulted in substitution error. For example, in the Tamil characters, the rejection of symbols such as ா , ெ , ே , etc., in the multisymbol characters such as கா , கெ , கே , கொ resulted in the substitution error. As an illustration, if the symbol ா is rejected in the character கொ (KO), then the character is recognized as கெ (KE). In the characters like கௌ (KAU), rejection of ெ , results in the recognition of the remaining symbols as two characters க (KA) and ள (LLA); rejection of ள (LLA) results in the recognition of கெ (KE), and the rejection of க (KA) results in the rejection of ெ and the recognition of ள (LLA). Figure 4.9 contains some such characters. Substitution errors have also occurred in the symbols having considerable amount of similarities like ண and ன (NA), க (KA) and சு (CU), மு (MU) and மூ (MUU). Confusion occurred between உ (U) and ௨ (TWO). However, it has been resolved successfully by performing a context checking, that is, to see whether the previous symbol is a language code symbol. If it is, then the current symbol is treated as TWO instead of U when the character is confused. Similar confusion among multisymbol characters occurred in the case of Malayalam also.

Confusion matrices have been constructed from the substitution errors. Tables 4.5 and 4.6 are the confusion matrices for the Tamil and Malayalam respectively. To avoid lengthy listing of large number of characters, only symbols are used in the confusion matrix. Even though it may not directly give the substitution errors of some of the multisymbol characters, we can easily infer that from their symbols. The numerical value in the matrices indicate how many times a symbol has been rejected or confused as another symbol.

4.4.3 Merits and Demerits

The preprocessor designed to convert the input image to directed attribute graph is very basic. It has the following advantages and disadvantages. By using the thinning algorithm of Stafanelli and Rosenfeld, the preprocessor succeeds in the case of input characters having variable line thickness. By the labelling scheme developed, small distortions and rotations of the primitives in the symbols are taken care of. The output of the preprocessor is independent of the sizes of various primitives in the image.

The preprocessor is not efficient in extracting the curved primitives since the approximation of curves to straight lines as shown in Figure 3.1 is not sufficiently accurate. It is very sensitive to distortions in the curved primitives.

Table 4.5 Confusion matrix for Tamil.

Actual symbol			Success #	Rejected #	Confused as		
code	sym	#			sym, #	sym, #	sym, #
A	அ	40	37	3			
Ā	ஆ	17	15	2			
I	இ	30	26	4			
Ī	ஈ	12	11	1			
U	உ	20	19	1			
Ū	ஊ	14	12	2			
E	எ	30	26	2	sE 1	Ē 1	
Ē	ஏ	15	14	1			
AI	ஐ	12	11	1			
O	ஒ	17	15	2			
Ō	ஓ	15	14	1			
AK	ஃ	9	9	0			

Table 4.5 Confusion matrix for Tamil (contd.)

Actual symbol			Suc-cess #	Reje-cted #	Confused as		
code	sym	#			sym, #	sym, #	sym, #
K	க	266	239	20	CU 7		
Ṅ	ங	37	27	7	sAA 3		
C	ச	86	78	8			
Ñ	ஞ	22	18	4			
Ṭ	ட	97	87	7	P 3		
Ṇ	ண	43	31	12			
T	த	235	213	22			
N	ந	82	70	6	R 6		
P	ப	171	166	5			
M	ம	142	133	9			
Y	ய	110	96	9	P 5		
R	ர	87	75	5	N 7		
L	ல	94	85	9			
V	வ	154	138	10	L 6		
Z	ழ	29	22	7			
Ḷ	ள	70	56	11	sE 3		
R̤	ற	77	66	7	N 4		
N̤	ன	197	180	12	sAI1 5		

Table 4.5 Confusion matrix for Tamil (contd.)

Actual symbol			Suc-cess #	Reje-cted #	Confused as		
code	sym	#			sym #	sym #	sym #
sCon	்	492	482	10			
sAA	ா	165	154	9	sI 2		
sI	ி	221	214	7			
sII	ீ	71	61	7	sI 3		
sU	ு	37	37	0			
sUU1	ூ	19	16	3			
sUU2	ூ	23	21	2			
sUU3	ூ	22	20	2			
sE	ெ	167	162	5			
sEE	ே	129	123	6			
sAI1	ை	95	79	7	N̤ 5	N2AA 3	L 1
sAI2	ை	12	11	1			
LC	ட	20	20	0			
TWO	உ	20	20	0			
TI	டி	25	22	2	M 1		
TII	டீ	15	14	1			
KU	கு	12	11	1			
CU	சு	15	14	1			
NU	ஞு	6	5	1			
TU	டு	35	33	2			
N3U	ணு	11	10	1			
THU	து	65	58	7			
NU	நு	15	12	1	RRU 2		
MU	மு	15	14	1			
RU	ரு	53	45	4	RAA 4		
LU	லு	17	15	2			
ZU	ழு	14	12	1	MU 1		
LLU	ளு	17	16	1			
RRU	று	27	24	2	NU 1		
N2U	னு	17	15	1	RRU 1		
KUU	கூ	12	11	1			
CUU	சூ	11	10	1			
RAA	ரா	10	9	1			
N3AA	ணா	12	11	1			
N2AA	னா	16	14	1	sAI1 1		

Table 4.6 Confusion matrix for Malayalam.

Actual symbol			Success #	Rejected #	Confused as		
code	sym	#			sym, #	sym, #	sym, #
A	അ	15	12	2	Ā 1		
Ā	ആ	10	9	1			
I	ഇ	14	12	2			
U	ഉ	12	10	1	Ḷ 1		
Ṛ	ഋ	4	4	0			
L'	ഌ	4	4	0			
E	എ	11	10	1			
Ē	ഏ	10	9	1			
O	ഒ	12	11	1			
AM	ം	25	19	2	ṬH 4		
AH	ഃ	5	5	0			

Table 4.6 Confusion matrix for Malayalam (contd.)

Actual symbol			Suc-cess #	Reje-cted #	Confused as		
code	sym	#			sym, #	sym, #	sym, #
K	ക	66	59	7			
KH	ഖ	45	41	3	V 1		
G	ഗ	39	34	4	S' 1		
GH	ഘ	40	32	7	ṖH 1		
Ṅ	ങ	37	29	6	D 1	sE 1	
C	ച	55	51	2	P 1	V 1	
CH	ഛ	41	37	4			
J	ജ	39	34	4	I 1		
JH	ഝ	40	31	7	DH 1	miT 1	
Ñ	ഞ	42	32	9	T 1		
Ṭ	ട	37	35	2			
ṬH	ഠ	30	25	2	AM 3		
Ḍ	ഡ	40	35	3	DH 2		
ḌH	ഢ	35	31	4			
Ṇ	ണ	54	44	9	NTA 1		
T	ത	61	56	5			
TH	ഥ	40	37	2	PH 1		
D	ദ	43	35	6	BH 1	O 1	
DH	ധ	43	39	4			
N	ന	39	34	3	R̤ 1	miN 1	
P	പ	50	45	3	V 1	C 1	
PH	ഫ	41	36	4	TH 1		
B	ബ	39	30	7	V 1	sE 1	
BH	ഭ	36	32	3	D 1		
M	മ	50	47	3			
Y	യ	35	31	4			
R	ര	50	47	2	R̤ 1		
L	ല	37	34	3			
V	വ	40	36	2	P 2		
S'	ശ	36	34	1	G 1		
Ṣ	ഷ	25	21	4			
S	സ	40	36	4			
H	ഹ	33	32	1			
Ḷ	ള	30	23	6	U 1		
L̤	ഴ	29	27	2			
R̤	റ	46	44	2			

Table 4.6 Confusion matrix for Malayalam (contd.)

Actual symbol			Suc-cess	Reje-cted	Confused as		
code	sym	#	#	#	sym #	sym #	sym #
sA	ാ	119	109	6	TH 3	AM 1	
sI	ി	125	123	2			
sĪ	ീ	84	79	3	sI 2		
sU	ു	95	91	4			
sŪ	ൂ	63	53	7	sU 3		
sṚ	ൃ	39	34	3	sŪ 2		
sE	െ	90	82	6	Ṛ̤ 1	miT 1	
sĒ	േ	81	76	5			
sAU	ൗ	75	70	5			
sCon	്	105	96	7	AM 2		
miY	്യ	10	9	1			
miM	്മ	5	4	1			
miL	്ല	6	6	0			
miN	്ന	20	17	1	R 2		
miT	്ത	14	12	1	sE 1		
miS	്ഥ	4	4	0			
hcL	ള	5	4	1			
hcT	ത്ത	11	9	2			
hcṆ	ണ്ണ	9	7	2			
hcN	ന്ന	11	9	2			
hcR	റ്റ	15	14	1			
KKA	ക്ക	11	9	2			
TTA	ട്ട	9	9	0			
NTA	ണ്ട	9	6	2	BH 1		
Lc	ൾ	10	10	0			
One	൧	10	10	0			

Errors in the DAG of the input image affect the computed correlation coefficients which in turn affects the recognition result. The preprocessor and the classifier work well with variations in the size and shape of the characters.

The preprocessing, feature extraction and building the DAG are common to all languages. They could be used for other Indian languages and European languages. The features used may not be directly applicable for Chinese characters since in Chinese strokes are used to form characters. However, the concept of DAG could be used to represent the Chinese characters. Stages I and II of the classifier are language dependent; State II is more language specific than State I. However, the proposed methodology and the algorithms can be used for other languages too. A separate knowledgebase and CRT must be built for each language.

4.4.4 Future Research for Improvements

The symbol matching program can be modified to compute the correlation coefficients taking into account the possible rotational transformation on the input image. Suppose we find that junctions x_1 and x_2 of symbol Ω correspond to junctions y_1 and y_2 respectively of Ψ and the junction pairs (x_1,x_2) and (y_1,y_2) are connected by primitives. Let the primitive connecting x_1 to x_2 be p_1 and the primitive connecting y_1 and y_2 be p_2. Then by finding out the transformation T needed so that the primitive p_1 can correspond to p_2, we can extend the correspondence by the following rule:

"If the junction x_1 of symbol Ω corresponds to the junction y_1 of Ψ, then the junction connected to x_1 by a primitive will correspond to a junction connected to y_1 by the transformed primitive T(p)."

This should enable us to identify characters that are rotated by 90 degrees, for instance.

To further improve the successful recognition rate, additional structural features such as loop ,could be used. Different schemes which are less sensitive to distortion could be used to extract curved primitives. Special or additional processing such as statistical classification, parallel processing, etc., could be applied to the rejected or all characters. Application of statistical methods will require extraction of statistical features; it will also add an order of magnitude of complexity to the recognition system. Parallel processing will improve the throughput with increased cost of the product.

The classifier assumes that the characters (and symbols where applicable) are isolated, thus avoiding the problem of segmenting the characters. However, handwritten characters sometimes touch and overlap each other. The recognizer could be enhanced to take care of segmentation.

Another enhancement that is possible is to use contextual feature such as syllables, words, syntax, etc., of the language. Again this will add different dimension and complexity to the system.

In the case of Malayalam, more *ccv* characters can be included in the test data. Accordingly, the Malayalam CRT could be improved to recognize such *ccv* characters.

5. MULTILINGUAL TERMINAL DESIGN

DETERMINATION OF CHARACTER SIZE

Present day computers are capable of displaying and printing Latin characters in the form of dot matrix patterns. Laser printers with high resolutions are now widely used. Some efforts have also been made to enable the computers to handle non-Latin characters such as Chinese, Japanese, Russian, Indian characters (Balasubramanian [1990]), etc. Due to the graphic properties of these characters, the dot matrix type of output devices seems to be more suitable than the regular type-face. Hence, the dot matrix type output device is considered for the multilingual terminal.

The design of a multilingual (hardcopy or softcopy) terminal involves the following aspects:

- Analysis of the character sets of the languages to be supported
- Decomposition of characters into symbols
- Design of keyboard for manual text input
- Determination of dot matrix character size, if output device is dot matrix
- Determination of dot matrix character set, if output device is dot matrix
- Character generation

The analysis of scripts and the theory of symbolization have been covered in Chapter

1. More on symbolization will be covered in Chapter 7. The methodology proposed for the determination of dot matrix character size is explained in this chapter. A scheme for the determination of dot matrix character set is explained in the next chapter. Algorithms developed for character generation are described in Chapter 7.

5.1 Introduction

The parameters to be considered in designing dot matrix characters are character size, luminance, font, display area, dot size, dot shape, dot spacing, number of dots, etc. Significant research has been conducted to determine these parameters for English fonts (SID [1980]). It is yet in the infant stage for Indian characters. Certain results from the experiments conducted for English characters such as dot shape, dot size, etc., could be used in designing fonts for Indian characters. However, the 5x7 and 7x9 dot matrix size commonly used for Latin characters are not suitable for Indian characters because they are complex graphemes. Therefore, to develop dot matrix type output devices capable of handling Indian, in general non-Latin, characters, it is necessary to identify an optimal dot matrix size to code the character patterns required to generate characters of Indian languages.

Manual coding of the dot matrix patterns for each and every character of a language, assuming an arbitrary dot matrix size, is very tedious and time consuming. Above all the dot matrix size may not be optimal. To store these characters as they are for character generation (printing/displaying) will require a large amount of storage. Therefore, graphic symbols, which require less storage, will be used to generate the characters. By suitably composing one or more symbols, any character in the language could be generated.

In this chapter, a computer-aided interactive method to determine a nearly optimal dot matrix size is proposed. A new specialized knowledge based thinning algorithm based on the techniques of pattern recognition and graph theory has been developed to skeletonize the symbols. This algorithm is different from the thinning algorithm proposed in Chapter 2. Experiments have been conducted using a graphic terminal and the symbols of the Tamil language shown in Figure 5.1.

5.2 Definitions

The definitions of a few terms used in this chapter are given below.

Definition 5.1: Nearly Optimum Dot Matrix Size

The dot matrix size of a character/symbol is said to be nearly optimum if the following two conditions are satisfied:

1. The character/symbol size should be as small as possible so that it will require minimum space for storage, display, and printing.
2. Each character/symbol must be legible to human eyes as a separate

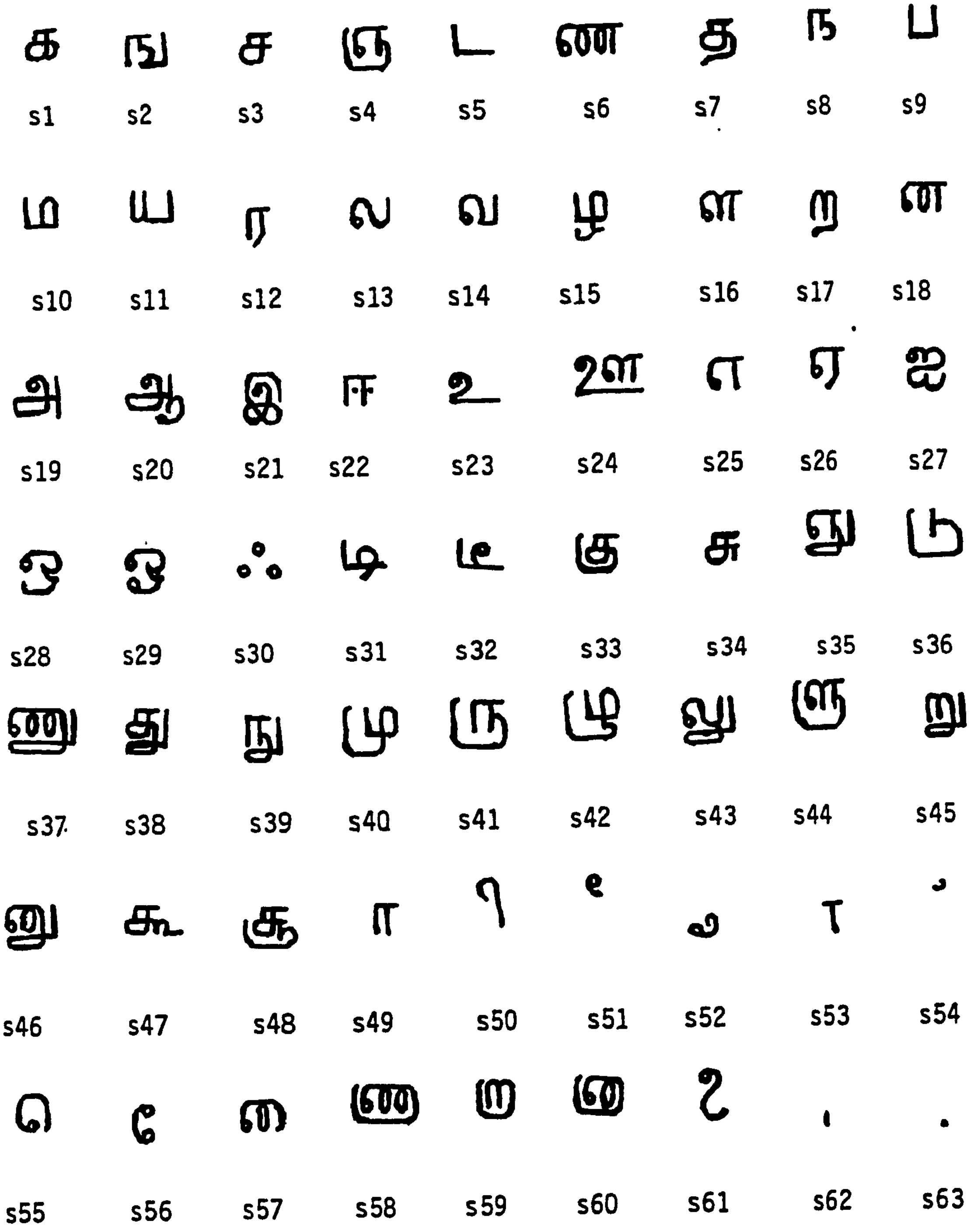


Figure 5.1 Set of Tamil symbols used for size computation.

entity.

What is the 'smallest' and 'legible' size depends on the individual as well as the equipment used. Hence, the 'nearly optimal size', as defined here, is to certain extent subjective.

Definition 5.2: Character Pattern

Let

$$P = p_{ij}, \quad i=1 \ldots m, \quad j=1 \ldots n$$

be the binary picture pattern of a symbol, where p_{ij} is a pixel in row i and column j. p_{ij} = '.' (dot) when it belongs to the character and p_{ij} = 'b' when it belongs to the background, where 'b' is a blank (space).

Definition 5.3: Component

A component is a set of consecutive dots in a row.

$$c_{ij} = \{p_{ik} \mid p_{ia} = p_{ib} = \text{'}b\text{' and } p_{ik} = \text{'.'}\},$$

where c_{ij} is the j^{th} component in the i^{th} row and $a<k<b$.

Definition 5.4: Component Length

The length of a component $L(c_{ij})$ is given by

$$L(c_{ij}) = b\text{-}a\text{-}1,$$

where a and b are as in the 'component' definition 5.3.

Definition 5.5: Component Connectivity

Two components c_{ij} and c_{i+1k} are said to be connected if any one of the following conditions are satisfied:

1. $\{a, a+1, \dots b\} \cap \{d, d+1, \dots e\} \neq \{\emptyset\}$,
2. $d = b$-1,
3. $a = e$-1,

where

$$c_{ij} = \{p_{ia}, p_{ia+1}, \cdots p_{ib}\}$$

and

$$c_{i+1k} = \{p_{i+1d}, p_{i+1d+1}, \cdots p_{i+1e}\}.$$

It is denoted as $c_{ij} <=> c_{i+1k}$.

Definition 5.6: Segment

A segment is a set of connected components satisfying the conditions specified in the 'segmentation process' described later. In other words, $c_{pq} \epsilon s_i$, $c_{rs} \epsilon s_i$ => $c_{pq} \Re c_{rs}$.

Definition 5.7: Segment Number

Every segment is assigned a unique number. The function $S(c_{pq})$ gives the segment number to which the component c_{pq} is assigned.

The function *NS*() gives the next number that could be assigned to a new segment.

5.3 Symbol Size Determination Method

The methodology for determining the optimal size is shown in Figure 5.2. This method does not involve manual coding of symbols: instead, the printed symbols of the language under study are digitized and binarized using a digital scanner and binarization technique stated in Chapter 2. (It should be pointed out that printed characters are available for Indian languages.) This method makes use of some of the pattern recognition techniques, graph theory, and a-priori defined knowledge base to get a smoothly thinned symbol.

Most of the time the binary patterns contain noise due to ink smudging, distortion, etc. To eliminate part of the noise, single gaps are filled using Algorithm 2.2 and isolated pixels are removed using Algorithm 2.3 described in Chapter 2.

A number of generalized thinning algorithms have been reported in the literature for pattern recognition (Dessimoz [1980], Zhang and Wang [1988], Xia [1988], Ogawa and Taniguchi [1982], Hilditch [1969], Stefanelli and Rosenfeld [1971]). They do not require any knowledge of the input pattern for thinning. However, they tend to give unwanted limbs (refer Figure 5.3) due to noise in the digitized input pattern. These

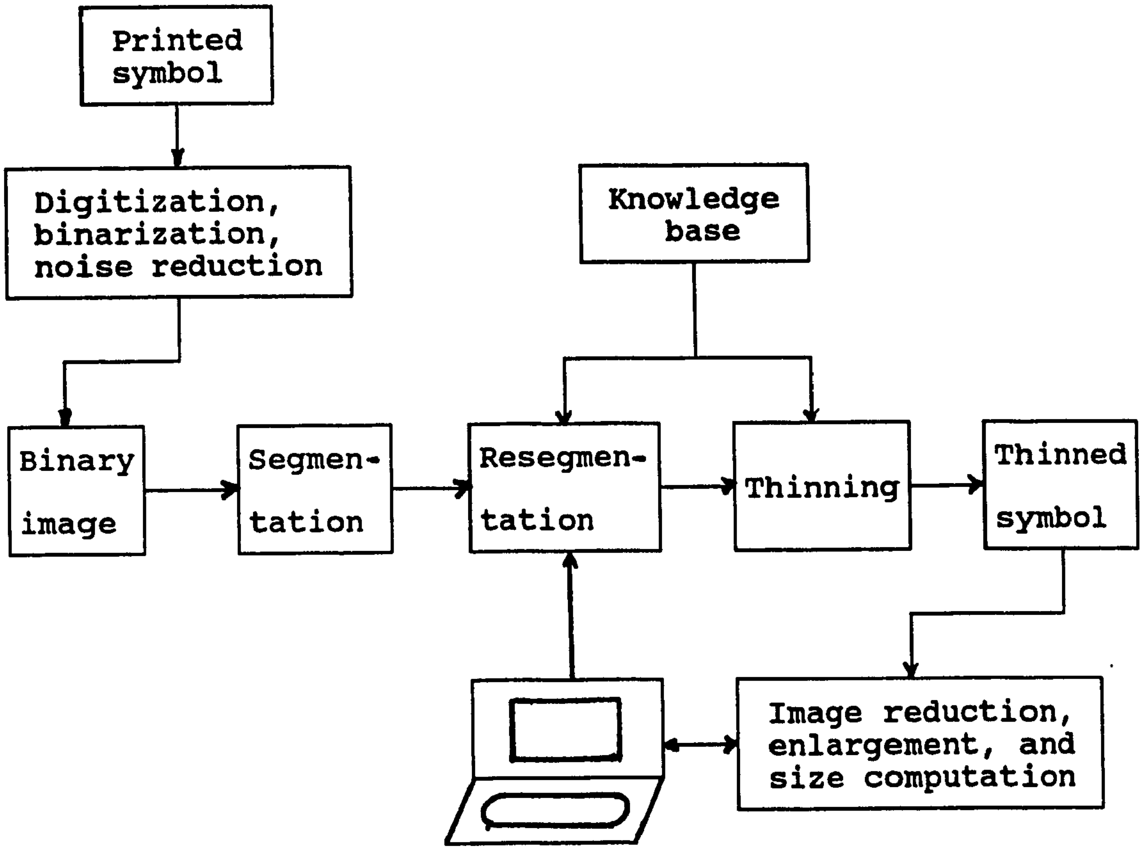

Printed symbol
Digitization, binarization, noise reduction
Knowledge base
Binary image
Segmen- tation
Resegmen- tation
Thinning
Thinned symbol
Image reduction, enlargement, and size computation

Figure 5.2 Automatic method of size determination.

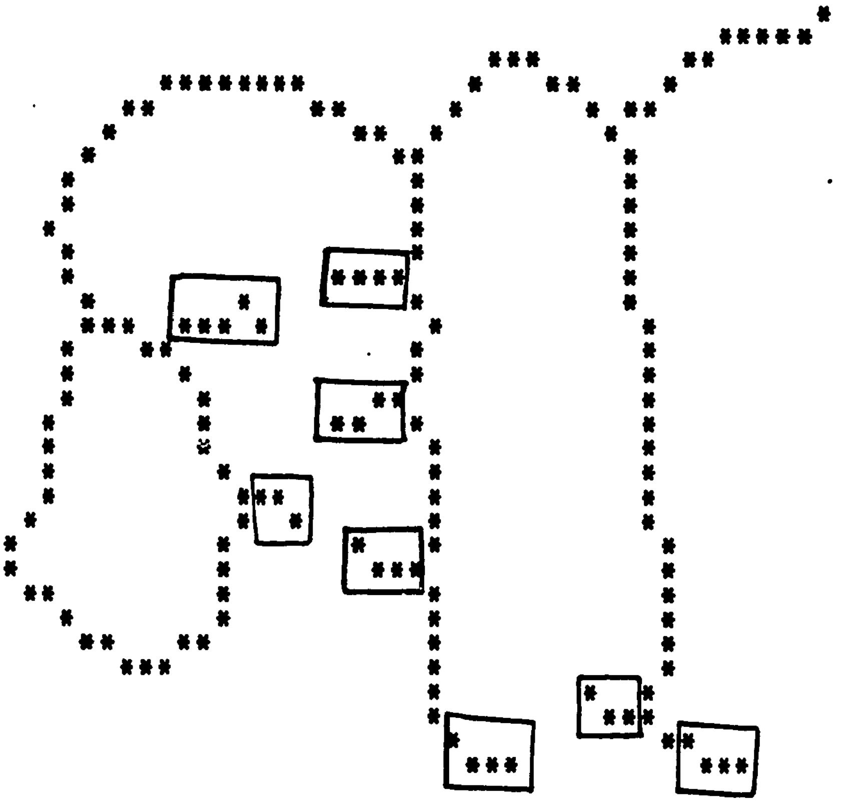


Figure 5.3 Unwanted limbs in ला (LA).

limbs hinder the experimental results considerably, hence they need to be eliminated. A structural knowledge of the input symbols is required to eliminate these limbs and to smooth the symbol shape. The knowledge based thinning algorithm which eliminates the unwanted noise to give a smooth pattern is described in the next section. The thinned symbols are studied using the algorithm explained later in this chapter. The new thinning algorithm is explained in the next section.

5.4 Thinning Process

The thinning process consists of three stages. In the first stage the input symbol is segmented. In the second stage the segmented input symbol is resegmented using the a-priori knowledge base and the symbol identification supplied by the experimenter. In the last stage the segments are skeletonized. These three stages are described in the following paragraphs.

5.4.1 Segmentation Process

The character pattern P is scanned row by row from top to bottom. For the purpose of segmentation, the pixels p_{i-1} and p_i of two consecutive rows are considered at a time for processing. i-1 is the previous row and i is the current row under processing. This approach has the advantage of less memory requirement. The components in p_{i-1} and p_i are identified and numbered separately as 1, 2, 3 and so on.

In order to identify the connectivity of components in two rows, a component connectivity matrix (*c*-matrix) *CC* is constructed as shown below:

$$CC = [cc_{pq}] \qquad p=1, \ldots m, \;\; q=1 , \ldots n,$$

where

$$cc_{pq} = \begin{cases} 1, & \text{if } c_{ip} <=> c_{(i-1)q} \\ 0, & \text{otherwise} \end{cases}$$

m and *n* are the number of components in the (*i*-1)th row and *i*th row, respectively.

From *CC*, two additional vectors, α and β are computed

$$\alpha_k = \sum_{j=1}^{n} cc_{kj} \qquad k=1, 2, \ldots ,m$$

$$\beta_k = \sum_{j=1}^{m} cc_{jk} \qquad k=1, 2, \ldots ,n$$

α_k indicates that the k^{th} component in p_i is connected to β_k number of components in p_{i-1}. The reverse is indicated by the values of β.

To begin with each component in the first row is assigned a new segment number and the row is declared as the (*i*-1)th row. Then, the components in the current row (*i*th row) are assigned segment numbers. The status of segments and components are updated. In the process of assigning components to segments, any of the following five states may arise:

1. start of the segment at the *i*th row,

2. continuation of a segment from the $(i\text{-}1)^{th}$ row to the i^{th} row,
3. termination of a segment at the i^{th} row,
4. merging of two or more segments from $(i\text{-}1)^{th}$ row with the i^{th} row and,
5. splitting of a segment in the $(i\text{-}1)^{th}$ row into two or more segments at the i^{th} row.

In state (4), the merging segments are terminated at the $(i\text{-}1)^{th}$ row and the components in the i^{th} row on which the segments merge is assigned a new segment number. In state (5), the splitting segment is terminated at the $(i\text{-}1)^{th}$ row and each of the components emerging from this segment is assigned a new segment number.

Theorem 5.1: There are at most five states (mentioned above) in the process of assigning components to segments.

Proof: The possible components appearance in row (i-1) and row i are as follows:

1. No components in row (i-1) and row i. In this case there is no segment.
2. One or more components in row (i-1) and none in row i. In this case, all segments in row (i-1) terminate at row i which is state 3.
3. No components in row (i-1) and one or more components in row i. In this case segments start at row i which is state 1.
4. One or more components in row (i-1) and one or more components in row i. In this case all five states: starting, continuation, termination,

merging and splitting of segments can take place.

Thus there could be at most five states.

The conditions to detect the five states are stated below:

1. If $\alpha_k = 0$ then a segment starts with c_{ik}.
2. If $(\alpha_k = c_{kj} = \beta_j = 1) \wedge [\,|L(c_{kj})-L(c_{i-1j})| \leq \tau\,]$, then $S(c_{i-1j})$ continues with c_{kj}, where τ is a threshold value.
3. If $\beta_k = 0$, then $S(c_{i-1k})$ terminates.
4. If $\alpha_k > 1$, then $\forall\ cc_{kj} = 1$, $S(c_{i-1j})$ terminates. c_{ik} is assigned a new segment if not already assigned.
5. If $\beta_k > 1$, then $S(c_{i-1k})$ terminates. $\forall\ cc_{jk} = 1$, c_{ij} is assigned a new segment if not already assigned.

The complete segmentation algorithm is given below. The result of segmenting the symbol க (KA) is given in Figure 5.4.

Algorithm 5.1: Segmentation Process

```
segmentation_process procedure;
begin
 $C_1 := [c_{1q}]$,  $q = 1, \dots ,n_1$;
 $\forall\ c_{1q} \in C_1$,  $S(c_{1q}) := NS(1)$;
 for $i := 2$ to $m$
   begin (* for each row *)
     $C_i := [c_{iq}]$, $q = 1, \dots ,n_i$;
```

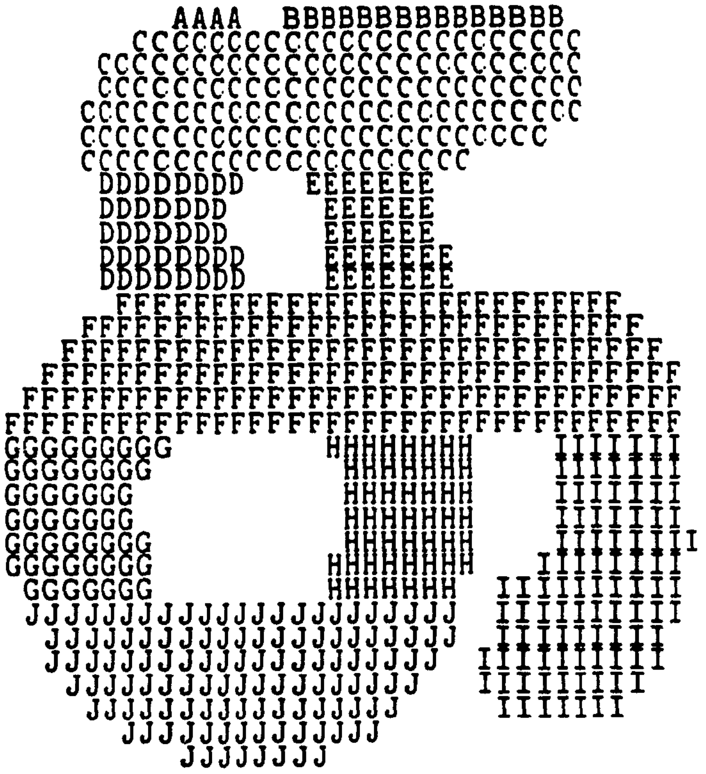


Figure 5.4 Segmented ก (KA).

$CC := [cc_{pq}],\ p=1, \ldots, n_i,\ q=1, \ldots, n_{i-1};$

$$\text{where}\ \ cc_{pq} = \begin{cases} 1, & \text{if } c_{pq} <=> c_{i-1q} \\ 0, & \text{otherwise} \end{cases},$$

$$\alpha_k = \sum_{j=1}^{n_{i-1}} cc_{kj} \quad k=1, \ldots, n_i;$$

$$\beta_k = \sum_{j=1}^{n_i} cc_{kj} \quad k=1, \ldots, n_{i-1};$$

```
for k := 1 to $n_i$
  begin (* for each component in row i *)
    if $\alpha_k$ = $cc_{kj}$ = $\beta_j$ = 1
     then
      if $|L(c_{ik}) - L(c_{i-1j})| > \tau$
        then
          begin
            terminate $S(c_{i-1j})$;
            $S(c_{ij})$ := NS();
          end
        else
            $S(c_{ik})$ := $S(c_{i-1j})$;
      fi
     elseif $\alpha_k$ = 0 then $S(c_{ik})$ := NS();
     elseif $\alpha_k$ > 1 then
      begin
        $S(c_{ik})$ := NS();
        $\forall$ $cc_{kj}$ = 1 terminate $S(c_{i-1j})$;
      end
   fi;
  end; (* for each component in row i *)
for k := 1 to $n_{i-1}$
  begin (* for each component in row i-1 *)
    if $\beta_k$ = 0
     then
       terminate $S(c_{i-1k})$
     elseif $\beta_k$ > 1 then
       begin
         terminate $S(c_{i-1k})$;
         $\forall$ $cc_{ik}$ = 1 if $S(c_{ij})$ is undefined
             then $S(c_{ij})$ := NS(); fi;
            end
         fi
        end; (* for each component in row i-1 *)
      $C_i := C_{i-1}$;
    end;(* for each row *)
end segmentation_process; (* end of algorithm *)
```

Theorem 5.2: The segmentation algorithm will work correctly iff all the above five conditions are considered.

Proof: In order to prove this theorem, we have to show that the five conditions are necessary and sufficient. We could easily see that omission of any one condition will not detect one or more of the five states and therefore, all five conditions are necessary. Since the five conditions detect all the five states, they are sufficient and no additional conditions are needed, thus, proving the theorem.

Theorem 5.3: The segmentation algorithm has a time complexity of $\bigcirc(mn)$.

Proof: The outer i loop is repeated $(m-1)$ times since there are $(m-1)$ pairs of rows out of m rows to be compared. The two k loops are repeated as many times as there are number of components in each row. In the worst case, there could be a maximum of $n/2$ components in each row. Therefore, the total number of times the loops are executed is $(m-1)$ times $(n/2 + n/2)$. Thus, the complexity of the algorithm is $\bigcirc(mn)$.

Theorem 5.4: The segmentation algorithm has a space complexity of $O(n(m+n/4))$.

Proof: The character pattern, *P* requires a size of *mn*. The *cc*-matrix, CC requires a worst case size of $(n/2)\mathrm{x}(n/2)$. The vectors α and β each requires a size of $n/2$. Thus, the total space requirement is $mn+(n/2)^2+(n/2)+(n/2)$. By ignoring the constant and linear terms of n, we get $mn + n^2/4 = \mathrm{n(m+n/4)}$.

5.4.2 Knowledge Base

The knowledge base contains the segmentation details for each symbol in a language. The following assumptions are made in describing the symbols.

1. A symbol is made up of one or more primitive segments from the set PS given below. The pattern corresponding to the segment is specified within square brackets.

   PS = {
   joint[.],
   horizontal line[-],
   vertical line [|],
   left diagonal line [\],
   right diagonal line [/],
   left curve [(],
   right curve [)],
   cup [⌣],
   cap [⌢]
   }

2. Each segment except joint has three 'connectivity points' from the set

CP given below. The abbreviation for the elements of CP are specified within brackets.

```
CP = {
    left end(L),
    right end (R),
    top end(T)
    bottom end(B),
    middle(M)
  }
```

2. The sets {L,R} and {T,B} are mutually exclusive. A joint which is a junction point for many segments has only one connectivity point. It could be represented by any member of CP. The connectivity points for the members of CP are shown in Figure 5.5(a).
3. One segment can be connected to another segment only at the connectivity point(s).
4. The segments of a symbol are numbered from top to bottom and left to right in the order of appearance as done in the segmentation process.

The symbol (KA) as represented in the knowledge base is shown in Figure 5.5(b). Note that these segments are the 'expected' segments and not the 'actual' segments (compare with Figure 5.4). It is easy to specify these segments for the symbols of a language.

Let S = $\{s_1, s_2, \ldots s_k\}$ be the segments constituting the symbol p and T = $\{t_1,$

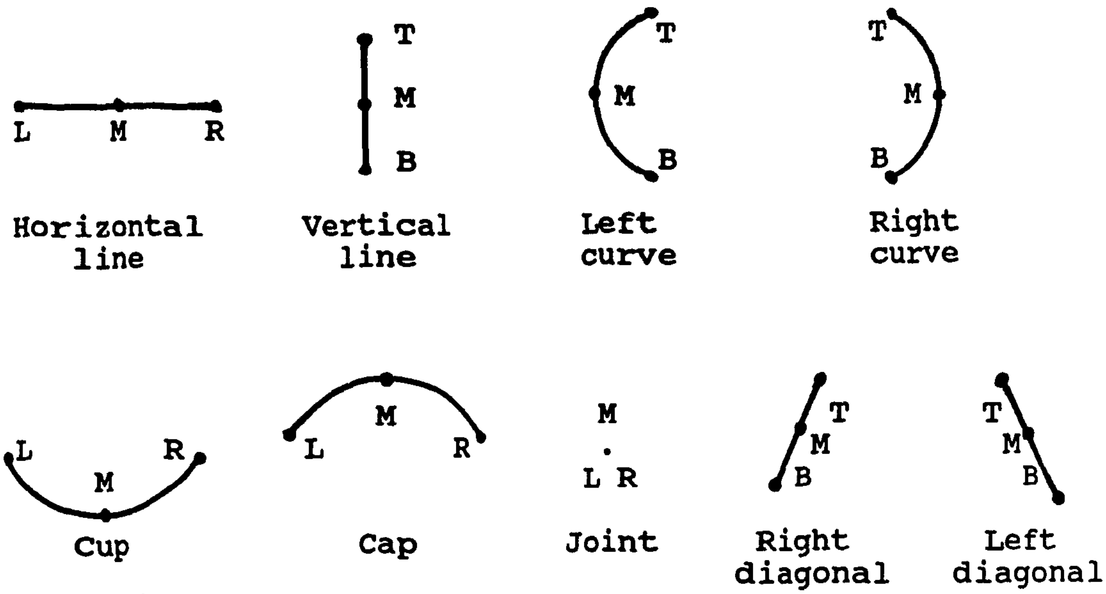


Figure 5.5a Connectivity points of segments.

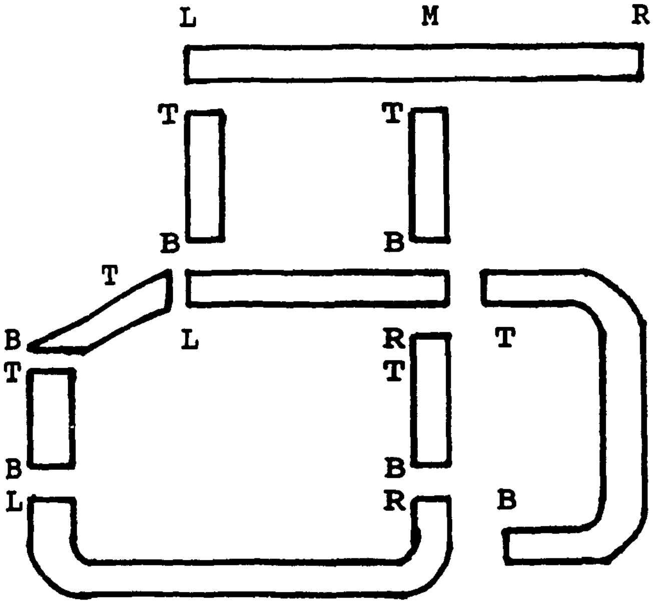


Figure 5.5b Segments of क (KA) as in the knowledge base.

t_2, ... $t_k\}$ be the set of attributes of the segments of S. That is, $t_i \in T$ is the attribute of the segment $s_i \in S$.

Each symbol in the language is represented as an attribute graph AG(N,E). The symbol p is stored in the knowledge base as (1) the *c*-matrix CAG of AG(N,E) and (2) the node attribute set $A = \{a_1, a_2, \ldots, a_k\}$ of N. The set $N = \{n_1, n_2, \ldots n_k\}$ is the set of nodes of AG. The node n_i represents the segment s_i. The type of segment s_i is represented as the element a_i of A. There exists a 1:1 mapping $f{:}S \rightarrow N$ and $g{:}T \rightarrow A$. E is the set of edges of AG: that is, the unordered pair of elements given by

$$E = \{<n_i,n_j> \mid i \neq j \wedge s_i <=> s_j,]$$

where $1 \leq i \leq \|N\|$ and $1 \leq j \leq \|N\|$.

CAG gives the connectivity of the segments of p.

$$\text{CAG} = [x_{ij}], \quad 1 \leq i \leq \|N\|, \quad 1 \leq j \leq \|N\|,$$

where

$$x_{ij} = \begin{cases} <p,q>, & \text{If } n_k <=> n_j \text{ at } p \wedge q \text{ where } p,\ q \in cp \\ 0, & \text{otherwise} \end{cases}$$

$<p,q>$ is an ordered pair. The matrix CAG and the set A for the symbol ಕ (KA) are shown in Figures 5.5(c) and 5.5(d), respectively.

	1	2	3	4	5	6	7	8	9
1		L,T	M,T						
2	T,L				B,L				
3	T,M				B,R				
4					T,L	B,T			
5		L,B	R,B	L,T			R,T	R,T	
6				T,B					B,L
7					T,R				B,R
8					T,R				
9						T,B	B,R		

Figure 5.5c Connectivity of ಕ (KA).

H	V	V	RD	H	V	V	RC	CUP
1	2	3	4	5	6	7	8	9

Figure 5.5d Attribute set of ಕ (KA).

5.4.3 Resegmentation

In this stage the segmented input symbol from the segmentation process is compared with the corresponding symbol in the knowledge base. The comparison may involve one or more of the following operations:

1. merging of two segments into one,
2. reallocation of part of the segment to another segment,
3. elimination of part of a component which is estimated to be noise due to irregular boundaries of a symbol, and
4. filling gaps larger than one pixel wide.

Comparison of symbols requires the usage of the *c*-matrix CAG, the attribute set *A* and the analysis of individual segments. The symbol ಕ (KA) after resegmentation is shown in Figure 5.6 as an example.

Compare the segments of ಕ (KA) in the knowledge base and in Figure 5.4. The segments *A* and *B* in Figure 5.4 are not the same as those in the knowledge base. The sizes of *A* and *B* are too small compared to other segments. The segment *C* is the same as in the knowledge base. Therefore, *A* and *B* are declared as part of *C* and connected. While resegmenting, the gap between *A* and *B* is filled. The segments *D* and *E* are as defined in the knowledge base. The *c*-matrix in the knowledge base indicates that the segments *E*, *H*, and *I* are connected to the right end of segment *F*. However, it is not so in Figure 5.4. Therefore, the pixels of *F* to

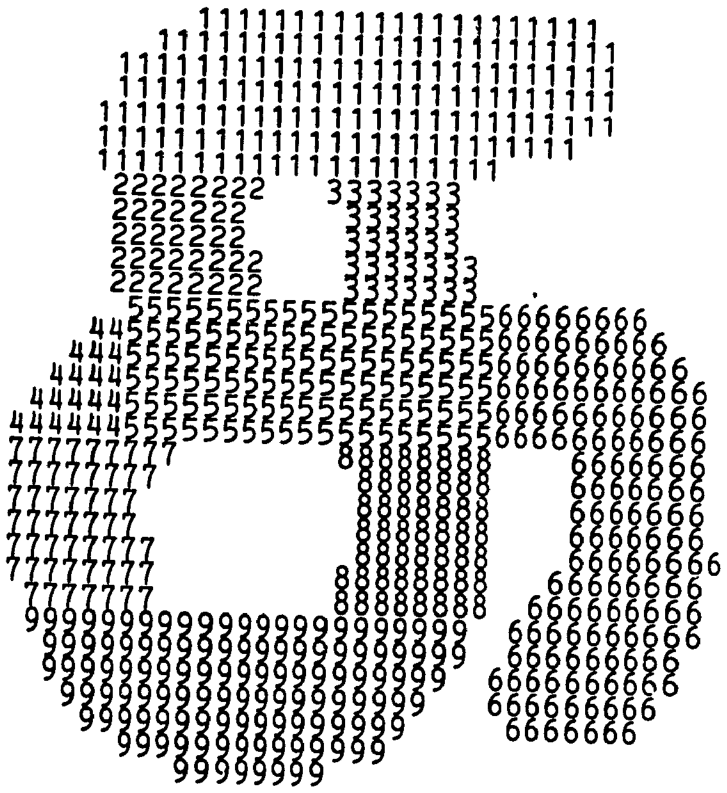


Figure 5.6 Resegmented ക (KA).

the right of *E* and *H* are reallocated to *I*. By similar analysis, the pixels of *F* to the left of segment *D* are assigned to a new right diagonal segment during resegmentation. The segments *G*, *H*, and *J* are as indicated in the knowledge base, hence not modified.

5.4.4 Thinning of Segments

A symbol after resegmentation contains only segments of type *A*. Hence, the thinning process has to thin the (1) joint, (2) line, (3) diagonal, and (4) curve only. Vertical and horizontal lines will result in the same type of operations except for the direction of thinning. Therefore, they could be treated as just lines. Similarly the four different curves could be treated as just 'curves'. During the thinning process care is taken to preserve the connectivity of segments.

A joint is thinned to a single pixel. However, the connectivity criteria may force to retain more pixels from the joint. A simple way to thin a line is to take the mid points of each component in the appropriate direction. However, this may result in an uneven line. To avoid the unevenness, we analyze the line segment and identify an even line which is approximately at the middle of the segment. To preserve the connectivity, a thinned vertical line is extrapolated, if necessary, in either direction [refer to Figure 5.7(b)]. Unwanted limbs emerge at the junction during extrapolating horizontal and vertical segments [refer to Figure 5.7(c)]. Care is taken to eliminate these limbs [Figure 5.7(d)].

```
          1 1 1 1 1 1 1 1
      1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1 1 1     1 1 1 1 1 1 1 1 1 1 1 1      1 1 1 1 1 1 1 1 1 1 1      1 1 1 1 1 1 1 1
1 1 1 1 1 1 1 1 1 1 1 1                                        1                    1
1 1 1 1 1 1 1 1 1 1 1 1                                        1                    1
1 1 1 1 1 1 1 1 1 1 1 1                                        1                    1
  22222222                        2                            2                    2
  2222222                         2                            2                    2
  2222222                         2                            2                    2
  22222222                        2                            2                    2
  22222222                        2                            2                    2

         (a)                          (b)                         (c)                (d)
```

Figure 5.7 Extrapolation and limb removal from (a) to (d).

Methods reported in the literature (Pavlidis [1977]) can also be employed to thin the curves. Curves are approximated to polygons. The result of thinning the symbol క (KA) is shown in Figure 5.8.

5.5 Analysis

5.5.1 Analysis of Symbols

The thinned symbols may or may not be optimal in size. Therefore, the symbols have to be analyzed for optimum size. The images of the symbols are either reduced or enlarged as necessary and displayed. Algorithm 5.2, listed below, is used for image reduction and enlargement. The aesthetic features and distinctness of the images have been studied for different sizes.

Algorithm 5.2: Image Reduction and Enlargement

The digitized picture is treated as binary matrix P of size *m* rows and *n* columns.

s: scale factor ($s<1$ for scaling down and $s>1$ for scaling up).

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

sp_{kl}: value (dot or blank) of the pixel at the k^{th} row and l^{th} column in the

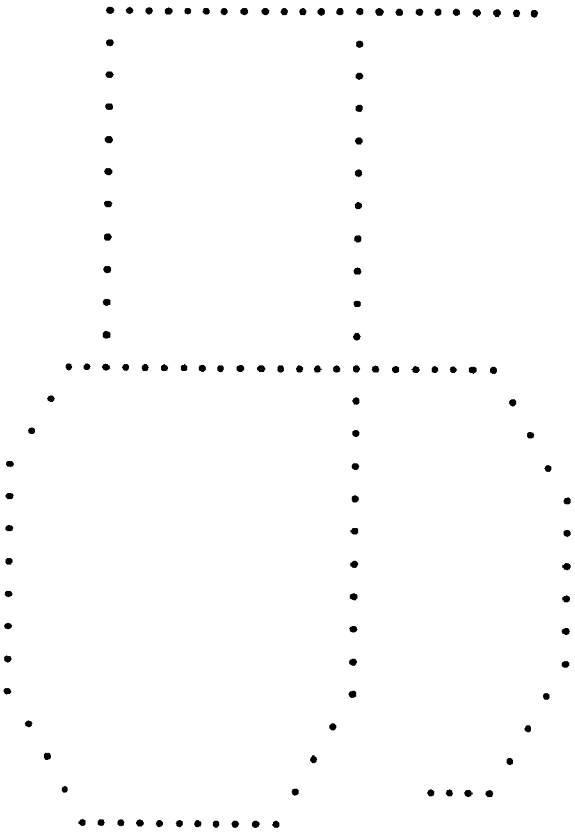


Figure 5.8 Thinned ക (KA).

scaled picture.

Then, the pixels in the scaled picture are given by

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

where $i=1 \ldots m$, $j=1 \ldots n$, $k= \lfloor i x s \rfloor$ and $l= \lfloor j.s \rfloor$.

One may also employ functions such as $\lceil i.s \rceil$ and $\lceil j.s \rceil$ instead of $\lfloor i.s \rfloor$ and $\lfloor j.s \rfloor$.

5.5.2 Algorithm 5.3: Nearly Optimum Size Computation

For each distinct symbol p_i in a language, let o_i be the corresponding nearly optimum size. The nearly optimum size, *OS*, required to code any symbol in the language is given by

$$OS = o_k \mid o_k \geq o_j, \quad 1 \leq j \leq m \text{ and } j \neq k$$

where *m* is the number of symbols in the language. The optimum dot matrix size to code any character was derived from the symbol size.

Theorem 5.5: Algorithm 5.3 has a time complexity of $O(m)$.

Proof: To start with let us assume that $OS = o_1$. Compare *OS* with the rest of the $(m-1)$ symbol sizes. If $OS < o_j$ then assign $OS = o_j$ for $2 \leq j \leq m$. Thus, the algorithm requires $(m-1)$ comparisons and, in the worst case, *m* assignments. Thus, the complexity is $O(m)$.

Theorem 5.6: Algorithm 5.3 has a space complexity of $\bigcirc(1)$.

Proof: Even though the size of input is *m*, we could get (read) one value at a time and compare with *OS* as shown below:

```
$OS$ := get($o_1$);
for $j$ := 2 to $m$
          begin
                    get($o_j$);
                    compare $OS$ and $o_j$;
          end
```

Therefore, the space requirement for the algorithm is independent of the size of *m*. Hence, its space complexity is $\bigcirc(1)$.

Experiments have been conducted using the Tamil symbols in Figure 5.1. A sample output of the analysis of the symbols is shown in Figure 5.9. Figure 5.10 gives the nearly optimum size of each symbol obtained from the experiment according to Definition 5.1. As stated earlier the results are subjective. It may vary to certain extent depending on the individual and the equipment used in the experiment. For example, a high resolution graphics terminal will display a dot matrix character of certain size much clearer than a low resoiution or a raster scan display. A 60 dots per inch resolution graphics terminal has been used in this experiment. The results reported are accurate to the judgement of the author and a couple of subjects used in the

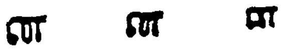


Figure 5.9 Analysis of symbols.

Symbol	Size	Symbol	Size	Symbol	Size
s1	7x8	s2	7x10	s3	7x8
s4	8x9	s5	7x7	s6	7x14
s7	10x8	s8	7x8	s9	7x6
s10	7x7	s11	7x9	s12	9x5
s13	7x9	s14	7x9	s15	11x7
s16	7x9	s17	8x7	s18	7x10
s19	7x11	s20	11x13	s21	11x11
s22	7x9	s23	7x9	s24	7x14
s25	7x7	s26	9x7	s27	9x9
s28	9x9	s29	9x9	s30	6x6
s31	9x9	s32	7x9	s33	10x10
s34	7x9	s35	10x10	s36	7x7
s37	11x15	s38	11x10	s39	10x10
s40	10x11	s41	10x10	s42	11x10
s43	11x10	s44	11x12	s45	9x12
s46	11x11	s47	7x14	s48	10x12
s49	7x6	s50	7x4	s51	4x4
s52	4x6	s53	7x3	s54	3x3
s55	7x7	s56	9x6	s57	7x9
s58	10x15	s59	7x12	s60	10x11
s61	7x4	s62	2x1	s63	2x2

Figure 5.10 Near optimum size of Tamil symbols.

experiments.

The nearly optimum dot matrix size *OS* for the Tamil symbols was computed to be 11x14. The character size is larger than the symbol size since certain characters like க் (K), பு (PU), தூ (THŪ), தி (THI), etc., are obtained by appending symbols like • , ı , ா , ி at the top, bottom and sides. From the symbol size, the character size has been computed as 15x18.

5.5.3 Analysis of Characters

Apart from the intrafeatures of symbols which appear separately, the following intersymbol (dependency) features must also be considered in determining the size. It was mentioned earlier that symbols are to be appended to generate certain characters. For example, the symbol ி is appended to the symbol க (KA) to obtain the character கி (KI); the symbol ு is appended to மு (MU) to get the character மூ (MŪ). It should be notcd that the smallest distinguishable size of the symbol ி as a separate entity need not necessarily make the character கி distinguishable when appended with the symbol க (KA). It may be necessary to consider a larger size for the symbol ி than the smallest size to make the appended character distinguishable. Thus, the nearly optimal size of symbols such as ி , ு , etc., are influenced by the symbols to which they get appended. Another dependency feature to be taken into consideration is the degree of confusion between the most similar symbols. For example, symbols such as அ and க , etc. This dependency is not taken into consideration in the size determination.

5.6 Discussion

The interactive method is best suited for the determination of the dot matrix size of the characters or symbols. The actual font design needs to be done after determining the size. In this respect the thinned and scaled characters from the automatic method could be used as the final pattern whereas the method involving 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.

The knowledge base which has the structure of the symbols enables the specialized thinning algorithm produce a better skeleton compared to existing generalized thinning algorithms which do not have any knowledge about the pattern being thinned. To thin the characters of other Indian languages, it is only necessary to create the corresponding knowledge base.

6. DETERMINATION OF SYMBOL SET

6.1 Introduction

Given a dot matrix of *m* rows and *n* columns, it is possible to design dot matrix characters of different fonts. Many 'typeface' fonts are available at present. Character styles make a great difference in legibility, particularly for Indian characters due to their complex graphemes. Also, a font design needs to be evaluated for its effectiveness and ergonomic properties before production or manufacturing. Therefore, various fonts, when made available, need to be evaluated to identify the most suitable character set. An iterative methcd of determining the most distinct set of dot matrix symbols for output systems such as printers, displays, etc., is described in this chapter. Shiau and Suen [1980] have studied various types of English fonts that are in commercial use.

Let *S* be the set of symbols of a language. There are a number of fonts, called models, in each symbol to be evaluated with respect to other symbols. The principle of font evaluation is based on the distances and information content of models. Seven different quantitative measurements are made for each model of a symbol with respect to models of other symbols or the entire corpus. These measurements were obtained from the average values of the following functions:

- identity
- hamming distance
- normalized identity
- self-information content
- entropy
- nearest neighbor distance.

Depending on the outcome of these measurements, the desirability of a model is declared. The method of evaluation is an iterative process of eliminating the least desirable models from the corpus until we obtain the most distinct set of models. The measurements and evaluation procedure are explained below. To test the method, the author has hand designed three different fonts for a set of Tamil symbols. They are shown in Figure 6.1. For illustration, the three fonts for two Tamil symbols க (KA) and ட (TA) are used. The results of the evaluation of these fonts are reported.

Let $D(ij)_{mxn}$ be the dot matrix of j^{th} model of i^{th} symbol in the corpus, S. $D(ij)$ consists of dots and blanks. Let,

$$M(ij) = [m(ij)_{pq}] \quad 1 \le p \le m, \quad 1 \le q \le n$$

$$m(ij)_{pq} = \begin{cases} 1, & \textit{if } D(ij)_{pq} \textit{ is a dot} \\ 0, & \textit{otherwise} \end{cases}$$

$$s(i) = \{M(ij) \mid 1 \le j \le c_i\}$$

$$S = \{s(i) \mid 1 \le i \le c_s\}$$

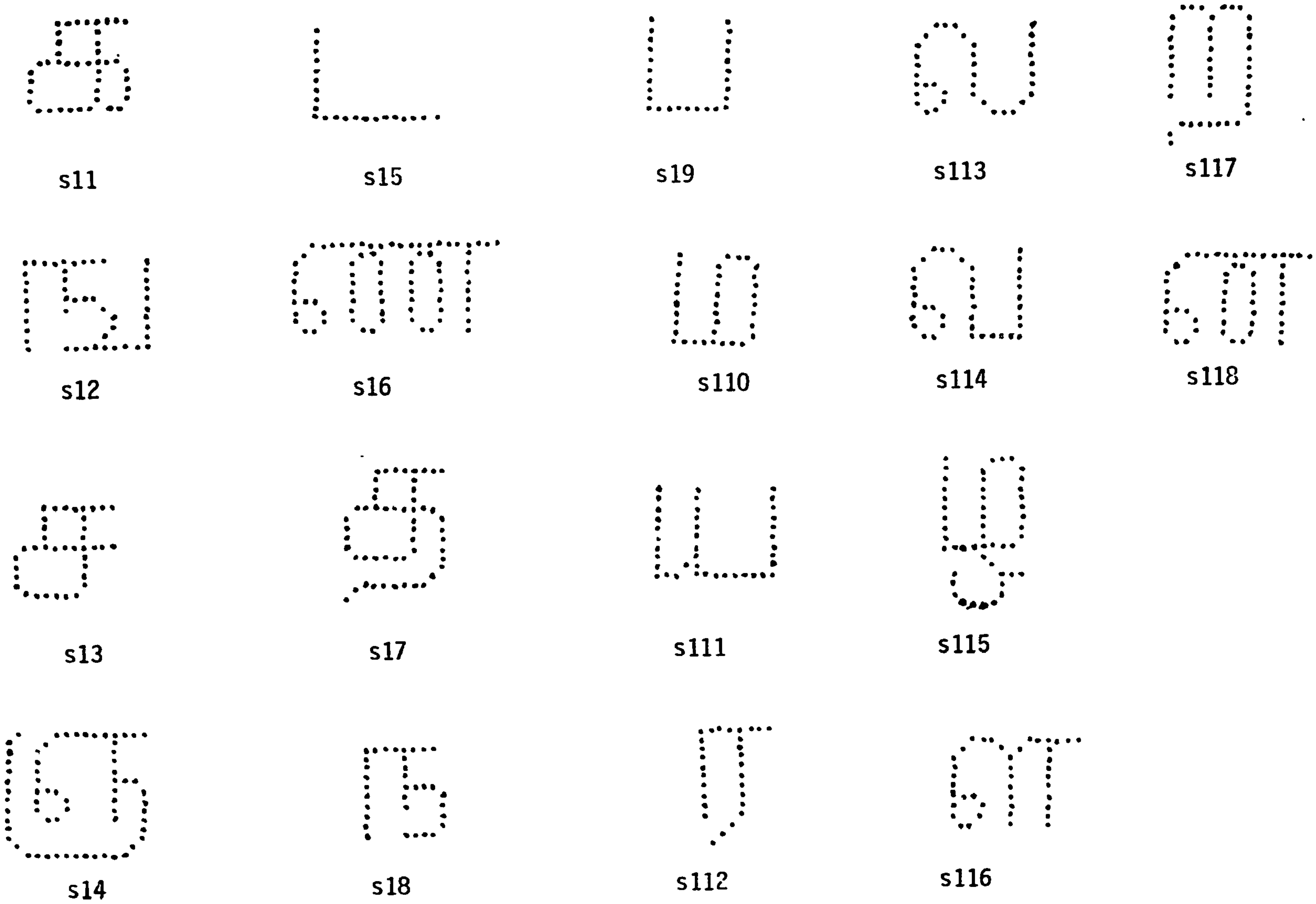


Figure 6.1 Font set 1 for Tamil symbols.

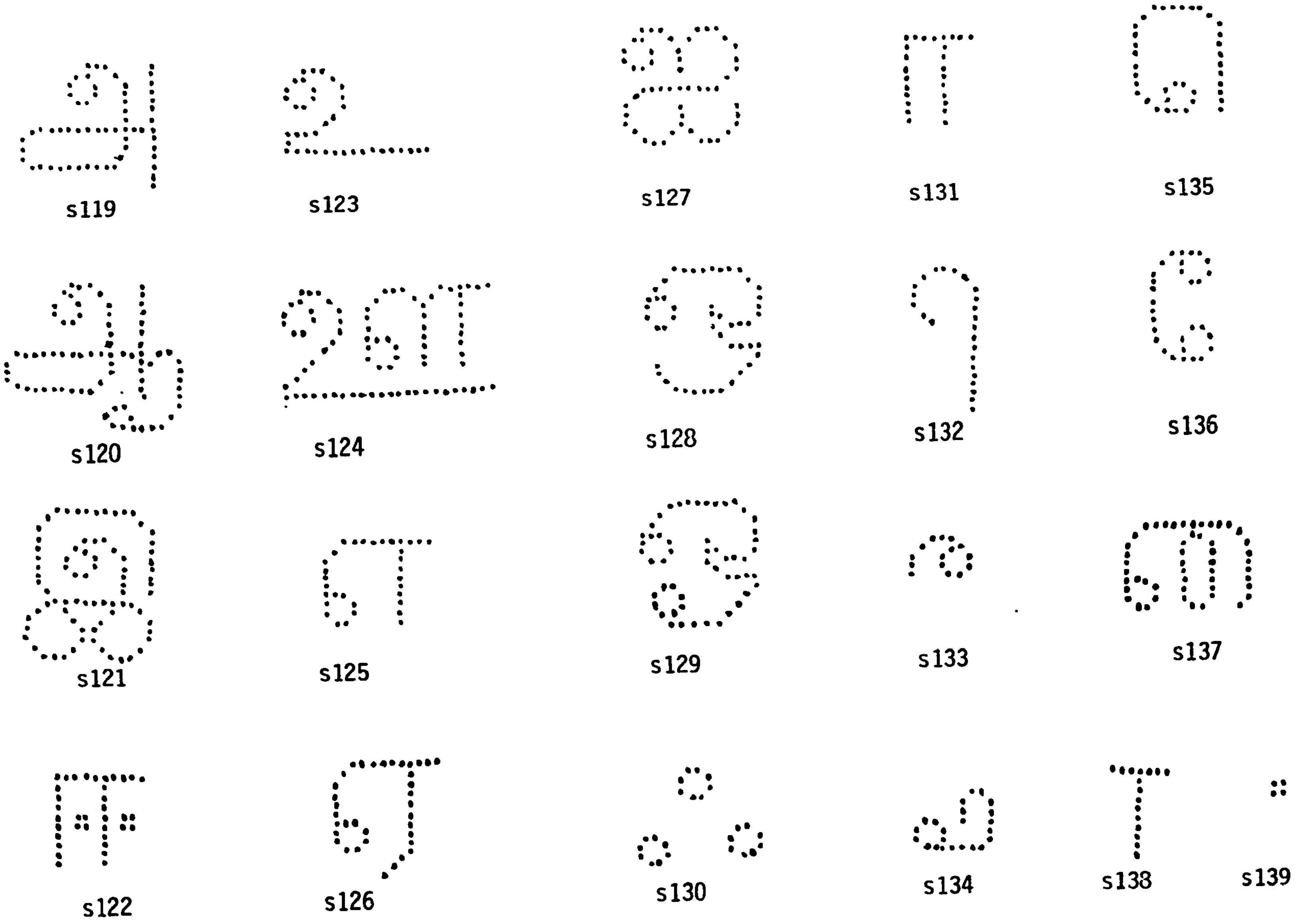


Figure 6.1 Font set 1 for Tamil symbols (contd.).

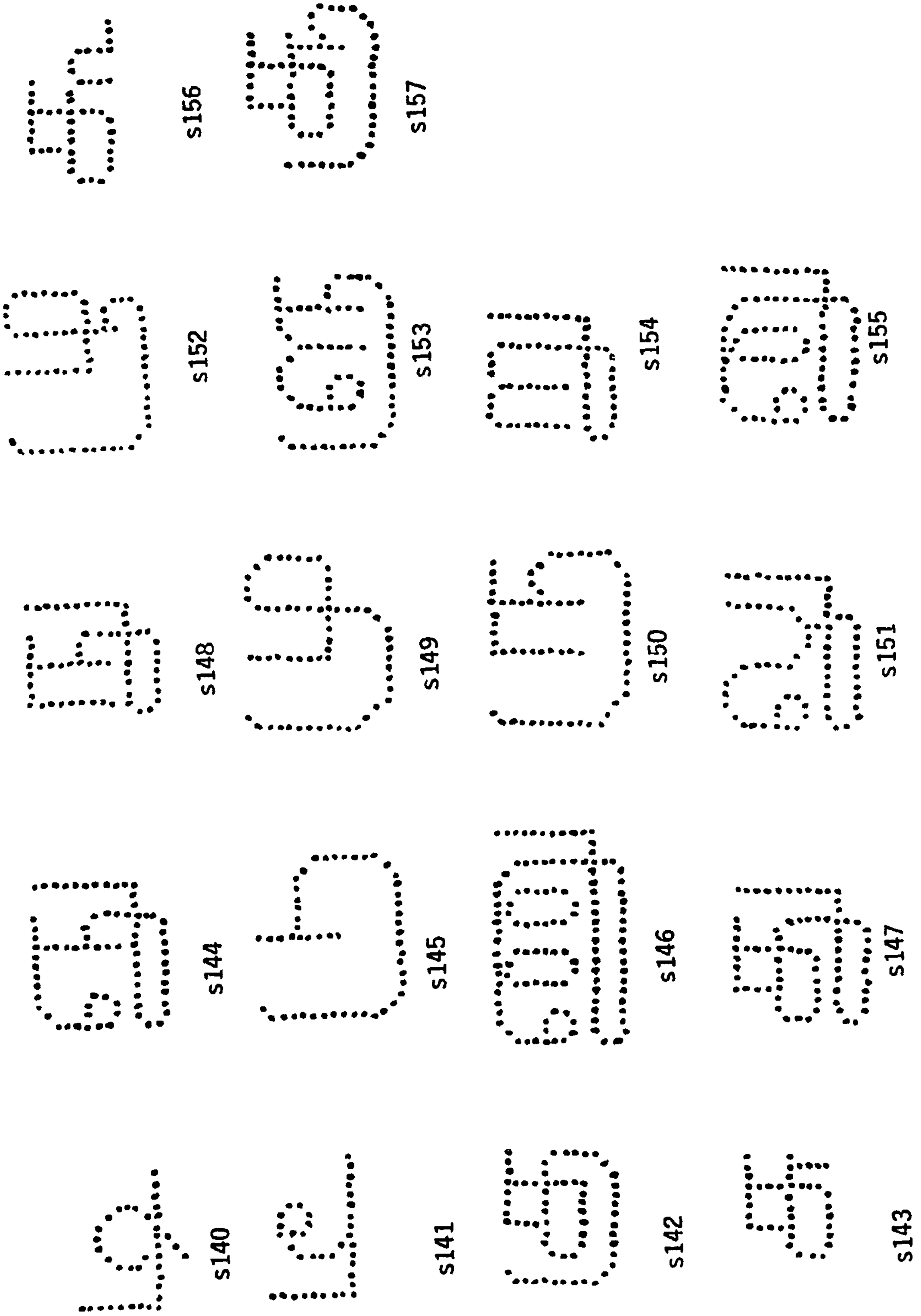


Figure 6.1 Font set 1 for Tamil symbols (contd.).

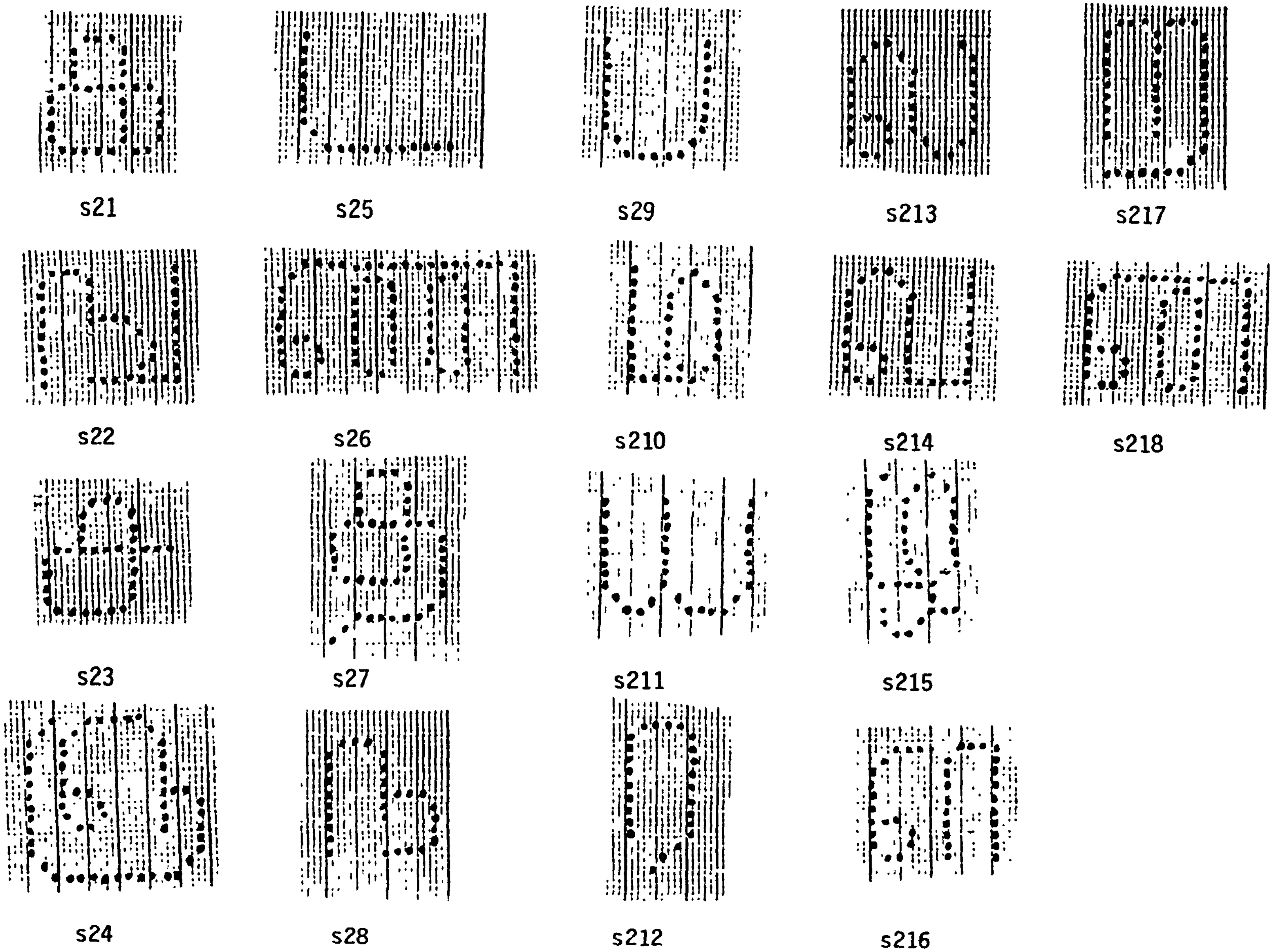


Figure 6.1 Font set 2 for Tamil symbols.

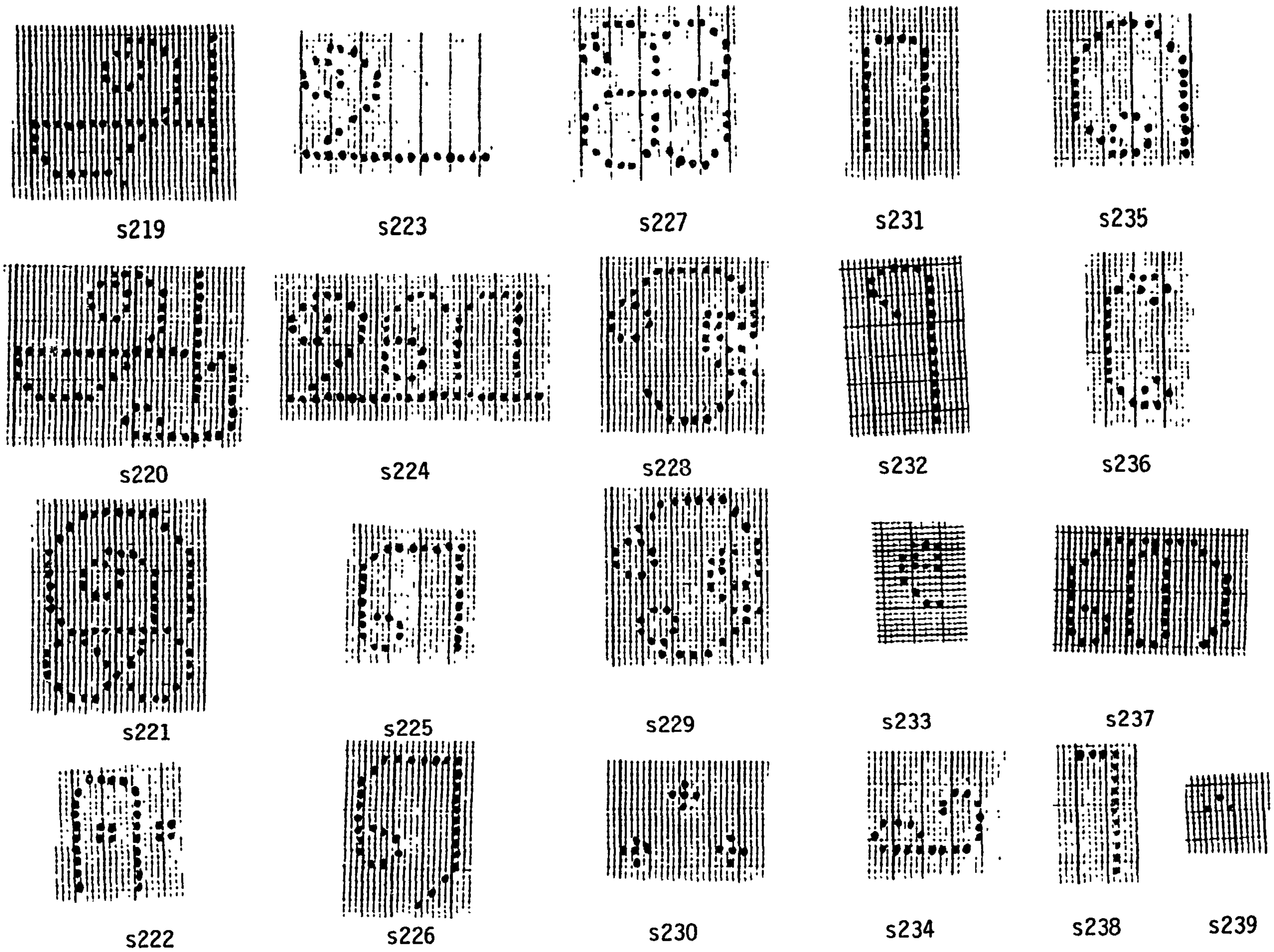


Figure 6.1 Font set 2 for Tamil symbols (contd.).

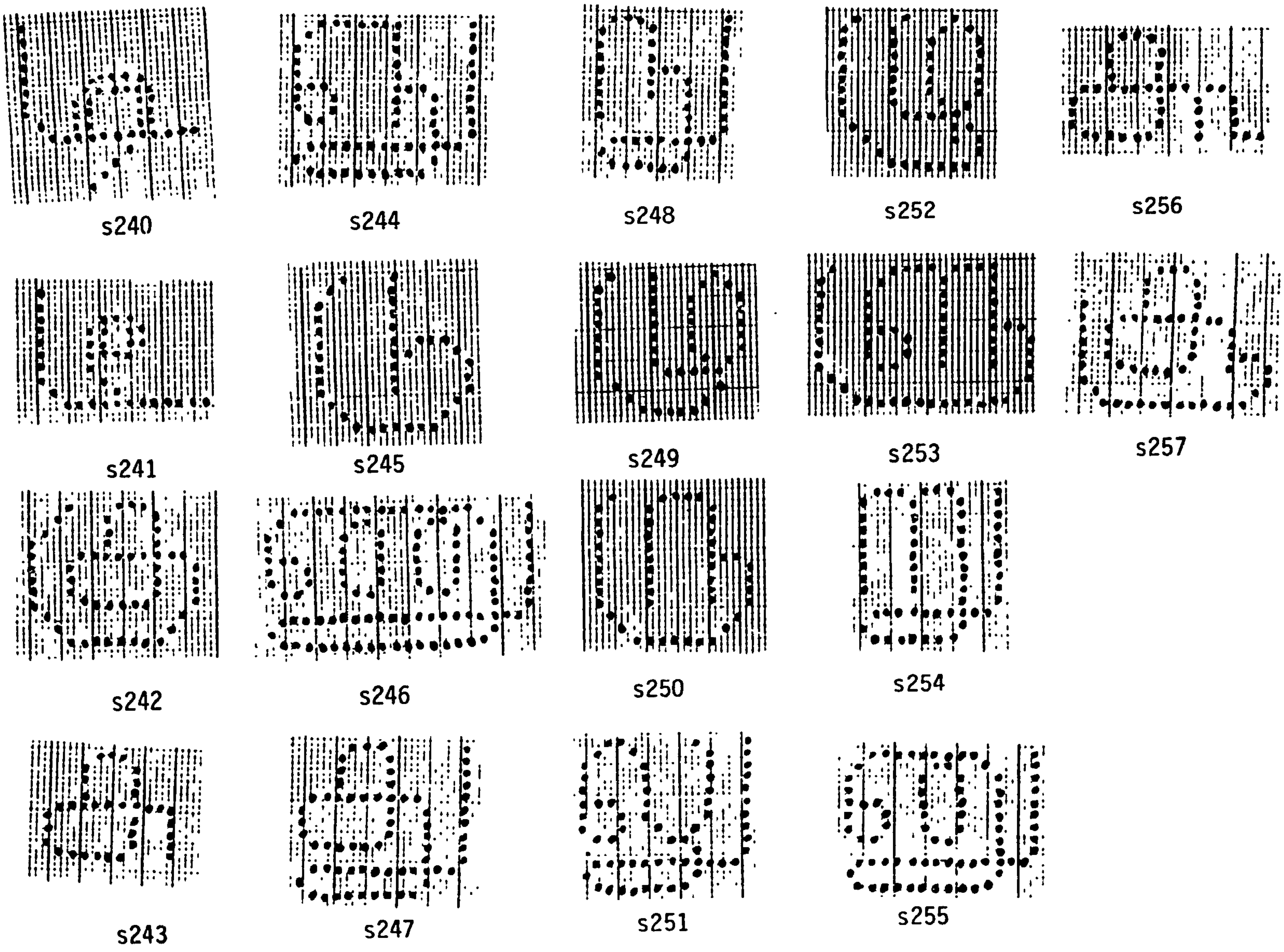


Figure 6.1 Font set 2 for Tamil symbols (contd.).

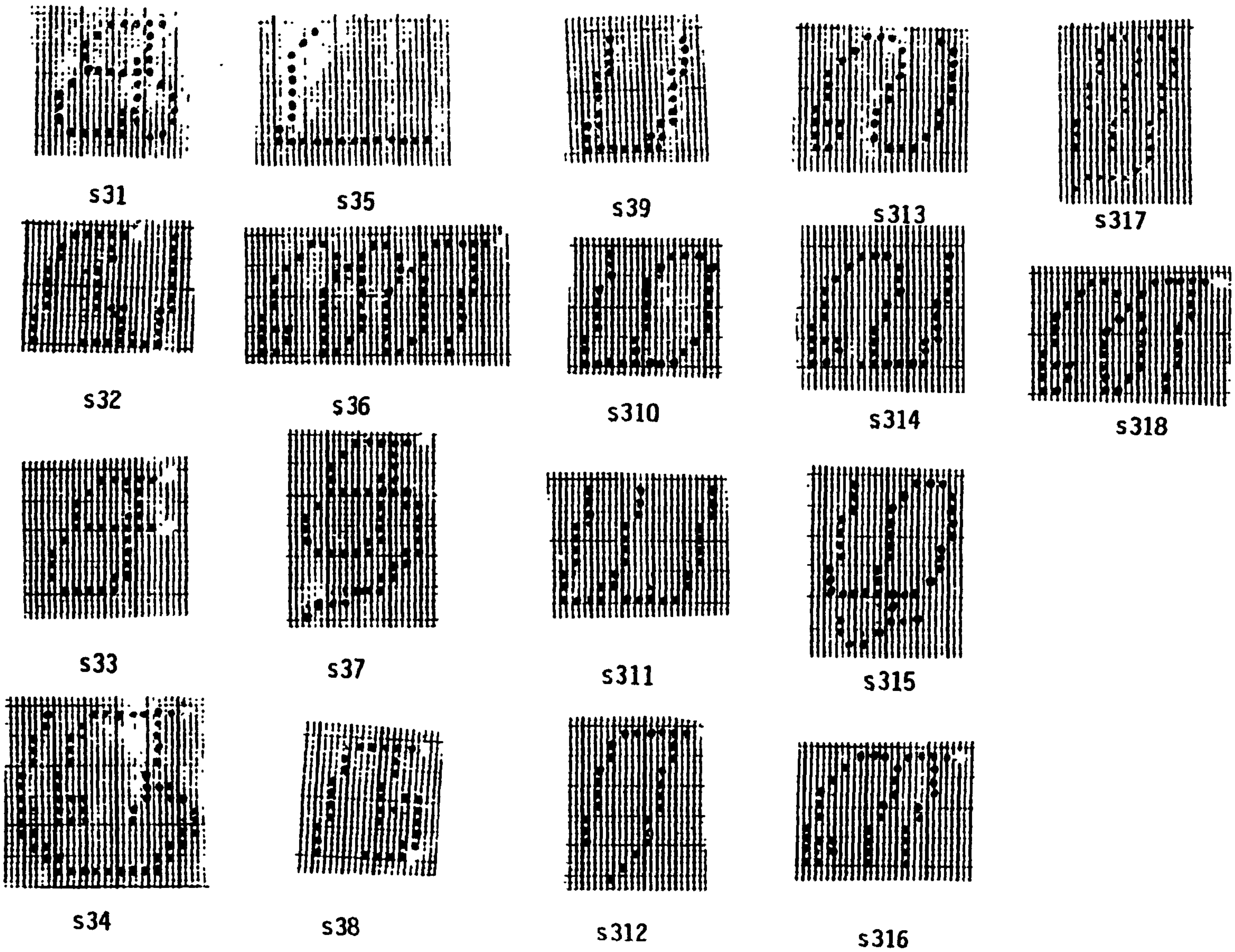


Figure 6.1 Font set 3 for Tamil symbols.

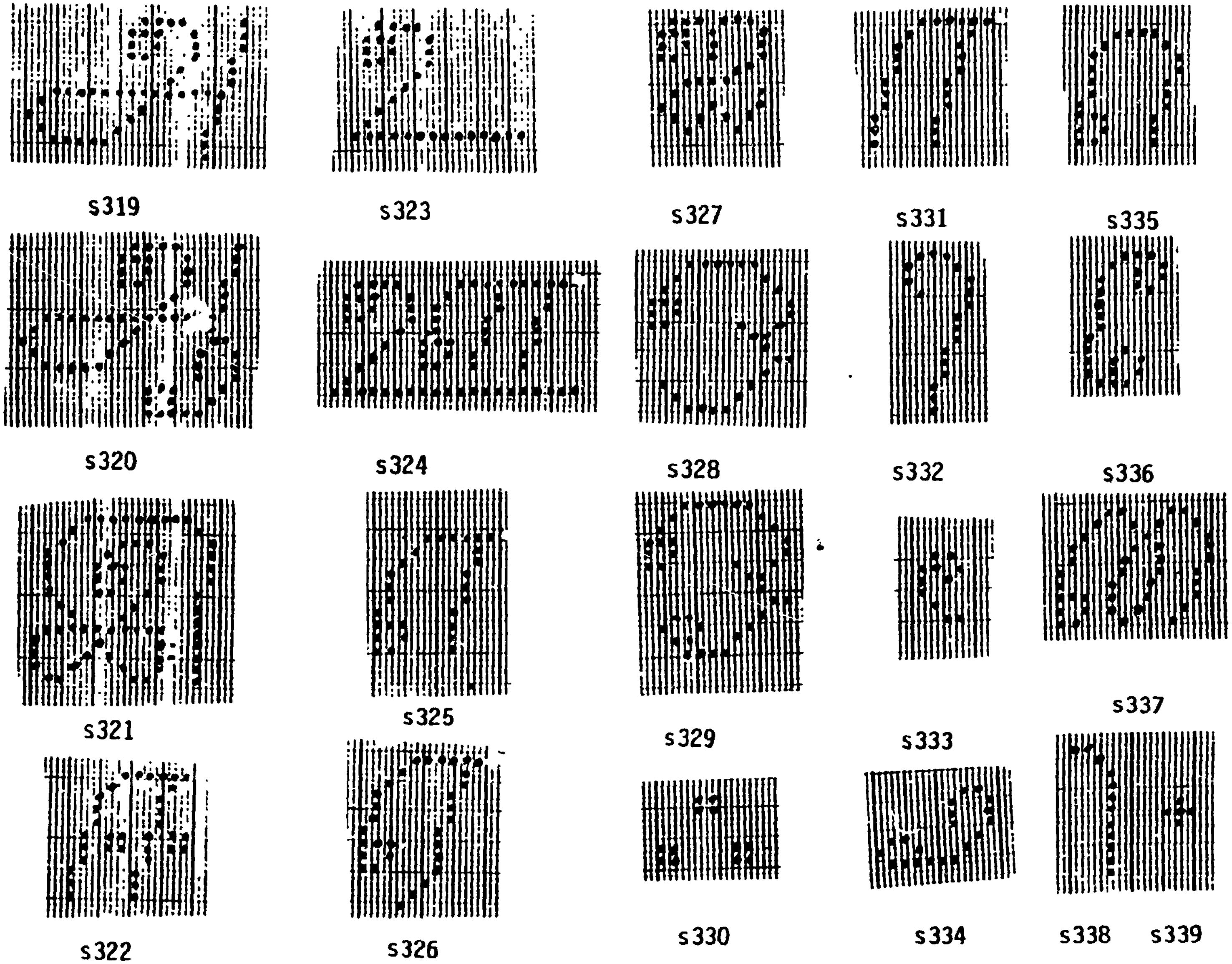
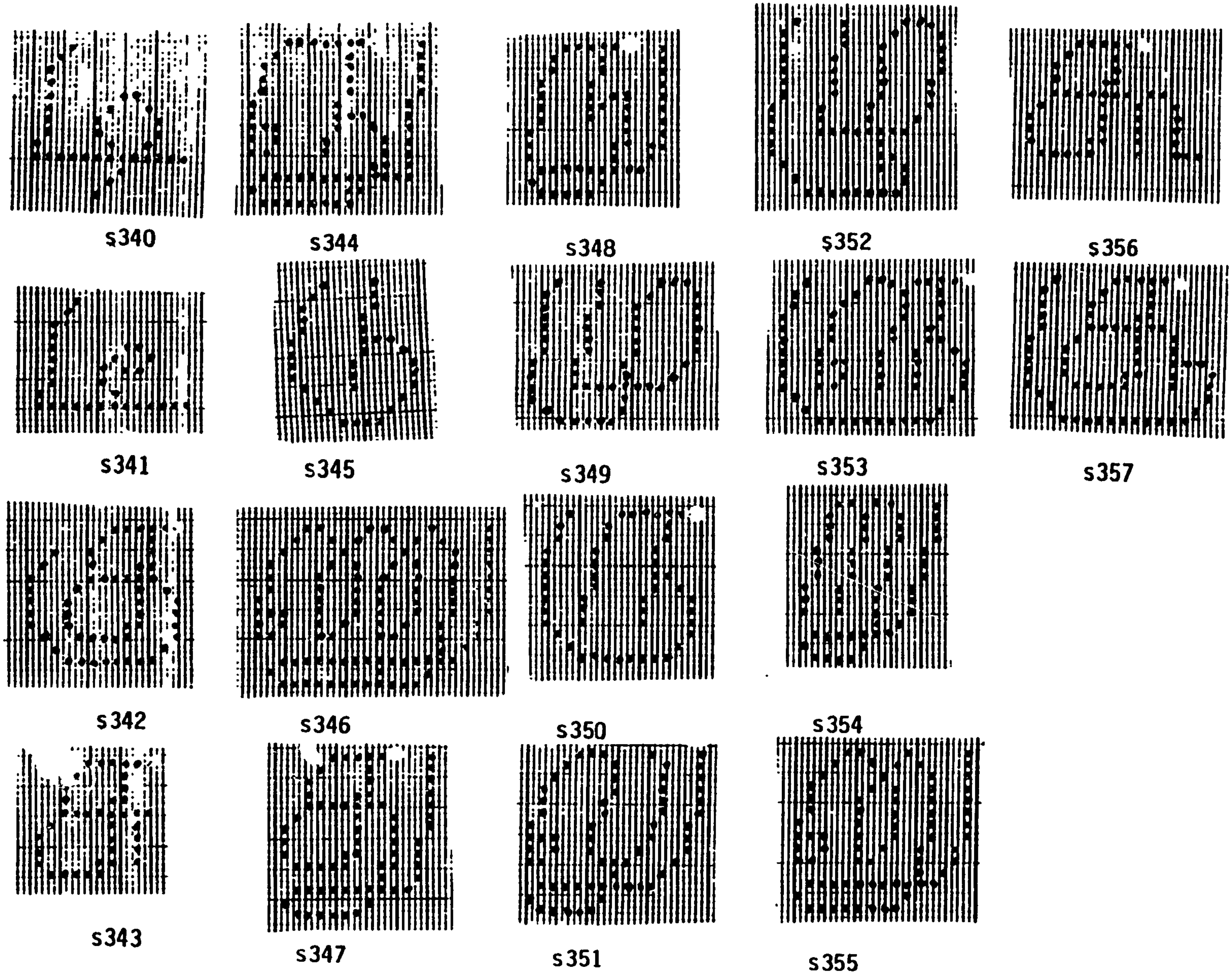


Figure 6.1 Font set 3 for Tamil symbols (contd.).



s340 s344 s348 s352 s356

s341 s345 s349 s353 s357

s342 s346 s350 s354

s343 s347 s351 s355

Figure 6.1 Font set 3 for Tamil symbols (contd.).

$$s'(i) = S\text{-}\{s(i)\}$$

$$a = \sum_{i=1}^{|S|} c_i$$

$$a' = a\text{-}c_i$$

where c_i and c_s are the cardinalities of $s(i)$ and S respectively. Also, a is the number of models in the corpus and a' is the count of all symbols in the corpus except the models of the symbol $s(i)$.

6.2 Notations and Definitions

The following notations and definitions are used in explaining the functions and the measurements.

Notation 6.1: k-function

It refers to any one of the functions listed earlier. The letter '*k*' will be replaced by a symbol representing that function. For example, α-function stands for the identity function.

Notation 6.2: A-function

The *A*-function, denoted as *Ak*(*x*), gives the average or magnified average of the *k*-function.

Notation 6.3: U-function

The *U*-function, denoted as *Uk*(*m*), gives the least desirable model $M(iu) \in s(i)$ when a given condition '*CU*' is satisfied for the measurement of the *k*-function.

Definition 6.1: MIN-function

For a set of values, *X*, the *MIN*-function, *MIN*(*X*) is defined as,

$$MIN(X) = y \;\mid\; y \in X \wedge \forall\, z \in X, \quad y \leq z$$

Definition 6.2: E-function

Given an array, *u* of models, the *E*-function is defined as follows:

$$E(u,m,n) = \begin{cases} M(ie) \\ 0, & otherwise \end{cases}$$

where $M(ie) \in s(i)$ and $M(ie)$ occurs *n* times in *u*, *m* is the size of *u* and $m/2 \leq n \leq m$.

6.3 Functions and Measurements

6.3.1 Identity Function

The identity function, $\alpha()$, is a measure of 'common dots (area)' shared between two models. It is defined as follows:

$$\alpha(ij,kr) = \sum_{p=1}^{m} \sum_{q=1}^{n} m(ij)_{pq} \wedge m(kr)_{pq}$$

The larger value of $\alpha()$ indicates a larger degree of similarity between two models.

The average identity function, $A\alpha()$ of $M(ij)$ is computed as

$$A\alpha(ij,kr) = \frac{1}{a^{i}} \sum_{k=1}^{|S|} \sum_{r=1}^{|s(k)|} \alpha(ij,kr) \qquad \textit{where } k \neq i$$

A larger value of $A\alpha(ij)$ indicates that the average common area shared by $M(ij)$ with all the models of $s(i)$ is greater and hence less distinguishable. The U-function which yields the least desirable model, $M(iu) \in s(i)$ is given by

$$U\alpha(i) = M(iu) \quad | \quad A\alpha(ij) \leq A\alpha(iu), \quad \forall\ M(ij) \in s(i) \text{ and } M(ij) \neq M(iu)$$

The first row in Table 6.1 gives the values of $A\alpha$ for the three models of the symbol (KA). The least desirable model is KA(1) since it has the highest $A\alpha$ value.

6.3.2 Hamming Distance Function

The Hamming distance function $\beta(ij,kr)$ is given by the equation

$$\beta(ij,kr) = \sum_{p=1}^{m} \sum_{q=1}^{n} m(ij)_{pq} \oplus m(kr)_{pq}$$

The Hamming distance is the complement of identity function. The smaller the value of $\beta(ij,kr)$, the lesser the dissimilarity between the models $M(ij)$ and $M(kr)$ and hence less desirable.

Table 6.1. Seven measurements for KA

Ak	KA(1)	KA(2)	KA(3)
$A\alpha$	9.33	8.33	4.33
$A\beta$	49.00	44.00	53.00
$A\gamma$	1.66	1.74	1.10
$A\zeta$	0.63	0.67	0.30
$A\psi$	282.8	229.90	252.00
$A\xi$	3.30	3.00	2.49
$A\phi$	4.42	3.72	4.40

The average Hamming distance measurement $A\beta()$ of $M(ij)$ is calculated as

$$A\beta(ij) = \frac{1}{a'} \sum_{k=1}^{|S|} \sum_{r=1}^{|s(kr)|} \beta(ij,kr) \qquad \textit{where } k \neq i$$

The least dissimilar model $M(iu) \epsilon s(i)$ is given by the U-function as

$$U\beta(i) = M(iu) \;\; | \;\; A\beta(ij) \geq A\beta(iu), \;\forall\; M(ij) \epsilon s(i) \text{ and } M(ij) \neq M(iu)$$

The second row of Table 6.1 gives the values of $A\beta$ for the three models of the symbol (KA). The least desirable model is KA(2) since it has the least dissimilarity among the three models.

6.3.3 Normalized Identity Functions

The identity function, $\alpha()$ does not take into account the various degrees of misalignment and stroke width variation of the models. Two normalized functions, γ and ζ are defined to take these variations into account. Let,

$$x(ij) = \sum_{p=1}^{m} \sum_{q=1}^{n} m(ij)_{pq}$$

$$\gamma(ij,kr) = \frac{\alpha(ij,kr)}{[x(ij) + x(kr)]/2}$$

$$= \frac{2.\alpha(ij,kr)}{x(ij) + x(kr)}$$

An '*a*' times magnified average function $A\gamma\,(ij)$ is computed as

$$A\gamma(ij) = \frac{1}{a}\sum_{k=1}^{|S|}\sum_{r=1}^{|s(k)|}\gamma(ij,kr)\,a$$

$$= \sum_{k=1}^{|S|}\sum_{r=1}^{|s(k)|}\gamma(ij,kr)$$

$$= 2\sum_{k=1}^{|S|}\sum_{r=1}^{|s(k)|}\frac{\alpha(ij,kr)}{x(ij)+x(kr)}$$

The function, $\zeta\,(ij,kr)$ and the corresponding magnified average function $A\zeta\,(ij)$ are as follows:

$$\zeta(ij,kr) = \left(\frac{\alpha(ij,kr)}{\sqrt{x(ij).x(kr)}}\right)^2$$

$$= \frac{\alpha(ij,kr)^2}{x(ij).x(kr)}$$

$$A\zeta(ij) = \frac{1}{a}\sum_{k=1}^{|S|}\sum_{r=1}^{|s(k)|}\zeta(ij,kr)\,a$$

$$= \sum_{k=1}^{|S|}\sum_{r=1}^{|s(k)|}\zeta(ij,kr)$$

$$= \sum_{k=1}^{|S|}\sum_{r=1}^{|s(k)|}\frac{\alpha(ij,kr)^2}{x(ij).x(kr)}$$

The larger value of $A\gamma\,(ij)$ and $A\zeta\,(ij)$, indicates that the normalized common dots shared by the model $M(ij)\in s(i)$ and all the models in the corpus are larger and hence less desirable.

The *U*-functions for the $A\gamma$ and $A\zeta$ measurements are given by the following equations:

$$U\gamma(i) = M(iu) \mid A\gamma(ij) \leq A\gamma(iu)$$

$$U\zeta(i) = M(iu) \mid A\zeta(ij) \leq A\zeta(iu)$$

$$\forall\ M(ij) \epsilon\, s(i) \text{ and } M(ij) \neq M(iu) \text{ where } M(iu) \epsilon\, s(i).$$

The third and fourth rows of Table 6.1 give the $A\gamma$ and $A\zeta$ values for the three models of $\mathfrak{S}$ (KA). The least desirable model is KA(2) since it has the largest $A\gamma$ and $A\zeta$ (normalized common area) among the three models.

6.3.4 Self-information and Entropy

The self-information of an element d_{pq} of the dot matrix, *D* is given by

$$\psi(d_{pq}) = -\log_2 P_{pq}$$

where P_{pq} is the probability of the pq^{th} element being a 'dot' for all the models under consideration. The probability is computed as follows:

$$P(pq) = \frac{y(pq)}{b}$$

where

$$y(pq) = \sum_{i=1}^{|S|} \sum_{j=1}^{|s(k)|} m(ij)_{pq}$$

and

$$b = \sum_{p=1}^{m} \sum_{q=1}^{n} y(pq)$$

The self-information, $\psi(ij)$ of a model $M(ij) \epsilon s(i)$ is given by

$$\psi(ij) = \sum_{p=1}^{m} \sum_{q=1}^{n} \psi(d_{pq})\, m(ij)_{pq}$$

The entropy is the mean of the self-information content of a model. For the model $M(ij)$, it is defined as

$$\xi(ij) = \sum_{p=1}^{m} \sum_{q=1}^{n} P_{pq}\, \psi(d_{pq})\, m(ij)_{pq}$$

Since P_{pq} may be zero, $\xi(ij)]$ could be indeterminate and so when $P_{pq} = 0, \xi(ij) = 0$. The self-information of an element increases as its uncertainty grows. Therefore, the entropy may be regarded as a measure of uncertainty. The entropy vanishes iff there is complete certainty. Thus, the smaller the value of $\psi()$ or $\xi()$, the better the model is. The *A*-functions for $\psi()$ and $\xi()$ are given below:

$$A\psi(ij) = \psi(ij)$$

$$A\xi(ij) = \xi(ij)$$

The corresponding *U*-functions are as follows:

$$U\psi(i) = M(iu) \quad | \quad A\psi(ij) \leq A\psi(iu)$$

$$U\xi(i) = M(iu) \quad | \quad A\xi(ij) \leq A\xi(iu)$$

$\forall\ M(ij) \epsilon\, s(i)$ and $M(ij) \neq M(iu)$.

The least desirable model among the three $\mathcal{S}$ (KA), from Table 6.1, is KA(1), since it has the highest $A\psi$ () and $A\xi$ () values.

6.3.5 Nearest-neighbor Distance

The 'nearest cell distance', $\sigma(ij,kr)_{pq}$ of cell d_{pq} corresponding to $M(ij)$ with respect to the model $M(kr)$ is defined as follows:

$$\sigma(ij,kr)_{pq} = \begin{cases} MIN(||(s-p)^2+(t-p)^2||) \\ 0, \quad if\ m(ij)_{pq}=0 \end{cases}$$

such that $m(kr)_{st} \neq 0$ and $m(ij)_{pq} \neq 0$, for $1 \leq s \leq m$, $1 \leq t \leq n$.

A larger value of σ (ij,kr) indicates a larger 'distance' between cell $m(ij)_{pq}$ and the 'nearest cell' occupied by the model $M(kr)$. For any pair of models $M(ij)$ and $M(kr)$, nearest-neighbor distance, $\phi(ij,kr)$ is defined as follows:

$$\phi(ij,kr) = \frac{1}{X(ij)} \sum_{p=1}^{m} \sum_{q=1}^{n} \sqrt{\sigma(ij,kr)_{pq}} + \frac{1}{X(kr)} \sum_{p=1}^{m} \sum_{q=1}^{n} \sqrt{\sigma(kr,ij)_{pq}}$$

In general, it is true that $\phi\,(ij,kr) \neq \phi\,(kr,ij)$. A larger value of $\phi\,(ij,kr)$ indicates that the pair $M(ij)$ and $M(kr)$ is more distinct. The A-function and U-function are:

$$U\phi\,(i) = M(iu) \ \mid\ A\phi\,(ij) \geq A\phi\,(iu)$$

$$\forall\ M(ij) \epsilon\, s(i) \text{ and } M(ij) \neq M(iu)$$

$$A\phi(ij) = \frac{1}{a'} \sum_{k=1}^{|S|} \sum_{r=1}^{|s(k)|} \phi(ij, kr) \qquad \textit{where } k \neq i$$

Thus, the model having the smallest value of $A\phi\,()$ is the least desirable. For instance, the values in the last row of Table 6.1 indicate that KA(2) is the least desirable model among க (KA).

6.4 Elimination Algorithm

The elimination process is iterative. In each stage of the iteration, all seven measurements were computed and compared among different models of the same symbol. Those models rated as "undesirable" were successively eliminated by 100%, 87.5%, 75% and 62.5% elimination rules (Shiau and Suen [1980]). The elimination algorithm is given below. When the elimination algorithm as applied to the six models of க (KA) and ட (TA), the models KA(2) and TA(1) were eliminated in one iteration, KA(1) in the next iteration and TA(3) in the final iteration. The most distinct models obtained after successive elimination are KA(3) and TA(2) (refer Table 6.2).

Algorithm 6.1: Elimination Algorithm

```
procedure elimination_algorithm;
begin
  no_of_measurements := 7;
  m, n := no_of_measurements;
```

Table 6.2. Results of Elimination of KA and TA

****

Iteration #1 100.0 percent Elimination Rule

Eliminated Models: none

****

Iteration #2 87.5 percent Elimination Rule

The Measurements are as Follows:

Ak	KA(1)	KA(2)	KA(3)	TA(1)	TA(2)	TA(3)
$A\alpha$	9.33	8.33	4.33	6.33	7.00	8.67
$A\beta$	49.00	44.00	53.00	48.00	44.67	53.33
$A\gamma$	1.66	1.74	1.10	2.23	2.23	2.05
$A\zeta$	0.63	0.67	0.30	1.48	1.43	1.11
$A\psi$	282.86	229.87	251.99	128.14	114.58	201.67
$A\xi$	3.30	3.00	2.49	2.34	2.23	2.84
$A\phi$	4.42	3.72	4.40	4.38	4.51	3.71

****

Eliminated Models: KA(2) TA(1)

****

Table 6.2. Results of Elimination of KA and TA (contd.)

Iteration #3 87.5 percent Elimination Rule

The Measurements are as Follows:

Ak	KA(1)	KA(3)	TA(2)	TA(3)
$A\alpha$	10.00	4.50	6.50	8.00
$A\beta$	49.00	54.00	47.00	56.00
$A\gamma$	0.88	0.59	1.04	1.05
$A\zeta$	0.27	0.13	0.52	0.51
$A\psi$	276.81	243.47	117.38	198.91
$A\xi$	3.43	2.72	2.10	2.85
$A\phi$	7.25	7.27	7.68	6.41

****

Eliminated Models: KA(1)

****

Table 6.2. Results of Elimination of KA and TA (contd.)

Iteration #4 87.5 percent Elimination Rule

Eliminated Models: none

****

Iteration #5 75.0 percent Elimination Rule

The Measurements are as Follows:

Ak	KA(3)	TA(2)	TA(3)
$A\alpha$	4.50	4.00	5.00
$A\beta$	54.00	49.00	59.00
$A\gamma$	0.29	0.76	0.76
$A\zeta$	0.04	0.42	0.42
$A\psi$	231.84	110.76	187.47
$A\xi$	3.12	2.44	3.36
$A\phi$	7.51	12.46	9.73

****

Eliminated Models: TA(3)

****

Optimal Models are: KA(3) TA(2)

****

```
  corpus := $\{M(ij) \mid M(ij) \in s(i),\ 1 \le j \le |s(i)|,\ 1 \le i \le |S|\}$;
  If m mod 2 = 0
    then
      limit := m div 2 - 1
    else
      limit := m div 2;
  for 0 to limit do
    begin
      If $\forall\, s(i) \in S,\ |s(i)| = 1$
        then
          exit;
      e_list := $\{\emptyset\}$;
      eliminate_models;
      n := n-1
    end
end elimination_ algorithm;

procedure eliminate_models;
begin
  for ever do
    begin
      $\forall\, s(i) \in S$ and $|s(i)| > 1$
        begin
          for k := 1 to no_of_measurements do
            begin
              $\forall\, M(ij) \in s(i)$ do
                begin
                  compute k(ij,rs);
                  compute Ak(ij);
                end;
             u(k) := Uk(i);
            end;
          If $E(u,n,m) \neq 0$
            then
              e_list := e_list $\cup$ $\{E(u,n,m)\}$;
        end;
      If e_list = $\{\emptyset\}$
        then
          exit
        else
          corpus := corpus - e_list;
    end;
end eliminate_models;
```

Theorem 6.1: The elimination algorithm has a time complexity of $\bigcirc(mn(k+1)(2a\text{-}k))$ where k is the number of distinct symbols in the corpus.

Proof: In the worst case, let us assume, the algorithm eliminates one model at a time. To compute a k-function, the summation operation is executed mn times. At each iteration, the summation for Ak-function is executed as many times as there are number of models in the corpus. Since at each iteration one model is eliminated, it will reduce by one for every iteration starting from a until k. There are seven Ak-functions to compute at each iteration. Therefore, the number of times the operations in a k-function will be executed is:

$$= 7mn\{a+(a\text{-}1)+(a\text{-}2)\ \dots\ (a\text{-}k)\}$$

$$= 7mn\{a(k+1) - \frac{k(k+1)}{2}\}$$

$$= \frac{7}{2}mn(k+1)(2a-k)$$

Ignoring the constant terms gives a time complexity of $\bigcirc(mn(k+1)(2a\text{-}k))$.

Theorem 6.2: The elimination algorithm has a space complexity of $\bigcirc(amn)$.

Proof: The space required to store a model is of the order of mn. There are a models in the corpus. Therefore, the total space required for *a* model is of the order of *amn*. Also the space required to store the *Ak* and *Uk* functions is of the order of 7.*a* for each. This space is considerably smaller than the space required for the model and hence, let us ignore it. Therefore, the space complexity is $\bigcirc(amn)$.

The symbols in Figure 6.1 have been evaluated using the above method. Figure 6.2 shows the most distinct set of dot matrix symbols produced by the elimination algorithm.

6.5 Conclusion

Dot matrix character styles need to be evaluated for its effectiveness, legibility and ergonomic properties. In this chapter an iterative method of determining the most distinct character set, among various dot matrix character types, is described. The method is based on the distances and information content of the dot matrix characters. The examination of the seven quantitative measurements revealed that the 'desirability' of models indicated by any two measurements was not always identical. The method described is not limited to Tamil characters. It could be used to evaluate dot matrix characters of any language.

s31	s25	s19	s113	s117	s32	s16
s110	s114	s118	s33	s37	s211	s115
s14	s38	s312	s116	s119	s123	s127
s331	s235	s120	s124	s128	s132	s136
s121	s125	s129	s133	s337	s322	s126
s130	s134	s138	s339	s240	s144	s348
s152	s356	s241	s145	s149	s153	s357
s342	s146	s350	s154	s343	s347	s151
s155						

Figure 6.2 Distinct set of Tamil symbols.

7. MULTILINGUAL KEYBOARD DESIGN AND CHARACTER GENERATION

7.1 Multilingual Keyboard Design

For most applications data entry is, and will be for the foreseeable future, through a keyboard. This is the most successful, reliable and cost effective way of entering information at this time. However, existing mechanical typewriters, peculiar to each of the Indian languages, cannot be converted into man-computer communication terminals which meet our design goals. The design of a multilingual keyboard suitable for entering text in one or more languages is described.

Let,

K : Number of key-positions in a keyboard

N_i : Number of characters in language *i*

T : Total number of languages to be supported

Ψ : Number of language independent characters such as % , * , etc.

During keyboard design, we may encounter the following cases:

Case 1: $(\Psi + N_i) \leq K$ for any *i*, $1 \leq i \leq T$

Case 2: $(\Psi + N_i) > K$ for any *i*, $1 \leq i \leq T$

Case 3: $(\Psi + M) \leq K$ where $M = N_1 + \ldots + N_j$

In Case 1, a sufficient number of key-positions are available on the keyboard so that one character in a language can be assigned to one key-position. Such is the case with the existing English keyboards.

However, in Case 2, a language has more characters than key-positions. In this case, it is not possible to assign one character to one key-position. Such is the case with Indian languages, Chinese and many other languages of the world. In order to design a keyboard to support these languages it is necessary to use the principle of *symbolization* described in Chapter 1.

In Case 3, the union of the characters of two or more languages are assigned to the K key-positions. This is possible if the set of languages have a small number of characters or the union of the symbols of the set of languages are less than or equal to K.

7.1.1 Symbolization

In symbolization, we analyze the graphemes of the characters set as well as the linguistic characteristics of the language. By *graphemes*, we mean the shape of the characters. Symbolization gives us a set of *symbols* of the language under analysis.

Let,

c_{ij} be the j^{th} character in the i^{th} language

s_i be the set of symbols obtained through symbolization for the i^{th} language

An important characteristic of the symbols of a language is that there exists a mapping

$$f_i : c_{ij} \xrightarrow{\mathit{construct}_i} s_i \quad \text{for } 1 \leq j \leq N_i \text{, and } 1 \leq i T$$

Let, $\{s_{i1}, s_{i2}, \ldots, s_{im}\}$ be the set of symbols constituting c_{ij}. Given the sequence of symbols $\{s_{i1}, s_{i2}, \ldots, s_{im}\}$, the *construct*$_i$ will identify it as c_{ij} as well as generate it for display or printing.

The following are the constraints on the *symbolization*:

1. $(\Psi + s_i) \leq K$
2. $c_{ij} \xrightarrow{\mathit{construct}_i} s_i$ should exist for $1 \leq j \leq N_i$
3. s_i should be optimal in terms of computing time, storage requirement and complexity.

Definition 7.1: g-symbols

If the members of s_i correspond to (partial) graphemes of the characters of a language, then they are called *g*-symbols.

Definition 7.2: p-symbols

If the members of s_i are based on the phonetic characteristics of a language, then they are called *p*-symbols.

g-symbols are used in the existing keyboards. The characters of Tamil and Malayalam languages were analyzed to obtain the *g*-symbols. Figures 7.1 and 7.2 give the sets of *g*-symbols devised for the Tamil and Malayalam characters respectively.

The analysis of the characters of ten different Indian languages has resulted in a set of new symbols called *p*-symbols, based on the phonetic characteristics of Indian languages. Figures 7.3a and 7.3b show the *p*-symbols of Tamil and Malayalam characters respectively. Only the transliteration into Roman script (according to the Library of Congress Cataloging Bulletin) is shown in the figures for readability. Notice that the number of key-positions required to design the *p*-symbol keyboard for Tamil or Malayalam is less than the 52 *g*-symbol key-positions needed for the English language. Also observe that there are a number of *p*-symbols common between Tamil and Malayalam.

It has also been found that the cardinality of the union of the *p*-symbols of the

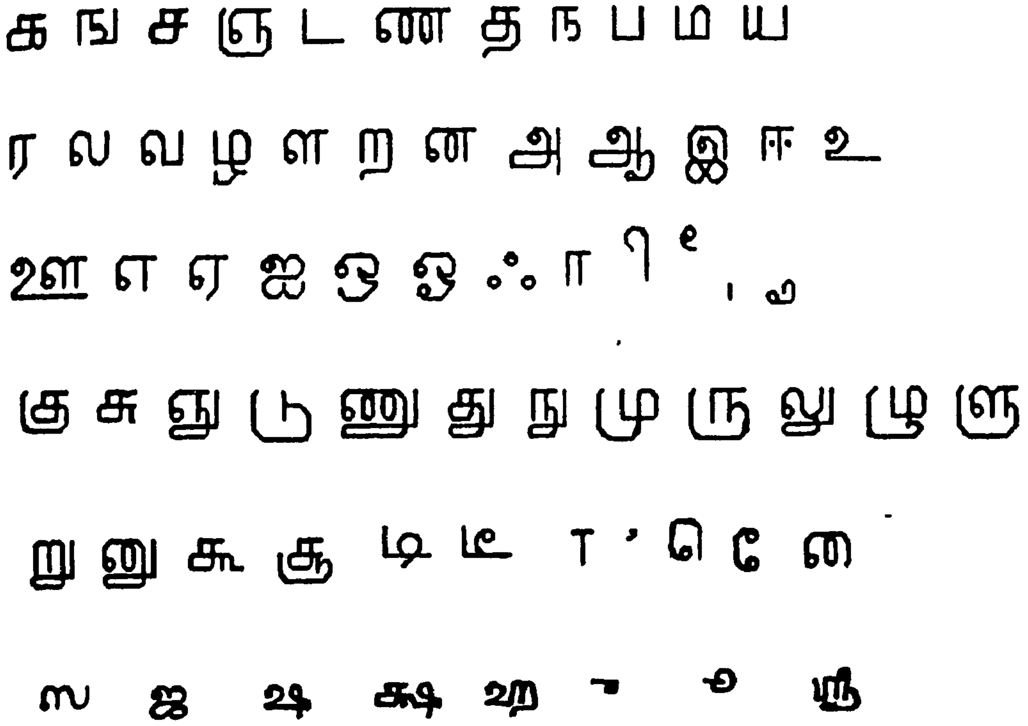


Figure 7.1 Set of *g*-symbols for Tamil characters.

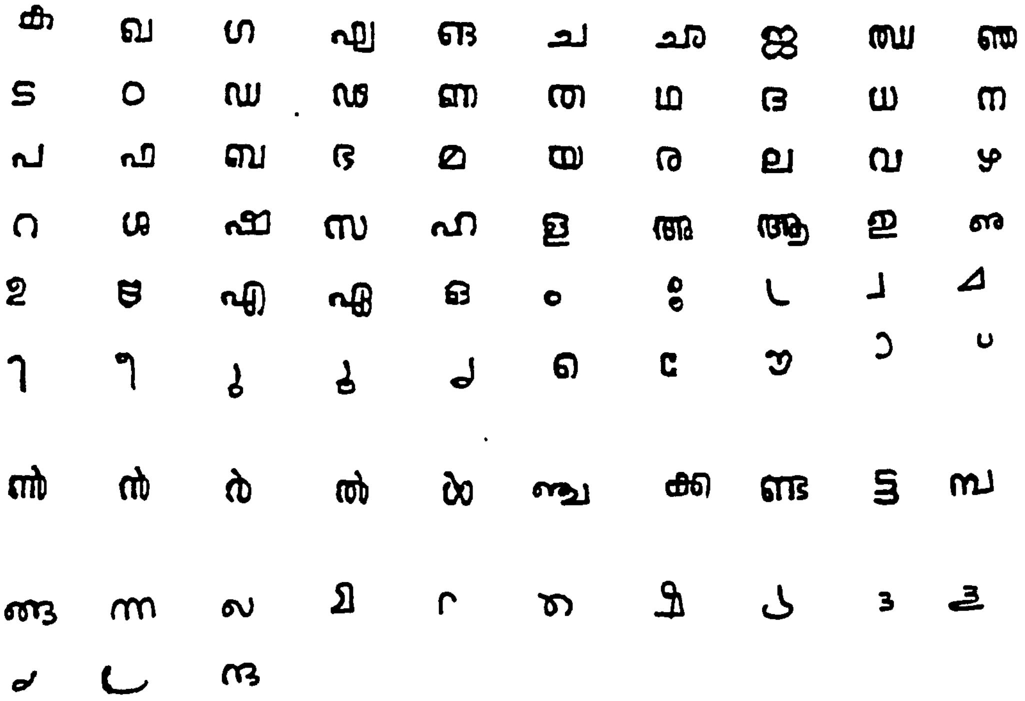


Figure 7.2 Set of *g*-symbols for Malayalam characters.

A Ā I Ī U Ū E Ē AI O

Ō AU AKH SHRI J S Ṣ H KṢ

K Ṅ C Ñ Ṭ Ṇ T N P M

Y R L V Z Ḷ R̤ N̤

(a) Set of *p*-symbols for Tamil characters.

A Ā I Ī U Ū Ṛ Ṝ Ḹ E

Ē AI O Ō AU AM AK

K KH G GH Ṅ C CH J JH Ñ

Ṭ ṬH Ḍ ḌH N T TH D DH N

P PH B BH M Y R L V S'

Ṣ S H Ḷ Z R̤

(b) Set of *p*-symbols for Malayalam characters.

Figure 7.3 Set of *p*-symbols of Tamil and Malayalam.

ten Indian languages is not a large number like in Case (3) above. Because of the common phonetic characteristics of the Indian languages, there are several common *p*-symbols between these languages. This makes it possible to design a keyboard containing all the *p*-symbols of the ten languages. Figure 7.4 gives a set of *p*-symbols covering the character sets of ten Indian languages.

7.1.2 Assignment of Symbols

The next important step is the assignments of the symbols of s_i to the key-positions. That is, finding the mapping: $f_k : s_i$ ---> K. We could use the *n*-gram frequencies of the symbols, the characters and frequently occurring words in the language to arrive at f_k. These frequencies 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 a *p*-symbol is given below.

Let,

β_{ki} : the character frequency of i^{th} character in k^{th} language where $1 \leq i \leq N_i$ and $1 \leq k \leq T$

P : the set of *p*-symbols

| P | : cardinality of P

s_j : j^{th} symbol in P

ω_{kj} : the frequency of j^{th} symbol in k^{th} language

A Ā I Ī U Ū Ṛ Ṝ Ḷ' Ḹ

Ĕ E Ê Ē ĂI AI Ŏ O Ô Ō

ĂU AU AM AK AKH SHRI

K KH G GH Ṅ C Ĉ CH J Ĵ

JH Ñ Ṭ ṬH Ḍ Ṛ' ḌH ṚH Ṇ T

T̲ TH D DH N P PH B BH M

Y Ẏ R L W V Ḻ Ḷ R̤ N̤

S' Ṣ S H KṢ F' Z Ṛ RXH Q

Figure 7.4 Set of *p*-symbols of ten Indian languages.

Ω_j : the average frequency of j^{th} symbol over T languages

Then,

$$\omega_{kj} = \sum_{l=1}^{|N_k|} m_l \, \beta_{kl}$$

$$\Omega_j = \frac{1}{T} \sum_{k=1}^{T} \omega_{kj}$$

where m_l is the number of times the s_{kj} occurs in the character c_{kl}.

As an illustration consider the Tamil characters. The frequencies of these characters obtained from a sample of approximately 24,000 characters taken from modern prose are given in Table 7.1. For example, the frequency of the *p*-symbol U is obtained by summing the values in column 5 and that of the *p*-symbol C is obtained by summing the values in row 4.

The above expression for Ω_j is based on the assumption that the probability of occurrence 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 terminal is used, the probability will differ. For example, a terminal in Tamil Nadu is likely to handle more Tamil text than text of any other language. Taking this into account, the expression could be modified as:

Table 7.1 Frequencies of Tamil Characters.

	A	Ā	I	Ī	U	Ū	E	Ē	AI	O	Ō	AU	AK	SHRI
	478	133	386	010	154	016	345	041	017	100	012	001	001	010
K	1053	217	290	016	400	071	009	043	090	126	038	001	742	
Ṅ	003	000	000	000	000	000	000	000	000	000	000	000	233	
Ç	198	035	209	017	055	005	083	022	026	053	012	002	150	
Ñ	012	003	000	000	000	000	000	000	002	000	000	000	019	
Ṭ	286	076	252	005	304	002	003	028	145	002	000	000	326	
Ṇ	097	009	047	002	003	000	000	000	009	000	000	000	252	
T	659	204	428	012	573	014	057	047	109	031	035	000	624	
N	107	114	064	040	003	019	012	012	003	000	017	000	419	
P	493	141	188	012	112	010	107	038	019	064	105	000	433	
M	248	165	069	012	019	010	007	024	081	007	007	002	850	
Y	426	031	164	000	121	003	009	043	067	000	019	000	107	
R	326	107	209	019	479	001	004	010	045	003	009	000	556	
L	217	091	105	001	084	003	008	038	131	000	008	000	548	
V	610	143	259	025	092	003	043	110	032	000	001	000	043	
Z	055	002	054	000	053	000	000	000	025	000	000	000	071	
Ḷ	095	029	110	006	074	007	001	003	115	001	006	000	393	
R̤	203	075	137	001	255	000	001	014	073	004	005	000	280	
N̤	258	100	103	001	065	004	000	016	081	002	012	000	850	
J	034	004	001	002	000	000	000	007	003	000	000	000	006	
SH	020	001	003	000	000	000	000	000	000	000	000	000	014	
S	002	000	002	000	000	000	000	000	000	000	000	000	046	
H	011	001	001	000	000	000	000	000	000	000	000	000	000	
KS	000	002	000	000	000	000	000	000	000	000	000	000	000	

$$\omega_j = \sum_{k=1}^{T} p_k \, \omega_{kj}$$

where p_k is the probability of occurrence 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.

7.1.3 g-symbol and p-symbol Keyboards

Figures 7.5 and 7.6 give *g*-symbol keyboards for Tamil, and English respectively. The *p*-symbol keyboards for Tamil and Malayalam are shown in Figures 7.7 and 7.8. The keyboards shown in Figures 7.7 and 7.8 are based on the set of *p*-symbols shown in Figure 7.3. In these keyboards, some key positions will remain unused since there are more key positions than the *p*-symbols in each language. The Tamil *g*-symbol keyboard is based on the modern (newer or modified) character set. Hence, it is not possible to type some of the characters in the older scheme such as றா (R̤A), ணா (ṆA), னா (N̤A), ணை (ṆAI), னை (N̤AI), லை (LAI), and ளை (ḶAI) using this keyboard.

The symbols have been assigned to the key positions according to their frequencies such that the highest load is assigned to the middle finger, then to ring finger, index finger and the least to the small finger. It is assumed that middle finger is the strongest and the small finger is the weakest. It is also assumed that the right

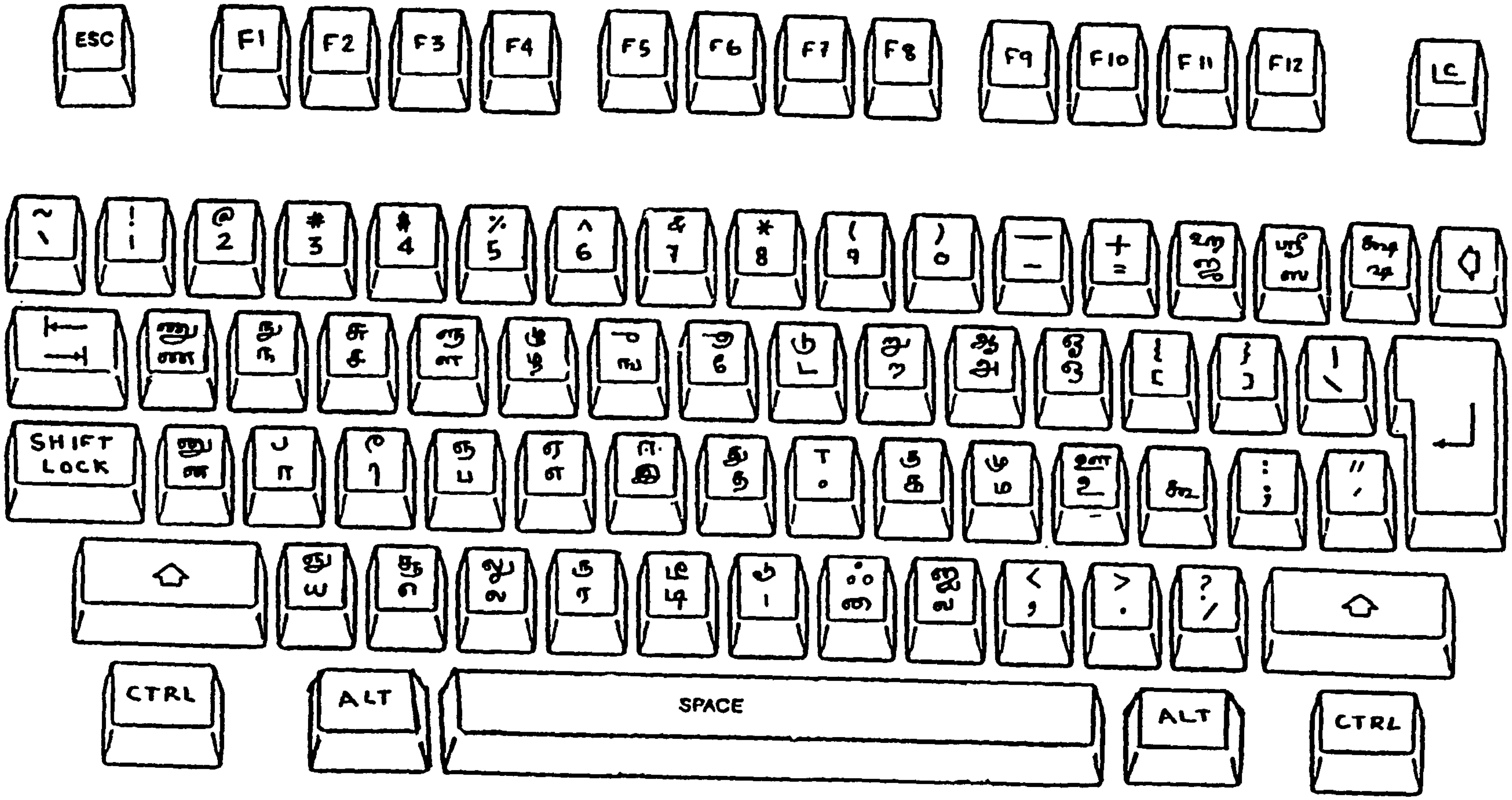


Figure 7.5 Multilingual *g*-symbol keyboard for Tamil.

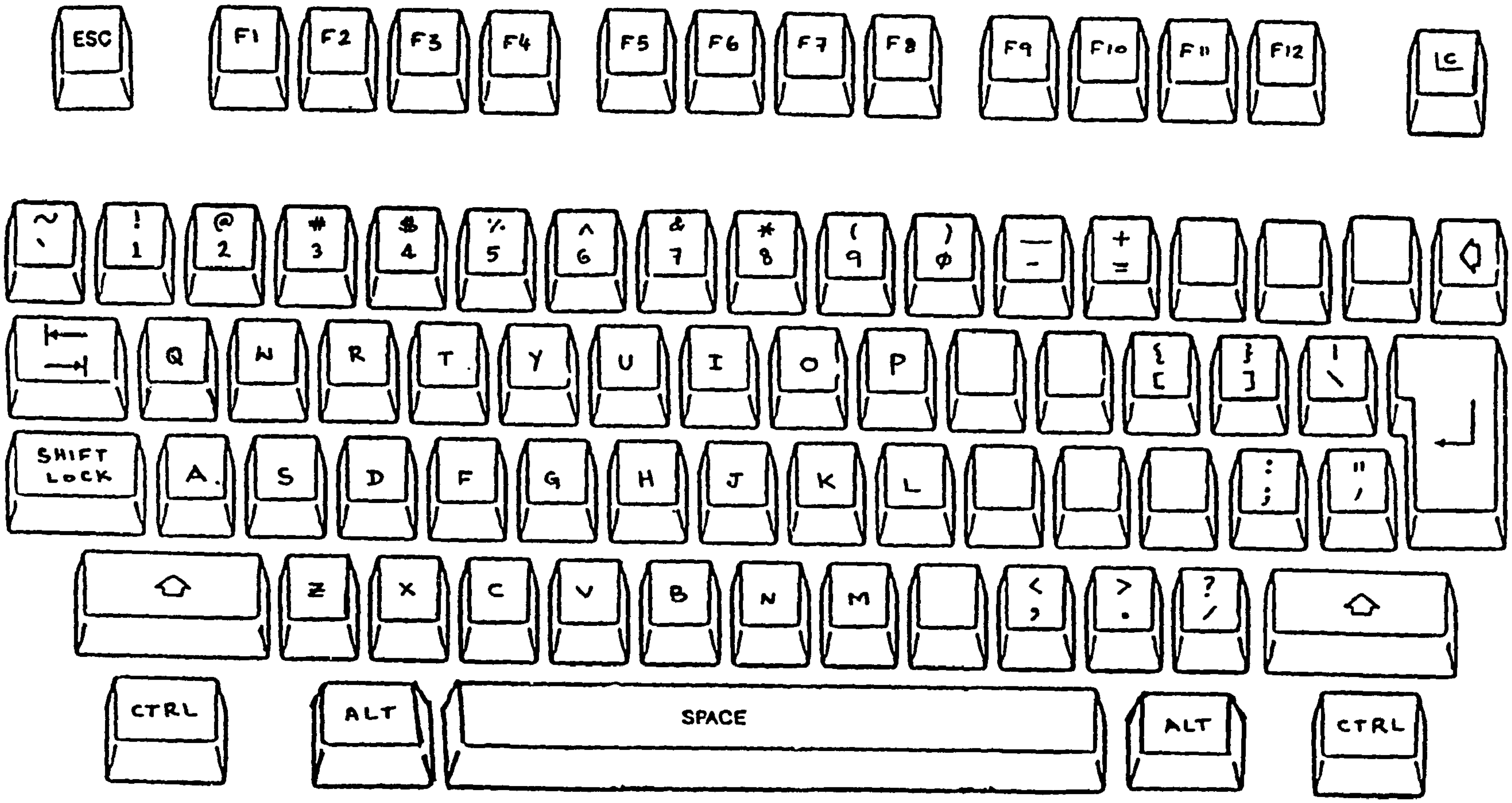


Figure 7.6 Multilingual *g*-symbol keyboard for English.

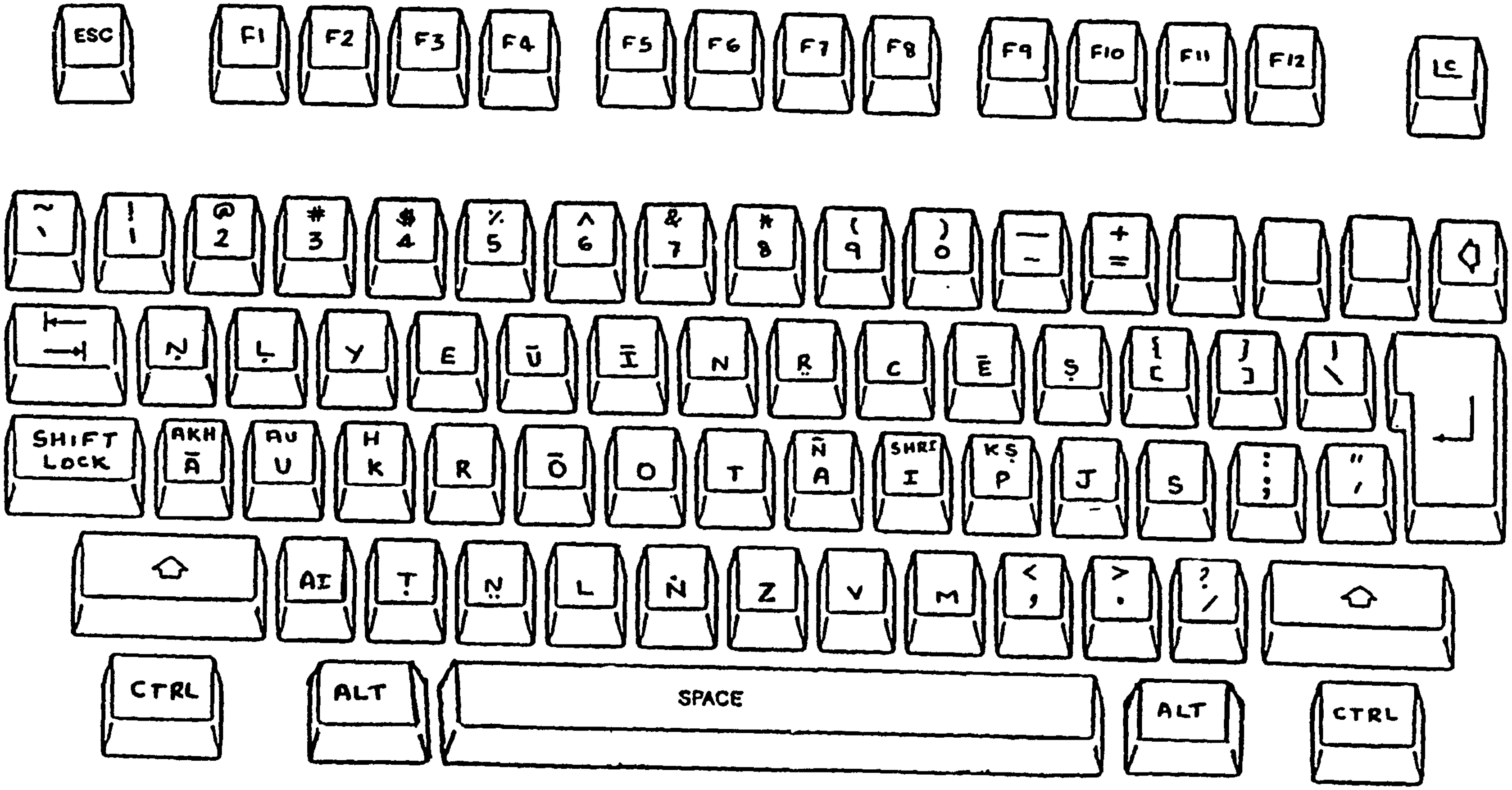


Figure 7.7 Multilingual *p*-symbol keyboard for Tamil.

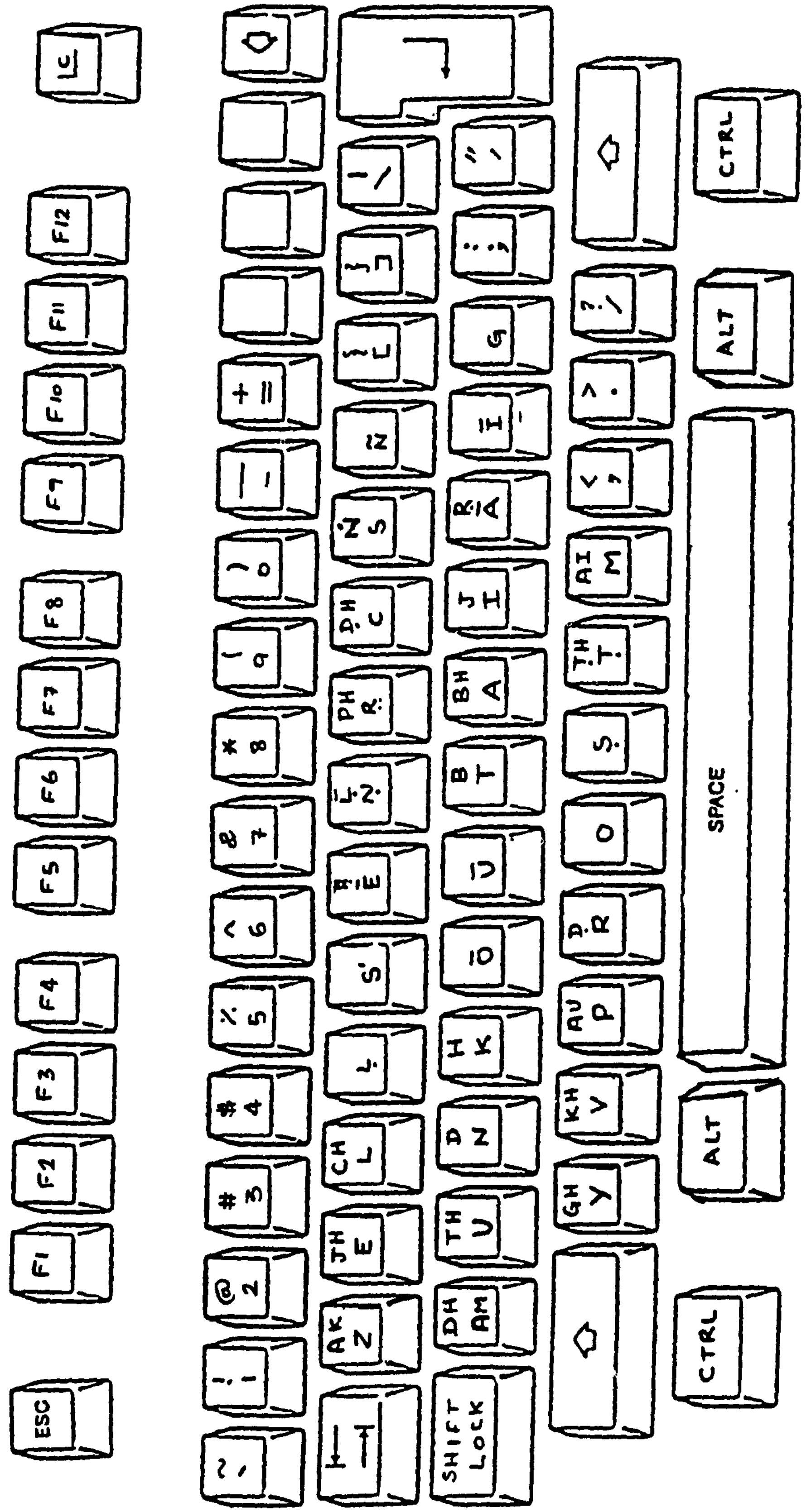


Figure 7.8 Multilingual *p*-symbol keyboard for Malayalam.

hand fingers are stronger than the corresponding left hand fingers. Symbols were distributed first to the row where the fingers are normally kept, then to the row below and then to the row above with the assumption that it is easier to move the fingers down than up.

The *g* anc *p*-symbol frequencies for Tamil have been computed from the character frequencies shown in Table 7.1. The *p*-symbol frequencies for Malayalam have been computed by collecting the characters from modern Malayalam text books. They are listed in Figure 7.9. These frequencies have been used in assigning the symbols to key positions.

In the Tamil *g*-symbol keyboard, in order to easily remember the symbols on the keys, related symbols (for example, எ (E) and ஏ (Ē), ண(N̤A) and ணு(N̤U), etc.) have been assigned to the lower and upper positions of the same key as much as possible. This has affected the placement of some of the higher frequency symbols such as து (TU), ரு (RU), கு (KU), டு (TU) and று (R̤U) which may result in some loss of efficiency. However, the advantages of easy to remember and (hence) fast learning should more than offset the decrease in efficiency.

In a similar manner, a keyboard can be designed for the *p*-symbols shown in Figure 7.4 (Krishnamoorthy [1981]). It could be used to enter texts of ten different Indian languages without changing the key-top display. In this keyboard some of the

Symbol	Frequency	Symbol	Frequency
A	1061	Ṭ	179
Ā	324	ṬH	11
I	406	Ḍ	11
Ī	41	ḌH	4
U	383	Ṇ	103
Ū	61	T	381
Ṛ	17	TH	20
Ṝ	0	D	30
Ḹ	0	DH	17
E	106	N	411
Ē	47	P	198
AI	14	PH	9
O	50	B	19
Ō	59	BH	31
AU	12	M	203
AM	267	Y	159
AK	0	R	176
K	364	L	142
KH	12	V	180
G	35	S'	44
GH	10	Ṣ	51
Ṅ	0	S	71
C	110	H	19
CH	4	Ḷ	99
J	27	Z	68
JH	4	R̤	157
Ñ	31		

Figure 7.9 Frequencies of Malayalam *p*-symbols.

key positions will remain unused for a particular language. An optimal placement of the symbols for this keyboard will be very difficult since what is optimal for one language may not be optimal for another language because the frequencies of the same symbol are likely to differ between languages; in addition, the *p*-symbol frequencies for ten different languages are not available.

Notice that there is a separate key-position for *language selection code* specification. Whenever text of a different language has to be entered, type *language selection code* key followed by the *language number* and then type the text of that language. To type English text, type the language selection key followed by the language code for English and then the English text like in conventional keyboard.

Since the same keyboard unit is used to enter multilingual text, it is necessary to display on the key-top the appropriate symbols of the language it currently handles. Methods such as template replacement and display panel could be used for this purpose. The former method is practical and cost effective whereas keyboards with display panel are not yet available in the market because of the need for new chips and economical considerations.

The English language keyboards in use today are not well designed for the operator's convenience and attempts are being made to produce better keyboard designs (Montgomery [1982]). The keyboard suggested by Montgomery for example, would allow an operator to type the most common English words with *wiping motions*. The layout for the Indian language keyboards do not take into account the principles discussed in Montgomery. Mechanical typewriters based on *g*-symbol keyboards are available for Indian languages. Computer terminals based on *p*-symbol keyboards are being built and sold for a limited number of Indian languages in India. These terminals handle English in addition to Indian languages.

A unilingual keyboard based on the phonetic properties of Hindi and Telugu languages was proposed by Narasimham, Prasada and Rajaraman [1971], and Laturkar and Sinha [1978]. Their proposals differ from the approach outlined in this thesis in the following ways.

- In their input, it is necessary for the keyboard operator to indicate explicitly which of the elements of the *p*-symbol sets are to be grouped to form composite characters.
- The keys are laid in a non-standard "V" shape.
- There are 62 keys in a single shift, which makes touch typing difficult.
- There are diacritical marks on the keyboard which implies that their design is based on a combination of *g*-symbols and *p*-symbols.

The *g*-symbol keyboard proposed in this thesis closely resembles the existing mechanical typewriter keyboard for Tamil. Hence, the existing typists could easily use this keyboard without extensive training. However, it has the following differences.

The character ணு(ṆU) is obtained by one key stroke whereas mechanical typewriters need two key strokes. The character ஹ (HA) is provided as a separate character whereas in mechanical keyboard it is obtained by concatenating the two characters உ (U) and ர (RA). Similarly, the characters ஷ (SHA), ஸ (SA) are provided as separate characters. In the mechanical keyboard, they are obtained by using the characters கூ (KUU), உ (U) and the symbol ஃ . Such usage in the electronic keyboard will be confusing while generating characters. The character ஞு (GNU) is provided, thus getting the complete Tamil character set. This character is not present in the mechanical typewriter. It is also not used by the printers. Unlike mechanical keyboard, there is no dead key in the proposed keyboard. With the proposed keyboard certain characters like கி (KI) can be typed in the natural order of writing like க followed by ி whereas it is typed in the mechanical keyboard as ி followed by க using the dead key containing the symbol ி .

The *p*-symbol keyboard proposed in this thesis has the advantage that it could be configured to be language specific or for a set of languages. We can collate the text in alphabetical order for any language. The proposed keyboard can also be used to enter non-Indian texts such as English, French, German, Sinhalese and other European and non-Roman languages. The keyboards proposed in this thesis have the advantage that we could have *g*-symbols on the keyboard and use *p*-code for internal representation as shown in Chapter 8. The *p*-code gives the text in collating sequence on sorting; also, with *p*-code we could easily transliterate texts from one

Indian language to another.

There is still room for improvements in the design of the proposed keyboards. The number of keys could be reduced by associating the special symbols such as + - * % & ' " etc., only with the English version of the keyboard. However, this will require language change for special characters and a little bit extra typing. In the placement of symbols only 1-gram frequency has been taken into consideration. The placement of symbols, in particular for *p*-symbols, on the keys could be further refined by considering word frequencies and n-gram frequencies. Character frequencies have yet to be obtained for several Indian languages and used successfully in the keyboard design.

7.1.4 Comparison of g-symbol and p-symbol Keyboards

- In the *g*-symbol keyboard there will be immediate feedback (echo) for each key depression whereas in a *p*-symbol keyboard, the character will appear only after typing the complete *p*-symbols of a character.
- A key-top symbol display is needed for the *g*-symbol keyboard, whereas it may be eliminated with the *p*-symbol keyboards using the union of *p*-symbols of Indian languages like in Figure 7.9.
- With the *p*-symbol keyboard, the operator has to learn, in addition to the graphemes, the *p*-symbols and the structure of the script.
- The keyboards can be compared on the basis of the average number

of key depressions required per character. This could be computed by using the following expression:

$$\mu = \sum_{i=1}^{N_l} p_l \, d_l$$

where p_i is the probability of occurrence of character *i*, d_i is the number of key depressions required for character *i*. d_i will be different for *g*-symbol and *p*-symbol keyboards. Using the character frequency in Table 7.1, the average number of key depressions required using the *g*-symbol keyboard was found to be 1.56, and that for *p*-symbol keyboard 1.63. This shows that both keyboards are almost equally efficient.

- Internal machine coding (refer Chapter 8) based on the *p*-symbol gives the text in collating sequence on sorting, whereas *g*-symbol based coding may not.

7.2 Multilingual Text Generation System

A major task in the design of a multilingual output device is the development of a scheme for displaying, printing or plotting the multilingual texts that have graphically different scripts. The text entered from the keyboard must be echoed. As well as, output from the computer needs to be either printed or displayed in the selected language. 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. The following definitions are used in this section.

Definition 7.3: Context Dependent Language

A language, from text generation point of view, is said to be context dependent, if any character the language takes different graphemes depending on the preceding or succeeding character.

Definition 7.4: Context Dependent Text Generator

The text generator for a context dependent language is called a context dependent text generator.

7.2.1 Issues in Text Generation

Before going into the techniques of generation, we must note important differences between generating characters in Indian languages and in English. On the English keyboard, each symbol on the key is a complete character (alphabet, digit or special characters). Hence, the character could be displayed without the need to keep track of the preceding or succeeding characters keyed in. The generation of English characters is trivial with the present technology due to its small number of simple characters.

Such is not the case for Indian characters, Chinese characters, etc. There are many characters in each Indian language ranging from 247 in Tamil to over 2000 in Telugu. The characters are larger in size than English and are more complex in shape. The characters also vary in both height and width. The symbols in a character do not always appear side by side without touching as in English. Some symbols appear one below the other, with or without touching. In some cases the same *g*-symbol touches other symbols at different places; the physical distance necessary to move back to append a *g*-symbol, in certain cases, depends on the preceding *g*-symbol. Above all character generation is *context* dependent in certain languages like Malayalam.

7.2.2 Unified Approach

The output device of a multilingual terminal need not be restricted to one type. In other words many types of output devices such as dot matrix printer, display device, plotter, etc., could be used. Therefore, a multilingual text generator should be as device independent as possible in order to be portable between different types of devices. The analysis of the scripts of major world languages, has revealed that it is possible to develop a unified approach for multilingual text generation as a hierarchical system of software on a microcomputer that could perform (a) switching between scripts, (b) parsing the code for the input text, (c) character generation including context dependencies, (d) intercharacter spacing and (e) interline spacing. The text generator could consist of (1) a set of device dependent primitives, (2) symbol generator, (3) character generator, (4) text generator and (5) input decoder.

For Indian languages, a unified multilingual text generator based on the p-symbol coding could be constructed. The p-symbols of an Indian language can be grouped as follows:

1. set of vowels, $V = \{v_1, v_2, \dots, v_n\}$
2. set of consonants, $C = \{c_1, c_2, \dots, c_m\}$
3. set of special symbols, $S = \{s_1, s_2, \dots, s_q\}$
4. set of p-symbols, $P = V \cup C \cup S$

A vowel, v can combine with one or more consonants and make a composite character. Such a consonant-vowel sequence for a character could be represented as

$$K = c_1c_2 \dots c_m \mathrm{v}$$

It is possible that in certain languages like Tamil a member of P can appear as a separate character also. Only one vowel need to be present in any given composite character.

A state machine could be constructed to generate texts of a language from p-symbol code of that language.

Theorem 7.1: Given that the longest sequence of consonant-vowel for a character in a language be $c_1c_2...c_kv$. It is possible to build state machine with $k+1$ states to generate text from a string of p-symbols of that language assuming that each state can seo the input string and each state is capable of declaring the sequence up to the current symbol as valid or invalid character sequence.

Proof: Let s_0 be the starting state. For any input p-symbol which is not a consonant-vowel sequence, stay in s_0 and output the p-symbol as valid or invalid character. For a consonant-vowel sequence of the form $c_1c_2...c_kv$, on c_1 move from state s_0 to state s_1, on c_2 move from state s_1 to state s_2, and so on. The transition from c_1 to c_k needs k states. On encountering v, declare the sequence as valid, generate the character and change state from s_k to s_0. If the genarator is context dependent then look before/ahead and then generate the character. At any state, if the sequence encountered so far is invalid, declare the sequence as invalid and change state to one of the previous states; before state transition one or more p-symbols can be discarded as invalid or generated as independent character. After transition, continue decoding from that state with the remaining input string. Thus, we could build a state machine with $k+1$ states.

In Indian languages, the p-symbols are consonants, vowels and special characters. The composite characters are formed by combining one or more consonants with a vowel. In Tamil one consonant can combine with one vowel and form a composite character with a form cv. In Malayalam, one or two consonants can combine with a vowel to make a composite character; it is of the form *ccv*. The state machines developed for Tamil and Malayalam are given in Figures 7.10 and 7.11 respectively.

7.2.3 Techniques For Graphemes Generation

For display purposes, the graphemes of the characters may be generated, by one of three methods: (1) linguistic, (2) line segment and (3) dot matrix. (Krishnamoorthy, et al [1980]). These three methods are briefly described in the following paragraphs.

Linguistic Method: This approach is well suited to a language with a large number of characters. It uses 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, until a complete character is formed.

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

state	input			
	v	c1	c2	k
s1	s1/v	s2	-	s1/k
s2	s1/c1v	-	s1/c1	s1/c1,k

s1	: Initial state
-	: Invalid input for that state
v	: Vowel
c1, c2	: Consonant
k	: Special Character

Figure 7.10 State machine for generating Tamil characters.

state	input				
	v	c1	c2	c3	k
s1	s1/v	s2	-	-	s1/k
s2	s1/c1v	-	s3	-	s1/c1,k
s3	s1/c1c2v	-	-	s2/c	s1/c1,c2,k

s1	: Initial state
-	: Invalid input for that state
c	: output whatever consonant input so far
v	: Vowel
c1, c2, c3	: Consonant
k	: Special Character

Figure 7.11 State machine for generating Malayalam characters.

components.

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 [1969]. However, if this grammar model were directly applied to Indian characters, 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 the complexity of the characters, the number of intermediate p'ırases would be large. Further, such a straightforward model would not take into account the similarities in the structural derivations (Krishnamoorthy and Issac [1978]) 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 (Krishnamoorthy [1981]).

Line Segment Method: In this method, each symbol is represented as a set of line segments. The relative coordinates of the segments are stored in a table. By displaying the line segments of a symbol one after another, the symbol gets displayed. The difference between linguistic method and line segment method is that in the later there is no grammar involved. The line segment method is simple and straightforward as compared to the linguistic method, but requires more memory. In line segment method diagonal lines look continuous whereas in dot matrix method the strokes are discontinuous and they look more spotted. Small dot matrix characters look much more clearer than characters made up of line segments.

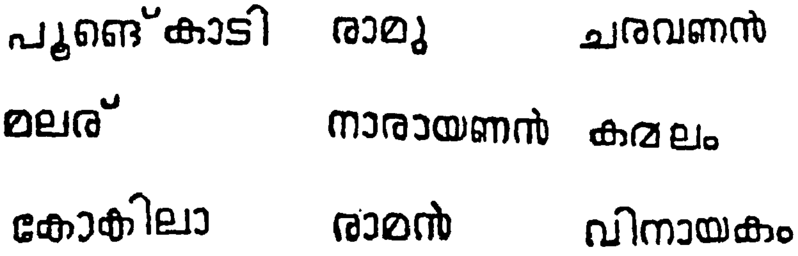
Dot matrix Method: This method differs from the previous two methods in the sense that each symbol is represented by a matrix of dots instead of line segments. Dot matrix type displays and printers are widely used for Roman characters. In these devices a fixed size of 9x7, dot matrix for each Roman character is assumed. However, a much larger size is needed for Indian and Chinese languages. The dot matrix printers could be either column printers or line matrix printer. In the former case, a symbol is printed column by column whereas in the later case one line is printed row by row. Figure 7.12 and 7.13 give samples of Tamil and Malayalam texts generated using the state machines and the dot matrix method described above.

7.2.4 Storage Optimization

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 primitives. However, the number of rewriting rules would be small and the computation time for interconnecting the primitives to generate

மார்கழி கழிந்தது மாதையும் பிறந்தது
வார்கழல் மாதவன் வாழ்த்தொலி கேட்டது
சேரிழை மஞ்சள் செவ்வள்ளிக் கிழங்கோடு
வேருடைக் கரும்பு வெண்ணரிசி நிறைந்தது
பாருடை மாந்தர் பரிவுடன் பொங்கலே
பொங்கலோ பொங்கல் பொங்கலோ பொங்கல்

Figure 7.12 Sample Tamil text.



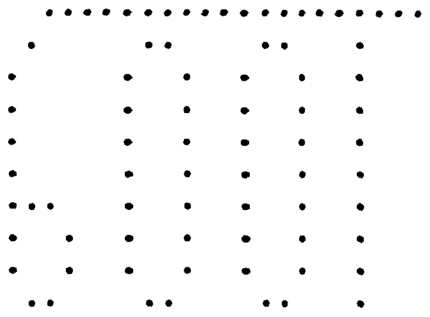
പൂങ്കാടി	രാമു	ചരവണൻ
മലര്	നാരായണൻ	കമലം
കോകിലാ	രാമൻ	വിനായകം

Figure 7.13 Sample Malayalam text.

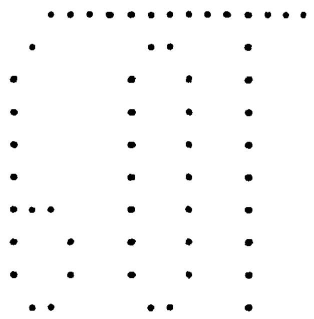
the picture elements would be small. In the second case, the storage required for primitives would be smaller, but the number of rewriting rules would be large. Also the computation time for interconnection of picture primitives would be high. The primitives should be selected such that an acceptable compromise is achieved between the storage and computation requirements. In the case where the set of primitives is the same as the *g*-symbol set to be generated, the syntactic picture generation reduces to triviality for symbol generation but not character generation. Unfortunately no formal methods are available to select the set of primitives for a specified class of pictures. Scripts 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 characters.

Storage could be minimized in the case of dot matrix method also by minimizing the storage required for the dot matrices of the symbols. For example, consider the dot matrices for the symbols NNA and NA as shown in Figure 7.14. There are ten dots in each column. These dot matrices could be represented by a string of (octal) integers as given below, where each integer represents a vertical column and is obtained by treating the dots in each column as a binary number of size 10 binary digits.

NNA = <0376, 0411, 1011, 1006, 1000, 1000, 1376, 1401, 1401, 1376, 1000, 1000, 1376, 1401, 1401, 1376, 1000, 1000, 1777, 1000, 1000, 1000>



(a) Dot matrix of ण्ण (NNA).



(b) Dot matrix of ण (NA).

Figure 7.14 Dot matrices for ण्ण (NNA) and ण (NA).

NA = <0376, 0411, 1011, 1006, 1000, 1000, 1376, 1401, 1401, 1376, 1000, 1000, 1777, 1000, 1000, 1000>

Coding is done column by column, from left to right, considering the bottom-most dot as the least significant digit.

Comparison of the two strings reveals that there is a common substring which corresponds to the common graphic pattern between NNA and NA. Suppose,

ψ_1 = <0376, 0411, 1011, 1006, 1000, 1000, 1376, 1401, 1401, 1376, 1000, 1000>

ψ_2 = <1376, 1401, 1401, 1376, 1000, 1000, 1777, 1000, 1000, 1000>

ψ_3 = <1777, 1000, 1000, 1000>

now, NNA and NA could be written by the production rules:

$$\omega_1 = \psi_1 + \psi_2$$

$$\omega_2 = \psi_1 + \psi_3$$

where + is used as the linear concatenation operator. Ψ_1, Ψ_2 and Ψ_3 are the picture primitives in the linguistic method; ω_1 and ω_2 are the production rules.

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

$$\Phi = \{\phi_1, \phi_2, \dots \phi_m\}$$

$$\Psi = \{\psi_1, \psi_2, \dots ,\psi_n\}$$

$$\Omega = \{\omega_1, \omega_2, \dots ,\omega_m\}$$

The set Ψ is constructed such that every ψ_j is a subsequence of integers contained in at least one ϕ_i; and every ϕ_i can be constructed as a linear concatenation of one or more ψ_js. Let ω_i denote the production rule for ϕ_i and α_κ denote the storage required for the item κ. Then,

$$\alpha(\Phi) = \sum_{i=1}^{m} \alpha(\phi_i)$$

$$\alpha(\Psi) = \sum_{i=1}^{n} \alpha(\psi_i)$$

$$\alpha(\Omega) = \sum_{i=1}^{m} \alpha(\omega_i)$$

$\alpha(\Phi)$ is unoptimized storage requirement for the set of symbols Φ. If $\alpha(\Phi) > (\alpha(\Psi)+\alpha(\Omega))$, we have reduced the storage requirement by an amount $\alpha(\Psi)+\alpha(\Omega)$. The objective is to optimize the storage requirement by minimizing $\alpha(\Psi)+\alpha(\Omega)$.

Theorem 7.2: The optimization of the storage requirement for the set of dot matrices of the symbols, Φ is *np*-complete.

Proof: The theorem will be proved by showing that this optimization problem is the same as the *external macro data compression* problem discussed in Garey and Johnson [1979] which is an *np*-complete problem. Designate the set Ψ as the set of dictionary strings and the set of production rules Ω as the compressed strings. Replace each occurrences of ψ_j in ω_i by a pointer to ψ_j. Represent the concatenation operator as a null entry $\emptyset$ such that $\alpha(\emptyset)=0$. Thus, each compressed string will contain a string of integers and a set of pointers to members of Ψ. Given the dictionary strings, the compressed string and a mechanism to identify pointers, we can construct the dot matrix of a symbol from the compressed string, by repeatedly replacing pointers in the compressed string by their corresponding dictionary string. Now, the original storage optimization problem is equivalent to optimizing the storage requirement for the dictionary and compressed strings. This is an external macro data compression problem which is an *np*-complete problem (Gary and Johnson [1979]). Thus, dot matrix storage optimization problem is *np*-complete.

Storage Reduction: Since optimization is *np*-complete, we could only try to reduce the storage requirement as much as possible by using data compression techniques (Storer [1988], Held [1983]). In the above example for NNA and NA, assuming one word of memory for each integer as well as for a pointer, the uncompressed version requires 38 words of memory to represent both NNA and NA whereas the compressed version requires only 26 words which is more than 30% reduction in memory requirement. Notice that storage for ψ_2 and ψ_3 could be further minimized by treating <1777, 1000, 1000, 1000> as another common string.

There is a tendency to think that memory is cheap, hence we should not worry about storage optimization for the dot matrices. However, one should remember that a terminal supports not just one but *n* text generators; hence the overall saving will be *n* times. It should be realized that the terminal market is very competitive. A ten dollar saving in components results in almost a fifty dollar saving in the overall product cost. Hence, storage minimization problem should be given serious consideration.

7.3 Technical Problems

The following are the important technical problems encountered in designing a multilingual terminal:

- Changing symbol display on keyboard with change in language.
- Symbolization and construct are difficult to obtain for languages like Chinese.
- Symbolization and construct development is manual.
- No technique/method exists to verify optimality of symbol set.
- Coding of symbols should give lexical ordering of the text of a language on sorting. It is difficult for the *g*-symbol set for Indian languages.
- Assignment of symbols to key-positions needs frequency distribution. If a common set of symbols is used to represent more than one language, an allocation may result in optimum distribution for one language and not for others.

8. MULTILINGUAL COMMUNICATION SYSTEM

8.1 Introduction

The various aspects in multilingual document interchange are shown in Figure 8.1. They are:

1. accepting document
2. entering into Computing and Communication Systems (CCSs)
3. storing in the database, if needed, before transmission
4. transmitting according to a known protocol
5. receiving
6. storing in the database, if needed, before delivery
7. delivering

Suitable multilingual I/O devices described in the previous chapters are assumed to be available for entering text into CCSs. Also, multilingual programming tools explained later must be available for developing multilingual CCSs. Various communication aspects are described in the following sections.

The aim of this chapter is to address the basic issues such as document structure, character coding, etc., which are common to all data communication protocols. It is not intended to revise every communication protocol in use today. However, as an illustration, a possible way to enhance the NFS (Network File System)

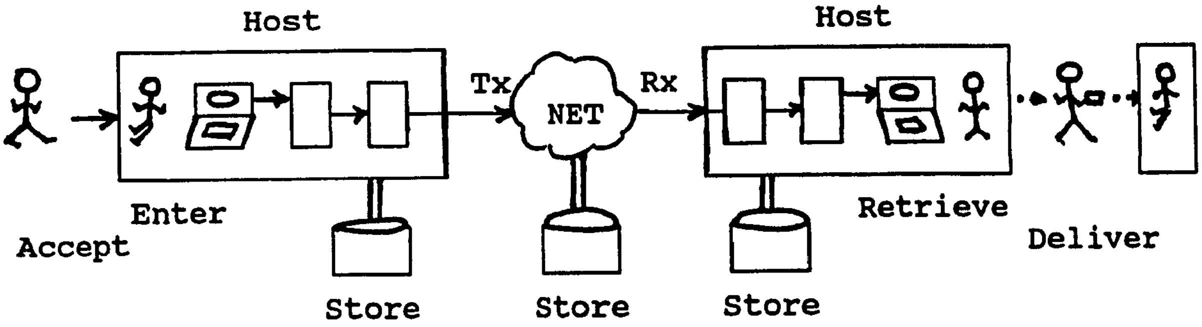


Figure 8.1 Elements of multilingual data communication system.

protocol is shown later. Similarly other specific protocols could be modified.

8.2 Multilingual Document Structure

In the case of unilingual document, the text in the document belongs to one particular language, say English. If the language used in a CCS is always the same, the language information could be omitted like in the existing systems. However, a multilingual document (file) contains texts of one or more languages as illustrated in the previous chapters. In this case it is essential to indicate the linguistic information explicitly in the document.

Therefore, according to our unified document structure, every document, either unilingual or multilingual is made of two components

$$<\text{document}> := <\alpha> <\text{texts}>$$

where α is the set of *attributes* of the *texts* component. The *attributes* component is further subdivided as follows:

$$<\alpha> := <\alpha_1> <\alpha_2> \ldots <\alpha_i> \ldots <\alpha_n>$$

where α_i is a set of attributes of texts belonging to the language *i* in a given document.

Let,

$$<\Omega> := <\omega_1> <\omega_2> \ldots <\omega_i> \ldots <\omega_m>$$

where Ω is the Universal set of document attributes such as bold, italic, underline, superscript, subscript, etc., and the set of language specific attributes. The relationship $\alpha_i \subseteq \Omega$ is true. According to this scheme non-language documents such as graphics can be easily handled by treating graphics as a language. The Braille script used by the blind people can also be stored and transmitted as long as suitable I/O devices are available with the CCSs.

8.3 Language, Character Set and Coding

A given *text* is associated with a language and the character set of that language. In order to code a text, it is essential to code the character set. For multilingual texts, it is essential to code the language also. In the following paragraphs, the evolution of the existing character set and its coding are reviewed. A new universal multilingual coding scheme is described.

8.3.1 Evolution of Character Set

Telex is one of the earliest computer communication systems developed for international information interchange. The telex character set contains the uppercase letters A to Z of the English language and other control and special characters such as - * , etc. Computers at this age had 6 bits per byte to accommodate the uppercase letters A to Z.

The technological advances, the decreasing cost and wide spread applications of CCSs among the public and office environments in non-numeric computation resulted in the necessity to handle the lowercase letters a to z of the English language. Computing systems increased the byte size from 6 to 8 bits to handle the letters a to z. The data communication systems were forced to handle the lowercase letters too. Hence, CCITT improved the communication system from telex to *teletex* and enhanced the character set as G0 character set to handle the lowercase letters a to z as shown in Figure 8.2 [CCITT Recommendation S63].

Many European languages, in addition to the English alphabet, use diacritical marks in their character sets as shown in Chapter 1 (Gilyarevsky [1970]). The pressure and need from these nations made the CCITT and the computer manufacturers to further enhance the character set as G2 character set to include the diacritical marks as shown in Figure 8.3 [CCITT Recommendation S63].

Notice that these character set enhancements are not necessitated by technological needs; it is purely linguistic implications on CCSs.

8.3.2 Future Character Set

The existing computer and communication character set is limited to Latin script languages. It does not handle languages using Indo-Asian, Arabic and Cryllic scripts shown earlier. The majority of the population of the world use Indo-Asian and Cryllic

	2	3	4	5	6	7
0	SP	0	@	P	'	p
1	!	1	A	Q	a	q
2	"	2	B	R	b	r
3		3	C	S	c	s
4		4	D	T	d	t
5	%	5	E	U	e	u
6	&	6	F	V	f	v
7	'	7	G	W	g	w
8	(	8	H	X	h	x
9	)	9	I	Y	i	y
10	*	:	J	Z	j	z
11	+	;	K		k	
12	,	<	L		l	|
13	-	=	M		m	
14	.	>	N		n	
15	/	?	O	①	o	▨

Figure 8.2 CCITT G0 character set.

	2	3	4	5	6	7
0	▨	°		―	Ω	ĸ
1	¡	±	`		Æ	æ
2	¢	^{2}	´		Đ	đ
3	£	^{3}	^		ª	ð
4	$	×	~		Ħ	ħ
5	¥	µ	¯			ı
6	#	¶	˘		Ĳ	ĳ
7	§	·	˙		Ŀ	ŀ
8	¤	÷	¨		Ł	ł
9	‘	’			Ø	ø
10	“	”	˚		Œ	œ
11	<<	>>	¸		º	β
12	←	¼			Þ	þ
13	↑	½	˝		Ŧ	ŧ
14	→	¾	˛		Ŋ	ŋ
15	↓	¿	ˇ		ŉ	▨

Diacritical marks

Figure 8.3 CCITT G2 character set.

scripts. The nations using these scripts are using more and more computers and communication systems for information handling and interchange. They want the world CCSs to handle their languages in addition to Latin script languages. The current techniques used by the computer manufacturers, the CCITT, and other standards making organizations to enhance the character set is not suitable to handle varying number of multiple languages of the world. Therefore, it is essential to enhance the character set and adopt a new character coding scheme like the one proposed below.

8.3.3 Language Coding

In the existing unilingual CCSs, the language is implied and not explicitly specified. That is,

data := *text*

However, in a multilingual text representation, it is necessary to explicitly specify the language code. As shown in the earlier chapter, we represent any textual data as

data := <*lid*> <*text*>

lid := <*lc*> *n*

where *lid* is the language identifier and *text* is a set of characters represented by *lid*. In this scheme the *text* belonging to a language is preceded by its language code. Thus, it is possible to represent, store and retrieve multilingual text as a sequence of bytes of information.

The language code *lc* can be constructed as an ESC (escape) sequence or by a unique 8 bit code. Since ESC sequences are being used for other purposes by existing hardware and software, it has been decided to use a unique 8 bit code for <*lc*>. In both cases, globally unique number, *n*, representing a language is needed to follow the <*lc*>. For example, assign 0 to English, 1 for French, 2 for Tamil, etc. To make the least changes to the existing CCSs, English can be treated as the default language. International bodies such as CCITT can easily accomplish the task of assigning global number, *n* to each language in the world.

8.3.4 Character Coding

In 7-bits and 8-bits per byte, there are 128 and 256 character codes respectively. If one distinct code is used for <*lc*>, then the remaining codes could be used for the letters, numerals and special characters of each language.

As an illustration, the coding of the Indian languages Tamil and Malayalam is shown in Figures 8.4 and 8.5 respectively. It is based on the *p*-symbols of the languages given in the earlier chapters. Let us call such a *p*-symbol based codings as *p*-codes. It is possible to device character codes based on *g*-symbols also; let us call such codings as *g*-codes. For Indian languages, the *p*-codes have the advantage of giving the text, on sorting, in collating sequence; however, the *g*-codes may not work for all Indian languages. The character code for Latin script languages will be

	0	1	2	3	4	5	6	7
0				0	@	Ṅ	•	
1			!	1	A	C	L	
2			"	2	Ā	Ñ	V	
3				3	I	Ṭ	Z	
4				4	Ī	Ṇ	Ḷ	
5			%	5	U	T	R̤	
6			&	6	Ū	N	N̤	
7			'	7	E	P	J	
8			(	8	Ē	Y	S	
9			)	9	AI	Y	Ṣ	
A			*	:	O	R	H	
B			+	;	Ō		KṢ	
C			,	<	AU			|
D			-	=	AKH			
E			.	>	SHRI			
F			/	?	K			Lᶜ

Figure 8.4 7-Bit *p*-code for Tamil.

	0	1	2	3	4	5	6	7
0				0	@	AM	'	M
1			!	1	A	AK	Ñ	Y
2			"	2	Ā	K	Ṭ	R
3				3	I	KH	ṬH	L
4				4	Ī	G	Ḍ	V
5			%	5	U	GH	ḌH	S'
6			&	6	Ū	Ṅ	Ṇ	Ṣ
7			,	7	Ṛ	C	T	S
8			(	8	Ṝ	CH	TH	H
9			)	9	Ḹ	J	D	Ḷ
A			*	:	E	JH	DH	Z
B			+	;	Ē		N	R̤
C			,	<	AI		P	|
D			-	=	O		PH	
E			.	>	Ō		B	
F			/	?	AU		BH	L^c

Figure 8.5 7-Bit *p*-code for Malayalam.

the same as the existing ASCII code except for <*lc*>. As one can see, adding or deleting a new language is trivial in this scheme. It is a Universal Coding Scheme for computer and communication systems of today and tomorrow. This new character set coding scheme accommodates any variable number of languages without increasing the number of bits per byte (like going from 6-bits per byte to 8-bits per byte to accommodate lowercase letters of English) or using 16 bits per character for Chinese. In India, a coding scheme called ISCII (Indian Script Code for Information Interchange) like ASCII code has been developed (Nag [1989]). It conforms to the ISO 8-bit recommendations according to which ASCII remains in the bottom 7-bit region, while the Indian script characters are accommodated in the top 96 codes. It is similar to the *p*-code outlined in this thesis. However, ISCII code is valid only for a set of Indian languages and English. It is not general enough to handle other languages in the world like the *p*-code described in this thesis. Moreover, an ISCII code does not have provision to specify language code.

Graphics characters and Braille script could be easily accommodated by treating them as separate languages and assigning language codes for them.

8.4 Protocol Changes

Major well known data communication protocols are similar to the seven layer Open System Interconnect (OSI) model as shown in Figure 8.6 (Tanenbaum [1981]).

Layer	ISO	ARPANET	SNA	DECNET
7	Application	User	End user	Application
6	Presentation	Telnet, FTP	NAU services	
5	Session	(None)	Data flow control	(None)
4	Transport	Host-host	Transmission control	Network Services
3	Network	Source to dest. IMP	Path control	Transport
2	Data link	IMP-IMP	Data link control	Data link control
1	Physical	Physical	Physical	Physical

ISO	: International Standards Organization
ARPANET	: Advanced Research Project Agency Network
SNA	: System Network Architecture
DECNET	: Digital Equipment Corporation Network

Figure 8.6 Layers of major protocols.

OSI is gaining grounds to become *the standard* international protocol for data communication. Out of the seven layers, language has profound influence on the presentation and application layers. In order for a document to be received and handled by a destination host, it should have the capability to handle the attributes of the document to be transmitted. Therefore, it is essential that before transmitting a document, the end-to-end protocol should exchange the *capability list* containing the attributes of that document. This exchange could be performed as part of establishing the session between hosts or by other suitable means. Currently, provisions exist in certain protocols such as *telnet* to exchange 'options' between hosts. This feature could be generalized to exchange options and attributes.

There are two possible cases to be considered in exchanging the capability list for document transfer:

1. for storage
2. for display/printing

In the first case, a document may be transferred from one system (host) to another for temporary or permanent storage. For example, a large host may be used as a network file server by smaller systems like personal computers and workstations. In this case, the document will not be used for display/printing purposes at the host. Hence, even if the host does not have the capability to handle the document attributes, the document can be transferred to the host, stored and retrieved. In this case, it is not necessary to exchange the capability list; the document is treated like a binary file.

In the second case, since the receiving host will be using the document for display/printing, it is essential to exchange the capability list between the sending and receiving hosts. If the receiving host does not have the ability to handle the attributes of the document, it may not be able to display/print it properly.

In either case, the network switching elements called IMPs (Interface Message Processors in ARPANET terminology) need not have the capability to handle the document attributes since they only store the document temporarily in their databases, if necessary; also the content of the document is used as data in the packets being transferred by the network. The network never analyses the data in a packet; it is treated as either a bit or byte stream.

8.4.1 NFS Protocol Enhancement

NFS (Network File System) allows a user to extend the local host file system to a remote file system transparently by mounting the remote file system onto a local directory. Once mounted, remote files in the remote host can be accessed as if they were files on the local disk. Whenever a remote file is accessed, the NFS protocol is used to access the remote file. As an illustration consider that a user wants to edit a remote file *rf* mounted over the directory "*/remote*" using a terminal. Let us say a user types

edit /remote/rf

on the terminal.

In this case, the terminal used to edit the file has capabilities to handle certain attributes. Let,

$$C = \{c_1, c_2, \dots, c_n\}$$

be the capabilities of the terminal, where c_i is an attribute that the terminal is capable of handling. Let,

$$F = \{f_1, f_2, \dots, f_n\}$$

be the attributes of the file *rf*. If $C \subset F$, the terminal will not be able to display all the characters in the file properly. Under this circumstance, the user should be notified about this fact and given a chance to either continue or terminate editing. This way unnecessary reading of the file could be avoided, thus reducing network traffic, the local resources and the user time. It also facilitates the user to know the type of file being edited and the possible effect of it.

Currently, no provision exists in the NFS protocol to get just the attributes of a file. NFS protocol could be enhanced by adding a new procedure to the list of existing NFS procedures to get the attributes of a file. Model include file, the server module and the client module needed are given in Figures 8.7 to 8.9 respectively. NFSPROG is the NFS program number as it exists today, NFSVERS is the version

```
 /*
  * multi.h
  */

#define NFSPROG ((ulong)100003)
#define NFSVERS ((ulong)2)
#define NFSPROC_GET_ATTRIBUTES ((ulong)100)
extern char **clnt_nfsproc_get_attributes();
extern char **svc_nfsproc_get_attributes();
```

Figure 8.7 NFS client and server include file.

```
  /*
   *  multi_svc.c
   */

#include <stdio.h>
#include <string.h>
#include <rpc/rpc.h>
#define xdr_uint xdr_u_int
#define xdr_ushort xdr_u_short
#define xdr_ulong xdr_u_long
#define xdr_uchar xdr_u_char
#include "multi.h"

static void multprog();

main()
{
    SVCXPRT *transp;

    pmap_unset(MULTPROG, MULTVERS);

    transp = svcudp_create(RPC_ANYSOCK);
    if (transp == NULL) {
        fprintf(stderr, "cannot create udp service.\n");
        exit(1);
    }
    if (!svc_register(transp, MULTPROG, MULTVERS,
                            multprog, IPPROTO_UDP)) {
      fprintf(stderr,
            "unable to register (MULTPROG, MULTVERS, udp).\n");
      exit(1);
    }

    svc_run();
    fprintf(stderr, "svc_run returned\n");
    exit(1);
 }
```

Figure 8.8 NFS server module.

```
static void
multprog(rqstp, transp)
    struct svc_req *rqstp;
    SVCXPRT *transp;
{
    char *argument;
    char *result;
    bool_t (*xdr_argument)(), (*xdr_result)();
    char *(*local)();

    switch (rqstp->rq_proc) {
    case NULLPROC:
        svc_sendreply(transp, xdr_void, NULL);
        return;

    case NFSPROC_GET_ATTRIBUTES:
        xdr_argument = xdr_wrapstring;
        xdr_result = xdr_wrapstring;
        local = (char *(*)()) nfsproc_get_attributes_2;
        break;

    default:
        svcerr_noproc(transp);
        return;
    }
    memset(&argument, '\0', sizeof(argument));
    if (!svc_getargs(transp, xdr_argument, &argument)) {
        svcerr_decode(transp);
        return;
    }
    result = (*local)(&argument, rqstp);
    if (result != NULL &&
       !svc_sendreply(transp, xdr_result, result)) {
        svcerr_systemerr(transp);
    }
    if (!svc_freeargs(transp, xdr_argument, &argument)) {
        fprintf(stderr, "unable to free arguments\n");
        exit(1);
    }
}
```

Figure 8.8 NFS server module (contd.)

```
char **svc_nfsproc_get_attributes(argp, rqstp)
      char  **argp;
      struct svc_req   *rqstp;

{
       static  char  buffer[];

   verify_authentication;
   if(open_file(argp) = = SUCCESSFUL)  {
      read_attributes(buffer);
      return(&buffer[0]);
   }
   else
      return(NULL);
}
```

Figure 8.8 NFS server module (contd.)

```
    /*
     *   multi_clnt.c
     */

#include <rpc/rpc.h>
#define xdr_uint xdr_u_int
#define xdr_ushort xdr_u_short
#define xdr_ulong xdr_u_long
#define xdr_uchar xdr_u_char
#include <sys/fs/nfs/time.h>
#include "multi.h"

static struct timeval TIMEOUT = { 25, 0 };


multi_edit(remotefile)
   char *remotefile;
{
         char  *remotehost;
         CLIENT  *clnt;
         char *cap, *fattr;

   cap = get_terminal_capabilites();
   remotehost = get_remotehost();
   clnt = clntudp_create(remotehost);
   fattr = *clnt_nfsproc_get_attributes(&remotefile, clnt);
   if(cap ⊇ fattr)
     edit(remotefile);
   else
     display(cap, attr);
}
```

Figure 8.9 NFS client module.

```
char **
clnt_nfsproc_get_attributes(argp, clnt)
    char **argp;
    CLIENT *clnt;
{
    static char *res;

    memset(&res, '\0', sizeof(res));
    if (clnt_call(clnt, NFSPROC_GET_ATTRIBUTES, xdr_wrapstring,
          argp, xdr_wrapstring, &res, TIMEOUT) != RPC_SUCCESS) {
        return (NULL);
    }
    return (&res);
}
```

Figure 8.9 NFS client module (contd.).

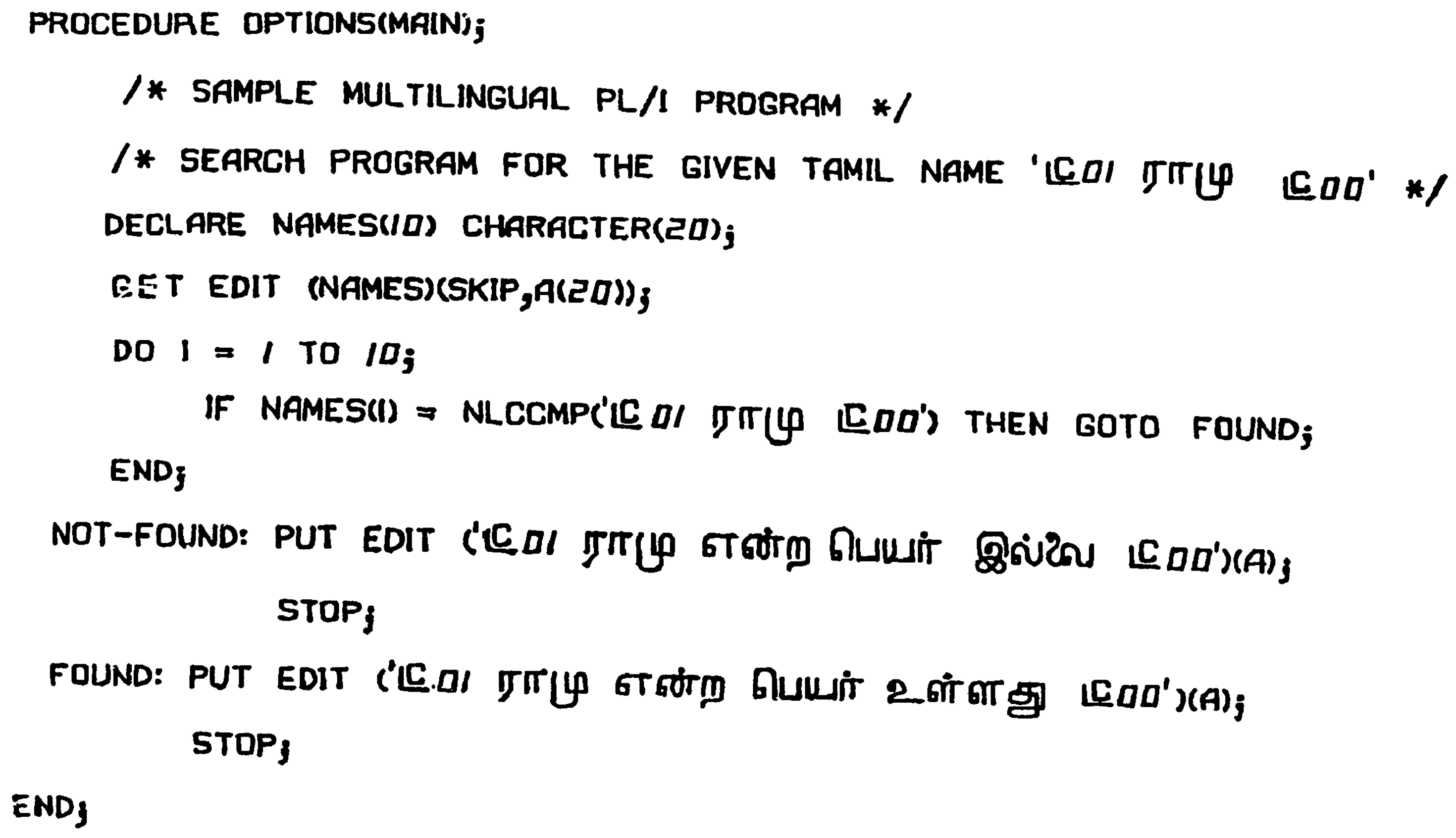
number and NFSPROC_GET_ATTRIBUTES is the new server procedure number that will send the attributes of a given file to the client. The client invokes the procedure using the RPC (Remote Procedure Call) procedure *clnt_call* as shown in the *clnt_nfsproc_get_attributes* procedure. When the RPC request is received by the NFS server on the remote host, it executes the service procedure *multprog* which in turn calls the *svc_nfsproc_get_attributes procedure* to read the attributes of the given file. The RPC procedure *svc_sendreply* sends the result (attributes, if file exist) to the client.

The client side editor program will compare the capabilities of the terminal and the attributes of the file. If the set of terminal capabilities is a super set of the set of file attributes, it will edit the file, else it will inform the user that the terminal is not capable of handling all the attributes of the file.

8.5 Multilingual Programming Interface

For developing multilingual CCSs, it is essential to provide programming language interfaces capable of handling multilingual texts. Work has been done to develop programming languages in other than English (Proc. LICBIS [1978]). However, these developments are individualistic in nature. In other words, they cannot handle multiple languages. Moreover, developing individual language processors such as compilers, etc., one for each language is expensive and time consuming.

A new scheme has been developed to specify multilingual texts in the existing English based programming languages using the coding scheme outlined earlier. In this scheme, the non-English texts are specified as string constants. The language code for the text is specified as part of the string using the language code described earlier. A sample computer print out of a PL/1 program written in English with Tamil strings is shown in Figure 8.10. The advantage of this method is that programming can be done using any existing English based programming language which has the capability to handle character strings. As long as the compiler treats a character string as a 7 or 8-bit code without looking into the linguistic aspect, the non-English texts could be specified as strings constant (by enclosing them in quotes, for example). This method, practically, needs no change in the existing compilers. It only needs the multilingual I/O devices described in the previous chapters.



```
PROCEDURE OPTIONS(MAIN);
     /* SAMPLE MULTILINGUAL PL/1 PROGRAM */
     /* SEARCH PROGRAM FOR THE GIVEN TAMIL NAME 'LC01 ராமு  LC00' */
     DECLARE NAMES(10) CHARACTER(20);
     GET EDIT (NAMES)(SKIP,A(20));
     DO I = 1 TO 10;
          IF NAMES(I) = NLCCMP('LC01 ராமு LC00') THEN GOTO FOUND;
     END;
  NOT-FOUND: PUT EDIT ('LC01 ராமு என்ற பெயர் இல்லை LC00')(A);
             STOP;
  FOUND: PUT EDIT ('LC01 ராமு என்ற பெயர் உள்ளது LC00')(A);
         STOP;
END;
```

Figure 8.10 Multilingual PL/1 program.

9. CONCLUSION AND FUTURE RESEARCH

9.1 Conclusion

The impact of Natural languages on Computing and Communication Systems (CCSs) have been pointed out. The need and the lack of a unified solution to handle and interchange information in any (variable) number of scripts and languages have been pointed out. In this research, a unified approach based on the symbolization and multilingual text representation schemes is proposed to fill this gap. This approach has the advantage of universality to handle major world languages and the flexibility to accommodate and omit languages with ease.

This research makes contributions to several areas of computer science. The design of a universal multilingual information interchange system with character reader and terminal has been described in detail. Simple line like primitives have been used as features for character recognition. The feature extractor can handle variations in size and shape of the characters. However, it is not efficient to extract curved primitives since it is sensitive to distortions. Errors in the DAG affect the recognition result. The classifier assumes that the characters are isolated and hence segmentation has been performed manually wherever needed. Testing the recognizer with Tamil and Malayalam characters gave good results.

Stage II of the two stage classifier developed for this research is language dependent and Stage I is pattern dependent. The pattern in the knowledge base and the CRT must be changed for different languages. However the structural feature extraction method is very general and it could also be u[illegible]ner languages and applications other than character recognition.

The multilingual terminal design approach developed is systematic. The computer-aided interactive method proposed is well suited for the determination of the dot matrix size of the characters or symbols. The optimality criteria based on the smallest and legible size depends on the individual as well as the equipment used. Hence, the nearly optimal sizes computed were to certain extent subjective. It is likely that they may vary slightly under other conditions. The new specialized knowledge based thinning method developed for automatic dot matrix character fonts reduces time and labor. This method could also be used to obtain the dot matrix of any pattern.

The iterative method of determining the most distinct character set based on their distances and information content provides a scientific method to evaluate different dot matrix character styles for their effectiveness, legibility and ergonomic properties. Even though Tamil symbols were used for evaluation, the method is not limited to Tamil characters. It could be used to evaluate dot matrix characters of other languages as well as other binary patterns. The undesirable models were eliminated iteratively using the majority elimination rules because the examination of the seven quantitative measurements revealed that the desirability of models indicated by any two measurements was not always identical.

Of the two symbol based keyboard designs, the *g*-symbol keyboards are more suitable for human interface compared to *p*-symbol keyboards since they resemble the existing keyboards displaying the graphemes of the characters. However, with the current technology, changing the display of *g*-symbols with the change of language is not simple. The *p*-symbols are better suited for internal representation than the *g*-symbols because they can collate properly on sorting. It is possible to convert valid sequences from one to the other since the character representation is unique in both cases. Hence, it is possible to use *g*-symbols for human interface and *p*-symbols for internal representation. The collection of characters to obtain *p*-symbol frequencies for Malayalam is moderate. Collection of more characters from non-text books may result in changes in frequencies of some of the symbols and better placement of the symbols. As one can see the algorithms for generating Tamil and Malayalam characters (in general Indian language characters) were more complicated than English. In certain cases, like Malayalam, the generator is context dependent. The multilingual terminal has the flexibility to handle different languages. It can be used not only for Indian languages but also for other languages.

The proposed changes in data communication such as document structure, coding scheme and appropriate protocol changes are simple but basic. Character sets can be added and deleted easily. Braille script for the blind and even non-natural language character sets like graphics characters can be added. Practically, it can accommodate all the major languages of the world with the existing 8 bit code with each character set having 255 ccde combinations. Since the proposed changes have global impact, international standards committees must take cooperative efforts to make uniform changes.

When the proposed system becomes a large scale commercial reality, it will bring unification among systems and help the world community; in particular, nations like India with fifteen different languages and ten different complex scripts will benefit the most. The system can be used for various applications such as multilingual telex, producing computerized reservation lists, utility billing, telephone directories, wordprocessing, teaching scripts, etc., in regional languages which directly affect the society. The experience from this research indicated that research like this needs proficiency in the language under consideration.

Nowadays CCS developers are paying more attention to international applicability of their products with respect to linguistic content such as ability to produce messages to the users in their native languages. Upcoming and future releases of the most popular and widely used operating system UNIX, will have provision to accommodate character sets which use 16 bits per character such as Japanese and Chinese. It clearly shows the direction and the importance of CCSs to be capable of handling multiple languages.

The future CCSs will be and must be multilingual in nature to be applicable and tradable in any part of the globe. The number of languages supported by individual CCS will depend on the place and the application; it will be, in most cases, a configurable parameter and the character set will be downloadable.

It is our contention that only multilingual CCSs, like the one proposed in this research, will be flexible, universally applicable, cost effective and globally tradable in the future; only such CCSs will cross the linguistic boundaries of the nations and bring harmony and more interaction among the information processing communities and the societies in the world.

9.2 Future Research

Several areas of computer science have been covered in this thesis. There is room for improvement in almost every area. The research reported here can be extended further in several directions:

(a) The algorithms and methodologies developed for a multilingual character reader can be realized in a special purpose parallel hardware-firmware architecture. Applicability of such architectures in pattern recognition is reviewed in (Krishnamoorthy

[1987], Siddiqui [1985]).

(b) Automatic character segmentation could be incorporated.

(c) Alternate feature extraction methodologies such as the one reported in Chapter 5 could be investigated to eliminate some of the deficiencies in extracting features from thinned patterns.

(d) More features such as loop, etc., and additional classification stages using statistical methods could be added to the classifier to improve the recognition rate.

(e) The multilingual terminal can be constructed in hardware using the algorithms developed in the research.

(f) Efforts must be devoted to coordinate with the standards organizations to incorporate the scheme proposed in this research into the standard communication protocols.

(g) Existing communication protocols and document structures have to be enhanced to incorporate the proposed scheme.

(h) Several more world languages and scripts, in particular African languages, must be examined for symbolization, character generation and for any special needs and peculiarities.

REFERENCES

AHME-86 Ahmed, P., (1986), Computer Recognition of Totally Unconstrained Handwritten ZIP Codes, Ph.D. Dissertation, Dept. of Computer Science, Concordia University, Montreal, Canada.

AKAM-83 Akamatsu, S., Kawatani, T., (1983), "Hierarchical classification of handprinted Kanji by using density feature and configuration feature", Proc. of 1983 Internat. Conf. on Text Processing with a Large Character Set, Chinese Language Computer Society (USA), pp. 175-180.

ALA-77 ALA-LC Romanization Tables (1977), Library of Congress Cataloging Service, Bulletins, pp. 118-120.

ALI-77a Ali, F. and Pavlidis, T., (1977), "Computer recognition of handwritten numerals by polygonal approximations", IEEE Trans. Syst. Man Cybernet., Vol. SMC-7, pp. 537-541.

ALI-77b Ali, F. and Pavlidis, T., (1977), "Description and recognition of handwritten numerals", Proc. Workshop Picture Data Processing and Management, pp. 26-32.

ALWA-89a Alwar, N., Raman, S., Shanthi, A., Venkataraman, R., Venkata Subramaniam, (1989), "A multipurpose multilingual package for Indian languages", Proc. of the Regional Workshop on Computer Processing of Asian Languages (CPAL), Sept. 26-28, Bangkok, Thailand, pp. 18-26.

ALWA-89b Alwar, N. and Raman, S., (1989), "A natural language generator for Hindi", Proc. of the Regional Workshop on Computer Processing of Asian Languages (CPAL), Sept. 26-28, Bangkok, Thailand, pp. 102-108.

AMIN-84 Amin, A., Masini, G., and Haton, J. P., (1984), "Recognition of handwritten Arabic words and sentences", Proc. 7th Int. J. Conf. on Pattern Recognition, Montreal, Canada, pp. 1055-1057.

AMIN-86 Amin, A. and Masini, G., (1986), "Machine recognition of multifont printed Arabic texts", Proc. 8th Inter. Conf. on Pattern Recognition, Paris, France, pp. 392-395.

AMIN-88 Amin, A., (1988), "OCR of Arabic texts", Lecture Notes in Computer

Science, G. Goos and J. Hartmanis (Ed.), Springer-Verlag, New York, pp. 616-625.

ANDE-69 Anderson, P. L., (1969), "Optical character recognition-a survey", Datamation, pp. 43-48.

ARAK-83 Arakawa, H., (1983), "On-line recognition of handwritten characters, alphanumerics, Hiragana, Katakana, Kanji", Pattern Recognition, Vol. 16, No. 1, pp. 9-22.

BALA-90 Balasubramanian, K., (1990), "Microprocessor based multiligual character display", IEEE Trans. Consumer Electronics, Vol. 36, No. 4, pp. 933-938.

BANN-88 Banno, K., Kawamata, T., Kobayashi, K., and Nambu, H., (1988), "Text recognition system for Japanese documents", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 176-180.

BANS-88 Bansal, V. S., and Metha, S., S., (1988), "Computer based recognition of hand-written Devanagari Characters using possibility theory", Proc. of Computer Society of India, pp. 706.2.

BARR-86 Barr, P. C. and Krishnamoorthy, S. G., (1986), "Architecture of a fiber optics based distributed information network FORTIS: local area network", Proc. Fall Joint Computer Conference, Informat, Dallas, pp. 390-399.

BECK-83 Becker, J.D., (1983), "User friendly design for Japanese typing", Proc. of 1983 Internat. Conf. on Text Processing with a Large Character Set, Chinese Language Computer Society (USA), pp. 231-234.

BERT-79 Berthod, M., and Maroy, J. P., (1979), "Learning in syntactic recognition of symbols drawn on a graphic tablet", Computer Grahics and Image Processing, Vol. 9, No. 2, pp. 166-182.

BERT-82 Berthod, M., (1982), "On-line recognition of cursive writing", Computer Analysis and Perception: Vol. 1, Visual Signals, (C. Y. Suen, and R. De Mori, eds.), pp.55-81, CRC Press, Boca Raton, Florida.

BOZI-84 Bozinovic, R., and Srihari, S. N., (1984), "Knowledge based cursive script interpretation", Proc. Internat. Conf. Pattern Recognition, pp. 774-776.

BOZI-89 Bozinovic, R., and Srihari, S. N., (1990), "Off-line cursive script word recognition", IEEE Trans. Pat. Analysis and Machine Intelligence, pp. 68-83.

CCIT-81 CCITT Recommendation S63.

CHAN-88 Chang, P. Sang-Lei, H., and Muller, V., (1988), "Recognition of handprinted Chinese characters by stroke order codes", Proc. of Internat. Conf. on Computer Processing of Chinese and Oriental Languages, Aug. 29 - Sept. 1, Toronto, Canada, pp. 54-57.

CHEN-86 Cheng, F. H., and Hsu, W. H., (1986), "Three stroke extraction methods for recognition of handwritten Chinese characters", Proc. of Internat. Conference on Chinese Computing, pp. 191-195.

CHEN-87a Cheng, K. Y. and Yu, F. K. (1987), "On disambiguous phonetic input", Int. Conf. on Chinese and Oriental Language Computing, Chicago, pp. 2-8.

CHEN-87b Cheng, F. H. and Hsu, W. H., (1987), "Radical extraction by background thinning method for handwritten Chinese characters", Int. Conf. on Chinese and Oriental Language Computing, Chicago, pp. 175-182.

CHOM-56 Chomsky, Noam, (1956), "Three models for the description of language", PGIT, Vol. 2, No. 3, pp.113-124.

CIAR-88 Ciardiello, G., Scafuro, G., Degrandi, M. T., Spada, M. R., and Roccotelli, M. P., (1988), "An experimental system for office document handling and text recognition", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 739-743.

DATT-80 Dattatreya, G. R. and Sarma, V. V. S., (1980), "The decision tree design for pattern recognition including feature measurement cost", Proc. 5th Int. Conf. Pattern Recognition, pp. 1212-1214.

DESS-80 Dessimoz, J. D. (1980), Specialized Edge-Trackers for Contour Extraction and Line Thinning, Signal Processing, Vol. 2, pp. 148.

DING-88 Ding, X., Wu, Y., and Zhu, X., (1988), "Recognition of multi-font printed Chinese characters by structure analysis of strokes", Proc. of Internat. Conf. on Computer Processing of Chinese and Oriental Languages, Aug. 29 - Sept. 1, Toronto, Canada, pp. 123-125.

DONG-86 Dong, C. Y., Wang, J. K., Wan, J. C., and Hu, Q. S., (1986), "A study of the coding of Chinese characters", Proc. of Internat. Conference on Chinese Computing, pp. 405-406.

DONG-88a Dong, H., Wu, Y., and Ding, X., (1988), "An ARG representation for Chinese characters and a radical extraction based on the representation", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 920-922.

DONG-88b Dong, H., Wu, Y., and Ding, X., (1988), "A multifont Chinese character recognition method based on the attributed relational graph (ARG) representation", Proc. of Internat. Conf. on Computer Processing of Chinese and Oriental Languages, Aug. 29 - Sept. 1, Toronto, Canada, pp. 156-160.

DOWN-88 Downton, A. C., Kabir, E., and Guillevic, D., (1988), "Syntactic and contextual post-processing of handwritten addresses for optical character recognition", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 1072-1076.

DUEE-80 Dueer, B., Haettich, W., Tropf, H. and Winkler, G., (1980), "A combination of statistical and syntactical pattern recognition applied to classification of unconstrained handwritten numerals", Pattern Recognition, Vol. 12, pp. 189-199.

ETHI-88 Ethiaraj, G., and Ethiyajeevakaruna, S. W., (1988), "Computer-based Publishing in Tamil", Proc. of 23rd Annual Convention of the Computer Society of India, Jan. 6-9, Madras, India.

FARI-83 Faris, B. and Behrouz, P., (1983), "Approximation of multipath planar shapes in pattern recognition", Int. Journal of Computer and Information Sci., Vol. 12, No. 2, pp. 99-110.

FU-74 Fu, K. S., (1974), Syntactic Methods in Pattern Recognition, Academic Press, New York and London.

FU-82 Fu, K. S., (1982), Syntactic Pattern Recognition and Applications, Prentice-Hall, Englewood Cliffs, NJ.

FU-83 Fu, K. S., (1983), "A step towards unification of syntactic and statistical pattern recognition", IEEE Trans. Pattern Anal. Mach. Intell., Vol. PAMI-5, pp. 200-205.

GARE-79 Garey M. R. and Johnson, D. S., Computers and Interactability: A Guide to the Theory of NP-Completeness, W. H. Freeman and Company, San Francisco.

GHOS-84 Ghosh, P. K., (1984), "Basic design issues in a multilingual type font design and typesetting workstation", IETE Vol. 30, No. 6.

GILY-70 Gilyarevsky, R. S. and Grivnin, V. S. (1970), "Languages Identification Guide", "Nauka" Publishing House, Central Department of Oriental Literature, Moscow.

GONG-88 Gong, W. -X., and Bertrand, G., (1988), "A fast skeletonization algorithm using derived grids", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 776-778.

GONA-78 Gonzalez, R. C. and Thomason, M. G., (1978), Syntactic Pattern Recognition: An Introduction, Addison-Wesley, Reading, Mass. USA.

GOOS-88 Goos, G., and Hartmanis, J., (1988), Lecture Notes in Computer Science: Pattern Recognition, Springer-Verlag, New York.

GUTK-87 Gutknwecht, C., and Gerlinger, F., (1987), "An approach to multilingual texts", Research in Word Processing Newsletter, South Dakota School of Mines and Technology, Vol. 5, No. 8, pp. 3-20.

HAGI-83 Hagita, N., Naito, S., Masuda, I., (1983), "Handprinted Kanji characters recognition based on pattern matching method", Proc. of 1983 Internat. Conf. on Text Processing with a Large Character Set, Chinese Language Computer Society (USA), pp. 169-174.

HELD-83 Held, G., (1983), Data Compression: Techniques and Applications: Hardware and Software Considerations, Wiley, New York.

HILD-69 Hilditch, C. J. (1969), "Linear Skeletons from Square Cupboards, Machine Intelligence", Vol. 4, edited by B. Meltzer and D. Michie, American Elsevier, New York, pp. 403-420.

HOLD-88 Holder, S., and Dengler, J., (1988), "Font - and size invariant character recognition with grey value image features", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 252-254.

HOLT-74 Holt, A. W., (1974), "Algorithm for a low-cost hand print reader", Comput.

Design, pp. 85-89.

HSIE-90 Hsieh, A. J., Kung, S. J., Shiau, S. L., Kao, M. C., and Chen, J. W., (1990), "An experimental system for stroke-number free on-line Chinese character recognition", Frontiers in Handwriting recognition, C. Y. Suen (Ed.), Centre for Pattern Recognition & Machine Intelligence, Concordia University, Montreal, Canada, pp. 73-86.

HSU-85 Hsu, W. H., and Cheng, F. H., (1985), "Recognition of handwritten Chinese characters by stroke structure analysis method", Proc. of 1985 Internat. Conference on Chinese Computing, San Francisco, Feb. 26-28, pp. H-3.

HU-82 Hu, C. H., Lin, P., Ling, H. Y. and Wu, F. F., (1982), "A handwritten numeral recognition machine for automatic mail-sorting", Signal Process. Theor. Applic., pp. 195-198.

HUAN-86 Huang, J. S. and Chuang, K., (1986), "Heuristic approach to handwritten numeral recognition", Pattern Recognition, Vol. 19, No. 1, pp. 15-20.

HULL-88 Hull, J. J., Srihari, S. N., Cohen, E., Kuan, L., Cullen, P., and Palumbo, P., (1988), "A blackboard-based approach to handwritten ZIP code recognition". Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 111-113.

HUNG-87 Hung, W. W., (1987), "Chinese characters encoded in stroke sequences", Int. Conf. on Chinese and Oriental Language Computing, Chicago, pp. 130-133.

ISO-82 ISO/TC97/SC18/WG3 N84, ECMA/TC29/82/54 : Office document interchange formats, Second Working draft for an ECMA standard ('82-10).

IZAK-83 Izaki, Y., Nakanishi, M., Kabuyama, Y., Hai, T., (1983), "Postprocessing of handwritten Kanji character recognition", Proc. of 1983 Internat. Conf. on Text Processing with a Large Character Set, Chinese Language Computer Society (USA), pp. 197-202.

JIAN-88 Jian-long, T., and Wenhao, S., (1988), "An approach to stroke extraction and radical classification of handwritten Chinese characters", Proc. of Internat. Conf. on Computer Processing of Chinese and Oriental Languages, Aug. 29 - Sept. 1, Toronto, Canada, pp. 104-107.

JOSH-88 Joshi, W. S., (1988), "Text processing in Devanagari", Proc. of 23rd Annual Convention of the Computer Society of India, Jan. 6-9, Madras, India.

KANA-72 Kanal, L. N. and Charndrasekaran, B., (1972), "On linguistic, statistical and mixed models for pattern recognition", Frontiers of Pattern Recognition, S. Watanabe (ed.), Academic Press, New York, p.163.

KRIS-78a Krishnamoorthy, S. G. and Isaac, J. R., (1978), "Input-output device based multilingual information handling system", in Proc. of Symposium of Linguistic Implications of Computer Based Information Handling Systems, Electronic Commission, New Delhi, India.

KRIS-78b Krishanmoorthy, S. G., (1978), "Use of Malayalam in computers", Proc. of Symposium on Linguistic Implications of Computer Based Information Systems, Electronics Commission, New Delhi, India.

KRIS-80a Krishnamoorthy, S. G. and Isaac, J. R., (1980), "Microprocessor-based Multilingual information handling system : Acquisition and representation", Computer Society of India, Annual Convention, Bombay.

KRIS-80b Krishnamoorthy, S. G., Isaac, J. R. and Bhavsar, V. C. (1980), "A Microprocessor Based Multilingual Terminal for Computerized Information Handling System", Journal of Computers and Humanities, Vol. 14, pp. 91-104.

KRIS-80c Krishnamoorthy, S. G., (1980), "A microprocessor based multilingual telex system with phonetically coded keyboarding", Proc. Pacific Telecommunications Conference, Honolulu, Hawaii, pp. 1C-19-1C-31.

KRIS-81 Krishnamoorthy, S. G. (1981), "A Processor-Based Multilingual Text Handling System and Its Applications", M. Comp. Sci. Thesis, Dept. of Computer Science, Concordia University, Montreal, Canada.

KRIS-83 Krishnamoorthy, S. G., Ahmed, P. and Suen, C. Y., (1983), "Computer-aided determination of nearly optimal dot-matrix size for display of Indian characters", Proc. Society for Information Display, Vol. 24, No. 3, pp. 271-279.

KRIS-86 Krishnamoorthy, S. G., Suen, C. Y. and Mutalik, P. (1986), "Computer Aided Determination of Optimal Dot Matrix Character Set For The Design of Display and Printing Devices for Indian Characters", Fifth Annual Internat. Phoenix Conference on Computers and Communications, Scottsdale, Arizona, pp. 554-559.

KRIS-87a Krishnamoorthy, S. G., Suen, C. Y. and Jesuraj, R., (1987), "Linguistic implications of data base systems", Proc. Inter. National Conf. on Chinese and Oriental Language Computing, Illinois Institute of Technology, Chicago, U.S.A.

KRIS-87b Krishnamoorthy, S. G., (1987), "Parallel and specialized architectures for image processing", Technical Report, Concordia University, Montreal.

KRIS-88 Krishnamoorthy, S. G. and Suen, C. Y., "Universal multilingual information interchange system", RIAO 88, M.I.T., Cambridge, pp. 781-809.

KRZY-90 Krzyzak, A., Dai, W., and Suen, C. Y., (1990), "Unconstrained handwritten character classification using modified backpropagation model", Frontiers in Handwriting recognition, C. Y. Suen (Ed.), Centre for Pattern Recognition & Machine Intelligence, Concordia University, Montreal, Canada, pp 155-166.

KUNG-83 Kung, L. Y., Ho, Y. K., Chou, C. L., (1983), "Chinese keyboard design using modified four-corner index and phonetic encoding method", Proc. of 1983 Internat. Conf. on Text Processing with a Large Character Set, Chinese Language Computer Society (USA), pp. 224-227.

KURA-88 Kurakake, S., (1988), "Predictive category learning for handwritten Kanji characters", Proc. of Internat. Conf. on Computer Processing of Chinese and Oriental Languages, Aug. 29 - Sept. 1, Toronto, Canada, pp. 90-95.

KUSH-82 Kushnir, M., Guo, B. L. and Matsumoto, K., (1982), "Recognition of handwritten Hebrew characters by the double inclusive matching method", IECE Japan, J65-D, pp. 1011-1017.

KUSH-83a Kushnir, M., Abe, K. and Matsumoto, K., (1983), "An application of Hough transform to the recognition of printed Hebrew characters", Pattern Recognition, Vol. 16, pp. 183-191.

KUSH-83b Kushnir, M., (1983), Machine Recognition of Hebrew characters, Ph.D dissertation, Shizuoka Univ.

KUSH-85 Kushnir, M., Abe, K. and Matsumoto, K., (1985), "Recognition of handprinted Hebrew characters using features selected in the Hough transform space", Pattern Recognition, Vol. 18, No [illegible], pp. 103-114.

LAI-81 Lai, M. T. Y. and Suen, C. Y., (1981), "Automatic recognition of characters by Fourier descriptors and boundary line encodings", Pattern Recognition, Vol. 14, No. 1-6, pp. 383-393.

LAM-88 Lam, L., and Suen, C. Y., (1988), "Structural classification and relaxation matching of totally unconstrained handwritten zip-code numbers.", Pattern Recognition, Vol. 21, pp. 19-31.

LAM-90 Lam, L., and Suen, C. Y., (1990), "A dynamic shape preserving thinning algorithm", Tech. Report, CENPARMI, Concordia University, Montreal, Canada.

LATU-78 Laturkar, K. P. and Sinha, R. M. K. (1978), "Devanagari script composition from phonetically coued symbol strings", Proc. Symp. Linguistic Implications of Computer Based Information Systems, Electronic Commission, New Delhi, India.

LECU-90 Le Cun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., Jackel, L. D., and Baird, H. S., (1990), "Constrained neural network for unconstrained handwritten digit recognition", Frontiers in Handwriting recognition, C. Y. Suen (Ed.), Centre for Pattern Recognition & Machine Intelligence, Concordia University, Montreal, Canada, pp. 145-154.

LEGA-90 Legault, R., Suen, C. Y., and Nadal, C., (1990), "Classification of confusing handwritten numerals by human subjects", Frontiers in Handwriting recognition, C. Y. Suen (Ed.), Centre for Pattern Recognition & Machine Intelligence, Concordia University, Montreal, Canada, pp. 181-194.

LEOW-86 Leow, W. K., (1986), "Syntactic approach to Chinese character recognition", Proc. of Internat. Conference on Chinese Computing, pp. 124-130.

LEOW-88 Leow, W. K., Hsu, L. S., and Lua, K. T. (1988), "Recognition of handprinted Chinese characters with a heuristic parser", Proc. of Internat. Conf. on Computer Processing of Chinese and Oriental Languages, Aug. 29 - Sept. 1, Toronto, Canada, pp. 23-27.

LEVE-88 Leveridge, P. C., and Leedham, C. G., (1988), "Experiments with an n-tuple recognizer for fast "first try" recognition of unconstrained handwritten symbols", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 905-907.

LIFC-75 Lifco Tamil-Tamil-English Dictionary (1975), The Little Flower Company, Madras, India.

LIU-88 Liu, Y. J., and Tai, J. W., (1988), "A structural approach to on-line Chinese character recognition", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 808-810.

LOH-88 Loh, S., Chan, C., and Chan, S., (1988), "On-line recognition of handwritten Chinese characters", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov.14-17, pp. 808-810.

LINQ-88 Linquan, W., (1988), "Recognition of handprinted Chinese characters by outline direction and background density", Proc. of Internat. Conf. on Computer Processing of Chinese and Oriental Languages, Aug. 29 - Sept. 1, Toronto, Canada, pp. 39-43.

LU-90 Lu, S. W., and Suen, C. Y., (1990), "Hierarchical attributed graph representation and recognition of handwritten Chinese character recognition", Frontiers in Handwriting recognition, C. Y. Suen (Ed.), Centre for Pattern Recognition & Machine Intelligence, Concordia University, Montreal, Canada, pp. 87-100.

MANT-86 Mantas, J., (1986), "An overview of character recognition methodologies", Pattern Recognition, Vol. 19, No. 6, pp. 425-430.

MATH-87 Mathur, A., and Fowler, F., (1987), "Design of a dynamically reconfigurable keyboard", Proc. 1987 Inter. Conf. on Chinese and Oriental Language Computing, Chicago, June 15-17, pp. 20-23.

MCLA-68 McLaughlin, R. A. and Raviv, J., (1968), "Nth-order autocorrelations in pattern recognition", Inform. Contr., Vol. 12, pp. 121-142.

MINN-66 Minneman, M. J., (1966), "Handwritten character recognition employing topology, cross correlation and decision theory", IEEE Trans. Syst. Sci. Cybern., Vol. 2, pp. 89-96.

MONT-82 Montgomery, E. B. (1982), "Bringing manual input into the 20th century: new keyboard concept", IEEE Computer, 15(3), pp. 11-18.

MORA-89 Morasso, P., (1989), "Neural models of cursive script handwriting", International Joint Conf. on Neural Networks, Washington, D.C., June.

MURA-88 Murase, H., (1988), "Online recognition of free-format Japanese handwritings", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 1143-1147.

MUTH-89 Muthuvel, C., and Alwar, N., (1989), "A font generator for Indian languages", Proc. of the Regional Workshop on Computer Processing of Asian Languages (CPAL), Sept. 26-28, Bangkok, Thailand, pp. 154-159.

NADA-90 Nadal, C., Legault, R., and Suen, C. Y., (1990), "Complementary algorithms for the recognition of totally unconstrained handwritten numerals", Proc. 10th International Conf. on Pattern Recognition.

NAG-89 Nag, B., (1989), "Information technology for Indian scripts: Problems and prospects", Proc. of the Regional Workshop on Computer Processing of Asian Languages (CPAL), Sept. 26-28, Bangkok, Thailand.

NAGY-88 Nagy, G., (1988), "Chinese character recognition: A twenty-five year retrospective", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 163-167.

NARA-64 Narasimhan, R., (1964), "Labelling schemata and syntactic description of pictures", Information and Control, Vol. 7, p. 151.

NARA-66 Narasimhan, R., (1966), "Syntax directed interpretation of classes of pictures", Comm. of ACM, Vol. 9, p. 166.

NARA-69 Narasimhan, R., (1969), "On the description, generation and recognition of classes of pictures", Automatic Interpretation and Classification of Images (Ed. A. Grasselli), Academic Press, New York, pp. 1-42.

NARA-71a Narasimhan, R. and Reddy, V. S. N., (1971), "A syntax-aided recognition scheme for handprinted English letters", Pattern Recognition, Vol. 3, p. 345.

NARA-71b Narasimham, P. V. H. M. L., Prasada, B. and Rajaraman, V. (1971), "Code based keyboard for Indian languages", Journal of the Computer Society of India, 2(2), 33-37.

NISH-88 Nishino, F., Takao, T., (1988), "Post processing for Japanese document readers", Proc. of Internat. Conf. on Computer Processing of Chinese and Oriental Languages, Aug. 29 - Sept. 1, Toronto, Canada, pp. 166-169.

OGAW-82 Ogawa, H. and Taniguchi, K., (1982), "Thinning and stroke segmentation for handwritten Chinese character recognition", Pattern Recognition, Vol. 15, No. 4, pp. 299-308.

OTT-74 Ott, R., (1974), "On feature selection by means of principle axis transform and nonlinear classification", Proc. 2nd Int. Joint Conf. Pattern Recognition, pp. 220-222.

PAL-81 Pal, S. K. and King, R. A., (1981), "Image enhancement using smoothing with fuzzy sets", IEEE Trans. on Systems, Man, Cybern., Vol. SMC-11, No. 7, pp. 494-501.

PAL-83 Pal, S. K., King, R. A. and Hashim, A. A., (1983), "Automatic gray level thresholding through index of fuzziness and entropy", Pattern Recognition Letters, Vol. 1, pp. 141-146.

PARH-81 Parhami, B. and Taraghi, M., (1981), "Automatic recognition of printed Farsi text", Pattern Recognition, Vol. 14, No. 1-6, pp. 395-403.

PING-88 Ping, C. K., and Cheung, Y. S., (1988), "Fuzzy-attribute graph and its application to Chinese character recognition", Proc. of Internat. Conf. on Computer Processing of Chinese and Oriental Languages, Aug. 29 - Sept. 1, Toronto, Canada, pp. 136-140.

RADH-83 Radhakrishnan, T., Atwood, J. W. and Krishnamoorthy, S. G. (1983), "A multi-lingual input/output device for Indian scripts", Journal of Man Studies, Vol. 19, pp. 137-146.

RAJA-77 Rajasedaran, S. N. S. and Deekshatulu, B. L., (1977), "Recognition of printed Telugu characters", Computer Graphics and Image Processing, Vol. 6, p. 335.

ROMA-73 Romanization Tables (1973), Library of Congress Cataloging Bulletin.

ROSE-81 Rosenfeld, A. and Smith, R. C., (1981), "Thresholding using relaxation", IEEE Trans. on Pattern Analysis and Machine Intelligence, Vol. PAMI-3, No. 5, pp. 598-606.

SABO-90 Sabourin, R., and Plamondon, R., (1990), "Progress in the field of automatic handwritten written signature verification systems using gray-level images", Frontiers in Handwriting recognition, C. Y. Suen (Ed.), Centre for Pattern Recognition & Machine Intelligence, Concordia

University, Montreal, Canada, pp. 1-12.

SCHA-82 Schantz, H. F., (1982), "The history of OCR optical character recognition", Recognition Technologies Users Association (RTUA), USA.

SCHU-82 Schurmann, J., (1982), "Reading machines", Proc. 6th Int. J. Conf. on Pattern Recognition, Munich, pp. 1031-1044.

SELV-88 Selvarajagopal, E., and Ethiyajeevakaruna, S. W., (1988), "Implementation of Indian languages on computers: A case study of Tamil", Proc. of 23rd Annual Convention of the Computer Society of India, Jan. 6-9, Madras, India.

SETH-77 Sethi, I. K. and Chatterjee, B., (1977), "Machine recognition of constrained handprinted Devanagari", Pattern Recognition, Vol. 9, p. 69.

SHIA-80 Shiau, C., and Suen, C. Y., (1980), "An iterative technique of selecting an optimal 5x7 matrix character set for display in computer output systems", Proc. of the Society for Information Display, Vol. 21, No. 1, pp. 9-15.

SHRI-85 Shridhar, M. and Badreldin, A., (1985), "A high accuracy syntactic recognition algorithm for handwritten numerals", IEEE Trans. Syst. Man Cybernet., Vol. SMC-15, pp. 152-158.

SHRI-86 Shridhar, M. and Badreldin, A., 'Recognition of isolated and simply connected handwritten numerals", Pattern Recognition, Vol. 19, No. 1, pp. 1-12.

SHYU-88 Shyu, I., Jeng, S. C., Huang, Y. S., Lin, W. W., Tu, L. T., and Chen, Y. H., (1988), "Design of a decision tree and its application to large-set printed Chinese character recognition", Proc. of Internat. Conf. on Computer Processing of Chinese and Oriental Languages, Aug. 29 - Sept. 1, Toronto, Canada, pp. 126-130.

SID-80 (1980) Special Issue devoted to research on display of matrix characters, Proceeding of the Society for Information Display, Vol. 21, No. 1.

SIDD-85 Siddiqui, K. J. and Ahmed, P., (1985), "Special architecture for optical character recognition and image processing", Technical Report Cat. No. HF5001C6+1984, No. 46, Concordia University, Montreal.

SIMO-90 Simon, J. C., and Baret, O., (1990), "Handwriting recognition as an application of regularities and singularities in line pictures", Frontiers in Handwriting recognition, C. Y. Suen (Ed.), Centre for Pattern Recognition & Machine Intelligence, Concordia University, Montreal, Canada, pp. 23-38.

SINH-73 Sinha, R. M. K., (1973), A Syntactic Pattern Analysis System and its Application to Devanagari Script Recognition, Ph.D. Thesis, Dept. of Electrical Engg., Indian Institute of Technology, Kanpur, India.

SINH-84 Sinha, R. M. K., (1984), "A knowledge-based script reader", Proc. 7th Int. J. Conf. on Pattern Recognition, Montreal, pp. 763-765.

SIRO-78 Gift Siromoney, Chandrasekaran, R. and Chandrasekaran, M., (1978), "Recognition of printed Tamil characters", Pattern Recognition, Vol. 10, p. 243-248.

SOM-77 Som, A. and Nath, A. K., (1977), "On some methods of sequential pattern recognition", ISI Symposium on Digital techniques and Pattern Recognition, Calcutta.

SONG-86 Song, H. Y., and Suen, C. Y., (1986), "A survey of Chinese character generators with a proposed new method", Proc. of Internat. Conference on Chinese Computing, pp. 421-428.

STEF-71 Stefanelli. R. and Rosenfeld, A. (1971), "Some parallel thinning algorithms for digital pictures", Journal of ACM, Vol. 18, No. 2, pp. 255-264.

STOR-88 Storer, J. A., (1988), Data Compression: Methods and Theory, Computer Science Press, Rockville, MD, USA.

STRI-89 Stringa, L., (1989), "Efficient classification of totally unconstrained handwritten numerals with a trainable multilayer network", Pattern Recognition Letters, Vol. 10, pp. 273-280.

STRI-90 Stringa, L., (1990), "A structural approach to automatic primitive extraction in hand-printed character recognition", Frontiers in Handwriting recognition, C. Y. Suen (Ed.), Centre for Pattern Recognition & Machine Intelligence, Concordia University, Montreal, Canada, pp. 65-72.

SUEN-77 Suen, C. Y. and Shillman, R. J., (1977), "Low error rate optical character recognition of unconstrained handprinted letters based on a model of human perception", IEEE Trans. Syst., Man, Cybern., Vol. 7, pp. 491-495.

SUEN-78 Suen, C. Y., (1978), "Advances in optical character recognition system", Canadian Computer Conference, Edmonton, Canada, pp. 263-268.

SUEN-80a Suen, C.Y., Berthod, M. and Mori, S., (1980), "Automatic recognition of handprinted characters - the state of the art', Proc. IEEE, Vol. 68, No. 4, pp. 469-487.

SUEN-80b Suen, C. Y., (1980), "Feature extraction in automatic recognition of handprinted characters", Signal Processing: Theories and Applications, M. Kunt and F. DeCoulon (editors), North-Holland Publishing Co., pp. 491-501.

SUEN-82a Suen, C. Y., (1982), "Distinctive features in automatic recognition of handprinted characters", Signal Processing, Vol. 4, pp. 193-207.

SUEN-82b Suen, C. Y., (1982), "The role of multi-directional loci and clustering in reliable recognition of characters", Proc. 6th Int. J. Conf. on Pattern Recognition, Munich, pp. 1020-1022.

SUEN-83 Suen, C. Y., (1983), "Computer recognition of Kanji characters", Proc. Intl. Conf. on Text Processing with a Large Character Set, pp. 429-435.

SUEN-86a Suen, C. Y., (1986), "Character recognition by computer and applications", Handbook of Pattern Recognition and Image Processing, Academic Press. pp. 569-585.

SUEN-86b Suen, C. Y., Tang, Y. Y., and Wang, Q. R., (1989), "Feature extraction in the recognition of Chinese characters printed in different fonts", Proc. of Internat. Conference on Chinese Computing, pp. 136-146.

SUEN-90a Suen, C. Y., (1990), Frontiers in Handwriting Recognition, Centre for Pattern Recognition & Machine Intelligence, Concordia University, Montreal, Canada.

SUEN-90b Suen, C. Y., Nadal, C., Mai, T. A., Lagault, R., and Lam, L., (1990), "Recognition of totally unconstrained handwritten numerals based on the concept of multiple experts", Frontiers in Handwriting recognition, C. Y. Suen (Ed.), Centre for Pattern Recognition & Machine Intelligence, Concordia University, Montreal, Canada, pp. 131-144.

SUZU-90 Suzuki, T., and Mori, S., (1990), "A thinning method based on cell structure", Frontiers in Handwriting recognition, C. Y. Suen (Ed.), Centre for Pattern Recognition & Machine Intelligence, Concordia University, Montreal, Canada, pp. 39-52.

TAKA-88 Takahashi, K., Amanuma, H., Adachi, Y., and Iwasa, M., (1988), "Reduction transform for application to pattern recognition", Proc. of Internat. Conf. on Computer Processing of Chinese and Oriental Languages, Aug. 29 - Sept. 1, Toronto, Canada, pp. 44-48.

TAMI Tamil Nadu Government, Income Tax Department, Land Owners Record.

TANE-81 Tanenbaum, A. S. (1981), Computer Networks, Prentice-Hall, Inc.

TANG-88 Tang, Y. Y., Cheng, H. D., and Suen, C. Y., (1988), "Size-rotation-invariant character recognition", Proc. of Internat. Conf. on Computer Processing of Chinese and Oriental Languages, Aug. 29 - Sept. 1, Toronto, Canada, pp. 161-165.

TAPP-88 Tappert, C. C., Suen, C. Y., and Wakahara, T., (1988), "On-line handwriting recognition - a survey", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 1123-1132.

TAPP-90a Tappert, C. C., Suen, C. Y., and Wakahara, T., (1990), "The state of the art in on-line handwriting recognition", IEEE Trans. Pattern Analysis and Machine Intelligence, Vol. 12, No. 8, pp.787-808.

TAPP-90b Tappert, C. C., (1990), "Rationale for adaptive online handwriting recognition", Frontiers in Handwriting recognition, C. Y. Suen (Ed.), Centre for Pattern Recognition & Machine Intelligence, Concordia University, Montreal, Canada, pp. 13-22.

TARA-78 Taraghi, M., (1978), Automatic Recognition of Printed Farsi Texts, M.S. Thesis in Computer Science, Arya-Mehr Univ. of Technology, Tehran, (in Farsi).

TOU-74 Tou, J. T. and Gonzalez, R. C., (1974), Pattern Recognition Principles, Addison-Wesley Pub. Co., Reading, Mass., USA.

TSAI-80 Tsai, W. and Fu, K. S., (1980), "Attributed grammar - A tool for combining syntactic and statistical approaches to pattern recognition", IEEE Trans. on Systems, Man Cybern., Vol. SMC-10, No. 12, pp. 873-885.

TSUK-88 Tsukumo, J., and Tanaka, H., (1988), "Classification of handprinted Chinese characters using non-linear normalization and correlation methods", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 168-171.

TUCK-74 Tucker, N. D. and Evans, F. C., "A two-step strategy for character recognition using geometrical moments", Proc. 2nd Int. Joint Conf. Pattern Recognition, pp. 223-225.

UEDA-83 Ueda, S., Okunaka, J., Ujiie, M., Hattori, K., (1983), "Japanese text standardization for teletex communication", Proc. of 1983 Internat. Conf. on Text Processing with a Large Character Set, Chinese Language Computer Society (USA), pp. 275-280.

VERW-88 Verwer, B. J. H., (1988), "Hilditch skeleton", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 137-142.

VIDY-89a Vidya Sagar, A., and Muralidhar, G., (1989), "CFONTS - A font design system", Proc. of the Regional Workshop on Computer Processing of Asian Languages (CPAL), Sept. 26-28, Bangkok, Thailand, pp. 137-146.

VIDY-89b Vidya Sagar, A., and Sanjeev Chadda, (1989), "Composite characters formation in Indian scripts with a small set of working patterns -- A postscript implementation", Proc. of the Regional Workshop on Computer Processing of Asian Languages (CPAL), Sept. 26-28, Bangkok, Thailand, pp. 160-167.

WAKA-88 Wakahara, T., (1988), "On-line cursive script recognition using local affine transformation", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 1133-1137.

WANG-84 Wang, Q. R. and Suen, C. Y., (1984), "Analysis and design of a decision tree based on entropy reduction and its application to large character set recognition", IEEE Trans. Pattern Anal. Mach. Intell., Vol. PAMI-6, pp. 406-417.

WANG-85 Wang, P. S. P., (1985), "A new character recognition scheme with lower ambiguity and higher recognizability," Pattern Recognition Letters, Vol. 3, pp. 431-436.

WANG-88a Wang, Z. -X, and Faure, C., (1988), "Structural analysis of handwritten mathematical expressions", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 32-34.

WANG-88b Wang, K., Tang, Y. Y., and Suen, C. Y., (1988), "Multi-layer projections for the classification of similar Chinese characters", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 842-844.

WANG-89 Wang, P. S. P., and Zhang, Y. Y., (1989), "A fast and flexible thinning algorithm," IEEE Trans. Comput., Vol. 38, No. 5, pp. 741-745.

WESZ-78 Weszka, J. S., (1978), "A survey of thresholding selection techniques", Computer Graphics and Image Processing, Vol. 7, pp. 259-268.

WHIT-83 White, J. M. and Rohrer, G. D., (1983), "Image thresholding for optical character recognition and other applications requiring character image extraction", IBM Journal of Research and Development, Vol. 27, No. 4, pp. 400-410.

XIA-88 Xia, Y., (1988), "Minimizing the computing complexity of interactive sequential thinning algorithm", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 721-723.

XIA-90 Xia, Y., and Sun, C., (1990), "Recognizing restricted handwritten Chinese characters by structural similarity method", Pattern Recognition Letters, Vol. 11, No. 11, pp. 67-73.

XIU-88 Xiu-Guang, Z., and Jia-Ruo, W., (1988), "Hyper-semigroup algebraic structure and its application to Chinese character recognition", Proc. of Internat. Conf. on Computer Processing of Chinese and Oriental Languages, Aug. 29 - Sept. 1, Toronto, Canada, pp. 170-176.

YAMA-88 Yamada, H., Yamamota, K., and Saito, T., (1988), "A nonlinear normalization method for handprinted Kanji character recognition: Line density equalization", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 172-175.

YOSH-88 Yoshimura, I., and Yoshimura, M., (1988), "Writer identification based on the ARC pattern transformation", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 35-37.

YUHA-87 Yuhang, M., (1987), "Direct radical-consonant coding of Chinese characters", Int. Conf. on Chinese and Oriental Language Computing, Chicago, pp. 239-244.

ZENG-88 Zeng, J., Inoue, T., Sanada, H., and Tezuka, Y., (1988), "A data structure suitable for representing the calligraphic rules for Chinese character evaluation", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 181-183.

ZHAN-80 Zhang, S., (1980), "The regular expressions inference for syntactic recognition of handwritten numerals", Proc. 5th Int. Conf. on Pattern Recognition, pp. 1004-1006.

ZHAN-84 Zhang, T. Y., and Suen, C. Y., (1984), "A fast parallel algorithm for thinning digital patterns", Comm. ACM, Vol. 27, No. 3, pp. 236-239.

ZHAN-88a Zhang, Z., He, M., and Ge, C., (1988), "A research on printed Chinese character recognition based on stroke features with optical/digital hybrid realization", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 573-575.

ZHAN-88b Zhang, Y. Y., and Wang, P. S. P., (1988), "A maximum algorithm for thinning digital patterns", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 942-944.

ZHAN-88c Zhang, Y. Y., and Wang, P. S. P., (1988), "A modified parallel thinning algorithm", Proc. of 9th Internat. Conf. on Pattern Recognition, Ergife Palace Hotel, Rome, Italy, Nov. 14-17, pp. 1023-1025.