TEXT-TO-SPEECH CONVERSION BY RULES

SOME ASPECTS OF TEXT-TO-SPEECH CONVERSION

BY RULES

By

NARAYANA RAMASUBRAMANIAN, Ph.D.

A Thesis

Submitted to the School of Graduate Studies

in Partial Fulfilment of the Requirements

for the Degree

Master of Science

McMaster University

September 1976

MASTER OF SCIENCE (1976) (Computation)	McMASTER UNIVERSITY Hamilton, Ontario

TITLE:	Some Aspects of Text-to-Speech Conversion By Rules		
AUTHOR:	Narayana Ramasubramanian,	B.Sc.	(Madras)
		M.A.	(Agra)
		Ph.D.	(Kurukshetra)
SUPERVISOR:	Professor R.A. Rink		
NUMBER OF PAGES:	ix, 139		

ABSTRACT

A critical survey of the important features and characteristics of some existing Text-to-Speech Conversion (TSC) system by rules is given. The necessary algorithms, not available for these systems in the literature, have been formulated providing the basic philosophies underlying these systems. A new algorithm TESCON for a TSC system by rules is developed without implementation details. TESCON is primarily concerned with the preprocessing and linguistic analysis of an input text in English orthography. For the first time, the use of function-content word concepts is fully utilized to identify the potential head-words in phrases. Stress, duration modification and pause insertions are suggested as part of the rule schemes. TESCON is general in nature and is fully compatible with a true TSC system.

ACKNOWLEDGEMENTS

My thanks are due to : Dr.R.A. Rink, my thesis supervisor; the School of Graduate Studies, McMaster University, Hamilton (Ontario) for providing financial assistance during my stay at McMaster (1974-1976) and also for awarding a travel grant to attend and participate in the 1976 IEEE International Conference on Acoustics, Speech and Signal Processing, Philadelphia, Pa. (U.S.A) and my friends for their direct and indirect help during my stay at Hamilton (Ontario), Canada.

T A B L E   O F   C O N T E N T S

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

SPEECH AND TEXT

1.0 INTRODUCTION

The purpose of this present thesis is to investigate some of the theoretical aspects of a scheme for 'Text-to-Speech Conversion-by-Rules'. In addition, a formulation of an algorithm TESCON for such a scheme is also proposed.

Definition:

A Text-to-Speech Conversion scheme (TSC) may be defined as a transformation of an abstract message embedded in an alphabetic string in a given language into its corresponding acoustic wave form, from which the message can be perceived by a normal human being.

In general, the realization of such a transformation will be possible by the following four blocks (or major steps):

(i) A Pattern Recognition Block:

The input to this block will be text from a printed page, or from other sources, such as a teletype, paper tape, punched cards, etc.

The purpose of a Pattern Recognition block is to isolate the patterns embedded in the input text. The patterns may be ordinary words, mathematical symbols, pictures, punctuations and styles of printing. This block then converts these identified patterns into a single pattern, such as a string of alphabets in a language or code.

(ii) A Linguistic Analysis Block:

The input to this block will be the standardized alphabetic string generated by the Pattern Recognition block. The purpose of this block is to perform a specified linguistic analysis on the input string. The linguistic analysis is the comparisons of input patterns with the given entries in a dictionary, determination of the uniqueness of the results, determination of the word categories, syntactic categories, and syllabic structures, and any additional relevant information of the results. This block will also decide the necessary pauses (or silence gaps) to be introduced in the input text, intonation, stress and duration modifiers, etc. Thus, the output from this block will be a complete linguistic code or simply, a phonetic code.

(iii) An Acoustic Specification Block:

The input to this block will be the phonetic code generated by the Linguistic Analysis block. The purpose of this block is to produce a spectrum matrix. The spectrum matrix will specify the steady-state acoustic parameters for the individual phonetic alphabets of the input phonetic code, the transition between a pair of phonetic elements, etc. Thus, the results of this block will be a dynamic acoustic specification of a given phonetic code suitable for speech synthesis.

(iv) A Synthesis Block:

The input for this block will be the dynamic acoustic specifications of a phonetic string (or code). The purpose of this block is to produce necessary control signals to operate a speech synthesizer in real time. The results of this block will be a speech wave form in real time corresponding to the input text.

It is clear that a Text-to-Speech Conversion system (TSC) involves some aspects of pattern recognition, linguistics, acoustics, and engineering. All these are considered here as computational problems.

There are many schemes for producing speech synthetically. These schemes may use one or more of the blocks given above. A few examples of speech synthesis schemes can be given here : Resynthesis of natural speech via linear predictive code (LPC) [ATA 1971]; automatic text-to-speech via a pronouncing dictionary lookup scheme [TER 1968],[UME 1975], [COK 1973]; and speech synthesis by rules [HOL 1964],[THO 1971]. Of these, methods, we restrict ourselves to Speech-Synthesis-by-Rules schemes only.

In this thesis, we have investigated some aspects of the pattern recognition block and the linguistic block which enable us to obtain a transformation of an input text into its corresponding phonetic text. The remainder of the transformations are incidental and will be discussed briefly for the purposes of completeness.

Before we go into details of a TSC system, let us first define some important terminology which will be used in this thesis.

Audio-response unit : a hardware setup which accepts an analogue voltage output from a computer via digital-to-analogue converter and generates corresponding audio-frequencies through an electronic amplifier and a loudspeaker.

Language: a code consisting of a set of alphabets or characters that can form well defined sentences according to a given set of rules (Grammar).

Machine: a hardware computer setup capable of performing well defined functions within certain limitations.

Morph: a smallest linguistic unit capable of conveying either a lexical or a grammatical meaning. For example, the words go, come, of, or the past tense suffix ed in English are morphs.

Orthography: a set of given alphabets, punctuations and conventions used to represent a discourse (collection of sentences conveying some message) in a given language providing visual symbolic form.

P-mode: an abbreviation for print-mode. This is one of the most commonly used input-output modes in a computer today. P-mode also represents a formal writing by a person in a natural language, such as English.

Pre-processing: a processing performed on some input, producing a normalized and uniform output. This output may then become input to some other well defined processing. For example, replacing a capital letter of a word in English by a small letter is preprocessing.

Rule: a process of rewriting one set of a given symbols or alphabets in terms of another set of symbols either without any restriction as to any context (context-free) or with restrictions (context-sensitive).

Spacial processing: a visual processing of information in two-dimensional space without regard to time.

Spectrum: a band of frequencies observed when some energy of sound radiates from a source and is passed through a filter-bank separating each of the components of the sound according to its frequency and the power (or intensity) in relation to time.

Speech: a process of encoding a message as an audible acoustic wave through the organs of speech (vocal setup) of a person or a synthesizer, such that a listener can perceive and decode the acoustic wave, recognizing a message within a linguistic convention.

SYnthesis: a process of creating an acoustic wave form of a message in a language without involving vocal organs of a person.

Synthesizer: a hardware device consisting of a set of digital or analogue filters with one or more sources of excitation capable of producing an audible waveform.

Temporal-processing: a sequential processing of information related to time.

Text: a body of matter on a written or printed page in a given orthography.

Voice: sound produced by the vocal organs of a person (or by synthesis) in a linguistic context.

In this chapter we will examine three important aspects of natural language based communication which are relevant to a TSC system. These are speech, orthography and text. Towards the end of this chapter, we will present an overview of the organization of the present dissertation.

1.1 Speech Code:

Speech is a code [LIB 1968] and is the primary mode of human communication. Individuals within a speech community are able to transmit information through voice coding. The transmitting of information through voice has been well developed in the human race. Voice can carry more information than other codes [NEW 1971] and voice is a preferred mode of communication [CHA 1971; OCH 1974].

1.2 Limitations of Speech:

Voice communication has its own limitations. Individual voices loose energy over a distance. Hence, the proximity of a speaker and a listener was a must in voice communication or in speech mode until recently. Further, the communication that is taking place in an air medium cannot support unlimited variations in acoustic pressure due to the voice signal without distortion [FLA 1972]. However, modern communication channels, such as, the telephone, have overcome some of these difficulties though they are in no way a substitute for full communication involving both speech and pictures, such as, in a class room.

1.3 Orthography:

The secondary method of human communication is through the coding of the information in P-mode, though as many as ten different modes of communications are possible by human [CHA 1975]. An alphabet or a picture may be the basic unit of such an orthographic or written system of communication in P-mode. All modern communication involves the use of orthography.

1.4 Advantages of Orthography:

Orthography or written code is versatile. A written code may be an alphabetic code, a picture code, a criptogram, or a combination of these. Including pictures all codes are transmittable over various media, such as paper, cloth, hard-surface,teletype,etc. These codes are devoid of the personal mannerisms, age, sex and health of a person producing these codes, which are often interwoven in the information of the voice communication. Thus, spatial processing of written codes is simpler when compared to temporal processing of speech signals. While the rate of coding affects the decoding process in a listener, written code does not affect the decoding rate of a person familiar with such a code. We must realize that errors can exist in both the modes of communication. In an idealized orthography errors would be absent.

1.5 Variation in Orthography:

There are many kinds of orthographic systems. For instance, a voice code may not have one-to-one correspondence for a given orthography. That is, for a given orthographic symbol, there can be more than one phonetic value depending on contexts. Further, different shapes and sizes of alphabets, different kinds of alphabets to represent mathematical symbols, different kinds of mathematical symbols, different conventions to code pictures, are all introducing variations in an orthography. All of these may be used in a printed text. Thus, we may say that a text is a combination of various orthographic systems involving normal alphabets associated with a particular orthographic system, mathematical systems, and pictorial systems.

1.6 Text:

Apart from the combination of various orthographic systems found in a text, there are classifications of subject matters within a text, such as physics, mathematics, geography, computer science, etc. While all the texts are composed of some basic alphabets for a given language, each text is related to a particular area of knowledge which selects its own special vocabulary, defintions, mathematical symbols, and pictorial representations according to certain conventions. While the vocabulary may differ from one subject area to another, texts use the same basic alphabets for a given language. However, the pictures differ in their form and functions with respect to each subject matter or a group of subjects. There is no apriori rule that a text must make use of pictures. However, general technical subjects, such as science and engineering, etc. make use of classes of pictures, though they may be limited in number. Thus, operationally, a text may be either a literary-text involving only alphabets of a language or a technical-text involving both mathematical symbols and the alphabets of a language. Both literary and technical texts may have pictures.

1.7 Machines and Texts:

Alphabetic coding via printed texts is being used not only in human communication, but is also used in man-machine communication systems. Computer programming languages are the major linguistic codes used for man-machine communication systems. Especially, the higher level languages, such as, FORTRAN, ALGOL, etc., use codes that resemble natural languages or the literary text of an English speaking community. Thus, the use of written codes in a language is the rule of the day involving documentation for future use.

It is interesting to note that while human beings are capable of encoding and decoding information in both the S-mode (Speech-mode) and the P-mode (Print-mode), in man-machine communication only P-mode is used. Both the input and the output in a computer system is in most instances the P-mode.

It is understandable that conversion of speech into P-mode is complex when compared to the decoding of a text from P-mode to a message in S-mode. The complexities arising in natural languages like English are due to the complex coding schemes at sound level, morphological level (word level), syntactical level, and the semantic vevel. Further, contexts, subject matter, and the area of knowledge are also involved.

Conceptual bases, styles, personal choices, paraphrasing, and individual preferences introduce complexities in written codes. While a filtering process at all levels may be able to create the basic concepts strictly in mathematical or logical terms, this is not necessary in a general communication context. Though the domain of knowledge has expanded by leaps and bounds in the past thirty years, speech recognition in S-mode has become very complex to handle by simple methods. Unless the complete S-mode can be split into subsystems with their inter-relationships clearly defined, this area of research will be difficult to understand for some time to come. The complexities of speech recognition and various strategies to handle some of these problems are reported in the literature [RED 1976].

A non-trivial area of interest in speech communication is the decoding of texts to speech-mode (S-mode). That is, given a text in P-mode, how to convert it into S-mode. The state of art in speech synthesis technology shows that a voice-readout of computed numerical values is available in pocket size calculators [COM 1975]. The availability of hardware speech synthesizers, such as the VOTRAX, are being used more and more in voice readout technology. Limited commercial applications for stock exchange information nave been reported in the literature [BUR 1968] and for wiring telephone apparatus elsewhere [FLA 1972]. In general, English has been used in such attempts.

1.8 Tuning a System:

If natural languages are used in a voice communication system, the first step is to identify the language that is being used in the communication at a given time. This involves the selection of a language from among the many possible languages at a given time, which is referred as tuning. Thus, tuning a system may be viewed as the language selection process and the selection of related information. such as the mode of the language like P-mode or S-mode, etc., and also the allowed interactions. Thus, there exist the necessity to allow the embedding of the rules of many language systems, such as natural language, formal language and pictorial language in a system. A system is thus general purpose one, only if continuous tuning within the system is possible.

A general system setup that allows tuning is shown in figure 1.1.

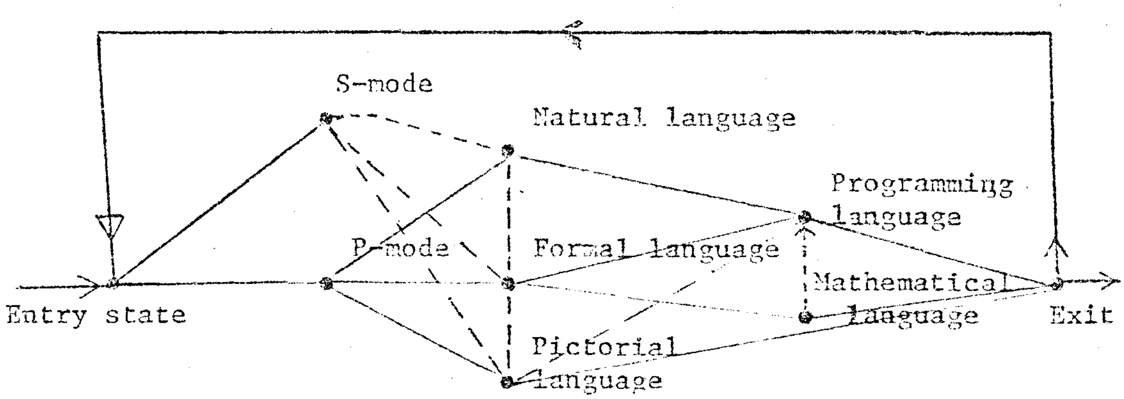

Figure 1.1 State-diagram for the language tuning.

Notice that in figure 1.1, the broken lines indicate the possibilites of inter-language communication links. This allows different sets of codes to proliferate in a TSC system. We will assume the existence of an appropriate mapping function in the system to facilitate these linkages.

We note that in a human communication system, a human being can use a variety of natural languages to communicate with different speech communities. This roughly corresponds to the tuning of the human communication system for one or more of the languages. For a machine we have restrictions at the present time. For a machine the tuning is through a formal language state only in P-mode, especially, a computational language. Normally, all the language states used should be mapped onto a formal language code (state) and then mapped onto a computational language state. However, we do not have a single programming language at present that will allow all the language states within its domain; something which is possible in a human communication system. This is a major problem suitable for future research.

1.9 Assumptions made in this thesis:

In this thesis we are selecting a somewhat limited problem for investigation. The major theme of the thesis is that given a text in a printed form, it is possible to convert into speech by synthesis via a machine (computer) through a suitable TESCON (TExt-to-Speech CONversion) algorithm. The complete system is called the TSC system (Text-to-Speech Conversion system). In this thesis we will propose a new algorithm TESCON for a TSC system bu rules. In doing this, the following assumptions are made:

1.9.1 Type of Input Text:

(a) A literary text entirely composed of the alphabets of a given natural language, numerals, and punctuation marks used therein is acceptable.

(b) A text can be a mathematical text composed of mathematical symbols and the alphabets of a natural language and the words (or vocabulary or lexicon), numerals and the alphabets of some other natural language(s).

(c) Scientific texts composed of words and the alphabets of a natural language, mathematical symbols, and formulae are acceptable.

While a literary text can be handled by a Pattern Recognition block mentioned in section 1.0 (i), the input equipment of the system are the common types, such as teletypes, paper tapes or punched cards. Both mathematical and scientific texts are difficult to handle unless Optical Character Reader (OCR) and picture scanner units are utilized in the system.

1.9.2 Language of the Input Text:

While in theory any natural language is allowed in the P-mode within a TSC system, a standard dialect of either North American English or standard British English is assumed. Usages and spellings will not affect the text or its conversion. In addition, no error detection procedures are assumed.

1.9.3 Input Mode of Texts:

The given text may be input in one of the following three P-modes:

(i). Punched on cards or paper tapes.

(ii). Typed on a computer console or a teletype.

(iii). Printed text on a sheet of paper.

1.9.4 Processing Language:

Any programming language which can accept a normal English text in Latin alphabets as input (or an equivalent ASC-II code) is acceptable in a TSC system. For example, the string processing language SNOBOL, a list processing language LISP, a problem oriented language like FORTRAN with SLIP (Symmetric LIst Processing) for dynamic memory allocation, are all acceptable in a TSC system. A few examples of incomplete systems are found in the literature [ELO 1976; THO 1971].

1.9.5 Preprocessing:

Existence of facility for preprocessing of the input text is assumed in a TSC system. This facility should be such as to enable us to produce a uniform code for further processing and conversion to speech.

1.9.6 Audio-response unit:

Existence of either hardware or software (simulated) compatible audio-response unit to generate voice-output from the synthesis scheme is assumed in a TSC system.

1.9.7 Computer System:

A high speed medium size general purpose computer system with adequate memory size of the order of 128 K words with 16-bit word size is assumed. In addition, suitable conventional input-out devices are assumed to exist in the system.

A block diagram for a simple computer setup for a TSC system is shown in figure 1.2.

In at least one system [ALL 1973] attempts have been made to use an OCR as an input device and all other systems to be discussed in the next chapter use normal input devices.

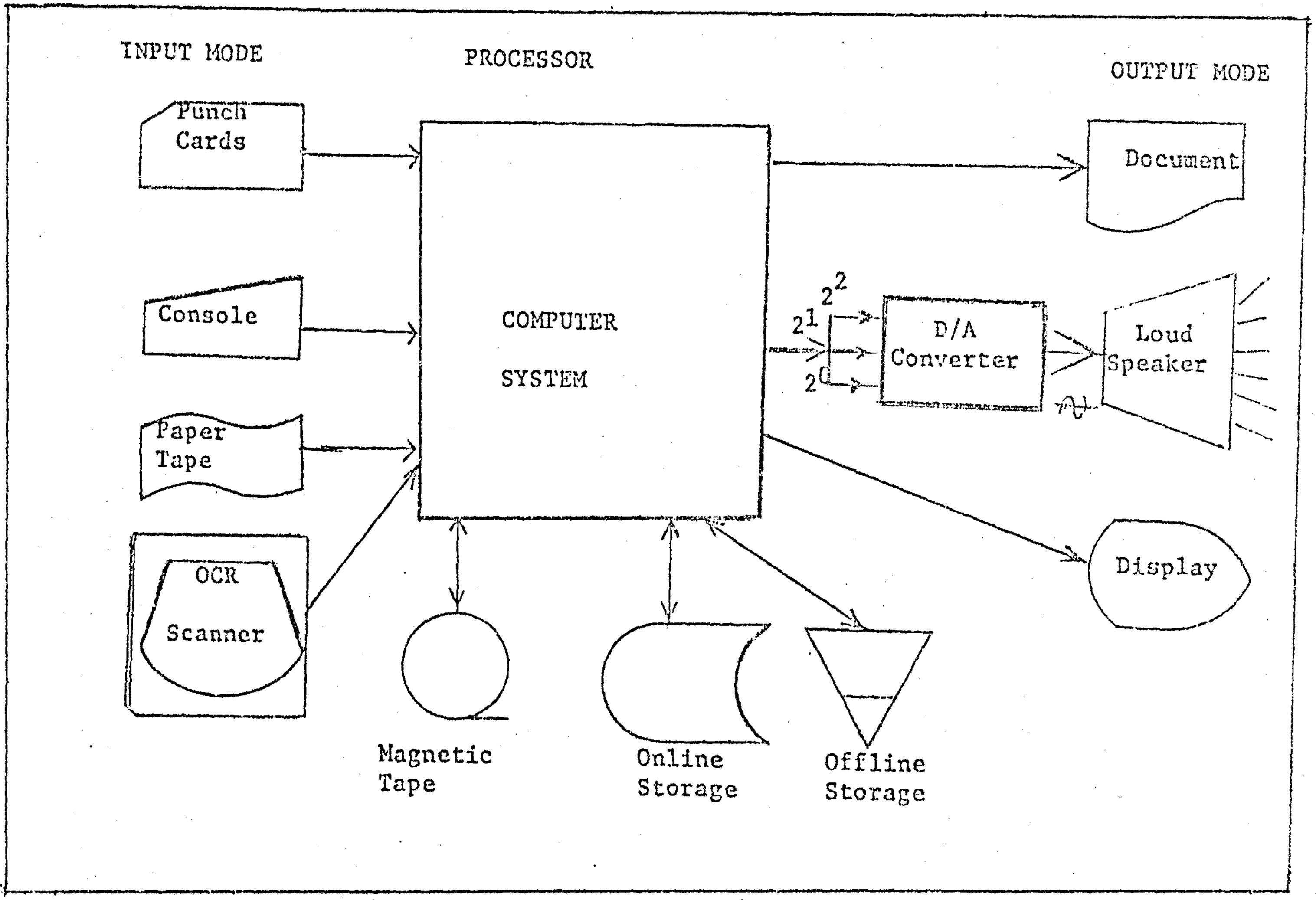

Figure 1.2 Block Diagram of a Computer Based TSC System.

1.10 Overview of the dissertation:

This dissertation is devided into six chapters.

Chapter 2 surveys some of the existing systems reported in the literature.

Chapter 3 considers the problems of preprocessing and analysis of input text and the normalization of the input text in an alphabetic system. In this chapter, we propose a new algorithm called the STANDARD - IZER, to deal with some of the problems of preprocessing.

Chapter 4 discusses the problem of stress, duration assignment for English words in various contexts,and the proposed algorithm ANALYZER to handle some of these problems using function-content word concepts.

Chapter 5 provides the necessary overall rules and the TESCON algorithm for a TSC system by rule. TESCON integrates STANDARDIZER and the ANALYZER with TUNER algorithm.

Chapter 6 outlines the possibilities for implementation of the proposed TSC system in terms of Text-to-Phonetic form and from Phonetic to speech output. It concludes by summarizing the contributions that this thesis has made and discusses future research problems.

CHAPTER 2

SOME EXISTING SYSTEMS

2.0 MOTIVATION

Speech code has the highest capacity for carrying information [NEW 1971]. Because of this, there is a high motivation to utilize this capacity in the communication industry. Computer based voice terminals have many potential applications. Some of the commercial applications envisaged are:

(a) a reading machine for the blind [ALL 1973; COO 1969],

(b) voice based encyclopedic information service [UME 1976],

(c) voice answering systems at remote terminals making use of a centralized data base in a given natural language [LEE 1968],

(d) voice announcement of a current status of a computer system, calling the attention of a computer operator when necessary,

(e) voice based flight information system [SCH 1975],

(f) wiring of telephone connections based on computer generated voice commands [FLA 1972],

(g) voice based telephone directory assistance [LEA 1968],

(h) voice readout for hand-held calculators [COM 1975],

(i) other uses [FLA 1973, LEA 1968].

While a general purpose reading machine is yet to be developed, various realizations of the subsystems have been reported in the literature [FLA 1973; COO 1969; CHA 1971].

2.1 A General Purpose TSC System:

A general purpose computer based TSC system is given as a block diagram in figure 2.1. Block names in figure 2.1 are defined in section 1.0.

The block diagram in figure 2.1 serves three purposes:

(a) it provides a broad conceptual frame work of a general purpose TSC system;

(b) it identifies and names the subsystems explicitly;

(c) and with the overall system being clearly defined, it permits us to investigate any one or more subsystems without going into details.

As stated earlier, we will concentrate more on the first two blocks, namely, the Pattern recognition block (block A) and the Linguistic analysis block (block B), and the other blocks will be briefly discussed only for the sake of completeness.

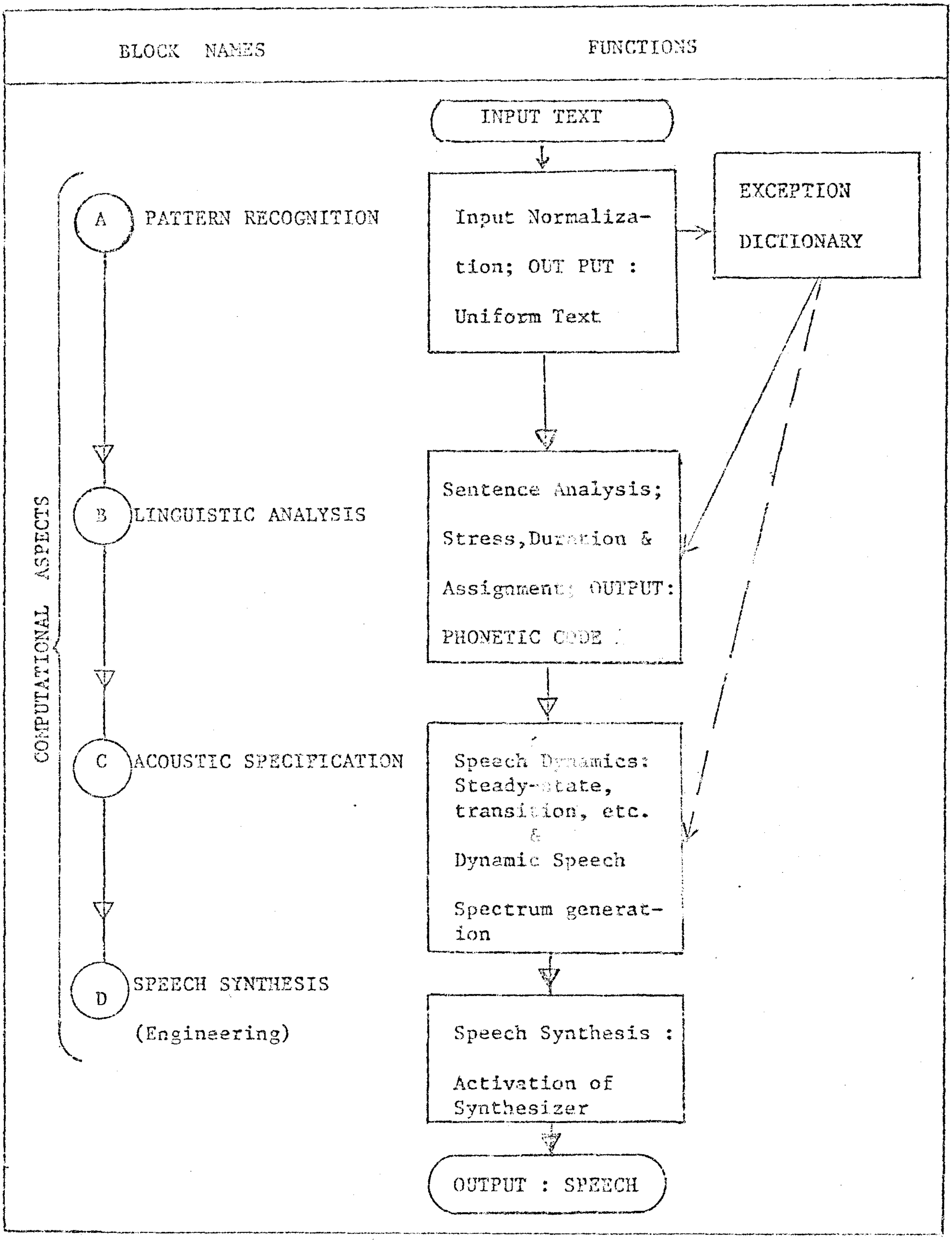

Figure 2.1 Block diagram of a general purpose TSC System.

2.2 Speech Synthesis:

The earliest attempt to produce synthetic speech was mainly an engineering aspect (shown in block (D) in figure 2.1). Dudly [DUD 1939] exhibited his synthetic speaker in 1939 at the New York world fair. It is interesting to note that this piece of hardware is the ancester of today's hardware speech synthesizers. Today's commercial speech synthesizers are becoming part of many computer systems. In addition, there are other communication equipments, like a narrowband digital voice transmission system[KAN 1974], which include a synthesizer suitable for voice output. All the digital harware synthesizers are compatible with digital computers, and thus, programmable in real time. The purpose of a synthesizer is to accept control commands from a computer corresponding to the acoustic specifications given in block (C) in figure 2.1, and generate a continuous acoustic spectrum (or speech wave) in digital code suitable for conversion into real time analogue signal as shown in block (D) in figure 2.1.

There are many software realizations of hardware speech synthesizers via digital simulation reported in the literature [FLA 1973, HOL 1964;THO 1971]. While each of these have their own merits and limitations, they serve equally well as a synthesizer. Therefore, we will assume a well defined and documented subsystem for the synthesizer part (block D) in all existing systems to be discussed.

2.3 Overview of some existing TSC systems:

Before we discuss some of the existing systems reported in the literature, let us explicitly state the requirements of any general purpose TSC system. In doing this, we will be able to evaluate some of the existing systems with respect to our requirements. The requirements may be stated as follows:

(1) The input to the system should be general texts, such as non-scientific and scientific texts, without involving pictures at the present time.

(2) The number of rules used in the system should be minimal, say a few hundred and also the dictionary entries for exceptions should be minimal at any given time.

(3) The system should not involve a detailed and exhaustive syntactic analysis of the input text.

(4) The letter-to-sound rules should be general and should be a set of external data and modifiable, thus permitting the tuning of the system for dialects of natural languages.

(5) The system should be independent of any particular synthesizer and its characteristics.

(6) The system should be implementable on a general purpose medium or mini-computer with real time performance.

(7) The memory requirements should be minimal, say around 120 K words.

(8) The system performance should be statistically measureable.

Having defined the requirements of a TSC system, the next step is to define the system specifications. These system spefications can be broken down into sub-systems specifications. In figure 2.2, we propose modular sub-system blocks having the following five components:

1. a system goal,
2. a control,
3. an input to the sub-system,
4. a process in the sub-system,
5. and an output from the sub-system.

Each block of a TSC system can be represented as a schematic system flow diagram as shown in figure 2.2.

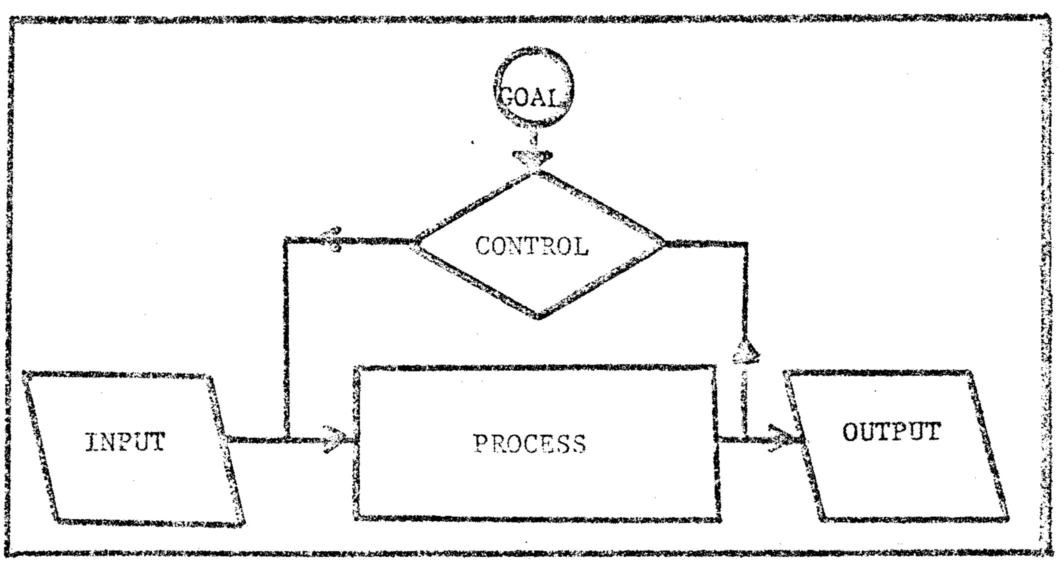

Figure 2.2 A schematic system flow diagram for a subsystem.

We note that this representation enables us to consider the goal of each block explicitly in comparison to analgorithm that would only have four of the five components of a sub-system (2-5). The reason for using a block diagram approach rather than algorithmic approach for a subsystem description is that the algorithms to be described in the following sections have been re-created from the literature. The literature however, is incomplete and inadequate. Hence, where pertinent information is missing, we have used block names suggestive of an appropriate action in our re-created algorithms. This approach enables us to specify at least the goal of a block, through a block name. Since we know the input and output for the previous and the following blocks with respect to a given block, we can at least infer the goal of an unexplained block from its name. This approach somewhat overcomes the incompleteness and inadequacy for a given recreated algorithm in this chapter.

In the following discussions, we have tried to reconstruct the algorithms of several systems from the literature. We have used our own logic in the reconstructions, thus preserving the general philosophy of these algorithms. Therefore, the common underlying principles should be clear in each algorithm even though some block names represent inadequate data and details. Since we are mainly concerned with block A and block B of a TSC system (figure 2.1), the algorithmic details of the remainder of the systems to be discussed have no effect on the overall setup in our discussions. Keeping the above restrictions in mind, we now discuss some existing systems.

2.3.1 The MIT System:

2.3.1.1 The Building Blocks:

Allen et al [ALL 1973; LEE 1969] have reported on the MIT system. This system, a basic text-to-speech conversion, attempts to handle some limited printed text using an OCR and picture scanner setup. The basic building blocks of this system are:

Block-1: employs a dictionary lookup to aid in the pronounciation of the homographs (i.e., words with identical spelling but with difference in meaning, such as, wind, refuse, lives, watch, etc.).

Block-2: analyses phrases and assigns the stress and inflection (pitch, etc) to phonetic transcription.

Block-3: employs a hardware speech synthesizer to produce speech output for converted phonetic text.

2.3.1.2 The MIT Algorithm:

[Initialize an INTEGER WORD count]

Step0: WORD $\longleftarrow$ 0

Step1: Read a character from an input text.

Step2: If end of input, Terminate the algorithm.

Step3: If the character is not a blank, or not a punctuation mark, jump to Step1.

Step4: WORD $\leftarrow$ WORD + 1

Step5: Hash the word (input) with a dictionary and if a word has unique phonetic equivalence, jump to Step10.

Step6 [Invoke the PART-OF-SPEECH Block]

If the phonetic equivalence is found for the input word, jump to Step10.

Step7: Segment the input word into morphs and affixes.

Step8: Hash the morphs with a morph-dictionary and if hashing is successful, jump to Step10.

Step9: Apply the letter-to-sound rule.

Step10: [Invoke the PHRASE-ANALYSIS block]

Assign stress and inflection for an input word.

Step11: [Invoke SIGNAL-GENERATION block]

Generate signals to operate a synthesizer.

Step12: [Activation of a Hardware Synthesizer]

Synthesis speech signal and play in real time.

Step13: Jump to Step1.

2.3.1.3 The System Setup:

HARDWARE:

A PDP-9 minicomputer with a high speed drum storage facility has been used in this system. The system has an OCR for reading a page at a time.

The estimated external storage is about 4 million bits for a 32,000 word dictionary [LEE 1968]. The program, data,etc.,will fit within the 16K word memory of the PDP-9 system.

SOFTWARE:

The MIT system has a dictionary of 11,000 words and 400 letter-to-sound rules. The phrase analyzer does not handle phrases of sentences completely. For every dictionary entry, the parts-of-speech details, alternative transcriptions and some internal flags are necessary. Syntax and stress analysis are absent and are to be added later.

2.3.1.4 Performance Measurements:

The system has been tested with a fourth grade text, and with an OCR reading rate of two and half minutes per page. We do not know the processing time, and the type of programming languages used in this setup. A list of letter-to-sound rules, and a quantitative evaluation of the performance of the system are also not available.

2.3.2 The Keele University System (KUS):

2.3.2.1 The Building Blocks:

Ainsworth has developed a system at the Keele University and is reported in the literature [AIN 1973; 1974]. This system is based on a letter-to-sound rule scheme. It converts a text punched on a paper tape to phonetic symbols (or their equivalent ASC-II codes). The symbols are used to generate parameter controls for a synthesizer. An analogue hardware speech synthesizer produces speech signals in real time.

The basic building blocks of the KU system are:

Block-1 : produces segmentation of the input text into breath-groups. The breath-groups introduce pauses (silence gaps) in the text at desired places. A buffer of 50 character size, which is a part of the system. Input text is stored in this buffer and rules are applied to achieve the breath-groups.

Block-2 : translates an input alphabet into a phoneme (a linguistic unit witha fixed steady-state characteristics[RAM 1973]) via a set of letter-to-sound rules in a table lookup scheme.

Block-3 : assigns stress via set of rules. A small exception dictionary is provided.

Block-4 : computes the speech parameters and their values for individual speech sounds and generates speech signals via a speech synthesizer.

2.3.2.2 The KUS Algorithm:

Step0 : Initialize a buffer of 50 character size.

Step1 : Read an input character from the paper tape.

Step2 : If the character readin is a punctuation mark, jump to Step8.

Step3 : Fill the buffer with the input character

Step4 : If the buffer is not full, jump to Step1.

Step5 : Search for a conjunction/auxiliary verb/preposition/ article of English and if not successful, jumpt to Step7.

Step6 : Copy the contents of the buffer upto (but not including) the conjunction or auxiliary verb or preposition or article, on a magnetic tape, insert a silence gap mark; give a left shift to the remainder of the contents of the buffer and jump to Step9.

Step7 : Concatenate the next input word to the contents of the buffer.

Step8 : Introduce a pause, marking the breath-group.

Step9 : Convert the orthography into phonemic representation by table lookup rules.

Step10: Assign the lexical stress to appropriate syllables. (Function words, such as, articles, prepositions,etc. are not stressed;for the rest if the first syllable of a word does not contain a prefix, stress the first syllable, else stress the second syllable).

Step11: By a synthesis-by-rule scheme generate parameter values for each phoneme, stress and breath-group mark to control a synthesizer.

Step12: Jump to Step1 if input text is not exhausted.

Step13: Terminate the algorithm.

2.3.2.3 The System Setup:

HARDWARE:

A PDP-8 computer system has been used in this setup. A terminal analogue speech synthesizer hardware was connected to the PDP-8 to produce the actual speech signals. The memory used in this program is not known (4Kor 8K words ?). Processing and synthesis times are not available. The input text was punched on a paper tape and processed on the PDP-8 system.

SOFTWARE:

There are about 159 rules for the letter-to-sound conversion rules and the rules are given in table form. These rules are embedded in assembly codes. In this system, changing the rules imply the modification of assembly codes. This in turn, involves reassembly of the complete program. Thus, rules are not external data, but rather are part of the program.

2.3.2.4 Performance Measurements:

Performance measurements for this system were based on three sources of texts: a text book on phonetics, a modern fiction and one news paper article on a political theme. In all, a total of 1000 words passages of texts were used as test material. Of these, the correct translation score for the phonetic text was 92%; for the fiction 89%; and for the article 89%. Listening tests involving three subjects based on the same passage showed a correct scoring ranging from 50 to 90%. The correct score of 8 to 90% was achieved when the author of the system and a highest scoring listener of the previous three subjects were involved.

For a typical seven word sentence, the error rate is on an average less than one phonetic error [AIN 1973; 1974]. Since the system was in its initial stages of development, there were certain limitations of this system, such as, absence of sentence stress and the poor intelligibility of the synthesized speech for a naive listener,etc.

2.3.3 The Bell Telephone Laboratory System (BTLS):

2.3.3.1 The Building Blocks:

McIlroy's system developed at the BTL [McI 1974] consists of a letter-to-sound rule scheme. The basic building blocks are:

Block-1 : segments the input string into words delimited by spaces or certain punctuation marks or line breaks.

Block-2 : compares an input word with an exception dictionary.

Block-3 : performs the preprocessing of an input word converting capital letters to lower-case letters, deleting the word final s and substituting for y ie before a final consonant, and comparing it with a dictionary entry.

Block-4 : applies the letter-to-sound rules for each character of the input word.

Block-5 : generates the control signals for a hardware speech synthesizer.

2.3.3.2 <u>The BTL Algorithm</u>:

Step0 : Input a string of characters (i.e., type-in or pipe-in out of any other process on the machine).

Step1 : Hash the input word with the exception dictionary and if hashing is successful, jump to Step11.

Step2 : Map capital letters onto small letters; strip puntuations and jump to Step1.

Step3 : Strip word-final <u>s</u>; change final <u>ie</u> into <u>y</u> regardless of the final <u>s</u>; if any change is made, jump to Step1.

Step4 : [Invoke <u>AUTOMATIC-PRONOUNCIATION</u> block]
Reject one letter word or a word without a vowel.

Step5 : Mark endings, such as, final <u>e</u>, long vowels indicated by word final <u>e</u>; equivalent endings, such as, -<u>ed</u>, -<u>able</u>,etc.

Step6 : Mark potential long vowels, such as, <u>u</u>, <u>i</u>, and <u>a</u> (e.g. in word medial position followed by a consonant in mono-syllables).

Step7 : Mark medial silent <u>e</u> and the long vowels therein.

Step8 : Mark potential voiced word medial <u>s</u>.

Step9 : If Step5 to Step8 are successful, replace any stripped final <u>s</u>; scan from left to right applying pronounciation rules to word fragments and jump to Step11.

Step10 : If Step4 to Step9 fail, spell the word, punctuations and all.
Emit a burp when no spelling rule exists for a symbol.

Step11 : Output synthesized speech.

Step12 : Terminate algorithm if the input is over, else jump to Step0.

2.3.3.3 The System Setup:

HARDWARE:

A PDP 11/45 minicomputer has been used in this setup. The rules occupy about 11,000 bytes on the machine. The program runs at about 15 words per second of the CPU time [McI 1974]. There are about 4500 bytes of phonetic code, including table search and the special hand-coded paradigms, and 1900 bytes of code for interactive display and maintenance of the tables. A VOTRAX hardware speech synthesizer has been used to synthesize the speech output.

SOFTWARE:

The program is written in a higher level language, called the C-language. The system consists of more than 750 letter-to-sound rules for American English, including 100 words, 580 word fragments and 70 letters. While the program is not efficient according to the designer of this system, it is self-contained and requires no other supporting programs.

2.3.3.4 Performance Measurements:

McIlroy has reported that his program performs satisfactorily for 97% of 2000 most commonly used words in running English listed in Brown Corpus [KUC 1967]. The performance is satisfactory for 88% of the tail consisting of a 1% sample of the remainder of the Corpus, with an overall weighted performance of 97.2%. Furthermore, for 3% of the 18,000 word source in a Webster's dictionary [WEB 1966], the performance is about 94.5% correct.

McIlroy has admitted [McI 1976] that his criterion of satisfactory performance is subjective and satisfactory pronounciation is by no means the same as 'correct' pronounciation. The criterion for acceptability is merely that a word be easily understood by someone experienced to listening to the device (not a naive listener). He further reports [McI 1976] that on a recent test based on the 100 sentences (readapted to American idioms) from Ainsworth [AIN 1974], his system performance is 99.1%, 99% and 98.7% respectively. This again appears to be evaluated by subjective criterion.

McIlroy has reported that his 750 rules are in a table and are easily modifiable [McI 1974]. His scheme of rules are applied from left-to-right and right context only. This rule lookup can be done by a simple variant of binary search in an ordinary alphabetical list of rules. No careful ordering or concomitant linear searching of rules is necessary. Thus, the performance is very nearly within real time [McI 1974].

The major drawback of this system is that it lacks a stress marking scheme. The present system will become complicated and the program will grow in size when stress marking scheme is implimented.

2.3.4 The Naval Research Laboratory System (NRLS):

2.3.4.1 The Building Blocks:

The NRL system has been developed by Elovitz,et.al.[ELO 1976]. This system is based on letter-to-sound rules. The basic building blocks of this system are:

Block-1 : applies a limited preprocessing on the input text.

Block-2 : TRANS(the translation block) applies the letter-to-sound rules to the input text character by character and produces an IPA code (International Phonetic Alphabetic Code) or its equivalent ASC-II code when desired. Thus, a text to phonetic conversion is achieved.

Block-3 : applies the direct phonetic to synthesizer rules to the IPA code and produces a VOTRAX code.

Block-4 : generates the speech signal in real time via a Federal Screw Works VOTRAX VS-6 hardware speech synthesizer under the control of TI960A minicomputer.

2.3.4.2 The NRL Algorithm:

Step0 : Input a character of a text (via a terminal or a text file).

Step1 : If a terminal special character is encountered, jump to Step6.

Step2 : If the input character is not a blank, write the character on an output file and jump to Step0.

Step3 : If an input character is a punctuation mark, introduce a pause character in the output file and jump to Step0.

Step4 : If a special word is encountered, apply word-to-phonetic rules to the input word; produce IPA code and save it; jump to Step0. (When a special rule is applied to a whole input word, the input word is considered as a single character).

Step5 : Scan the input word on the output file from left to right character by character and for each character apply an appropriate letter-to-sound rule through a sequential search of the rule-file; produce IPA code and store it; jump to Step0.

Step6 : Apply translation rules to the IPA code string in the output file character by character with special symbols if any and produce VOTRAX code.

Step7 : Generate speech signal and play in real time.

Step8 : Terminate the algorithm if text is over; else jump to Step0.

2.3.4.3 <u>The System Setup</u>:

<u>HARDWARE</u>:

A remote time sharing minicomputer TI960A (Texas Instrument) with a 12,000 16-bit word memory is used in this system. This is connected to the PDP-10 time sharing system at the NRL. A teletype key-board, a CRT terminal with a key-board, a Federal Screw Work's Phonetic key-board form the input terminals. A TI-733 silent terminal and a VOTRAX VS-6 speech synthesizer form the output terminals. The NRL's PDP-10 accepts English texts from the TI960A and returns the IPA codes to the TI-733 terminal. This terminal has dual cassetts. From TI-733 terminal and a VOTRAX speech synthesizer speech is synthesized.

<u>SOFTWARE</u>:

The programming language used is SNOBOL IV [GRI 1971]. A set of 329 rules for letter-to-sound translations for American English are embedded in the program. These rules translate the English text into the IPA code by pattern matching principles of the SNOBOL language. The SNOBOL program runs on the NRL's PDP-10 system.

The system has software facilities for producing the evaluation statistics when any rule is applied to a text. A <u>STST</u>-file listing every instances of every rule used in the translation of every word in a text file is created. A program <u>STAT</u> reads the STAT-files and produces statistics on the relative importance of the rules [ELO 1976].

2.3.4.4 Performance Measurements:

The SNOBOL processor on the PDP-10 is an interpretive implementation of SNOBOL IV. TRANS, the translation block, operates under the SNOBOL processor. This is a very inefficient system. However, when the SNOBOL program is replaced by FASBOL II [SAN 1972] compiler, the efficiency of TRANS block increased by a factor of 25. The translation rates are increased from one word every half a minute or minute to one word every second or two seconds. Thus a factor of 4 or 5 of real time speech rate is achieved.

Memory requirements have been reduced three fold in some cases. TRANS block's performance shows a correct pronounciation rate of about 96% of the thousand most frequently used words in English and words of very low frequency of occurrance in the Brown Corpus [KUČ 1967], produce a 4% error rate (mispronounciation). The overall correct pronounciation performance rate is 90% or an error rate of 2 words per sentence of ordinary English. Comprehensive statistics for the performance of each rule has been produced supporting the above given performance measurements.

There is no rule for inflection (pitch, stress and timing) in this system. Therefore, only a monotonous speech is produced. This system will include such features in the future.

2.3.5 <u>The Tata Institute of Fundamental Research System</u> (TIFRS):

2.3.5.1 <u>The Building Blocks</u>:

A system consisting of blocks other than <u>A</u> and <u>B</u> in figure 2.1, has been reported by Thosar [THO 1971] and Ramasubramanian [RAM 1973]. This system has been developed at the TIFR and accepts a phonetic input (in ASC-II code), including the duration modifiers, stress and pause and the punctuations as part of the input string. This is a very powerful system in that given a hardware setup to replace the simulated synthesizer and for accepting an unrestricted text, the system will be a complete speech-synthesis-by-ryle system. Since we will be suggesting some algorithms such as TESCON, keeping this TIFR system in mind, it will be useful to study this system here.

The basic building blocks of the current TIFR system are :

Block-1 : validates the input string based on the stored symbol list and identifies the input errors, and selects the attributes of individual input symbols, such as vowel or consonant, etc.

Block-2 : forms a steady-state spectrum matrix for the input symbols and converts the duration modifiers to increase or decrease the duration of the preceding phonetic symbol. The stress mark is converted to an increase in the fundamental pitch of a preceding phonetic symbol (usually a vowel) and so on. The steady-state parameters are provided as a set of external data and are independent of the program. These parameters are user-defined values.

Block-3 : selects appropriate rules for the concatenation of relevant parameters for two adjacent phonetic symbols. These rules are external data and are user-defined.

Block-4 : generates a dynamic spectrum matrix based on the rules selected previously for the complete input string.

Block-5 : simulates a terminal analogue speech synthesizer and generates a digital speech spectrum for the phonetic string.

Block-6 : outputs the digital speech wave generated from the digital spectrum of the previous block and decodes the signal in real time via a D/A converter and audio-response unit.

2.3.5.2 The TIFR Algorithm:

In view of the fact that the blocks necessary to generate phonetic texts from an input English text is absent in this system, the algorithm regarding other blocks are omitted. In chapters 3, 4 and 5 we suggest some algorithms for blocks A and B for this system.

2.3.5.3 The System Setup:

HARDWARE:

A CDC-3600 computer system is used in this system. Memory is 32 K words of 48 bits word size. The compiled program is stored on a magnetic tape and is overlayed when necessary. 20 K memory words are allocated via a SLIP subroutine [WEI 1963] to accomodate and store the dynamic speech spectrum matrix generated from the input text. An average input sentence is approximately of one second duration when spoken by a person and on an average there are seven words per sentence. The computed speech spectrum matrix occupies roughly 20 K words in the CDC 3600 computer memory.

SOFTWARE:

A FORTRAN program simulates the complete system. SLIP subroutines compatible with the FORTRAN compiler, allow the flexibility of data specification in tree structure or list structure and the achievement of dynamic memory allocation. There are less than 50 rules for the concatenation procedures, one set for each of the duration, transition, and transition ratio required for the dynamic spectral computations.

2.3.5.4 Performance Measurements:

The actual processing time for producing one second of real time speech is about 4 seconds on the CDC 3600 system. The test samples were in English, Hindi and Tamil languages. About 7 sentences in English, 200 sentences in Tamil and 50 sentences in Hindi were generated. More than 90% intelligibility were recorded for all these samples with naive listeners (about 50 listeners were involved). The rules are adhoc and the actual values are not provided for the various parameters.

2.4 DISCUSSIONS:

Using the criteria given in section 2.3 (on page 23), the systems considered so far show that no one system can be called a TRUE TSC system. The reason for each system failing to be a true TSC system can be stated as follows:

(a) All the systems considered so far fail to accept unrestricted English text, let alone an elmentary scientific text without pictures. The only system, BTL system preprocesses an input text to handle capital letters and apostrophe symbol.

(b) Regarding the number of rules in any one system, we find that not one system has minimal sets of rules. The lowest number of rules is about 159 in KU system [AIN 1973], while the maximum number of rules run to about 750 in BTL system [McI 1974]. The NRL system has about 329 rules excluding any rules for stress marking. The MIT system depends heavily on an extensive morph dictionary and the rule part is minimal or incidental. Thus, we note that our second criterion, namely, 'the system should have minimal number of rules' is not met by any system discussed so far. The addition of stress, duration, and other types of rules when added to these systems, will increase the requirements of memory and computation time. The designers of these systems have failed to take advantage of the fact that the decoders (human beings) ignore mispronounciation in various contexts [WHI 1976], Hence, there is no need to burden the systems with too much information, such as, an exhaustive syntax analysis, dictionary lookup, etc. In other words, the failure to meet our second criterion is the result of the systems' failure to take advantage of the decoder's abilities.

(c) The MIT utilizes an exhaustive syntactic analysis of input sentences. The state-of-art situation with respect to the other systems indicates that there is a need to utilize some sort of dictionary setup for exception words, such as suffix analysis, abbreviations, stress assignment and irregular pronounciations. The absence of stress analysis in the KUS, BTLS, and the NRL system suggest that these systems will be forced to include some sort of syntactic analysis of the input sentences in the future to take care of the stress assignments, though the KUS lexical stress assignment will not be sufficient in this regard. Thus, the third requirment that system should not require exhaustive syntactic analysis of the input sentences is not met by systems discussed so far.

(d) With regard to the modifiability of the rules in a TSC system, and as far as the letter-to-sound rules are concerned, only two system seem to be general, the NRLS and the BTLS. While NRLS has formulated the letter-to-sound rules following the KU system, it is general in that rules are based both on the left and right contexts, whereas the BTLS is based on the left-to-right and right only contexts. The BTLS utilizes a variant of the binary search technique in an ordinary alphabetic list of rules [McI 1976]. Thus, computationally, the BTLS performs comfortably within a real time environment. However, the NRL system imposes certain severity on the rules. These are that they be ordered, hence introducing the necessity of a concomitant linear search of rules, and thereby increasing the processing time [McI 1976]. KU system has the rules embedded in assembly code. Hence, modifiability of the rules involves the rewriting the assembly code and reassembling the entire code each time a rule is changed. Thus, instead of being a set of data, the rules become the assembly code,thereby increasing the setup time, running time and the modification time of the system. Apparently, the absence of any comprehensive set of rules of rules and the nature of the dictionary searching and other formalities connected with the MIT system suggest that it is too poor to be modified and is loaded with too much book-keeping responsibilites. The mere failure to recognize the role of the listener introduces a higher processing time to compute too much information, such as exhaustive syntax analysis, parts of speech, etc., and storage requirements.

Even the suggested generality of the NRLS and the BTL system will suffer once they enter into the stress rule schemes. Thus, it is clear that in the future, each system will increase in cost due to higher processing time, memory requirements,etc,, hence a <u>TRUE</u> TSC system may not be easily realized.

(e) The adaptability of a system for any speech synthesizer is not practical at the present time. Since hardware specifications are not standardized in the systems discussed, even if BTL system and the NRL system are using the same type of VOTRAX synthesizer, the hardware itself has restrictions. Unless a general purpose synthesizer is conceived capable of producing any speech sound, the generality and adaptability requirements in a true TSC will have to wait.

(f) The measurements of the system's performance is well documented only in the NRL system. The statistical validity of rules from the automatically measured performance score of the NRL system gives a high confidence on ths system. Compared to the NRL system, the KU system is also somehat acceptable, though the statistical details in this system are poor (or nil). The subjective (personal feeling) evaluation of the performance of the BTL system [McI 1974; 1976] is unscientific, hence the system should not be seriously considered. While the KU system employs three preprocessing rules and 150 variable rules, the BTL system has four times the number of rules as that of the KUS to achieve less than 4% error rate. McIlroy claims a 99% overall performance score on a more recent test again based on subjective criterion [McI 1976].

The MIT system can never come near a TRUE TSC system due to its inability to document its performance in anyway.

Thus, it is clear that the ultimate performance of these systems will be different from the expected performance of a TRUE TSC system.

(g) The system implimentations are, in general, on mini-computers. However, the concept of parallel processing and micro-processors have not been utilized in any of the above systems. The serial nature of the block-by-block processing of the input as shown in figure 2.1, results in roughly four times the processing time for one second of real time speech as demonstrated by the BTL system. The TIFR system is a promising system in this regard. In the TIFR system, even with the simulation setup, the processing time is about four times for one second of speech (or its equivalent one sentence) and given a hardware setup for the synthesizer, this system will be a real time system in the future.

The sucessful use of mini-computers in the above systems (except for the TIFR system), suggest that a special purpose microprocessor with a parallel processing capabilites will be a practial one in the near future. Thus, our requirement of the true TSC system is met with by all the systems we have considered so far.

(h) Considering the memory requirments of a true TSC system, only the MIT system violates our criterion. The MIT system requires about four million bits of storage for the dictionary alone. All other systems use roughly the core memory of their host mini-computers and make use of auxiliary storage when necessary. This, however, is bound to increase when a full scale true TSC system is implemented in the future. Thus, the memory requirments for any future TSC system remain unanswered.

We have summarized the results of our discussions in Table 2.1 (a) and 2.1 (b). In the next chapter, we investigate some printing style problems involved in restricted scientific texts, including such problems as pattern recognition suitable for preprocessing. We then, suggest a new method to handle such input texts in a general purpose TSC system.

System Studied	Implimenter	HARDWARE FACILITIES				
		Year	Machine	Memory	Processing time w.r.t real time	Synthesizer Used
BTLS	McIlroy	1974	PDP-11/45	11,000 bytes	4 times	Hardware (VOTRAX)
KUS	Ainsworth	1973	PDP-8	8 K words	unknown	Hardware (Analogue)
MITS	Allen et al.	1973	PDP-9	16 K words	unknown	Software (Simulated)
NRLS	Elovitz et al.	1976	TI960A - PDP-10	12 K words	3 times	Hardware (VOTRAX)
TIFRS	Thosar	1971	CDC 3600	20 K words	4 times	Software (Simulated)

Table 2.1 (a) Summary of Hardware features of some existing Text-to-Speech Conversion Systems.

Systems Studied	SOFTWARE FACILITIES					
	Type of Text	Language of Input Text	Programming Language Used	% of Rules in the system	Usage of Dictionary	Performance & Evaluation Criterion
BTLS	Literary	American English	C- language	90 %	Minimal	Satisfacotry & Subjective
KUS	Literary	British English	Assembly Language	90 %	Minimal	Good & Statistical
MITS	Literary	American English	Unknown	10%	Maximal	Unknown
NRLS	Literary	American English	SNOBOL	98%	Minimal	Good & Statistical
TIFRS	Literary	English, Tamil and Hindi	FORTRAN & SLIP	25%	Nil	Good & Statistical

Table 2.1 (b) Summary of Software features of some existing Text-to-Speech Conversion Systems.

CHAPTER 3

PATTERN RECOGNITION AND PREPROCESSING

3.0 Pattern Recognition within a Text:

In any general TSC system, a textual input will have to be handled by a pattern recognition block (block A in figure 2.1). This will further involve preprocessing of the input text and the normalization to a single code scheme for the conversion into speech.

The patterns to be recognized may be pictures and script-related patterns. Of these, we restrict ourselves to the script-related problems,and in this chapter, we will assume that the pattern recognition block A has produced the necessary output from an input text to our system. Hence, we will consider how to transform these outputs into suitable code (phonetic code or alphabetic code as the case may be)for the purposes of speech synthesis.

First, we will consider the problems of script-related patterns, and then proceed to propose solutions to some of these problems.

The patterns to be transformed may be one or more of the following types:

1. Numerals in a text,
2. Upper-case vs lower-case letters in a text,
3. Abbreviations and pseudo-names,
4. Press-style conventions, such as italics, bold face or other related type faces,
5. Different sizes and shapes of type faces conveying different information, such as foot notes, bibliography, etc.,
6. Alphabets of different languages in a text, such as Greek, Latin, etc.,
7. Mathematical symbols and formulae,
8. Special punctuation marks, such as quote, braces, brackets, etc.

3.1 Numerals:

Almost all texts will have some numerals embedded in them. This may be from simple one to four digit integers to represent date, page, etc., to complex number representation as in mathematics. Depending upon the subject matter it will be possible to assign a probability for the occurrence of a particular kind of numeral, such as integer, real, fraction, to a text. This can be helpful in providing proper algorithms to handle such numeric patterns in a text. However, the output may be different depending upon the usual conventions in a subject matter. For example, numerals may be spelled character by character or expressed in terms of units, such as million, thousand, hundred and tens or in similar regional conventions is a matter of choice. Hence, totally independent pronounciation for numerals will depend upon many factors and standardization may be helpful, such as the spelling of digit by digit from left to right as done in some calculators [COM 1975]. If a convention is made available in the area of specialization, the system should be tuned to adopt it in the pronounciation scheme, thus satisfying the local needs.

In the algorithm-section of this chapter, a simple algorithm to handle simple numeric patterns is given illustrating our approach.

3.2 Upper vs Lower-case letters:

In languages where conventions exist for using different types of letters, such as Upper-case and Lower-case letters, the purpose of such differentiation should be reflected in the pronounciation of words and sentences. For example, in English, the Upper-case letters are used in the following contexts:

1. to begin a sentence,
2. to begin a proper name, such as a personal name or place name,
3. to signify an abbreviation or pseudoname,
4. and to signify that the word under consideration is to be emphasized.

Computationally, since the code values are different for Upper- and lower- case letters, it is necessary to normalize them at the pre-processing level.

At the algorithmic level, the personal names and pseudonames can be handled via data bases (dictionary setup and lookup schemes)[McI 1974]. Another data base would be required to handle abbreviations, since abbreviations used in different disciplines would have different connotations. This would involve tuning the system in the initial stage, depending upon the input text. In all other cases, either a reduction in the code or a spell character-by-character scheme would be necessary. In a later discussion, an algorithm will be provided to perform such a function.

3.3 Press-style conventions:

In any language the preparation of printed text is subjected to certain rules and editorial conventions. These rules are called press-style(or print-style or house-style) in the literature [MAN 1975]. However, press-style is different from any style of writing, in particular the way an author might write, and even an individual author's style cannot violate the press-style. It is customary to refer to format in presenting input and output from a computer program. This itself is a subset of press-style. The problem in press-style may be classified as problems of type faces and thier sizes, the type shapes, special punctuations and paragraphing conventions.

3.3.1 Type faces and their sizes:

In press-style,every character has a type face and size. These are specified in terms of 'points' or units of size equivalent to 1/72 of an inch and the type may vary from five to seventy two points per character. Different shapes of type faces are used to convey different information. For example,italic, bold face and decorative forms(or quaint characters [MAN 1975]) can be used in different contexts to convey a particular meaning or place emphasis on certain textual material.

The following rules for using different sizes of characters come from A Manual of Style [MAN 1975, p 442] ;

1. When an extract (quote from another source) is used, use a type face smaller by one point with respect to the other characters in the body of the text.
2. For foot notes, the type face should be at least two sizes smaller than that of the body of the text, but not less than 8 points.
3. When using indention (or indentation) use different measures of idention, but the type sizes are identical (i.e., the number of blank spaces from the left margin can be used to convey certain information and in such cases normal size of type face can be used).

Thus, there exists a problem in recognizing the various type sizes and indention at the preprocessing level in a TSC system, since computationally, the code values for type-faces will be different.

3.3.2 Type-shapes:

The second problem in the press-style is type-shapes. A Manual of Style [MAN 1975,p 459] specifies at least 10 types of styles in which each character may have different type-shape and size. Of these, we consider only two, italics and bold face (or script). Italics can be used in 32 different contexts according to press conventions [MAN 1975,p 532], while bold face or script is used mainly for bibliographic information, index and for mathematical symbols (or as variable letters).

The main use of the italics and bold face however, is placing emphasis on a word (or group of words) since slighly exaggerated stress on the word(s) represented by italics may be helpful in the pronounciation of such words.

We propose that the specific application of press-style rules should be considered in a TSC system and the speech output should somehow reflect this fact. In our proposed algorithm we will take care of these rules.

3.3.3 Mixed Alphabets:

For printed text in English, when alphabets of other languages, such as Greek, German, Russian are encountered, the computational code is different. Alphabets of many languages are used for the following reasons:

(a) Latin names are used in Medicine, Biology and Botony.

(b) Greek symbols are used in Mathematics and Physical Sciences.

(c) French words are used in cosmetics and food preparations etc.

It appears, in general, that foreign words in English are area-oriented and therefore become area-specific involving pronounciation different from those of normal English words. In handling such words, it is necessary to tune the particular language or enter into a separate data-base to aid in the pronounciation of such words. As we have mentioned earlier, words also differ in type-face, shapes and sizes in a text in addition to different linguisiic codes (i.e., words from different languages). These should be handled at the preprocessing level in a TSC system. One way to handle such preprocessing would be to provide a 4-tuple (Page, Line, Word, Flag) at the beginning of a text which could be utilized during the preprocessing stage. This will be discussed later.

3.3.4 Mathematical Symbols:

By far the most difficult part of a TSC system is the design of an algorithm to handle mathematical symbols and formulae in a text. Isolated mathematical symbols can be spelled by a dictionary lookup after being recognized by the pattern recognizer. This would require a specific data base as mentioned in section 3.3.3. If more than one symbol is involved, and a formula is encountered, it becomes difficult to pronounce the symbols and formula by any simple rule scheme. For example, in a typical definition-dictionary for mathematical systems, some definitions run for number of pages [CRC 1959]. Various definitions for the same symbol or fomula can complicate the meaning. For example, the symbol $\bar{x}$ can be interpreted as 'mean' value of x in statistics and a mere x bar, a variable different from a symbol x. With different data bases for different subject-areas, different interpretive rules can be easily formulated. As far as the pronounciation of mathematical and other formulae are concerned, it is possible to propose a standard temporary measure, which in the long run can become necessary for the pronounciation of such systems.

The complexities of mathematical formulae in the printing industry computer based typesetting and CRT display have been reported in the literature [MAN 1975; KER 1975; MAR 1967].

As observed by Kernighan [KER 1975], two major difficulties are encountered with respect to mathematical formulae. They are:

1. The text involves a multiplicity of characters, sizes, and fonts.

   For example, the expression :

$$\lim_{x \to \pi/2} (\tan x)^{\sin 2x} = 1$$

requires an intimate mixture of Roman, italic and Greek letters in three sizes and one or more special characters (note: in our reproduction due to limitation of type-writer, we have failed to show the differences of type shapes, sizes discussed above).

2. The text involves two-dimensional mathematical characters with subscripts, superscripts, braces, radial line drawings and positional problems.

   For example,

$$a_o + \cfrac{b_1}{a_1 + \cfrac{b_2}{a_2 + \cfrac{b_3}{\cdots\cdots\cdots\cdots}}}$$

illustrates such a problem.

Computer typsetting attempts by Kernighan et al [KER 1975] indirectly suggest to us how to generate a description for a given formula. The suggestions are :

1. A description for a mathematical formula can be in fragmentary English. That is, English sentences peculiar to mathematics are either ungrammatical, or incomplete sentences. For example, the input command for a typical typesetting formula [KER 1975] will be :

   SUM FROM I = 0 TO INFINITY X SUB I = PI OVER 2

   produces:

$$\sum_{i=0}^{\infty} x_i = \pi/2$$

   whereas, in ordinary English, the command might be:

   FORM THE SUM OF THE VARIABLE X WITH SUBSCRIPT I, WHERE THE VALUE OF THE SUBSCRIPT I IS FROM ZERO TO INFINITY.

2. The description of a formula is linear and one-dimensional and the output is two-dimensional. Hence, proper ordering of the input words can take care of the intended message in the output.

A fragmentary grammar has both production rules and also restriction rules, but a general grammar has only production rules. Thus, in the example above, an ambiguity is detected where the term PI over 2 is used. The question is whether this term is the property of the subscript index or the equivalence for the sum of the variable x_i. It is obvious that certain restrictions are placed on the interpretation of the input to prevent misinterpretation. Thus, the second observation is also simultaneously satisfied. Fragmentary grammars (Or Sublanguages) have been investigated and reported in the literature [SAG 1972a, 1972b, 1975a, 1975b; GRI 1973]. We note thus, in a TSC system fragmentary transformation of mathematical formulae will be produced and this should not be subjected to further analysis later.

Speech output corresponding to a given mathematical formula in a text will depend on the following:

1. How is the formula to be divided into its building blocks and represented as a linear string?
2. How is the output to be produced so as to make the grouping clear and unambiguous? That is, should we generate necessary silence duration between the fragmentary words in a formula that can distinguish the output of this fragmentary words from other ordinary English words and sentences?

If we assume the break down of the formula in the reverse order of its construction, then we can determine how to divide the formula into its constituents. Kernighan [KER 1975] observes : " Equations are pictures, constituting a set of 'boxes', pieced together in various ways. For example, something with a subscript is just a box followed by another box moved downward and shrunk by an appropriate amount. A fraction is just a box centered above another box, at the right altitude, with a line of correct length drawn between them". A grammar to generate mathematical formulae from a give input description is reported in the literature [KER 1975]

In a TSC system, since the reverse of the construction, the question is, given a mathematical formula, how do we obtain a closed description in a natural language suitable for speech synthesis?

One possible approach to handle a mathematical formula in a TSC system is given below.

3.3.4.1 Vector Representation of Mathematical Symbols:

Consider a mathematical formula as a set of elements of a linear string. Let each element be stored in an array. Accordingly, a mathematical symbol may be represented as a vector of three elements in the following order : hat-script, base and shoe-script. A typical vector representation of a mathematical symbol is given in figure 3.1.

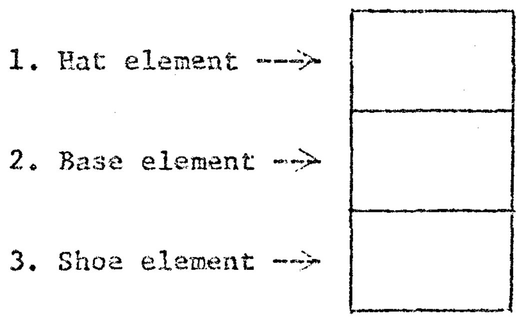

Figure 3.1 Vector representation of a typical mathematical symbol.

In an actual vector implementation scheme, the appropriate elements themselves can be stored as three elements of a vector. For example, a typical representation of a mathematical symbol for summation is shown in figure 3.2.

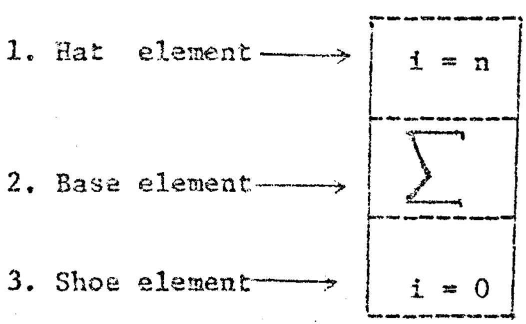

Figure 3.2 Vector representation of the mathematical symbol $\sum\limits_{i=0}^{i=n}$.

We assume that recovering the components of a mathematical symbol can be done through suitable pattern recognition algorithm. Even the variations of a mathematical symbol can be handled in such a way that they can be represented as a vector. An alternative approach to represent a mathematical formula and its components will be to use a list structure, as in given in Clapp et al [CLA 1966].

3.3.4.2 Matrix Representation of Mathematical Variables:

A mathematical formula consists of a mathematical symbol, at least one variable (mathematical variable) and an optional mathematical operator, such as + ,- or /. The elements of mathematical formuale are in juxtaposition. In these, however, a variable could be either single dimensioned, such as, in an alphabet, or two-dimensioned, as in a subscripted/superscripted variables. A variable can be subscripted, superscripted, with or without a hat, or shoe-script. Therefore, while a simple variable symbol can be represented as a special case of a general variable symbol, (in terms of the displaced symbols attached to it in the sense of Kernighan [KER 1975]), a general variable symbol can be represented as elements of a matrix. For example, the variable:

$$\bar{x}_{\dot{A}}^{2}$$

may be composed and represented as elements of a (3x2) matrix as in figure3.3.

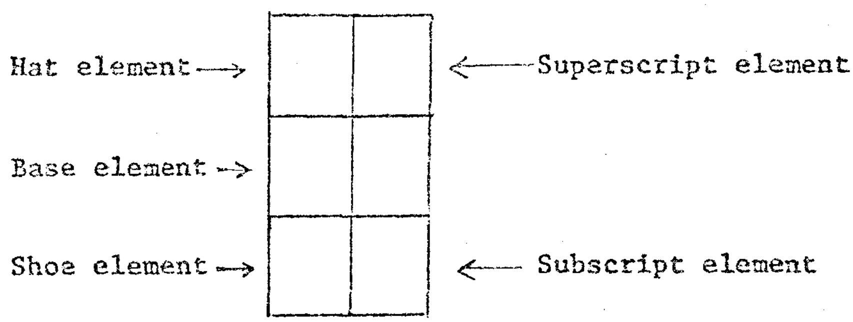

Figure 3.3 Matrix representation of a typical general mathematical variable symbol.

Similarly, a Tensor variable symbol can be broken down into elements of a (3x3) matrix.

3.3.4.3 <u>Conversion to Description:</u>

When a mathematical formula is to be converted into ordinary orthographic form (or directly into phonetic form) suitable for speech synthesis, the question is which representation (or description) will be acceptable? That is, when a person listens to a description of a mathematical formula, will the person be able to reconstruct the description back to its original form? Omalley et al [OMA 1973] have reported how listeners identify algebraic expressions when a 300 millisecond duration for paranthesis is given. However, no answer can be given on all aspects of the above question. So, we propose the following scheme:

1. A vector representation of a mathematical symbol should be linearized as in figure 3.4.

2nd element # 3rd element # 1st element #

Figure 3.4 Linear representation of a vector for a mathematical symbol.

The symbol # represents a potential pause (or silence gap) allowable in a linear string to separate the components suitably. Putting it differently, the linear representation, in terms of a pattern will be :

Base element# Shoe-script #Hat-script #

For example, if we represent $\sum$ as SUM, = as EQUALS, then

$\sum_{i=0}^{i=n}$ will be converted into the following description:

SUM # (FROM) # I EQUALS ZERO # TO # I EQUALS # N #

Similarly, $\int_{z}^{a}$ will become : INTEGRATE # BETWEEN # THE LIMITS # ZEE # TO A #

A vector representation of a simple variable appendable to a mathematical symbol will also be converted in the same manner.

For example,

$\bar{x}$ will become X BAR

$\ddot{x}$ will become X DOUBLE DOT

$\tilde{x}$ will become X TILDA etc.

(Note: we are not interested here on the interpretation of a variable, such X BAR is a description, than a representation for 'mean of the variable x'.).

2. When a mathematical symbol and a simple variable are involved, the conversion will be as follows:

BASE Mathematical Symbol # Variable Symbol # Limits of Base Mathematical Symbol (range of bounds) #

where the variable symbol will be expanded as above.

For example,

$\lim_{x \to 2} 3x$ will be converted into a following description in our system:

LIMIT # THREE TIMES # X # AS # X TENDS # TO# TWO #

When two-dimensional variable symbols are encountered along with mathematical symbols in an average size mathematical formula, the conversion can be done as follows:

Base Mathematical symbol # Variable symbol # Shoe-script of Variable symbol # Hat-script of Variable symbol # Subscript of Variable symbol # Superscript of Variable symbol# Lower range of mathematical symbol # Upper range of mathematical symbol #

For example,

$$\sum_{i=0}^{i=n} \bar{x}_i^k$$

can be written as follows:

SUM # X BAR # SUB I # RAISED TO POWER # K # FOR # I EQUALS #ZERO# TO # I EQUALS N #

where this generated description can be synthesized into speech later.

When more than one identical mathematical symbol is involved in a formula with respect to a particular variable, such as double subscripted variables, the usual algorithm for pronounciation appears as follows:

1. Count the number of times the same mathematical symbol is observed in the formula.
2. Spell the count.
3. Spell once the mathematical symbol (or provide the equivalent description).

4. Spell the base variable symbol (2nd element in the variable symbol matrix).

5. Spell the other elements of the variable, such as hat-script from the matrix representation as in figure 3.3, then the first subs- script, then the second subscript and so on.

7. Spell the 1st range of the mathematical symbol for the 1st subs- script of the variable.

8. Spell the 2nd range of the mathematical symbol for the subscript of the variable and proceed backward until all the ranges have been spelled out.

For example,

$$\sum_{j=1}^{j=9} \sum_{i=1}^{i=n} x_{ij}$$

can be pronounced as:

DOUBLE SUM # X SUB I # COMMA # J # I FROM ONE # TO # N # AND# J FROM ONE # TO # NINE #

The problem of representing a mathematical formula as a description is more complex than for simple cases we have considered so far. We will not go into any further details.

We now give some algorithms to preprocess a part of a text, restricting ourselves to the problems we have discussed in previous sections.

3.4 Some Proposed Algorithms:

The aim of the present section is to provide individual algorithms for each problem we have discussed above at the preprocessing level without giving the implementation details, such as storage requirements and computation time. Combining these algorithms into one algorithm would only create a single complicated procedure. In doing this, we would lose the parallel nature of the algorithms. Unfortunately, the parallelism of the algorithms introduces complexities in understanding the execution of the algorithms themselves.

To reduce the problem of storage and processing time of different words in an input text, such the problem of type face, size, numerals, special mathematical symbols and special punctuation symbols, we propose a table setup which will be a part of a text. This table will be called the Preprocessing table. We assume that this can be provided by the publisher at the time of publication of a text. Otherwise, the required information must be computed first before the text can be converted into speech.

3.4.1 Preprocessing Table:

A preprocessing table entry is a 4-tuple [PAGE,LINE,WORD,FLAG].

PAGE indicates the page in which preprocessing is required; LINE indicates the line in which the preprocessing is required; WORD indicates the word to be preprocessed and the type of preprocessing is indicated by FLAG.

For example,

45 15 4 6

might mean that 45th page, 15th line, 4th word (from left of the line) and flag value 6 denoting the presence of GREEK letter(s). The flag value 6 specifies that the preprocessing algorithm has to invoke the subalgorithm SPELL at page 45.

We assume that when a new page of input text is read, the first task of the preprocessor will be to compare the current page number with the preprocessing table entry PAGE and decide whether the current page requires any preprocessing or not. When a page number matches with the page number in the preprocessing table entry PAGE, then the current line number is compared with the line number of the preprocessing table entry LINE and so on. Without going into further details, such a preprocessing table will be assumed in the following algorithms. (For a pure mathematical text alternative arrangements should be worked out).

3.4.2 Algorithm NUMERAL:

3.4.2.1 Purpose: To convert numerals within a text into alphabetic form.

3.4.2.2 Method 1:

Isolate the individual digits of an input numeral and spell.

Step0 : Scan an input word and if it is not numeral continue to search other input words and no numeral is found terminate the algorithm.

Step1 : If the input numeral contains a decimal point Invoke DECIMAL-PART.

Step2 : Invoke INTEGER-PART.

Step3 : Jump to Step0.

(End of algorithm)

//INTEGER-PART//

CHARACTER ← 0

Step0 : CHARACTER ← CHARACTER + 1

Step1 : Read a character. If the character is a blank, terminate the algorithm.

Step2 : If character is a comma, ignore it and jump to Step0.

Step3 : Spell the character. Output pause #. Flag the word. Jump to Step0.

(End of algorithm)

// DECIMAL-PART//

CHARACTER ← 0

Step0 : CHARACTER ← CHARACTER +1

Step1 : Read an input character.

Step2: If character is a blank, terminate the algorithm.

Step3: If character is a period, output POINT , output pause #, flag the word and jump to Step0.

Step4: Spell the character, output pause #, flag the word and jump to Step0.

(End of algorithm)

3.4.2.2 Method 2:

Ignore comma in a numeral and by successive integer division obtain the digits and concatenate the spelled digit with the divisor unit and repeat the operation till all digits are spelled out. This will work only for INTEGER numbers. Thus, for example, 1009 in this method will be spelled as : ONE # THOUSAND # NINE #.

Observation

Even if the digit contains the comma to signal digit grouping in terms of units (million, thousand, hundred or tens) method one will be easy to implement and will be acceptable to all listeners. Spelled digits with appended units as given in method 2 above, requires storage and increased processing time. In our proposed algorithm TESCON described in chapter 5, we assume method-1 only.

3.4.3 Algorithm STANDARDIZER:

3.4.3.1 Purpose: 1. To convert upper-case letters to lower-case when necessary.

2. To convert Italics/ Bold face letters to normal size letter.

3. To convert simple mathematical symbols and formulae into an equivalent description.

3.4.3.2 Method: This algorithm invokes several subalgorithms. When a flagged page is encountered, each line is checked and then the words to be normalized are checked for proper preprocessing and so on.

Certain details, such as the input, output handling both unflagged and pages are given in Chapter 5. In the following setup, PAGER is a subalgorithm invoked in the main program which when called reads a new page for a given input text.

//PAGER//

[SAVE PAGE-FLAG]

PAGE-FLAG $\longleftarrow$ OFF (flag indicates preprocessing)

Step0 : Read current page number.

Step1 : If current page is blank, terminate the algorithm.

(All pages are assumed to be numbered)

Step2 : Check if current page is flagged under preprocessing table

parameter PAGE.

If true, PAGE-FLAG <--- ON

Step3 : Invoke LINE-NUMBER.

(End of algorithm)

// LINE-NUMBER//

[SAVE PAGE-FLAG]

LINE-FLAG <-- OFF

LINE COUNT <- 1

Step0 : Scan the lines and find the next line. (i.e., the line where, non-numeric, non-blank character begins a page is a first line.).

Step1 : If LINE-COUNT > Maximum lines on a page, Invoke PAGER.

Step2 : If PAGE-FLAG is OFF, jump to Step4.

Step3 : If current line is flagged under preprocessing table parameter LINE, set LINE-FLAG <-- ON

Step4 : Invoke WORD-FINDER.

(End of algorithm)

// WORD-FINDER//

[SAVE : LINE-FLAG1, WORD-COUNT, WORD-COUNTED, LINE-COUNT,
LINE-FLAG]

LINE-FLAG $\longleftarrow$ LINE-FLAG1

CHARACTER-COUNT $\longleftarrow$ 0

Step0 : If LINE-FLAG is <u>ON</u> , Invoke LOOP.

Step1 :If CHARACTER-COUNT > Maximum number of characters in a line,
LINE-FLAG $\longleftarrow$ ON ; Invoke LOOP.

Step2 : Scan a character on the input line.

Step3 : If a character is a blank, increment the CHARACTER-COUNT by unity, and jump to Step2.

Step4 : If a character is a punctuation [. , ; : ? /], store it, increment WORD-COUNT by unity; increment CHARACTER-COUNT by unity and jump to Step2.

Step5 : Store the non-blank, non-punctuation characters in a word; increment the CHARACTER-COUNT by the total number of characters stored in the current word, increment the WORD-COUNT by unity; jump to Step2.

(End of algorithm)

// LOOP//

[SAVE : LINE-FLAG1, WORD-COUNT, WORD-COUNTED, POINTER]

Step0 : Set the pointer to a current word of a line.

Step1 : If pointer value exceeds the total number of words in a line (WORD-COUNT's value),

WORD-COUNTED ⟵ WORD-COUNTED + WORD-COUNT

LINE-FLAG1 ⟵ OFF

Invoke LINE-NUMBER.

Step2 : If current line is flagged (LINE-FLAG ⟵ ON), and if current word is flagged (when compared with the preprocessing table entry parameter [WORD],) Invoke NORMALIZER.

Step3 : Invoke UPPER-CASE.

(End of algorithm)

[Note: WORD-COUNT accounts for the total number of words in a line and WORD-COUNTED gives the value of total number of words processed so far].

// NORMALIZER//

[FLAG provides the value of the 4th parameter in the preprocessing table entry]

Step 0 : If FLAG value is 1, Invoke QUOTE.

Step 1 : If FLAG value is 2, Invoke FOOT-NOTE.

Step2 : If FLAG value is 3, Invoke ITALICS.

Step3 : If FLAG value is 4, Invoke MATHS.

Step4 : Invoke WORD-FINDER.

(End of algorithm)

//UPPER-CASE//

[SAVE : POINTER]

Step0 : Get a character from the input word.

Step1 : If the character is not an UPPER-case letter,

POINTER $\leftarrow$ POINTER + 1; Invoke LOOP.

Step2 : If (character+1) is not an UPPER-case letter, Invoke PROPERNAME.

Step3 : Invoke ABBREVIATION.

(End of algorithm)

// PROPERNAME//

[SAVE : POINTER]

Step0 : Hash the complete input word with proper-name table.

Step1 : If no match is found, jump to Step3.

Step2 : Replace the current input word with its equivalent table entry and flag the word; POINTER $\leftarrow$ POINTER + 1 ; Invoke LOOP.

Step3 : Replace the upper-case letter by its equivalent lower-case letter; POINTER $\leftarrow$ POINTER + 1 ; Invoke LOOP.

(End of algorithm)

// ABBREVIATION//

[SAVE : POINTER]

Step0 : Hash the complete input word with abbreviation-table.

Step1 : If no match is found, Invoke SPELL.

Step2 : Replace the abbreviation by its equivalent table entry; flag the word;

POINTER $\leftarrow$ POINTER + 1

Invoke LOOP.

(End of algorithm)

//SPELL//

[SAVE : POINTER]

CHARACTER $\leftarrow$ 1

Step0 : Read a character from the input word.

Step1 : If the character is blank, POINTER $\leftarrow$ POINTER + 1, Invoke LOOP.

Step2 : Hash the character with proper-name table and copy the corresponding entry, flag the word, introduce pause #.

Step3 : CHARACTER $\leftarrow$ CHARACTER + 1

Step4 : Jump to Step0.

(End of algorithm)

// QUOTE //

[SAVE : POINTER]

Step0 : Output # QUOTE BEGINS # , flag the word.

Step1 : Get the next character.

Step2 : If the character is blank, jump to Step7.

Step3 : Get the FONT-size.

Step4 : If FONT-size is equivalent to normal size, jump to Step1.

Step5 : Replace non-standard FONT-size by normal size, jump to Step1.

Step6 : Output # QUOTE ENDS 3, flag the word.

Step7 : POINTER $\leftarrow$ POINTER + 1

Step8 : INVOKE LOOP.

(End of algorithm)

[Note: Even within a quote there may be words requiring preprocessing. In QUOTE a few more steps can take care of them. We have left them out here].

//FOOT-NOTE//

[SAVE : POINTER]

COUNT $\longleftarrow$ 0

Step0 : Output # FOOT NOTE # , flag the output word.

Step1 : Get the next character of the input word.

Step2 : If character is blank, POINTER $\leftarrow$ POINTER + 1, Invoke LOOP.

Step3 : If the character is numeral, COUNT $\leftarrow$ numeral (current character);
Invoke SPELL-NUMERAL.

Step4 : If line-count exceeds maximum lines in a page, FOOT-NOTE-FLAG $\leftarrow$ ON;
Invoke PAGER.

Step5 : If terminator is encountered, Output #FOOT NOTE OVER #,
Flag the word POINTER $\leftarrow$ POINTER + 1; Invoke LOOP.

Step6 : Replace the current character FONT-size by normal character FONT.

Step7 : Jump to Step1.

(End of algorithm]

[Note: As mentioned earlier, other preprocessing may be involved here too].

// ITALICS //

[SAVE : POINTER]

DOUBLE-STRESS—FLAG $\leftarrow$ ON [= HEADER value in ANALYSER given in chapter 5]

Step0 : Get the next character.

Step1 : If the character is blank, or a standard character,
POINTER $\leftarrow$ POINTER + 1 ; Invoke LOOP.

Step2 : Replace italic/ bold face character by a normal character.

Step3 : Jump to Step0.

// MATHS //

[SAVE : POINTER, COUNT]

COUNT $\leftarrow$ 0

Step0 : Count the identical mathematical symbols.

Step1 : COUNT = total count of mathematical symbols.

Step2 : If COUNT = 0, Invoke VARIABLE.

Step3 : Invoke SPELL-NUMERAL if COUNT $\geq$ 1, and flag the output words.

Step4 : Hash the mathematical symbol with mathematical-symbol table and copy the equivalent description, flag the word, jump to Step6.

Step5 : Invoke VARIABLE if mathematical symbol is absent.

Step6 : POINTER$\leftarrow$POINTER + 1, Invoke LOOP.

(End of algorithm)

// SPELL-NUMERAL//

[SAVE : POINTER]

Step0 : Hash the value of count with spell-numeral table.

Step1 : Copy the equivalent description on the output file.

Step2 : Output pause #.

Step3 : Flag the word.

Step4 : POINTER$\leftarrow$POINTER + 1 ; Invoke LOOP.

(End of algorithm)

// VARIABLE //

[SAVE : POINTER ,COUNTER]

COUNTER ← 0

Step0 : COUNTER ← COUNTER = 2 (this gives the 2nd element - base element - in a vector)

Step1 : If base element is blank, POINTER ← POINTER + 1, Invoke LOOP.

Step2 : Output pause #.

Step3 : Hash the base element with proper-name table and copy its equivalent form, flag the word, COUNTER←COUNTER +1 .

Step4 : If the current element is not a blank, VARIABLE-FLAG ON, WORD-FLAG ← ON, Invoke SHOE-SCRIPT.

Step5 : COUNTER ← COUNTER-2(1st element is given)

Step6 : If element is not a blank, VARIABLE FLAG ON, WORD-FLAG ON Invoke HAT-SCRIPT.

Step7 : COUNTER ← COUNTER+4 (5th element).

Step8 : If element is blank, COUNTER ← COUNTER + 1.

Step9 : If element is not a blank, Invoke SUBSCRIPT.

Step10: COUNTER ← COUNTER - 2

Step11: If element is blank, POINTER←POINTER + 1, Invoke LOOP.

Step12: Invoke SUPERSCRIPT.

(End of algorithm)

// SHOE-SCRIPT //

[SAVE POINTER , COUNT]

Step0 : If VARIABLE-FLAG is ON and the variable is not blank, jump to Step6.

Step1 : If COUNT = 1, back tract to the nearest mathematical symbol

Step2 : If no mathematical symbol is found, POINTER ← POINTER + 1, Invoke LOOP.

Step3 : If shoe-script element is blank, jump to Step7.

Step4 : Output pause #.

Step5 : Spell the shoe-script character by character, flag the words, Introduce pause between any two words.

Step6 : If VARIABLE-FLAG is ON, Invoke HAT-SCRIPT.

Step7 : If COUNT > 1, COUNT ← COUNT - 1.

Step8 : If COUNT =0, POINTER ← POINTER + 1, Invoke LOOP.

Step9 : Back tract one step and jump to Step6.

(End of algorithm)

// HAT-SCRIPT//

[SAVE POINTER, COUNT]

Step0 : If VARIABLE-FLAG is ON, and the variable is not a blank, jump to Step5.

Step1 : Back tract to the nearest mathematical symbol.

Step2 : If no mathematical symbol, POINTER POINTER + 1, Invoke LOOP.

Step3 : If HAT-SCRIPT is blank, jump to Step6.

Step4 : Output pause #.

Step5 : Hash the hat-script character by character with symbol table, copy the description, flag the words.

Step6 : POINTER $\leftarrow$ POINTER + 1, Invoke LOOP.

Step7 : If COUNT $>$ 1, COUNT $\leftarrow$ COUNT - 1 .

Step8 : If COUNT = 0, jump to Step6.

Step9 : Back tract one step and jump to Step5.

(End of algorithm)

// SUBSCRIPT //

[SAVE : POINTER, COUNTER]

Step0 : Get the left-most subscript.

Step1 : If subscript is blank, COUNTER $\leftarrow$ COUNTER -2 , Invoke SUPERSCRIPT.

Step2 : Output pause #.

Step3 : Spell the subscript, flag the word.

Step4 : Left-shift the subscript.

Step5 : Jump to Step0.

(End of algorithm)

// SUPERSCRIPT //

[SAVE : POINTER]

Step0 : Get the left-most superscript.

Step1 : If superscript is a blank, POINTER←POINTER + 1, Invoke LOOP.

Step2 : Spell the superscript, flag the word, output pause #.

Step3 : Left-shift the superscript and jump to Step0.

(End of Algorithm)

CHAPTER 4

LINGUISTIC ANALYSIS OF INPUT TEXT

4.0 MOTIVATION

In natural speech communication, a listener decodes a perceived speech signal on the basis of certain acoustic cues present in the signal. The encoder of the signal (speaker) provides some of the perceptually significant cues, such as duration of an utterance, pauses in between a part of an utterance, stress in some portion of an utterance and the clear articulation of speech sounds [KLA 1976; LIB 1968; SCH 1968; UME 1976; VEN 1970; WHI 1976]. In addition to decoding the incoming speech signals according to perceived acoustic cues, a listener employs also other factors, such as subject matter, familiarity with the speaker, context of the conversation and related matters. Therefore, in a TSC system, it will be necessary to introduce at least a minimal set of acoustic cues that are not explicitly given on a printed text, such as duration of individual speech sounds, stress, pause and letter-to-sound rules of the input alphabets.

In this chapter, we briefly investigate some of the parameters of speech, namely duration, stress, pause and letter-to-sound rules. Our main concern is to use the linguistic information computable for an input English text, rather than the theoritical investigation of such linguistic details. Further details are provided in the references cited in this chapter.

4.1 DURATION:

4.1.1 Definition:

Every speech sound that is to be perceived by a normal human being has an inherent duration corresponding to the duration of its ideal articulation in real time. This inherent duration of a speech sound is called the steady-state duration of a phoneme [RAM 1973].

Operationally, Klatt [KLA 1976] defines the duration in the acoustic domain as the duration of stops (such as p, t, k, b, d, g) corresponding to the duration of the closure for stops, while for fricatives (such as f, z, s, sh) the duration corresponds to the interval of turbulent friction noise above some threshold (or to changes in the voicing source if no friction energy is visible in the spectrum) and so on.

In dynamic speech, we can compute the dynamic duration of speech sounds from the steady-state duration using the following formula given by Klatt [KLA 1976] :

$$D_j = K * (D_i - D_{min}) + D_{min}$$ where,

D_j is the dynamic duration of a given speech sound,

D_i is the assigned steady-state duration,

D_{min} is the (average) absolute minimum duration for a satisfactory articulation of a speech sound and K is a factor. For a duration shortening rule K stands for the relationship $0 \leq K \leq 1$, while for a duration lengthening rule, $K > 1$.

We notice that the systems investigated in chapter 2, have no such rules for the computation of dynamic duration of speech sounds. Only the TIFR system has this facility. Computationally, the computation of the contexts, such as the preceding and following speech sounds, whether a given sound is stressed or not, is necessary in a TSC system. How various linguistic contexts affect the duration of speech sounds have been reported in the literature [HUG 1974a,1974b; KLA 1976; UME 1975, 1976]. In this thesis we propose the following rules for duration, some which are utilized in our algorithm ANALYZER.

4.1.2.1 Vowel Duration:

(a) The duration of a vowel (such as a,e,i,o,u,y) in English, in a word-final syllable before a pause (#) is increased by 100% (twice its steady-state duration).

(b) The vowel duration in a word non-final syllable and before a non-pause is shortened. (This will not be included in our algorithm).

(c) Positional factors that affect the vowel duration include a consonant after a vowel: (not incorporated in our algorithm ANALYZER)

(i) Vowel duration is shortest before a voiceless stop, such as p, t, k.

(ii) Vowel duration is longest before a voicelss fricative, such as f, s, sh.

(d) A vowel that is stressed is longer in duration (roughly one and half times longer than the steady-state minimal duration).

4.1.2.2 <u>Consonantal Duration</u>:

(a) Position of a consonant relative to the stressed syllable and word, or sentence boundary increases the consonantal duration by about 50% of the steady-state duration of the consonant.

(b) When a consonant is both preceded and followed by other consonants, the middle consonant is reduced by about 50% in its steady-state duration and in an uni-consonantal context (either preceded or followed by a consonant), the reduction in the duration of the consonant under study is about 25%.

Consonantal duration modifications are not implemented in our algorithm ANALYZER.

4.1.3 Duration of function words:

The duration of function words, such as of, the, an, at, in, is the sum of the minimal duration of the individual constituents (i.e., the vowel's and consonant's in it).

While Umeda [UME 1976] provides a detailed analysis of the durational aspects of speech sounds in a Text-to-Speech Synthesis context, these studies are based only on three or four speakers and therefore the inclusion of such details in a TSC is questionable. Klatt [KLA 1976] considers many other factors that affect the duration of dynamic speech sounds. Computation of all these contexts also requires large storage and processing time.

In order to incorporate the dynamic duration of speech sounds in a TSC system, the following is proposed:

(a) The specification of the dynamic duration of speech sounds should be part of the selection rules for every speech sounds. For example, let @ represent a written symbol in a language L, and $ be its equivalent sound code. Let + represent the increment in duration and (-) represent the decrement in duration where either of the symbols + or - can follow $. Let N represent the fractional numeric value of the duration used in the increment or the decrement of the duration of a speech sound. Let & represent a definable context before which $ can occur.

Then we can define the following two rules: ($\rightarrow$ represents rewriting)

(i) <u>Context Sensitive Rule</u>:

$$@\ \& \longrightarrow \$ + N \quad (\text{or} \quad \$ - N)$$

where $N \neq 0$, and the inherent duration of $ is assumed to exist.

(ii) <u>Context Free Rule</u>:

$@ \longrightarrow \$$ where the duration of the $ is assumed to exist and and is minimal.

Notice that we have only one rule in our TSC system, namely the duration incremental rule (DIR). The decrement is implied by the minimal duration of a speech sound.

(b) A rule cannot be selected, that is, the duration modification cannot be computed, unless the linguistic context is computed. For example, the presence or absence of a pause indicating a phrase boundary, such as <u>Having come # he thought # that it was nice.</u>, where # indicates possible phrase boundaries, number of syllables in an input word to determine the stress assignment, are all computable. After preprocessing, the input text in the Pattern Recogntion block (block A in figure 2.1), the linguistic block (block B in figure 2.1) must compute the contexts and other linguistic information. The necessary minimal analysis of an input text as an algorithm is proposed in chapter 5.

4.2 Stress:

4.2.1 Definition:

The 'prominence' that is perceivable in any spoken syllable is called stress. Acoustically, stress is realized through interacting parameters, such as duration, intensity and fundamental frequency. Thus, a stressed syllable will be usually high in voice-pitch, long and loud [GAI 1967].

Stress increases the vowel duration (cf. 4.2.1 (d)). The increase of duration in a stressed vowel is one and half times the normal duration of a vowel.

4.2.2 Types of Stress:

There are three types of stresses in English. The first one is called the lexical stress, the second is phrase stress and the third is emphatic stress.

4.2.2.1 Lexical Stress:

Following the method for constructing standard English dictionary, such as Oxford Dictionary, Random House Dictionary, the simplest way to solve the problem of lexical stress for written words is the creation of a dictionary having hand-coded phonetic entries and stress mark for each entry. This approach has been utilized in various ways in some of the existing systems [COK 1973; TER 1968; UME 1975; ALL 1976].

The second general method is the assignment of stress by rules. Basically a set of rules, such as rules for root stress, suffix stress, Foreign-stress, Anacrusis stress and pretonic stress [SLO 1974] and a set of exceptions to each rule make up the stress-by-rules schemes. Theoretical studies have centered around such schemes and are reported in the literature [CHO 1968; HOA 1971; HOA 1973; SET 1974; SLO 1974]. In a TSC system context, certain practical approaches with ad hoc rules are reported in the literature [BRO 1970; GAI 1968; RAB 1969].

The motivation to use a rule scheme for stress analysis comes from the study of English Orthography by Dolby et al [DOL 1963a; 1963b; 1964; RES 1966; VEN 1970]. These studies show that approximately 95% of the present day English vocabulary (in written form) can be handled by rules. The remaining 5% can be handled by a dictionary of exceptions. Also rules can predict the stress assignment for new words that may not be found in a dictionary. Thus, rule approach is much general and pragmatic.

Major results relevant to our purposes of stress assignment based on the above studies and of others [PIK 1945; PAL 1966; NIC 1916] is summarized by the following set of rules:

(a) Function words such as <u>a</u>, <u>an</u>, <u>the</u>, <u>at</u>, <u>on</u> are not stressed unless emphasised [PIK 1945].

(b) All non-function words are called content words by Pike [PIK 1945]. Content words are composed of the following : Roots which are free forms, such as <u>go</u>, <u>come</u>, <u>look</u>; bound forms that cannot come as free forms in English, such as the form <u>-ceive</u> in con<u>ceive</u>, re<u>ceive</u> etc; and either free or bound forms with affixes, such as the suffix -<u>ic</u> in ion<u>ic</u>, etc., the prefix in <u>pre</u>tend, <u>pre</u>fer and so on.

(i) All free roots are stressed, excepting the function words. (It is possible that even function words may be stressed in mathematical formulae, as shown in chapter 3).

(ii) If a suffix is present in a word, the suffix will determine the place of stress in a word. In the absence of a suffix, the prefix will determine the place of stress in a word.

Following Gaitenby [GAI 1968] we can formulate ad hoc rules for stress in English as follows:

1. Stress the first syllable of two and three syllable words, if no affix is present in the word or the word is not a function word without emphasis.

2. If a suffix is present, stress the syllable preceding the suffix.

3. If a prefix is found, stress the next syllable (that is the 1st syllable after the prefix).

4. For exception words, such as child, hash the word with exception dictionary and copy the phonetic form including the stress.

5. In all other cases proceed to stress depending on the number of syllables in the input word.

For the purposes of deciding the number of syllables in an input word, we assume the following criteria based on Dolby [DOL 1963a; 1963b].

(a) Count the number of orthographic vowels, such as a, e, i, o, u, y in a given input word from left to right.
(b) A word final e, such as in determine, prepare, should not be counted as a syllable provided that there is at least a syllable in the word other than word final e (This rule will ensure that the final e is counted in words like he, she, be, the, etc.).
(c) When two or more vowels come together, such as in meet, meal, fail, etc., consider the vowel sequence as a single vowel, hence count all as one syllable.

Stress rules are incorporated in our algorithm in the next chapter.

4.2.2.2 Phrase Stress:

If we view an English sentence as composed of phrases, such as noun phrase, verbal phrase, etc., then every phrase has a word in it called the head-word which gets emphasised(traditionally known as a subject, verb, and object). A pause (discussed in the next section) is introduced after the head-word and the presence of punctuation marks, such as comma, period, question mark, etc., signify the potential pause at phrase boundaries in English. Acoustically, the fundamental frequency at the phrase boundaries show a decrease in the frequency and intensity and results in a silence gap of significant nature. For example, if we read aloud a sentence in English, such as

"After running a long distance # he was suddenly aware # that he had gone # too far.".

where the symbol # is introduced to illustrate our point, though such a symbol never comes in a text.

Phrase level stres has been investigated and reported in the literature [ALL 1976; MAT 1966; BRO 1970; GAI 1972]. The following at hoc rules are proposed to handle phrase level stress in a TSC system.

1. Any content word before a punctuation is a potential head-word. Hence, it receives a phrase level stress.

2. Any content word before and after an auxiliary verb, such as is, was has, etc.,and all function words , other than the function words, can be a potential head word. Hence, it is given a phrase level stress.

3. Potential head-words receive double stress while all other content words receive a single stress.

For example, in a sentence, such as :
A Tall Strong Man was Looking at him., the words Tall, Strong, Man and Looking are all content words, but only Tall and Man are head-words, and receive double stress (Capital letters have been used to signify the content words in the above sentence).

The phrase level stress assignment as per our ad hoc rules are incorporated in our ANALYZER algorithm in the next chapter.

4.2.2.3 Emphatic Stress:

Any word, including a function word, may be stressed to signify emphasis. Italics, bold face or under-scoring, are all the techniques used to emphasis a word. We have provided a sub-algorithm //ITALICS// to produce emphasis in an input text in chapter 3. Emphatic stress will be considered as double stress and it will over-ride all other stresses.

4.3 Pause:

4.3.1 Definition:

A pause is a silence gap in a speech utterance conveying some information. Acoustically, there is zero-spectrum for a given duration of a pause. In our analysis the symbol # is used to signify the presence of a pause.

4.3.1.1 Word Pause:

Any two content words are separated by a pause.

4.3.1.2 Phrase Pause:

Two pause marks signify the boundaries of phrases.

4.3.1.3 Sentence Pause:

Three pauses mark the boundaries of sentences.

4.3.1.4 Paragraph Pause:

More than three pauses signify the end of a paragraph.

4.3.2 Pause Rules:

1. Introduce one after every content word.

2. Introduce two pauses when a punctuation mark, such as comma, is encountered, or an auxiliary verb is encountered, such as <u>has,</u> <u>have</u>,etc.
3. Introduce three pauses before a sentence, final punctuation marks, such as period, question mark, etc.
4. Introduce a pause if the algorithm for preprocessing is invoked when a special word is encountered as discussed in chapter 3.
5. Introduce a paragraph pause when a few characters (non-blanks) are followed by many blanks or when blank lines are introduced to designate a paragraph. (In our algorithm ANALYZER we have not provided for paragraph pauses, though this could be easily incorporated if desired).

4.4. <u>Letter-to-sound Rules:</u>

Spelling a word (i.e., character by character naming in a word) is not an aid in the pronounciation of a word. To transform written symbols in a word, we require letter-to-sound rules in a TSC system. Letter-to-sound rules have been investigated and reported in the literature [AIN 1973; THO 1958; CHO 1968; GAR 1964; HOL 1964; MAT 1968; HAG 1968; VEN 1970].

Both the BTL system [McI 1974] and the NRL system [ELO 1976] have adopted the KU system's rules [AIN 1973] for letter-to-sound conversions. We propose the following modification to these rules: stress, pause,etc,, will appear in the rules with the given necessary context to help in the transformation of letters to sound. NRL system [ELO 1976] reflects our conception. We leave out the details here.

CHAPTER 5

THE TESCON ALGORITHM

5.0 PURPOSE:

In this chapter, we propose a new algorithm, TESCON (TExt-to-Speech CONversion). In TESCON, we have integrated the preprocessing algorithm, STANDARDIZER, developed in Chapter 3, and the ANALYZER algorithm, which will be developed in this chapter and corresponds to the rules of linguistic analysis given in Chapter 4, and a TUNER algorithm to tune the system. Thus, TESCON will accept an input text in English orthography. It will preprocess and analyze it and produce a phonetic output suitable for speech synthesis.

In the course of the development of the TESCON algorithm, we will avoid repeating the STANDARDIZER algorithm. This algorithm is listed in table 5.2 at the end of this chapter. For the sake of conveneience, all algorithms developed in this thesis will also be listed in various tables at the end of this chapter.

In this chapter, we employ the concept of a sub-system discussed in figure 3.1 to illustrate the functional aspects of our algorithm. First, we provide an overview of the subsystem-composition of the TESCON algorithm.

5.1 SUBSYSTEMS OF TESCON:

The TESCON algorithm is composed of the following four subsystems:

1. the TUNER,
2. the STANDARDIZER,
3. the ANALYZER, and
4. the OUTPUTTER.

Of these, the OUTPUTTER can be considered as an integrated subsystem of the acoustic and engineering subsystems (block C and D in figure 2.1) mentioned earlier. This is beyond the scope of this thesis and is reported in the literature [HOL 1963; THO 1971]. To provide a clear understanding to the reader, we describe each of these three subsystems, namely, TUNER, STANDARDIZER, and ANALYZER in two sub-sections. Subsections 5.2.1, 5.2.2 and 5.2.3 explain the purpose of the subsystem, and the subsections 5.3.1, 5.3.2, and 5.3.3 provide the necessary algorithms.

5.2.1 THE TUNER:

5.2.1.1 PURPOSE:

(i) to identify the general subject area of the input text, such as Scientific and non-Scientific input,

(ii) to identify the sub-area of the input text, such as Linear Algebra, Software, Magnetic resonance, etc.,

(iii) to identfy the language of the input text, such as natural language, formal language, programming language, etc.,

(iv) to identify the sub-classification of the identified language, such as American English, British English, etc.,

(v) to identify the data base from among the many data bases stored in an auxiliary storage device. This selected data base will provide the propernames, abbreviations, etc., for the sub-area selected in (ii) above ,

(vi) to identify and retrieve the preprocessing table given at the begining of an input text or compute the same for a given text,

(vii) to store in core memory the function words, suffixes etc.,

(viii) and to initiate the subsystems in the preprocessing algorithm.

5.2.1.2 ASSUMPTIONS:

The TUNER sub-system assumes the existence of the following Information:

(a) Each input text will have an area-code and sub-area code which are unique, such as specific code for various sub-areas in mathematics, like a four digit number for each area, e.g. 2121 for Topology in Mathematics, 2163 for Experimental Psychology and so on. This will be provided at the begining of the input text or will be computed from the area names specified in alphanumeric characters.

(b) Each input text will have a 4-tuple (Page, Line, Word, Flag) to help in selecting proper sub-algorithms during the pre-processing of press-style problems. This will be in a table.

(c) Each input text will have a language code and a sub-language code. Values are assumed to be for American English by default.

(d) Rate of speech preferred, speech dialect, etc., will be either provided at the begining of the input text or by default to be approximately 150 words/ minute for standard American English.

5.2.2 THE STANDARDIZER:

5.2.2.1 PURPOSE:

(i) to identify the numerals and convert them into an equivalent description (either in phonetic form or in English Orthography),

(ii) to identify the various press-style problems, such as Capital-letter and font-sizes,

(iii) to provide proper description for quotes, and foot-notes,

(iv) to convert mathematical symbols and formulae into equivalent description in either English orthography or in phonetic form,

(v) to set word flag ON, when a phonetic description is generated for numerals, or press-style problems or mathematical symbols, etc. This will prevent further processing under ANALYZER which handles the normal English orthographic input other than those handled under the STANDARDIZER.

5.2.2.2 ASSUMPTIONS:

The following information are assumed to exist with respect to the STANDARDIZER:

(a) The input text is in English or in one of the languages acceptable to the system.

(b) When an unidentified symbol is encountered, such an inverted question mark, and hypen, these will be ignored for the present or they do not exist as far as the system is concerned.

(c) The description for mathematical symbol will be provided either as a set of data or in a data base. Complex symbols that cannot be handled by the algorithm in its existing form, will be ignored.

(d) When a description exclusively in IPA code is required, the system will be provided with suitable rules. In all other cases, only ASCII character description will be provided by the system.

(e) There is no rule to resolve ambiguity in the expansion of abbreviations. It is assumed that the data base will contain proper expansions. Where two different descriptions exist, the description from sub-area data base will be preferred.

(f) The character by character pronounciation for numerals is assumed. For example, 15 will be pronounced here as <u>one</u> # <u>five</u>, rather than fifteen. We can provide alternative algorithm to generate other descriptions, such as fifteen.

5.2.3 THE ANALYZER:

5.2.3.1 PURPOSE:

(i) to compare a given input word with the exception dictionary and copy the corresponding phonetic form if an entry exists in the exception dictionary,

(ii) to check whether a given word is flagged and if so, not to to process it ,

(iii) to count the number of characters in an input word,

(iv) to count the number of syllables in a word,

(v) to determine whether a given input word is a function word or a content word (function words are given in a dictionary),

(vi) to compute whether an input word contains a suffix, or prefix or both and if so, where to stress on the word (i.e., which syllable and character counted from the left),

(vii) to decide whether a given word is a head-word or not,

(viii) to decide whether the duration of a final syllable in a word is to be increased and by how much,

(ix) where to introduce pauses, and

(x) to convert a given alphabet in an input word into its corresponding phonetic form based on the context, duration, stress and other computed information applying letter-to-sound rules for each of the characters in the input word.

5.2.3.2 ASSUMPTIONS:

(a) Letter-to-sound rules for proper English dialect exist either in the data base or as an external data.

(b) The input from this algorithm is in ASCII code and an equivalent phonetic code may be generated by another algorithm, provided such an algorithm exist in the system.

(c) The Output from this algorithm can be converted into spectrum specification by an algorithm given by Thosar [THO 1971], which in turn produces speech output.

(d) All non-function words, non-punctuation symbols can be considered as content words.

5.3 ALGORITHM TESCON:

Invoke TUNER.

Invoke STANDARDIZER.

Invoke NUMERAL. (not discussed here)

Invoke ANALYZER.

Invoke OUTPUTTER.

Terminate the algorithm.

5.3.1 Algorithm TUNER:

(Note that the variables, such as ACODE, SCODE, etc., are alphanumeric or numeric)

Step0 : Read code for major subject area ACODE.

Step1 : If the code is not blank, compute ACODE, jump to Step2.

ACODE ⟵ 001 (fiction)

Step2 : Read sub-area code SCODE.

If SCODE is not blank, compute SCODE, jump to Step3.

SCODE ⟵ 002 (modern fiction)

Step3 : Read language code LCODE.

If LCODE is not blank, compute LCODE, jump to Step4.

LCODE ⟵ 777 (ENGLISH)

Step4 : Read dialect code DCODE.

If DCODE is not blank, compute DCODE, jump to Step5.

DCODE ⟵ 888 (standard American)

Step5 : Read preprocessing table PREPROS.

If the first entry in PREPROS is blank, set PREPROS-FLAG OFF, jump to Step7

Step6 : Compute the address of dictionary (abbreviation, propernames etc)

COMCODE ⟵ ACODE + SCODE

Copy contents of locations starting at address computed under COMCODE into DICTIONARY.

Step7 : Copy affixes into AFFIX.

Step8 : Copy function words into FUNCTION-WORDS.

Step9 : Copy Global information in GLOBAL.

Step10 : Read letter-to-sound rules. If nor rule exist in data, copy rules from storage device into RULES.

Step11 : Read PREPAGE table.

Step12 : If prepage table is blank, set TOTAL-LINES 30

TOTAL-PAGES ⟵ 100

TOTAL-CHARACTER ⟵70 (in a line of text)

Step13 : Invoke STANDARDIZER.

Step14 : Terminate the algorithm.

5.3.2 <u>ALGORITHM STANDARDIZER:</u>

Step0 : Invoke PAGER (... see chapter 3)

Step1 : Invoke ANALYZER.

Step2 : Terminate the algorithm.

5.3.3 ALGORITHM ANALYZER:

[SAVE : WORD-COUNTED , WORD , HEAD]

WORD $\leftarrow$ 0

Step0 : WORD $\leftarrow$ WORD + 1

Step1 : If WORD is greater than WORD-COUNTED, terminate the algorithm.

Step2 : If WORD-FLAG is ON, jump to Step0.

Step3 : Invoke COMPARATOR.

Step4 : Invoke SYLABIFIER.

Step5 : Invoke AFFIXER.

Step6 : If WORD = 1, jump to Step0.

Step7 : Invoke HEADER.

Step8 : Invoke STRESSER.

Step9 : Invoke PAUSER.

Step10: Invoke RULES.

Step11: Jump to Step0.

//COMPARATOR//

Step0 : Compare input WORD with exception dictionary entries.

Step1 : If an equivalent entry is found, copy the phonetic form,
flag the word.

Step2 : Terminate the algorithm.

//SYLABIFIER //

[SAVE : SYLABLE-COUNT]
SYLABLE-COUNT ⟵ 0

CHARACTER ⟵ 0

Step0 : CHARACTER ⟵ CHARACTER + 1

Step1 : Scan a character from the word (input).

Step2 : If character is blank, terminate the algorithm.

Step3 : If character is not one of [A, E, I, O, U, Y], jump to Step0.

Step4 : If character is [E] and (character+1) is blank and SYLABLE-COUNT $\geq$ 1, jump to Step0.

Step5 : If (character+1) is one of [A,E,I,O,U,Y], jump to Step0.

Step6 : SYLABLE-COUNT ⟵ SYLABLE-COUNT + 1

Step7 : Jump to Step0.

(Note: Word final e is dropped in Step4 provided there is at least one other vowel in the word. Step5 ensures that consecutive vowels are not counted as separate vowels).

//AFFIXER//

[SAVE : SYLABLE-COUNT]

Step0 : If SYLABLE-COUNT = 1, jump to Step5.

Step1 : Check if there is any suffix in the word.

Step2 : If there is suffix, move backward and compare characters with [A,E,I,O,U,Y] , and if a vowel is found, introduce stress mark (') after the vowel, jump to Step5.

Step3 : Check if the input word has any prefix and if yes, findout the first vowel after the prefix, introduce the stress mark (') after the vowel, jump to Step5.

Step4 : Count the second syllabic vowel (vowel in the second syllable) and introduce the stress mark (') after it.

Step5 : Terminate the algorithm.

//HEADER//

[SAVE : WORD, HEAD]   HEAD ⟵ 0

Step0 : If input word WORD is a function word, jump to Step4.

Step1 : If input WORD is a punctuation [. , ; : ? !]

WORD ⟵ WORD - 1 , HEAD ⟵ 2 , Output pause ##, jump to Step4.

Step2 : HEAD ⟵ 1 , Output pause # .

Step4 : Terminate the algorithm.

//STRESSER//

[SAVE : WORD, HEAD]

Step0 : If HEAD = 2, scan the word backward until the stress mark is found, introduce another stress, Output +1.5 after the vowel. (Duration of doubly stressed vowel has been increased by 50%)

Step1 : Terminate the algorithm.

//PAUSER//

[SAVE : WORD, HEAD]

Step0 : If HEAD $\geq$ 1, intoduce +.25 after the final vowel in the word,jump to Step5.

Step1 : If WORD is #, output +.75 after #, jump to Step5.

Step2 : If WORD is ##, output +1.25 after ##, jump to Step5.

Step3 : If WORD is ###, ouput +2.0 after ###, jump to Step5

Step4 : If WORD is ### , output +5.0 after ####(...).

Step5 : Terminate the algorithm.

//RULES//

[SAVE : WORD, HEAD]

CHARACTER $\longleftarrow$ 0

Step0 : CHARACTER $\longleftarrow$ CHARACTER + 1

Step1 : If current CHARACTER is not one of [A,B,.......,Z],or not a blank, copy the CHARACTER on Output file, jump to Step0.

Step2 : If current CHARACTER is blank, terminate the algorithm.

Step3 : Transform the character with the help of proper rules and header value etc. Copy the transformed form on output file.

Step4 : Jump to Step0.

Name of Sub-algorithm	Function of a given Sub-algorithm
TUNER	Tune the system for subject area of input text, proper data base identification etc.

Table 5.1 The TUNER sub-algorithm of TESCON and its function.

Name of Sub-algorithm	Function of a given Sub-algorithm
NUMERAL	Activates INTEGER-PART when a numeral is integer, otherwise for a decimal activates DECIMAL-PART.
INTEGER-PART	Isolates the digits in an integer, and spells them.
DECIMAL-PART	Isolates digits and period and spells them.

Table 5.1(a) NUMERAL sub-algorithm of TESCON and its function.

Name of Sub-algorithm	Function of a given Sub-algorithm
PAGER	Reads-in one page of an input text and activates the sub-algorithm LINE-NUMBER.
LINE-NUMBER	Points to a non-blank line in an input page and activates the sub-algorithm WORD-FINDER.
WORD-FINDER	Isolates non-blank words from an input text and stores them in an array. It activates the sub-algorithm LOOP.
LOOP	Checks the input word for possible preprocessing for press-style, etc., and activates the sub-algorithm NORMALIZER if necessary; otherwise activates UPPER-CASE for normalizing capital letters that begin a word.

Table 5.2 Sub-algorithms of STANDARDIZER and their Functions.

Name of Sub-algorithm	Function of a given Sub-algorithm
UPPER-CASE	Checks whether a given input word is a propername and if so activates PROPERNAME. Otherwise, activates ABBREVIATION.
PROPERNAME	Copy the equivalent phonetic form for a propername when possible, otherwise converts a capital letter to a lower-case letter.
ABBREVIATION	Copy the expansion for a given abbreviation when possible, otherwise, activates SPELL.
SPELL	Isolates the characters of a word and provides character-pronounciations.
NORMALIZER	Activates QUOTE,FOOT-NOTE, ITALICS and MATHS depending upon the pre-process required.

Table 5.2 STANDARDIZER's Sub-algorithms and their functions continued.

Name of Sub-algorithm	Function of a given Sub-algorithm
QUOTE	Identifies the begining and ending of a quote and signals the presence of a quote in a text,in a given page.
FOOT-NOTE	Signals the presence, begining and ending of a foot-note in a text, when the foot-notes appears at the bottom of a page.
ITALICS	Identifies Italics and Bold-face letters in a text and signals that those words require double-stress and converts them to normal characters.
MATHS	Identifies mathematical symbols and formulae and provides a description for them.

Table 5.2 STANDARDIZER's SUB-algorithms and their functions (continued).

Name of Sub-algorithm	Function of a given Sub-algorithm
COMPARATOR	Compare a given input word with exception dictionary entries and copies the equivalent form if any.
SYLABIFIER	Determines the number of syllables in a given input word.
AFFIXER	Determines whether a given input word contains any prefix or suffix or both and determines which syllable should be stressed.
HEADER	Determines whether a given input word is a potential head-word of a possible phrase.
STRESSER	Finds out where in a given input word double stress marks are given and increases the duration accordingly.
PAUSER	Finds out the pause marks in an input text and increases the duration of silence gap accordingly.

Table 5.3 Sub-algorithms of ANALYZER and their functions.

Name of Sub-algorithm	Function of a given Sub-algorithm
RULES	Determines which rule is to be applied for a given character in an input word within a given context, such as preceding and following sound (or symbol) , duration modifier present after the symbol, etc. Produces a phonetic code for each symbol based on letter-to-sound rules.

Table 5.3(a) ANALYZER's Sub-algorithms and their functions continued.

CHAPTER 6

CONCLUSIONS

6.0 Our Contributions:

In chapter 2, we investigated some existing systems on TSC by rules. This survey has shown these systems are primitive and incapable of becoming a TRUE TSC system as they presently exist. Thus, the first objective of this thesis, that is, to provide a state-of-art study on TSC by rules has been achieved.

In chapter 3, we have investigated some of the press-style problems encountered in a text not presented in the literatures. Even though we have considered only a limited number of press-style problems, we feel this to be only a beginning. These problems show the need for close cooperation between the publishers and the computer industry for standardization of certain aspects of printing styles and the necessity to provide a preprocessing table. To our knowledge, the idea of providing preprocessing table at the beginning of a text is new. Even an elementary investigation of the problems of converting mathematical systems into a descriptive system is very complex, requiring variations in the press-styles to be eliminated in the future. We believe that the investigation of the problems connected with the conversion of mathematical system is a very useful area of research for the future.

In chapter 4, we have developed operational rules for the analysis of words in English and suggested a new algorithm to handle the problems of stress assignment, duration modification, and the introduction of pausses in a text. The idea of using function words to signal the boundaries of major phrases and the presence of potential head-words of a phrase is new. Others have used function words to distinguish them from other content words and have avoided stressing the function words. Our approach for syntactical analysis requires minimum computation time and storage when compared to all other systems we have investigated.

In chapter 5, we have provided an integrated TESCON algorithm to handle a TSC -by-rules system. Though we have leftout many details that may be required at the time of implementation, we have provided a clear overview of the system. The details, we feel, can be included but depend upon whether one is using parallel processing, micro-processors, programming languages, etc. We leave the details of such implementation problems for future analysis. Even in our algorithms, we have avoided structured programming concepts. The reason for this is that in the TSC systems that we have investigated, many programming languages having different structures have been successfully utilized. By far, we narrowed out attention to two possible programming languages, namely, SNOBOL utilized in [ELO 1976] and FORTRAN-SLIP used in the TIFR system [THO 1971]. Variables in out algorithms are global in the same sense as SNOBOL variables but can be changed depending upon the actual implementation. Hence, we have not gone into details here.

In this thesis we have used a pragmatic approach in the conception of a true TSC system. We feel that this approach closely follows our intuition in the course of reading a text in real time. Thus, our algorithms are open-ended.

In the TESCON algorithm we have offered practical suggestions regarding the subject area, subarea, introduction of a preprocessing table, language and tuning, etc. This will aid in the preparation of different types of data bases required in a TSC context.

For the first time, we have suggested the need for the creation of separate data bases for various sub-areas of knowledge. The TUNER subalgorithm can select the proper data bases and reduce the active core memory requirments in a practical system.

6.1 Implementation:

6.1.1 Storage Requirements:

If we provide a dynamic memory allocation for TESCON, then approximately 25 K words should be sufficient for tuning the system and the related data bases, 5 K words dynamic memory for synthesis related rules, and 10 K words for programming, book-keeping, etc. In all less than 120 K words will be required as we have stipulated in chapter 2.

However, if we use parallel processing and micro-processors with PROM (Programmable Read Only Memory) for the rules, the memory requirments can be reduced considerably. However, it is difficult to speculate on this at the present time withour further analysis.

6.1.2 Processing time:

A real time setup can be achieved by parallel processing procedures. Once the tuning of the system is over in our present system, the STANDARDIZER and the ANALYZER operate on the input serially. However, even when the STANDARDIZER is operative, many of the ANALYZER's functions can be handled by the STANDARDIZER, such as counting the total number of syllables in an input word, presence and absence of affixes (suffix or prefix) in a given word, etc. Except for waiting time t (as a function of the processing time of a processor), the parallel processing can reduce the computation time. The scheduler design will have to take care of this.

6.1.3 Data Bases:

We have suggested the use of various data bases in the system, such as the subarea data base (for proper names, abbreviations, etc.), affix data base, function word data base, etc. While these may normally reside on storage devices, there are many search techniques available for retrieving information from these data bases, such as sequential file search, binary search, etc. We have not considered the best strategies and have left this as an open problem for further investigation.

6.2 Future Problems:

In the course of the investigations of a TSC system, we have come upon a number of problems suitable for future research. Some of the more important problems follow.

While our algorithms should work on any printed text, it appears that press-style conventions in journals are slightly different from text books. In order to minimize unwanted computation, it should be possible to standardize the printing press-style both in books and in journals. For example, the convention of foot-notes, quotes, etc. pose many problems. This will be a very useful area of research in the future. Some studies already are being considered [COU 1975].

With regard to mathematical systems, we had touched only the surface. There are a number of problems, such as translation of formal proofs into English, breaking down of large nested mathematical expressions, chemical notations and formula and graphs, etc. This at present remains an open problem. One recent paper dealing with the translation of a formal proof into English provides useful algorithms [CHE 1976] that could be incorporated into our TESCON algorithm, thereby making TESCON general.

The next problem is how to generate descriptions for a given picture? If some standardization can be achieved in this regard, then it may be possible to formulate a number of description generators. How do we decide the need for a picture? When do we need pictures? What kind of pictures? What kind of information are the pictures suppose to convey? All of these are questions that will require lengthy examination.

Yet another major problem is the necessity to control the explosion of analysis and the introduction of information in a TSC system. The linguistic analysis discussed under various references in chapter 4, show that there is too much information that people try to provide in a system. How much of this information is required in a TSC system (i.e. necessary and sufficient information)? Would comprehensive listening tests based on a very large sample (i.e., statistically valid) of the order of a few thousand naive listeners of English be helpful in this regard? If it is a question of training to accept a reasonably good speech output from a TSC system, then it should not be difficult to control the added information in the system. But how do we go about doing this? Only future analysis can answer this question.

Finally, we have seen a TSC system as an interdeciplinary area of knowledge requiring both non-numeric data processing techniques and and numerical analysis (at the acoustic and engineering aspects). This is in addition to the fact that pattern recognition,linguistics, acoustics, and engineering aspects proliferate in a TSC system.

APPENDIX A

AINSWORTH'S LIST OF RULES FOR LETTER-TO-SOUND TRANSLATION [AIN 1973]

(PARTIAL LIST)

Letter	Phoneme	Letter	Phoneme	Letter	Phoneme
-(a)-	/ə/	(b)	/b/	y(ou)	/u/
-(are)	/ɑ/	(ch)	/tʃ/	(ou)s	/ʌ/
(a)E	/ɛi/	(ck)	/k/	(ough)t	/ɔ/
(ar)	/ɑ/	(c)y	/s/	b(ough)	/ɑu/
(a)sk	/ɑ/	(c)e	/s/	t(ough)	/ʌf/
(a)st	/ɑ/	(c)i	/s/	c(ough)	/of/
(a)th	/ɑ/	(c)	/k/	-r(ough)	/ʌf/
(a)ft	/ɑ/	(d)	/d/	r(ough)	/u/
(ai)	/ɛi/	VC(e)-	/ /	(ough)	/əu/
(ay)	/ɛi/	th(e)-	/ə/	(oul)d	/u/
(aw)	/ɔ/	-C(e)-	/i/	(ou)	/ɑu/
(au)	/ɔ/	-C(e)d-	/ɛ/	(oor)	/ɔ/
(al)l	/ɔ/	(o)ld	/əʊ/	(oo)k	/u/
(a)ble	/ɛi/	(oy)	/ɔi/	f(oo)d	/u/
(a)ngSUF	/ɛi/	(o)ing	/əʊ/	(oo)d	/u/
(a)	/æ/	(oi)	/ɔi/	f(oo)t	/u/

REFERENCES

[AIN 1973] Ainsworth, William A.,
"A System for Converting English Text into Speech",
IEEE Trans. on Audio and Electroacoustics, AU 21, 3,
(June 1973), 288-290.

[AIN 1974] Ainsworth, William A.,
"Performance of a Speech Synthesis System",
Int. J. Man-Machine Studies,

[ALL 1968] Allen, Jonathan.,
"Machine-to-man Communication by Speech,Part II :
Synthesis of Prosodic Features of Speech by Rule",
SpringJoint Computer Conference,(1968), 339-344.

[ALL 1971] Allen, Jonathan.,
"Speech Synthesis from Unrestricted Text",
IEEE International Convention Digest,(1971), 108-109.

[ALL 1973] Allen, Jonathan.,
"Reading Machines for the Blind : The Technical Problems
and the Methods Adopted for Their Solution",
IEEE Trans. on Audio and Electroacoustics,AU 21, 3,
(June 1973), 259-264.

[ALL 1976] Allen, Jonathan.,
"Synthesis of Speech from Unrestricted Text",
Proceedings of IEEE.,(April 1976), 433-442.

[ATA 1971] Atal, Bishnu S.,
"Speech Analysis and Synthesis by Linear Prediction of the
Speech Wave",
J.Acous. Soc. Am., 50, (1971), 637-655.

[BAR 1965] Barnett, Michael P.,
Computer Typesetting : Experiments and Prospects.,
Cambridge, Mass : The M.I.T. Press, (1965).

[BRO 1970] Browman,C.P., C.H. Coker., and Noriko Umeda.,
"Toward Rules for Natural Prosodic Features in American English",
J.Acous.Soc. Am., 47, 1, (1970), 95.

[BUR 1968] Buron, R.H.,
"Generation of a 1000 Word Vocabulary for a Pulse Excited Vocoder Operating as an Audio Response Unit",
IEEE Trans. on Audio and Electroacoustics, AU 16,1, (March 1968), 21-25.

[CAR 1974] Carlson, R., and B. Granstrom.,
"Phonetically Oriented Programming Language for Rule Description of Speech",
Proc. Speech. Comm. Seminar., Stockholm, (1974), 245-253.

[CHA 1971] Chapman, W.D.,
"Techniques for Computer Voice Response",
IEEE International Convention Digest, (1971), 98-99.

[CHE 1976] Chester, Daniel.,
"The Translation of Formal Proofs into English",
Artificial Intelligence,7, 3, (Fall 1976), 261-278.

[CHO 1968] Chomsky, Noam and Morris Halle.,
The Sound Patterns of English,
New York : Harper and Row, (1968).

[CLA 1966] Clapp, Lewis C., et al.,
"Magic Paper - An On-line System for the Manipulation of Symbolic Mathematics",
Report R-105-1, Airforce Cambridge Research Laboratories.,
Bedford, Mass : Hanscom Field, (April 1966), 1-65.

[COK 1970] Coker,C.H., and Noriko Umeda.,
"On Vowel Duration and Pitch Prominence",
J. Acous. Soc. Am., 47, 1, (1970), 94.

[COK 1970a] Coker,C.H., and Noriko Umeda.,
"Acoustical Properties of Word Boundaries in English",
J. Acous. Soc. Am., 47, 1, (1970), 94.

[COK 1970b] Coker,C.H., and Noriko Umeda.,
"Text-to-Speech Conversion",
IEEE International Convention Digest,(March 1970), 216-217.

[COK 1973] Coker,C.H., N. Umeda, and C.P. Browman., "Automatic Synthesis for Ordinary English Text", IEEE Trans. On Audio and Electroacoustics,AU 21, (June 1973), 293-297.

[COM 1975] Computer (November 1975), 86, "Calculator Gives Voice Readout of Both Input and Output., Master Specialities Company, ARC-950 Audio Response Calculator.

[COO 1969] Cooper, Franklin S., Jane H. Gaitenby, Ignatius G. Mattingly, and Noriko Umeda., "Reading Aids for the Blind : A Special Case of Machine-to-Man Communication", IEEE Trans. on Audio and Electroacoustics,AU 17, 4, (December 1969), 266-279.

[COU 1975] Coueignoux, Philippe., "A Parametric Representation of Roman Printed Fonts", Progress Report 116,(July 1975), 250-262, Cambridge : M.I.T. RLE.

[CRC 1959] C.R.C. Mathematical Tables., (ed) Hodman., Cleveland : Chemical Rubber Publishing Co., Ohio, (1959), 12th Edition.

[DOL 1963a] Dolby, J.L., and H.L. Resnikoff., "On the Structure of Graphemic Syllabification", Preprint, 1963 Meeting of the Assoc. for Machine Translation and Computational Linguistics, Denver, (1963).

[DOL 1963b] Dolby,J.L., and H.L. Resnikoff., "Prolegomena to a Study of Written English", Lockheed Document, Lockheed Missiles and Space Corp., California, (1963).

[DOL 1964] Dolby, J.L., and H.L. Resnikoff., "On the Structure of Written English Words", Language, 40, (1964), 167-196.

[DUD 1939] Dudley, H., R.R. Riesz, and S.A. Watkins., "A Synthetic Speaker", J. Franklin Institute, 227, (June 1939), 739-764.

[ELO 1976] Elovitz, Honey Sue., Rodney W. Johnson., Astrid McHugh, and John E. Shore., "Automatic Translation of English Text to Phonetics by Means of Letter-to-Sound Rules", NRL Report 7948, Washington D.C. : Naval Research Laboratory, (January 1976).

[FLA 1970] Flanagan, J.L., C.H.Coker., L.R. Rabiner, R.W.Schafer and Noriko Umeda., "Synthetic Voices for Computers", IEEE Spectrum, (October 1970), 22-46.

[FLA 1972] Flanagan, J.L., Speech Analysis, Synthesis and Perception., New York : Springer-Verlag, (1972), 2nd edition.

[FLA 1973] Flanagan, J.L., and L.R. Rabiner., (ed) Speech Synthesis., Stroudsburg, PA : Dowden, Hutchinson and Rodd, (1973).

[FLA 1976] Flanagan, J.L., "Computers that Talk and Listen : Man-Machine Communication by Voice", Proceedings IEEE , (April 1976), 405-415.

[GAI 1967] Gaitenby, Jane H., "Rules for Word Stress Analysis for Conversion of Print to Synthetic Speech", Preprint, 74th Meeting of the Acous. Soc. A., (November 1967), 131-155.

[GAI 1972] Gaitenby, Hane H., G.N.Sholes and K.M. Khun., "Word and Phrase Stress by Rule for a Reading Machine", IEEE Conf. Record, Joint IEEE and AFCRL Conf. Speech Communication and Processing, (1972), 27-29.

[GAR 1964] Garvin, P.L., and E.C. Trager., "The Conversion of Phonetic into Orthographic English : A Machine Translation Approach to the Problem", Phonetica, 11, (1964), 1-18.

[GRI 1973] Grieshman, R., N.Sager, C.Raze and B. Bookchin., "The Linguistic String Processor", National Computer Conference, Montvale, NJ : APIPS Press., (1973), 427-434.

[GRI 1971] Griswold, R.E., J.F. Poage , and I.P. Polonsky.,
The SNOBOL 4 Programming Language,
Englewood Cliffs : Prentice-Hall , (1971), 2nd edition.

[HAG 1968] Haggard, Mark P, and Ignatius G. Mattingly.,
"A Simple Program for Synthesising British English",
IEEE Trans.on Audio and Electroacoustics.,AU 16, 1,
(March 1968), 95-99.

[HOA 1971] Hoard, James E.,
"Review of Chomsky and Halle 1968",
Glossa, 5, (1971), 222-268.

[HOA 1973] Hoard, James E., and Clarence Sloat.,
"Variation in English Stress Placement",
Proceedings: Colloquium on New Ways of Analysing Variation in English,Georgetown, (1973).

[HUG 1974a] Huggins, A.W.F.,
"An Effect of Syntax on Syllable Timing",
QPR, No. 114, M.I.T., (July 1974), 179-184.

[HUG 1974b] Huggins, A.W.F.,
"More Temporally Segmented Speech: Is Duration Or Speech Content the Critical Variable in its Loss of Intelligibility",
QPR., No. 114, M.I.T. (July 1974), 185-193.

[HOL 1964] Holmes,J, I. Mattingly., and J. Sherme.,
"Speech Synthesis by Rule",
Language and Speech,7, (July-September 1964), 127-143.

[KAN 1974] Kang, G.S.,
"Application of Linear Prediction Encoding to a Narrowband Voice Digitizer",
NRL Report, 7774, (October 1974), Washington D.C. :
Naval Research Laboratory.

[KER 1975] Kernighan, Brian W., Lorinda L. Chery.,
"A System for Typesetting Mathematics",
COMM. ACM., 18, 3, (March 1975), 151-156.

[KLA 1975] Klatt, Dennis H., and William E. Cooper.,
"Perception of Segment Duration in Sentence Contexts",
RLR. PR., No. 116, M.I.T., (July 1975), 189-205.

[KLA 1976] Klatt, Dennis W.,
"Linguistic Uses of Segmental Duration in English : Acoustic and Perceptual Evidence",
J. Acous. Soc. Am., 59, 5, (May 1976), 1208-1221.

[KUC 1967] Kucera, H., and W. N. Francis.,
Computational Analysis of Present-Day American English,
Providence : Brown University Press., (1967).

[LEA 1968] Lea Wayne A.,
"Establishing the Value of Voice Communication with Computers",
IEEE. Trans. on Audio and Electroacoustics, AU 16, 2,
(June 1968), 184-197.

[LEE 1968] Lee, Francis F.,
"Machine to Man Communication by Speech Part I :
Generation of Segmental Phonemes from Text",
Spring Joint Computer Conference, (1968), 333-338.

[LEE 1969] Lee, Francis F.,
"Reading Machines : From Text-to-Speech",
IEEE Trans. on Audio and Electroacoustics, AU 17, 4,
(December 1969), 275-282.

[LIB 1968] Liberman, A.M., et al.,
"Perception of Speech Code",
Psych. Review, 74, 6, (November 1967), 431-461.

[LES 1972] Lesk, M.,
"User-activated BTL Directory Assistance",
Internal Report, Bell Telephone Laboratories,
Murry Hill, (1972).

[LIS 1974] Lisker, Leigh.,
"On 'Explaining' Vowel Duration Variation",
Glossa, 8, 2, (1974), 233-246.

[LYN 1975] Lynette, Hirschman, Ralph Grishman, and Naomi Sager.,
"Grammatically Based Automatic Word Class Formation",
Inform. Process. and Management, II Great Britain:
Pergamon Press, (1975), 39-57.

[MAN 1975] A Manual of Style.,
Chicago : The University of Chicago Press, (1975),
12th Edition, 6th Impression.

[MAR 1967] Martin, William A.,
"Symbolic Mathematical Laboratory",
MAC. TR-36, Unpublished Ph.D., Thesis, M.I.T (1967).

[McI 1974] McIlroy, M.D.,
"Synthetic English Speech by Rule",
Computer Science Technical Report # 14,
Murray Hill, New Jersey : Bell Telephone Laboratories,
(March 1974).

[McI 1976] McIlroy, M.D.,
Personal Communication.

[MAT 1966] Mattingly, I.G.,
"Synthesis by Rule of Prosodic Features",
Language and Speech,9, 1, 1, (1966), 1-13.

[MAT 1968] Mattingly, I.G.,
"Experimental Methods for Speech Synthesis by Rule",
IEEE Trans. on Audio and Electroacoustics, AU 16, 2,
(June 1968), 198-202.

[NAG 1968] Nagy, George.,
"State of Art in Pattern Recognition",
Proceedings IEEE, 56, 5, (May 1968), 836-862.

[NEW 1971] Newell A., J.Barnett, J. Forgie et al,
Speech Understanding Systems, Final Report of a Study
Group, 1971.
Reprinted by : Amsterdam , Netherlands : North-Holland/
American Elsevier, (1973).

[NIC 1916] Nicholson, George A.,
English Words with Native Roots and with Greek,
Latin or Romance Suffixes,
Chicago : The University of Chicago Press, (1916).

[NYE 1973] Nye, P.W., J.D. Hankins, T.Rand, I.G.Mattingly and F.S.Cooper.,
"A Plan for the Field Evaluation of an Automated Reading
System for the Blind",
IEEE Trans. on Audio and Electroacoustics, AU 21, 3, (Nune 1973),
265-268.

[OCH 1974] Ochsman, R., and B.A. Chapanis.,
"The Effect of 10 Communication Modes on the Behaviour
of Teams during Co-operative Problem Solving",
Int. J. Man-Machine Studies, 6, 5, (1974), 579-619.

[OMA 1973] O'malley, Michael H, Dean R. Kloker, Benay Dara-Abrams,
"Recovering Parantheses from Spoken Algebraic Expressions",
IEEE Trans. on Audio and Electroacoustics, AU 21, 3,
(June 1973), 217-220.

[PAL 1966] Palmer, F.R.,
A Linguistic Study of English Verb.,
London: Longmons, (1966), Second Impression.

[PIK 1945] Pike, K.L.,
The Intonation of American English.,
Ann Arbor, Mischigan: University of Michigan Press,
(1945).

[RAB 1969] Rabiner, L.R., et al.,
"Investigation of Stress Pattern for Speech Synthesis",
J. Acous. Soc. Am., 45, 1, (1969), 92-101.

[RAB 1971] Rabiner, L.R., R.W.Schafer, and J.L. Flanagan.,
"Computer Voice Response using Low Bit Rate Synthetic Speech",
IEEE. Int. Convention Digest., (1971), 96-97.

[RAM 1973] Ramasubramanian, N, and R. B.Thosar.,
"The Role of Articulation Based Rules in Speech Synthesis",
Phonetica, 27, (1973), 65-81.

[RED 1968] Reddy, Raj D., and Ann E.Robinson.,
"Phoneme-to Grapheme Translation of English",
IEEE Trans. on Audio and Electroacoustics, AU 16, 2,
(June 1968), 240-246.

[RED 1976] Reddy, Raj D.,
"Speech Recognition by Machine : A Review",
Proceedings IEEE., (April 1976), 501-531.

[RES 1965] Resnikoff, H.L., and J.L.Dolby.,
"The Nature of Affixing in Written English",
Mechanical Translation, 8, (1965), 84-89.

[RES 1966] Resnikoff, H.L. and J.L. Dolby.,
"The Nature of Affixing in Written English Part II",
Mechanical Translation, 9, (1966), 23-33.

[SAG 1972a] Sager, Naomi.,
"Syntactic Formatting of Science Information",
Proc. Fall Joint. Computer Conf, Montvale, N.J :
AFIPS Press, (1972), 791-800.

[SAG 1972b] Sager, Naomi.,
"A Two Stage BNF Specification of Natural Language",
J. Cybernetics., 2, 3, (1972), 39-50.

[SAG 1975a] Sager, Naomi.,
"Sublanguage Grammars in Science Information Processing",
J. Am. Soc. Infor. Sci., (Jan- Feb 1975), 10-16.

[SAG 1975b] Sager, Naomi.,and Ralph Grishman.,
"The Restricted Language for Computer Grammars of Natural
Language",
Comm. ACM., 18, 7, (July 1975), 390-400.

[SAM 1975] Sambur, Marvin R.,
"Selection of Acoustic Features for Speaker Identification",
IEEE Trans. on Acoustics, Speech and Signal Processing.,
ASSP 23, 2, (April 1975), 176-182.

[SAN 1972] Santos, P.J.,Jr.,
"FASBOL II, A SNOBOL Compiler for the PDP-10",
DECUS No. 10-179,Digital Equipment Compter User's
Society, (December 1972).

[SCH 1968] Schlesinger, I.M.,
Sentence Structure and the Reading Process,
The Hague : Mouton, Janua Linguarum Series Minor 69, (1968).

[SCH 1975] Schmidt, C.E., and A.E. Rosenberg.,
"Bell Alabs Flight Information System — An Application
of theVoice Response System on the NOVA 800",
Internal Report, Bell Telephone Laboratories,N.J. (1975).

[SET 1974] Settera, George E.,
"English Stress",
Glossa, 8, 1, (1974), 83-107.

[SLO 1974] Sloat, Clarence.,
"The Formulation of English Stress Rules : Theoritical
Considerations",
Annual Meeting. Western Conference on Linguistics,
Victoria,B.C., (October 20, 1973).

[TER 1968] Teranishi, Ryunen., and Noriko Umeda.,
"Use of Pronouncing Dictionary in Speech Synthesis Experiments",
Report. 6th International Cong. Acoustics, Tokyo, (1968), B155-158.

[THO 1958] Thomas, Charles Kenneth.,
An Introduction to the Phonetics of American English.,
New York : The Roland Press, (1958), Second Edition.

[THO 1971] Thosar, R.B.,
"A Unified Approach to Computer Speech Synthesis and Recognition",
Unpublished Ph.D. Thesis., Indian Institute of Technology, Bombay, and TIFR, Bombay (1971).

[UME 1970] Umeda, N. and C.H. Coker.,
"Some Prosodic Details of American English",
J.Acous. Soc. Am., 47, 1, 1, (1970), 123.

[UME 1975] Umeda, Noriko and Ryunen Teranishi.,
"The Parsing Program for Automatic Text-to-Speech Synthesis Developed at the Electrotechnical Laboratory in 1968",
IEEE Trans. on Acoustics, Speech and Signal Processing, ASSP 23, 2, (April 1975), 183-197.

[UME 1976] Umeda, Noriko.,
"Linguistic Rules for Text-to-Speech Synthesis",
Proceedings IEEE, (April 1976), 443-451.

[VEN 1970] Venezky, Richard L.,
The Structure of English Orthography,
The Hague : Mouton, Janua Linguarum Series Minor 82, (1970).

[WEB 1966] Webster's Elementary Dictionary,
Springfield : G & C Merriam, (1966).

[WEI 1963] Weizenbaum, Joseph.,
"Symmetric LIst Processor",
Comm. ACM., 6, (1963), 524-544.

[WHI 1976] White, George M.,
"Speech Recognition : A Tutorial Overview",
Computer, (May 1976), 40-53.