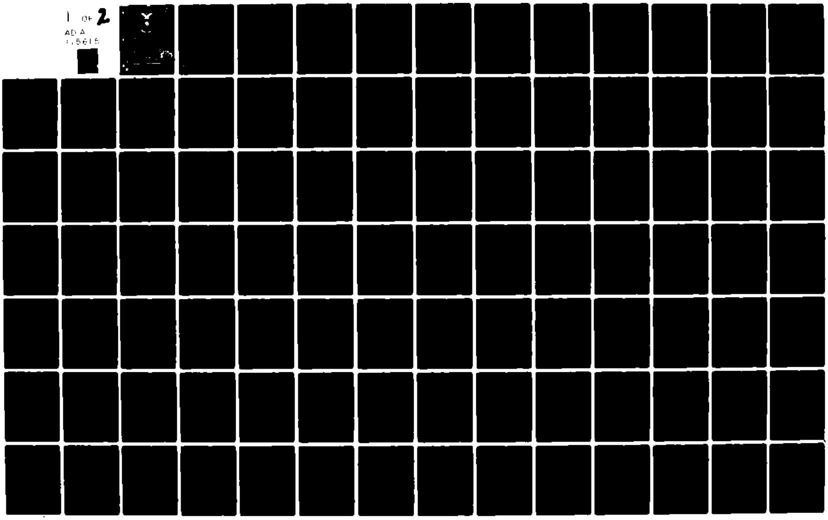
AD-A115 615   AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 17/2
THE RECOGNITION OF WORDS FROM PHONEMES IN CONTINUOUS SPEECH.(U)
DEC 81 C A BAKER
UNCLASSIFIED AFIT/GE/EE/81D-9   NL

1 OF 2
AD A
115615



UNITED STATES AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY
Wright-Patterson Air Force Base, Ohio

AFIT/GE/EE/81D-9

①

THE RECOGNITION OF WORDS
FROM PHONEMES
IN CONTINUOUS SPEECH

THESIS

AFIT/GE/EE/81D-9 Claude A. Baker
Captain USAF

AFIT/GE/EE/81D-9

THE RECOGNITION OF WORDS
FROM PHONEMES
IN CONTINUOUS SPEECH

THESIS

Presented to the Faculty of the School of Engineering

of the Air Force Institute of Technology

Air University

in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

by

Claude A. Baker, B.S.E.E.

Captain USAF

Graduate Electrical Engineering

December 1981

Approved for public release; distribution unlimited

Preface

The work performed in this thesis is a first attempt at word recogniton based upon the phonemic output of the Air Force Institute of Technology (AFIT) acoustic analyzer. The acoustic analyzer is the result of Dr. Matthew Kabrisy's long term guidance, hopes, and energy and, oh yes, let us not forget the sleepless nights of many of his students.

The final report for several speech recognition projects which I encountered during my research contained phrases which, when paraphrased, become, "Given more time and money, the recognizer would have worked perfectly." Let me not be different. For many reasons, but primarily those of time and a lack of data to process, the vocabulary is limited to the 10 digits. The system has had insufficient training and experience to determine whether or not it will really work. Preliminary results are encouraging. The system has parsed all words thus far presented, but there have been, at most, only four utterances of each word incorporated into the state table.

My thanks to Dr. Kabrisky, Capain Larry Kizer, and Major Walt Seward. It has been an interesting though frustrating experience.

My thanks and love to my wife, Polly, who was always supportive and always acquiesed to my plaintive plea of, "I just need one more thing for my computer to finish the thesis." Also, my love to Yvette and Dane who patiently endured Daddy's studies and waited for tomorrows that rarely came.

Claude Baker

Contents

List of Figures

List of Tables

Abstract

Thesis examines continuous speech recognition for a vocabulary of the digits 0 through 9. Recogniton is accomplished by parsing phoneme strings obtained from the AFIT (Air Force Institute of Technology) acoustic analyzer. Word recognition is accomplished by a finite state automata implemented in UCSD Pascal. Novel features of the program include the ability to load and exercise state tables representing different vocabularies, as well as the ability to interactively edit and modify the state tables without recompiling. Complete details of the program are included.

The quality of recognition is indeterminate at this time. Sufficient training of the system has not been accomplished to determine validity of the recognition strategy. The system will parse all utterances for which it has been trained, although no more than four utterances of each word have been incorporated in the state tables.

THE RECOGNITION OF WORDS FROM PHONEMES IN CONTINUOUS SPEECH

I. Introduction

This thesis treats the topic of continuous speech recognition by machine. The ability to converse with machines through natural speech is a long sought goal of Man. We dream of intelligent, omniscient companions; god-like in their power but existing merely to serve. The literature of science-fiction is replete with examples of these machines. HAL, of the movie, "2001, A Space Odyssey", is one of the better known sentient computers that listens and talks. Though we desire these servants, we fear them. A recurring theme of the stories is, "Who will be master, man or machine?" Few of the super-computers of imagination have been benign. Many turned on their masters and destroyed them, as did HAL. We love them and fear them, yet as scientists and engineers we continue to pursue the "intelligent" machines for the great benefits to be obtained.

Current news magazines and the trade magazines of the electronic industry seem to have frequent predictions or announcements of soon to be released equipment which will replace the office secretary. If we are to believe these announcements, in the not distant future we will need no secretaries, nor grammar teachers, nor a wide range of other people who deal with words.

Background

The Information Processing Technology Office of the Advanced Research Projects Agency of the Department of Defense (ARPA) in November, 1971 initiated a speech understanding research (SUR) project to achieve computer understanding of human speech. The ARPA SUR project had a large budget, involved many separate groups, and spanned five years. Its goal was a major breakthrough in speech understanding capabilities (Ref 1:249-251). Now, ten years later, we have yet to attain meaningful, natural speech recognition by computer. The phrase natural speech recognition is used in the sense of continuous or conversational speech recognition.

The fact that the SUR project was sponsored by the Department of Defense is significant. The military has much to gain from achieving speech recogniton capability. A wide range of tasks will benefit from the automation of speech recognition. Table I lists a set of tasks in which speech recogniton would be useful to the military. More than any other task this thesis is directed toward system control through speech recognition.

Table I

Speech Recognition Tasks with Military Application

1) Security
   - 1.1 Speaker verification
   - 1.2 Speaker identification
   - 1.3 Determining emotional state of speaker
   - 1.4 Recognition of spoken codes
   - 1.5 Secure access voice identification
   - 1.6 Surveillance of communication channels

2) Command and Control
   - 2.1 System control (ships, aircraft, fire control, etc.)
   - 2.2 Voice-operated computer interfaces
   - 2.3 Data handling and record control
   - 2.4 Material handling (mail, baggage, publications, industrial applications)
   - 2.5 Remote control (dangerous material)
   - 2.6 Administrative record control

3) Data Transmission and Communication
   - 3.1 Speech synthesis
   - 3.2 Vocoder systems
   - 3.3 Transmission data rate reduction
   - 3.4 Ciphering, coding, scrambling

4) Processing Distorted Speech
   - 4.1 Diver (helium) speech
   - 4.2 Astronaut communication
   - 4.3 Underwater telephone
   - 4.4 High "G" force speech

(Ref 1:310)

Problem

The research objective of this thesis was to develop an algorithm to recognize words from phoneme strings in continuous speech. The phoneme strings were to be provided by an acoustic processor which has been under development at the Air Force Institute of Technology (AFIT) for several years (Ref 2).

Felkey (Ref 2) indicated that the acoustic processor was 70% to 90% accurate in locating and identifying phonemes and presented results to support the figures. Based upon these figures, an existing algorithm was selected to carry out the word recognition task. The algorithm was to be tailored to the specific problems of the acoustic processor.

Scope

The target vocabulary was to be small, less than 100 words. The vocabulary was to support a minimal cockpit environment, consisting of the digits and a few command words. Table II is a listing of the selected vocabulary.

The phoneme strings to be parsed to words were to be constructed from a set of 39 universal phoneme prototypes. These phoneme prototypes were to be under revision by another researcher. Table III is a list of the initial phonemes used.

The speakers used for testing the system were to be principally this investigator (Baker) and other American males deemed useful for inclusion in the testing. Speech to be used was to be confined to digit strings such as ZERO ONE

Table II

Vocabulary Set

1.	ZERO	8.	SEVEN
2.	ONE	9.	EIGHT
3.	TWO	10.	NINE
4.	THREE	11.	CCIP
5.	FOUR	12.	SET
6.	FIVE	13.	ENTER
7.	SIX	14.	FREQUENCY
		15.	THREAT

THREE FIVE NINE with a maximum time constraint of three seconds of speech. The time constraint was imposed by an inability of the available hardware to sample and store more than three seconds of speech.

Recognition was to be achieved by a relatively simple pattern matching process since it was believed that little linguistic/syntactic information would be available in the limited vocabulary.

Table III

Phoneme set

Computer Symbol		Key Word	Computer Symbol		Key word
1.	PX	pay	21.	JX	you
2.	BX	be	22.	LX	let
3.	TX	to	23.	RX	read
4.	DX	day	24.	EE	eve
5.	KX	key	25.	IX	it
6.	GX	go	26.	EI	hate
7.	HX	he	27.	EH	met
8.	H@	ahead	28.	AE	at
9.	FX	for	29.	AX	ask
10.	VX	vote	30.	AA	father
11.	TH	thin	31.	AH	not
12.	DH	then	32.	AW	all
13.	SX	see	33.	OU	obey
14.	ZX	zoo	34.	UX	foot
15.	SH	she	35.	UU	boot
16.	ZH	azure	36.	UH	up
17.	MX	me	37.	UH	about
18.	NX	no	38.	ER	word
19.	NG	sing	39.	XX	NA
20.	WX	we			

Approach

The research divides naturally into the categories of: software development, system training, and system testing. The software to be developed included a system to execute the pattern matching algorithm and an editor to alter the word templates.

Ultimately, two separate systems were constructed with one being discarded. The final system was a state machine capable of parsing arbitrary patterns within the memory limits of the host processor. The actual software for executing a state machine is quite simple, however, the primary work of implementation lies in determining the state change rules. The main result of the system training becomes embodied in the memory contents representing the system states. The minimal software necessary to execute the state machine resulted in a program which combined the execution algorithm into the same program module as the editor. This resulted in an interactive system which was easy to debug and train. The combined editor and execution module permitted observation and editing of the executing state machine.

II. The Acoustic Processor

Hardware

The acoustic processor, which is the major component of this speech recognition system, is a powerful, generalized assembly of hardware and software. Figure 1 outlines in block diagram form the major components of the acoustic processor.

The computer which performs the signal acquisition and processing is relatively straight forward in its construction. Its design is a product of cooperative work between the AFIT faculty, principally Captain Larry Kizer, and various AFIT students, most notably Captains George Beasley and Dan Fredahl. Beasley and Fredahl (Ref 6) are responsible for the I/O processor embodied in the Cromemco computer.

The signal processor is composed of three interconnected computers with limited communications between the processors. They do not constitute a network in the sense that all the processors communicate directly with one another. The Nova 2 is the central processor in that it communicates with the Eclipse and the Cromemco, but the Eclipse and Cromemco do not communicate with each other.

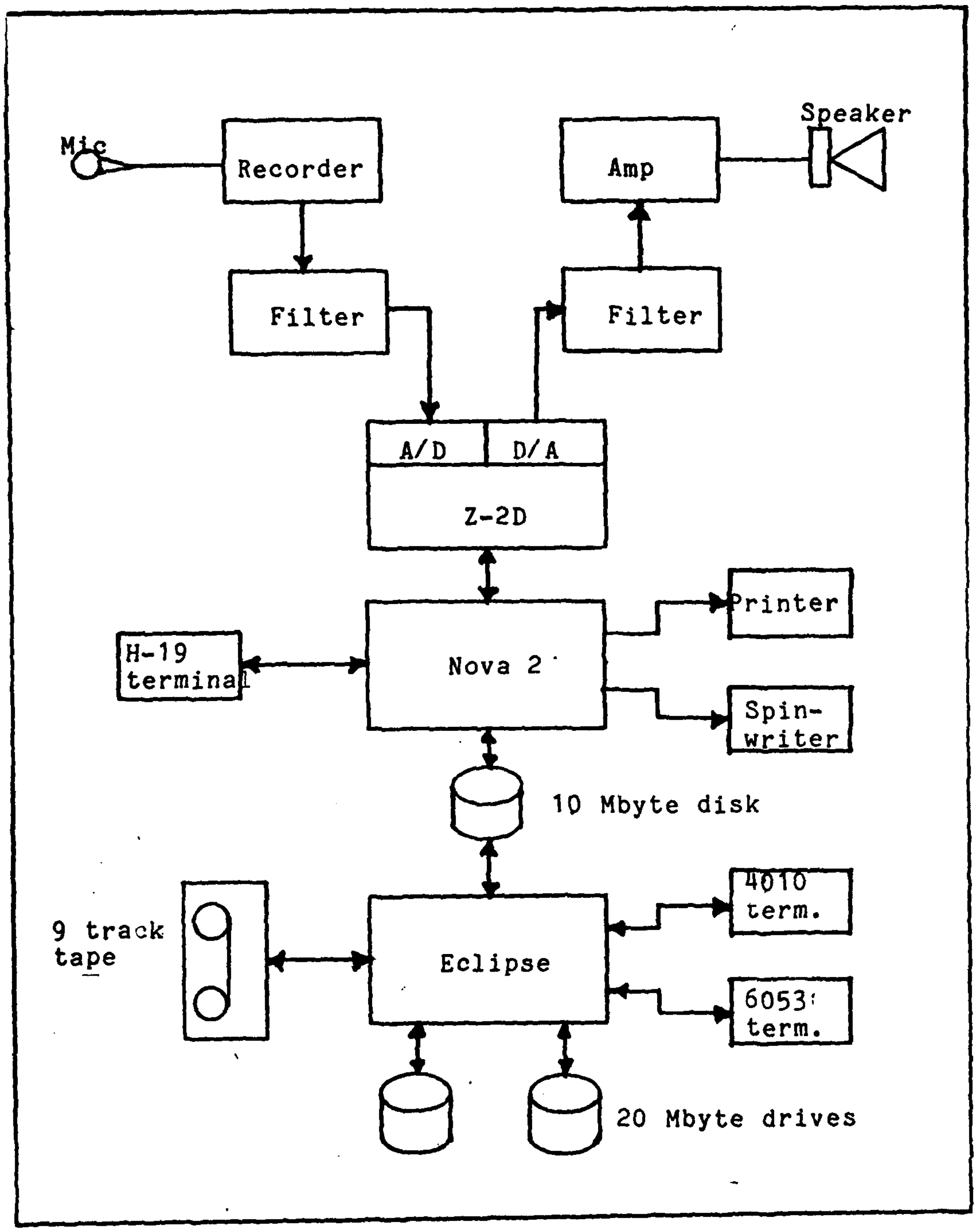


Figure 1. Acoustic processor block diagram

The signal processor can process both audio and video information. The Cromemco Z-2D is a Z-80 based microcomputer containing 12 bit digital converters, both A-D (analog to digital) and D-A (digital to analog). It also contains 4 bit video A-D and D-A converters for video input and output. The video converters contain their own high speed memory but the memory for the audio digital converters is the 64K byte memory of the Cromemco computer. The information is moved in and out of the Cromemco memory through a DMA (direct memory addressing) process. The 64k byte Cromemco memory is the constraint which limits the processing of speech to utterances less than three seconds duration. We customarily process speech at a digitization rate of 8KHz. Since 2 bytes are required for each 12 bit word passed to or from the audio digital converters, 16 kilobytes of memory are used each second. Leaving room in memory for the operating system results in a practical time limit of approximately three seconds of speech at 8KHz. The Cromemco is interfaced to the Nova 2 through a high data rate buss for speech storage. Speech segments may be digitized, stored, and played back through the digital to analog converters very conveniently.

The Data General Eclipse computer is a relatively powerful general purpose mini-computer. It is interfaced to the Data General Nova 2 and three hard disk drives. One drive shared with the Nova 2 is of 10 megabyte capacity and the other drives are 20 megabyte capacity. The 10 megabyte disk drive is of the common figuration which provides a 5 Mbyte 'fixed' disk and a 5 Mbyte removeable disk. The fixed disk is fixed only in the sense that it is not to be removed from the drive by the computer user. The other disk may be removed to permit the transport or protection of data on a disk. The Eclipse also has a 9 track tape drive used primarily for archival storage of data and programs. The two Data General computers, sharing a common disk drive, communicate with each other through an inter-processor communication buss. The inter-processor buss functions primarily to prevent the two computers from accessing the common disk drive simultaneously. The normal mode of passing data between the two computers is through files placed on the common disk drive. One computer writes a file which may then be accessed and processed by the other computer.

The Eclipse operates two CRT (cathode ray tube) terminals. One of them is a standard text display terminal but the other is a Tektronix 4010 vector graphics display terminal. The graphics terminal serves an important function in speech work in that a program has been developed which permits it to display speech spectrograms. The current function of the system permits an operator to designate a section of a speech spectrogram on the graphics terminal and then hear the exact segment of speech responsible for producing the spectrogram. The segments of speech played back may have a duration ranging from a few milliseconds up to approximately three seconds.

The Nova 2 provides system I/O for both the Data General computers. It controls both printers currently connected to the system. The primary printer is a Printronix high speed dot matrix printer. This allows the output of relatively high quality graphics for scene recognition work, as well as the printing of speech sprectrograms. The Printronix also performs the mundane task of listing data and programs. The second printer connected to the Nova 2 is a high quality daisy wheel printer.

Signal Handling

The speech signals may be acquired from a microphone and digitized immediately, or recorded on tape and then digitized. In the direct acquistion mode, the tape recorder electronics function as a preamplifier for the microphone. The amplified signal is passed through a Rockland active filter. This filter is capable of 8 pole high pass and low pass function over a wide frequency range but for speech processing it is set as an 8 pole low pass Butterworth filter with a 3db cutoff at 4KHz. An 8 pole filter results in an attenuation slope of 48 db per octave. Following the filter, the signal is immediately digitized and stored. For speech recognition, the signals receive a 6db per octave logarithmic pre-emphasis beginning at 500 Hz. This pre-emphasis is not a hardware function; it is accomplished by software.

The signal processing system is configured to allow very convenient acquisition of speech signals. Acquiring a speech sample requires that the operator execute a program which supervises the process. The supervisory program currently used is AUDIOHIST, written by Captain Paul Finkes. When executed, this program requests a file name for the speech sample and allows the operator to choose between input and output. Upon selecting input, AUDIOHIST allocates file space and sets up the Cromemco I/O processor to accept a speech input. When all is ready, a lighted button, convenient to the control console and microphone, begins flashing. Depressing the button immediately starts the analog to digital converter. The normal action is to press the button and begin speaking. Upon reaching the time limit of three seconds, the lighted button signals the end of input. AUDIOHIST is available to allow an immediate scan of the acquired samples. It provides a condensed listing of the maximum input voltages attained, with an approximate indication of when they occurred. This permits a convenient check to ascertain that the audio gain was within reasonable limits. The general goal in checking speech input voltages is to attain as high a level as possible without saturating the A-D (analog to digital) converters. AUDIOHIST also permits immediate playback of the acquired speech sample. This is done routinely to insure that parts of words are not missed.

Software

The software and associated algorithms naturally constitute the important functional parts of the speech recognition system. The current software design is largely the work of Lt. Mark Felkey (Ref 2) but has recently been modified extensively by Lt. Karl Seelandt.

Felkey's algorithm in a strong sense follows from the work of Potter, Kopp, and Green (Ref 3) in their work on speech spectrograms, published in 1947. They made the observation that persons familiar with speech spectrograms were able to read the spectrograms in much the same sense that we read text. They also noted that spectrograms from different speakers were similar for the same utterance.

This similarity between different speakers provides the justification for Felkey's approach of accomplishing phoneme recognition based on spectrographic data (Ref 2:12).

In processing an analog speech file for phoneme recognition, the first step is the production of a spectrogram of the entire utterance. Each vertical column of the spectrogram, Figure 2, represents 64 digitized samples or 8 milliseconds of analog speech. Performing a DFT (discrete Fourier transform) on the data results in a 32 point DFT. Only the magnitude of the spectral components is retained for further processing. The entire file is processed 64 samples at a time, using rectangular windowing. The resulting spectrogram contains frequency magnitude information from DC to 4Khz, with frequency samples at 125 Hz intervals. The choice of 64 samples for each DFT is very much empirical.

After the spectrogram is available, a correlation program determines which phoneme is the best possible choice for each column of the spectrogram. The phoneme templates used in the correlation process are prepared by listening to short segments of a speech file. A program developed by Seelandt permits veiwing a spectrogram on the 4010 graphics terminal and then designating a short sequence to be played back over the system speakers. The spectrogram is not

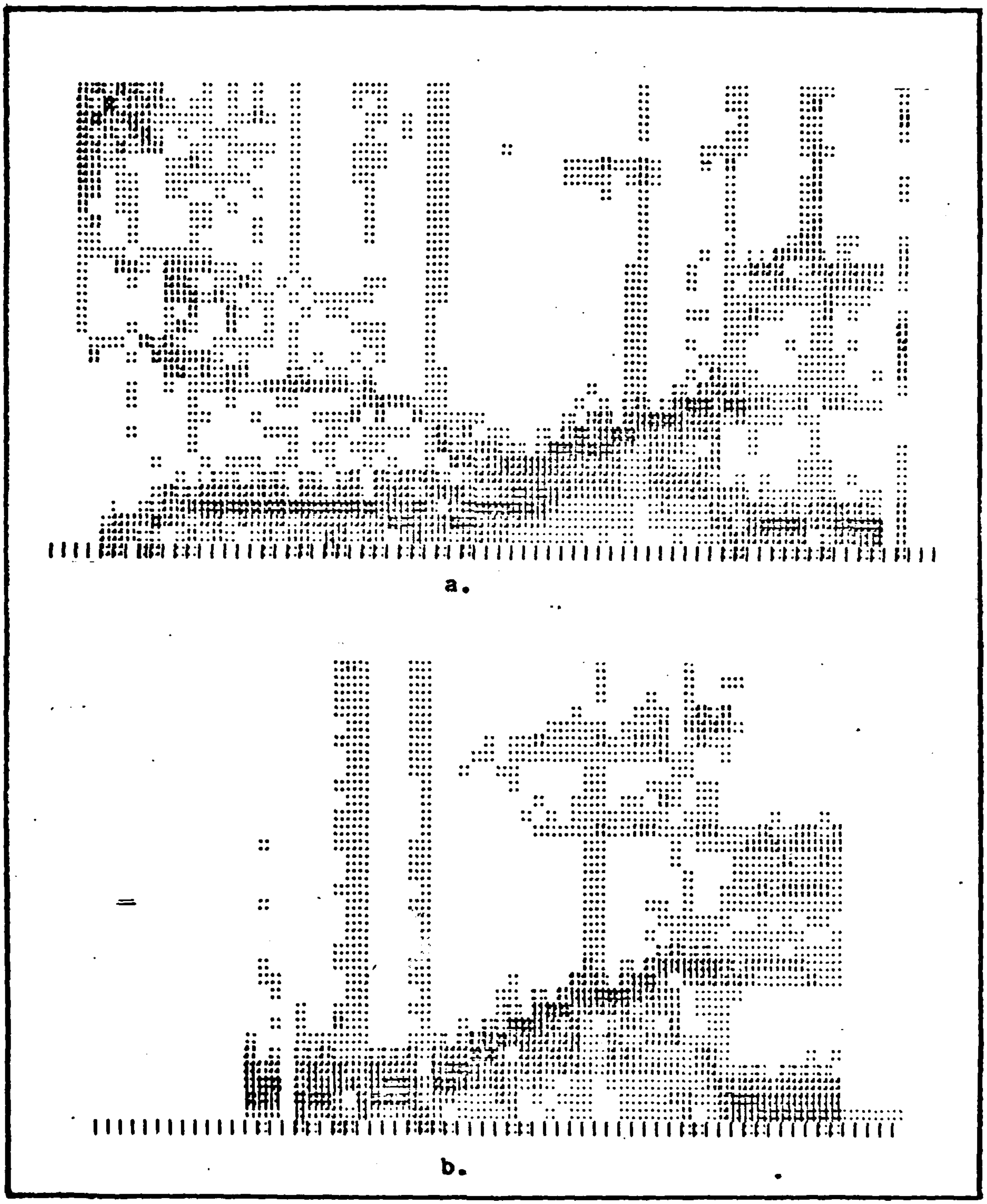


Figure 2. Spectrogram of the word "one" uttered in (a) connected speech and (b) from isolated speech.

played back but rather the segment of digitized speech which produced the spectrogram is played back. This permits the isolation of potentially very short, well defined phoneme segments to use as templates. The templates are placed in a file which the correlation routine compares against the spectrogram, one column at a time. Figure 3 is a spectrogram of several of the phoneme templates. Each phoneme template is 5 columns wide or 40 milliseconds time duration. The correlation process begins at the fifth column of the speech spectrogram. Each phoneme template, which is five columns wide, is compared to columns 1 through 5 of the speech spectrogram. The results are tabulated and ranked then the process is repeated by comparing each phoneme template to columns 2 thru 6 of the speech spectrogram. A tabulation is produced for each 8 milliseconds of speech until the entire speech file has been been through the correlation process.

Felkey produced spectrograms based upon several different sample sizes. He judged the 64 sample spectrogram to be the one which produced maximum intelligibility for a human, yet which reached a 'good' compromise between time and frequency resolution (Ref 2:15). (An unfortunate feature of the DFT is that as frequency resolution increases, time resolution decreases and vice versa.) The

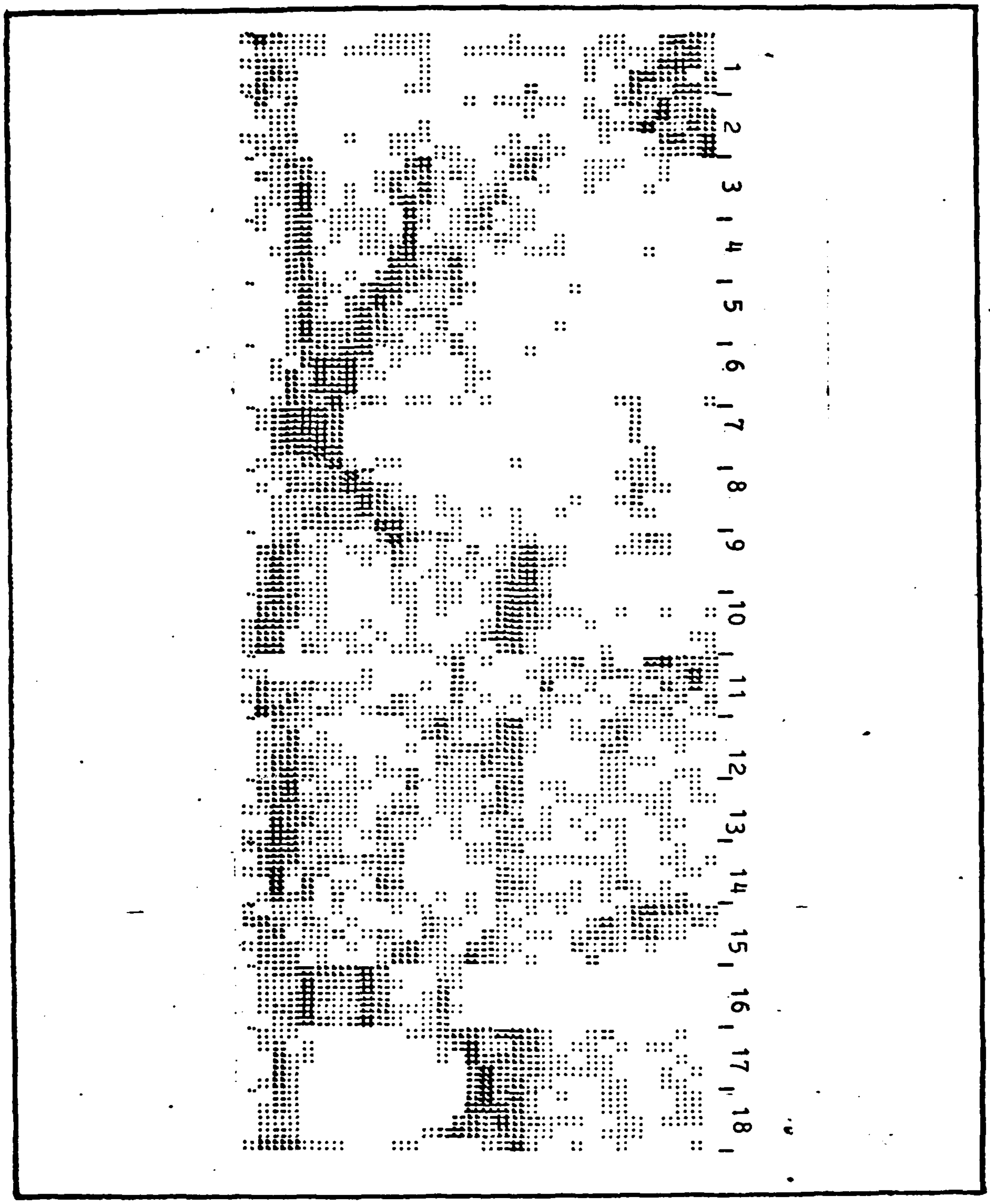


Figure 3. Phoneme Templates

'good' compromise is one which allows resolution of the shortest phonemes which Felkey observed to occur in the speech samples he examined. In current work with the system, we have seen phonemes which extended over only two columns of the spectrogram. This represents a time period of 16 milliseconds since each column of the spectrogram represents 8 milliseconds of speech. The short phonemes were observed on the very beginning of the 't' in the word 'two'.

Using short phoneme templates as this algorithm does appears to obviate the need to account for variations in speech rate. Since a phoneme selection is made every 8 milliseconds, the normal output of this program produces multiple matches over time. For instance, a phoneme lasting 100 milliseconds in the speech file will result in the phoneme recogniton algorithm producing 12 or 13 output matches. If someone speaks slowly and stretches phonemes, the Felkey algorithm simply indicates that multiple occurences of the phonemes are joined to produce the longer utterance.

III. The State Machine

Introduction

The recognition algorithm studied in this thesis is implemented as a finite state machine. The primary motivation for choosing such an implementation was a recognition that the more successful of the ARPA SUR projects were based upon state machines. The HARPY machine, constructed as part of the ARPA SUR project, was probably the most successful speech recognition machine yet built; it was a finite state machine (Ref 1). HARPY provided the initial germ of idea for this algorithm. A review of the properties of a finite state machine resulted in deciding that a finite state machine would serve my purposes well.

Reasons for Choice

First, a state machine is, or may be, a very powerful algorithmic device. Every digital computer is conceptually a finite state machine. Though even a modest digital computer is capable of a very large number of states, it is nevertheless, a finite state machine (Ref 5:63).

Second, the operation of a state machine is generally quite easy to visualize. This may be a personal bias, but I believed that the ability to visualize the operation of the recognition algorithm was quite important. I am much more comfortable with state diagrams than computer program flowcharts and was thus strongly biased to choose a state machine.

Third, a state machine is intrinsically capable of determining order. The first recognition algorithm attempted under this thesis was not designed to consider the order in which phonemes occurred. Work performed with that algorithm, however, showed that recognizing the order of phonemes was necessary even in the small vocabulary which we are using. This was an unanticipated result and stemmed from the nature of the errors made by the specific acoutic analyzer being developed in the AFIT signal processing laboratory.

Fourth, a state machine is directly capable of timing events. In the algorithm under discussion, a state machine would accomplish timing by counting events. The actual time duration of each event is measured by the analog to digital convertor in the acoustic processor. Phoneme duration was not considered important for this application, but the inherent timing capability which the algorithm proffered gave some hope of simple correction should the assumption prove wrong.

Fifth, a state machine simulation appeared to be one of the few software implementations capable of being "rolled into and out of" a computer memory and modified without recompiling. No research was performed to find other techniques but the capability is unique within my experience. What this really means is that a state machine simulation was constructed in which the state transition information was held in a data file. The data file was separate from the program which exercised the state machine. We were thus able to modify the state transition information quickly, easily and without recompilation of the program.

What is a State Machine?

Let's begin by considering a state machine as a concept rather than a physical device. Viewing it as a concept is particularly appropriate here, since the state machine implementation is in software.

A digital system may be modeled as a state machine. In the model, a digital system is regarded as a system which moves from one state to another in discrete steps. Each transition or step is determined by the inputs to the system as well as the current state of the system. At each transition, the system may output information. In a software implementation of a state machine, each state is represented by a series of ones and zeroes stored in memory (Ref 5:63).

To accomplish useful tasks, a state machine must have a defined start state. In terms of a computer program, the start state corresponds to the initialization process which any program must accomplish. Certain memory locations or registers must be set to zero, other storage locations must be set to other predefined values. If this initialization is not accomplished, the program will produce essentially random results based upon the content of the storage locations.

Implementation

The word recognition algorithms discussed were implemented on an Apple II computer in UCSD Pascal. The Apple had 64 kilobytes of random access memory, two disk drives allowing online storage of 286 kilobytes of programs and data, a printer and serial interface devices which permitted direct communication with the Nova computer in the acoustic processor.

The Apple, using Pascal, is a powerful, flexible machine which permits simple access to the inner workings of various programs and processes. Pascal implemented on the Apple cannot be relegated to the status of a mere subset of the standard Pascal language. It is missing only one significant capability and that is the ability to pass a procedure name as a formal parameter to another procedure or function. When compared to the Pascal implementation on the Cyber 6600, used by AFIT, the UCSD Pascal is actually more powerful in certain respects. The UCSD Pascal will accomdate set definitions with 512 members. The Cyber is limited to approximately 54 set members. These facts are normally of little significance but one algorithm considered and tried briefly required large sets. The language on the Apple was actually more suitable and attractive than the Cyber. Certainly the Cyber executes code much faster and has greater arithmetic precision, but neither of these capabilities was of any value to this project. The convenience and accessability of the Apple overshadowed the power of the Cyber or other computers available.

The state machine, as implemented, is little more than an array of data with a mechanism to accept inputs and determine the next position in the array. A position in the array corresponds to the current state of the machine. The state machine constructed was capable of 256 states, and a single input. THe machine was implemented with 256 states because this number of states may be addressed with a single byte, or 8 bits, of information. Exceeding 256 states would immediately double memory storage requirements. The input accepts 40 different values, corresponding to 40 different phonemes. Any state is capable of being an output state. An output state is one in which the algorithm determines that a word has been found. The normal output condition is to indicate "no output" at each state transition. However, when the machine has stepped through a sequence of states, indicating detection of a word, that word is typed out.

Data Structure

Since the state machine implementation and algorithm constitute a major goal of this thesis, it is appropriate to discuss the data structure and and recognition strategy.

Figure 4 is a diagramatical representation of the state machine data structure. In the terminology of Pascal, this is a record of arrays. Records are an important structure within Pascal, they are responsible for much of the power of the language. A record is a high level data structure which the programmer defines. The structure represented in Figure 4 is composed of three different arrays. The power of a record resides in the ability of the system to treat the record as a single entity. The arrays which compose the record are :

1. A two dimensional array of integers with values from 0 to 255 with subscript ranges from 0 to 255

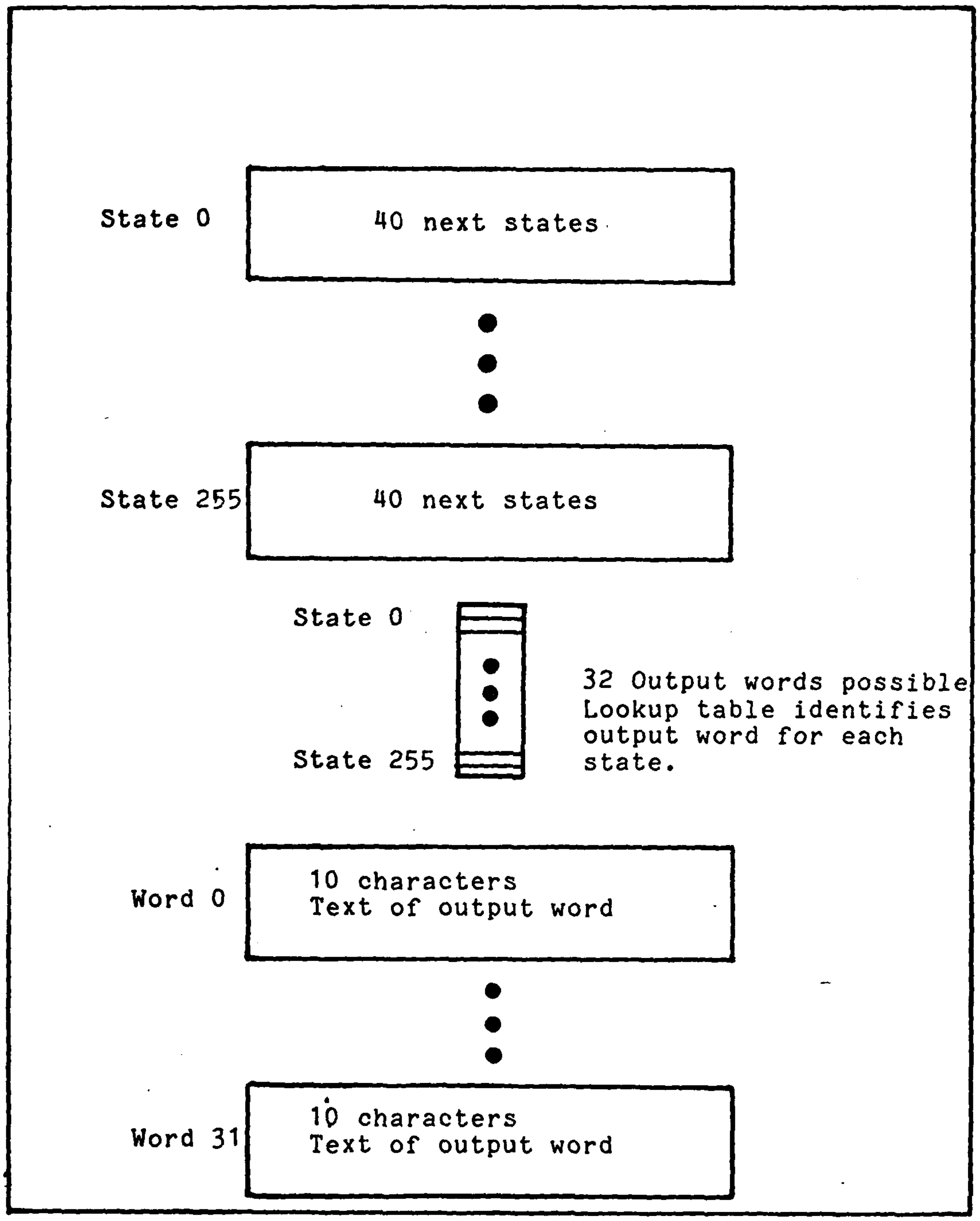


Figure 4. State machine data structure

and 0 to 39.

2. An array subscripted from 0 to 255 taking integer values from 0 to 31.

3. A two dimensional array with subscripts of 0 to 40 and 1 to 10 with each cell of the array being a character variable.

The actual arrangement of the arrays is insignificant. What is important is the ability to define such a structure, place data within the structure and then write the data into permanent storage as binary image of the structure as it existed in memory. The system treats the record as a entity and reads the record from storage or writes it to storage as a single entity. The record as described in Figure 4 (and as implemented) occupies approximately 10,000 bytes of storage. The programmer is naturally allowed access to the individual items of the record to read or modify them.

Recognition Strategy

Figure 5 is a state transition diagram to recognize the word zero. Let's discuss it in some detail. The discussion will illuminate the inner workings of the state machine and reveal the recognition strategy.

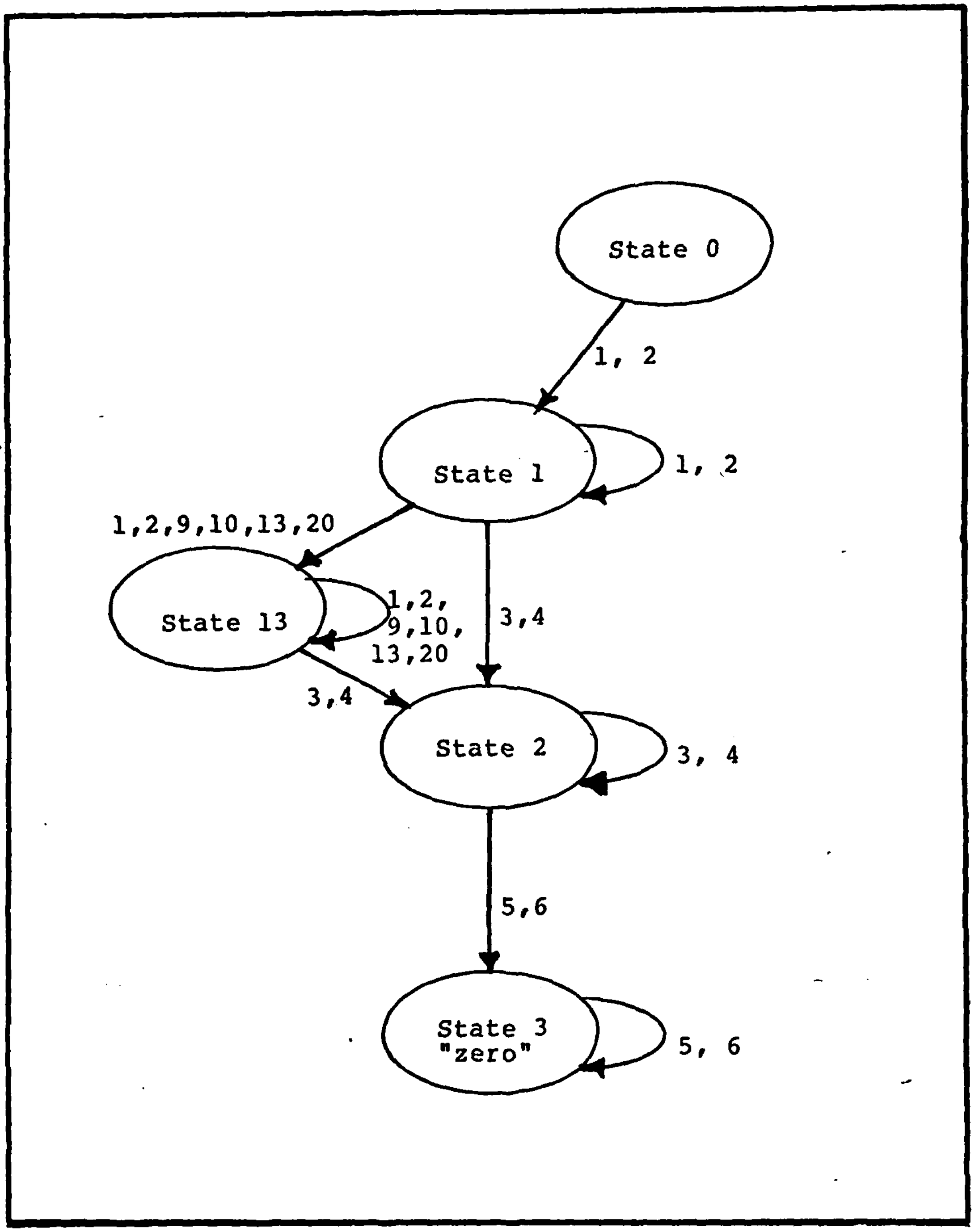


Figure 5. State transition diagram of word "zero"

Each circle represents a state and the number within the circle is the state number. As previously mentioned, state 0 is the starting point for all recognition attempts. There are many paths leading out of state 0 but we will only discuss two of them for the moment. The phonemes which will produce a transition from one state to another are listed between states. The phonemes are the numbers beside each arrow. State 1 has an arrow which loops back to state 1. Listed beside that arrow are phonemes 1 and 2. The meaning is that so long as the input phoneme is 1 or 2, the machine will remain in state 1. Once the machine has reached state 1, an occurrance of phonemes 9, 10, 13, or 20 will change the state to state 13. The phonemes listed are explicit state changes. Another state change which is implied, but listed nowhere, is that upon reaching state 1, if any phoneme occurs other than 1, 2, 3, 4, 9, 10, 13, or 20, the the machine transitions to state 0. This is both a strong point and a weakness of the chosen recognition strategy. It is a strength of the algorithm in the sense that it appears highly unlikely that this algorithm will confuse one word for another. If one single phoneme occurs which is not within the experience of the system, the decision is that the most recent phoneme string is unrecognizable. The weakness, of course, lies in the intolerance of the system to deviations from patterns on which it has been trained.

The term "within the experience of the system" requires explanation. It means that the system must have been trained to recognize the string under consideration. Each possible deviation from the expected path to a word must have been defined. In practical terms, this requires that great effort be devoted to training the recognition algorithm. All tolerated deviations from the expected state path must be explicitly determined and entered into the state transition matrix. The training strategy adopted requires that the trainer examine the state transition diagram for a word an integrate new word templates into the existing structure. The expectation is that as many patterns are processed and entered into the state transition table, few changes will be made as each new word is processed. I'm not speaking of "new" in the sense of words not previously in the vocabularly. It means new utterances of words already in the vocabularly.

Returning to the explanation of Figure 5, phonemes 1 and 2 are the "z" sounds of zero. Phonemes 3 and 4 are the "er" and phonemes 5 and 6 the "oh" sound to terminate the word zero. All other phonemes listed are observed errors and deviations from the expected phoneme string.

Obviously, this technique is highly dependent upon the function of the acoustic processor and the speaker. It does appear possible, though, to allow for speaker variations by simply allowing new paths to a word. Certain types of deviations are allowed but if the deviations are not within the experience of the system, the state machine returns to state 0 and begins attempting to parse a new word.

IV. Training the Recognizer

Training the word recognizer is at best tedious and time consuming. Furthermore, it does not appear that the process is amenable to automation. The reason for this is simple. Automating the training process requires a more capable recognizer to accomplish the training than is required to carry out the word recognition process.

The training process which I have followed is very time comsumptive but minimizes memory use in a general sense. Most important, the training process appears capable of producing generalized word recognition templates. An automatic training algorithm is a desireable goal but it is not clear that one can be constructed. It appears that an automatic trainer capable of generalizing the word templates would be more complex than the word recognition algorithms.

The training process begins with the state machine initialized so that all possible state transitions result in a jump to state 0. A jump to state 0 is the required action when the input phoneme results in a string which the state machine has not been trained to recognize. We shall begin with a description of training the machine to recognize the word "zero". However, before we begin the description, several details of the acoustic analyzer must be illuminated.

The phoneme set used for this work consists of phonemes taken from the words "zero", "one", "two", "three", "four", "five", and "six". There are 28 phonemes in this set. They are listed in Table IV. The keyword listed for each sound is not appropriately marked in a phonetic spelling sense. The underlined portion of each word is an approximation of where the phoneme was obtained. Note that the phonemes are duplicated in several instances. The phoneme "ou" appears in three different places as phoneme numbers 5, 6, and 20, for instance. In general, when a phoneme appears with a number as part of the name, as in UU1, the lower numbered phoneme represents a beginning segment of the phoneme and the higher numbered phoneme a later segment of the same sound.

Very briefly, the steps required to train the state machine to recognize a word are:

1. Obtain a digitized speech file.

2. Process the speech file to obtain a spectrogram.

3. Process the speech file to obtain phoneme recognition results from the acoustic analyzer.

Table IV

Phoneme Set

Computer Symbol		Key Word	Computer Symbol		Key Word
1.	Z1	zero	15.	TH	three
2.	Z2	zero	16.	R	three
3.	ER1	zero	17.	EE1	three
4.	ER2	zero	18.	EE2	three
5.	OU1	zero	19.	F	four
6.	OU2	zero	20.	OU	four
7.	WO	one	21.	UH	five
8.	AH	one	22.	IE	five
9.	N1	one	23.	V	five
10.	N2	one	24.	S1	six
11.	T	two	25.	S2	six
12.	UU1	two	26.	IX1	six
13.	UU2	two	27.	IX2	six
14.	UU3	two	28.	K	six

4. Examine the spectrogram to determine the time at which each word in the speech file begins and ends.

5. Mark the phoneme recognition results with the data obtained in step 4.

6. Determine which phonemes in the phoneme recognition results represent "correct" phonemes and which are errors from the acoustic analyzer. This step is simply a best guess and may later be revised.

7. Form a state diagram based upon the results of step 6.

8. Enter the state diagram information into the state machine.

"Zero" is ideally composed of the phonemes "z", "er", and "ou". Table V is a list of phonemes, for the word "zero", which have been extracted from a larger list produced by the acoustic analyzer in processing a speech file containing the words "zero", "one", "two", "three", and "four". The phonemes listed are those which the acoustic analyzer indicated as having the highest correlation for each time segment of the speech file. The numbers 48 through 78 indicate the time slice number since the beginning of the speech file. Recall that each time slice represents 8 milliseconds of speech.

Table V

Phonemes of word "zero"

Segment Number	Phoneme Number	Phoneme Name	Segment Number	Phoneme Number	Phoneme Name
48	2	Z2	64	3	ER1
49	2	Z2	65	4	ER2
50	2	Z2	66	4	ER2
51	1	Z1	67	4	ER2
52	1	Z1	68	4	ER2
53	1	Z1	69	4	ER2
54	1	Z1	70	4	ER2
55	2	Z2	71	4	ER2
56	1	Z1	72	13	UU2
57	2	Z2	73	13	UU2
58	2	Z2	74	16	R
59	1	Z1	75	5	OU1
60	13	UU2	76	5	OU1
61	13	UU2	77	16	R
62	13	UU2	78	16	R
63	3	ER1			

Based upon the information in Table V, we can form the state diagram of Figure 6. Note that the first phoneme of Table V is "Z2" or phoneme 2. Figure 6, the state diagram,

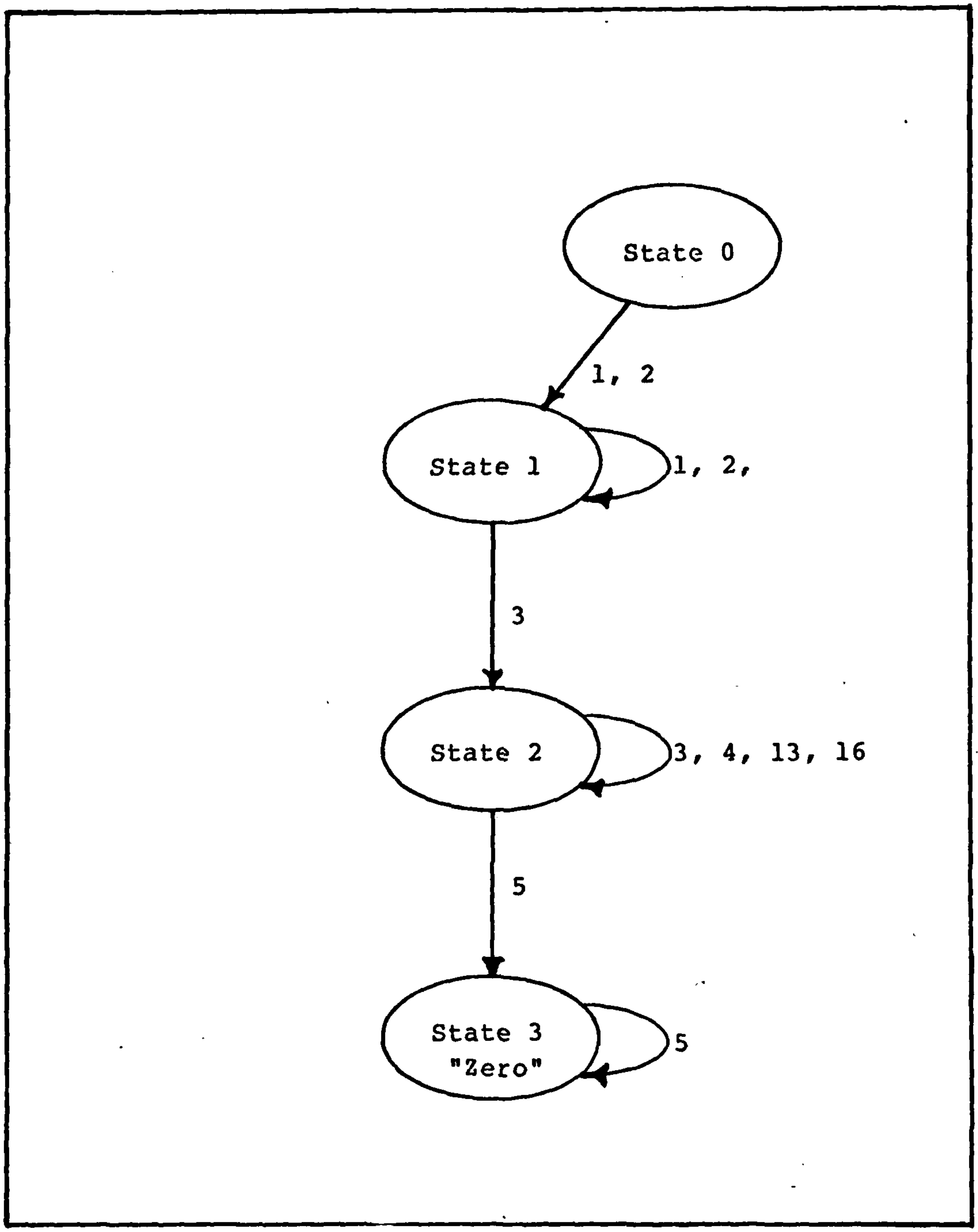


Figure 6. State transitions for "zero"

has been chosen so that Z1 or Z2 will produce a transition to state 1, the start of the "zero" string. Upon reaching state 1, input phonemes 1, 2, and 13 will yield a next state of state 1. Phoneme 13, occurring at time segments 60, 61, and 62 of Table V, is an error from the acoustic processor. However, because we have now seen this phoneme occurring in a "zero" string, we shall include it in the "experience" of the state machine. This is accomplished simply by programming the state machine so that while in state 1, input phoneme 13 produces a next state of state 1. From state 1, an input of "ER" , which is phoneme 3 or 4, will cause a transition to state 2. While in state 2, the machine waits for the occurrence of the final phoneme in the "zero" string, which is an "OU", phoneme 5. At time segments 72, 73, and 74 observe the errors. Before completing the "zero" string, phonemes 13 and 16 appear. We choose to incorporate these by remaining in state 2 and waiting for phoneme 5. When phoneme 5 occurs, the machine transitions to state 3 and produces an output to indicate that the word "zero" has been found.

Several assumptions are implicit in this procedure used to train the machine. The first assumption is that when the machine reaches state 3, the word "zero" has occurred.

Given the nature of the machine this is not necessarily a true assumption. It is possible to step from state 0 to state 3 with only three inputs. Three inputs would only represent 24 milliseconds of speech. Obviously, no one speaks the word "zero" in that length of time. However, we have not observed the sequence to occur other than in the word "zero". The best we can say is that it appears to be a reasonable assumption to believe that upon reaching state 3, the word "zero" has been processed.

Another potentially nettlesome assumption is the error processing. Is it reasonable to simply wait for the correct phonemes to occur when we are observing "errors"? This problem is eliminated by redefining the "errors". The "error" phonemes are not really errors within the context of recognizing the phoneme string as a word. They are simply phonemes which have been observed to occur while processing the word. From the viewpoint of a human observer, the phonemes have no place in the word. However, since we have observed that they occur and do lead to a correct parsing of the word, they will be incorporated in the knowledge of the state machine.

The final step in the training process is to incorporate the state diagram of Figure 6 into the state transition table of the state machine.

Incorporating the second, third, and successive occurences of "zero" into the state table will follow much the same process. Where possible, new error phonemes should be handled by self looping, as was done for phonemes 13 and 16 on states 1 and 2. It is our belief that this procedure will generalize the recognition process so that new occurrences of the word "zero" will eventually produce no changes in the state transition table. If a new path to state 3 were created for each occurrence of "zero" which differed from previous "zeros", we would quickly consume all available memory and simply have several paths which lead to state 3. By attempting to incorporate all possible variations into one path, the path becomes very generalized and permits variations within the experience of the system, yet forces certain criteria to be met.

V. Results and Conclusion

The primary result of this thesis is an algorithm described in previous chapters which may be a useful continuous speech word recognizer. The technique described is well adapted to the data produced by the AFIT acoustic analyzer and appears to have a high potential of functioning to recognize a small vocabulary. For several reasons the technique does not appear capable of handling a large vocabulary. The foremost reason is simply the problem of finding sufficient differences between words. For the small vocabulary considered for this project, the differences between the words are great enough as to present no problem. As the vocabulary increases, it is safe to say that the distance between various words will decrease and the algorithm will encounter difficulty parsing the words. A major difficulty in a larger vocabulary will be the method selected to handle error phonemes.

What can be said of this algorithm is that it successfully parses each utterance which has been incorporated into the state transition table. A total of fifteen utterances have been incorporated for the words "zero" through "six". No utterance has been recognized before the state machine was trained to recognize that particular utterance. It is understood that a machine which must be trained for each individual pattern it is to recognize is useless. However, it appears that the changes necessary to incorporate each utterance are decreasing as the incorporated experience grows.

VI. Recommendations

The most important recommendation which may be made to anyone contemplating continuing this project or another of similar nature is that the project should not be considered unless the acoustic analyzer is under the control of the individual attempting word recognition. The recognition work attempted under this thesis has done little more than verify that the software functions as intended. It does not suffice as as a test of the recognition algorithm.

The reasons for the recommendation are relatively simple. If another individual is extending the acoustic analyzer, he or she must be free to make changes to suit their goals of phoneme recognition. In general, the goals of phoneme recognition and word recogniton are compatible. Unfortunately, the word recognition goal requires a stable, copious output from the acoustic analyzer to build and verify the word recognition process. Neither goal of stable or copious output will be attained while someone else is modifying the acoustic analyzer.

The next recommendation is incorporate more utterances into the state transition table described and verify or disprove the algorithm described. I estimate that the word recognizer is likely to begin functioning before 10 utterances of each word have been incorporated in the state table. That is, begin functioning in the sense that words will be recognized based on the templates already incorporated in the state table.

Bibliography

1. Klatt, Dennis H. "Overview of the ARPA Speech Understanding Project" in Trends in Speech Recognition, edited by Wayne A. Lea. Englewood Cliffs, N.J.: Prentice-Hall, Inc., 1980.

2. Felkey, Mark A. Automatic Recognition of Phonemes in Continuous Speech. MS thesis. Wright-Patterson AFB, Ohio: School of Engineering, Air Force Institute of Technology, December 1980.

3. Potter, R.K., George Kopp, and Harriet Green. Visible Speech. New York: D. Van Nosrand Company, Inc., 1947.

4. Hensley, William R. Computer Identification of Phonemes in Continuous Speech. MS thesis. Wright-Patterson Air Force Base, Ohio: School of Engineering, Air Force Institute of Technology, December 1976.

5. Barret, William A. and John D. Couch. Compiler Construction: Theory and Practice. Science Research Associates, Inc., 1979.

6. Fredal, Dan and George C. Beasley. An Analog Speech I/O Channel for the Nova 2 Computer Based on the S-100 Bus. MS thesis. Wright-Patterson Air Force Base, Ohio: School of Engineering, Air Force Institute of Technology, March 1981.

7. Beek, Bruno, Edward Newberg, and David Hodge. "An Assessment of the Technology of Automatic Speech Recognition for Military Applications," IEEE Transactions on Acoustics, Speech, and Signal Processing, ASSP-25: 310-322 (Aug 1977).

APPENDIX A

COMPUTER PROGRAM NOTES

The program described is a single input state machine of 256 states. Each state has 40 possible next states, based upon 40 different input phonemes.

Combined in one program are all the tools needed to execute the state machine, to edit and change any information contained in the state table, to alter the output information from the machine and to transfer the next state tables to the printer in a readable form.

Upon executing the program, there is a prompt line which is "Which state machine?" The proper response is a single character such as "1" or "A", etc. The character will be incorporated into the file name, STMACHx.DATA, where x is the input character. The program will attempt to open that file. If the file is opened successfully, the program continues, otherwise the prompt is repated. Entering a period will cause the program to ignore any data files and continue but the memory locations reserved for the state machine will all be set to zero. This is the proper condition for defining a new state machine, you might say a "clean slate". The previously described action is accomplished by the init procedure.

When the state machine has been loaded to memory, a prompt line appears with the options, "E, P, Q, X, V, W".

Typing "E", calls the editor. The editor allows the user to freely examine and modify the contents of the state machine. The screen display is similar to Figure 9. The numbers 1 through 40 are the possible input phonemes. The other information, mostly zeroes, are the next states. While in this state, input phoneme 2 will cause a jump to state 1, input phoneme 4 will cause a jump to state 25. The cursor may be moved from one phoneme to another with the Apple Pascal editor cursor movement commands. The move keys are cntrl-O, cntrl-L, right arrow, left arrow, and return. Cntrl-O moves the cursor up and cntrl-L moves the cursor down. The phoneme to the immediate right of the cursor may be modified by by entering digits. You may terminate the entry with a return or a cursor move. The editor may also be entered from either execution mode.

Typing "P", for print, will result in the program asking which states are to be printed. If it were told to print states 1 through 25, the result would be an output similar to Figure 9, repeated 25 times. Three next state tables are printed on each page.

Typing Q will start the process of exiting the program. You will be asked whether the state machine in memory should be saved. If no data is to be saved on disk, the program may be exited by typing a series of "Q's".

```
  1     2     3     4     5     6     7     8     9    10
  0     1     1    25     3     0     0     0     0     0

 11    12    13    14    15    16    17    18    19    20
  0     0     0     0     0     0     0     0     0     0

 21    22    23    24    25    26    27    28    29    30
  0     0     0     0     0     0     0     0     0     0

 31    32    33    34    35    36    37    38    39    40
  0     0     0     0     0     0     0     0     0     0

State: 3
Output word is [no output]
```

Figure 7. Next state screen display

Typing "X" will put the machine into an execution mode in which the only display is the output words and input phonemes. No next state table is displayed.

Typing "V" places the program in an execution mode in which the next state table is constantly displayed and updated for each state transition. For either execution mode the phoneme inputs may come from the keyboard or a file.

Typing "W" allows entry of the text of the output words. A word is displayed for each state while editing or executing. The normal output is "no output".

The user may exit the program by typing a aeries of "q's". The number varies depending upon the exact status of the program. If you are using the editor from inside the state machine execution module, four "q's" may be required to get out of the program.

APPENDIX B

COMPUTER PROGRAM

```
{
*************************************************************************

*************************************************************************
}

{ $l printer:}
program state;
uses io, applestuff;
   const
      starray = true;
      outarray= false;
      maxphons=39;
      stline=16;
      wordline=17;
      esc=';
   type
      stmach = record
                  nxtst:packed array[0..255,0..maxphons] of 0..255;
                  word :packed array[0..255] of 0..31;
                  wtext:packed array[0..40,1..10] of char;
               end;
      numtype = '0'..'9';
      numsetype = set of numtype;
   var
      sm:stmach;
      lines:integer;
      xmode:boolean;
      numset:numsetype;
      name:string;
      ch:char;
      visible,quit:boolean;
      eoline:char;
      phonum,edstate:integer;
      xpos:packed array[0..maxphons] of 0..maxphons;
      ypos:packed array[0..maxphons] of 0..maxphons;
      inphons:text;
```

```
{
************************************************************************

          Init  performs  the  task  of  initializing   global
          variables and and setting up the state machine.

************************************************************************
}


procedure init;
var
   ch:char;
   result:integer,
   smfile: file of stmach;
   i,j,pnum:integer;
begin
   visibie:=false;
   quit:=false;
   edstate:=0;eoline:=chr(13);
   numset:=['0'..'9'];

{
........................................................................
          Go  to  the console and ask which state machine file
          is to be used.  The  state  machine  files  are  all
          identified as  STMACHx.DATA.   The "x" is any single
          character other than a "." which the  Pascal  system
          will accept.    Typing  a  period in response to the
          prompt  "Which  state  machine"  will  result  in
          initializing  the  state  machine  so  that all next
          states are State 0.
........................................................................
}

   repeat
      writeln('which state machine? ');
      read(keyboard,ch);
      if eoln(keyboard) then ch:='0';
      name:='stmach .data';
      name[7]:=ch;
      (*$i-*)
      reset(smfile,name);
      (*$i+*)
      result:=ioresult;
      if (result<>0) and (ch<>'.') then
          writeln(name,' not on prefixed volume');
   until (result=0) or (ch='.');
```

```
{
.........................................................................
          Now read in the state machine or set every thing  to
          zero.
.........................................................................
}

   if result=0 then
      begin
         get(smfile);
         sm:=smfile^;
      end
   else fillchar(sm,sizeof(sm),chr(0));
   pnum:=-1;

{
.........................................................................
          Set  up  the  arrays,  xpos  and  ypos, with the x y
          co-ordinates where the next state will be  displayed
          for each input phoneme.
.........................................................................
}

   for i:=0 to 3 do
      begin
         for j:=1 to 10 do
            begin
               pnum:=pnum+1;
               xpos[pnum]:=(j-1)*4;
               ypos[pnum]:=i*3 + 4;
            end;
      end;
end;
```

```
{
**************************************************************************

          Procedure  savesm saves the state machine by writing
          it to a data file.  It defaults to using the name of
          the state  machine  which  was  read  from  disk  at
          initialization.   If it is desired to save the state
          machine under a new name, simply type  a  character.
          The  character  will  be  incorporated into the data
          file name as STMACHx.DATA where "x" is the character
          entered.

**************************************************************************
}


procedure savesm;
var
   smfile:file of stmach;
   result:integer;
   quit:boolean;
   ch,ch1:char;
begin
   quit:=false;
   repeat
      repeat
         clear;
         writeln('save the file as ',name,'?');
         writeln('<return> to use this name or enter char');
         read(keyboard,ch);
         if not eoln(keyboard) then name[7]:=ch;
         writeln('saving as ',name);
         writeln('ok?  y/n/q');
         ch1:=getchar(['q','Q','y','Y','n','N']);
         case ch1 of
            'q','Q': begin
                        quit:=true;
                     end;
         end;   (*case*)
      until (ch1='Y') or (ch1='y') or quit;
      if not quit then
         begin
            (*$I-*)
            rewrite(smfile,name);
            smfile^:=sm;
            put(smfile);
            result:=ioresult;
            close(smfile,lock);
            (*$i+*)
         end;
```

```
    until quit or (result=0);
end;
```

```
{
*************************************************************************

          The topline  procedure  writes  a  list  of  options
          acrossthe top of the screen.  Called by pntscrn.

*************************************************************************
}


procedure topline;
begin
if not xmode then promptat(0,'S(tate, O(utword, Q(uit edit');
end;
```

```
{
****************************************************************************

          Procedure  wrtst  writes  the state machines current
          state near  the  bottom  of  the  screen  while  the
          program is executing in ?he visible mode.

****************************************************************************
}


procedure wrtst;
begin
   promptat(stline,'State: ');
   write(edstate);
end;
```

```
{
***************************************************************************

          The editor function allows the cursor to be moved
          freely from one phoneme to another on the screen.
          Keyboard inputs to move the cursor actually only
          change the phoneme number pointed at.  Procedure
          curspos accepts the phoneme number and moves the
          cursor to the correct position on the screen.

***************************************************************************
}


procedure curspos(var inphon:integer);
begin
   gotoxy(xpos[inphon],ypos[inphon]);
end;
```

```
{
**************************************************************************

          Procedure wrtword  is  a  screen  update  procedure.
          Each  time  a  state change is accomplished, wrtword
          interrogates the state machine  for  an  output  and
          writes it to the screen.

**************************************************************************
}


procedure wrtword;
begin
   promptat(wordline,'Output word is ');
   write('[',sm.wtext[sm.word[edstate]],']');
   curspos(phonum);
end;
```

```
{
***************************************************************************

          Procedure pntscrn  paints  the  screen.  This is the
          primary display procedure  while  operating  in  the
          visible mode.    In  the visible mode, the screen is
          updated for each state change.  The  screen  display
          presents  the  next  state  for  each possible input
          phoneme.   In  addition,  the  screen  displays  the
          output for the current state.

***************************************************************************
}


procedure pntscrn;
var
   j,phonumb:integer;
begin
   clear;
   topline;
   gotoxy(0,3);

{
...........................................................................
          The  first  for loop involving phonumb writes a line
          which labels the next states.   On  the  first  pass
          through the "for j:=1" loop the labels are 1 through
          10.  On  the  next pass, 11 through 20, etc.  Fourty
          next states will be  displayed  and  labeled.   The
          second  for  loop  involving phonumb actually writes
          the next states.
...........................................................................
}

   for j:=1 to 4 do
      begin
         for phonumb:=1+(j-1)*10 to j*10 do
            write(phonumb:4);
            writeln();
         for phonumb:=1+(j-1)*10 to j*10 do
            write(sm.nxtst[edstate,phonumb-1]:4);
            writeln();
            writeln();
      end;
   wrtst;
   wrtword;
   curspos(phonum);
end;
```

```
{
************************************************************************

          The editor procedure allows the user access  to  all
          portions of the state machine to permit modification
          of the  next  states  and  outputs.   The constants
          declarations it contains imbedded control characters
          which are specific to cursor movement from the Apple
          keyboard.  The declarations are the  characters  for
          up, down, right, and left cursor movement.

************************************************************************
}


procedure editor;
const
   up=' ';                (*cntrl-O*)
   down=' ';              (*cntrl-L*)
   right=' ';             (*cntrl-U*)
   left=' ';              (*cntrl-H*)
var
   promptline:integer;
   xcord,ycord,nxtst,wordnum:integer;
   quit:boolean;
```

```
{
******************************************************************************

          Mover is passed characters which are keyboard inputs
          intended to  move  the  cursor.   It  uses  modulo
          arithmetic to calculate which phoneme the cursor has
          been placed on.   This  information  is  calculated
          based  upon  the  fact  that  there  are 10 phonemes
          displayed on each line of the screen display.  After
          calculating the new cursor position, the  cursor  is
          moved by calling curspos.

******************************************************************************
}


procedure mover(var ch:char);
begin
   case ch of
      down: begin
               phonum:=(phonum+10) mod (maxphons+1);
               curspos(phonum);
            end;
        up: begin
               phonum:=(phonum+30) mod (maxphons+1);
               curspos(phonum);
            end;
     right: begin
               phonum:=(phonum+1) mod (maxphons+1);
               curspos(phonum);
            end;
      left: begin
               phonum:=(phonum+maxphons) mod (maxphons+1);
               curspos(phonum);
            end;
   end;
end;
```

```
{
*************************************************************************

          The  statjmp  procedure permits modification of each
          next state  while  in  the  edit  mode.     Statjmp
          determines  which next state to alter based upon the
          cursor position.  Statjmpm is  called  by  the  main
          editor   procedure   when  the  character  from  the
          keyboard is a numeral.  When the input is a numeral,
          there is an assumption that the next  state  pointed
          to by  the cursor is to be changed.  Once statjmp is
          entered,  the  procedure  performs  a  character  to
          integer  conversion  in  the repeat until loop until
          the user moves the cursor with a return, a right  or
          left arrow, or a control O or L (Cntrl-O and cntrl-L
          move the  cursor  up  and  down respectively).  Once
          entered the procedure reads the keyboard directly.

*************************************************************************
}


procedure statjmp(var ch:char);
var
   tot:integer;
   quit:boolean;
begin
   quit:=false;
   tot:=0;
   repeat

{
.........................................................................
          The character to integer conversion is  accomplished
          by  by  taking  the  ordinal  value  of  the  input
          character and subtracting the form  it  the  ordinal
          value of the character "0" which is decimal 48.  The
          mod  256 operation prevents the input from exceeding
          the maximum state number and crashing the program.
.........................................................................
}

      tot:=(tot*10 + ord(ch)-48) mod 256;
      write(tot:4);
      curspos(phonum);
      read(keyboard,ch);
      case ch of
         up,down,right,left:quit:=true;
      end;
```

```
{
.....................................................................
         Set the next state to the value  obtained  from  the
         character to integer conversion.
.....................................................................
}

     sm.nxtst[edstate,phonum]:=tot;
   until quit or eoln(keyboard);

{
.....................................................................
         If the input number was terminated by a return, then
         increment the  phoneme number, i.e.  move the cursor
         to the next higher numbered input phoneme.   If  the
         return  was  not used then we got an up, down right,
         or left cursor movement.  Update the cursor position
         with mover.
.....................................................................
}

   if not quit then phonum:=(phonum+1) mod (maxphons+1)
   else mover(ch);
   curspos(phonum);
end;
begin                              (*editor*)
   promptline:=22;
   phonum:=0;
   pntscrn;
   quit:=false;
   repeat
      read(keyboard,ch);
      case ch of
         's','S': begin

{
.....................................................................
         Get an integer from the user to indicate which state
         is to be  viewed  or  edited.   After  getting  the
         integer, update the screen.
.....................................................................
}

                     promptat(promptline,'which state? ');
                     readln(edstate);
                     edstate:=edstate mod 256;
                     pntscrn;
                  end;
         'q','Q': begin
```

```
{
.....................................................................
         Leave  the  editor and return to the procedure which
         called the editor.
.....................................................................
}

                    quit:=true;
                 end;
        'o','O': begin

{
.....................................................................
         This function wants an integer.    I  have  assigned
         word 0 to be "no output".  The integer is used as an
         index  into  an  array  of characters to display the
         text of the output word.  To display the output word
         "zero", enter a "1"  here,  "one"  is  displayed  by
         entering "2", etc.
.....................................................................
}

                    promptat(promptline,'What is output word number? ');
                    readln(wordnum);
                    gotoxy(0,promptline);
                    crt(eraseol);
                    wordnum:=wordnum mod 41;
                    sm.word[edstate]:=wordnum;
                    wrtword;
                 end;
        '+',';': begin

{
.....................................................................
         Increment  edstate and update the screen to show the
         new state.  Edstate is the variable which determines
         the current state of the state machine while editing
         or executing.
.....................................................................
}

                    edstate:=(edstate +1) mod 256;
                    pntscrn;
                 end;
            '-': begin
```

```
{
..............................................................................
          Decrement edstate and update the screen.
..............................................................................
}

                     edstate:=(edstate + 255) mod 256;
                     pntscrn;
                  end;
   up,down,right,
            left: mover(ch);
'0','1','2','3','4','5',
'6','7','8','9' : begin

{
..............................................................................
          There is a presumption that the next state is to  be
          altered when the input is a numeral.
..............................................................................
}

                     statjmp(ch);
                  end;
      end;  (*case*)
   until quit;
end;                                    (*editor*)
```

```
{
******************************************************************************

          Statmach  is the procedure which exercises the state
          machine.  As  mentioned  previously,  the  variable
          edstate  holds  the  current  state  of the machine.
          This program began life simply as an editor for  the
          state machine.      It  beoame  apparent  that  many
          benefits  might  be  derived  from   combining   the
          execution and editing functions.

               The  integer,  laststat,  holds  the last state
          that the state machine was in.  If laststat does not
          equal edstate, the screen is updated to display  the
          current state.       Otherwise,   the   screen   is
          undisturbed. This is  done  because  rewriting  the
          screen is time consuming.

******************************************************************************
}


procedure statmach;
var
   ch:char;
   running,goodnum,quit:boolean;
   laststat,inphon:integer;
```

```
{
******************************************************************************

          The  prompt  procedure  writes  the  top line of the
          screen in the visible execute mode  to  display  the
          input options.

******************************************************************************
}


procedure prompt;
begin
   promptat(0,'   0..9, E(dit, Q(uit, R(un ');
end;
```

```
{
*************************************************************************

          Procedure run is used to open a text file and set up
          conditions  necessary  to  read  input phonemes from
          that file.  The name of the file  is  obtained  from
          the keyboard.    The  procedure  is repeated until a
          file is successfully opened or the  user  enters  an
          empty line.    A  empty  line  is  a  return with no
          preceding characters.

*************************************************************************
}


procedure run;
var
  name:string;
  j:integer;
begin
   clear;
   repeat
      writeln('Which file?');
      readln(name);
      j:=length(name);
      (*$i-*)
      reset(inphons,name);
      (*I+*)
   until (j=0) or (ioresult=0);

{
.........................................................................
          Set the boolean variable, running, true.     The
          variable, lines, is used to display the number of
          phoneme inputs read from the input file.
.........................................................................
}

   running:=true;
   lines:=0;
end;
```

```
{
****************************************************************************

        The getphon function probably contains the most
        obsure code in this program. While reviewing this
        procedure, keep firmly in mind the function of the
        procedure.  Its function is to obtain an integer,
        result, to pass back to the procedure statmach.
        Result is an integer which is the number of the
        current input phoneme. The input phoneme may come
        from the keyboard or a file. The boolean variable,
        running, determines whether getphon goes to the
        keyboard or a file for the phoneme. Getphon is a
        boolean function. The value it returns indicates
        whether or not a good number has been obtained. A
        good number has not been obtained when the user
        enters anything other than digits. Running is set
        to false by reaching the end of the input file or
        detecting that a key has been pressed.

****************************************************************************
}

function getphon(var ch:char;var quit:boolean;var result:integer):boolean;
var

{
............................................................................
        Number is used internally by getphon to determine
        the final result of the function.   Number simply
        indicates whether getphon has processed an integer
        or called another function such as the editor.
        While in getphon, if all keyboard entries are
        digits, number will remain true.

             Savestat is used to remember the current state
        the machine has reached while executing. Savestat
        is used only when getphon calls the editor. Use of
        this variable permits the user to step around
        through the editor and make changes but upon exiting
        the editor, the machine returns to same state it was
        in on entry to the editor.
............................................................................
}

   savestat:integer;
   number:boolean;
   first,second:integer;
   third:real;
```

```
begin
   result:=0;
   number:=true;
   if not running then
      begin
         prompt;
         repeat
            read(keyboard,ch);
            case ch of
         '0','1',
         '2','3',
         '4','5',
         '6','7',
         '8','9':begin

{
.....................................................................
         Perform character to integer conversion
.....................................................................
}

                     result:=result*10+ord(ch)-48;
                     if result<1 then result:=1;
                     if result>40 then result:=40;
                     gotoxy(37,23);
                     write(result:2);
                  end;
         'e','E':begin

{
.....................................................................
         Call the editor
.....................................................................
}

                     savestat:=edstate;
                     xmode:=false;
                     editor;
                     xmode:=true;
                     if edstate<>savestat then
                        begin
                           edstate:=savestat;
                           pntscrn;
                        end;
                     promptat(0,'Execution mode');
                     number:=false;
                  end;
         'q','Q':begin
```

```
{
.......................................................................
        Quit the getphon procedure and set quit true.   This
        will  cause  an  exit  from  the statmach procedure.
        Leaving statmach produces a return to the main  loop
        of the program.
.......................................................................
}

                    number:=false;
                    quit:=true;
                    running:=false;
                 end;
          'r','R':begin

{
.......................................................................
        Call run and open a phoneme input file.
.......................................................................
}

                    run;
                 end;
              end;                  (*case*)
           until eoln(keyboard) or not number;
           if number and (result>0) then
              begin
                 result:=result-1;
                 getphon:=true;
              end
           else getphon:=false;
        end
     else                           (* running is true *)
        begin

{
.......................................................................
        If we haven't reached the end of the file and no key
        has  been pressed then read a phoneme from the input
        file and pass it back to statmach.   If  a  key  has
        been pressed or we've reached the file end, then set
        running false  and  close  the  input  file.   With
        running false, the next entry to getphon  will  have
        getphon expectig input from the keyboard.
.......................................................................
}

           if not eof(inphons) and not keypress then
              begin
                 readln(inphons,first,second,third);
```

```
                    result:=first;
                    getphon:=true;
                    lines:=lines+1;
                    gotoxy(33,23);
                    write(lines);
                    gotoxy(37,23);
                    write(first:2);
                 end
             else
                begin
                   running:=false;
                   close(inphons,lock);
                end;
          end;
      end;
```

```
{
**************************************************************************

          Statmach is the main procedure which exercises the
          state machine. It remains in a loop of getting an
          input phoneme and and then interrogating the next
          state table and then returning for a new input
          phoneme. While in the procedure, getphon, it is
          possible to enter the editor and modify the next
          state table.

**************************************************************************
}


begin                               (* statmach *)
   edstate:=0;
   laststat:=0;
   quit:=false;
   running:=false;
   clear;
   if visible then pntscrn;
   repeat
       goodnum:=getphon(ch,quit,inphon);
       if goodnum then
          begin
             edstate:=sm.nxtst[edstate,inphon];
             if visible and (laststat<>edstate) then
                begin
                   pntscrn;
                   laststat:=edstate;
                end;
             if not visible then
                begin
                   promptat(18,'output word is ');
                   write('[',sm.wtext[sm.word[edstate]],']');
                end;
          end;
   until quit;
end;                                        (*statmach*)
```

```
{
***********************************************************************

          Write a prompt line for the program's main loop.

***********************************************************************
}


procedure screen;
begin
   clear;
   promptat(0,'E(dit, P(rint, Q(uit, X(ecute, V(is, W(ords');
end;
```

```
{
**************************************************************************

          The  vocab  procedure  permits  defining the text of
          each word in the vocabulary of the state machine.

**************************************************************************
}


procedure vocab;
var
   wordnum,i,l:integer;
   name:string;
begin
   repeat
      clear;
      promptat(0,'Vocabulary text entry. Neg # to exit.');
      promptat(1,'Which word number?');
      writeln();
      readln(wordnum);
      if (wordnum>=0) and (wordnum<=40) then
         begin
            promptat(1,'Word ');
            write(wordnum,' is ','[',sm.wtext[wordnum],']');
            promptat(2,'Enter text. Max is 10 characters');
            writeln();
            readln(name);
            l:=length(name);
            if l>0 then
               begin
                  for i:=1 to l do
                     sm.wtext[wordnum,i]:=name[i];
                  for i:=l+1 to 10 do
                     sm.wtext[wordnum,i]:=' ';
               end;
         end;
   until wordnum<0;
end;
```

```
{
***************************************************************************

        The print procedure permits printing the next  state
        information for  an arbitrary number of states.  The
        information is printed in the format  used  for  the
        screen display.    Three  screens of information are
        printed on each page.

***************************************************************************
}


procedure print;
var
   j,k,phonumb:integer;
   counter,state,start,endst:integer;
   answer:char;
   printer:text;
begin
   (*i-*)
   rewrite(printer,'printer:');
   (*i+*)
   if ioresult=0 then
      begin
         writeln;
         write('Enter beginning state to print ');
         readln(start);
         write('Enter end state to print ');
         readln(endst);
         counter:=-1;
         write(chr(12));
         for state:=start to endst do
            begin
               counter:=(counter + 1) mod 3;
               if counter=0 then
                  begin
                     for k:=1 to 7 do
                        write(printer,chr(13));
                  end;
               for j:=1 to 4 do
                  begin
                     for phonumb:=1+(j-1)*10 to j*10 do
                        write(printer,phonumb:4);
                        writeln(printer);
                     for phonumb:=1+(j-1)*10 to j*10 do
                        write(printer,sm.nxtst[state,phonumb-1]:4);
                        writeln(printer);
                        writeln(printer);
                  end;
```

```
                    writeln(printer,'State: ',state);
                    writeln(printer,'Output word is [',
                          sm.wtext[sm.word[edstate]],']');
                    for k:=1 to 4 do
                       write(printer,chr(13));
                    if counter=2 then write(printer,chr(12));
                end;
             if keypress then
                begin
                   writeln;
                   write('Quit printing?  y/n');
                   readln(answer);
                   if (answer='y') or (answer='Y') then
                      begin
                         writeln(chr(12));
                         exit(print);
                      end;
                end;
          end;
      end;
```

```
{
**********************************************************************

          The main procedure is a keyboard input routine
          followed by a case statement. The user may edit or
          execute from the main loop.

**********************************************************************
}


begin
   init;
   repeat
      screen;
      ch:=getchar(['q','Q','p','P','x','X','e','E','v','V','w','W']);
      case ch of
         'e','E':begin
                    xmode:=false;
                    editor;
                 end;
         'p','P':print;
         'q','Q':begin
                    quit:=true;
                 end;
         'x','X':begin
                    visible:=false;
                    xmode:=true;
                    phonum:=0;
                    statmach;
                 end;
         'v','V':begin
                    visible:=true;
                    xmode:=true;
                    phonum:=0;
                    statmach;
                 end;
         'w','W':begin
                    vocab;
                 end;
      end;    (*case*)
   until quit;
   savesm;
   clear;
end.
```

```
{
**************************************************************************

        The following intrinsic unit is a separately
        compiled unit which must be placed in the system
        library for the main program to use. Alternatively,
        the procedures may be extracted from their context
        in this intrinsic unit and compiled as in line
        procedures in the main program.

             The following procedures were extracted from a
        program titled, DISKIO, on the APPLE3: diskette
        supplied with the Apple Pascal system. The
        procedures are useful in that they provide terminal
        independent screen output as well as "bullet proof"
        input by the procedure getchar;

**************************************************************************
}


{$s+}
unit io; intrinsic code 25 data 26;

   interface
   type
      crtcommand = (eraseos,eraseol,up,down,right,left,leadin);
      setofchar = set of char;
   var
       f: file;
       crtinfo: packed array[crtcommand]of char;
       prefixed: array [crtcommand] of boolean;

       procedure beep;
       procedure crt(c: crtcommand);
       procedure promptat(y: integer; s: string);
       procedure clear;
       function getchar(okset: setofchar): char;

   implementation
      var
         buffer: packed array[0..511] of char;
         i,byte: integer;
```

```
{
***********************************************************************

          The  function of beep is to send an ASCII charater 7
          to the terminal and ring the bell

***********************************************************************
}


procedure beep;
begin
   write(chr(7));
end;
```

```
{
**********************************************************************

         The crt procedure permits clearing to the end of the
         current line, clear to  end  of  screen  and  cursor
         moves up, down, right, and left.

**********************************************************************
}


procedure crt;
(*  crt commands are: eraseos,eraseol,up,down,right,left.        *)
begin
  if prefixed[c] then unitwrite(1,crtinfo[leadin],1,0,12);
  unitwrite(1,crtinfo[c],1,0,12);
end;
```

```
{
****************************************************************************

          Promptat  writes message to the screen on aspecified
          line.

****************************************************************************
}


procedure promptat;
begin
  gotoxy(0,y);
  write(s);
  crt(eraseol);
end;
```

```
{
**************************************************************************

          Getchar returns a single character.  The  programmer
          specifies  an  acceptable  set  of  inputs  when the
          procedure is called.  If a user attempts to  type  a
          character  not  specified by the programmer, getchar
          beeps and waits for a proper input.

**************************************************************************
}


function getchar;
(*  get a character, beep if not in okset, echo only if printing  *)
var ch: char;
    good: boolean;
begin
  repeat
    read(keyboard,ch);
    if eoln(keyboard) then ch:=chr(13);
    good:= ch in okset;
    if not good then beep
      else if ch in [' '..'}'] then write(ch);
  until good;
  getchar:=ch;
end;
```

```
{
**************************************************************************

          Clear provides a terminal independent screen clear.

**************************************************************************
}


procedure clear;
begin
   gotoxy(0,0);
   crt(eraseos);
end;
```

```
{
***********************************************************************

          The code in this main procedure of the unit  is  run
          automatically  by  the Pascal system before the main
          program is  loaded.    This  code  reads  the  file,
          SYSTEM.MISCINFO  to obtain the information necessary
          to control the terminal.

***********************************************************************
}


begin
  reset(f,'*system.miscinfo');
  i:=blockread(f,buffer,1);
  close(f);
  byte:=ord(buffer[72]); (* prefix information byte *)
  crtinfo[leadin]:=buffer[62];  prefixed[leadin]:=false;
  crtinfo[eraseos]:=buffer[64]; prefixed[eraseos]:=odd(byte div 8);
  crtinfo[eraseol]:=buffer[65]; prefixed[eraseol]:=odd(byte div 4);
  crtinfo[right]:=buffer[66];   prefixed[right]:=odd(byte div 2);
  crtinfo[up]:=buffer[67];      prefixed[up]:=odd(byte);
  crtinfo[left]:=buffer[68];    prefixed[left]:=odd(byte div 32);
  crtinfo[down]:=chr(10);       prefixed[down]:=false;
end.
```

AD-A115 615 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOO--ETC F/G 17/2
THE RECOGNITION OF WORDS FROM PHONEMES IN CONTINUOUS SPEECH.(U)
DEC 81 C A BAKER
UNCLASSIFIED AFIT/GE/EE/81D-9 NL

2 OF 2
AD A
115615

END
DATE
FILMED
07-82
DTIC

Vita

Claude A. Baker was born 12 January 1946 in Maywood, California. He graduated from Abilene High School, Abilene, Texas in 1963 and attended L.A. Pierce College in Woodland Hills, Ca for approximately one year before enlisting in the U.S. Air Force in 1965. After serving as a Special Electronics Technician for nine years, he was selected for the Airman Education and Commissioning Program and attended the University of Texas at Austin from 1974 to 1976, graduating with the degree of Bachelor of Science in Electrical Engineering in May 1976.

Upon commissioning, he was assigned to the Oklahoma City Air Logistics Center. Initial duties involved the B-1 bomber acquisition. More recent duties included two years experience designing and modifying automated industrial systems used in jet engine overhaul at the Air Logistics Center.

Permanent address: 3933 Don Juan
Abilene, Texas 79605

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GE/EE/81d-9	2. GOVT ACCESSION NO. AD-A115615	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) THE RECOGNITION OF WORDS FROM PHONEMES IN CONTINUOUS SPEECH		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) CLAUDE A. BAKER Captain, USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS AF Institute of Technology (AFIT-EN) Wright-Patterson AFB OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Department of Electrical Engineering Air Force Institute of Technology		12. REPORT DATE December 1981
		13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

Dean for Research and Professional Development
Air Force Institute of Technology (ATC)
Wright-Patterson AFB, OH 45433

AIR FORCE INSTITUTE OF TECHNOLOGY (ATC)
WRIGHT-PATTERSON AFB, OH 45433

18. SUPPLEMENTARY NOTES

Approved for public release IAW AFR 190-17
~~FREDRICK C. LYNCH, Major~~, USAF
~~Director of Public Affairs~~

14 JUN 1982

APPROVED FOR PUBLIC RELEASE: IAW AFR 190-17

Lynn Wolaver
LYNN E. WOLAVER
Dean for Research and Professional Development

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Speech recognition
Continuous speech recognition
Connected speech recognition
phoneme recognition

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Thesis attempts continuous speech recognition by parsing phoneme strings to words. The phoneme strings are obtained from the Air Force Institute of Techonology acoustic analyzer. Vocabulary is limited to the 10 digits. Recognition is accomplished by a software simulation of a finite state ~~machine~~ automata. State tables are read in as a datasfile and manipulated without recompiling.

DD FORM 1473, 1 JAN 73 — EDITION OF 1 NOV 65 IS OBSOLETE

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)