TOOLS AND UTILITIES FOR THE DEVELOPMENT OF SPEECH RECOGNITION SYSTEMS

Roger Werner, David Kotick, Dana Smith
Naval Training Systems Center
Orlando, Florida

ABSTRACT

The reliability of speech recognition systems can be enhanced considerably through the use of pre-defined wordsets and phraseology syntaxes. Historically, syntax structures have been defined manually using hand-drawn state diagrams, which in turn are converted manually to ASCII files. When many words, nodes or connections between nodes are involved, the processes of defining, debugging, and modifying a syntax can be quite tedious. A tool has been developed which automatically generates a graphical state diagram from an ASCII syntax definition file. It can also check for various hazards in the structure, check a list of phrases for compliance with the syntax, count words, and write and check against word dictionaries. This automates much of the clerical tedium of dealing with syntax structures and phrase lists. A further enhancement, which allows graphical editing of the state diagram, and subsequent automatic generation of the descriptive ASCII file is in the design phase. An additional real-time nodal flow analyzer is also included in the tool package.

With the development and addition of post-processed phraseology checking and word scoring/word thresholding utilities, apparent voice system accuracy and user acceptability may be significantly increased. This paper discusses the use of these speech recognition tools and utilities in the NTSC Under Ice Navigation Trainer Test-bed.

ABOUT THE AUTHORS

Roger Werner received his BS degree in Chemistry from Tennessee Technological University in 1975. Following a tour in the Navy, he completed an MS degree in Engineering Math and Computer Systems from the University of Central Florida. Since then he has been working in applications programming with emphasis in graphics, networking, user interfaces, and, most recently, in speech technology.

David Kotick is an Electronics Engineer with the Systems and Computer Technology Division of the Naval Training Systems Center. His principal responsibilities include the integration, evaluation and design of microcomputer-based board-level products. He has received both a B.S.E. and M.S.E. in Electrical Engineering from the University of Central Florida. He has worked in the area of real-time multi-microprocessing at NTSC since 1983. Recent work includes speech system syntax tool development, speech post-processing algorithms and word-scoring methods in simulation systems incorporating speech technology.

Dana Smith is a graduate of the University of Central Florida where she earned a Bachelors degree in Electrical Engineering in 1983. Since that time she has worked at the Naval Training Systems Center in Orlando, FL. She has a broad range of experience in simulation technology. One of her most memorable projects is the AV-8B Harrier Night Attack Weapon System Trainer, on which she worked with the U.S. Marines to develop a new weapon system. During the past year, Ms. Smith began work in the field of speech technology. She is currently involved in a number of on-going speech related projects.

TOOLS AND UTILITIES FOR THE DEVELOPMENT OF SPEECH RECOGNITION SYSTEMS

Roger Werner, David Kotick, Dana Smith
Naval Training Systems Center
Orlando, Florida

THE PROBLEM

The reliability of speech recognition systems can be considerably enhanced through the use of pre-defined wordsets and phraseology syntaxes. Smaller vocabularies imply fewer distinctions which must be made in order to decide which word has been spoken. Decision fanout at any given point in a phrase can be further reduced through use of a pre-defined syntax. For example, if only some subset, A, of words of a total vocabulary, V, are allowed to be first words of a phrase, then the speech-recognition system need only compare against words in set A for the first word. Subsequently, depending on which first word was recognized, there may be further subsets of words which the syntax allows to follow each first word, and so forth. In this manner, the processing required to recognize a sequence of words, which must form one of a limited set of acceptable phrases, is considerably reduced in comparison to that which would recognize any sequence of words from a given set (many of which sequences could be nonsensical). Using a syntax improves the reliability of a speech recognition system because (1) it reduces the number of words to compare input against at any given point in a phrase, and (2) it provides for rejection of phrases which are out of syntax even though they may use only words which are included in the overall vocabulary. Further restriction of spoken phrases to those within a list of pre-defined approved phrases (a specified phraseology) will further improve reliability of the recognition system.

Historically, syntax structures have been defined manually using hand-drawn state diagrams, such as that shown in figure 1. Phrases can begin only with words following the starting node. The speech recognition system moves to a successor node along a connecting vector when a word contained in that successor node is recognized. It is a common practice to specify silence nodes (S-nodes) between words to allow for silences between words in a phrase.

If an S-node is allowed to follow itself, then the silence can be of variable duration. Silence nodes are shown for the first few words in the syntax shown in figure 1 to give a flavor for how this works. In an actual system, such silence branches might be included between every node and its successors. Most of them are omitted here in order that we may draw a clearer picture of the syntax relation of the spoken words. After a transition has been made to a node, only nodes which follow that node are then accessible. It is not possible to return to the predecessor (unless it is a follower as well, as is frequently the case with S-nodes). A word may appear in more than one node.

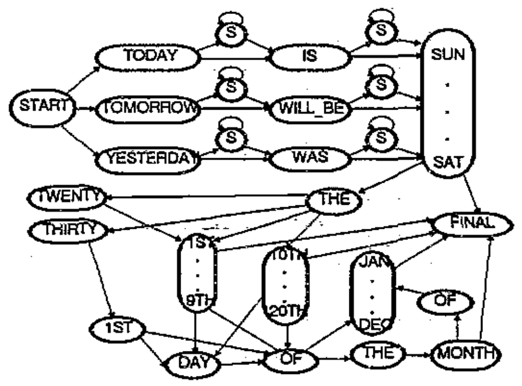

Figure 1. State Diagram

How do we specify a syntax to computational hardware? Historically, syntax state diagrams have been converted manually to ASCII files, such as that depicted in figure 2.

Each node entry in the ASCII file specifies a node ID number, the words contained in that node, and the ID numbers of nodes which may follow that node. As one might expect, the manual processes of drawing such diagrams, encoding them into ASCII files, and debugging and modifying them can become somewhat tedious and at times frustrating –

```
NODE1:
 _sil
(1 final)

NODE3:
 _sil
(3 6 12 16)

NODE6:
 today
(7 8)

NODE8:
 is
(9 10 20)

NODE9:
 _sil
(9 10 20)

NODE10:
 Sunday
 Monday
 Tuesday
 Wednesday
 Thursday
 Friday
 Saturday
(1 20)

NODE12:
 tomorrow
(13 14)

NODE13:
 _sil
(13 14)

NODE14:
 will_be
(10 15 20)

NODE15:
 _sil
(10 15 20)

NODE16:
 yesterday
(17 18)

NODE17:
 _sil
(17 18)

NODE18:
 was
(10 19 20)

NODE19:
 _sil
(10 19 20)

NODE20:
 the
(22 24 26
28)

NODE22:
 twenty
(26)

NODE24:
 thirty
(30)

NODE26:
 first
 second
 third
 fourth
 fifth
 sixth
 seventh
 eighth
 ninth
(1 32 34)

NODE28:
 tenth
 eleventh
 twelfth
 thirteenth
 fourteenth
 fifteenth
 sixteenth
 seventeenth
 eighteenth
 nineteenth
 twentieth
 thirtieth
(1 32 34)

NODE30:
 first
(32 34)

NODE32:
 day
(34)

NODE34:
 of
(36 42)

NODE36:
 the
(38)

NODE38:
 month
(1 40)

NODE40:
 of
(42)

NODE42:
 January
 February
 March
 April
 May
 June
 July
 August
 September
 October
 November
 December
(1)
```

Figure 2. **ASCII Encoding**

especially if there are many phrases, words, nodes, or connections.

The need for tools to automate, or at least help with, the processes of syntax definition and editing will be recognized early on by anyone working in this area. Since the processes are very mechanical in nature, it seems reasonable that it should be possible to automate them. The purpose of this paper is to describe and share some tools which have been developed thus far at NTSC to work with speech recognition systems.

A PARTIAL SOLUTION

A tool has been developed at NTSC which automatically generates a state diagram from an ASCII syntax definition file. It can also check for various hazards in the structure, can check a list of phrases for compliance with the syntax, and can write and check against word lists. This automates much of the clerical tedium of dealing with syntax structures and phrase lists.

It is a relatively simple matter to build a data structure which holds the information given in the ASCII syntax-description file. The tool developed also builds, for each node, a list of predecessor nodes. The predecessor list is useful for syntax debugging. Obviously, if a node has no predecessors, it is unreachable, and may therefore be eliminated from the structure.

After the data structure has been built, we have the problem of how to display it. It is desirable to have as few as possible crossing vectors. It is also desirable for defined phrases to proceed more-or-less from left to right. Processing is simplified if all nodes can be treated equally, using some sort of recursive procedure to define locations of each node and its followers and their followers, etc. S-nodes are special in that they usually have only one predecessor and one follower in addition to themselves. After coordinates are assigned for each node, locations of nodes are compared against one another. If there is a match between any two nodes, then the location of one is offset by some small amount.

A sample state diagram, drawn in EGA graphics using the DIAGRAM tool, is shown in figure 3. State diagram manipulation functions implemented thus far include pan and zoom. Each node is drawn as a circle with its number. Other functions to be added include (1) listing of words contained in a selected node, (2) graphical editing capability with capturing of the syntax from the graphic and automatic writing of the ASCII file, and (3) real-time highlighting of the path followed through the network as a spoken phrase is entered.

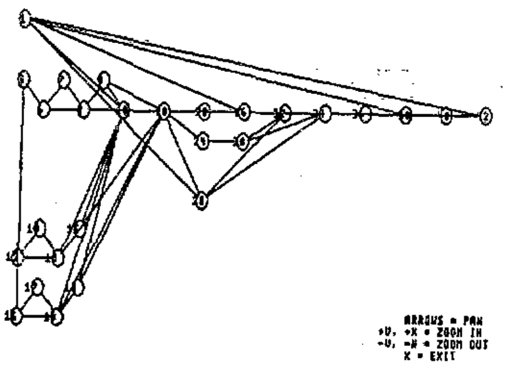

Figure 3. EGA State Diagram

OTHER TOOLS

Other tools developed in addition to the state-diagram drawer deal with checking for various errors or hazards in the syntax. Functionality implemented thus far includes (1) writing a dictionary file listing separately all unique words used in the syntax, (2) checking the words specified in nodes of an ASCII syntax file against an existing dictionary, with or without upper/lower case distinction, (3) checking of a list of phrases against a dictionary or a syntax, (4) listing of node data, including word, *successor, and predecessor lists, and* compilation of statistics and (5) checking for hazards or probable errors in the syntax.

*Syntax errors/hazards checked for thus far in*clude (1) multiple S-nodes in parallel or series, (2) multiple parallel paths from any one source node to the same follower word in different follower nodes, (3) unreachable nodes, (4) unleavable nodes. The first hazard is dangerous because, generally once the speech recognition system has decided on a transition to a node, it is committed; there is no way back. Consider the syntax fragment depicted in figure 4(a). Node 1, containing the word "Today_is", is followed only by the two S-nodes, 2 and 3. Both nodes contain the same silence "word". The transition to be taken is non-deterministic. Somehow, the recognition system must arbitrate which node to go to. If a transition is made to node 2, then "Sunday" is the only acceptable word which may follow. If "Monday" is received next, the phrase will be rejected as out of syntax because "Monday" is not an allowed follower of node 2. One correct specification of a syntax containing both the phrases "Today_is Sunday" and "Today_is Monday" is shown in figure 4(b). The transition from "Today_is" may go directly to node 6 or 7 *with the recognition of either "Sunday" or* "Monday" or it may go via a silence of any duration, as indicated in node 2. Generally, the followers of an S-node should be the same as those of its predecessor.

The hazard of multiple S-nodes in series is exacerbated if the first one is allowed to follow itself (Figure 5). Unpredictable results may also stem from a syntax in which a following word is contained in more than one following node to a given source node (Figure 6). The situation can be complicated by the presence of one or more S-nodes along the way. Again, we have a non-deterministic system similar to that shown in figure 4(a). Since both "is"'s are the same word, it is impossible to predict which route the recognizer will take when that word is spoken. After it has taken that route, phrases on the other route are no longer acceptable. The problem may be further com*plicated by the presence of S-nodes between* actual words or by looping in the syntax.

The tool developed at NTSC checks for mul*tiple routes to the same word in different nodes.*

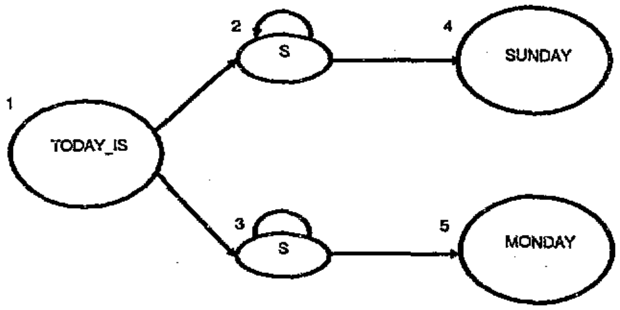

Figure 4 (a). Parallel S-Nodes

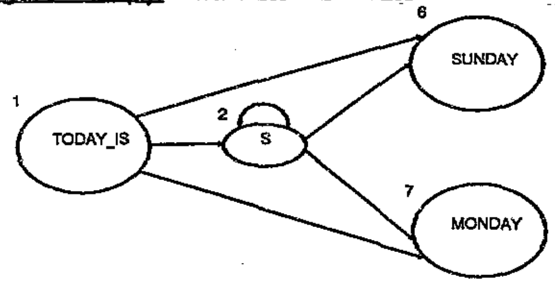

Figure 4 (b). A Better Solution

At the same time, it tallies fanout statistics. It is felt that fanout is one of the metrics by which the complexity and processing delays of a syntax might be measured.

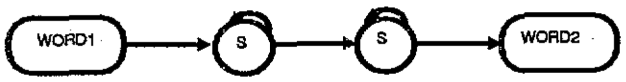

Figure 5. Serial S-Nodes

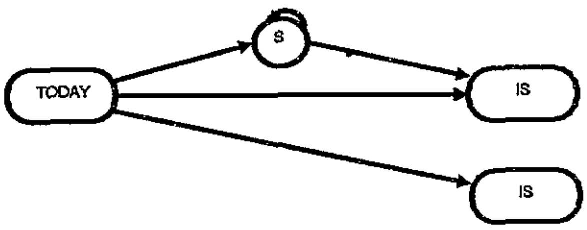

Figure 6. Parallel Paths

OTHER UTILITIES

When out-of-phraseology speech (OOPS) does occur (usually human error) a real-time/run-time utility can be added to the speech processing system (post-processor) to probabilisticly detect these word and command errors and prompt the user to repeat the phrase.

During development of the Naval Training System Center's speech driven (speaker dependent) Under Ice Navigational Training system, an algorithm was derived and a utility was written to detect and measure these OOPS errors in both the Helm and Sonar Operator phraseologies. The utility is a real-time post-processor written in the 'C' programming language. The processor monitors the speech board's output and 'traps' the probable OOPS errors. The result is a generated speech output asking the user to repeat his last command ('Say Again'). This 'trap' is a programmable 'confidence level' of the input speech. The algorithm utilizes individual 'word scores' along with 'rejection template scores' to determine word pass or fail criteria. Word scores, provided by the speech board, are Euclidean distance (pre-trained templates vs real-time speech) numbers representing the quality or goodness-of-fit of the speech for that particular word. Similarly, rejection template scores are distance scores based on real-time speech compared to rejection templates (rejection templates are small samples of dependent speech concatenated together for that period of speech). The utility compares the above mentioned scores and determines, based on a pre-set value, if the word passes or fails. A percentage of word failures is then taken for each speech command, and a command threshold test is made to determine command recognition acceptance. If a command fails, it is not sent to the simulation dynamics processor for further processing. Instead, a 'say again' message is sent to the speech output routine. This eliminates most 'forced recognition' events that could occur in the simulation. Results extracted from this utility are given in figure 7. It is noted from these results that on average 88 percent of major OOPS are detected and handled properly.

Raw Recog Cmd. Rate%	Corrected (PTT/ Clipping) Rate%	Minor Phrase Reject Rate%	Major Phrase Reject Rate%	Incorrect Reject Rate%
100	100	28	84	3
94	97	40	91	11
92	99	36	89	3
90	99	52	94	11
89	97	45	93	17
87	100	36	80	3
83	96	18	83	0
99	100	39	88	3
82	99	30	79	11
97	97	42	91	10
96	99	32	90	5
94	96	55	94	10
Avg. Rates %				
92	98	38	88	7

Figure 7. OOPS Rejection Test Results

SUMMARY

With the speech tools and utilities described above, the Under Ice Navigational Trainer speech sub-system was integrated with very few 'bugs'. This in turn made the implementation/integration of the voice system very quick. These speech tools and utilities provided the design team with a mechanism to detect early speech-related problems. The problems were then corrected before integration into the simulation system.

The real-time command-scoring utility has shown very good preliminary results for both the Helm and Sonar Operator phraseologies. Major OOPS have been rejected on average 88 percent of the time, while false rejection only occurred on average 7 percent of the time (based on 100 percent command accuracy). Without the use of this utility numerous false or forced commands might have been processed by the simulators dynamics processor (resulting in incorrect submarine action for a given command).

FURTHER WORK TO BE DONE

Further enhancements, which allow graphical editing of the state diagram, and subsequent automatic generation of the descriptive ASCII file are planned. Work is also continuing in speech post-processing technology with further work in scenario restriction post-processing.