Developing Minimal Comprehensive Sets of Test Cases for Natural Command Grammars

TR 29.3272

January 21, 2000

James R. Lewis

Speech Development and Customization

West Palm Beach, Florida

Abstract

Encoded command grammars are common in speech-enabled systems. Simple grammars are relatively easy to develop and test, but more complex grammars are very difficult to develop and test comprehensively. For natural command grammars (finite state grammars that allow natural-language-like flexibility in the number of allowable expressions per function) the number of valid commands is essentially infinite, precluding exhaustive testing. This report describes a procedure for defining the smallest possible set of test cases that can comprehensively test the grammar with regard to verifying correct output to a translation rule interpreter. This set is also appropriate for use in accuracy testing.

ITIRC Keywords

BNF grammar
Natural command grammar
Minimal test set
Accuracy assessment

About the Author

James R. Lewis has worked as a human factors engineer for IBM since 1981, with experience in both hardware and software human factors. He received his MA in Experimental Psychology (Engineering Psychology) from New Mexico State University in 1982, and received his Ph.D. in Experimental Psychology (Psycholinguistics) from Florida Atlantic University in 1996. He has published over 150 technical papers (both within and external to IBM) and currently holds an 11th plateau in the IBM Patent Program.

Contents

Introduction

Background

In the rapidly emerging field of speech-enabled systems, encoded command grammars are becoming common. While simple grammars are relatively easy to develop and test, more complex grammars (including natural language and natural command grammars) are very difficult to develop and test comprehensively. For natural command grammars (finite state grammars that allow natural-language-like flexibility in the number of allowable expressions per function) the number of valid commands is for all practical purposes infinite, precluding exhaustive testing. Thus, there is a need for a technique that produces the smallest possible set of test cases that can comprehensively test the grammar with regard to verifying correct output to translation rule interpreter or other parsing system.

A grammar contains a representation of the valid expressions for a speech-enabled computer program. The next section contains an example of a natural command-style grammar using BNF notation. Note that the grammar has (1) comments, (2) a root node branching out to sentence set nodes, (3) sentence set nodes that branch out to sentences (natural commands), (4) variables in sentences that connect to phrase nodes, and (5) a translation rule at the end of each line of the grammar. Although there might be a number of approaches to structuring a natural command grammar, this is one that has been used in practice (for example, IBM's ViaVoice[1] '98), and seems the most logical. Note that there is no limit to the extent to which the designer of a grammar can use variables in one node that point to a lower node (see the subphrase nodes in the sample grammar). At some point, though, all variables in sentences will connect to a node with no variables (called a terminal node).

Sample Grammar

```
//This is a comment.

<<root>> = <go_sentence>            -> {1}
| <move_sentence>                   -> {1}
| <copy_sentence>                   -> {1}
.

//go(where)
<go_sentence> = <phrase1>           -> go({1})
| <phrase2> NOW                     -> go({1})
| <phrase3> <phrase4>               -> go({2})
.

//move(what,where)
<move_sentence> = MOVE THIS TO TOP  -> move(selection,top)
| RELOCATE THIS TO BOTTOM           -> move(selection,bottom)
| PUT <phrase5>                     -> move({2})
| PUT <phrase6>                     -> move({2})
.
```

[1] ViaVoice is a trademark or registered trademark of International Business Machines Corporation.

```
//copy(what,where)
<copy_sentence> = COPY <phrase6>     -> copy({2})
| COPY <phrase5>                     -> copy({2})
.

<phrase1> = GO TO THE TOP            -> topfile
| GO TO THE BOTTOM                   -> bottomfile
| <subphrase1>                       -> {1}
| GO TO THE <subphrase2>             -> {4}
| <subphrase1> NOW                   -> {1}
.

//This is another comment.

<phrase2> = END OF FILE              -> bottomfile
.

<phrase3> = JUMP TO THE              -> null
.

<phrase4> = TOP OF THE DOCUMENT      -> topfile
| BOTTOM OF THE DOCUMENT             -> bottomfile
| BEGINNING OF THE LINE              -> begline
| END OF THE LINE                    -> endline
.

<phrase5> = THE PREVIOUS WORD TO THE TOP  -> previousword,top
THE PREVIOUS WORD TO THE BOTTOM           -> previousword,bottom
.

<phrase6> = THE PREVIOUS WORD TO THE END OF THE LINE
      -> previousword,endline
THE PREVIOUS WORD TO THE BEGINNING OF THE LINE
      -> previousword,begline
.

<subphrase1> = END OF LINE           -> endline
.

<subphrase2> = BEGINNING OF LINE     -> begline
| START OF LINE                      -> begline
.
```

In the given example, the root points to three sentence sets, defined below the root. Each sentence contains (1) the valid phrase for the grammar and (2) a translation rule (following the `->`) that translates the natural command into a functional statement. Note that translation rules are a convenience in a grammar for getting the spoken phrase into a form that back-end programs can use to execute their functions without requiring a separate parsing program, but are not the only way to accomplish this. Two other approaches are the use of annotations (which make parsing routines easier to write) and the use of parsing routines that work without annotations. The precise method doesn't matter, but BNF code with translation is the easiest type of grammar to work with.

Method

Finding the Test Set in a Grammar with Translation Rules

A grammar with translation rules provides a clear indication of where the sentence sets are -- this is the level in the grammar that has translation rules with the form:

FunctionName(parameter 1, parameter 2, . . . , parameter n)

The sentence sets connect directly to the root node. All valid phrases in the grammar correspond to a sentence in a sentence set, and when a speaker utters a valid phrase, one consequence of this is the transmission of the translation rule (filled with the parameters indicated by the content of the phrase) to a translation rule interpreter. Once interpreted, the system performs the actions as defined by the translation rule's function.

Logical analysis makes it clear that all translation rules will be exercised by a set of sentences that number the same as the number of lines (excluding comments) in the grammar. The logic is that there should be a test sentence for each translation rule in the grammar. In the example above, this set would contain 29 sentences. However, the act of testing translation rules at the sentence-set level (and seeing the system produce a correct translation rule) eliminates the need for a specific test case for the translation rules at the root node level -- at least, no need to test translation rules that have the form {1}. For the example above, this reduces the required number of test cases to 26.

Because the act of testing subsentence translation rules necessarily involves creating a test sentence, it is possible to further reduce the required number of test sentences. The amount of this reduction depends on the structure of the specific grammar under test. At one extreme, if it were possible to create test sentences for subsentence translation rules using all of the sentence-level translation rules, then the number of required sentences would be reduced by the number of sentence-level translation rules. At the other extreme, if it were only possible to test all subsentence translation rules using a single sentence-level translation rule, then the number of required sentences would be reduced by one.

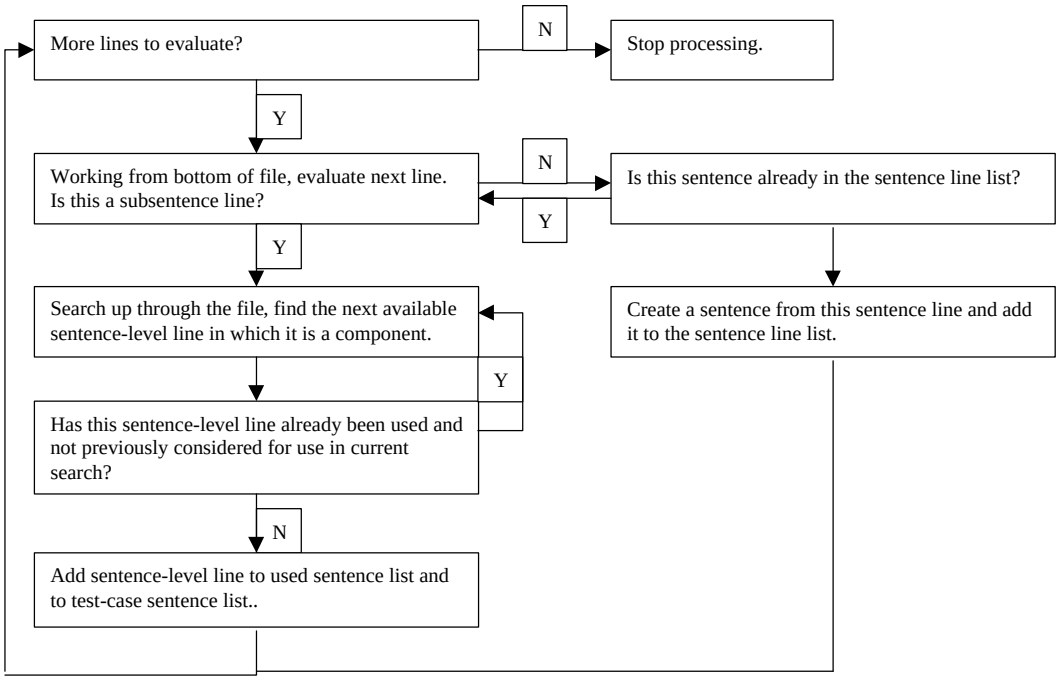
Creating a test sentence for each translation rule in the sample grammar produces the set of test sentences shown in Table 1 (note that the sample grammar expands to 21 valid sentences), with duplicate sentences indicated. It is easiest to produce the test sentences by starting with the last node in the grammar and working up because to move up through the grammar, it is only necessary to check the variable name of the node, then finding it referenced higher in the grammar, checking the variable name of that node, finding it referenced higher in the grammar, and continuing until hitting a node that produces a sentence-level translation rule. As shown in the table, it takes 18 sentences to test the sample grammar.

Table 1. Sample set of test sentences

Tested Structure	**Test Sentence**	**Function Produced**	**Duplicate if X**
START OF LINE	GO TO THE START OF LINE	go(begline)	
BEGINNING OF LINE	GO TO THE BEGINNING OF LINE	go(begline)	
END OF LINE	GO TO THE END OF LINE	go(endline)	
THE PREVIOUS WORD TO THE BEGINNING OF THE LINE	COPY THE PREVIOUS WORD TO THE BEGINNING OF THE LINE	copy(previousword,begline)	
THE PREVIOUS WORD TO THE END OF THE LINE	PUT THE PREVIOUS WORD TO THE END OF THE LINE	move(previousword,endline)	
THE PREVIOUS WORD TO THE BOTTOM	COPY THE PREVIOUS WORD TO THE BOTTOM	copy(previousword, bottomfile)	
THE PREVIOUS WORD TO THE TOP	PUT THE PREVIOUS WORD TO THE TOP	move(previousword,topfile)	
END OF THE LINE	JUMP TO THE END OF THE LINE	go(endline)	
BEGINNING OF THE LINE	JUMP TO THE BEGINNING OF THE LINE	go(begline)	
BOTTOM OF THE DOCUMENT	JUMP TO THE BOTTOM OF THE DOCUMENT	go(bottomfile)	
TOP OF THE DOCUMENT	JUMP TO THE TOP OF THE DOCUMENT	go(topfile)	
JUMP TO THE	JUMP TO THE TOP OF THE DOCUMENT	go(topfile)	X
END OF FILE	END OF FILE NOW	go(bottomfile)	
<subphrase1> NOW	END OF LINE NOW	go(endline)	
GO TO THE <subphrase2>	GO TO THE START OF LINE	go(begline)	X
<subphrase1>	END OF LINE	go(endline)	
GO TO THE BOTTOM	GO TO THE BOTTOM	go(bottomfile)	
GO TO THE TOP	GO TO THE TOP	go(topfile)	

COPY <phrase5>	COPY THE PREVIOUS WORD TO THE BOTTOM	copy(previousword,bottom)	X
COPY <phrase6>	COPY THE PREVIOUS WORD TO THE BEGINNING OF THE LINE	copy(previousword,begline)	X
PUT <phrase5>	PUT THE PREVIOUS WORD TO THE TOP	move(previousword,top)	X
PUT <phrase6>	PUT THE PREVIOUS WORD TO THE END OF THE LINE	move(previousword,endline)	X
RELOCATE THIS TO BOTTOM	RELOCATE THIS TO BOTTOM	move(selection,bottom)	
MOVE THIS TO TOP	MOVE THIS TO TOP	move(selection,top)	
<phrase3> <phrase4>	JUMP TO THE TOP OF THE DOCUMENT	go(topfile)	X
<phrase2> NOW	END OF FILE NOW	go(bottomfile)	X
<phrase1>	GO TO THE TOP	go(topfile)	X
<copy_sentence>	COPY THE PREVIOUS WORD TO THE BOTTOM	copy(previousword,bottom)	X
<move_sentence>	MOVE THIS TO TOP	move(selection,top)	X
<go_sentence>	GO TO THE TOP	go(topfile)	X
		Total unduplicated sentences:	**18**

So, to create this minimum test sentence set, the strategy for sentence selection differs depending on whether the line for which the test case is being created is a subsentence- or sentence-level line. For subsentence-level lines, the appropriate strategy is to keep track of which sentence-level lines were used in those test cases and, as new test cases are developed, seek to use unused sentences. For sentence-level lines, the appropriate strategy is to develop new test cases only for those lines that were not used in subsentence-level test cases. The following flow chart shows (at a high level) the steps required to implement these strategies:



Finding the Test Set in a Grammar without Translation Rules

Translation rules provide a convenient way to discriminate between subsentence- and sentence-level lines in a natural command grammar. If the grammar doesn't contain translation rules, then there are a couple of approaches that could be used to make this identification. One approach would be to annotate the lines of the grammar, using SS for subsentence and S for sentence level. Another approach would be to present the grammar to the user (the developer in this case) and let him or her click on the sentence level lines. Once this identification has been made, the flowchart above would work.

Discussion

It is impossible (or at least very difficult and inefficient) to test every possible phrase in a natural command grammar to verify correct interpretation by a translation rule interpreter (or other parsing technique). The technique described in this report produces the smallest possible set of test cases that can comprehensively test this aspect of a natural command grammar (the minimal test set). This, in turn, reduces the effort required to test a natural command grammar in this respect, leading to improved efficiency in development.

In addition to providing a test set appropriate for verifying the accuracy of translation rules, the minimal test set provides an appropriate and efficient set for testing the recognition accuracy for a natural command grammar. Even though the number of legal utterances for such a grammar can be essentially infinite, the space of legal utterances is not unbounded. The grammar binds this space, with the boundaries defined by the intersection of the sentence and subsentence lines of the grammar. The use of the minimal test set for recognition accuracy testing ensures that every word in the grammar will appear at least once. The test set will contain phrases representative of the lengths of allowable phrases, both short and long. The number of phrases in the minimal test set provides an answer to the question of how many phrases to test to exercise a grammar. If this number is, however, less than 100, it is easy to add additional test sentences to improve the precision of measurement. For most minimal test sets derived from a commercially developed natural commands grammar, the number of sentences in the test set should easily exceed 100.

When a grammar contains large lists (such as thousands of names in a telephone directory or thousands of stock names in a stock quotation application), even the minimal test set may prove unwieldy for testing purposes. One strategy for dealing with this is to develop the minimal test set for recognition accuracy without attempting to cover all members of a long list. To test recognition accuracy for the long list, order the members of the list according to some reasonable external criterion (such as frequency of use), and to conduct a separate accuracy test for the most important members of the list. If no reasonable external criterion exists, then use random selection to select a predefined percentage of the members of the list.