A More Thorough Syntax Checker for FORTH

R. D. Dixon
David Hemmendinger
Department of Computer Science
Wright State University
Dayton, Ohio 45435

ABSTRACT

A sample grammar for FORTH is developed using ideas from categorial grammars which are used in natural language processing. This grammar is in the spirit of FORTH compilers but in addition a simulation of certain runtime activities is conducted to attempt a verification of correctness of parameters.

INTRODUCTION

Syntax diagrams for FORTH were presented at the 1982 Rochester FORTH Conference by K. Moerman [1]. Syntax diagrams and BNF grammars are useful because they give a short specification of what syntax is allowable in the language. Many compilers are derived directly from the BNF grammar.

Any purely syntactical specification of a programming language will accept or generate some strings or programs which are not meaningful. Normally, the semantics of the language limits what is a valid program just as syntax does. FORTH, because it uses a parameter stack to pass arguments, has a very loose syntax. Arguments may be generated a very long way, syntactically, from their use. Thus, the standard FORTH compiler has no way at compile-time to know if the proper arguments for a function are available when it is called. Even if we assume that integer is the only data type, FORTH compilers still cannot determine the number of input and output arguments necessary for the proper operation of a colon definition.

A second deficiency with a BNF specification for FORTH is that compiler implementations derived from BNF grammars usually contain the specification of the language syntax in their structure or in a processed table. A FORTH compiler has most of its syntactic information distributed in the dictionary. It is this distributed nature of the compiler that permits the extensibility of the FORTH language.

Categorial grammars [2] are a much older method of syntax specification than the phrase structure grammars [3] from which BNFs are derived. In categorial grammars each word has one or more categories to which it belongs. The category may be primitive (indivisible) or complex. Complex categories relate the word to the categories of adjacent words. Just as with phrase structure grammars, different category schemes yield different classes of languages. A simple category scheme leads to the class of context-free languages, the class into which most programming languages fall [4].

We propose a category scheme and a parser here that is

similar to the technique used in FORTH compilers. The categories may be used to generate either a top-down or a bottom-up parse. The combination rule described here generates a top-down pushdown automaton parser.

SYNTACTIC CATEGORIES

A primitive category is a name starting with a lower case letter. Names starting with upper case letters are PROLOG type variables which stand for any category. A complex category is an expression of the form X/Y where X and Y are categories. If necessary, parentheses are used to remove ambiguity. Thus a, a/b, and (a/b)/c are categories.

A dictionary of the categories of some FORTH words might be stated as follows with each category being given as an attribute of the word:

```
Word(sym)
:((s/sc)/sym)   ;(sc)          +(X/X)          drop(X/X)
dup(X/X)        rot(X/X)       '((X/X)/sym)    if((X/X)/t)
else(t/t1)      then(t)        then(t1)        Int(X/X)
```

The Word(sym) means every word has as its first category sym which stands for "symbol". Int(X/X) means every integer has category X/X. s stands for "sentence", sc for "semicolon" and t and t1 for "then" categories.

The compilation starts with the category s on a stack, processes words from left to right and uses the following combination rule:

The current word considers its categories in their given order. It compares its leftmost symbol with the topmost symbol on the stack. If the symbols match or can be made to do so by instantiating a variable then that symbol is deleted from the stack and the remaining symbols from this category are placed in order on the stack. Otherwise that category fails. If all the categories for a word fail, the word fails. Backtracking will generate all possible parses.

Consider the example:

```
definition      :          dog       dup       +         ;
category     s/sc/sym      sym       X/X      X/X       sc
stack
  0      s
  1              sc,sym
  2                        sc
  3                                sc (X=sc)
  4                                        sc (X=sc)
  5                                                    —
```

The parse succeeds if the stack is empty and the input string is consumed.

The if-else- then structure parsed as follows:

```
definition     :      boy     if     +     else    drop     then    ;
category   s/sc/sym   sym     X/X/t X/X   t/tl    X/X       tl    sc
stack
  0        s
  1                sc,sym
  2                       sc
  3                           sc,t
  4                                  sc,t
  5                                        sc,tl
  6                                               sc,tl
  7                                                      sc
  8
```

Notice that category t for "then" fails in this sentence.

SYNTACTIC AND SEMANTIC CATEGORIES

In order to go beyond this context free scheme we augument the categories with a description of the run time stack, that is the semantic action of the word. Let n be a name that represents an integer. The new category of + is

+(X/X,[n,n]=>[n]).

Here the stack pictures before and after follow the comment conventions used by many FORTH programmers. The following list gives the augmented categories for some FORTH words.

```
Word(sym,[]=>[])
Word1(t01/t01,[]=>[]))      Word1(t11/t11,[]=>[])
:(s/sc/sym,[]=>[])          ;(sc,[]=>[])
+(X/X,[n,n]=>[n])           drop(X/X,[n]=>[])
dup(X/X,[n]=>[n,n])         rot(X/X,[n,n,n]=>[n,n,n])
'(t01/t01/sym,[]=>[])       '(t11/t11/sym,[]=>[])
'(X/X,[]=>[n])               Int(X/X,[]=>[n])
if(t01/t01/t01,[]=>[])
if(t11/t11/t11,[]=>[])
if(X/X/t00,[n]=>[])
if(X/X/t11,[n]=>[])
else((t01,t01)/(t01,t01)),[]=>[])
else((t11,t11)/(t11,t11)),[]=>[])
else(t00/t01,[]=>[])
else(t11/t10,[]=>[])
else(t10/err,[]=>[])
then(t00,[]=>[])
then(t01,[]=>[])
then(t10,[]=>[])
then(t11,[]=>[])
```

These categories need further explanation. In the first three lines we indicate some general statements about many words. First all words used as symbols have no semantic action. Word1 is used here to indicate all words except if,else, then. The two assertions concerning Word1 mean that all semantic actions are suspended when t01 or t11 is on top

of the stack. These two primitive categories are used to indicated we are parsing one of the two alternatives in an if clause and we are assuming it is not the selected alternative.

The categories for if, else, then are the most interesting. In this grammar a simple if-else-then statement will have two correct parses. The first includes the semantic actions of the if-else clause but not the else-then clause, and the second includes only the else-then. The t categories are coded with two digits. The first digit is 0 for the first pass and 1 for the second. The second digit is 0 for the the selected alternative and 1 for the neglected alternative.

In addition to the categories shown here it is convienent to place extra conditions on the assignments of variables and to also limit backtracking. This is done by adding a guard to each category statement.

This parser has been implemented in PROLOG and that, no doubt, affects the presentation. The goal is to simplify the approach to the point that it can be implemented in low level FORTH.

The authors are indebted to Merrie Bergmann whose work with category grammars and English served as a model for this approach.

REFERENCES

1. Moerman, K. FORTH Syntax Diagrams. Proceedings of the Rochester FORTH Conference 1982,263-266
2. Adjukiewicz, Kazimierz. Uber die Syntaktische Konnexitaet. Studia Philosophica 1, 1935, 1-27
3. Chomsky, Noam. Three Models for the Description of Language. IRE Transactions on Information Theory IT-2, 1956, 113-124
4. Bar-Hillel, Yehoshua, Chaim Gaifman, and Eliyahu Shamir. On Categorial and Phrase Structure Grammars. The Bulletin of the Research Council of Israel 9f, 1960, 1-16