OUR HERITAGE
ISSN: 0474-9030 Vol-68, Special Issue-27 (Feb. 2020)
5th International Conference On "Innovations in IT and Management"
Organised by: Sinhgad Technical Education Society's
SINHGAD INSTITUTE OF MANAGEMENT AND COMPUTER APPLICATION (SIMCA),
Narhe Technical Campus, Pune, Maharashtra (India) 411041.
Held on 6th & 7th February 2020

Semantic Analyser for C-Minus Language

Vaishali P. Bhosale

Assistant Director/ Assistant Professor
Yashwantrao Chavan School of Rural Development, Shivaji University, Kolhapur

Abstract

The design and development of Tiny compiler [3] forms the basis of this study. From the specifications for C-Minus language such as tokens and grammar rules [3] a scanner [12] and recursive descent parser [13] are designed and developed for C-Minus language. This paper begins with the basics of semantic analysis. It is the analysis carried out by the compiler prior to execution. The semantic analysis includes semantic rule definitions, symbol table management and type checking tasks for C-Minus language. The symbol table management keeps track of meaningfulness of the program in semantic analyser module. The correctness of expressions and statements within the type rules of the language is performed by type inference and type checking tasks.

Keywords: *Semantic Analyser, Tiny language, C-Minus Language, Symbol table, Type inference, Type checking.*

I. Introduction:

The concept of finite automaton and its application in compiler design is well known. Louden [3] has explained implementation of the compiler in detail along with a case study of the Tiny language and specification for the C-Minus language. Compilation is the analysis of source language program and synthesis of target language program. Analysis broadly categorised as lexical analysis which separates the tokens of the program, syntax analysis which checks the correctness of syntax (grammar rules) and semantic analysis is to determine the information that is beyond the capabilities of the context free grammars and checks for the meaningfulness of the program. The synthesis phase includes intermediate code generation, code optimization and target language program as final task. Along with basic concepts from theory of automaton, implementation of compiler also needs major data structures. The tokens in a language are represented using enumerated data types. The syntax analyser is implemented using pointer based tree data structure. Frequent access to symbol table management in compiler is done using hash table data structure. Linked list of structures used to keep Intermediate code. Temporary files can be used to store the results of intermediate steps of compiler.

ISSN: 0474-9030 Vol-68, Special Issue-27 (Feb. 2020)
5th International Conference On "Innovations in IT and Management"
Organised by: Sinhgad Technical Education Society's
SINHGAD INSTITUTE OF MANAGEMENT AND COMPUTER APPLICATION (SIMCA),
Narhe Technical Campus, Pune, Maharashtra (India) 411041.
Held on 6th & 7th February 2020

II. Basics of Semantic Analysis

The Tiny compiler [3] studied and used as base for design of scanner [12] and parser [13] for C-Minus language [3]. The lexical structure for C-Minus language is given below [12]:

Reserved words	Special symbols	Other
if	+ - * /	Numbers
else	< <= > >= == !=	(i.e.NUM)
int	=	
return	( ) [ ] { }	Identifiers
while	/* */	(i.e.ID)
void	; ,	

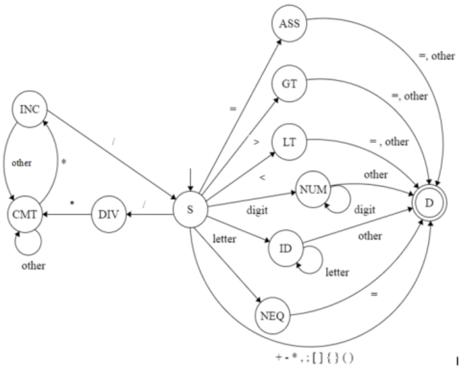


Figure 1: DFA for C-Minus Tokens [12]

The minimal state DFA given above is encoded in C language. By applying left factoring and elimination of left recursion methods the C-Minus grammar is made suitable to implement it as a recursive descent parser [13]. Recursive descent parser for C-Minus language forms the basis for the C-Minus Language. Syntax tree is the result of parsing phase.

Scanning and parsing phase has limitation. Consider the declaration

Int num= "string"

The scanner separates the token and parser creates a parse tree from the tokens but the error is generated by the semantic analyser as the string is assigned to an integer. The semantic analyzer is checks for following errors:

-Data Type mismatch

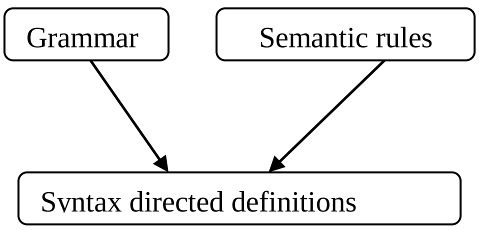
- Variables which are not declared

- Misuse of keywords

- Declaration of variable more than once

- Variable accessed out of scope etc.

These attributes or properties of programming language are specified using attribute equations also known as semantic rules. These rules are specified using syntax directed definitions. Thus



III. Semantics in C-Minus Language:

The static semantic requirements of C-Minus language are listed below with implementation details:

1.All variables and functions must be declared before they are used.

2. There must be at least one declaration.

3. Function declaration can have any of the void or int type specifier.

4.In a variable declaration only the type specifier int can be used.

5. If the return type of the function declaration is void , then function returns no value. This can be simulated as:

```
procedure typeval (n: node)

begin

    case stmtkind of
```

```
    decl:
        case declkind of
            variable:
              if n.dtype= void then
      print error {variable cannot be void}
            function:
              if n.dtype= void then
             temp=n→child[1]→child[1]            while temp is not null
 if temp→nodekind is stmt and stmtkind is return then
print error {void function cannot return value}
else
        temp=temp→sibling
        end if
     end if
end
```

6. Last declaration in a program must be a function declaration of the form void main ( ).

This can be simulated as:

```
procedure nodecheck ( n: node)
begin
        node temp;
        if n is root node then
                temp=n;
        while temp→sibling is not null
```

```
            temp=temp→sibling
    end while
    if temp→ attr.name is not "main" then
print error {main must be the last declaration}
     end if
  end if
end
```

7. Only one variable can be declared per declaration.

8. Simple integer parameters are passed by value. Array parameters are passed by reference i. e. as pointers.

9. Functions may be recursive.

10. The local declarations have scope equal to the statement list of the compound statement and supersede.

11. Both declarations and statement list in function declaration can be empty.

12. The while statement is the only iteration statement in C-Minus, it is executed by repeatedly evaluating the expression and then executing the statement if the expression evaluates to a non-zero value, ending when the expression evaluates to zero.

13. Functions not declared void must return value.

14. A return causes transfer of control back to the caller or termination of the program if it is inside main.

15. Upper bounds of array subscripts are not checked.

16. Functions must be declared before they are called and the numbers of parameters in a declaration must be equal to the number of arguments in a call.

17. Input function has no parameters and returns an integer value from the standard input device as keyboard.

18. The output function takes one integer parameter whose value it prints to the standard output as monitor.

IV. Semantic analyzer Implementation:

The syntax tree traversals carries out symbol table management and type checking. Pre-order traversal of the syntax tree builds symbol table. The function to build symbol table is

void symboltbl(node*);

While traversing, the variable identifiers not yet seen and inserted into the symbol table are inserted to the table using the procedure

In_symtbl(n->var.name, n->lineno, loc++)

The procedure inserts an identifier together with its line number and memory location into the symbol table. If the variable is already in the symbol table call is made as below

In_symtbl(n->var.name, n->lineno, 0)

Symtbl_lookup returns value -1 if a variable is not present or inserted earlier in the symbol table. After the traversal is complete, it displays the stored information using function

Display_symtab(FILE *)

The post order traversal of the syntax tree performs type checking. It inserts data types into the tree nodes as they are computed and reports type checking error if any. The function for type checking is

void checktype (node*);

The symbol table is implemented as separately chained hash table to avoid collision. It is same for Tiny hash table [3].

```
# define TBL_SIZE ....

# define MOVE   4

Int hash_function (char *ele)

{

   Int place=0;

   Int i=0;

   While(element [i]! ='\0')

   {

place=((place<<MOVE)+ele[i])%TBL_SIZE;
```

```
i=i+1;
    }
Return place;
}
```

Hash function converts key into an integer value which is index of the bucket where the key will be inserted or searched for. The bucket lists record for each variable includes name, memory location, and the line numbers list in which it appears in the source code. It is same for Tiny language [3] as below:

```
typedef struct BktLstRec
  { char * name;
    Lineno lines;
    int memloc ;
    struct BktLstRec * next;
  } * BktList;
```

The line numbers list in which a variable is referenced is implemented using following structure [3],

```
 typedef struct LnLstRec
  { int lineno;
    struct LnLstRec * next;
  } * Lineno;
```

Tthe hash table is declared with predefined size as below:

```
static BktLst hashTable[SIZE];
```

A variable or function name is a string. Hash function converts each character in the string to a non-negative integer using ASCII value. These integers combined to form a single integer which is mapped to hash table size using modulo division method.

A sample program to C-Minus compiler is given and semantic analyser developed is executed.

```
int hcf (int a, int b)
{
        if(b==0) return a;
        else return hcf (b, a - a/b *b);
}
void main(void)
{
        int u; int v;
        hcf (u, v);
}
```

The output after semantic analyser execution is given below:

```
Symbol table:
name    location        line numbers
------------------------------------------------------
main    3               6
a       1               1   3   4    4
b       2               1   3   4    4    4
u       4               8   9
v       5               8   9
hcf     0               1   4   9

Checking Types.....
```

Type Checking Finished

V. Conclusion:

The paper explains implementation details of semantic analysis. It is possible by optimal use of data structures and concepts from theory of computer science. Symbol table being inherited attribute requires pre-order traversal of the syntax tree. The data type of an expression is a synthesized attribute hence type checking requires post-order traversal of the syntax tree. The paper along with [12] and [13] can be used as a baseline for front end implementation of compiler for simple language.

References:

[1] Hopcroft, Motwani, Ullman, Introduction to Automata Theory, Languages and Computation, Pearson Education.
[2] Aho, Sethi, Ullman Compilers – Principles, Techniques, and Tools, Pearson Education, 2005.
[3] Kenneth C. Louden, Compiler Construction Principles and Practice, Cengage Learning, India Edition, 2008
[4] Asveld P.R.J., A fuzzy approach to erroneous inputs in context-free language recognition, In proceeding of the 4th International Workshop on Parsing Technologies, Prague/Karlovy Vary, Czech Republic. pp. 14-25, 1995.
[5] Asveld P. R. J., A bibliography on fuzzy automata, grammars and languages, Bulletin of the European Association for Theoretical Computer Science 58, pp. 187-196,1996.
[6] Asveld P.R.J., Towards robustness in parsing – Fuzzifying context-free language recognition, In: J. Dassow, G. Rozenberg and A. Salomaa (eds.), Developments in Language Theory II – At the Crossroads of Mathematics, Computer Science and Biology, World Scientific, Singapore, pp. 443-453, 1996
[7] Asveld P. R. J., Controlled fuzzy parallel rewriting, In: Gh. Păun, A. Salomaa (Eds.), New Trends in Formal Languages—Control, Cooperation, and Combinatorics, Lecture Notes in Computer Science, Vol. 1218, Springer, Berlin, pp. 49–70. 1997.
[8] Asveld P.R.J., Algebraic aspects of families of fuzzy languages, Theoret. Comput. Sci., 293(1-2), pp. 417–445, 2003.
[9] Asveld P.R.J., Fuzzy context free languages – Part 1, Generalised fuzzy context free grammars, Theoretical Computer Science, 347(1–2), pp. 167–190, 2005
[10] Asveld P.R.J., Fuzzy context free languages – Part 2, Recognition and parsing algorithms, Theoretical Computer Science, 347(1–2), pp.191–213, 2005.
[11] Vaishali Bhosale, Dr. S. R. Chaudhari, Fuzzy Lexical Analyzer: Design and Implementation, International Journal of Computer Applications, 0975-8887, vol.123-No.11, Aug.2015, p.p.1-7
[12] Vaishali Bhosale, Dr. S. R. Chaudhari, C-Minus: Fuzzy Scanner, International Journal of Latest Research in Science and Technology, ISSN: 2278-5299,Vol. 4, Issue 6: p.p. 30-34.
[13] Vaishali Bhosale, Dr. S. R. Chaudhari, Recursive Descent Parser for C-Minus Language, International Journal of Information Systems, Vol VI, Issue 1, June-Dec.2015, ISSN: 2229-5429, p.p. 13-16

[14] Wei X., Zhang Q., Approach of text search based on semantic parsing model, In Proceeding of the Fourth International Conference on Fuzzy Systems and Knowledge Discovery, IEEE, 2, pp. 355 – 359, 2007.