

Implementation of Tree and Recursion in Checking C Syntax's using Context Free Grammar

William Sentosa / 13513026
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
e-mail : williamsentosa@students.itb.ac.id

Abstract— Tree is a graph which don't has directions, all of its nodes is connected and it contains no circular path. Tree is generally used in science, including in computer science. In this science, it will work pairs with the recursion. Recursion is a process that defined itself in the similar way. It is playing an important role not only in computer science, but also in language theory. One of the theorem that use this recursion is Context Free Grammar. Context Free Grammar is a formal notation for expressing the recursive definitions of languages. This notation can be used in expressing the syntax of C programming language. The tree will be used to parse the syntax and it can determine whether the syntax is valid.

Keyword—Context Free Grammar, parse tree, recursive, syntax.

I. INTRODUCTION

Tree is defined as the structure of graph which is mostly used in science. Not only used in computer science, but also in language. Tree is undoubtedly play an important role in the language theorem, especially when it has to be processed by the computer. The ability to parse down the sentence into word in a root tree is the key why tree is used in this theory.

Language theory in computer science brought a drastic change in compiler technology, especially after Context Free grammar was found. Since that time, they turned the implementation of parsers (function that discover the structure of program) from a time-consuming, ad hoc-implementation task into a routine job that can be done in an afternoon. More recently, the context-free grammar has been used to describe document formats, via the so-called document-type definition (DTD) that is used in the XML (extensible markup language) community for information exchange on the Web.

Context Free grammar is structured by recursive notations. So, the recursion theorem play a major role here. Recursion is quite well-known in language theory, since the language is also recursive by nature. The word "and", for example, can be construed as a function that can apply to sentence meanings to create new sentences, and likewise for noun phrase meanings, verb phrase meanings, and others. It can also apply to intransitive verbs, transitive verbs, or ditransitive verbs. In order to provide a single denotation for it that is suitably flexible, and is typically defined so that it can take any of these

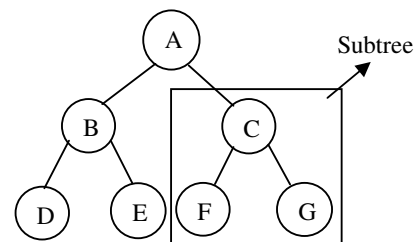
different types of meanings as arguments. This can be done by defining it for a simple case in which it combines sentences, and then defining the other cases recursively in terms of the simple one.

In this paper, we will see if tree and recursion can be useful to check the syntax of the C programming language. C language is a well-known programming language. It can be proven by seeing facts that C compiler is available in almost available computer architectures and operating systems. Not only that, C is also used in many competitive programming contest and was used in lots programming class.

II. RELATED THEORIES

1. Tree

Tree is defined as an undirected graph without any circular vertex. A graph is called as a tree if it doesn't have any circular vertex and it was connected each other. Tree, in computer science, is usually processed recursively because of its ability to act as a subtree where every subtree is also a tree, as shown in picture 2.1.



Picture 2.1

In picture 2.1, We can see that the subtree is also a tree. So it is easier to process it recursively by having an empty tree as a basis.

The definition of tree :

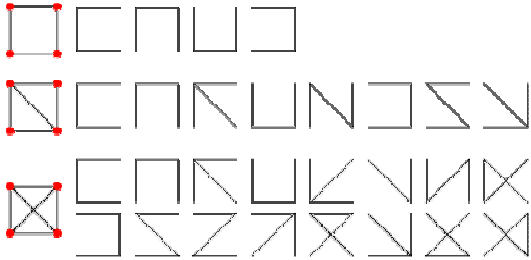
Assume that $G = (V, E)$ is an undirected graph and has n vertices. Then, all of those statement is true :

1. G is a tree.
2. Every pairs of vertices in G is connected by a singular path.
3. G is connected each other.
4. G contains no circuit.

5. Adding one edge in a vertex will only create one circuit.
6. G has $(n-1)$ edges.

1.1 Spanning tree

Spanning tree is a subgraph which can be defined as a tree. This tree can be obtained by cutting edges in a graph. One graph may have two or more spanning tree. The illustration is in picture below



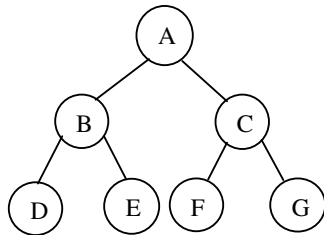
Picture 2.2

<http://mathworld.wolfram.com/SpanningTree.html>
retrieved in 7 Dec 2014

The rightmost side is graphs and the rest side is its spanning trees.

1.2 Rooted tree

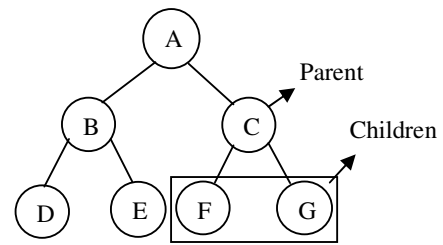
Rooted tree is defined as a tree which has a root and all of its edges have directions. It was largely used in computer science especially in data structure. The tree below is classified as a rooted tree.



Picture 2.3

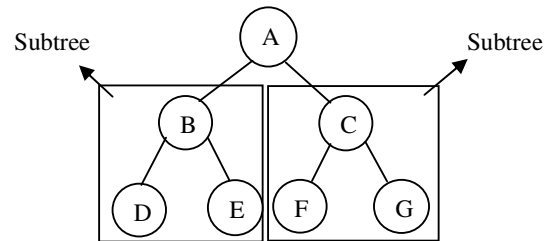
There are some terminology in rooted tree, such as:

a. *Child / Children and parent*, supposed that T is a rooted tree. If V is a vertex in T other than root, the parent of V is the unique vertex U such that there is a directed edge from U to V. In picture below, F and G are children from C and C is the parent of them.



Picture 2.4

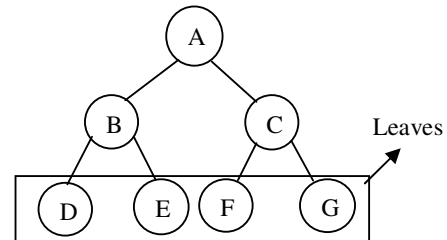
b. *Subtree*, the subtree of A is a tree consisting only a node in A and all of its descendants. Picture below clearly showed the subtree from node A.



Picture 2.5

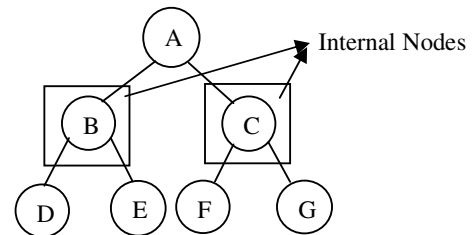
c. *Degree*, the degree of a node is defined as the total of children or subtrees which it has. In picture 2.5, Node C has two degrees because it has two children (F and G) and also two subtrees.

d. *Leaf / leaves*, defined as nodes without having any child or, in a similar way, have a zero degree. In picture below, we can say that D, E, F, G is leaves for it have no child.



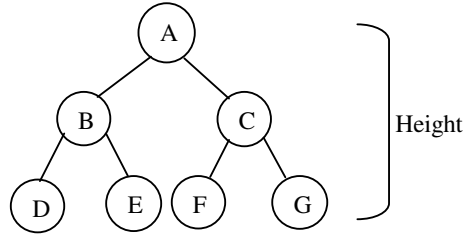
Picture 2.6

e. *Internal nodes*, all of the nodes that have a child / children is called as internal nodes. In picture 2.1, node B, alongside with node C, is internal node.



Picture 2.7

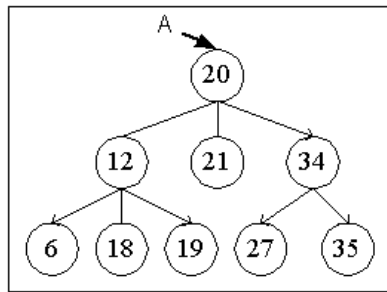
f. *Height or depth*, height is the longest path from the root to a leaf. In illustration below, the height of the rooted tree is 3, since one of its longest path is A, C, G.



Picture 2.8

1.3 N-ary tree

N-ary tree is a rooted tree with all nodes have at most N children. N-ary tree will be used, in this paper, as a parse tree which will be explained later.



Picture 2.9

<http://pioneer.netserv.chula.ac.th/~skrung/2301365/Lecture004.html> retrieved in 7 Dec 2014 16:40

In the picture 2.9 above, we can see a ternary tree, for its most children is three.

2. Recursion

Recursion is a process that called itself in similar way. It is used both in computer science and mathematics. It is also largely used in language theory.

A recursion function is defined by two parts:

a. Basis

The part that will stop the recursion and produce a value

b. Recursion

This part shall reduce all other case toward the basis.

Here is an example of recursive functions:

$f(x) = 0$	$x = 0$	Basis
$f(x) = x \cdot f(x-1)$	$x > 0$	Recursion

The functions above is factorial, which will return 1 if the input was 0. The recursion part will reduce input value and will return the multiplication of x and the value of the factorial of $x-1$. Then, the recursion will stop when x is zero, which was the basis in this case.

There are many definition in language that used the recursion :

1. String, a set of characters.
2. Null String, defined as an empty string
3. Alphabet, a set of characters which construct a string

3. Context Free Grammar

Context Free Grammar is a formal notation for expressing the recursive definitions of languages. The language it defined is called as Context Free Language. It was generally used to define the rules of the languages.

There are four important components in Context Free Grammar :

1. There is a finite set of symbol that form the strings of the language being defined. We call this alphabet as terminal symbols.
2. There is a finite set of variables, also called as non-terminals or syntactic categories
3. One of the variables represents the language being defined; it is called the start symbol.
4. There is a finite set of productions or rules that represent the recursive definition of language. Each production consist of :

- A variable that is being defined by the production. This variable is often called the head of the production.
- The production symbol \rightarrow
- A string of zero or more terminals and variables. This string, called the body of the production, represents one way to form strings in the language of the variable of the head.

We can represent CFG by four components, which is $G=(V,T,P,S)$, where V is the set of variables, T the terminals, P the set of productions, and S the start symbol.

Here is an example of CFG notation :

$G = (\{S,A\}, \{a,b\}, P, \{S\})$ $P :$ $S \rightarrow aSa \mid Ab$ $A \rightarrow Aa \mid b$

S is a start state and S can produce either aSa or Ab . A is a variable that can produce Aa or b . In both variable S and A , there are a recursive notation, aSa in S and Aa in A , which gives them ability to call itself. Now we already have the CFG, but how it works?

We can use the derivation to produce the language defined by the context free grammar. Those derivation can be represented as a tree which we called as parse tree.

3.1 Parse Tree

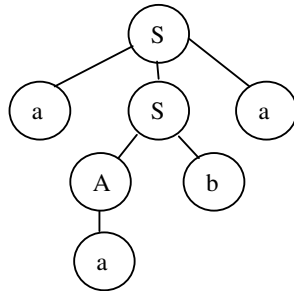
This tree has been proved to be extremely useful in representing the derivations. This parse tree is tied closely to the existence of derivations and recursive inference.

The parse tree for grammar $G = (V, T, P, S)$ is:

- Each interior node is labeled by a variable in V .
- Each leaf is labeled by either a variable, a terminal, or \square , then it must be the only child of its parent.
- If an interior node is labeled A , and its children are labeled

$$X_1, X_2, \dots, X_k$$

respectively, from left, then $A \rightarrow X_1, X_2, \dots, X_k$ is a production in P .



Picture 2.10

In picture 2.10 above, we see the parse tree. S is a start state have to be the root. And then, it can produce aSa . We found another S which is the child of the root S . Node a is also the child of S but since it was already a terminal, it can't produce anything. S then produced Ab . After that, A produced a terminal a . Finally, the derivation stopped. We can see the tree produce leaves $aaba$. It means that $aaba$ is a language for CFG G .

III INTRODUCTION TO C LANGUAGE

C language first developed by Dennis Ritchie between 1969 and 1963 at AT&T Bell Labs. Like most procedural languages, C has capability for structured programming. C is one of the most widely used programming language of all time, and C compilers are available in most of available computer architectures and operating systems.

C is an imperative language. It was designed to be compiled using a straightforward compiler. C is a coveted programming language since it was relatively simple and fast because most C implementations compile directly to machine code.

3.1 Some simple syntax for C language

C program's source text uses the semicolon as a statement terminator and curly braces for grouping blocks of statements. The language also allows you to declare new variables in the middle of the program.

There are some simple syntax in C that will be used in making the Context Free Grammar in this paper:

1. Conditional declaration

```
if (Expression) {
    Statement
}
```

```
if (Expression) {
    Statement
} else {
    Statement
}
```

2. Looping declaration

```
while (Expression) {
    Statement
}
```

```
do {
    Statement
} while (Expression);
```

3. Comment

```
{ Any String }
```

4. Assignment

```
a=b*5;
```

5. Expression

```
a<80
```

6. Input / Output

```
printf(".....");
scanf(".....");
```

7. Variable declaration

```
type var;
typedef struct {
    type var;
    .....;
} new_type
```

8. Statement

Statement contains conditional declaration, looping declaration, comment, assignment, Input/output, and variable declaration.

IV. CONTEXT FREE GRAMMAR FOR C'S SYNTAX

Context free grammar is the most effective way in representing the syntax. We have already limit the syntax

which will be checked here. So, the notation wouldn't be so complicated. We will use the recursive capability of CFG to express the repetitive notation and then we will parse it down using the parse tree.

Here is the CFG :

(Start state) $S \rightarrow \{ A \}$
 (Statement) $A \rightarrow \text{if } (E) S \mid \text{if } (E) S \text{ else } S \mid \text{while } (E) S \mid \text{do } S \text{ while } (E); \mid C \mid D \mid F \mid G \mid AA$
 (Comment) $C \rightarrow \text{Comment}$
 (Assignment) $D \rightarrow \text{Assignment}$
 (Expression) $E \rightarrow \text{Expression}$

It is the general notation for C syntax. We assume that the comment, assignment, expression, input/output, and variable declaration is already valid. So we didn't make the notations for them. As we can see, in variable A there are a production AA which indicates that there is a recursive notation. It gives capability to repeat the process of A and produce more terminals. You may also note that different notation can bring the same results, so this is not only the right version.

First, for the start state, we will have to create the curly braces in the beginning and end. A is the statement. Next, A can be filled with conditional, looping, comment, assignment, input/output, and variable declaration. Don't forget that we should put AA to make repeated productions. For the expression we will use it in looping and conditional.

4.1 Checking the syntax using the parse tree

Input languages can be checked by building the parse tree. We could make the tree started by defining its root as a start state. Then we can derive the start state into another state or terminal. We have to try for making leaves of the tree is equal to the input language. The process is shown below. The circle is mark as variables, while the square is marked as terminals.

Check whether this subprogram is valid :

```
{
    printf("Hello world");
}
```

Building parse tree :

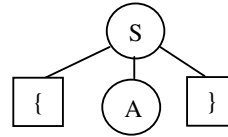
Put the start state as the root of the tree.

Step 1:



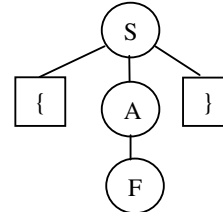
Search for the productions that have curly brace.

Step 2:



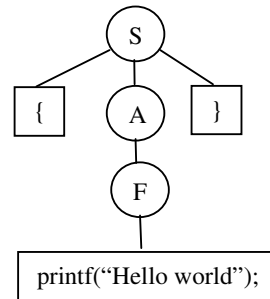
In variable A, search the production that will produce Input/Output, which is F

Step 3:



See the production of F, which is Input/output.

Step 4:



Since the leaves is equal to the subprogram, the syntax is valid.

Another example, Check whether this subprogram is valid:

```
{
    if b < 8 {
        x = 8 + b ;
    }
}
```

Building the parse tree:

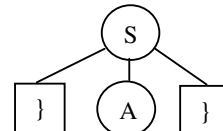
Put the start state as the root of the tree.

Step 1:



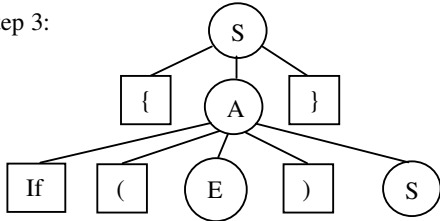
Search for the productions that have curly brace.

Step 2:



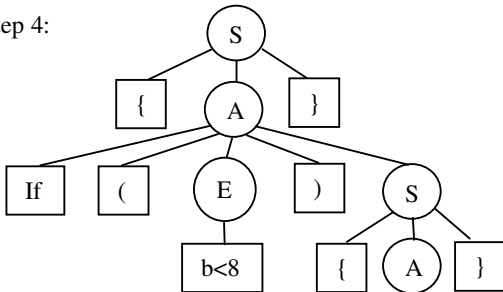
In variable A, search the production that will produce if then without else

Step 3:



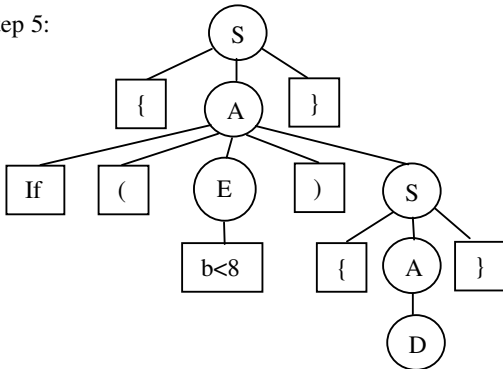
See the production for E and S.

Step 4:



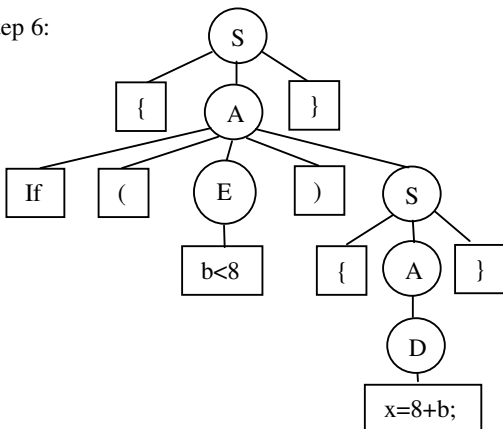
In variable A, search the production that will produce assignments, which is D

Step 5:



See the production is D which is the assignment.

Step 6:



As we can see, the leaves don't produce the exact language of the subprogram. So, the subprogram is not valid. The tree has produced another language, which should be the valid syntax for it. The language which has been produced by the tree is

```
{
    if ( b<8 ) {
        x = 8 + b ;
    }
}
```

The language above is a valid one, so the parse tree only can produce the language following the rule of production in CFG.

The key of using the parse tree is trying to make the exact terminal at the leaves to the input language by following the rule of production that have been made through context free grammar. Whenever the result is not same as the input, then the input is not valid, and the leaves it produced should be the correct one.

VI. CONCLUSION

Tree theorem can be applied in checking the syntax in C language by using it as a parser. Tree is the best representation for this problem, since we use the context free grammar which have a recursive notation and we all know that tree works perfectly with recursion. Recursion is also plays an important role in defining the rule of context free grammar, since it gives the ability to produce the variable repeatedly.

The implementation of tree in checking the syntax of the C language is by having it as a parser, also known as the parse tree. By putting the start state in the root and having the production to its children node, we can determine whether the language is defined by the context free grammar.

The implementation of recursion in checking the syntax of the C language is by having it in the rule of production. The nature of language is having the string recursively called and it was defined in the context free grammar which has a recursive notation.

In conclusion, the tree and recursion theorem, alongside with the context free grammar, can be used to determine if the syntax in any language is valid or not. And in this paper, it was proven to be worked in the simplified C language.

VI. ACKNOWLEDGMENT

I would like to thank to God for his guide while preparing, writing and editing this paper. I would also express my gratitude to my advisor Dr. Ir. Rinaldi Munir, MT, who has helped me a lot and Dra. Harlili S.,

M.sc that had taught me lots about Context free grammar. I also thank to my friends and family who has given me a strength to finish this paper. Finally, I would like to thank to Institut Teknologi Bandung for the support during the completion of this paper.

REFERENCES

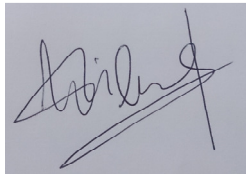
- [1] Rinaldi Munir, Matematika Diskrit 3rd, Bandung: Informatika , 2009
- [2] Giannini, Mario; Code Fighter, Inc.; Columbia University (2004). "C/C++". In Hossein, Bidgoli. The Internet encyclopedia 1. John Wiley and Sons. p. 164. ISBN 0-471-22201-1. Retrieved 16 December 2012.
- [3] Hopcroft, John, "Introduction to Automata Theory, Languages and Computation, 3rd edition, February 2006.
- [4] Barbara Partee and Mats Rooth. 1983. In Rainer Bäuerle et al., Meaning, Use, and Interpretation of Language. Reprinted in Paul Portner and Barbara Partee, eds. 2002. Formal Semantics: The Essential Readings. Blackwell.
- [5] <http://programmers.stackexchange.com/questions/141329/what-makes-c-so-popular-in-the-age-of-oop> accessed in 9 Dec 2014 13:48

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 27 November 2013

ttd

A square box containing a handwritten signature in black ink. The signature is stylized and appears to read 'William Sentosa'.

William Sentosa
13513026