University of Technology, Jamaica

Faculty of Engineering and Computing

School of Computing and Information Technology

Module Name: Analysis of Programming Languages

Module Code: CIT4004

APL Group Project Report

Academic Year 23/24: Semester 3

Tutor: David White

Group Members:

Kyle Parris - 1804904

Chloe Hibbert - 1902963

Pete Aris - 1704057

The paradigm to which our programming language belongs to is both the logical and functional paradigms as it runs programs using a set of functions and is used to evaluate if these functions are apart of the lambda calculus grammar. The programming language falls under the domain specific categorization as it is solely used to evaluate lambda calculus expressions and would be considered to be a high level language. The programming language used in developing the compiler for our lambda calculus programming language is C using LEX and YACC as at the time of development it was the easiest to use and it is efficient and portable. The C programming language is a general-purpose, imperative procedural language, designed for system programming (operating systems, device drivers) and application development.

The LEX File code:

```
%{
#include <stdio.h>
#include "y.tab.h"
%}

%%
[a-z]           return VARTKN;
#               return LAMBDATKN;
\(              return OBRACETKN;
\)              return CBRACETKN;
\.              return DOTTKN;
[ \t\n]+        /* ignore whitespace */;
%%
```

As shown above the tokens used for the language are VARTKN, LAMBDATKN, OBRACETKN, CBRACETKN, and DOTTKN. And the regular expressions used are [a-z], #, \(, \), \., and [\t\n]+.

The grammar used in developing this language is as follows:

```
<expr>::= <var> | <arg> | <lambda>|<expr> <expr>
<arg>::= ( <expr> )
```

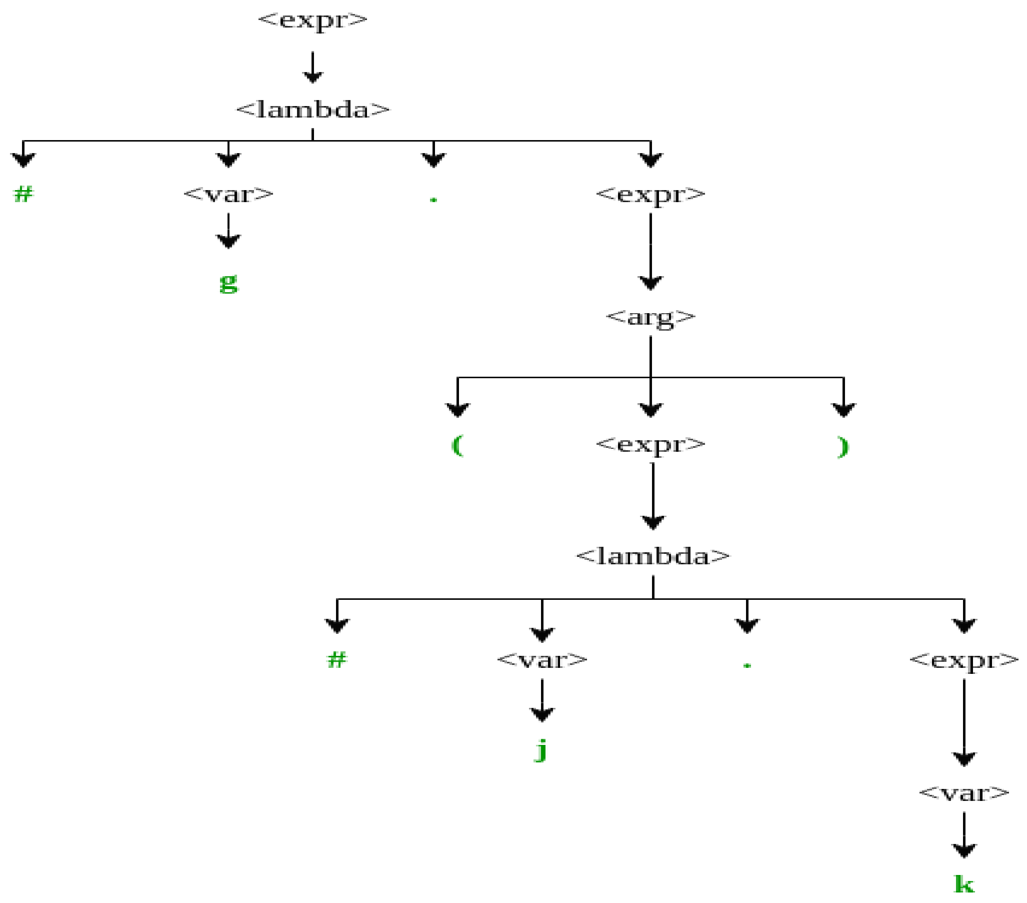
<lambda>::= # <var>. <expr>
<var>::= a-z

A sample program for our lambda calculus compiler would be #g.(#j.k) and bellow is the derivation and parse tree to prove that this program belongs to the grammar:

Derivation:

```
<expr> => <lambda>
         => #<var>.<expr>
         =>#g.<expr>
         =>#g.<arg>
         =>#g.(<expr>)
         =>#g.(<lambda>)
         =>#g.(#<var>.<expr>)
         =>#g.(#j.<expr>)
         =>#g.(#j.<var>)
         =>#g.(#j.k)
```

Parse Tree:



YACC File Code:

```
%{
#include <stdio.h>
#include <string.h>
#include "y.tab.h"

void yyerror(const char *str)
{
   fprintf(stderr,"%s: The Provide Expression is Not a Valid Lambda Calculus
Expression.\n\n",str);

   main();
}

int yywrap()
{
   return 1;
}

void main()
{
   printf("Enter a lambda calculus expression Press 'ctrl + z' to exit:\n");

   yyparse();
}

void valid()
{
  printf("Valid lambda calculus expression.\n\n");
  main();
}


%}

%token VARTKN LAMBDATKN OBRACETKN CBRACETKN DOTTKN

%%

commands: /*empty */
       | commands command
       ;

command:
       expr
       ;
```

```
/*
  Expression: the most generic rule for any lambda calculus expression.
  It can be a variable, an application of two expressions, or a lambda abstraction.
*/
expr:
    lambda
    {
      valid();
    }
    | arg
    {
      valid();
    }
    | expr expr
    {
      valid();
    }
    | VARTKN
    {
      valid();
    }
    ;

lambda:
    LAMBDATKN VARTKN DOTTKN expr
      {
       //printf("ABSTRACTION");
      }
      ;

arg:
    OBRACETKN expr CBRACETKN
    {
     //printf("ARGUMENT");
    }
    ;

%%
```

References

GeeksforGeeks. (2024, June 10). *C programming language tutorial*. https://www.geeksforgeeks.org/c-programming-language/.

W3Schools Online Web Tutorials. (2024, July 12). *C Tutorial*. https://www.w3schools.com/c/.