Universidade Federal de Minas Gerais
Departamento de Ciência da Computação
Compiladores

Trabalho Prático 1
Analisador Léxico e Sintático LALR

Professora: Mariza Andrade da Silva Bigonha
Aluno: Thiago Fioravante de Matos

Introdução

Análise léxica é o processo de analisar a entrada de linhas de caracteres (tal como o código-fonte de um programa de computadores) e produzir uma seqüência de símbolos chamado "símbolos léxicos" (lexical tokens), ou somente "símbolos" (tokens), que podem ser manipulados mais facilmente por um parser (leitor de saída).

A Análise Léxica é a forma de verificar determinado alfabeto. Quando analisamos uma palavra, podemos definir através da análise léxica se existe ou não algum caracter que não faz parte do nosso alfabeto, ou um alfabeto inventado por nós. O analisador léxico é a primeira etapa de um compilador, logo após virá a análise sintática.[1]

Em ciência da computação e linguística, análise sintática (também conhecido pelo termo em inglês *parsing*) é o processo de analisar uma sequência de entrada (lida de um arquivo de computador ou do teclado, por exemplo) para determinar sua estrutura gramatical segundo uma determinada gramática formal. Essa análise faz parte de um compilador, junto com a análise léxica e análise semântica.

A análise sintática transforma um texto na entrada em uma estrutura de dados, em geral uma árvore, o que é conveniente para processamento posterior e captura a hierarquia implícita desta entrada. Através da análise léxica é obtido um grupo de tokens, para que o analisador sintático use um conjunto de regras para construir uma árvore sintática da estrutura.[2]

Objetivo

Implementar o analisador léxico e o analisador sintático LALR para uma determinada gramático.

O seu trabalho receberá como entrada programas fontes, por exemplo, testeTP1, testeTP2, etc, e fará as análises léxica e sintática, produzindo como resultado as produções usadas durante a análise sintática e uma mensagem dizendo se os testes submetidos ao front-end do compilador estão sintáticamente corretos.

Ferramentas

As ferramentas utilizadas neste trabalho foram o JLex[3], para gerar o analisador léxico, e o Cup[4], para gerar o analisador sintático.

O JLex é um gerador de analisador léxico, derivado do Lex e escrito para a linguagem Java.

O Cup é um gerador de analisador sintático LALR, baseado no conhecida ferramenta YACC, mas também escrita para a linguagem Java.

JLex

O JLex gera um analisador léxico a partir de um arquivo de entrada, que contém as especificações dos tokens da gramática desejada. A estrutura do arquivo de entrada é definida a seguir:

Um arquivo de entrada JLex é organizado em três seções, separadas por uma porcentagem dupla(%%). O arquivo de entrada deve conter o seguinte formato:

Código do usuário
%%
Diretivas JLex
%%
Regras de expressões regulares

A seção do código do usuário é copiada exatamente como é definida para dentro do arquivo de saída. Esta área provê espaço para a implementação de classes utilitárias e tipos de retorno.

A seção de diretivas JLex contém definições de macros e nomes de estados.

A última seção, a de regras de expressões regulares, contém as regras para o analisador léxico, cada uma consistindo de três partes: uma lista opcional de estados, uma expressão regular e uma ação.[5]

Cup

O Cup gera um analisador sintático LALR a partir de um arquivo de entrada, que contém a gramática a ser utilizada.

O arquivo é dividido em quatro partes. A primeira parte provê espaço para declarações de como o parser será gerado. A segundo parte declara os terminais e não terminais que compões a gramática.

A terceira parte determina a precedência e associatividade dos terminais. A quarta e última parte contêm a especificação da gramática.

Modo de Utilização

Para utilizar o compilador, deve-se primeiro definir o arquivo léxico de entrada, `lexico.lex` para o JLex.

Considerando que o JLex está instalado[7], para gerar o analisador léxico deve-se digitar a seguinte linha de comando:

```
java JLex.Main lexico.lex
```

Este comando irá gerar como saída o arquivo `lexico.lex.java`.

Em seguida, deve-se utilizar o Cup, para gerar o analisador sintático. Com o arquivo de entrada parser.cup, devemos digitar a seguinte linha de comando:

```
java -jar java-cup-11a.jar parser.cup
```

Este comando irá gerar como saída os arquivos `parser.java` e `sym.java`

Por fim, para gerar o analisar final, deve-se digitar a seguinte linha de comando:

```
javac -classpath java-cup-11a-runtime.jar lexico.lex.java
              parser.java sym.java TP1.java
```

Saída do JLex

```
Processing first section -- user code.
Processing second section -- JLex declarations.
Processing third section -- lexical rules.
Creating NFA machine representation.
NFA comprised of 191 states.
Working on character classes.::::::::::.::..::::::::::.::
:.................::.
NFA has 37 distinct character classes.
Creating DFA transition table.
Working on DFA states.................................
.............................
Minimizing DFA transition table.
69 states after removal of redundant states.
Outputting lexical analyzer code.
```

Saída do Cup

```
---- CUP v0.11a beta 20060608 Parser Generation Summary ---
  0 errors and 0 warnings
  26 terminals, 14 non-terminals, and 35 productions
declared,
  producing 62 unique parse states.
  0 terminals declared but not used.
  0 non-terminals declared but not used.
  0 productions never reduced.
  0 conflicts detected (0 expected).
  Code written to "parser.java", and "sym.java".
---------------------------------- (v0.11a beta 20060608)
```

Conclusão

Este trabalho teve como objetivo construir uma parte do compilador, o analisador léxico e o sintático. Para tal, foi necessário a utilização das ferramentas JLex e Cup.

O JLex gera o analisador léxico, enquanto que o Cup gera o analisador sintático.

Isso proporcionou um bom aprendizado das ferramentas citadas acima, pois agilizam bastante a geração dos analisadores.

Por fim, este trabalho proporcionou um melhor entendimento das fases léxica e sintática de um compilador.

Apêndice

Código `lexico.lex`

```
import java_cup.runtime.*;

class Utility {
      public static void afirmar(boolean expr){
            if (false == expr){
                  throw (new Error("Erro: Assercao Falhou."));
            }
    }
}

%%

%{private int comment_count = 0;%}

%cup
%unicode
%class Ylex
%line
%char

%state COMMENT, LINECOMMENT

%cupdebug

%{
      private Symbol symbol(int type){
            return new Symbol(type, yyline, yychar);
      }

      private Symbol symbol(int type, Object value){
            return new Symbol(type, yyline, yychar, value);
      }
%}

ALPHA=[A-Za-z]
DIGIT=[0-9]
NONNEWLINE_WHITE_SPACE_CHAR=[\ \t\b\012]
WHITE_SPACE_CHAR=[\n\ \t\b\012]
STRING_TEXT=(\\\"|[^\n\"]|\\{WHITE_SPACE_CHAR}+\\)*
COMMENT_TEXT=([^/*\n]|[^*\n]"/"[^*\n]|[^/\n]"*"[^/\n]|"*"[^/\n]|"/"[^*
\n])*

num = ({DIGIT})({DIGIT})*
real = {DIGIT}(\.{DIGIT}+)?(E[+|-]?{DIGIT}+)?
id = ({ALPHA}|_)({ALPHA}|{DIGIT}|_)*

%%

<YYINITIAL> ";" { return new Symbol(sym.PV);}
<YYINITIAL> "(" { return new Symbol(sym.AP);}
<YYINITIAL> ")" { return new Symbol(sym.FP);}
<YYINITIAL> "{" { return new Symbol(sym.AC);}
<YYINITIAL> "}" { return new Symbol(sym.FC);}
<YYINITIAL> "+" { return new Symbol(sym.ADD);}
```

```
<YYINITIAL> "-" { return new Symbol(sym.SUB);}
<YYINITIAL> "*" { return new Symbol(sym.MULT);}
<YYINITIAL> "/" { return new Symbol(sym.DIV);}
<YYINITIAL> "<"  { return new Symbol(sym.LT);}
<YYINITIAL> "<=" { return new Symbol(sym.LE);}
<YYINITIAL> ">"  { return new Symbol(sym.GT);}
<YYINITIAL> ">=" { return new Symbol(sym.GE);}
<YYINITIAL> "=" { return new Symbol(sym.ATRIB);}
<YYINITIAL> "int" { return new Symbol(sym.INT);}
<YYINITIAL> "char" { return new Symbol(sym.CHAR);}
<YYINITIAL> "bool" { return new Symbol(sym.BOOL);}
<YYINITIAL> "float" { return new Symbol(sym.FLOAT);}
<YYINITIAL> "if" { return new Symbol(sym.IF);}
<YYINITIAL> "else" { return new Symbol(sym.ELSE);}
<YYINITIAL> "while" { return new Symbol(sym.WHILE);}
<YYINITIAL> {num}
{ return new Symbol(sym.num, new Integer(yytext())); }
<YYINITIAL> {real}
{ return new Symbol(sym.real, new Float(yytext())); }
<YYINITIAL> {id}
{ return new Symbol(sym.id, new String(yytext())); }

<YYINITIAL,COMMENT> \n { }

<YYINITIAL> "/*"
{
      yybegin(COMMENT);
      comment_count = comment_count + 1;
}

<YYINITIAL> "//" {yybegin(LINECOMMENT);}
<LINECOMMENT> [^\n] {}
<LINECOMMENT> [\n] {yybegin(YYINITIAL);}

<COMMENT> "/*" { comment_count = comment_count + 1; }

<COMMENT> "*/" {
      comment_count = comment_count - 1;
      Utility.afirmar(comment_count >= 0);
      if (comment_count == 0) {
            yybegin(YYINITIAL);
      }
}

<COMMENT> {COMMENT_TEXT} { }

<YYINITIAL> {NONNEWLINE_WHITE_SPACE_CHAR}+ { }

<YYINITIAL,COMMENT, LINECOMMENT> . {
        System.out.println("Caractere ilegal: <" + yytext() + ">");
}
```

Código `parser.cup`

```
import java_cup.runtime.*;

//Codigo do usuário
parser code {:
      public static void main(String args[]) throws Exception{
            new parser(new Yylex(System.in)).parse();
            System.out.println("Analise Sintatica Concluida.");
      }

      //Funcao do CUP: reporta um erro sintatico e continua execucao
      public void syntax_error(java_cup.runtime.Symbol current){
            String msg = "Sintatico em " +current.left+ "," +
            current.right+","+current.value;
            report_error(msg, current);
      }

      //Funcao do CUP: reporta um erro fatal e termina o programa
      public void report_fatal_error(String message, Object current){
            report_error("Fatal", current);
            System.exit(1);
      }

      //Funcao CUP: imprime o tipo do erro e a informacao
      public void report_error(String message, Object info){
            System.err.println("Erro "+message+" Info: "
            +info.toString());
      }

:};

/* TERMINAIS*/
terminal          AC, FC;     /*Abre Chave, Fecha Chave*/
terminal          AP, FP;     /*Abre Parenteses, Fecha Parenteses*/
terminal          PV;         /*Ponto e virgula*/
terminal          ATRIB;      /*Atribuicao*/
terminal          LT, LE, GT, GE;/*Less Than, Less Equal, Greater Than,
Greater Equal*/
terminal          ADD, SUB, MULT, DIV;    /*Soma, Subtracao,
Multiplicacao, Divisao*/
terminal          INT, CHAR, BOOL, FLOAT; /*Inteiro, Caracter,
Booleano, Real*/
terminal          IF, ELSE, WHILE;        /*Controle de fluxo*/
terminal          id;         /*Identificador*/
terminal          num;  /*Constante Inteira*/
terminal          real; /*Constante Real*/

/* NAP TERMINAIS */
non terminal      program, block, decls, decl, type, stmts, stmt, rel;
non terminal      matchedstmt, unmatchedstmt;
non terminal      expr, term, unary;
non terminal      factor;

/* PRECEDENCIA */

precedence left ADD, SUB;
precedence left MULT, DIV;
precedence left AC, FC;
precedence left id;
```

```
/* The grammar */

start with program;
program           ::= block;
block             ::= AC decls stmts FC;
decls             ::= decls decl | /*empty*/;
decl              ::= type id PV;
type              ::= INT | CHAR | BOOL | FLOAT;
stmts             ::= stmts stmt | /*empty*/;
stmt              ::= id ATRIB expr PV
                            | matchedstmt
                            | unmatchedstmt
                            | WHILE AP rel FP stmt
                            | block;
matchedstmt       ::= IF AP rel FP matchedstmt ELSE matchedstmt;
unmatchedstmt     ::= IF AP rel FP stmt
                        | IF AP rel FP matchedstmt ELSE unmatchedstmt;
rel               ::= expr LT expr | expr LE expr | expr GE expr |
                        expr GT expr | expr ;
expr              ::= expr ADD term | expr SUB term | term;
term              ::= term MULT unary | term DIV unary | unary;
unary             ::= SUB unary | factor;
factor            ::= num | real;
```

Código `TP1.java`

```
public class TP1
{
    public static void main(String args[]) throws Exception
    {
        //Lexico
        Yylex y = new Yylex(System.in);
        /*Yytoken t;
        while ((t = y.yylex()) != null)
        {
            System.out.println(t);
        }*/

        //Sintatico
        parser myParser = new parser(y);
        myParser.parse();
    }
}
```

Bibliografia

[1] http://pt.wikipedia.org/wiki/Análise_léxica
[2] http://pt.wikipedia.org/wiki/Análise_sintática_(computação)
[3] http://www.cs.princeton.edu/~appel/modern/java/JLex/
[4] http://www2.cs.tum.edu/projects/cup/
[5] http://www.cs.princeton.edu/~appel/modern/java/JLex/current/manual.html
[6] http://www2.cs.tum.edu/projects/cup/manual.html
[7] http://www.cs.princeton.edu/~appel/modern/java/JLex/current/README