语法分析器

2020.05.22

编译原理

目录

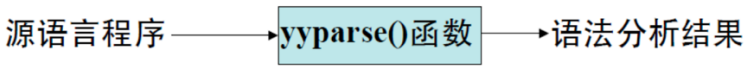
1实验目的以及要求

1.1 理解编译器的工作机制，并编译器的工作原理

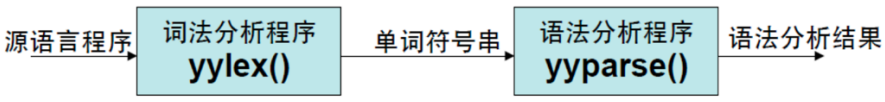
1.2 掌握语法分析器生成工具 bison 的用法

YACC 使用 LALR(1)文法表示上下文无关文法

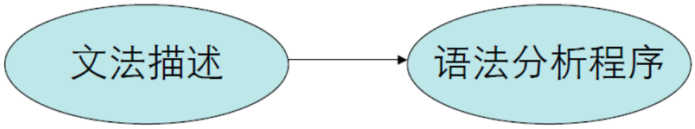
YACC是一个语法分析程序的自动产生系统



词法分析程序与语法分析程序的关系



YACC的工作原理:



1.3 测试输入文件的要求

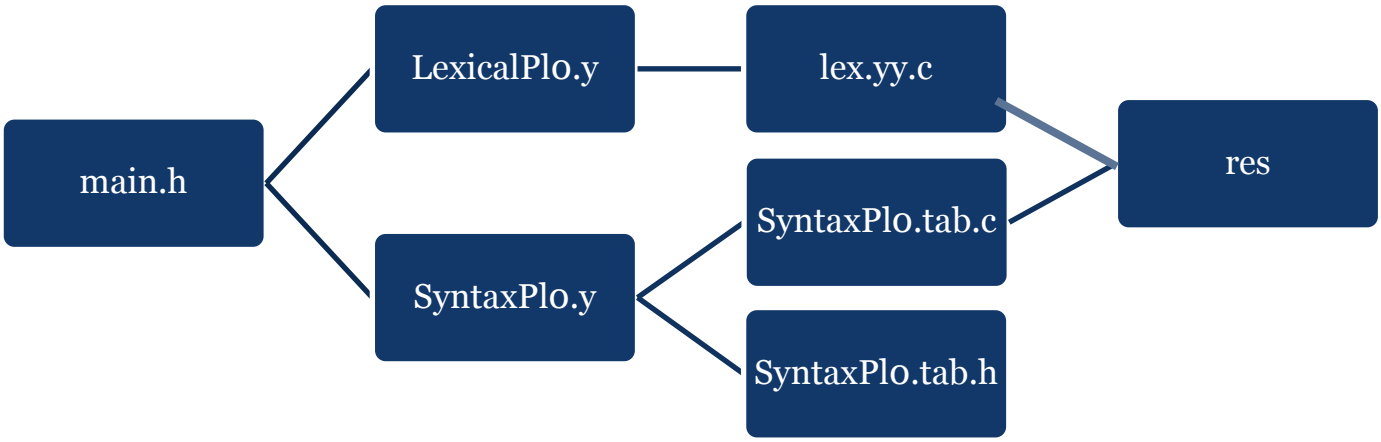
- 支持的所有语法成分（含扩展成分）
- 嵌套过程（三层）
- 并列过程

1.4 输出文件的要求

- 按归约顺序用到的语法规则序列。
- 语法树（或能表示语法单位的层次结构关系的其他形式）。

2 实验框架

2.1 核心程序层次结构



可执行文件 res 对测试程序进行语法分析，之后 draw.cpp 再对分析结果进行处理，得到处理结果后，使用 graphviz 中的 dot 工具画出语法树。

2.2 实验步骤

具体命令参考 readme.md 文件

- 用 flex 工具生成 lex.yy.c 文件
- 用 bison 工具生成 SyntaxPl0.tab.c 和 SyntaxPl0.tab.h 文件
- 使用 g++将 SyntaxPl0.tab.c 和 lex.yy.c 文件联合编译生成可执行文件 res
- 使用可执行文件 res 对测试程序 test_sum.pl0 语法分析
- 使用 draw.cpp 文件处理分析结果，构建语法树
- 使用 dot 工具画出语法树

3 设计思路

3.1 语法分析器

3.1.1 语法规则

语法规则对应于 EBNF 范式
例如：

左为 EBNF 范式
右为对应的语法规则

<程序> ::= <分程序>.

<分程序> ::= [<常量说明部分>][<变量说明部分>][<过程说明部分>]<语句>

语法规则就是对 EBNF 范式进行了翻译，并且在规约时，将所做的规约动作的规则输出到两个文件中。

- 第一个文件存放按照规约顺序用到的语法规则序列
- 第二个文件将用于构造语法树，在下文中具体解释

```
Program     : SubPro DOT {
                fprintf(fi,"Program -> SubPro DOT\n");
                fprintf(fh,"Program -> SubPro DOT\n");  }
            ;
SubPro      : DeclarePart Statement {
                fprintf(fi,"SubPro -> DeclarePart Statement\n");
                fprintf(fh,"SubPro -> DeclarePart Statement\n");  }
            | Statement {
                fprintf(fi,"SubPro -> Statement\n");
                fprintf(fh,"SubPro -> Statement\n");
            }
            ;
DeclarePart : ConstDec {
                fprintf(fi,"DeclarePart -> ConstDec\n");
                fprintf(fh,"DeclarePart -> ConstDec\n");
            }
            |ConstDec VarDec {
                fprintf(fi,"DeclarePart -> ConstDec VarDec\n");
                fprintf(fh,"DeclarePart -> ConstDec VarDec\n");
            }
            |ConstDec ProDec {
                fprintf(fi,"DeclarePart -> ConstDec ProDec\n");
                fprintf(fh,"DeclarePart -> ConstDec ProDec\n");
            }
            |ConstDec VarDec ProDec {
                fprintf(fi,"DeclarePart -> ConstDec VarDec ProDec\n");
                fprintf(fh,"DeclarePart -> ConstDec VarDec ProDec\n");
            }
            |VarDec ProDec {
                fprintf(fi,"DeclarePart -> VarDec ProDec\n");
                fprintf(fh,"DeclarePart -> VarDec ProDec\n");
            }
            |VarDec {
                fprintf(fi,"DeclarePart -> VarDec\n");
                fprintf(fh,"DeclarePart -> VarDec\n");
            }
            |ProDec {
                fprintf(fi,"DeclarePart -> ProDec\n");
                fprintf(fh,"DeclarePart -> ProDec\n");
            }
            ;
```

下面为扩展文法的 EBNF 范式和语法规则的对应

数组

<变量说明部分> ::= (VAR<标识符>[(ARRAY \[<无符号整数>..<无符号整数>|<表达式>{,<无符号整数>..<无符号整数>|<表达式>}\]OF<类别 1>) | (:<类别 2>)]{,<标识符>[(ARRAY \[<无符号整数>..<无符号整数>|<表达式>{,<无符号整数>..<无符号整数>|<表达式>}\]OF<类别 1>) | (:<类别 2>)};)

```
IdentiDef    : IdentiDef COMMA IdentiObject {
                  fprintf(fi,"IdentiDef -> IdentiDef , IdentiObject\n");
                  fprintf(fh,"IdentiDef -> IdentiDef , IdentiObject\n");
               }
               | IdentiObject{
                  fprintf(fi,"IdentiDef -> IdentiObject\n");
                  fprintf(fh,"IdentiDef -> IdentiObject\n");
               }
               ;

IdentiObject   : IDENTIFIER ARRAY Realm OF ArrayType {
                  fprintf(fi,"IdentiObject -> IDENTIFIER ARRAY Realm OF ArrayType\n");
                  fprintf(fh,"IdentiObject -> IDENTIFIER ARRAT Realm OF ArrayType\n");
               }
               | IDENTIFIER OF ValueType {
                  fprintf(fi,"IdentiObject -> IDENTIFIER OF ValueType\n");
                  fprintf(fh,"IdentiObject -> IDENTIFIER OF ValueType\n");
               }
               | IDENTIFIER {
                  fprintf(fi,"IdentiObject -> IDENTIFIER\n");
                  fprintf(fh,"IdentiObject -> IDENTIFIER\n");
               }
               ;
Realm        : LEFTBRAC ArrayNDim RIGHTBRAC {
                  fprintf(fi,"Realm -> [ ArrayNDim ]\n");
                  fprintf(fh,"Realm -> [ ArrayNDim ]\n");
               }
               ;
ArrayNDim    : ArrayNDim COMMA ArrayDimObj {
                  fprintf(fi,"ArrayNDim -> ArrayNDim COMMA ArrayDimObj\n");
                  fprintf(fh,"ArrayNDim -> ArrayNDim COMMA ArrayDimObj\n");
               }
               | ArrayDimObj{
                  fprintf(fi,"ArrayNDim -> ArrayDimObj\n");
                  fprintf(fh,"ArrayNDim -> ArrayDimObj\n");
               }
               ;
ArrayDimObj  : INTEGER_VAL OMIT INTEGER_VAL {
                  fprintf(fi,"ArrayDimObj -> INTEGER_VAL OMIT INTEGER_VAL\n");
                  fprintf(fh,"ArrayDimObj -> INTEGER_VAL OMIT INTEGER_VAL\n");
               }
ArrayType    : INT {
                  fprintf(fi,"ArrayType -> INT\n");
                  fprintf(fh,"ArrayType -> INT\n");
               }
               | FLOAT {
                  fprintf(fi,"ArrayType -> FLOAT\n");
                  fprintf(fh,"ArrayType -> FLOAT\n");
               }
```

For 和 Repeat

<FOR 循环语句>::= FOR<变量>:=<表达式>TO<表达式>DO<语句>

<REPEAT 循环语句>::= REPEAT<语句>UNTIL<条件>

```
ForStm       : FOR IDENTIFIER ASSIGN Expr TO Expr DO Statement {
                  fprintf(fi,"ForStm -> FOR IDENTIFIER := Expr TO Expr DO Statement\n");
                  fprintf(fh,"ForStm -> FOR IDENTIFIER := Expr TO Expr DO Statement\n");
               }
               ;
RepeatS      : REPEAT Statement  SEMI UNTIL Condition {
                  fprintf(fi,"RepeatS -> REPEAT Statement SEMI UNTIL Condition\n");
                  fprintf(fh,"RepeatS -> REPEAT Statement SEMI UNTIL Condition\n");
               }
               ;
```

更多的实现细节，可以在原代码和 EBNF 的 word 文档中看到。

3.2 词法分析器

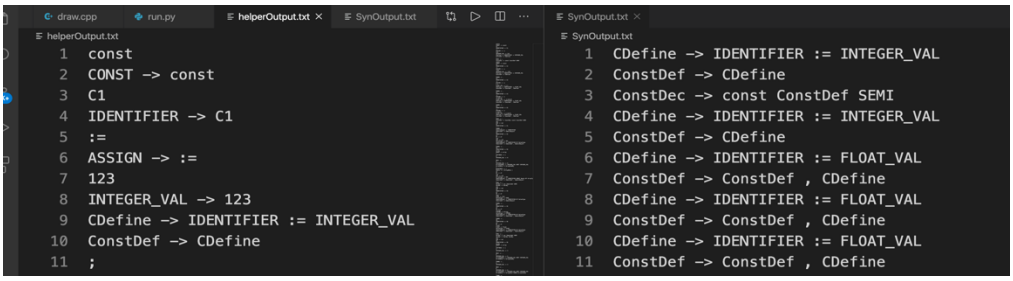
与上次实验的词法分析器相比，做出了以下改进

- 使用 strcasecmp()函数进行不区分大小写的匹配
- 每当扫描到一个词将它规约为 token 的时候返回对应的词法匹配结果
- 每当扫描到单词以及决定匹配方式分别将扫描的单词和匹配方式输出到一个文件中，和语法分析器中第二个输出的文件相同，下面建立语法树会用到

3.3 构建语法树

3.3.1 生成词法和语法分析的综合文档

词法分析器中扫描到词时立刻输出当前的 yytext，接着输出所选择的词法规则，如图 a 中所示，扫描测试程序 test_sum.pl0 时，扫描到 const,将 const 输出，词法选择将 const 匹配为 CONST，再输出“CONST->const”。



图表 a：左边为帮助文档，右边为语法规则的规约序列

语法分析器中，决定使用某个语法规则进行规约以后，将该规则输出到 2 个文件中，一个是图 a 中左边的帮助文档，一个是图 a 中右边的语法规则的规约序列文档。例如首先使用“CDefine -> INDETIFIER := INTEGER_VAL”进行规约，对应于左边第 9 行，右边第 1 行。

3.3.2 使用栈进行规约

遇到 yytext 的内容就压入栈中，遇到词法规则或者语法规则就进行规约，将栈顶的对应规则的“->”后数量的内容 pop 出栈，将“->”左侧的内容入栈。该内容在 draw.cpp 中实现，对应的核心函数如图 b 所示。

其中 reduce 函数将 pop 掉的部分作为子结点插入到规约后生成的结果当中。对“Program”项目特殊处理，如果是这个项目说明是根节点 root。

```
50  void read_file(string file_name)
51  {
52      fstream fin;
53      fin.open(file_name);
54      assert(fin.is_open());
55
56      string line;
57      while (!fin.eof())
58      {
59          getline(fin, line);
60          if (line.find("->") != -1)
61          {
62              //cout<<line<<endl;
63              vector<string> a = parse(line);
64              reduce(a);
65              if (a[0] == "Program")
66                  root = syntax.top();
67          }
68          else
69          {
70              syntax.push(new Node(line));
71          }
72      }
73
74      fin.close();
75  }
```

图表 b

3.3.3 用 dot 工具绘图

- 使用 DFS（深度优先搜索）将语法树的信息输出到 ASTvis.gv 文件中
- 在命令行使用 dot 绘制语法树

为什么不一边规约一边直接生成树

关键问题在于保证语法树中同一层结点的顺序。

如果在确定规约的时候就生成结点，那么非终结符不好判断应该将什么作为其子结点。

如果在词法分析器识别 token 的时候生成结点，考虑到 flex 会进行超前搜索（flex 的—Batch），那么某些终结符和非终结符的顺序会出现混乱。

4 遇到的问题和解决方法

问题 1

- 使用 bison 时出现 shift/reduce 错误，如图 c 所示

```
sizihua@MacBookPro:Desktop/syntax <master*>$ bison SyntaxPl0.y -d
SyntaxPl0.y: conflicts: 4 shift/reduce
sizihua@MacBookPro:Desktop/syntax <master*>$
```

图表 c

- 图 d 中，左侧为其中一处 shif/reduce 错误，右侧为解决方法
  - 出现该错误的原因是语法规则中同时出现左递归和右递归，我这样写原以为可以简化语法规则，没想到会出现错误
  - 使用右边的方法改为左递归以后，错误就消失了

```
75          ;
76  ConstDef    : ConstDef COMMA ConstDef {
77              fprintf(fi,"ConstDef -> ConstDef , ConstDe
78              fprintf(fh,"ConstDef -> ConstDef , ConstDe
79               }
80          | IDENTIFIER ASSIGN INTEGER_VAL {
81              fprintf(fi,"ConstDef -> IDENTIFIER := INTE
82              fprintf(fh,"ConstDef -> IDENTIFIER := INTE
83               }
84          | IDENTIFIER ASSIGN FLOAT_VAL {
85              fprintf(fi,"ConstDef -> IDENTIFIER := FLOA
86              fprintf(fh,"ConstDef -> IDENTIFIER := FLOA
87               }
88          ;
89
90  VarDec      : VarDec VAR IdentiDef SEMI {
91              fprintf(fi,"VarDec -> VarDec var IdentiDe
92               fprintf(fh,"VarDec -> VarDec var IdentiDe
93              }
```

```
76          ;
77  ConstDef    : ConstDef COMMA CDefine {
78              fprintf(fi,"ConstDef -> ConstDef , CDefine
79              fprintf(fh,"ConstDef -> ConstDef , CDefine
80               }
81          | CDefine{
82              fprintf(fi,"ConstDef -> CDefine\n");
83              fprintf(fh,"ConstDef -> CDefine\n");
84          }
85          ;
86  CDefine     :IDENTIFIER ASSIGN INTEGER_VAL {
87              fprintf(fi,"CDefine -> IDENTIFIER := INTEG
88              fprintf(fh,"CDefine -> IDENTIFIER := INTEG
89               }
90          | IDENTIFIER ASSIGN FLOAT_VAL {
91              fprintf(fi,"CDefine -> IDENTIFIER := FLOAT
92              fprintf(fh,"CDefine -> IDENTIFIER := FLOAT
93               }
94          ;
```

图表 d

问题 2

二义文法

- IF Than Else 和 IF Than 出现移进规约冲突
  - 产生冲突的代码如图 e 所示

```
CondiStm      : IF Condition THEN Statement  {
                fprintf(fi,"CondiStm -> if Condition then Statement\n");
                fprintf(fh,"CondiStm -> if Condition then Statement\n");
                 }
              | IF Condition THEN Statement ELSE Statement {
                fprintf(fi,"CondiStm -> if Condition then Statement else Statement\n");
                fprintf(fh,"CondiStm -> if Condition then Statement else Statement\n");
                  }
              ;
```

图表 e

解决方式

- %nonassoc 声明 LOWER_THAN_ELSE 和 ELSE,如图 f 所示

```
24  %nonassoc LOWER_THAN_ELSE
25  %nonassoc ELSE
26  %%
27
```

图表 f

- 使用%prec 标记说明 IF THAN ELSE 的优先级和 LOWER_THAN_ELSE 相同，该符号的声明在 ELSE 的上方，所以其优先级低于 ELSE。由此一来通过定义优先级解决了二义文法的冲突。

```
CondiStm      : IF Condition THEN Statement %prec LOWER_THAN_ELSE {
                fprintf(fi,"CondiStm -> IF Condition THEN Statement\n");
                fprintf(fh,"CondiStm -> IF Condition THEN Statement\n");
                 }
              | IF Condition THEN Statement ELSE Statement {
                fprintf(fi,"CondiStm -> IF Condition THEN Statement ELSE Statement\n");
                fprintf(fh,"CondiStm -> IF Condition THEN Statement ELSE Statement\n");
                  }
              ;
```

图表 g