实现语法分析器

徐子越

2019 年 10 月 28 日

目录

1 作业要求

对预备工作中自然语言描述的简化 C 编译器的语言特性的语法，设计上下文无关文法进行描述, 借助 Yacc 工具实现语法分析器。

考虑语法树的构造：

1. 语法树数据结构的设计：节点类型的设定，不同类型节点应保存哪些信息，多叉树的实现方式
2. 实现辅助函数，完成节点创建、树创建等功能
3. 利用辅助函数，修改上下文无关文法，设计翻译模式
4. 修改 Yacc 程序，实现能构造语法树的分析器

考虑符号表处理的扩充:

1. 完成语法分析后，符号表项应增加哪些标识符的属性，保存语法分析的结果
2. 如何扩充符号表数据结构，Yacc 程序如何与 Lex 程序交互，正确填写符号表项

以一个简单的 C 源程序验证你的语法分析器，可以文本方式输出语法树结构，以节点编号输出父子关系，来验证分析器的正确性，如下例：

```
main()
{
  int a, b;
  if (a == 0)
    a = b + 1;
}
```

example

可能的输出为：

```
0 : Type Specifier,     integer,       Children:
1 : ID Declaration,     symbol: a      Children:
2 : ID Declaration,     symbol: b      Children:
3 : Var Declaration,                   Children: 0    1    2
4 : ID Declaration,     symbol: a      Children:
5 : Const Declaration,  value:0,       Children:
6 : Expr,               op: ==,        Children: 4    5
```

```
7 : ID Declaration,     symbol: a      Children:
8 : ID Declaration,     symbol: b      Children:
9 : Const Declaration,  value:1,       Children:
10: Expr,               op: +,         Children: 8    9
11: Expr,               op: =,         Children: 7    10
12: if statement,                      Children: 6    11
13: compound statement,                Children: 3  12
```

result

2 作业解析

2.1 作业目的

通过本次实验，希望同学们能够较为深刻地体会并理解语法分析器的作用，并且加深对 yacc 和 lex 程序编写的掌握和使用。此外，还希望同学们能够结合课上所学的语法分析器的内容，对该部分内容复习和巩固，提高实践能力。

2.2 作业思路

2.2.1 节点类型设定以及节点信息

根据作业要求，本次实验主要是构造并打印一颗语法树，所以会涉及到节点数据结构的构造以及不同的树节点应当保存哪些信息。

这里给大家提供的思路是可以将树节点进行分类，可以分为语句 Stmt 类型，表达式 Exp 类型，变量声明 Decl 类型三种大类型，其中 Stmt 类型应当根据自己的上下文无关文法确定具体可以实现的语句，建议实现的语句类型有 If 语句，While 语句，For 循环语句，赋值语句，输入语句，输出语句以及复合语句；建议使用枚举类型标识，以便将节点类型用 int 类型表示。

```
enum {StmtK,ExpK,DeclK};
enum {IfK,WhileK,AssignK,ForK,CompK,InputK,PrintK};
enum {OpK,ConstK,IdK,TypeK};
enum {VarK};
enum {Void,Integer,Char};
```

example

根据示例，树节点信息主要包括节点号，孩子节点，区分不同节点类型的类型 nodekind，以及根据不同节点类型所对应的具体类型 kind，比如 nodekind 为 Stmt，kind 为 If 表明该节点是一个 If 语句的节点。此外，如果该节点为标识符、符号或者数字应当进行保存并输出，考虑到三者不会同时存在，所以推荐使用 union 结构进行保存。还需要加入 type 变量保存变量类型，具体变量类型根据你所定义的上下文无关文法确定。

```
struct TreeNode
   {
     struct TreeNode * child[MAXCHILDREN];
     struct TreeNode * sibling;
     int lineno;
     int nodekind;
     int kind;
     union{ int op;
             int val;
           char *name; }attr;
     int value;
     int type;
   } ;
```

example

2.2.2 实现并利用辅助函数

主要需要完成节点的创建，将对应元素的数值进行填充，并根据上下文无关文法将多个节点进行连接，最终形成一颗完整的语法树。具体来说，需要构造的辅助函数有以下几种：

1. 根据上一小节所述，主要有三种类型的节点，所以需要对应三个创建节点的函数，参数为该节点的具体类型，该函数主要用在产生式进行规约时的翻译模式中，创建一个新的节点，方便最终的输出。以 Stmt 类型节点为例：

```
TreeNode * newStmtNode(int kind)
{
  TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
  int i;
  if (t==NULL)
   printf("Out of memory error at line %d\n",line);
  else {
    for (i=0;i<MAXCHILDREN;i++)
```

```
    t->child[i] = NULL;
    t->sibling = NULL;
    t->nodekind = StmtK;
    t->kind = kind;
    t->lineno = line++;
  }
  return t;
}
```

example

2. 输出整个语法树的函数，根据实例，不难看出，是递归打印节点的，采取具体思路是先打印孩子节点再打印孩子节点的兄弟节点进行递归打印的。该函数参数为语法树的根节点，开始符号存在的产生式的语义动作中执行。

```
void Display(struct TreeNode *p)
{
  struct TreeNode *temp ;
  temp =(struct TreeNode *) malloc (sizeof(struct TreeNode));
    for(int i=0;i<MAXCHILDREN;i++){
     if(p->child[i] != NULL)
     {
    Display(p->child[i]);
     }
  }
  ShowNode(p);
  temp=p->sibling;
  if(temp!=NULL){
      Display(temp);
  }
  return;
}
}
```

example

3. 此外，还需要一个具体打印某个节点的函数 Display。根据上一小节所述，根据节点的不同类型（可以通过 nodekind 以及 kind）进行相应的输出。

2.3 词法分析器举例

```
{identifier}  { TreeNode * t = (TreeNode *) malloc(sizeof(TreeNode));
                    for (int i=0;i<MAXCHILDREN;i++)
                    t->child[i] = NULL;
                    t->sibling = NULL;
                    t->nodekind = ExpK;
                    t->kind = IdK;
                    t->lineno = line++;
                    t->attr.name=hash1[getid(yytext,yyleng)];
                    yylval = t;
                    return ID;
          }
```

identifier

在词法分析器识别标识符之后，进行的语义动作首先创建一个空白节点，之后是对该节点中的元素进行赋值，由于此时的孩子节点与兄弟节点尚未确定，故指针值为空。

之后对节点类型进行赋值，nodekind 为表达式类型（ExpK），kind 为标识符类型（IdK）；此外，还要对全局变量 lineno 进行自加操作，表示该节点的序号；最后将该标识符放入符号表中，并将标识符进行记录，具体的符号表构造方式以及 hash 函数可自行构造。

在所有赋值操作结束之后，将节点赋值给 yyval 变量，该变量是一个全局变量，用于在词法分析器与语法分析器之间传递变量。此外，还需要注意的是 yyval 变量类型的定义，需要在 lex 程序声明 yylval 的类型，声明如下：

extern YYSTYPE yylval;

还需要在语法分析器中定义 YYSTYPE 的类型，我们这里给出的类型为 TreeNode。

2.4 语法分析器举例

在语法分析器中，以 for 循环语句进行举例，由于 for 循环语句的主体部分可以理解为以下结构：

```
for(expression1;expression2;expression3)
{
    statement;
}
```

for 语句

其中 expression1，expression2，expression3 可以为空，所以不难写出以下产生式以及语义动作。

```
for_stmt  :FOR LP exp SEMI exp SEMI exp RP stmt
      {
        $$ = newStmtNode(ForK);
        $$->child[0] = $3;
                $$->child[1] = $5;
                $$->child[2] = $7;
                $$->child[3] = $9;
      }
      |FOR LP SEMI exp SEMI exp RP stmt
      {
        $$ = newStmtNode(ForK);
        $$->child[0] = $4;
                $$->child[1] = $6;
                $$->child[2] = $8;
      }
      |FOR LP exp SEMI SEMI exp RP stmt
      {
        $$ = newStmtNode(ForK);
        $$->child[0] = $3;
                $$->child[1] = $6;
                $$->child[2] = $8;
      }
      |FOR LP exp SEMI exp SEMI RP stmt
      {
        $$ = newStmtNode(ForK);
        $$->child[0] = $3;
                $$->child[1] = $5;
                $$->child[2] = $8;
      }
      |FOR LP SEMI SEMI exp RP stmt
      {
        $$ = newStmtNode(ForK);
        $$->child[0] = $5;
                $$->child[1] = $7;
      }
      |FOR LP SEMI exp SEMI RP stmt
      {
        $$ = newStmtNode(ForK);
        $$->child[0] = $4;
                $$->child[1] = $7;
      }
      |FOR LP exp SEMI SEMI RP stmt
      {
        $$ = newStmtNode(ForK);
```

```
        $$->child[0] = $3;
46              $$->child[1] = $7;
      }
48    |FOR LP SEMI SEMI RP stmt
      {
50      $$ = newStmtNode(ForK);
        $$->child[0] = $6;
52    }
      ;
```

for 语句