实验四、手工编写递归下降预测分析程序

19335019 陈泷

实验步骤 4.1、 设计 Oberon-0 语言的翻译模式

首先改造文法，使其适用于自顶向下的分析方法。这样的改造内容是消除二义性、消除左递归、提取左因子。之后根据改造后的文法设计其SDT。设计后的翻译模式如下。
说明：`{}` 内为语义动作；empty指代 ϵ；为了增强可读性，SDT中对每个`" "`内的终结符的识别动作和左右括号数目匹配的语义动作进行了省略，但在代码中仍会进行实现。

```
module -> "MODULE" identifier1 ";" declaration module_begin "END"
identifier2 "." {if(identifier1 !=                identifier2) throw
moduleNameMismatched;}
        | "MODULE" identifier1 ";" declaration "END" identifier2 "."
{if(identifier1 != identifier2)                throw moduleNameMismatched;}
module_begin -> "BEGIN" statement_sequence

declaration -> const_declare  type_declare  var_declare
 procedure_declaration
const_declare -> empty | "CONST" const_list
const_list -> empty | identifier "=" expression ";" const_list
type_declare -> empty | "TYPE" type_list
type_list -> empty | identifier "=" type_kind ";" type_list
var_declare -> empty | "VAR" var_list
var_list -> empty | identifier_list ":" type_kind ";" var_list

procedure_declaration -> empty | procedure_heading ";" procedure_body ";"
procedure_declaration
procedure_body -> declaration procedure_begin "END" identifier
procedure_begin -> empty | "BEGIN" statement_sequence
procedure_heading -> "PROCEDURE" identifier formal_parameters

formal_parameters -> empty | "(" fp_section ")"
fp_section -> empty | var_or_empty identifier_list ":" type_kind |
var_or_empty identifier_list ":" type_kind ";" fp_section
var_or_empty -> "VAR" | empty
identifier_list -> identifier {identifier_list += identifier} | identifier
"," identifier_list1 {identifier_list1 = identifier_list + identifier}

type_kind -> identifier| array_type | record_type | "INTEGER" | "BOOLEAN"
array_type -> "ARRAY" expression "OF" type_kind {array_type.type = ARRAY}
record_type -> "RECORD" field_list "END"

field_list -> field_one ";" field_list | field_one
field_one -> empty | identifier_list ":" type_kind

statement_sequence -> statement | statement ";" statement_sequence
statement -> empty | assignment | procedure_call | if_statement |
while_statement | rw_statement

assignment -> identifier selector ":=" expression
```

```
            { if (selector.type == null) {
                    if (identifier.type != expression.type) throw mistype;
              } else {
                    if (selector.type != expression.type) throw mistype;
              }
            }

procedure_call -> identifier actual_parameters
actual_parameters -> empty | "(" ")" | "(" expression_list ")"
expression_list -> expression {expression_list += expreesion.type}
                | expression "," expression_list1 {expression_list1 =
expression_list + expression.type}

if_statement -> "IF" expression "THEN" statement_sequence elsif_statement
else_statement
                {if(expression.type != BOOLEAN) throw TypeMismatched}
elsif_statement -> empty | "ELSIF" expression "THEN" statement_sequence
elsif_statement
else_statement -> empty | "ELSE" statement_sequence "END"

while_statement -> "WHILE" expression "DO" statement_sequence "END"
                    {if(expression.type != BOOLEAN) throw TypeMismatched}

rw_statement -> "READ" "LPAREN" identifier "RPAREN" | "WRITE" "LPAREN"
identifier "RPAREN" | "WRITELN" "LPAREN" identifier "RPAREN" | "WRITELN"
"LPAREN" "LPAREN"

expression -> simple_expression1 re_op simple_expression2
             { if (simple_expression1.type != INTEGER ||
simple_expression2.type !=INTEER)
                    throw mistype;}
           | simple_expression
re_op -> "=" | "#" | "<" | "<=" | ">" | ">="
simple_expression -> term_head term | term_head term low_op
simple_expression
term_head -> "+" | "-" | empty
low_op -> "+" | "-" | "OR"
high_op -> "*" | "DIV" | "MOD" | "&"
term -> factor | factor high_op term

factor -> identifier selector {factor.type = selector.type}
        | NUMBER {factor.type = INTEGER}
        | "(" expression ")" {factor.type = expression.type}
        | "~" factor1 {if (factor1.type != BOOLEAN) throw mistype; else
factor.type =BOOLEAN;}
```

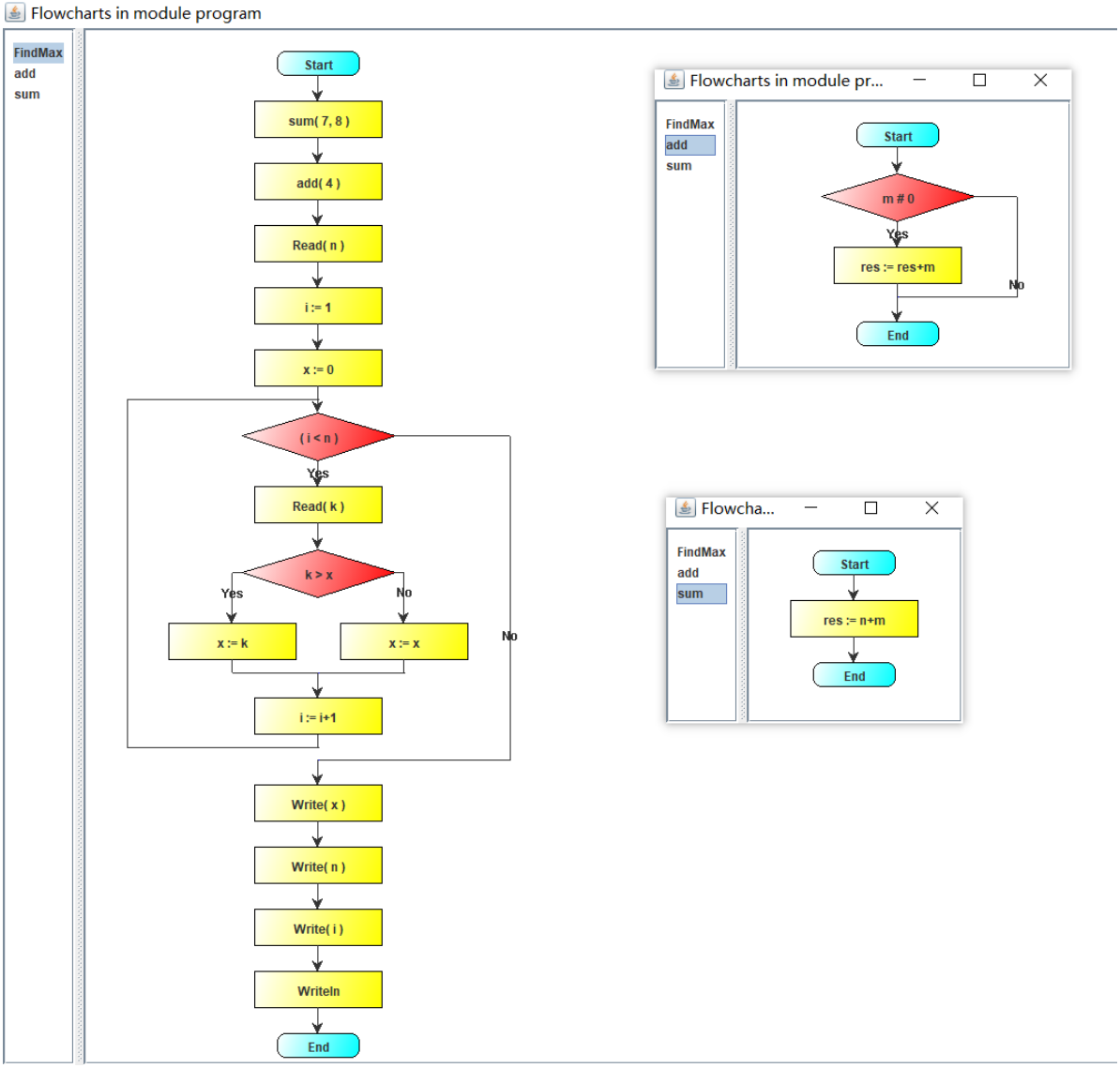
实验步骤 4.2、 编写递归下降预测分析程序

4.2.1 在正确代码上运行

根据4.1的翻译模式编写递归下降预测分析程序，在oberon-0的源代码上运行得到如下结果：

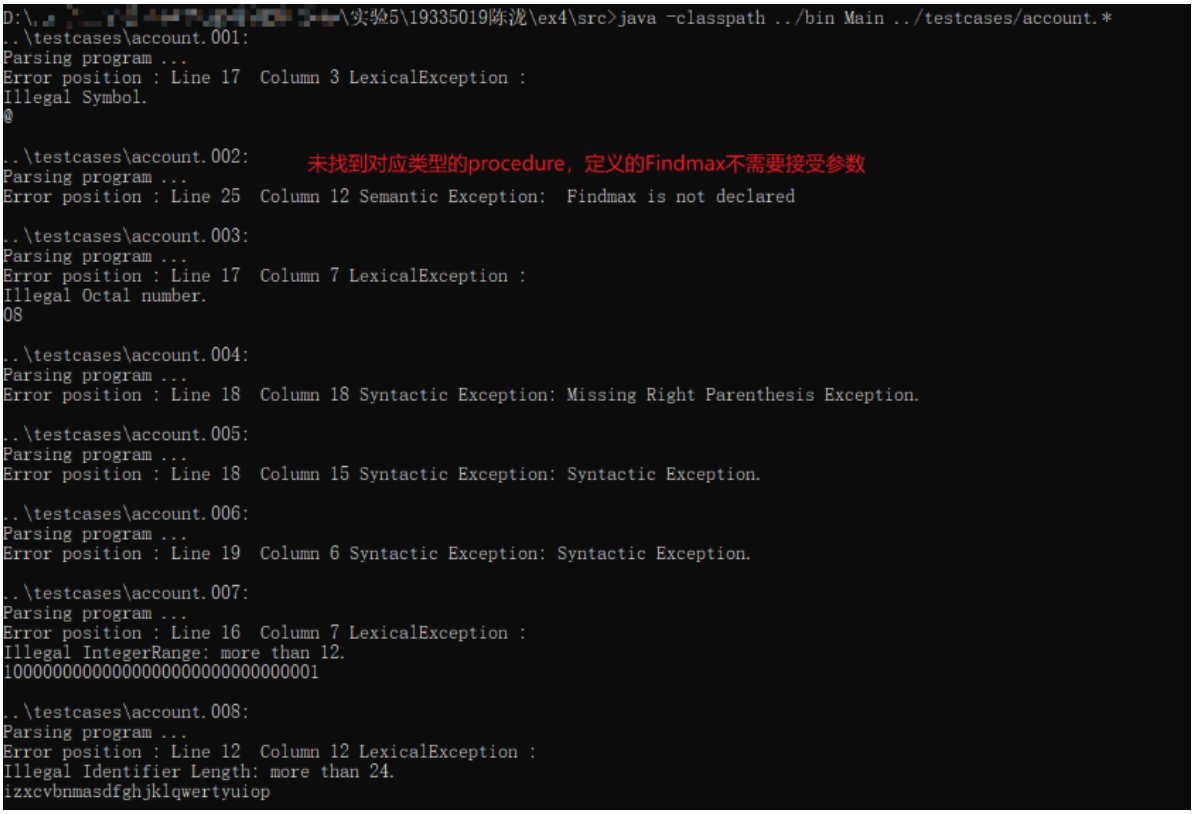
```
D:\[redacted]\实验5\19335019陈泷\ex4\src>java -classpath ../bin Main ../testcases/account.obr
../testcases/account.obr:
Parsing program ...
-- Parse Finished. No error found.
```

得到的流程图如下：



4.2.2 在错误的的代码上运行

运行结果：



各个文件对应的错误类型：

1. Illegal Symbol Exception：识别单词时遇到不合法的输入符号(譬如@、 $等符号)。
   eg.`sal$ary: INTEGER`
2. Semantic Exception：调用的函数的实际参数数目与过程声明的形式参数数目不一致。
   line 32和33之间添加：

```
PROCEDURE call();
    BEGIN
        Findmax(5);
    END call;
```

3. Illegal Octal Exception: 0 开头的整数常量中含有 0~7 之外的符号(包括 8 和 9或其他字母)。
   eg.`i := 08;`
4. Missing Right Parenthesis Exception：当左右圆括号不匹配、且缺少右括号时则抛出该异常。
   eg.`if (accounts[i].salary >= x THEN`
5. Missing Left Parenthesis Exception：当左右圆括号不匹配、且缺少左括号时则抛出该异常。
   eg.`if accounts[i].salary >= x) THEN`
6. Syntactic Exception：表达式、赋值语句、或参数传递等构造中出现类型不兼容错误。
   eg.`x := ~x`
7. 含有不合法的常量：常量中数字的个数超出限制
   eg.`i := 100000000000000000000000000001;`
8. 标识符长度不允许超过 24 个字符
   eg.`VAR x, n, izxcvbnmasdfghjklqwert: INTEGER;`

实验步骤 4.3、 语法分析讨论：自顶向下 vs. 自底向上

> 对递归下降预测分析技术和自底向上的 LR 分析技术这两种不同的分析策略进行比较

1. 技术的简单性：
    - 递归下降预测分析技术的实现本质上就是根据SDT进行递归调用，逐层下降进行分析，因此技术实现上更为简单，对于出现的问题，直接对应到SDT中进行分析，也更易于调试。
    - 相比之下，LR 分析技术更为复杂一些。因为在实现的过程中，相比递归下降预测分析技术，LR分析技术还要需要考虑到栈的内容等多种信息。对于出现的问题，调试的过程也因此变得更加复杂一些。
2. 技术的通用性：

    递归下降预测分析技术能处理的语言范围比LR分析技术小，技术通用性相比LR分析技术而言也更加有限。因为递归下降预测分析技术分析的文法属性只能是继承属性；而LR分析技术既能处理综合属性，也能出来继承属性。
3. 表达语义动作以完成语法制导翻译：
    - 递归下降预测分析技术表达语义动作时需要将继承属性作为参数传递给它调用的函数，即下一次的函数中。
    - 自底向上的 LR 分析技术表达语义动作时需要将需要使用的综合属性作为函数返回值返回给上一层的函数。
4. 出错恢复：
    - 递归下降预测分析技术在出错时，一般采取恐慌模式进行出错恢复，这一过程中会忽略某些输入，直到输入中出现由设计者选定的同步词法单元集合中的某个词法单元。恢复效果主要取决于同步集合的选取，因此
    - 自底向上的 LR 分析技术在出错时，一般采取恐慌模式错误恢复或者短语层次错误恢复。如果采用的是恐慌模式，虽然也是丢弃若干个输入符号，但是相比递归下降预测分析技术，少了同步词法单元集合的设计，因此更为简单些。
    - 另外，在速度方面，自底向上的 LR 分析技术往往能够更早地发现错误。
5. 以表格驱动方式取代递归程序实现，分析表大小的优劣比较：
    - 递归下降预测分析技术转换为表格驱动方式时，表格一般会更小。因为只需要分析各个非终结符遇到各个终结符时应该采用对应的表达式或动作。假设终结符的数量为 t ，非终结符的数量为 n，则表格的大小为 $t \times (n+1)$ 。
    - 自底向上的 LR 分析技术的表格一般会更大。因为需要分析各个状态遇到各个输入符号时应该采用对应动作，本质上 LR 分析技术的表格包括ACTION表格和GOTO表格两个部分。假设终结符的数量为 t ，非终结符的数量为 n，状态数为 s ，则ACTION表格大小为 $s \times (t+1)$ ，GOTO表格大小为 $s \times (t+n)$，整个表格的大小两部分大小相加，简化后大小为 $s \times (t+n+1)$。
6. 分析速度
    - 递归下降预测分析技术分析速度一般会更快，因为在分析时针对lookahead不断进行derive或match，直到acc或error，操作更为简单。
    - 自底向上的 LR 分析技术分析速度一般会更慢，因为分析时会不断在多个状态间切换，再进行shift或者reduce，因此会执行更多的动作，耗时会更长一些。

实验心得与体会

实验四的工作量和实验三差不多。通过实验四，我熟悉了递归下降预测分析程序的实现，对于自顶向下的语法分析过程以及流程图的生成有了更深入的理解与思考。另外，通过对自顶向下与自底向上的语法分析过程的比较，也加深了对语法分析和语义分析的认识与辩证性思考。总之，在这一过程中有一定的收获。