

实验报告：语法分析器的设计与实现

王贤义

计算机基地班

320210931221

实验目的

理解语法分析器的设计与实现： 掌握语法分析器的设计和实现原理。学习如何设计一个能够处理特定语言的语法分析器，并实现对源程序的语法分析，输出相应的语法树或抽象语法树。

掌握递归下降分析方法： 通过这个实验学习如何应用递归下降分析方法来构建语法分析器。这包括处理源程序的各种语法结构，并在遇到错误时进行适当的错误处理。

语法规则

```
PROG      -> {  DECLS  STMTS  }
DECLS     -> DECLS DECL | $
DECL      -> int  NAMES ; | bool NAMES ;
NAMES     -> NAMES , NAME | NAME
NAME      -> id
STMTS     -> STMTS STMT | STMT
STMT      -> id =  EXPR ; | id := BOOL ;
STMT      -> if id then STMT
STMT      -> if id then STMT else STMT
STMT      -> while id do STMT
STMT      -> { STMTS STMT }
STMT      -> read id ;
STMT      -> write id ;
EXPR      -> EXPR ADD TERM | TERM
ADD       -> + | -
TERM      -> TERM MUL NEGA | NEGA
MUL       -> * | /
NEGA      -> FACTOR | - FACTOR
FACTOR    -> ( EXPR ) | id | number
BOOL      -> BOOL || JOIN | JOIN
JOIN      -> JOIN && NOT | NOT
NOT       -> REL | ! REL
REL       -> EXPR ROP EXPR
ROP       -> > | >= | < | <= | == | !=
```

通过分析显然此语法存在不可避免的移进规约冲突（if else），本语法不属于SLR语法，但通过强制规定else与最近的if连接，当if id then STMT • 遇见else时，强制移进>规约。

设计约束

（1）从技术路线上说，建议在总体层面遵循递归下降分析方法。在处理表达式结构时，可以复用本章前趋任务2.1和2.2的工作成果，用算符优先分析方法来处理算术表达式和布尔表达式。处理语句结构时，可以复用本章前趋任务2.3和2.4的工作成果。实现时可以借助子程序之间的过程调用机制完成各个模块之间的功能协作，用全局数据结构（公共数据区）或子程序之间的数据传递（参数/返回值）来完成

成各个模块之间的数据传递。当然，完全使用递归下降方法或者LL（1）分析、SLR（1）分析来做也是很好的。

（2）从处理过程上来说，整个程序对输入的源程序应该从左到右处理一遍且仅此一遍。

（3）从结构上说，该程序应该至少包含3个模块：驱动模块、词法分析模块和语法分析模块。驱动模块包含了程序的入口和出口，主要负责输入、输出处理并调用另外2个工作模块；词法分析模块可以复用主线任务一中的工作成果；语法分析模块的主要任务是：把词法模块送来的单词符号串拼接成一个语法单位，再用树结构表达出来，同时检查是否存在语法错误。

（4）在分析时，可以把本次任务划分为若干个子任务，其中有些子任务可以进一步细分为若干个子任务.....在设计时，相应地可以把整个程序分解成若干个子模块及其子子模块。整个过程可以参照结构化程序设计的思路来做。如果用C语言来实现，程序可能就由若干个.c和若干个.h组成；若用OOP，比如说Java，程序可能就由若干个.class组成。

程序设计与实现

工作流程

1. **识别语法**：通过读取grammar.txt中的语法规则构造LR项目族，action_goto表等。
2. **执行递归下降语法分析**：通过递归下降一遍扫描根据action_goto表执行移进、规约等操作。
3. **生成完整语法树**：在上面移进、规约的过程中插入生成树结点并连接，最后图形化。

关键算法描述

- 主程序SyntacticAnalyzer()完成语法识别工作，sa.StartAnalyze()完成一遍扫描，并根据移进或规约生成语法树，sa.VisualizeTree()将语法树进行输出。

```
if __name__ == "__main__":
    file_name = "./sourceProgram/sourceProgram2.txt"
    sa = SyntacticAnalyzer()
    if sa.StartAnalyze(file_name):
        sa.VisualizeTree(
            sa.tree_node_stack_[0],
            "./treeOutput/treeOutput2.txt"
        )
```

- 识别语法，分别为从txt中生成文法产生式、生成拓广文法，生成非终结符First集、Follow集、生成文法符号集、生成LR项目和生成项目集规范族。

```
def build_grammar(self):
    self.GenProduction()
    self.GenAugmentedGrammar()
    self.GenFirstSet()
    self.GenFollowSet()
    self.GenGrammarSymbolSet()
    self.GenLrItems()
    return self.GenNormalFamilySet()
```

- 递归下降语法分析中循环从词法分析器中读入一个词，并根据上一步中生成的action_goto表决定移进、规约和报错等

```
while True:
    get_word = lexical_analyzer.Getword()
```

顶

```
word_string = get_word.word_string
if get_word.type == LEXICAL_TYPE.LUNKNOWN: # 错误处理
    print("词法分析器过程中, 发生unknown错误!")
    print(get_word.value)
while True:
    current_state = self.state_sequence_stack[-1] # 选择状态序列栈栈顶

    if (
        current_state,
        word_string,
    ) not in self.action_goto_tables_: # action_goto表中不存在对应的操作, 语法分析过程中出现错误, 报告错误并返回
        print("语法分析器过程中, 发生错误!")
        print(get_word.value)

        print(
            "state: ",
            current_state,
            " 与 ",
            word_string,
            " 在action_goto_table 中不含对应操作!",
        )
        return False

    if (
        self.action_goto_tables_[(current_state, word_string)].op
        == SLR_OPERATIONS.MOVE
    ): # 移进操作
        self.state_sequence_stack.append(
            self.action_goto_tables_[(current_state,
word_string)].state
        )
        self.move_conclude_string_stack.append(word_string)
        self.PrintAnalysisProcess(
            syntactic_step,
            self.action_goto_tables_[(current_state, word_string)],
            get_word
        )
        syntactic_step += 1
        self.grammar_symbol_info_stack.append(
            {get_word.word_string, get_word.value}
        ) # 文法符号信息压入
        break

    elif (
        self.action_goto_tables_[(current_state, word_string)].op
        == SLR_OPERATIONS.CONCLUDE
    ): # 规约操作
        conclude_production_number = self.action_goto_tables_[(current_state, word_string)].state
        if self productions_[conclude_production_number].right[0] == "$":
            production_length = 0
        else:
            production_length = len(
                self productions_[conclude_production_number].right
```

```

    )

    for i in range(
        production_length
    ): # 将两个栈都弹出production_length个元素
        self.state_sequence_stack_.pop()
        self.move_conclude_string_stack_.pop()

    self.move_conclude_string_stack_.append(
        self productions_[conclude_production_number].left
    ) # 用于规约的产生式的左部 压入栈中
    if (
        self.state_sequence_stack_[ -1 ],
        self productions_[conclude_production_number].left,
    ) not in self.action_goto_tables_: # 不存在goto
        print("语法分析器算法发生致命错误CONCLUDE中")
        print(get_word.value)
        return False

    self.state_sequence_stack_.append(
        self.action_goto_tables_[
            (
                self.state_sequence_stack_[ -1 ],
                self productions_[conclude_production_number].left,
            )
        ].state
    ) # goto对应的状态压入

elif (
    self.action_goto_tables_[(current_state, word_string)].op
    == SLR_OPERATIONS.ACCEPT
): # 宣布接受
    self.PrintAnalysisProcess(
        sytactic_step,
        self.action_goto_tables_[(current_state, word_string)],
        get_word
    )
    sytactic_step += 1

    print("语法分析正确完成！")
    return True

else: # 语法分析器算法错误
    print("致命错误！")
    print("语法分析器算法存在错误，请检查！")
    return False

self.PrintAnalysisProcess(
    sytactic_step,
    self.action_goto_tables_[(current_state, word_string)],
    get_word
)
sytactic_step += 1

return True

```

- 生成完整语法树结点并连接，位于PrintAnalysisProcess函数中，移进时生成一个结点放入临时栈中，规约时从栈中取出若干结点作为子节点，生成规约项目的父节点并连接。

```
# 输出操作
    if sl_op.op == SLR_OPERATIONS.MOVE:
        self.syntactic_analyzer_printer_.write("移进")
        if self.move_conclude_string_stack[-1]=="id" or
self.move_conclude_string_stack[-1]=="number":
            self.tree_node_stack_.append(TreeNode(get_word.value))
        else:

self.tree_node_stack_.append(TreeNode(self.move_conclude_string_stack[-1]))
    elif sl_op.op == SLR_OPERATIONS.ACCEPT:
        self.syntactic_analyzer_printer_.write("接受")
    elif sl_op.op == SLR_OPERATIONS.CONCLUDE:
        if self.productions_[sl_op.state].right[0] == "$":
            self.tree_node_stack_.append(TreeNode("$"))
        conclude_tree_node_ = TreeNode(self.productions_[sl_op.state].left)
        conclude_tree_node_.child = self.tree_node_stack_[
            -len(self.productions_[sl_op.state].right) :
        ]
        self.tree_node_stack_ = self.tree_node_stack_[
            : -len(self.productions_[sl_op.state].right)
        ]
        self.tree_node_stack_.append(conclude_tree_node_)
        self.syntactic_analyzer_printer_.write(
            "规约: " + self.productions_[sl_op.state].left + "->"
        )
        for symbol in self.productions_[sl_op.state].right:
            if symbol == ",":
                self.syntactic_analyzer_printer_.write(", ")
            else:
                self.syntactic_analyzer_printer_.write(symbol + " ")
```

数据结构设计

- **SLR_OPERATIONS**: 枚举，用于存储移进，规约和接受等操作类型。
- **TreeNode**: 类，用于存储树结点的值和子节点。
- **LrItem**: 类，存储LR项目，包括涉及的产生式和其中•的位置信息。
- **first_map, follow_map**: 字典，用于存储非终结符的first集和follow集。
- **state_sequence_stack**: 列表，状态栈。
- **move_conclude_string_stack**: 列表，符号栈。
- **tree_node_stack**: 列表，树结点栈。
- **normal_family**: 列表，项目集规范族。
- **action_goto_tables**: 集合，存储（状态，符号）对应的操作。

程序结构

```
class SLR_OPERATIONS(Enum): ...
class TreeNode: ...
class LrItem: ...
class SlrOperation: ...

class SyntacticAnalyzer:
    def __init__(self, show_detail=True):
```

```

def IsNonTerminalSymbol(self, symbol):
def GetProductionFirstSet(self, symbol_string):
def GenProduction(self):
def GenAugmentedGrammar(self):
def GenFirstSet(self):
def GenFollowSet(self):
def GenGrammarSymbolSet(self):
def GenLrItems(self):
def GenItemClosureSet(self, input_item):
def GenItemsClosureSet(self, items):
def GenNormalFamilySet(self):
def build_grammar(self):
def PrintAnalysisProcess(self, step, sl_op ,get_word):
def StartAnalyze(self, code_filename):
...
if __name__ == "__main__":
    file_name = "./sourceProgram/sourceProgram.txt"
    sa = SyntacticAnalyzer()
    if sa.StartAnalyze(file_name):
        sa.VisualizeTree(
            sa.tree_node_stack_[0],
            "./treeOutput/treeOutput.txt"
        )

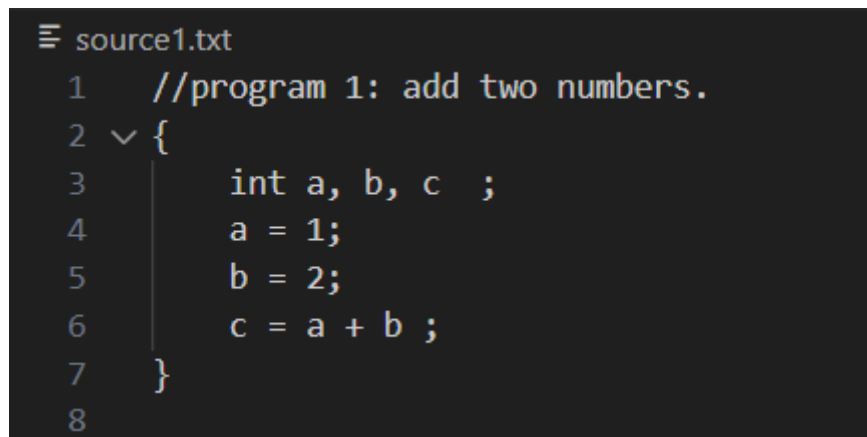
```

测试用例及结果

程序经过6组测试用例进行了充分测试，测试结果显示当程序符合语法规则时会输出完整的竖向语法分析树，当程序不符合语法规则时会抛出导致报错的位置和对应单词。

测试1

输入 sourceProgram1.txt:



```

≡ source1.txt
1  //program 1: add two numbers.
2  {
3      int a, b, c ;
4      a = 1;
5      b = 2;
6      c = a + b ;
7  }
8

```

输出 treeOutput1.txt:

```

1  PROG
2  |--{
3  |--DECLS
4  |   |--DECLS
5  |   |   \--$
6  |   \--DECL
7  |       |--int
8  |       |--NAMES
9  |           |--NAMES
10 |           |   |--NAMES
11 |           |   |   \--NAME
12 |           |   |   |   \--a
13 |           |   |   |   |--,
14 |           |   |   |   \--NAME
15 |           |   |   |   |   \--b
16 |           |   |   |   |   |--,
17 |           |   |   |   |   \--NAME
18 |           |   |   |   |   |   \--c
19 |           |   |   |   |   |   \--;
20 |--STMTS
21 |   |--STMTS
22 |   |   |--STMTS
23 |   |   |   \--STMT
24 |   |   |   |   |--a
25 |   |   |   |   |   |--=
26 |   |   |   |   |   |--EXPR
27 |   |   |   |   |   |   \--TERM
28 |   |   |   |   |   |   |   \--NEGA
29 |   |   |   |   |   |   |   |   \--FACTOR
30 |   |   |   |   |   |   |   |   |   \--1
31 |   |   |   |   |   |   |   |   |   \--;
32 |   |   |   \--STMT
33 |   |   |   |   |--b
34 |   |   |   |   |   |--=
35 |   |   |   |   |   |--EXPR
36 |   |   |   |   |   |   \--TERM
37 |   |   |   |   |   |   |   \--NEGA
38 |   |   |   |   |   |   |   |   \--FACTOR
39 |   |   |   |   |   |   |   |   |   \--2
40 |   |   |   |   |   |   |   |   |   \--;
41 |   |   \--STMT
42 |   |   |   |--c
43 |   |   |   |   |--=
44 |   |   |   |   |   |--EXPR

```

```

15 | --EXPR
16 |   \--TERM
17 |     \--NEGA
18 |       \--FACTOR
19 |         \--a
20 |
21 |   \--ADD
22 |     \--+
23 |       \--TERM
24 |         \--NEGA
25 |           \--FACTOR
26 |             \--b
27 |
28 |   \--;
29 |
30 | \--}

```

测试2

输入 sourceProgram2.txt:

```

≡ source2.txt
1  //program 2: do some calculation.
2  {
3      int a, b, c ;
4      a = 5;      //positive number
5      b = -3;     //negative number
6      c = (a+b)*(a-b); //calculation
7      write c ;  //output:16.
8  }
9

```

输出 treeOutput2.txt:

PROG

```

|--{
|--DECLS
|  |--DECLS
|  |  \--$
|  \--DECL
|    |--int
|    |--NAMES
|    |  |--NAMES
|    |  |  |--NAMES
|    |  |  |  \--NAME
|    |  |  |  |  \--a
|    |  |  |
|    |  |  |--,
|    |  |  |  \--NAME
|    |  |  |  |  \--b
|    |  |  |
|    |  |  |--,
|    |  |  |  \--NAME
|    |  |  |  |  \--c
|    |  |
|    |  \--;
|--STMTS
|  |--STMTS
|  |  |--STMTS
|  |  |  |--STMTS
|  |  |  |  \--STMT
|  |  |  |  |  |--a
|  |  |  |  |  |--=
|  |  |  |  |  |--EXPR
|  |  |  |  |  |  \--TERM
|  |  |  |  |  |  |  \--NEGA
|  |  |  |  |  |  |  |  \--FACTOR
|  |  |  |  |  |  |  |  |  \--5
|  |  |  |  |
|  |  |  |  |  \--;
|  |  |  |  \--STMT
|  |  |  |  |  |--b
|  |  |  |  |  |--=
|  |  |  |  |  |--EXPR
|  |  |  |  |  |  \--TERM
|  |  |  |  |  |  |  \--NEGA
|  |  |  |  |  |  |  |  |--
|  |  |  |  |  |  |  |  |  \--FACTOR
|  |  |  |  |  |  |  |  |  |  \--3
|  |  |  |  |
|  |  |  |  |  \--;
|  |  |  |  \--STMT
|  |  |  |  |  |--c

```

```

|--=
|--EXPR
  \--TERM
    |--TERM
      \--NEGA
        \--FACTOR
          |--(
            |--EXPR
              |--EXPR
                \--TERM
                  \--NEGA
                    \--FACTOR
                      \--a
            |--ADD
              \--+
            \--TERM
              \--NEGA
                \--FACTOR
                  \--b
          \--)
        |--MUL
          \--*
        \--NEGA
          \--FACTOR
            |--(
              |--EXPR
                |--EXPR
                  \--TERM
                    \--NEGA
                      \--FACTOR
                        \--a
              |--ADD
                \---
              \--TERM
                \--NEGA
                  \--FACTOR
                    \--b
            \--)
          \--;
        \--STMT
          |--write
          |--c
          \--;
      \--}

```

测试3

输入 sourceProgram3.txt:

```
≡ source3.txt
1  /*program 3: add numbers from 1 to 100
2  *and print the result.
3  */
4  {
5      int a , sum ;
6      bool b ;
7      a = 1 ;
8      sum = 0 ;
9      b := a <= 100 ;
10     while b do
11     {
12         sum = sum + a ;
13         a = a + 1 ;
14         b := a <= 100 ;
15     }
16     write sum ;
17 }
```

输出 treeOutput3.txt:

PROG

```

|--{
|--DECLS
|  |--DECLS
|  |  |--DECLS
|  |  |  |--$
|  |  |--DECL
|  |  |  |--int
|  |  |  |--NAMES
|  |  |  |  |--NAMES
|  |  |  |  |  |--NAME
|  |  |  |  |  |  |--a
|  |  |  |  |--,
|  |  |  |--NAME
|  |  |  |  |--sum
|  |  |--;
|--DECL
|  |--bool
|  |--NAMES
|  |  |--NAME
|  |  |  |--b
|--;
|--STMTS
|  |--STMTS
|  |  |--STMTS
|  |  |  |--STMTS
|  |  |  |  |--STMT
|  |  |  |  |  |--a
|  |  |  |  |  |--=
|  |  |  |  |  |--EXPR
|  |  |  |  |  |  |--TERM
|  |  |  |  |  |  |  |--NEGA
|  |  |  |  |  |  |  |--FACTOR
|  |  |  |  |  |  |  |  |--1
|  |  |  |  |--;
|  |  |--STMT
|  |  |  |--sum
|  |  |  |--=
|  |  |  |--EXPR
|  |  |  |  |--TERM
|  |  |  |  |  |--NEGA
|  |  |  |  |  |--FACTOR
|  |  |  |  |  |  |--0

```

```

|--;
\--STMT
|--b
|--:=
|--BOOL
|--JOIN
|--NOT
|--REL
|--EXPR
|--TERM
|--NEGA
|--FACTOR
|--true
|--ROP
|-->
\--EXPR
|--EXPR
|--TERM
|--NEGA
|--FACTOR
|--1
|--ADD
|--+
\--TERM
|--NEGA
|--FACTOR
|--1
|--;
\--STMT
|--while
|--b
|--do
\--STMT
|--{
|--STMTS
|--STMTS
\--STMT
|--sum
|--=
|--EXPR
|--EXPR
|--TERM
|--NEGA
|--FACTOR

```

```

    |--sum
    |--ADD
    |--++
    |--TERM
    |--NEGA
    |--FACTOR
    |--a
    |--;
|--STMT
|--a
|--=
|--EXPR
|--EXPR
|--TERM
|--NEGA
|--FACTOR
|--a
|--ADD
|--++
|--TERM
|--NEGA
|--FACTOR
|--1
|--;
|--STMT
|--b
|--:=
|--BOOL
|--JOIN
|--NOT
|--REL
|--EXPR
|--TERM
|--NEGA
|--FACTOR
|--a
|--ROP
|--<=
|--EXPR
|--TERM
|--NEGA
|--FACTOR
|--100
|--;

```

```
    \--}
    \--STMT
    |--write
    |--sum
    \--;
\--}
```

测试4

输入 sourceProgram4.txt:

```
1  //program 4: input 3 numbers, find the largest
2  //one, and output it .
3  {
4      int a,b,c;
5      int lg;
6      bool cond;
7
8      read  a;   read  b;   read  c;
9
10     cond := a > b  ;
11     if  cond  then  lg = a  ;
12     else  lg = b  ;
13
14     cond := lg < c  ;
15     if  cond  then  lg = c  ;
16
17     write lg  ;
18 }
```

输出 treeOutput4.txt:

```

|--{
|--DECLS
|  |--DECLS
|    |--DECLS
|      |--DECLS
|        |--$
|      \--DECL
|        |--int
|        |--NAMES
|          |--NAMES
|            |--NAMES
|              |--NAME
|                |--a
|              |--,
|                \--NAME
|                  |--b
|              |--,
|                \--NAME
|                  |--c
|            \--;
|          \--DECL
|            |--int
|            |--NAMES
|              |--NAME
|                |--lg
|            \--;
|          \--DECL
|            |--bool
|            |--NAMES
|              |--NAME
|                |--cond
|            \--;
|--STMTS
|  |--STMTS
|    |--STMTS
|      |--STMTS
|        |--STMTS
|          |--STMTS
|            |--STMTS
|              |--STMT
|                |--read
|                |--a

```



```

    |--;
    |--STMT
    |--read
    |--b
    |--;
    |--STMT
    |--read
    |--c
    |--;
    |--STMT
    |--cond
    |--:=
    |--BOOL
    |--JOIN
    |--NOT
    |--REL
    |--EXPR
    |--TERM
    |--NEGA
    |--FACTOR
    |--a
    |--ROP
    |-->
    |--EXPR
    |--TERM
    |--NEGA
    |--FACTOR
    |--b
    |--;
    |--STMT
    |--if
    |--cond
    |--then
    |--STMT
    |--lg
    |--=
    |--EXPR
    |--TERM
    |--NEGA
    |--FACTOR
    |--a
    |--;
    |--else
    |--STMT

```

```

|--lg
|--=
|--EXPR
  |--TERM
    |--NEGA
      |--FACTOR
        |--b
      |--;
    |--STMT
      |--cond
      |--:=
      |--BOOL
        |--JOIN
          |--NOT
            |--REL
              |--EXPR
                |--TERM
                  |--NEGA
                    |--FACTOR
                      |--lg
              |--ROP
                |--<
          |--EXPR
            |--TERM
              |--NEGA
                |--FACTOR
                  |--c
            |--;
          |--STMT
            |--if
            |--cond
            |--then
          |--STMT
            |--lg
            |--=
            |--EXPR
              |--TERM
                |--NEGA
                  |--FACTOR
                    |--c
              |--;
            |--STMT
              |--write
              |--lg

```

```

| | | \--;
| \--}

```

测试5

错误测试，空程序不符合语法规则，语法分析器会抛出报错单词。

輸入 sourceProgram9.txt:

```
//program 9: empty program.
{
}
```

输出报错：

```
语法分析器过程中, 发生错误!  
}      行号: 4  
state: 1 与 } 在action_goto_table 中不含对应操作!
```

测试6

错误测试，根据语法规则有"&&"的左右两边都必须接表达式非单独的id，例如本例中第九行的flag。

输入 sourceProgram10.txt:

```
//program10: test prime numbers.
{
    int number,i,j;
    bool cond,flag;
    read number;

    i = 2;
    flag = true;
    cond := i<number && flag;

    while cond do
    {
        j=number-(number/i)*i;
        flag:= j!=0;
        i=i+1;
        cond := i<number && flag;
    }
    if flag then write number;
}
```

输出报错：

语法分析器过程中, 发生错误!

; 行号: 9

state: 36 与 ; 在action goto table 中不含对应操作!