装

订

线

实验报告名称：语法分析器设计

班级：15 信息安全

姓名：端启航

学号：1552212

完成日期：2017.12.2

目录

装 订 线

1. 实验目的及要求

1．根据算符优先分析算法，编写一个语法分析程序，可以选择以下三项之一作为分析算法中的输入：

（1）直接输入根据已知文法人工构造的算法优先关系矩阵。

（2）输入已知文法和FIRST、LAST集合，由程序自动生成该文法的算法优先关系矩阵。

（3）输入已知文法，由程序自动生成该文法的算法优先关系矩阵。（完成项）

2．程序具有通用性，即所编制的语法分析程序能够适用于不同文法以及各种输入单词串，并能判断该文法是否为算符文法和算符优先文法。

3．对输入的一个文法和一个单词串，程序能正确判断此单词串是否为该文法的句子，并要求输出分析过程和语法树。

2. 实验原理

2.1. 算符文法

一个文法，如果它的任一产生式右部都不含两个相继（并列）的非终结符，即不含如下形式的产生式右部：

$$\cdots QR \cdots , \quad Q, R \in V_N$$

则称该文法为算符文法。

2.2. 算符优先文法

如果一个算符文法 G 中的任何终结符对(a, b)至多满足下述关系之一

$$a = b,\ a < b,\ a > b$$

则称 G 为算符优先文法

2.3. FIRSTVT(P)和LASTVT(P)

2.3.1. 定义

设 P∈VN ，定义 ：

FIRSTVT (P) =

{ a | P⇒a … 或 P⇒Qa … , a ∈VT , Q ∈VN }

LASTVT (P) =

{ a | P⇒… a 或 P ⇒… aQ , a ∈VT , Q∈VN }

2.3.2. FIRSTVT(P)和LASTVT(P) 构造

FIRSTVT (P) 构造
规则 1: 若 P→a … 或 P→Qa … , 则 a ∈FIRSTVT(P);
规则 2: 若 a ∈FIRSTVT(Q) , 且 P→Q … , 则
a ∈FIRSTVT(P)。
LASTVT (P) 构造
规则 1: 若 P→ … a 或 P→ … aQ , 则 a ∈LASTVT(P);
规则 2: 若 a ∈LASTVT(Q) , 且 P→ … Q, 则
a ∈LASTVT(P)。

2.3.3. FIRSTVT(P)和LASTVT(P) 构造算法

二维布尔矩阵 F[P,a]和符号栈 STACK
.T. a ∈FIRSTVT(P)
布尔矩阵 F[P,a] =
.F. a FIRSTVT(P)
栈 STACK: 存放使 FIRSTVT 为真的符号对
(P, a).
把所有初值为真的数组元素 F[P, a]的符号对(P, a)全都放在 STACK 之中。
如果栈 STACK 不空，就将栈顶逐出，记此项为(Q, a)。对于每个形如
P→Q…
的产生式，若 F[P, a]为假，则变其值为真，且将(P, a)推进 STACK 栈。
上述过程必须一直重复，直至栈 STACK 拆空为止。

2.4. 构造优先关系表

```
FOR   每条产生式 P→X1X2…Xn   DO
      FOR   i:=1   TO   n-1 DO
        BEGIN
            IF   Xi 和 Xi+1 均为终结符   THEN   置 Xi = Xi+1
            IF   i   n-2 且 Xi 和 Xi+2 都为终结符
                   但 Xi+1 为非终结符   THEN   置 Xi = Xi+2；
            IF   Xi 为终结符而 Xi+1 为非终结符   THEN
            FOR   FIRSTVT(Xi+1)中的每个 a   DO
                       置  Xi < a；
            IF   Xi 为非终结符而 Xi+1 为终结符   THEN
                     FOR   LASTVT(Xi)中的每个 a     DO
                       置   a > Xi+1
    END
```

2.5. 最左素短语

素短语
是一个短语，它至少含有一个终结符且除它自身之外不含有任何更小的素短语。

最左素短语
处于句型最左边的那个素短语。
算符优先文法句型(括在两个＃之间)的一般形式为:
＃N1a1N2a2 … NnanNn+1＃
其中:ai ∈VT, Ni∈VN (可有可无）
一个算符优先文法 G 的任何句型的最左素短语是满足下列条件的最左子串 Njaj … Niai Ni+1

aj-1 < aj
aj = aj+1 = … = ai-1 = ai
ai > ai+1

2.6. 算符优先分析算法

1）将输入串依次逐个存入符号栈 S 中，直到符号栈顶元素 Sk 与下一个待输入的符号 a 有优先关系 Sk>a 为止；
2）至此，最左素短语尾符号 Sk 已在符号栈 S 的栈顶，由此往前在栈中找最左素短语的头符号 Sj+1，直到找到第一个 < 为止；
3）已找到最左素短语 Sj+1…Sk，将其归约为某个非终结符 N 及做相应的语义处理。

3. 数据结构设计

gramma_pair_t：一个非终结符的产生式，如 E->E+T|T
gramma_items_t：一个产生式的所有右部
gramma_item_t：一个产生式的一个右部
vt_set_t：FIRSTVT 和 LASTVT 的结构

```
typedef vector<char> gramma_item_t;
typedef vector<gramma_item_t> gramma_items_t;
typedef pair<char, gramma_items_t> gramma_pair_t;
typedef map<char, vector<char>> vt_set_t;
```

语法树的结构

```
class TreeNode
{
public:
    char value;
    vector<TreeNode*> children;

    TreeNode(const char ch) : value(ch)
    {}
};
```

4. 实验步骤

4.1. 输入文法

获取一行输入，如果以#开头则退出输入函数。

```
string input_temp;
getline(cin, input_temp);
if (input_temp.front() == '#')
    break;
```

判断文法中是否含->否则报错

```
if (input_temp.find("->") == string::npos)
    error("输入错误!");
```

对于->左边的字符输入到产生式的左边

```
stringstream ss(input_temp.substr(0, input_temp.find_first_of("->")));
gramma_pair_t gramma_pair;
ss >> gramma_pair.first;
```

对于->右边的字符输入到产生式的右边，如果连着两个非终结符则报错。

```
stringstream ss2(input_temp.substr(input_temp.find_first_of("->") + 2));
gramma_items_t gramma_items;
string temp;
while (ss2 >> temp)
{
    gramma_item_t gramma_item;
    for (auto v : temp)
    {
        if (!gramma_item.empty() && !TerminalSymbolQ(gramma_item.back()) && !TerminalSymbolQ
          (v))
            error("非算符文法!");
        gramma_item.push_back(v);
    }
    gramma_items.push_back(gramma_item);
}
gramma_pair.second = gramma_items;
gramma.push_back(gramma_pair);
```

装 订 线

4.2. FIRSTVT(P)和LASTVT(P)

4.2.1. 数据结构

二维布尔矩阵 F[P,a]和符号栈 STACK

```
map<char, map<char, bool>> Fmatrix;
stack<pair<char, char>> pair_stack;
```

4.2.2. FIRSTVT(P)和LASTVT(P) 构造

把所有初值为真的数组元素 F[P, a]的符号对(P, a)全都放在 STACK 之中。

```
for (auto gramma_pair : gramma)// 对于文法的每一项
    for (auto str : gramma_pair.second)// 对于每一个产生式
    {
        if (order == LASTVT)
            reverse(str.begin(), str.end());
        bool first_b = true;
        for (auto ch : str)
            if (TerminalSymbolQ(ch) && first_b)// terminal and first
            {
                first_b = false;
                Fmatrix[gramma_pair.first][ch] = true;
                pair_stack.push(pair<char, char>(gramma_pair.first, ch));
            }
            else
                Fmatrix[gramma_pair.first][ch] = false;
    }
```

如果栈 STACK 不空，就将栈顶逐出，记此项为(Q, a)。对于每个形如

P→Q…

的产生式，若 F[P, a]为假，则变其值为真，且将(P, a)推进 STACK 栈。

上述过程必须一直重复，直至栈 STACK 拆空为止。

```
while (!pair_stack.empty())
{
    pair<char, char> temp = pair_stack.top();
    pair_stack.pop();
    for (auto gramma_pair : gramma)
        for (auto str : gramma_pair.second)
        {
            if (order == LASTVT)
                reverse(str.begin(), str.end());
            if (str.front() == temp.first && temp.first != gramma_pair.first)
            {
                Fmatrix[gramma_pair.first][temp.second] = true;
                pair_stack.push(pair<char, char>(gramma_pair.first, temp.second));
            }
        }
}
```

4. 3. 构造优先关系表

FOR 每条产生式 P→X1X2…Xn DO

FOR i:=1 TO n-1 DO

```
for (auto gramma_pair : gramma)// 对于文法的每一项
    for (gramma_item_t str : gramma_pair.second)// 对于每一个产生式
    {
        if (TerminalSymbolQ(str.front()))
            all_vt.insert(str.front());
        if (str.size() == 1)
            continue;
```

BEGIN

IF Xi 和 Xi+1 均为终结符 THEN 置 Xi = Xi+1

```
//  IF  Xi和Xi+1均为终结符  THEN  置Xi = Xi+1
if (TerminalSymbolQ(*it) && TerminalSymbolQ(*(it + 1)))
{
    if (relationship_table.find(*it) != relationship_table.end() &&
        relationship_table[*it].find(*(it + 1)) != relationship_table[*it].end())
        error("非算符优先文法！");
    relationship_table[*it][*(it + 1)] = 0;
}
```

IF i n-2 且 Xi 和 Xi+2 都为终结符

但 Xi+1 为非终结符 THEN 置 Xi = Xi+2;

```
//  IF  i<n-2且Xi和Xi+2都为终结符
// 但Xi + 1为非终结符  THEN  置Xi = Xi + 2;
if (str.size() >= 3 && it <= str.end() - 3 && TerminalSymbolQ(*it) && !TerminalSymbolQ(
    *(it + 1)) && TerminalSymbolQ(*(it + 2)))
{
    if (relationship_table.find(*it) != relationship_table.end() &&
        relationship_table[*it].find(*(it + 2)) != relationship_table[*it].end())
        error("非算符优先文法！");
    relationship_table[*it][*(it + 2)] = 0;
}
```

IF Xi 为终结符而 Xi+1 为非终结符 THEN

FOR FIRSTVT(Xi+1)中的每个 a DO

置 Xi < a;

```
// IF  Xi为终结符而Xi+1为非终结符  THEN
// FOR  FIRSTVT(Xi + 1)中的每个a  DO
// 置 Xi < a;
if (TerminalSymbolQ(*it) && !TerminalSymbolQ(*(it + 1)))
    for (auto a : first_vt.at(*(it + 1)))
    {
        if (relationship_table.find(*it) != relationship_table.end() &&
            relationship_table[*it].find(a) != relationship_table[*it].end())
            error("非算符优先文法！");
        relationship_table[*it][a] = -1;
    }
```

IF Xi 为非终结符而 Xi+1 为终结符 THEN

FOR LASTVT(Xi)中的每个 a DO

置 a > Xi+1

```
// IF  Xi为非终结符而Xi+1为终结符  THEN
// FOR  LASTVT(Xi)中的每个a   DO
// 置  a > Xi + 1
if (!TerminalSymbolQ(*it) && TerminalSymbolQ(*(it + 1)))
    for (auto a : last_vt.at(*it))
    {
        if (relationship_table.find(a) != relationship_table.end() &&
            relationship_table[a].find(*(it + 1)) != relationship_table[a].end())
            error("非算符优先文法！");
        relationship_table[a][*(it + 1)] = 1;
    }
```

END

4.4. 最左素短语

由于输入字符串是倒置的，所以从最后一个字符开始。找到第一个终结符，它的前一个终结符小于它。

```
// cs是倒过来的
auto it = cs.end() - 1;
auto next_nt = it;
while (true)
{
    if (*it == '#')
        return it + 1;
    if (!TerminalSymbolQ(*it))
    {
        --it;
        continue;
    }
    next_nt = it - 1;
    while (!TerminalSymbolQ(*next_nt))
        --next_nt;
    if (relationship_table.at(*next_nt).at(*it) == -1)
        break;
    --it;
}
return next_nt + 1;
```

4.5. 算符优先分析算法

1） 找到栈顶的非终结符，和输入串对比，如果没有优先级就报错；

```
// char_stack是倒置的，end为栈顶
auto stack_top = char_stack.end();
while (!TerminalSymbolQ(*--stack_top));

if (relationship_table.at(*stack_top).find(input.back()) == relationship_table.at(*stack_top).end())
    error("非文法的符号或无优先级");
```

2） 如果栈顶非终结符不大于输入串，则移进

```
else if (relationship_table.at(*stack_top).at(input.back()) != 1)// 移进
{
    if (relationship_table.at(*stack_top).at(input.back()) == 0)
        cout << setw(24) << left << "=";
    else
        cout << setw(24) << left << "<";
    cout << setw(24) << left << "移进";

    char_stack.push_back(input.back());
    input.pop_back();
}
```

3） 如果栈顶非终结符大于输入串，则规约最左素短语

```
else// 规约
{
    cout << setw(24) << left << ">";
    cout << setw(24) << left << "规约";
    // 获得最左素短语的位置
    const auto it = FindLeftmostPrimePhrase(char_stack, relationship_table);
    // 提取最左素短语
    auto leftmost_prime_phrase = char_stack.substr(it - char_stack.begin());
    reverse(leftmost_prime_phrase.begin(), leftmost_prime_phrase.end());
    cout << leftmost_prime_phrase;
    char_stack.erase(it, char_stack.end());
    char_stack.push_back('N');
```

4） 同时生成语法树节点

装 订 线

```
// 构建语法树节点
TreeNode* root = new TreeNode('N');
for (auto ch : leftmost_prime_phrase)// 对于最左素短语的每一个字符
{
    if (ch != 'N')
    {
        TreeNode* cur = new TreeNode(ch);
        root->children.push_back(cur);
    }
    else
    {
        root->children.push_back(nodes.back());
        nodes.pop_back();
    }
}
nodes.push_back(root);
```

5） 当规约完毕后栈中不为#N#则报错

```
if (char_stack != "#N#")
    error("规约失败");
```

5. 程序编译

本程序使用 c++编写，使用 Visual Studio 2017 Professional 静态编译，在任何 x86 平台均可运行

6. 程序使用说明

本程序为命令行程序，并无图形界面。
双击打开即可运行。

7. 程序使用演示

E->E+T|T
F->P^F|P
T->T*F|F
P->(E)|i
#
i+i*i+i#

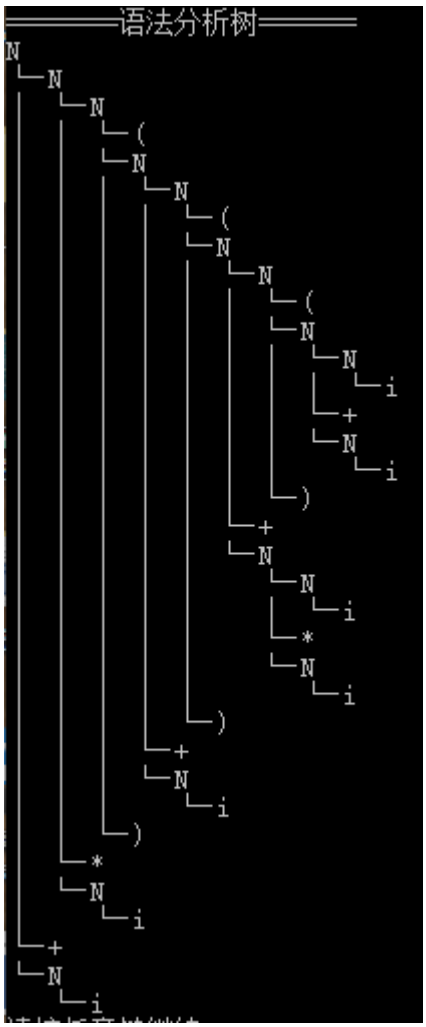
装 订 线

```
请输入文法，以一行#结束
E->E+T|T
F->P^F|P
T->T*F|F
P->(E)|i
#

═══════FIRST集═══════
FIRSTVT(E) = { (  *  +  ^  i  }
FIRSTVT(F) = { (  ^  i  }
FIRSTVT(P) = { (  i  }
FIRSTVT(T) = { (  *  ^  i  }
═══════LAST集═══════
LASTVT(E) = { )  *  +  ^  i  }
LASTVT(F) = { )  ^  i  }
LASTVT(P) = { )  i  }
LASTVT(T) = { )  *  ^  i  }

═══════优先关系表═══════
  #  (  )  *  +  ^  i
#  =  <  <  <  <  <  <
(  >  <  =  <  <  <  <
)  >     >  >  >  >
*  >  <  >  >  >  <  <
+  >  <  >  <  >  <  <
^  >  <  >  >  >  <  <
i  >     >  >  >  >
请输入单词串
i+i*i+i#

符号栈              输入串          关系    操作    最左素短语
#                   i+i*i+i#        <       移进
#i                  +i*i+i#         >       规约    i
#N                  +i*i+i#         <       移进
#N+                 i*i+i#         <       移进
#N+i                *i+i#          >       规约    i
#N+N                *i+i#          <       移进
#N+N*               i+i#           <       移进
#N+N*i              +i#            >       规约    i
#N+N*N              +i#            >       规约    N*N
#N+N                +i#            >       规约    N+N
#N                  +i#            <       移进
#N+                 i#             <       移进
#N+i                #              >       规约    i
#N+N                #              >       规约    N+N
#N                  #              =       移进

═══════语法分析树═══════
N
└─N
  └─i
└─+
└─N
  └─N
    └─N
      └─i
    └─*
    └─N
      └─i
  └─+
  └─N
    └─i
```

E->E+T
E->T
T->T*F
T->F

F->(E)
F->i
#
i+i*(i+(i*i+(i+i)))#

```
请输入文法，以一行#结束
E->E+T
E->T
T->T*F
T->F
F->(E)
F->i
#

========FIRST集=======
FIRSTVT(E) = { (  *  +  i  }
FIRSTVT(F) = { (  i  }
FIRSTVT(T) = { (  *  i  }
========LAST集=======
LASTVT(E) = { )  *  +  i  }
LASTVT(F) = { )  i  }
LASTVT(T) = { )  *  i  }

========优先关系表=======
  #  (  )  *  +  i
#  =  <  <  <  <  <
(  >  <  =  <  <  <
)  >        >  >  >
*  >  <  >  >  >  <
+  >  <  >  <  >  <
i  >        >  >  >
请输入单词串
i+i*(i+(i*i+(i+i)))#

符号栈              输入串              关系    操作    最左素短语
#                   i+i*(i+(i*i+(i+i)))#  <     移进
#i                  +i*(i+(i*i+(i+i)))#  >     规约    i
#N                  +i*(i+(i*i+(i+i)))#  <     移进
#N+                 i*(i+(i*i+(i+i)))#  <     移进
#N+i                *(i+(i*i+(i+i)))#  >     规约    i
#N+N                *(i+(i*i+(i+i)))#  <     移进
#N+N*               (i+(i*i+(i+i)))#  <     移进
#N+N*(              i+(i*i+(i+i)))#  <     移进
#N+N*(i             +(i*i+(i+i)))#  >     规约    i
#N+N*(N             +(i*i+(i+i)))#  <     移进
#N+N*(N+            (i*i+(i+i)))#  <     移进
#N+N*(N+(           i*i+(i+i)))#  <     移进
#N+N*(N+(i          *i+(i+i)))#  >     规约    i
#N+N*(N+(N          *i+(i+i)))#  <     移进
#N+N*(N+(N*         i+(i+i)))#  <     移进
#N+N*(N+(N*i        +(i+i)))#  >     规约    i
#N+N*(N+(N*N        +(i+i)))#  >     规约    N*N
#N+N*(N+(N          +(i+i)))#  <     移进
#N+N*(N+(N+         (i+i)))#  <     移进
#N+N*(N+(N+(        i+i)))#  <     移进
#N+N*(N+(N+(i       +i)))#  >     规约    i
#N+N*(N+(N+(N       +i)))#  <     移进
#N+N*(N+(N+(N+      i)))#  <     移进
#N+N*(N+(N+(N+i     )))#  >     规约    i
#N+N*(N+(N+(N+N     )))#  >     规约    N+N
#N+N*(N+(N+(N       )))#  =     移进
#N+N*(N+(N+(N)      ))#  >     规约    (N)
#N+N*(N+(N+N        ))#  >     规约    N+N
#N+N*(N+(N          ))#  =     移进
#N+N*(N+(N)         )#  >     规约    (N)
#N+N*(N+N           )#  >     规约    N+N
#N+N*(N             )#  =     移进
#N+N*(N)            #   >     规约    (N)
#N+N*N              #   >     规约    N*N
#N+N                #   >     规约    N+N
#N                  #   =     移进
```

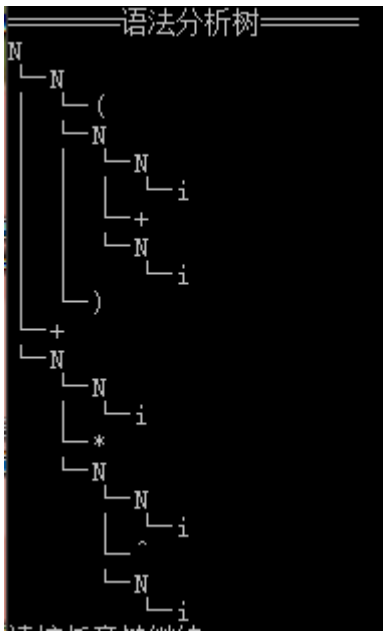


E->E+T|T
T->T*F|F
F->P^F|P
P->(E)|i
#
i^i*i+(i+i)#

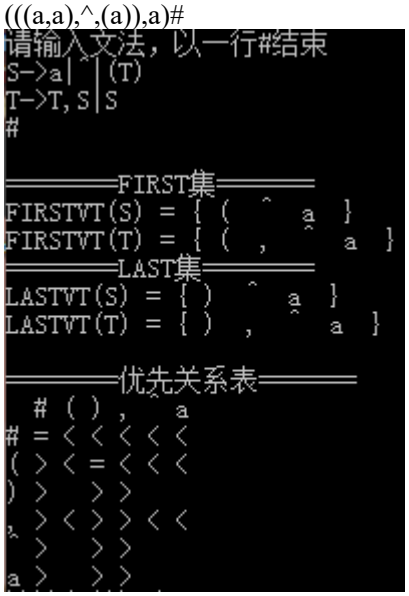
装 订 线

```
请输入文法，以一行#结束
E->E+T|T
T->T*F|F
F->P^F|P
P->(E)|i
#

=======FIRST集=======
FIRSTVT(E) = { (  *  +  ^  i  }
FIRSTVT(F) = { (  ^  i  }
FIRSTVT(P) = { (  i  }
FIRSTVT(T) = { (  *  ^  i  }
=======LAST集=======
LASTVT(E) = { )  *  +  ^  i  }
LASTVT(F) = { )  ^  i  }
LASTVT(P) = { )  i  }
LASTVT(T) = { )  *  ^  i  }

=======优先关系表=======
  # ( ) * + ^ i
# = < < < < < <
( > < = < < < <
) > > > > > >
* > < > > > < <
+ > < > < > < <
^ > < > > > < <
i > > > > > >
请输入单词串
i^i*i+(i+i)#

符号栈          输入串          关系    操作    最左素短语
#               i^i*i+(i+i)#    <       移进
#i              ^i*i+(i+i)#     >       规约    i
#N              ^i*i+(i+i)#     <       移进
#N^             i*i+(i+i)#      <       移进
#N^i            *i+(i+i)#       >       规约    i
#N^N            *i+(i+i)#       >       规约    N^N
#N              *i+(i+i)#       <       移进
#N*             i+(i+i)#        <       移进
#N*i            +(i+i)#         >       规约    i
#N*N            +(i+i)#         >       规约    N*N
#N              +(i+i)#         <       移进
#N+             (i+i)#          <       移进
#N+(            i+i)#           <       移进
#N+(i           +i)#            >       规约    i
#N+(N           +i)#            <       移进
#N+(N+          i)#             <       移进
#N+(N+i         )#              >       规约    i
#N+(N+N         )#              >       规约    N+N
#N+(N           )#              =       移进
#N+(N)          #               >       规约    (N)
#N+N            #               >       规约    N+N
#N              #               =       移进
```

装 订 线



S->a|^|(T)
T->T,S|S
#
(((a,a),^,(a)),a)#

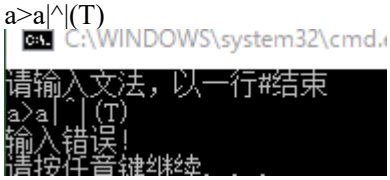


```
请输入单词串
(((a,a),^,(a)),a)#

符号栈                  输入串                  关系        操作        最左素
#                       (((a,a),^,(a)),a)#      <           移进
#(                      ((a,a),^,(a)),a)#       <           移进
#((                     (a,a),^,(a)),a)#        <           移进
#(((                    a,a),^,(a)),a)#         <           移进
#(((a                   ,a),^,(a)),a)#          >           规约        a
#(((N                   ,a),^,(a)),a)#          <           移进
#(((N,                  a),^,(a)),a)#           <           移进
#(((N,a                 ),^,(a)),a)#            >           规约        a
#(((N,N                 ),^,(a)),a)#            >           规约        N,N
#(((N                   ),^,(a)),a)#            =           移进
#(((N)                  ,^,(a)),a)#             >           规约        (N)
#((N                    ,^,(a)),a)#             <           移进
#((N,                   ^,(a)),a)#              <           移进
#((N,^                  ,(a)),a)#               >           规约        ^
#((N,N                  ,(a)),a)#               >           规约        N,N
#((N                    ,(a)),a)#               <           移进
#((N,                   (a)),a)#                <           移进
#((N,(                  a)),a)#                 <           移进
#((N,(a                 )),a)#                  >           规约        a
#((N,(N                 )),a)#                  =           移进
#((N,(N)                ),a)#                   >           规约        (N)
#((N,N                  ),a)#                   >           规约        N,N
#((N                    ),a)#                   =           移进
#((N)                   ,a)#                    >           规约        (N)
#(N                     ,a)#                    <           移进
#(N,                    a)#                     <           移进
#(N,a                   )#                      >           规约        a
#(N,N                   )#                      >           规约        N,N
#(N                     )#                      =           移进
#(N)                    #                       >           规约        (N)
#N                      #                       =           移进

=========语法分析树=========
N
└─(
└─N
  ├─N
  │ └─a
  ├─,
  └─N
    └─(
      └─N
        ├─N
        │ ├─(
        │ └─N
        │   └─a
        └─)
      ├─,
      └─N
        ├─N
        │ └─^
        ├─,
        └─N
          └─(
            └─N
              ├─N
              │ └─a
              ├─,
              └─N
                └─a
            └─)
      └─)
└─)
```

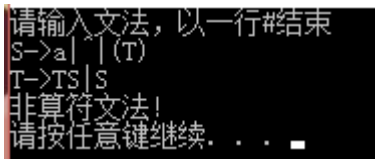
8. 错误测试

8.1. 输入错误

a>a|^|(T)



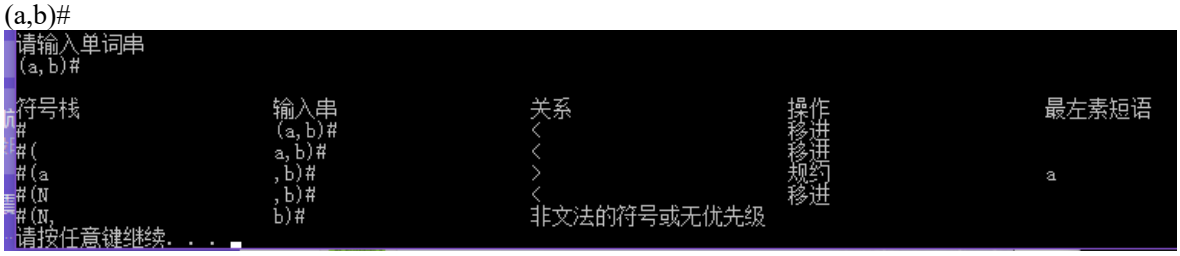
8.2. 非算符文法



8.3. 非算符优先文法

8.4. 非文法的符号

(a,b)#



8.5. 无优先级

a(a,a)#

```
请输入单词串
a(a,a)#

符号栈                  输入串          关系          操作          最左素短语
#                       a(a,a)#         <             移进
#a                      (a,a)#          非文法的符号或无优先级
请按任意键继续. . .
```

8.6. 最后不为#N#

```
请输入单词串
(a,a)##

符号栈                  输入串          关系          操作          最左素短语
#                       (a,a)##         <             移进
#(                      a,a)##          <             移进
#(a                     ,a)##           >             规约          a
#(N                     ,a)##           <             移进
#(N,                    a)##            <             移进
#(N,a                   )##             >             规约          a
#(N,N                   )##             >             规约          N,N
#(N                     )##             =             移进
#(N)                    ##              >             规约          (N)
#N                      ##              =             移进
#N#                     #               =             移进
规约失败
请按任意键继续. . .
```