

编译原理实验专题三：LL(1)语法分析设计原理与实现

编译原理实验专题三：LL(1)语法分析设计原理与实现

- 一、目标任务
- 二、设计说明
 - 2.1 背景介绍
 - 2.2 词法设计
 - 2.3 文法设计
 - 2.4 测试用例
- 三、实验步骤
 - 3.1 First集、Follow集的求解
 - 3.2 Select集的求解和LL1文法的判定
 - 3.3 LL1预测分析表的构建
 - 3.4 预测分析的实现和测试
- 四、实验心得

一、目标任务

编程实现下面的文法的 LL(1)分析过程： $G[S]: S \rightarrow V=E \ E \rightarrow TE' \ E' \rightarrow ATE' \mid \epsilon \ T \rightarrow FT' \ T' \rightarrow MFT' \mid \epsilon \ F \rightarrow (E) \mid i$
 $A \rightarrow + \mid - \ M \rightarrow * \mid / \ V \rightarrow i$

$G[S]: S \rightarrow V=E$
 $E \rightarrow TE'$
 $E' \rightarrow ATE' \mid \epsilon$
 $T \rightarrow FT'$
 $T' \rightarrow MFT' \mid \epsilon$
 $F \rightarrow (E) \mid i$
 $A \rightarrow + \mid -$
 $M \rightarrow * \mid /$
 $V \rightarrow i$

二、设计说明

2.1 背景介绍

在实验一、实验二所建工程的基础上继续进行实验。

背景介绍：实验一我完成了词法分析器，程序能够自动解析正则表达式定义的词源并自动生成NFA以完成词法分析。实验二我完成了扩展的巴克斯淖尔范式的解析，以及最左公因子提取和左递归的消除，并初步实现了First集、Follow集和Select集的求解以及LL1文法的判定。实验三（本次实验），我将在前两次实验的基础上实现LL1预测分析表法的语法分析。

2.2 词法设计

首先根据实验要求，设计词法分析器所需要的定义如下（lab3.lex）：

```
1  PATTERN ${
2      BLANK      \s+
3      IDENTIFIER  [\a_][\w]*
4      SEPARATOR   [\+|-|*|/]
5  $}
6
7  IGNORE ${
8      BLANK
9  $}
```

该词法定义描述了，词法分析器可以识别空白、标识符和分隔符。词法分析器将会把读入文件中符合正则表达式所识别的单词封装成tokens序列传递给后续文法分析程序。

2.3 文法设计

而后根据实验要求，设计文法分析器所需要的定义如下（lab3.stx）：

```
1  GRAMMAR ${
2      S* ::= $V '=' E;
3      E  ::= T E';
4      E' ::= A T E' | \e ;
5      T  ::= F T';
6      T' ::= M F T' | \e ;
7      F  ::= '(' E ')' | $v;
8      A  ::= '+' | '-' ;
9      M  ::= '*' | '/' ;
10 $}
11
12 MAPPING ${
13     $V    -->    @IDENTIFIER ;
14 $}
```

该文法定义使用扩展的巴克斯淖尔范式（由于题目所给文法过于简单，且为了方便与正确答案对比，此处刻意没有使用EBNF的高级语法）定义了题目所给文法（具体定义说明请参见实验二报告）。

值得注意的是，文法额外定义了将词法分析程序所识别的 IDENTIFIER 类型映射到 \$V 上，这样便实现了将任意标识符作为运算对象 i 的替代，而不会使文法分析器只能识别 i。

2.4 测试用例

根据2.3所述，我的分析程序可以自动完成词法类型到文法终结符类型的映射，故此特意将测试程序中的 i 用一些合法的标识符代替，参考如下（lab3.txt）：

```
1  res=a*b+(c/d-e)
```

三、实验步骤

3.1 First集、Follow集的求解

原理解析：

■构造FIRST的算法

(-)对 $G[S]$, $x \in V_n \cup V_t$, 计算FIRST(x)

① 若 $x \in V_t$

则 $\text{FIRST}(x) = \{x\}$

② 若 $x \in V_n$

有 $x \rightarrow a\alpha$, ($a \in V_t$)或/和 $x \rightarrow \varepsilon$

则 a 或/和 $\varepsilon \in \text{FIRST}(x)$

③ 对 $x \rightarrow Y_1 Y_2 \dots Y_k$ (且 $Y_1 \in V_n$), 反复使用以下直到每一个 $\text{FIRST}(x)$ 不再增大为止.

i 若 $Y_1 \in V_n$

则把 $\text{FIRST}(Y_1) - \{\varepsilon\}$ 元素加入 $\text{FIRST}(x)$ 中

ii 若 $Y_1, Y_2, \dots, Y_{i-1} \in V_n$ ($2 \leq i \leq k$)

且对于任何 j , $\varepsilon \in \text{FIRST}(Y_j)$ ($1 \leq j \leq i-1$)

则把所有 $\text{FIRST}(Y_i) - \{\varepsilon\}$ 元素加入 $\text{FIRST}(x)$ 中

iii 若 $Y_1, Y_2, \dots, Y_k \in V_n$

且对于任何 j , $\varepsilon \in \text{FIRST}(Y_j)$ ($1 \leq j \leq k$)

则把 ε 元素加入 $\text{FIRST}(x)$ 中

■构造FOLLOW(A)的算法 $A, B \in V_n$

①令 $\# \in \text{FOLLOW}(S)$ S 为文法开始符号

②对 $A \rightarrow \alpha B \beta$, 且 $\beta \neq \varepsilon$

则将 **FIRST(β) - $\{\varepsilon\}$** 加入**FOLLOW(B)**中

③反复, 直至每一个FOLLOW(A)不再增大

对 $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha B \beta$ (且 $\varepsilon \in \text{FIRST}(\beta)$)

则**FOLLOW(A)**中的全部元素加入**FOLLOW(B)**

实现代码:

```
1  set<term> Grammar::calcFirstOf(term t)
2  {
3      debug(0) << "Calculating First(" << t << ")" << endl;
4      if (first[t].size() > 0)
5      {
6          debug(1) << "First(" << t << ") has been calculated before" <<
endl;
7          return first[t]; // 已经计算过
8      }
9      // 对于每个产生式, 计算first集
10     for (auto &right : rules[t])
11     {
12         // 空产生式, 加入epsilon
13         if (right.size() == 0)
14         {
15             first[t].insert(EPSILON);
16             debug(1) << "First(" << t << ") <- {epsilon}" << endl;
17         }
18         else
19         {
20             // 对于产生式右部的每个符号
21             bool allHaveEpsilon = true;
22             for (auto &symbol : right)
23             {
24                 if (!_find(terminals, symbol))
25                 {
26                     // 终结符, 直接加入first集
27                     first[t].insert(symbol);
```

```

28         debug(1) << "First(" << t << ") <- {" << symbol << "}"
    << endl;
29         allHaveEpsilon = false;
30         break; // 终结符后面的符号不再计算
31     }
32     else
33     {
34         set<term> resFirst = calcFirstOf(symbol);
35         set<term> tmpFirst = resFirst;
36         tmpFirst.erase(EPSILON);
37         first[t].insert(tmpFirst.begin(), tmpFirst.end());
38         for (auto &f : tmpFirst)
39         {
40             debug(1) << "First(" << t << ") <- {" << f << "}"
    << endl;
41         }
42         if (!_find(resFirst, EPSILON))
43         {
44             // 该符号的first集不含epsilon, 后面的符号不再计算
45             allHaveEpsilon = false;
46             break;
47         }
48     }
49 }
50 if (allHaveEpsilon)
51 {
52     first[t].insert(EPSILON);
53     debug(1) << "First(" << t << ") <- {epsilon}" << endl;
54 }
55 }
56 }
57 return first[t];
58 }
59
60 set<term> Grammar::calcFollowOf(term t)
61 {
62     debug(0) << "Calculating Follow(" << t << ")" << endl;
63     if (follow[t].size() > 0)
64     {
65         debug(1) << "Follow(" << t << ") has been calculated before" <<
endl;
66         return follow[t]; // 已经计算过
67     }
68     if (t == startTerm)
69     {
70         follow[t].insert(SYM_END);
71         debug(1) << "Follow(" << t << ") <- {" << endl;
72     }
73     for (auto &rule : rules)
74     {
75         for (auto &right : rule.second)
76         {
77             auto symIt = right.begin();
78             symIt = find(symIt, right.end(), t);
79             while (symIt != right.end())
80             {

```

```

81         if (symIt == right.end() - 1)
82         {
83             // A -> aB
84             if (rule.first != t)
85             {
86                 set<term> resFollow = calcFollowOf(rule.first);
87                 follow[t].insert(resFollow.begin(),
resFollow.end());
88                 for (auto &f : resFollow)
89                 {
90                     debug(1) << "Follow(" << t << ") <-> {" << f <<
"} <A -> aB>" << endl;
91                 }
92             }
93         }
94         else if (_find(terminals, *(symIt + 1)))
95         {
96             // A -> aBb
97             follow[t].insert(*(symIt + 1));
98             debug(1) << "Follow(" << t << ") <- {" << *(symIt + 1)
<< "}" <A -> aBb>" << endl;
99         }
100        else
101        {
102            // A -> aBC
103            set<term> resFirst;
104            for (auto tmpIt = symIt + 1; tmpIt != right.end();
tmpIt++)
105            {
106                resFirst = calcFirstOf(*tmpIt);
107                set<term> tmpFirst = resFirst;
108                tmpFirst.erase(EPSILON);
109                follow[t].insert(tmpFirst.begin(), tmpFirst.end());
110                for (auto &f : tmpFirst)
111                {
112                    debug(1) << "Follow(" << t << ") = {" << f <<
"} <A -> aBC>" << endl;
113                }
114                if (!_find(resFirst, EPSILON))
115                {
116                    break;
117                }
118            }
119            if (resFirst.size() == 0 || _find(resFirst, EPSILON))
120            {
121                // A -> aBC, First(C) 含有epsilon
122                if (rule.first != t)
123                {
124                    set<term> resFollow = calcFollowOf(rule.first);
125                    follow[t].insert(resFollow.begin(),
resFollow.end());
126                    for (auto &f : resFollow)
127                    {
128                        debug(1) << "Follow(" << t << ") = {" << f
<< "}" <A -> aBC(e)>" << endl;
129                    }

```

```

130         }
131     }
132 }
133     symIt = find(symIt + 1, right.end(), t);
134 }
135 }
136 }
137 return follow[t];
138 }

```

实验结果：

```

Microsoft Visual Studio 调试
[1] First(E') <- {+}
[1] First(E') <- {-}
[0] Calculating First(F)
[1] First(F) has been calculated before
[0] Calculating First(M)
[1] First(M) <- {*}
[1] First(M) <- {/}
[0] Calculating First(S)
[1] First(S) <- {$V}
[0] Calculating First(T)
[1] First(T) has been calculated before
[0] Calculating First(T')
[1] First(T') <- {epsilon}
[0] Calculating First(M)
[1] First(M) has been calculated before
[1] First(T') <- {*}
[1] First(T') <- {/}
[info] Calculating Follow...
[0] Calculating Follow(A)
[0] Calculating First(T)
[1] First(T) has been calculated before
[1] Follow(A) = {$V} <A -> aBC>
[1] Follow(A) = {(} <A -> aBC>
[0] Calculating Follow(E)
[1] Follow(E) <- {} <A -> aBb>
[0] Calculating Follow(S)
[1] Follow(S) <- {#}
[1] Follow(E) <-> {#} <A -> aB>
[0] Calculating Follow(E')
[0] Calculating Follow(E)

```

```

Microsoft Visual Studio 调试
[0] Calculating Select(M -> [*])
[0] Calculating First(M->[*])
[1] First(M) <- {*}
[1] Select(M -> [*]) <- {*}
[0] Calculating Select(M -> [/])
[0] Calculating First(M->[/])
[1] First(M) <- {/}
[1] Select(M -> [/]) <- {/}
[info] First:
A : + -
E : $V (
E' : + - @
F : $V (
M : * /
S : $V
T : $V (
T' : * / @
[info] Follow:
A : $V (
E : # )
E' : # )
F : # ) * + - /
M : $V (
S : #
T : # ) + -
T' : # ) + -
[info] First of Product:
S -> [$V, =, E] : {$V}
E -> [T, E'] : {$V, (}
E' -> [A, T, E'] : {+, -}

```

3.2 Select集的求解和LL1文法的判定

原理解析：

在编译原理中，Select集是用于构建LL(1)分析表的关键概念。Select集的本质是在构造自顶向下的语法分析器时，用于预测输入串中的

下一个符号应该选择哪一条产生式进行推导。Select集帮助我们确定在某个非终结符的推导过程中，应该选择哪一条产生式来进行推导。

它与LL(1)预测分析表有密切关系，因为构建LL(1)预测分析表的过程就是计算各个产生式的Select集。

(二) 构造FIRST(α) $\alpha = X_1X_2 \dots X_n$
 $X_i \in V, \alpha \in V^*$

① **置** FIRST(α) = { }

② FIRST(X_1) - { ϵ } 加入 FIRST(α)

③ **若** $\epsilon \in \text{FIRST}(X_1)$,

则 FIRST(X_2) - { ϵ } 加入 FIRST(α)

若 $\epsilon \in \text{FIRST}(X_1)$ **且** $\epsilon \in \text{FIRST}(X_2)$

则 FIRST(X_3) - { ϵ } 加入 FIRST(α)

.....以此类推

若 $\epsilon \in \text{FIRST}(X_i) \quad 1 \leq i \leq n$

则 $\epsilon \in \text{FIRST}(\alpha)$

实现代码：

```
1  set<term> Grammar::calcFirstOf(production product)
2  {
3      debug(0) << format("Calculating First($->$)", product.first,
vec2str(product.second)) << endl;
4      if (firstP[product].size() > 0)
5      {
6          debug(1) << format(
7              "First($->$) has been calculated before.\n",
8              product.first,
9              vec2str(product.second));
10         return firstP[product]; // 已经计算过
11     }
12     set<term> resFirst;
13     bool allHaveEpsilon = true;
14     for (auto &symbol : product.second)
15     {
16         if (_find(terminals, symbol))
17         {
18             // 终结符，直接加入first集
19             resFirst.insert(symbol);
20             debug(1) << "First(" << product.first << ") <- {" << symbol <<
"}" << endl;
21             allHaveEpsilon = false;
```



```

22         break; // 终结符后面的符号不再计算
23     }
24     else
25     {
26         set<term> tmpFirst = calcFirstOf(symbol);
27         resFirst.insert(tmpFirst.begin(), tmpFirst.end());
28         debug(1) << "First(" << product.first << ") <- " <<
container2str(tmpFirst) << endl;
29         if (!_find(tmpFirst, EPSILON))
30         {
31             // 该符号的first集不含epsilon, 后面的符号不再计算
32             allHaveEpsilon = false;
33             break;
34         }
35     }
36 }
37 if (allHaveEpsilon)
38 {
39     resFirst.insert(EPSILON);
40     debug(1) << "First(" << product.first << ") <- { epsilon }" << endl;
41 }
42 firstP[product] = resFirst;
43 return resFirst;
44 }
45
46 set<term> Grammar::calcSelectOf(production product)
47 {
48     debug(0) << "Calculating Select(" << product.first;
49     debug_u(0) << " -> " << vec2str(product.second) << ")" << endl;
50     set<term> resSelect;
51     set<term> resFirst = calcFirstOf(product);
52     set<term> tmpFirst = resFirst;
53     tmpFirst.erase(EPSILON); // First(A) - {epsilon}
54     resSelect.insert(tmpFirst.begin(), tmpFirst.end());
55     debug(1) << "Select(" << product.first << " -> " <<
vec2str(product.second);
56     debug_u(1) << ")" <- " << set2str(tmpFirst) << endl;
57     if (_find(resFirst, EPSILON))
58     {
59         set<term> resFollow = calcFollowOf(product.first);
60         resSelect.insert(resFollow.begin(), resFollow.end());
61         debug(1) << "Select(" << product.first << " -> " <<
vec2str(product.second);
62         debug_u(1) << ")" <- " << set2str(resFollow) << endl;
63     }
64     return resSelect;
65 }

```

实验结果：

```
Microsoft Visual Studio 调试
[info] First of Product:
S -> [$V, =, E] : {$V}
E -> [T, E'] : {$V, ()}
E' -> [A, T, E'] : {+, -}
E' -> [] : { @}
T -> [F, T'] : {$V, ()}
T' -> [M, F, T'] : {*, /}
T' -> [] : { @}
F -> [C, E, )] : {C}
F -> [$V] : {$V}
A -> [+] : {+}
A -> [-] : {-}
M -> [*] : {*}
M -> [/] : {/}
[info] Select:
S -> [$V, =, E] : {$V}
E -> [T, E'] : {$V, ()}
E' -> [A, T, E'] : {+, -}
E' -> [] : { #, )}
T -> [F, T'] : {$V, ()}
T' -> [M, F, T'] : {*, /}
T' -> [] : { #, ), +, -}
F -> [C, E, )] : {C}
F -> [$V] : {$V}
A -> [+] : {+}
A -> [-] : {-}
M -> [*] : {*}
M -> [/] : {/}
[info] Grammar: Checking LL(1)
[0] Processing Non-Terminal A
```

3.3 LL1预测分析表的构建

原理解析：

■构造分析表的算法

P120上部

由每一个产生式 $A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$

确定 $M[A, a]$ 矩阵 $a \in V_t$

① 任何 $a \in \text{FIRST}(\alpha_i)$, 将 $A \rightarrow \alpha_i$ 规则填入 $M[A, a]$

② 若 $\epsilon \in \text{FIRST}(\alpha_i)$,

则对于任一个 $b \in \text{FOLLOW}(A)$ $b \in V_t$ 或 #

将 $A \rightarrow \epsilon$ 规则填入 $M[A, b]$

➤ 此时 b 不属于 $\text{FIRST}(A)$

③ 其它空白为出错

实现代码：

```
1 #define _find(s, t) (s.find(t) != s.end())
2
3 void PredictiveTableAnalyzer::calcPredictTable()
4 {
5     for (auto pdt : grammar.products)
6     {
```

```

7         auto first = grammar.firstP[pdt];
8         for (auto t : first)
9         {
10             if (t != EPSILON)
11             {
12                 predict[pdt.first][t] = pdt.second;
13             }
14         }
15         if (_find(first, EPSILON))
16         {
17             auto follow = grammar.follow[pdt.first];
18             for (auto t : follow)
19             {
20                 if (_find(grammar.terminals, t))
21                     predict[pdt.first][t] = pdt.second;
22             }
23         }
24     }
25 }

```

实验结果：

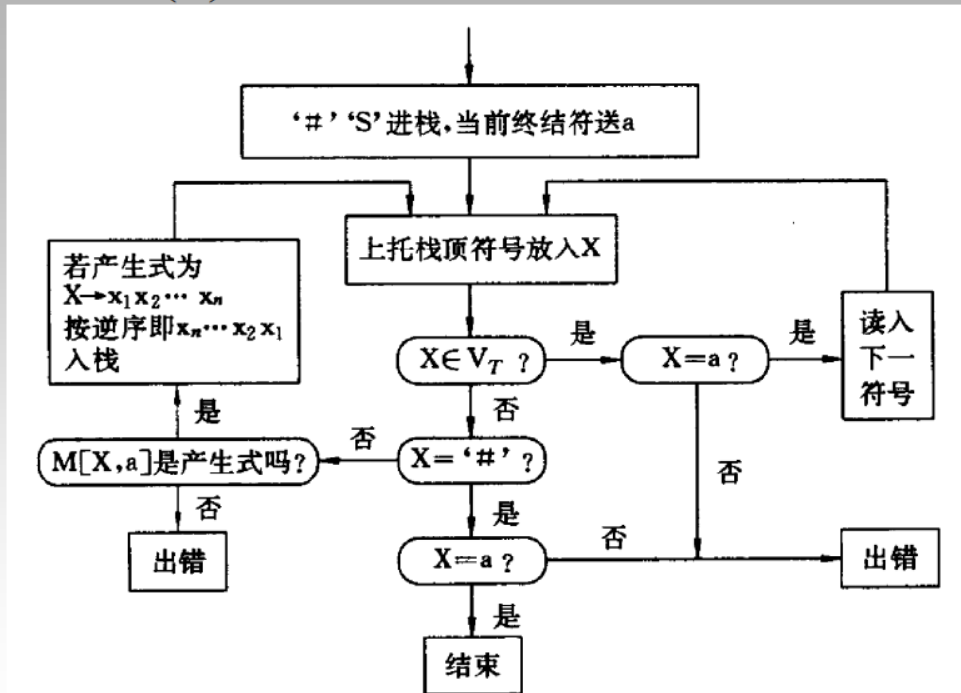
[Info] Predictive Table:

Non-Term	#	\$V	()	*	+	-	/	=
A						A → [+]	A → [-]		
E		E → [T, E']	E → [T, E']						
E'	E' → []			E' → []		E' → [A, T, E']	E' → [A, T, E']		
F		F → [\$V]	F → [(, E,)]						
M					M → [+]			M → [/]	
S		S → [\$V, =, E]							
T		T → [F, T']	T → [F, T']						
T'	T' → []			T' → []	T' → [M, F, T']	T' → []	T' → []	T' → [M, F, T']	

3.4 预测分析的实现和测试

原理解析：

LL(1)分析（预测分析）程序框图



实现代码：

```

1  bool PredictiveTableAnalyzer::analyze(vector<token> input)
2  {
3      stack<term> s;
4      input.push_back(token(make_shared<term>(SYM_END), SYM_END, 0, 0));
5      s.push(SYM_END);
6      s.push(grammar.startTerm);
7      int i = 0;
8      while (!s.empty() && s.top() != SYM_END && i < input.size())
9      {
10         std::cout << setw(8) << std::left;
11         printStack(s);
12         std::cout << " | ";
13         std::cout << setw(60) << std::right;
14         printVecFrom(input, i);
15         assert(_find(grammar.terminals, *(input[i].type)));
16         if (s.top() == *(input[i].type))
17         {
18             s.pop();
19             i++;
20             cout << setw(4) << right << " $ ";
21             stringstream ss;
22             ss << "match " << input[i - 1].value;
23             cout << setw(40) << right << ss.str();
24             cout << endl;
25         }
26         else if (_find(grammar.nonTerms, s.top()))
27         {
28             auto it = predict[s.top()].find(*(input[i].type));
29             if (it != predict[s.top()].end())

```

```

30         {
31             s.pop();
32             for (auto it1 = it->second.rbegin(); it1 != it-
>second.rend(); it1++)
33             {
34                 s.push(*it1);
35             }
36             cout << setw(4) << right << " $ ";
37             stringstream ss;
38             ss << it->first << "->" << vec2str(it->second);
39             cout << setw(40) << right << ss.str();
40             cout << endl;
41         }
42         else
43         {
44             error << "Error: " << input[i].line << ":" << input[i].col
<< ": "
45                 << "Unexpected token: " << input[i].type << std::endl;
46             return false;
47         }
48     }
49     else
50     {
51         error << "Error: " << input[i].line << ":" << input[i].col << ":
"
52             << "Unexpected token: " << input[i].type << std::endl;
53         return false;
54     }
55 }
56 std::cout << setw(8) << std::left;
57 printStack(s);
58 std::cout << " | ";
59 std::cout << setw(60) << std::right;
60 printVecFrom(input, i);
61 std::cout << std::endl;
62 return true;
63 }

```

实验结果：

