编译原理实验2 语法分析器

19335015 陈恩婷

1. 实验描述

分别用LL(1)分析法和LR(0)分析法，手动设计实现算法表达式语法分析器，至少支持加减乘除以及括号操作，即（+，-，*，/，（））。

2. 实验原理

语法分析是编译的第二阶段；其任务是识别和处理比单词更大的语法单位，如：程序设计语言中的表达式、各种说明和语句乃至全部源程序，指出其中的语法错误；必要时，可生成内部形式，便于下一阶段处理。

在本实验中，词法分析器的功能是输入一个文法和数学表达式，输出判断结果和错误原因。

3. 所使用的文法

根据实验要求和课本描述，本实验使用的文法如下：

$$
\begin{aligned}
E &\rightarrow T \mid E\ \omega_0\ T \\
T &\rightarrow F \mid T\ \omega_1\ F \\
F &\rightarrow i \mid (\ E\ )
\end{aligned}
$$

其中i表示数字或常数，ω0表示加号或减号，ω1表示乘号或除号。

根据LL(1)和LR(0)两种分析法，会在读入文法后进行相应的文法变换。

4. 算法过程及分析表

4.1 LL(1)分析法

LL(1)分析法的主要步骤如下：

1. 将输入的文法转化为LL(1)文法
2. 对于每个非终结符，计算其first集
3. 对于每个非终结符，计算其follow集
4. 对于每个产生式，计算其select集
5. 利用select集，生成LL(1)分析表
6. 利用分析表，对输入的表达式进行解析

相应的main函数如下：

```
int main(){
    get_grammar("grammar.txt");
    convert_to_ll1();
    get_first_set();
    get_follow_set();
    get_select_set();
    get_table();
    // ll1_parsing
    cout << "\nll1_parsing" << endl;
    bool result = ll1_parsing();
```

```
    if ( result ){
        cout << "success" << endl;
    }
    else{
        cout << "error" << endl;
    }
    return 0;
}
```

接下来逐步描述以上过程。

4.1.1 转化为LL(1)文法

将表达式文法转换为LL(1)文法的主要步骤就是消除左递归，也就是说，如果有一个非终结符，它有以下两个产生式：

$$A \to A\alpha|\beta$$

那么就需要将其转换为如下表达式：

$$A \to \beta R$$
$$R \to \alpha R|\epsilon$$

按照以上规则，转换后的表达式文法如下：

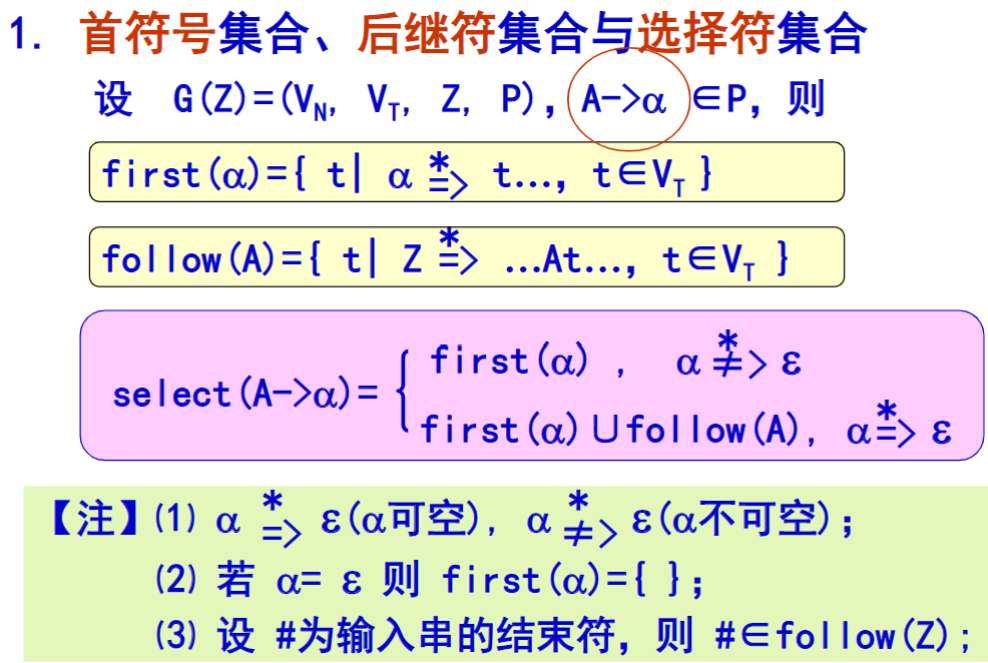
$$E \to T\ E_1$$
$$E_1 \to w_0\ T\ E_1 \mid \epsilon$$
$$F \to i \mid (\ E\ )$$
$$T \to F\ T_1$$
$$T_1 \to w_1\ F\ T_1 \mid \epsilon$$

程序输出如下：

```
E -> T E1 |
E1 -> w0 T E1 | @ |
F -> i | ( E ) |
T -> F T1 |
T1 -> w1 F T1 | @ |
```

4.1.2 计算first集、follow集和select集

根据课件，first集、follow集和select集的计算方法如下：



计算结果如下：

first集	follow集	select集
(-> (@ -> E -> (i E1 -> w0 F -> (i T -> (i T1 -> w1 i -> i w0 -> w0 w1 -> w1	E -> #) E1 -> #) F -> #) w0 w1 T -> #) w0 T1 -> #) w0	E -> T E1 select_set: (i E1 -> w0 T E1 select_set: w0 E1 -> @ select_set: #) F -> i select_set: i F -> (E) select_set: (T -> F T1 select_set: (i T1 -> w1 F T1 select_set: w1 T1 -> @ select_set: #) w0

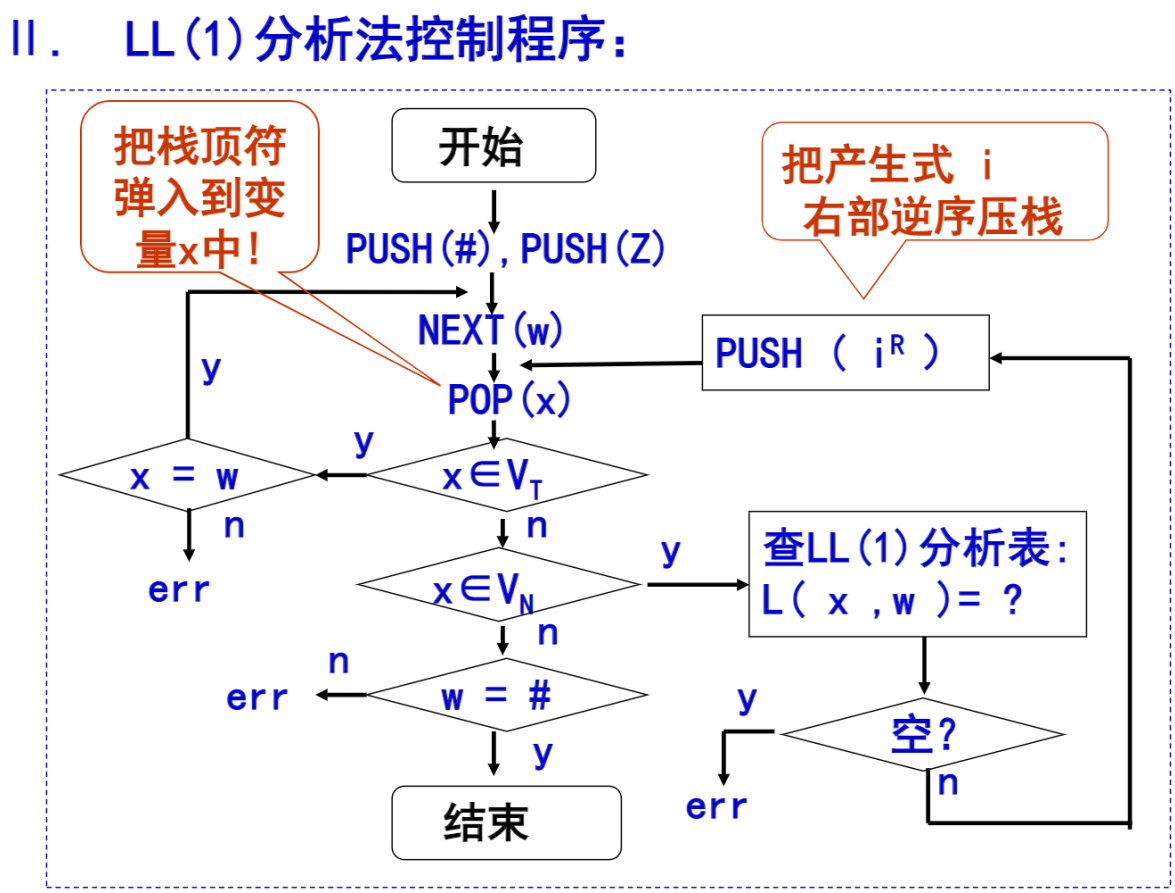
4.1.3 生成LL(1)分析表

接下来根据各个非终结符的select集，生成LL(1)分析表，其中分析表每一行代表一个非终结符，每一列代表一个终结符。对于每个产生式，其左部的非终结符和其select集中的终结符构成了表格的行和列，产生式的序号就是相应单元格的内容。

```
E,(:0   E,i:0   E1,#:2  E1,):2  E1,w0:1
F,(:4   F,i:3   T,(:5   T,i:5   T1,#:7
T1,):7  T1,w0:7 T1,w1:6
```

4.1.4 用分析表解析表达式

LL(1)分析法控制程序的算法如下：



根据以上算法，结合计算好的分析表，就可以实验LL(1)解析程序。

4.2 LR(0)分析法

LR(0)的主要步骤如下：

1. 对于读入的文法，计算它的LR(0)规范集合
2. 在计算规范集合的基础上，计算出LR(0)分析表
3. 利用LR(0)分析表，对输入的表达式进行解析

接下来逐步解释每一步的过程。

4.2.1 计算LR(0)规范集合

根据课本的介绍，计算LR(0)规范集合的算法如下：

```
void items(G') {
    C = {CLOSURE({[S' → ·S]})};
    repeat
        for ( each set of items I in C )
            for ( each grammar symbol X )
                if ( GOTO(I, X) is not empty and not in C )
                    add GOTO(I, X) to C;
    until no new sets of items are added to C on a round;
}
```

Figure 4.33: Computation of the canonical collection of sets of LR(0) items

其中计算CLOSURE的算法如下：

SetOfItems CLOSURE(I) {
    $J = I$;
    repeat
        for (each item $A \to \alpha \cdot B\beta$ in J)
            for (each production $B \to \gamma$ of G)
                if ($B \to \cdot\gamma$ is not in J)
                    add $B \to \cdot\gamma$ to J;
    until no more items are added to J on one round;
    return J;
}

Figure 4.32: Computation of CLOSURE

另外，GOTO(I, X)的计算方法如下：

$$GOTO(I; X) \textit{ is defined to be the closure of the set of all items } [A \to \alpha X \cdot \beta], \textit{ such that } [A \to \alpha \cdot X\beta] \textit{ is in } I$$

对表达式文法计算LR(0)的规范集合，结果如下：

```
size of sets: 12
{E -> . E w0 T
E -> . T
E' -> . E
F -> . ( E )
F -> . i
T -> . F
T -> . T w1 F
}
{E -> . E w0 T
E -> . T
F -> ( . E )
F -> . ( E )
F -> . i
T -> . F
T -> . T w1 F
}
{E -> E . w0 T
F -> ( E . )
}
{F -> ( E ) .
}
```

```
{E -> E w0 . T
F -> . ( E )
F -> . i
T -> . F
T -> . T w1 F
}
{T -> F .
}
{E -> E w0 T .
T -> T . w1 F
}
{F -> . ( E )
F -> . i
T -> T w1 . F
}
{T -> T w1 F .
}
{F -> i .
}
{E -> T .
T -> T . w1 F
}
{E -> E . w0 T
E' -> E .
}
```

共计12个规范集。

4.2.2 计算LR(0)分析表

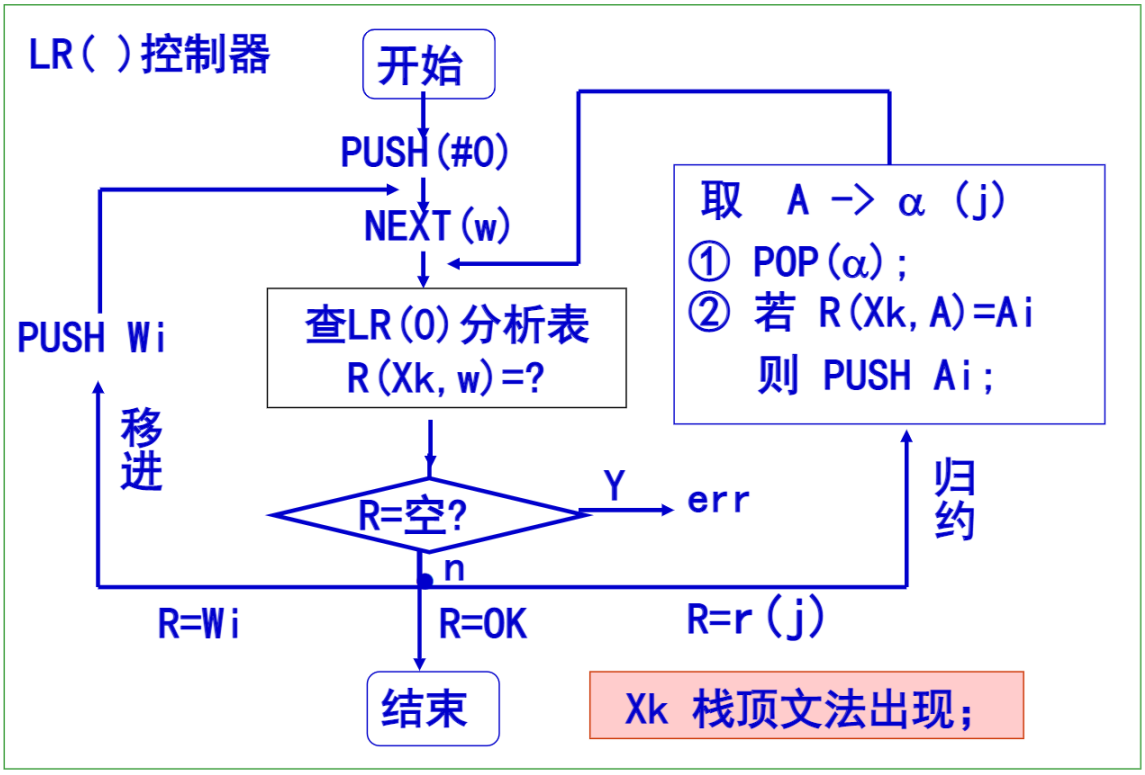
LR(0)分析表可以在计算LR(0)的过程中直接得到，计算结果如下：

```
# 3 r 4
# 5 r 5
# 6 r 2
# 8 r 6
# 9 r 3
# 10 r 1
# 11 acc 0
( 0 s 1
( 1 s 1
( 3 r 4
( 4 s 1
( 5 r 5
( 6 r 2
( 7 s 1
( 8 r 6
( 9 r 3
( 10 r 1
) 2 s 3
) 3 r 4
) 5 r 5
) 6 r 2
) 8 r 6
) 9 r 3
) 10 r 1
E 0 s 11
E 1 s 2
```

```
F 0 s 5
F 1 s 5
F 4 s 5
F 7 s 8
T 0 s 10
T 1 s 10
T 4 s 6
i 0 s 9
i 1 s 9
i 3 r 4
i 4 s 9
i 5 r 5
i 6 r 2
i 7 s 9
i 8 r 6
i 9 r 3
i 10 r 1
w0 2 s 4
w0 3 r 4
w0 5 r 5
w0 6 r 2
w0 8 r 6
w0 9 r 3
w0 10 r 1
w0 11 s 4
w1 3 r 4
w1 5 r 5
w1 6 s 7
w1 8 r 6
w1 9 r 3
w1 10 s 7
```

4.2.3 用LR(0)分析表解析表达式

3. LR(0)控制程序设计



根据以上算法，结合计算好的分析表，就可以实验LR(0)解析程序。

5. 实验结果

5.1 LL(1)分析法

5.1.1 例1: 1+a*b

用1+a*b作为输入样例1，可见程序输出了success，说明该表达式符合文法：

```
ll1_parsing
input: success
```

5.2.2 例2：1+a)*b

这个显然是个错误的例子。程序输出如下：

```
ll1_parsing
input: error: stack top is terminal, but is not equal to current symbol
x: #
error
input: i w0 i ) w1 i #
stack: #
w: )
```

如图所示，栈内剩余的元素为“#”，但是当前符为")"，无法进行解析，所以程序报错了。

5.2 LR(0)分析法

5.2.1 例1: 1+a*b

用1+a*b作为输入样例1，可见程序输出了accept，说明该表达式符合文法：

```
0 i
action: s
9 w0 i
action: r
5 w0 F
action: r
10 w0 T
action: r
11 w0 E
action: s
4 i E w0
action: s
9 w1 E w0 i
action: r
5 w1 E w0 F
action: r
6 w1 E w0 T
action: s
7 i E w0 T w1
action: s
9 # E w0 T w1 i
action: r
8 # E w0 T w1 F
action: r
6 # E w0 T
action: r
11 # E
action: acc
accept
```

在前面的输出中，每一步输出两行，第一行为栈顶、当前符号和symbols，第二行为相应的action，其中s为移进，r为规约。

5.2.2 例2：1+a)*b

这个显然是个错误的例子。程序输出如下：

```
error: cannot find table entry, cannot get action
input: i w0 i ) w1 i #
stack: 11 0 symbols: E w: )
```

如图所示，程序已经归约到了状态0，但是当前的符号为")"，在分析表中无法查到相应的action，所以程序报错了。

6. 实验总结

本次实验通过自己实现一个语法分析器，复习了理论课上讲到的内容，让我对这些部分有了更深刻的认识。

关于实验中的LR(0)的部分，我采用的是课本上的LR(0)算法进行实现，个人感觉课件上的算法不够完整，对初学者来说不易于程序实现。

这次实验让我受益匪浅，希望再接再厉，学好编译原理的知识。

7. 相关链接

1. 本实验的GitHub链接
2. 课本：Compilers: Principles, Techniques, and Tools