编译原理实验3 语义分析器

19335015 陈恩婷

1. 实验描述

使用LL(1)翻译法，实现高级编程语言的语义分析，将其翻译为四元式格式的中间语言，至少支持算术表达式的语义分析。算数表达式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三阶段，其任务是对结构上正确的源程序进行上下文有关性质的审查，进行类型审查。

在本实验中，语义分析器的功能是输入一个文法和数学表达式，输出四元式格式的中间语言。

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：

$$
\begin{aligned}
E &\rightarrow T\{\omega_0\ T\ \{GET(w0)\}\} \\
T &\rightarrow F\{\omega_1\ F\ \{GET(w1)\}\} \\
F &\rightarrow i\ \{PUSH(i)\} \mid (\ E\ )
\end{aligned}
$$

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)翻译法，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)翻译法

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，生成四元式

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();
}
```

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和闭包，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：

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

如果有一个非终结符有以下产生式：

$$A \to B\{\beta R\}$$

那么就将它转换为如下表达式：

$$A \to B\ C$$
$$C \to \beta\ R\ C \mid \epsilon$$

按照以上规则，转换后的表达式文法如下：

$$E \to T\ E_1$$
$$E_1 \to w_0\ T\ GET(w0)\ E_1 \mid \epsilon$$
$$F \to i\ PUSH(i) \mid (\ E\ )$$
$$T \to F\ T_1$$
$$T_1 \to w_1\ F\ GET(w1)\ T_1 \mid \epsilon$$

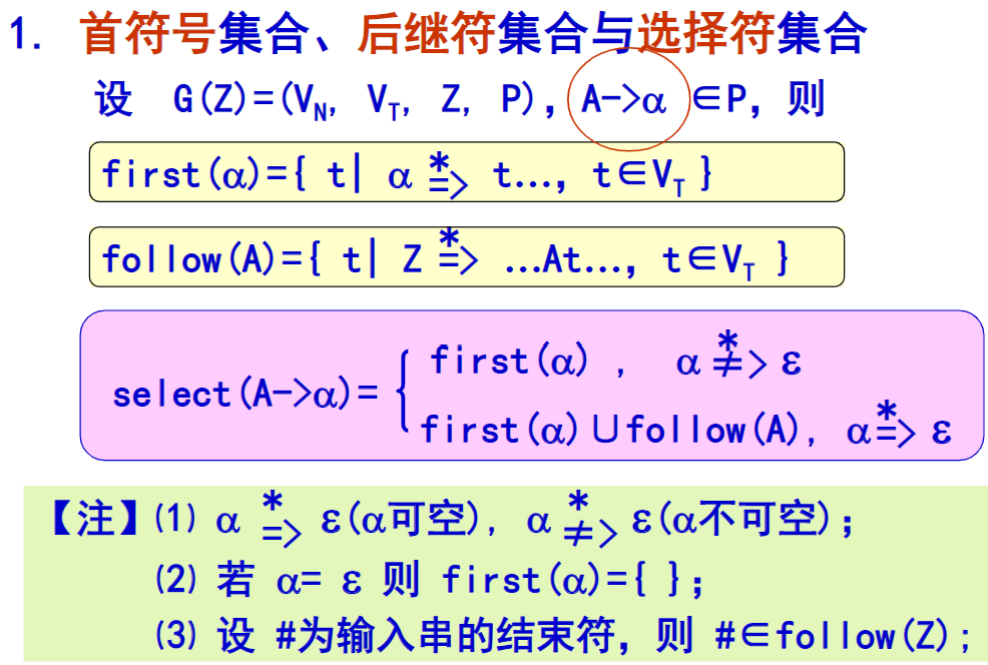
程序输出如下：

```
读入文法：
E -> T{w0 T GEQ(w0)}
F -> i PUSH(i)
F -> ( E )
T -> F{w1 F GEQ(w1)}

LL(1)文法：
E -> T E1
E1 -> w0 T GEQ(w0) E1
E1 -> @
F -> i PUSH(i)
F -> ( E )
T -> F T1
T1 -> w1 F GEQ(w1) T1
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：



本次实验中，我将GET(w0)，GET(w1)，PUSH也视为终结符。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 -> #) GEQ(w0) GEQ(w1) w0 w1 T -> #) GEQ(w0) w0 T1 -> #) GEQ(w0) w0	E -> T E1 select_set: (i E1 -> w0 T GEQ(w0) E1 select_set: w0 E1 -> @ select_set: #) F -> i PUSH(i) select_set: i F -> (E) select_set: (T -> F T1 select_set: (i T1 -> w1 F GEQ(w1) T1 select_set: w1 T1 -> @ select_set: #) GEQ(w0) w0

4.1.3 生成LL(1)分析表

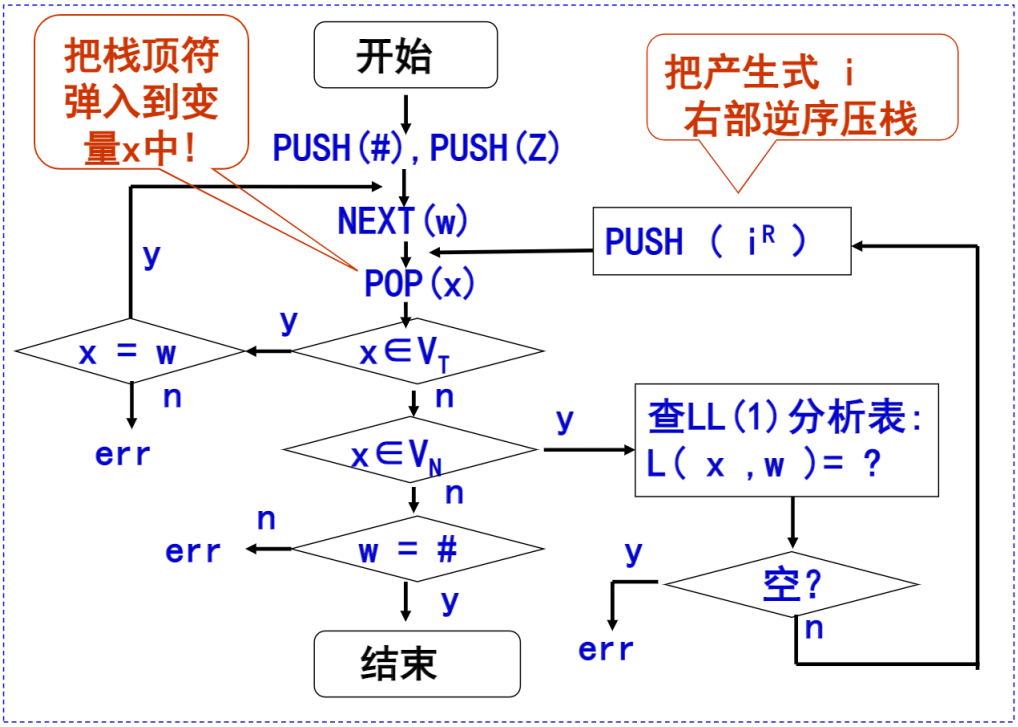
接下来根据各个非终结符的select集，生成LL(1)分析表，其中分析表每一行代表一个非终结符，每一列代表一个终结符。对于每个产生式，其左部的非终结符和其select集中的终结符构成了表格的行和列，产生式的序号就是相应单元格的内容。

```
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,GEQ(w0): 7   T1,w0: 7        T1,w1: 6
```

4.1.4 用分析表将表达式翻译为四元式

LL(1)分析法控制程序的算法如下：

II． LL(1)分析法控制程序：



在本次实验中，由于需要生成四元式，所以需要添加几个操作：

1. 当栈顶元素为PUSH(i)时，将相应的终结符推入sem栈；
2. 当栈顶元素为GEQ(w0)或GET(w1)时，将sem栈顶的两个终结符（如a, b）弹出sem栈，输出形如w a b t的四元式，并将t再推入sem栈。

根据以上算法，结合计算好的分析表，就可以实现LL(1)翻译程序。一个算术表达式实例的翻译过程如下：

SYN[n]	x	w	操作	SEM[m]	QT[q]
#E	E	a	PUSH		
#E’T	T	a	PUSH		
#E’T’F	F	a	PUSH		
#E’T’PUSH(a)a	a	a	NEXT		
#E’T’PUSH(a)		+		a	
#E’T’	T’	+	PUSH	a	
#E	E’	+	PUSH	a	
#E’GEQ(+)T+	+	+	NEXT(w)	a	
#E’GEQ(+)T	T	b	PUSH	a	
#E’GEQ(+)T	F	b	PUSH	a	
#E`GEQ(+)T`PUSH(b)b	b	b	Next(w)	a	
#E`GEQ(+)T`PUSH(b)		*		ab	
#E`GEQ(+)T`	T’	*	PUSH	ab	
#E`GEQ(+)T`GEQ(*)F*	*	*	NEXT(w)	ab	
#E`GEQ(+)T`GEQ(*)F	F	c	PUSH	ab	
#E`GEQ(+)T`GEQ(*)					
PUSH(c)c	c	c	NEXT	ab	
#E`GEQ(+)T`GEQ(*)					
PUSH(c)c		#		abc	
#E`GEQ(+)T`GEQ(*)		#		abc	(1) (*b,c,t1)
#E`GEQ(+)T`	T’	#		at1	
#E`GEQ(+)		#		at1	(2) (+a,t1,t2)
#E’	E’	#		t2	
#		#	ok	t2	

5. 实验结果

5.1 LL(1)翻译法

5.1.1 例1: a+b*c

用a+b*c作为输入样例1，可见程序输出了相应的四元式：

```
qt:
* c b t1
+ t1 a t2
```

5.1.2 例1: (a+b)*c

程序输出如下：

```
qt:
+ b a t1
* c t1 t2
```

如图所示，可见输出的四元式先计算了t1=a+b，再计算t2=t1*c，结果正确。

5.1.3 例1: a++b

这显然是一个错误的例子。程序输出如下：

```
error
input: i a w0 + w0 + i b # #
stack: T  GEQ(w0) + E1  # #
w: w0 +
```

如图所示，程序输出了error，这是由于当程序解析到第二个加号时，栈顶的元素为T，当前符号为+，而ll(1)分析表中不存在相应的文法规则，所以程序报错了。

6. 实验总结

在本次实验中，我通过自己实现一个表达式的语义分析器，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。在学习编译原理和前三次实验的过程中，我也慢慢领悟到了自动机和上下文无关文法的相关知识及其在词法分析、语法分析、语义分析中的重要性。

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