课程名称 编译技术 实验名称 实验四 中间代码生成器设计

班级 计科16-6 姓名 孟雪纯 学号 08163069 实验日期 10.30

实验报告要求：

1.实验目的 2.实验内容 3.实验要求与步骤

4.算法分析 5.运行结果 6.实验体会

一、实验目的

(1) 熟悉各种中间代码表示的方式，比较它们之间的优缺点；

(2) 掌握语法树到中间代码的转换线性处理方法；

(3) 设计符合源语言和目标语言得到综合平衡的中间语言；

(4) 属性文法和语法制导翻译法进行语义翻译。

二、实验内容

根据课堂讲授的形式化，编制程序实现一个中间代码生成器，该程序能够使用前面的词法分析器和语法分析器，完成语法树到中间代码的转换。

输入语句：

```
{
    i = 2;
    while (i <=100)
    {
        sum = sum + i;
        i = i + 2;
    }
}
```

三、实验要求

要求实现以下功能：

a) 设计比较完善的中间语言；

b) 对说明语句，要求将说明的各符号记录到相应符号表中；

c) 赋值语句的翻译，布尔表达式的翻译，过程调用的处理；

d) 对可执行语句，应产生出四元式中间代码并填写到三地址码表中；

b) 采用较为合理的结构实现生成器，简单的可以采用语法制导翻译法来实现，最好能够实现语法树直接线性化的中间代码生成，这样能够简单移植来实现其他类型的中间语言处理；

c) 在转换的过程中要进行错误处理。

四、算法分析

中间代码的形式有许多，如逆波兰式、三元式、四元式，DAG图等，本次实验以三地址代码为例。常见的三地址指令如表4.1：

表4.1 常见的三地址指令形式

指令	形式	描述
赋值指令	x = y op z	op 是双目运算符或逻辑运算符，x、y、z 是地址
	x = op y	op 是单目运算符，y 是地址
复制指令	x = y	把 y 的值赋给 x
无条件转移指令	goto L	下一步要执行的指令是带有标号 L 的三地址指令
条件转移指令	if x goto L	当 x 为真时，下一步执行带有标号 L 的指令，否则下一步执行序列中的后一条指令
	if FALSE x goto L	当 x 为假时，下一步执行带有标号 L 的指令，否则下一步执行序列中的后一条指令
	if x relop y goto L	当 x 和 y 满足 relop 关系时，下一步执行带有标号 L 的指令，否则下一步执行序列中的后一条指令
参数传递和过程调用指令	param x_1 … param x_n call p, n	“param x”进行参数传递，“call p, n”进行过程调用。“call p, n”中的 p 表示过程，n 表示参数个数。这个 n 是必须的，因为前面的一些 param 指令可能是 p 返回之后才执行的某个函数调用的参数
带下标的复制指令	x = y[i]	把距离位置 y 处 i 个内存单元的位置中存放的值赋给 x
	x[i] = y	把距离位置 x 处 i 个内存单元的位置中的内容设置为 y
地址及指针赋值指令	x = &y	把 x 的值设置为 y 的地址
	x = *y	把位置 y 中存放的值赋给 x
	*x = y	把位置 x 中的内容设置为 y

本次实验以实验三的文法为例，在实验三所实现的 LR 语法分析器的基础上实现中间代码生成。主要实现的为赋值语句、if 语句和 while 循环的中间代码生成。根据实验三所给出的文法，首先设计翻译模式，进而实现词法分析、实现 LR 语法分析，在 LR 语法分析的过程中，实现中间代码的生成。具体文法如下：

program ⟶ block
block ⟶ {stmts}
stmts ⟶ stmt stmts | ε
stmt ⟶ id = E;
|while(bool) *stmt*
|block
bool ⟶ T ≤ T
|T ≥ T
| T
E ⟶ E + T|T
T ⟶ id|num

(1) 翻译模式

program ⟶ block
block ⟶ {stmts}
stmts ⟶ stmt stmts | ε
stmt ⟶ id = E;{p ≔ lookup(id.name);
if p! = null then emit(p′ =′ E.place)
else error}
stmt ⟶ while M_1(bool) $M_2 stmt$ {$backpatch(stmt.nextline, M_1.quad)$;
$backpatch(bool.trulist, M_2.quad)$;
$stmt.nextline ≔ bool.falselist$;
$emit('goto' M_1.quad)$;}
stmt ⟶ block
bool ⟶ $T_1 \leq T_2$ {$bool.truelist ≔ makelist(nextquad)$;
bool.falselist ≔ makelist(nextquad + 1);
emit(′if′ T_1.place ≤ T_2.place ′goto—′);
eimt(′goto—′);}
bool ⟶ $T_1 \geq T_2$ {$bool.truelist ≔ makelist(nextquad)$;
bool.falselist ≔ makelist(nextquad + 1);
emit(′if′T_1.place ≥ T_2.place ′goto—′);
eimt(′goto—′);}
bool ⟶ T{bool.value: = T.value}
E ⟶ E_1 + T{E.place = newtemp;
emit(E.place′ =′ E_1.place′+′T.place)}
E ⟶ T{E.place = T.place}
T ⟶ id{p ≔ lookup($id.name$);
if p! = null then E.place ≔ p
else error}
T ⟶ num{T.place ≔ newtemp;
T.value ≔ num}

(2) LR 分析器的实现

实现过程与实验三相同。首先画出文法对应的可识别前缀的 DFA，DFA 的项目集存在移进-归约冲突，该文法用 LR(0)分析器无法进行识别，尝试用 SLR(1)分析器识别，求解得到冲突项目的 Follow 集如下：

Follow(program) = {‘#’}
Follow(block) = {‘#’, ‘}’, ‘id’, ‘while’, ‘{’}
Follow (stmts) = {‘}’}
Follow (stmt) = {‘}’, ‘id’, ‘while’, ‘{’}
Follow (bool) = {‘)’}
Follow (E) = {‘<=’, ‘>=’, ‘+’, ‘;’, ‘)’}
Follow (T) = {‘<=’, ‘>=’, ‘+’, ‘;’, ‘)’}

上述 Follow 集与其相应的移进符号集没有交集，可以判断上述文法可由 SLR(1)分析器识别。

根据该DFA构造出SLR(1)分析表。（DFA如图4.1，LR分析表如表4.2）

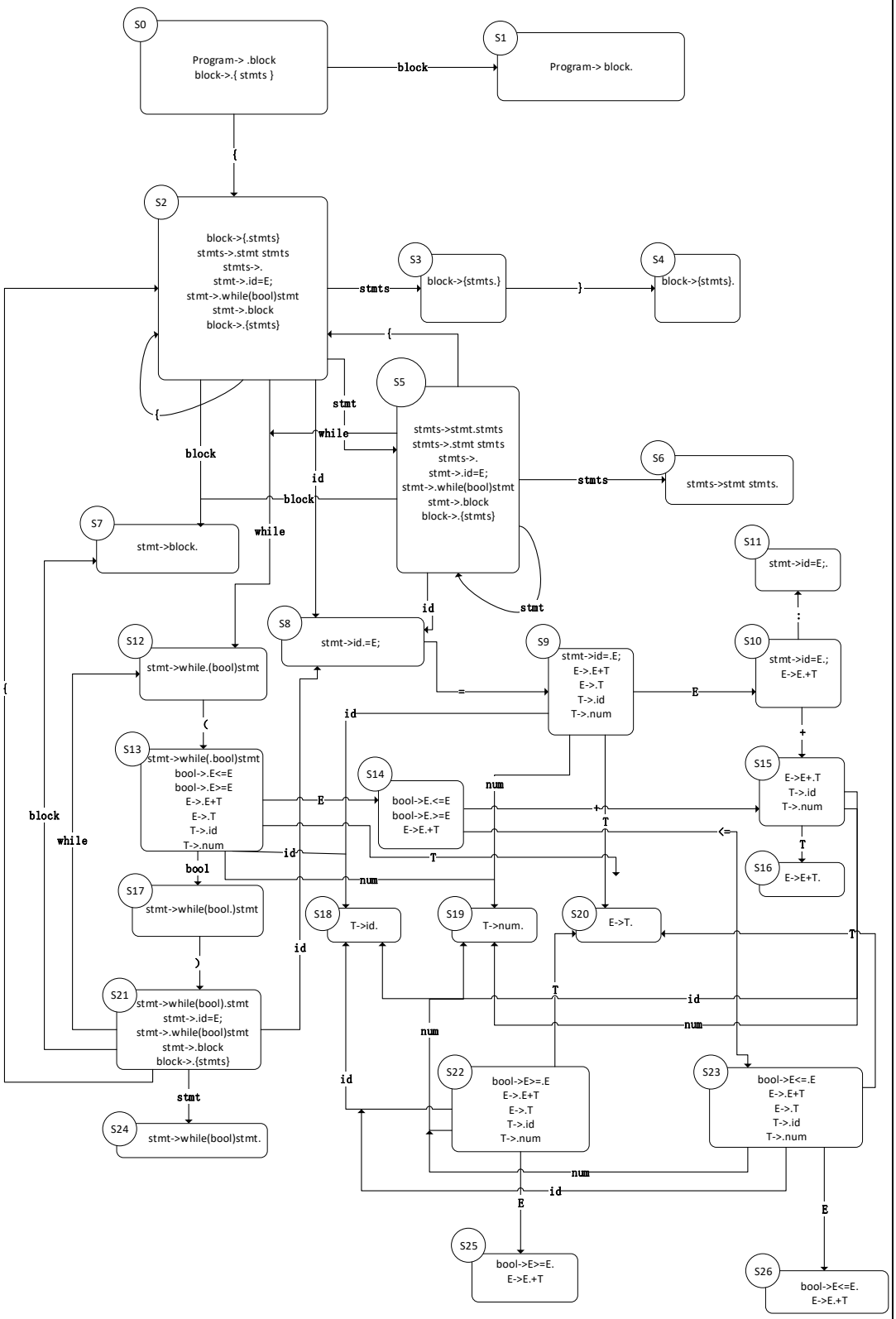

图4.1 可识别所有前缀的DFA

表 4.2 SLR(1)分析表

状态	Action												Goto					
	{	}	(	)	+	=	<=	>=	;	id	num	while	#	block	stmts	stmt	bool	E	T
S_0	S_2													1					
S_1													acc						
S_2	S_2	r_4									S_8		S_{12}	7	3	5			
S_3		r_4																	
S_4	r_2	r_2								r_2		r_2	r_2						
S_5	S_2	r_4								S_8		S_{12}		7	6	5			
S_6		r_3													S_8				
S_7	r_7	r_7								r_7		r_7	r_7						
S_8						S_9													
S_9										S_{18}	S_{19}							10	20
S_{10}					S_{15}				S_{11}										
S_{11}	r_5	r_5								r_5		r_5							
S_{12}			S_{13}																
S_{13}										S_{18}	S_{19}						17	14	20
S_{14}					S_{15}	S_{23}	S_{22}												
S_{15}										S_{18}	S_{19}								16
S_{16}				r_{10}	r_{10}		r_{10}	r_{10}	r_{10}										
S_{17}				S_{21}															
S_{18}				r_{12}	r_{12}		r_{12}	r_{12}	r_{12}										
S_{19}				r_{13}	r_{13}		r_{13}	r_{13}	r_{13}										
S_{20}				r_{11}	r_{11}		r_{11}	r_{11}	r_{11}										
S_{21}	S_2									S_8		S_{12}		7		24			
S_{22}										S_{18}	S_{19}							25	20
S_{23}										S_{18}	S_{19}							26	20
S_{24}	r_6	r_6								r_6		r_6							
S_{25}				r_9	S_{15}														
S_{26}				r_8	S_{15}														

首先向状态栈中压入（0），符号栈中压入（“#”）。令 s 为当前状态(位于分析栈的顶部)。则动作可定义如下：

（1）若状态 s 包含了格式 A →a·Xb 的任意项目，其中 X 是一个终结符，且 X 是输入串中的下一个记号，则动作将当前的输入记号移进到栈中，且被压入到栈中的新状态是包含了项目 A →aX·b 的状态。

（2）若状态 s 包含了完整项目 A →g·，则输入串中的下一个记号是在 Follow (A) 中，所以动作是用规则 A →g 归约。在所有的其他情况中，新状态都是如下计算的：删除串 a 和所有它的来自分析栈中的对应状态。相对应地，DFA 回到 a 开始构造的状态。通过构造，这个状态必须包括格式 B →g·Ab 的一个项目。将 A 压入到栈中，并将包含了项目 B →aA·b 的状态压入。

（3）若下一个输入记号都不是上面两种情况所提到的，则声明一个错误。

(3) 中间代码的生成

在 SLR 语法分析进行归约的同时，自下而上的生成中间代码。具体的算法流程图如图 4.2：

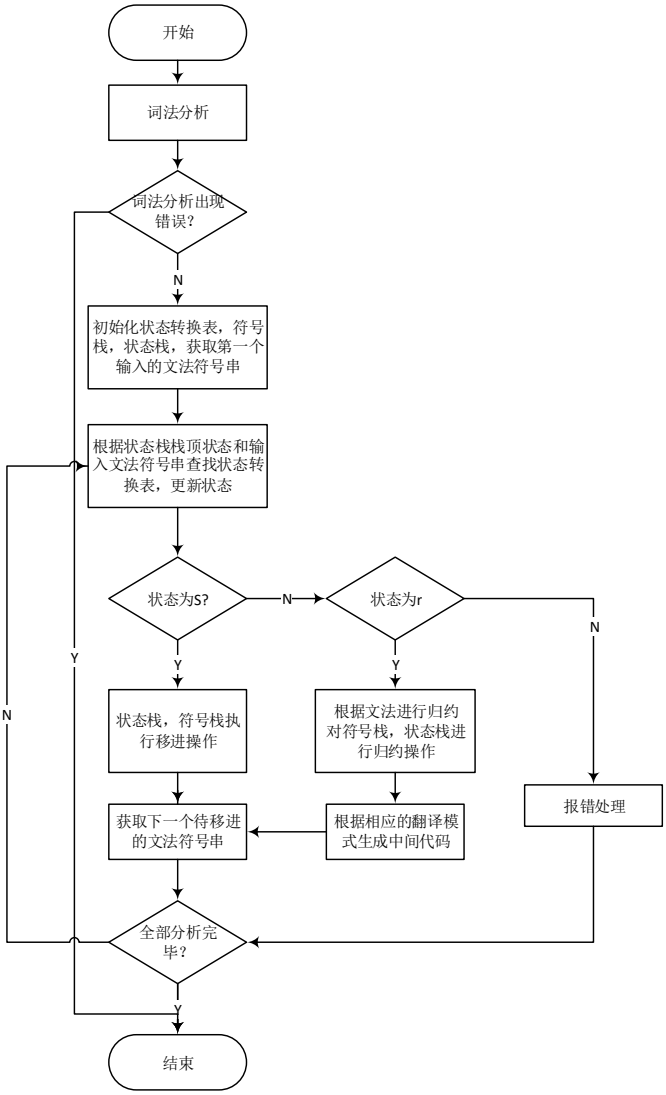

图 4.2 LR 语法分析流程图

五、实验结果

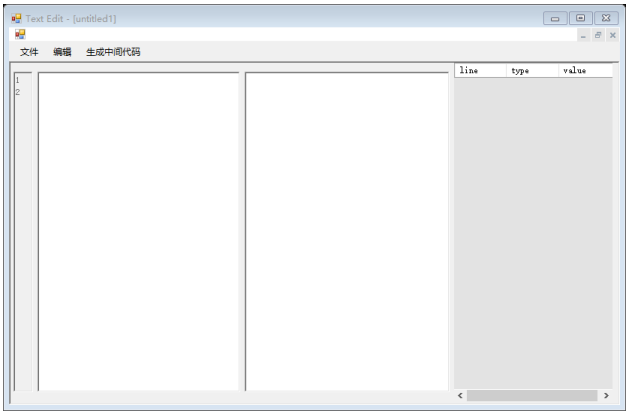

图 4.3 中间代码生成器界面

中间代码生成器界面如图 4.3 所示，左上方侧边栏用于显示行号；右侧表格用于显示词法分析的结果，包括所在行号、单词类型、单词的值；中间左侧的 richtextbox 用于编辑源程序，右侧的 richtextbox 用于显示生成的中间代码。

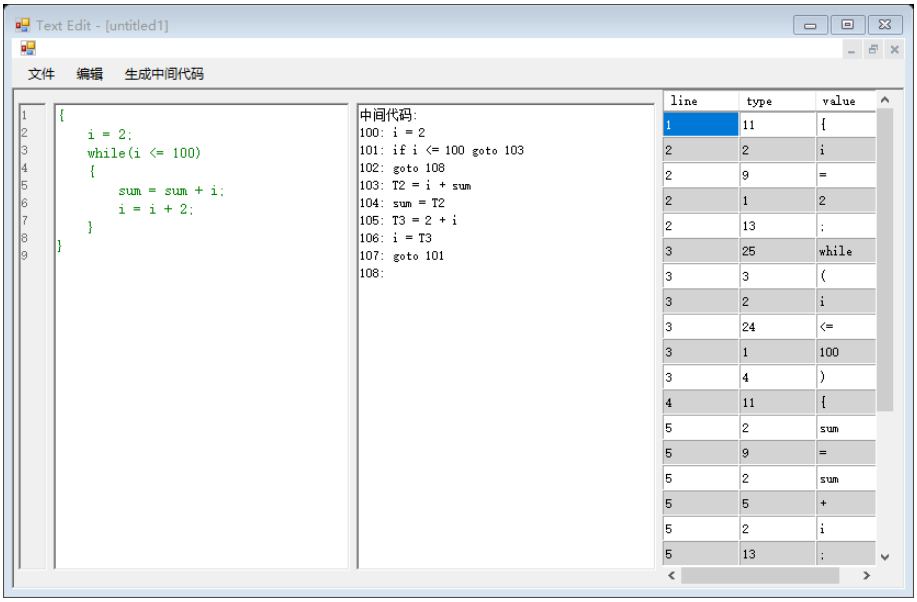

图 4.4 实验结果

生成的中间代码如图。

六、实验体会

通过本次实验，熟悉了各种中间代码表示的方式，对属性文法和语法制导翻译法进行语义翻译的过程有了更深刻的认识。能够编程实现简单的赋值语句、if判断语句以及while循环的中间代码生成。对于自顶向下和自下而上的搜索过程有了更深刻的认识，在进行遍历的同时，充分利用遍历过程中得到的许多信息，能够从中获取更多的信息，完成更加深层次的分析。