编 译 原 理

(Compiler Construction Principles)

徐金安

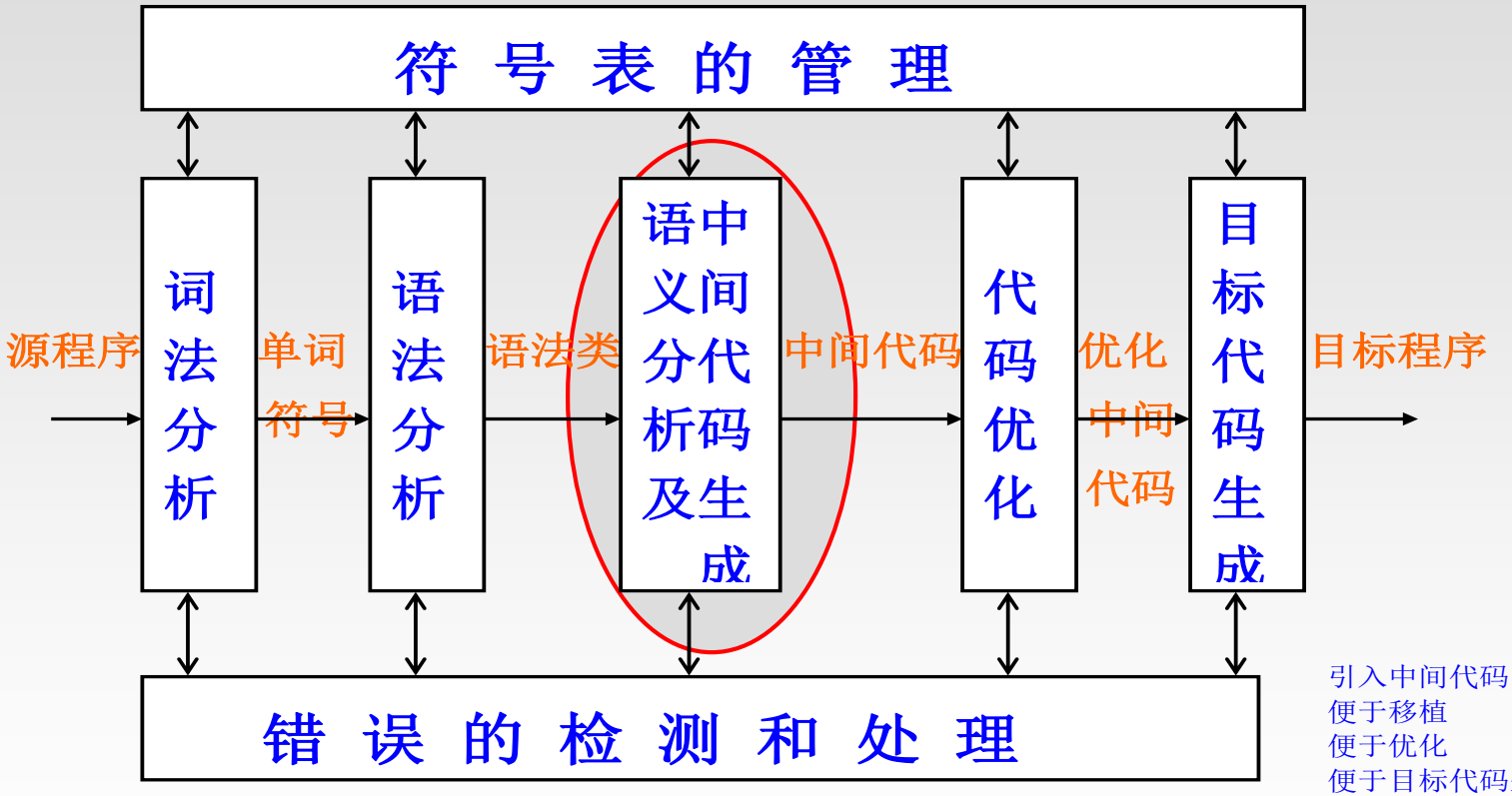
计算机与信息技术学院

计算机科学与技术系

教学目标

1. 要求明确语义分析的**任务**
2. 明确**属性文法**和**语法制导翻译**的含义
3. 明确**自底向上和自顶向下**语法制导翻译的区别和特点
4. 明确**生成中间代码的目的**，中间代码的几种形式

典型编译程序的组成



引入中间代码的目的：
便于移植
便于优化
便于目标代码生成

★ **词法分析，语法分析**：解决单词和语言成分的识别及词法和语法结构的检查。语法结构可形式化地用一组产生式来描述。给定一组产生式，我们能够很容易地将其分析器构造出来。

★ 本章要介绍的是**语义分析和中间代码生成技术**。

程序语言 —翻译→ 中间代码 —翻译→ 目标代码

5.1 引言

语义分析

一、任务：保证目标程序与源程序的语义等同

二、功能：1、静态语义检查，也称静态语义分析

2、生成某种中间代码或实际的目标代码

逻辑结构更清晰

便于移植便于优化

便于目标代码生成

不利于优化

不利于移植

代码质量不高

中间代码：含义明确，结构简单的记号系统，与具体的硬件无关，不是目标代码，却非常容易转成目标代码。

语义分析的任务

根据语义规则对识别出的各种语法成分、分析其含义，进行初步翻译，生成相应的中间代码或直接生成目标代码。

(1) 确定数据类型

(2) 语义检查

动态语义检查：在运行时刻进行

静态语义检查：在编译时完成

(3) 识别含义，进行真正的翻译

静态语义检查

①类型检查

②控制流检查

确保控制语句有合法的转向点。例如，C语言中的break语句使控制跳离包括该语句的最小的switch，while或for语句。如果不存在包括它的这样的语句，则应报错。

静态语义检查

③一致性检查。

很多情况下要求对象只能被定义一次。例如，C语言中规定一个标识符在同一作用域中只能被说明一次，同一case语句的标号不能相同，枚举类型的元素不能重复出现等。

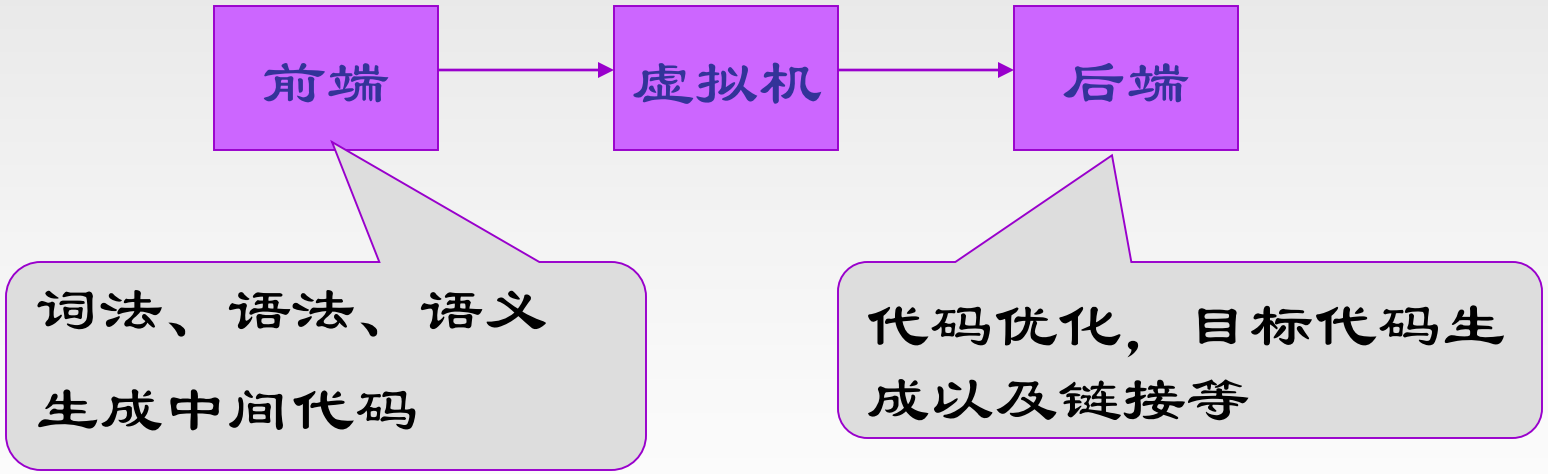
④相关名字检查。

有的语言中有时规定，同一名字必须出现两次或多次。例如，Ada语言中，循环或程序块可以有一个名字，它出现在这些结构的开头和结尾，如同语句括号一般，编译程序必须检查它们的配对情况。

5.1 引言

伪代码（虚拟机）

- 为使编译程序便于移植，应该尽量将编译程序中与机器无关的部分（前端）和与机器有关的部分（后端）分开。
- 虚拟机为前后端间的接口。



5.1 引言

三、语义分析方法研究

- 形式化方法：
  - 操作语义学
  - 公理语义学
  - 指称语义学

  符号系统比较繁杂；描述文本不易读；不能自动完成语义处理

- 接近形式化方法：语法制导翻译

  （目前大多编译器采用的方法）

5.1 引言

四、语法制导翻译　　利用翻译文法

一个翻译文法包含一个上下文无关文法和一系列语义规则（语义动作或语义子程序），这些语义规则附在文法的每个产生式上，在语法分析的同时执行这些语义规则，生成中间代码，实现语法制导翻译。

1、语义动作

- 指出了产生式符号串的意义
- 规定了生成某种中间代码应进行的基本操作
- 传送或处理信息、查填符号表、计算值、产生中间代码

5.1 引言

2、语义动作的执行方式

自上而下的语法分析：一个产生式用于推导时执行语义动作

自下而上的语法分析：一个产生式用于归约时执行语义动作

3、语法制导翻译适用于自上而下和自下而上的语法分析. 为了能及时完成语义动作，自下而上的语法分析有时需改写文法。

4、语法制导翻译既可用来产生中间代码、目标指令, 也可用于解释执行.

实际应用中比较流行的语义分析方法：

基于属性文法的语法制导翻译方法

属性文法与属性翻译文法

附加了 语义信息 （语义）规则的文法

1. 属性的表示

文法符号X的属性t常用X.t来表示

语义之间的关系

2.语义规则的表示

- 语义规则是根据产生式所蕴涵的语义操作建立起来的，并与语义分析的目标有关
- 不同的产生式对应不同的语义规则
- 不同的分析目标也对应不同的语义规则

静态语义检查、符号表操作、代码生成及打印各种错误信息

例5.1

- 非终结符E、T及F都有一个综合属性val,符号i有一个综合属性，它的值由词法分析器提供。
- 某些非终结符加下标是为了区分一个产生式中同一非终结符多次出现

产生式	语 义 规 则
$E \rightarrow E_1+T$	$E.val=E_1.val+T.val$
$E \rightarrow T$	$E.val=T.val$
$T \rightarrow T_1 * F$	$T.val=T_1.val \times F.val$
$T \rightarrow F$	$T.val=F.val$
$F \rightarrow (E)$	$F.val=E.val$
$F \rightarrow i$	$F.val=i.lexval$

语法制导翻译的过程

语法制导翻译： 将**语义规则**与**语法规则**相结合，在**语法分析**的过程中通过执行**语义动作**，计算语义属性值，从而完成预定的翻译工作。

Yacc利用的就是**语法制导翻译方法**，它使用符号$$表示产生式左端的属性，$n表示存取产生式右端第n个文法符号相联的属性

```
expr   : expr '+' expr          { $$ = $1 + $3; }
```

语法制导翻译的实现

自底向上语法制导翻译

自顶向下语法制导翻译

语法制导翻译分为两种处理方法：

（1）语法制导定义（自底向上）：

对每个产生式编制一个语义子程序，在进行语法分析的过程中，当一个产生式获得匹配时，就调用相应的语义子程序实现语义检查与翻译。这种实现方案隐藏了其中语义规则的计算次序等实现细节，不必规定翻译顺序。

（2）翻译方案（自顶向下）：

在产生式右部的适当位置，插入相应的语义动作，按照分析的进程，执行遇到的语义动作。这是一种动作与分析交错的实现方案。

翻译步骤

（1）分析输入符号串，建立分析语法树

（2）从分析树得到描述结点属性间依赖关系的依赖图，由依赖图得到语义规则的计算次序

（3）进行语义规则的计算，得到翻译结果

输入符号串

→ 分析树

→执行语义规则

→ 翻译结果

语法制导定义

对每个产生式编制一个语义子程序

在进行语法分析的过程中，当一个产生式获得匹配时，就调用相应的语义子程序实现语义检查与翻译

综合属性

自底向上传递信息

继承属性

自顶向下（自左向右）传递信息

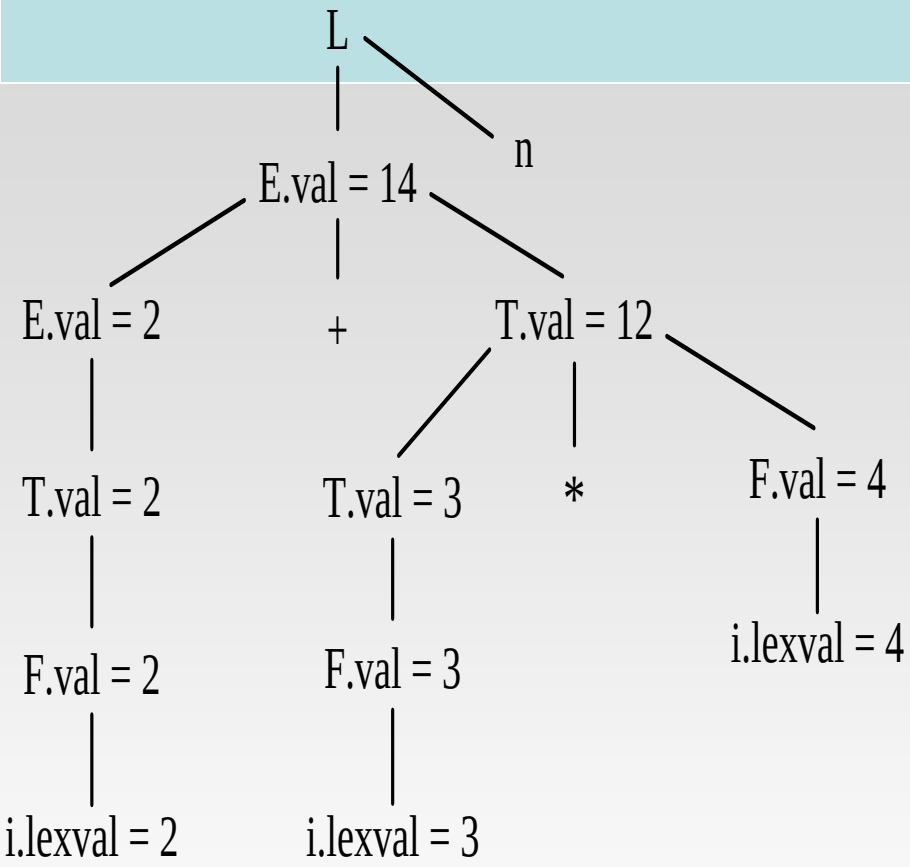
终结符：只有综合属性

非终结符：二者兼而有之

- print(E.val)打印由E产生的算术表达式的值，可认为这条规则定义了L的一个虚属性。

例5.2　综合属性

产生式	语 义 规 则
$L \rightarrow E$	print(E.val)
$E \rightarrow E_1+T$	$E.val=E_1.val+T.val$
$E \rightarrow T$	E.val=T.val
$T \rightarrow T_1 * F$	$T.val=T_1.val \times F.val$
$T \rightarrow F$	T.val=F.val
$F \rightarrow (E)$	F.val=E.val
$F \rightarrow i$	F.val=i.lexval



2 +3*4的注释分析树

- 一个结点的综合属性值是其子结点的某些属性来决定的

只含有综合属性的语法制导定义称为S属性定义

通常使用自底向上的分析方法

- 在每个结点处使用语义规则来计算综合属性值
- 当一个产生式获得匹配时，就调用相应的语义子程序
- 从叶结点到根结点进行计算

S属性定义与自底向上翻译

- LR分析器可以改造为一个翻译器，在对输入串进行语法分析的同时对属性进行计算
- LR分析器增加**属性值（语义）栈**

步　骤	状 态 栈	符 号 栈	属 性 值 栈	剩余符号串	分 析 动 作
1	0	#	−	2+3*4#	移进
2	05	#2	−	+3*4#	用F→i归约
3	03	#F	−2	+3*4#	用T→F归约
4	02	#T	−2	+3*4#	用E→T归约
5	01	#E	−2	3*4#	移进
6	016	#E+	−2−	*4#	移进
7	0165	#E+3	−2−−	*4#	用F→i归约
8	0163	#E+F	−2−3	*4#	用T→F归约
9	0169	#E+T	−2−3	*4#	移进
10	01697	#E+T*	−2−3−	4#	移进
11	016975	#E+T*4	−2−3−−	#	用F→i归约
12	01697 10	#E+T*F	−2−3−4	#	用T→T*F归约
13	0169	#E+T	−2− (12)	#	用 E→E+T 归约
14	01	#E	− (14)	#	acc

5.2 属性文法和属性翻译文法（略讲）

波兰翻译文法和属性翻译文法

1、翻译文法

文法产生式右部适当位置上插入语义动作而得到的新文法. 语义动作也称“动作符号”.

例：Expr′ →+Term {WriteCode(“+”);}Expr′

| ε

5.2 属性文法和属性翻译文法

2、波兰翻译文法

翻译文法中若每个产生式右部中的全部语义动作均出现在所有文法符号的右边.

$A \rightarrow \alpha \ \{f(\ldots\ldots);\}$ $\quad\quad\quad\quad$ $\alpha \in V^*$

5.2 属性文法和属性翻译文法

3、属性翻译文法

在一个属性翻译文法中，对应于每个产生式$A \rightarrow \alpha$都有一套与之相关的语义规则，每条语义规则的形式为

$b:=f(C_1, C_2 \ldots\ldots, C_k)$

这里，f是一个函数，而且或者

（1）b是A的一个综合属性并且$C_1, C_2 \ldots\ldots, C_k$是产生式右边文法符号的属性；或者

（2）b是产生式右边某个文法符号的一个继承属性并且$C_1, C_2 \ldots\ldots, C_k$是A或者产生式右边任何文法符号的属性；

在这两种情况下都说属性b依赖于$C_1, C_2 \ldots\ldots, C_k$。

5.2 属性文法和属性翻译文法

3、属性翻译文法(续)

- 属性可以计算和传递.
- 综合属性用于“自下而上”传递.
- 继承属性用于“自上而下”传递.

5.2 属性文法和属性翻译文法

说明

（1）终结符号只有综合属性，它们由词法分析程序提供；

（2）非终结符号既可以有综合属性，也可以有继承属性，文法开始符号的所有继承属性作为属性计算前的初始值；

一般

（1）对出现在产生式右边的继承属性和出现在产生式左边的综合属性都必须提供一个计算规则；

（2）属性计算规则中只能使用相应产生式中的文法符号的属性，这有助于在产生式范围内“封装”属性的依赖性；

5.2 属性文法和属性翻译文法

（3）出现在产生式左边的继承属性和出现在产生式右边的综合属性不由所给的产生式的属性计算机规则计算，由其它的产生式的属性计算机规则计算或由属性计算器的参数给出；

（4）语义规则所描述的工作包括属性计算、静态语义检查、符号表操作、代码生成等；

处理方法： 输入串→语法树→依赖图→语义规则计算次序

实现方式: S-属性文法自下而上计算 L-属性文法自上而下计算

以下采用波兰翻译文法实现语法制导翻译

5.2 属性文法和属性翻译文法

一个简单台式计算器的属性文法

产生式	语义规则
$L \rightarrow En$	Print(E.val)
$E \rightarrow E_1+T$	$E.val:=E_1.val+T.val$
$E \rightarrow T$	$E.val:=T.val$
$T \rightarrow T_1*F$	$T.val:=T_1.val*F.val$
$T \rightarrow F$	$T.val:=F.val$
$F \rightarrow (E)$	$F.val:=E.val$
$F \rightarrow digit$	$F.val:=digit.lexval$

5.3 中间代码

一、逆波兰------主要用于表达式

1、表达式的逆波兰表示

后缀式: $e_1e_2\ldots\ldots e_k\theta$　　θ是k目运算符(k>=1)

特点:运算量在前, 运算符在后, 无括号

中缀式	逆波兰
a+b*c	a bc* +
a*(b+c/d)	a b cd/ + *
(a+b)*(c+d)	ab+ cd+ *

5.3 中间代码

2、后缀式的计值:利用分析栈

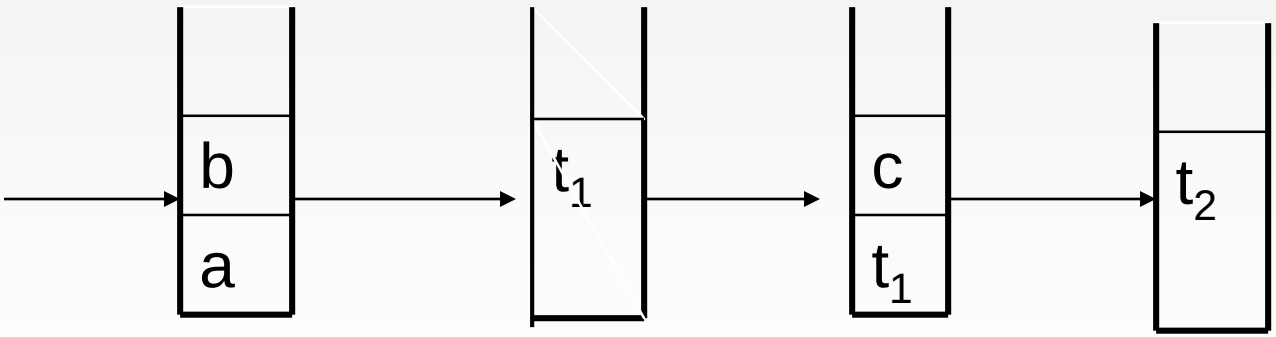
计算步骤:自左向右扫描后缀式

遇到运算量进栈

遇到k目运算符则作用于栈顶的k项

用运算结果代替这k项.

计算ab+c*



5.3 中间代码

3、后缀式的推广

(1)赋值语句的后缀式

a:=e　　逆波兰：ae:=

a:=3*(b+c)　逆波兰　a3bc+*:=

(2)条件语句的后缀式

例：if e then x else y

将if—then —else 看成三目运算符￥

逆波兰:exy ￥

缺点:x,y 都被计值

5.3 中间代码

二、三元式

1、三元式

（1）表示形式 （op，ARG1，ARG2）

其中：op：运算符

ARG1:第一运算量

ARG2:第二运算量

- 表达式及各种语句都可以表示成三元式
- 三元式出现的顺序为表达式的计值顺序
- 一目运算任选ARG1或ARG2

5.3 中间代码

例：A+B*C的三元式

	op	ARG1	ARG2
(1)	*	B	C
(2)	+	A	(1)

ARG1和 ARG2： 指示器，指向符号表的某一位置或 三元式表自身的某一项。

5.3 中间代码

例 X:=a*b+c

	op	ARG1	ARG2
(1)	*	a	b
(2)	+	(1)	c
(3)	:=	X	(2)

例 (A∧B)∨ (¬C ∨D)

	op	ARG1	ARG2
(1)	∧	A	B
(2)	¬	C	
(3)	∨	(2)	D
(4)	∨	(1)	(3)

5.3 中间代码

2、间接三元式　　便于优化

用一张间接码表辅以三元式表来表示中间代码.

间接码表按运算的先后顺序列出有关的三元式

在三元式表中的位置.

例:x:=(a+b)*c

y:=d*(a+b)

5.3 中间代码

不易于优化

比较易于优化

例:x:=(a+b)*c ; y:=d*(a+b)

一般三元式:

	op	ARG1	ARG2
(1)	+	a	b
(2)	*	(1)	c
(3)	:=	x	(2)
(4)	+	a	b
(5)	*	d	(4)
(6)	:=	y	(5)

间接三元式

	op	ARG1	AGR2
(1)	+	a	b
(2)	*	(1)	c
(3)	:=	x	(2)
(4)	*	d	(1)
(5)	:=	y	(4)

间接码表

(1)(2)(3)(1)(4)(5)

5.3 中间代码

三、四元式

1、表示 (op, ARG1,ARG2,RESULT)

其中:ARG1, ARG2, RESULT(运算结果):

用户定义的变量或引进的临时变量,

指向符号表的入口位置.

2、说明

①一目运算用ARG1

②四元式出现的顺序与表达式的计值顺序一致.

③临时变量:可以象变量名一样填入符号表.

④四元式之间通过临时变量联系,易于优化.

5.3 中间代码

例:A:= – B*(C+D)

	op	ARG1	ARG2	RESULT
(1)	@	B	_	T_1
(2)	+	C	D	T_2
(3)	*	T_1	T_2	T_3
(4)	:=	T_3	_	A

其中:T_i 是临时变量

易于优化

5.4 自下而上分析法：语句的翻译

一、语句的翻译

1、语句：各种语句均有语义动作

①可执行语句-----生成四元式

②不可执行语句---填写符号表

2、语句翻译的构思要点

①确定语句的目标代码结构

②中间代码

③添加语义动作

3、数据结构：符号表，四元式表，临时变量表

二、赋值语句和表达式到四元式的翻译

1、定义

- NEWTEMP:

  函数过程，产生新的临时变量$T_1, T_2, \ldots\ldots$。

- X .PLACE:

  与非终结符号X相关的语义变量，表示存放X值的变量名在符号表的入口（序号）或是一个整数码，或临时变量的编号。

- GEN(op, ARG1, ARG2, RESULT):

  语义过程，把四元式填入符号表中。

- ENTRY(id):语义过程，查找标识符id在符号表中的位置

2、文法

S→A

A →V:=E

E →E+T | T

T → T*F | F

F →(E) | i

V →i

此文法为SLR(1)文法,构造SLR(1)分析表.

3、产生式及语义动作

(1) S→A

{ }

(2) A →V:=E

{GEN(:=, E ·PLACE, _, V ·PLACE) }

(3) $E^{(1)}$ →$E^{(2)}$+T

{$E^{(1)}$ ·PLACE:=NEWTEMP;

GEN(+, $E^{(2)}$ ·PLACE, T ·PLACE, $E^{(1)}$ ·PLACE) }

(4)E → T

{E ·PLACE:=T ·PLACE}

(5) $T^{(1)} \rightarrow T^{(2)}*F$

{$T^{(1)}$ .PLACE:=NEWTEMP;

GEN(+, $T^{(2)}$ .PLACE, F .PLACE, $T^{(1)}$ .PLACE)

(6) $T \rightarrow F$

{T .PLACE:=F .PLACE}

(7) $F \rightarrow (E)$

{F .PLACE:=E .PLACE}

(8) $F \rightarrow i$

{F .PLACE:=ENTRY(i)}

(9)$V \rightarrow i$

{V .PLACE:=ENTRY(i)}

例:分析句子X:=B+C*D

步骤	状态栈	符号栈	输入	动作	四元式
1	0	#	X	S_2	
2	02	#X	:=	R_9	
3	03	#V^X	:=	S_4	
4	034	#V^X:=	B	S_9	
5	0349	#V^X:=B	+	R_8	
6	0347	#V^X:=F^B	+	R_6	
7	0346	#V^X:=T^B	+	R_4	
8	0345	#V^X:=E^B	+	S_{10}	
9	034510	#V^X:=E^B+	C	S_9	

步骤	状态栈	符号栈	输入	动作	四元式
10	0345109	#V^X:=E^B+C	*	R_8	
11	0345107	# V^X:=E^B+F^C	*	R_6	
12	03451013	#V^X:=E^B+T^C	*	S_{11}	
13	0345101311	#V^X:=E^B+T^C*	D	S_9	
14	03451013119	#V^X:=E^B+T^C*D	#	R_8	
15	03451013117	#V^X:=E^B+T^C*F^D	#	R_5	(*,C,D,T_1)
16	03451013	#V^X:=E^B+T^{T1}	#	R_3	(+,B,T_1,T_2)
17	0345	# V^X:=E^{T2}	#	R_2	(:=,T_2,_,X)
18	01	#A	#		

5.5 作为条件控制语句的布尔式的翻译

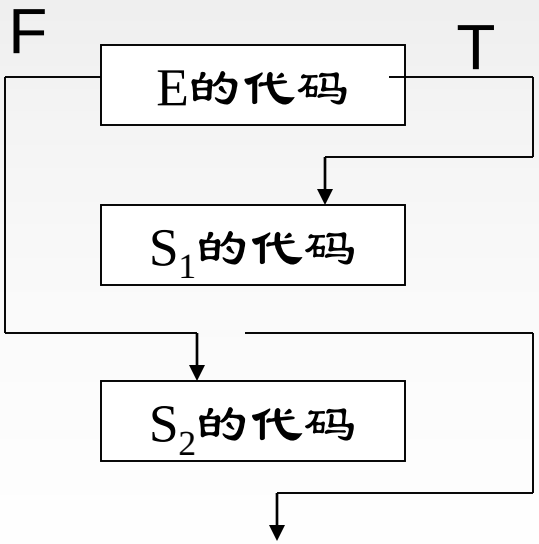
1、布尔式的作用

if E then S_1 else S_2;

E的作用:控制对S_1和 S_2的选择,找到出口,E值不留.

E的出口: 真出口----- S_1语句

假出口----- S_2语句



2、布尔式的源文法

G[E]:

E→E∧E

E→E∨E

E→^E

E→(E)

E→i rop　i

E→i

其中：rop为关系运算符，>,<,=, ≧,≦,≠

思考：JNZ：？

JEZ：？

3、布尔式的四元式序列形式

(jnz, A, _ , p): 若A为真转第p个四元式　　非0转运算符

(jez, A, _ , p): 若A为假转第p个四元式　　0转运算符

(jrop ,A_1,A_2,p): 若的A_1 rop A_2关系为真转第p个四元式　　关系转运算符

(j,_ , _ , p): 无条件转第p个四元式　　无条件转运算符

任何复杂布尔表达式均可由上述4个基本要素组合而成

例:if A∨B<D then S_1 else S_2的四元式序列

(1) (jnz, A, _ ,5) A的真出口5

(2) (j, _, _ , 3) A的假出口3

(3) (j<, B, D, 5) B的真出口5

(4) (j,_, _, p+1) B的假出口p+1

(5) (关于S_1的四元式序列) E的真出口

(p) (j , _, _, q)

(p+1) (关于S_2的四元式序列) E的假出口

(q)…… 问题：如何确定布尔式的真、假出口？

如何确定目标地址？

讨论:

(1)如何确定布尔式的真、假出口.

一般:

①$E \rightarrow E^{(1)} \vee E^{(2)}$ ($E^{(1)}$为真,则E为真)

$E^{(1)}$的真出口,就是E的真出口

$E^{(1)}$为假,计$E^{(2)}$,

$E^{(1)}$的 假出口,就是$E^{(2)}$的第一个四元式

$E^{(2)}$的真假出口,就是E的真假出口

②$E \rightarrow E^{(1)} \wedge E^{(2)}$　($E^{(1)}$为假,则E为假)

$E^{(1)}$的假出口,就是E的假出口

$E^{(1)}$为真,计$E^{(2)}$,

$E^{(1)}$的 真出口,就是$E^{(2)}$的第一个四元式

$E^{(2)}$的真假出口,就是E的真假出口

③$E \rightarrow \wedge E^{(1)}$

$E^{(1)}$的真出口,就是E的假出口

$E^{(1)}$的假出口,就是E的真出口

(2)如何确定目标地址

①拉链---回填技术:

自下而上的语法分析中,一个 布尔式的真假出口往往不能在产生四元式的同时填上,需要将这些未完成的四元式地址作为语义值保留下来,待到整个表达式的四元式产生后回来填这个未填的转移目标.

②对E设两个语义值:

E.TC:表达式E所对应的四元式需回填真出口的四元式的地址所构成的链的链首.

E.FC:表达式E所对应的四元式需回填假出口的四元式的地址所构成的链的链首.

例:p,q,r三个四元式需回填同一真出口的目标地址

(p) (X, X, X, 0)

(q) (X, X, X, 0)

(r) (X, X, X, 0)

拉链过程

(p) (X, X, X, 0)

E.TC=p　链尾=p

(p) (X, X, X, 0)

（q)(X, X, X, p)

E.TC=q　链尾=p

(p)(X, X, X, t)

(q)(X, X, X, t)

(r)(X, X, X, t)

E.TC=r　链尾=p

回填过程

设目标地址为t

③定义

- NXQ:指针,指向下一个将要产生但尚未产生的四元式的地址.每执行一次GEN过程,NXQ值自动加1.
- MERGE(P_1,p_2):函数.将以p_1为首和p_2为首的两条链合并,返回合并后的链首

链首 $\begin{cases} P_1 & p_2=0 \\ p_2 & p_2\neq 0 \end{cases}$

Procedure MERGE(P_1,p_2)

if p_2=0 then MERG:=p_1 else

begin p:= p_2 ;

while 四元式的第四区段不为0 do

p:=四元式的第四区段内容;

把p_1填入四元式p的第四区段;

MERGE:= p_2;

end

- BACKPATCH(p,t):回填过程,把p所链接的每个四元式的第四区段都填为t.

```
Procedure   BACKPATCH(P,t)
   begin
      Q:=p;
      while   Q≠0    do
         begin
           q:=四元式Q的第四区段内容;
           把t 填入四元式Q的第四区段;
           Q:=q
         end
    end
```

改写后的文法

G[E]: $E \rightarrow E^A E$

$E^A \rightarrow E \wedge$

$E \rightarrow E^0 E$

$E^0 \rightarrow E \vee$

$E \rightarrow \hat{} E$

$E \rightarrow (E)$

$E \rightarrow i\ rop\ \ i$

$E \rightarrow i$

5、产生式及语义动作

(1)E→i

{E·TC:=NXQ; E·FC:=NXQ+1;

GEN(jnz, ENTRY(i), 0 , 0);

GEN(j , _, _, 0)}

(2) E→$i^{(1)}$ rop $i^{(2)}$

{E·TC:=NXQ; E·FC:=NXQ+1;

GEN(jrop, ENTRY($i^{(1)}$), ENTRY($i^{(2)}$), 0);

GEN(j , _, _, 0)}

(3) $E \rightarrow (E^{(1)})$

{E·TC:= $E^{(1)}$ ·TC; E·FC:= $E^{(1)}$ ·FC}

(4) $E \rightarrow \wedge E^{(1)}$

{E·TC:= $E^{(1)}$ ·FC; E·FC:= $E^{(1)}$ ·TC}

(5)$E^{A} \rightarrow E^{(1)}$ ∧

{BACKPATCH($E^{(1)}$ ·TC , NXQ); //回填t

E^{A} ·FC:= $E^{(1)}$ ·FC}

(6) $E \rightarrow E^{A}E^{(2)}$

{E·TC:= $E^{(2)}$ ·TC;

E·FC:=MERG(E^{A} ·FC, $E^{(2)}$ ·FC} //合并链首

(7) $E^0 \rightarrow E^{(1)}$ v

{BACKPATCH($E^{(1)}$ ·FC , NXQ);

E^0 ·TC:= $E^{(1)}$ ·TC}

(8) E → $E^0E^{(2)}$

{E·FC:= $E^{(2)}$ ·FC;

E·TC:=MERG(E^0 ·TC, $E^{(2)}$ ·TC}

最后E ·TC和 E ·FC尚待回填.

5.6 控制语句的翻译

一、文法

G[L]:L →L;S ｜S

S →if E then S(1) ｜if E then S (1) else S (2)

｜while E do S (1)

｜begin L end

｜A

L:语句串　S:语句　A：赋值语句　E:布尔表达式

二、目标代码结构分析

5.6 控制语句的翻译

三、改写后的文法

$$L \rightarrow L^{S} S \mid S$$

$$S \rightarrow CS^{(1)} \mid T^{P} S^{(2)}$$

$$\mid W^{d} S^{(1)}$$

$$\mid \textbf{begin}\ L\ \textbf{end}$$

$$\mid A$$

$$C \rightarrow \textbf{if}\ E\ \textbf{then}$$

$$T^{P} \rightarrow CS^{(1)}\ \textbf{else}$$

$$W^{d} \rightarrow W\ E\ \textbf{do}$$

$$W \rightarrow \textbf{while}$$

$$L^{S} \rightarrow L;$$

5.6 控制语句的翻译

四、产生式及语义动作

(1) C →if E then
{BACKPATCH(E.TC,NXQ); C.CHAIN=E.FC}

(2) T^P →$CS^{(1)}$ else
{q=NXQ; GEN(j,_,_,0);
BACKPATCH(C.CHAIN,NXQ);
T^P.CHAIN=MERGE($S^{(1)}$.CHAIN,q)}

(3) S → T^P $S^{(2)}$
{S.CHAIN=MERGE(T^P.CHAIN,$S^{(1)}$.CHAIN)}

(4) S →$CS^{(1)}$
{S.CHAIN=MERGE(C.CHAIN,$S^{(1)}$.CHAIN)}

5.6 控制语句的翻译

四、产生式及语义动作

(5) W→while
{W .QUAD=NXQ}

(6) W^d →W E do
{BACKPATCH(E.TC,NXQ);
W^d.CHAIN=E.FC;
W^d .QUAD=W.QUAD}

(7) S → W^d $S^{(1)}$
{BACKPATCH($S^{(1)}$.CHAIN,W^d.QUAD);
GEN(j,_,_, W^d.QUAD);
S.CHAIN=$S^{(1)}$.CHAIN}

5.6 控制语句的翻译

四、产生式及语义动作

(8) S→begin L end
{S .CHAIN=L.CHAIN}

(9) S →A
{S.CHAIN=0}

(10) L → L^s $S^{(1)}$
{L.CHAIN= $S^{(1)}$.CHAIN}

(11) L → S
{L.CHAIN= S.CHAIN}

(12) L^s → L;
{BACKPATCH(L.CHAIN,NXQ}

5.7 说明语句的翻译

一、一般说明语句

1、文法

D→integer namelist ｜real namelist

namelist →namelist , i ｜i

例：integer A,B,C

语义动作：在符号表中添加标识符的属性信息。

文法缺点：所有标识符全部归约为namelist时才能填写符号表。

2、改写文法

D→D, i | integer i | real i

3、语义动作

定义：

- D .ATT:语义变量，记录说明语句标识符的性质。
- FILL(P,A):过程，把性质A填入P所指的符号表入口有关区段（栏）中。

(1) D→integer　i

{FILL(ENTRY(i),int); D ·ATT=int}

(2) D→real　i

{FILL(ENTRY(i),real); D ·ATT=real}

(3) D→$D^{(1)}$,i

{FILL(ENTRY(i), $D^{(1)}$ ·ATT); D ·ATT= $D^{(1)}$ ·ATT}

作 业（自己完成）

- P253

  5-3、5-4、5-5

  5-6、5-8

例题13

- 将下列语句翻译成四元式序列

if(A<C) and (B<D) then

if A= 1 then C:=C+1

else if A≤D then A:=A+2

例题13(解答)

(1)(J<,A,C,3)　拉链回填

(2)(J,_,_,14)　14-11-9-4-2

(3)(J<,B,D,5)　7-5-3

(4)(J<,_,_,14)　14-11-9-4

(5)(J=,A,1,7)　7-5

(6)(J,_,_,10)　12- 10-6

(7)(+,c,1,T1)

(8)(=,T1,_,c)

(9)(J,_,_,14)　11-9

(10)(J≤,A,D,12)　12-10

(11)(J,_,_,14)　14-11

(12)(+,A,2,T2)

(13)(=,T2,_,A)

(14)

已知if语句的执行过程(图)