编译原理

北方工业大学信息学院
School of Information Science and Technology,
North China University of Technology
束劼
shujie@ncut.edu.cn
瀚学楼1122，88801615

第七章 语义分析和中间代码产生

第七章 语义分析和中间代码产生

- 本章目录
- 7.1 中间语言
- 7.2 赋值语句的翻译
- 7.3 布尔表达式的翻译
- 7.4 控制语句的翻译
- 7.5 过程调用的处理

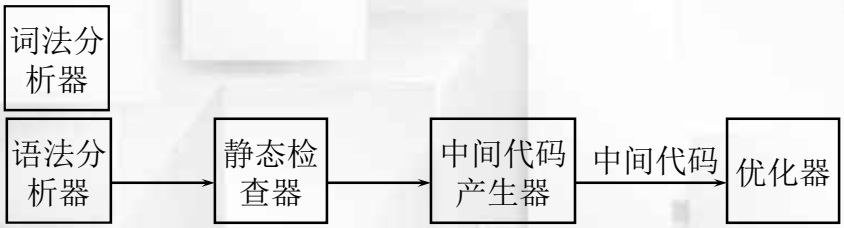
第七章 语义分析和中间代码产生

- 大纲要求
- 掌握：重点掌握两种中间语言：后缀式、三地址代码，掌握赋值语句的翻译、布尔表达式的翻译、控制语句的翻译、过程调用等语句翻译及中间代码生成方法。
- 理解：三地址中间语言的语法；语法制导定义与翻译模式的理解。
- 了解：DAG图、三地址代码的存储形式。

前言

静态语义检查和翻译

- 静态语义检查和翻译



静态语义检查和翻译

- 静态语义检查和翻译
  - ① 类型检查
  - ② 控制流检查
  - ③ 一致性检查
  - ④ 相关名字检查
  - ⑤ 名字的作用域分析

静态语义检查和翻译

- 静态语义检查和翻译
  - ① 类型检查

    验证操作数类型是否相容

  - ② 控制流检查

    控制流语句必须使控制转移到合法的地方。例如，在C语言中break语句使控制跳离包括该语句的最小while、for或switch语句。如果不存在包括它的这样的语句，则就报错。

静态语义检查和翻译

- 静态语义检查和翻译

③ 一致性检查

在很多场合要求对象只能被定义一次。例如Pascal语言规定同一标识符在一个分程序中只能被说明一次，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等。

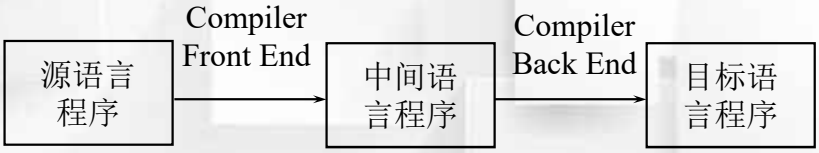
④ 相关名字检查

如Ada语言中，循环或程序块的名字，出现在结构的开头和结尾，必须检查同一名字两次等。

7.1 中间语言

7.1中间语言

- 中间语言Intermediate Languare
  独立于机器的的语言，其复杂性在源语言和目标语言之间。



11

7.1中间语言

- 中间语言Intermediate Languare

使用中间语言的好处：

① 便于进行与机器无关的代码优化工作

② 易于移植

③ 使编译程序的结构在逻辑上更为简单明确

12

7.1中间语言

7.1.1 后缀式

Postfix Expressions

7.1.2 图表示法

Directed Acyclic Graphs for Expressions

7.1.3 三地址代码

Three-Address Code

7.1.1 后缀式

7.1.1 后缀式

- 后缀式表示法 Postfix Expressions
  波兰逻辑学家卢卡西维奇发明的一种表示表达式的方法，又称**逆波兰**表示法。

- 一个表达式E的后缀形式可以如下定义：
  1. 如果E是一个变量或常量，则E的后缀式是E自身。
  2. 如果E是E_1 op E_2形式的表达式，其中op是任何二元操作符，则E的后缀式为E_1' E_2' op，其中E_1'和E_2'分别为E_1和E_2的后缀式。
  3. 如果E是(E_1)形式的表达式，则E_1的后缀式就是E的后缀式，括弧不写入后缀式。

7.1.1 后缀式

- 后缀式表示法 Postfix Expressions

例如：abc + *　　a * (b + c)

ab + cd + *　　(a+b) * (c + d)

后缀式没有括号

7.1.1 后缀式

- 把正常运算式变为后缀式的写法技巧

  按四则运算法则，挨个把运算式子变为后缀式，注意符号的顺序不能变。

  例如： a * (b + c)　　b + c 变为 b c +，设b+c的结果T_1

  a * T_1 变为 aT_1 *, 把T_1代入，得a b c + *

  (a+b) * (c + d)　　a + b 变为 a b +，设a+b的结果T_1

  c + d 变为 c d +，设c+d的结果T_2

  T_1 * T_2 变为 T_1 T_2 *, 把T_1、T_1代入，得a b + c d + *

7.1.1 后缀式

- 把表达式翻译成后缀式的语义规则描述

产生式	语义动作
$E \rightarrow E_1$ op E_2	E.code:= E_1.code ‖ E_2.code ‖ op
$E \rightarrow (E_1)$	E.code:= E_1 .code
E→id	E.code:=id

E.code表示E后缀形式
op表示任意二元操作符
“‖”表示后缀形式的连接

7.1.2 图表示法

7.1.2 图表示法

- DAG图表示法

  Directed Acyclic Graphs for Expressions

① 跟语法树相似，但DAG的叶子结点是最原子的操作数，内部结点是运算符号；

② 如果结点N是公共子表达式，则DAG中的一个结点N有多个父结点；

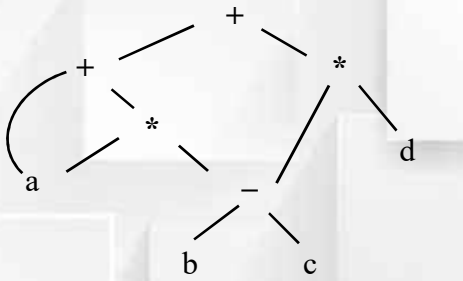
③ 比语法树更简洁，因为语法树中表示公共子表达式时，会重复多次；

7.1.2 图表示法

- DAG图表示法

  例：用DAG表示下述表示式

  a + a * (b – c) + (b – c) * d



7.1.2 图表示法

- DAG图表示法

  例：用语法制导翻译生成语法树或DAG

产生式	语义规则
$E \rightarrow E_1 + T$	E.*node* = new Node('+', E_1.*node*, T.*node*)
$E \rightarrow E_1 - T$	E.*node* = new Node('–', E_1.*node*, T.*node*)
$E \rightarrow T$	E.*node*→T.*node*
$T \rightarrow (E)$	T.*node*→ E .*node*
$T \rightarrow id$	T.*node*→new Leaf(id, id.*entry*)
$T \rightarrow num$	T.*node*→new Leaf(num, num.*entry*)

7.1.2 图表示法

- 用语法制导翻译生成语法树或DAG

p_1 = Leaf(id, entry-a)
p_2 = Leaf(id, entry-a) = p_1
p_3 = Leaf(id, entry-b)
p_4 = Leaf(id, entry-c)
p_5 = Node('–', p_3 , p_4)
p_6 = Node('*', p_1 , p_5)
p_7 = Node('+', p_1 , p_6)
p_8 = Leaf(id, entry-b) = p_3
p_9 = Leaf(id, entry-c) = p_4

a + a * (b – c) + (b – c) * d

先变为后缀式：a a b c– * + b c – d* +
然后从左往右依次输入并执行产生式

p_{10} = Node('–', p_3 , p_4)
p_{11} = Leaf(id, entry-d)
p_{12} = Node('*', p_5 , p_{11})
p_{13} = Node('+', p_7 , p_{12})

23

7.1.3 三地址代码

7.1.3 三地址码

- 三地址代码

  一般形式：x := y op z

  x, y, z是名字、常数或编译时产生的临时变量

  op是运算符。

7.1.3 三地址码

- 三地址代码
  例子：

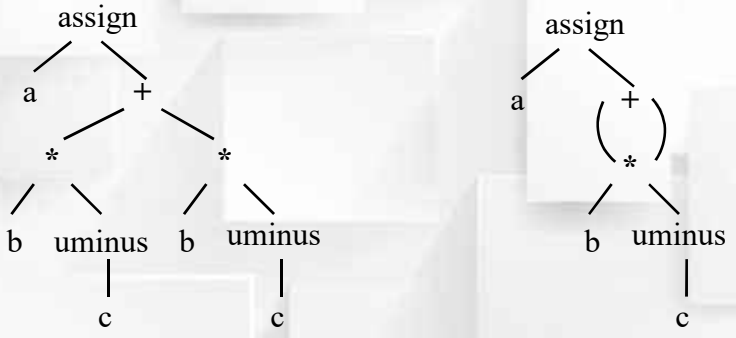
  x + y * z 翻译成：

  t_1 := y * z
  t_2 := x + t_1

  其中t_1，t_2 为编译时产生的临时变量

7.1.3 三地址码

- 三地址代码
  例子：画出a := b* – c + b * – c 的语法树和DAG图。



后缀式： a b c uminus * b c uminus * + assign

27

7.1.3 三地址码

- 三地址代码的9种类型
  ① x:=y *op* z　　赋值语句, *op*是二元算术算符或逻辑算符

  ② x:=*op* y　　赋值语句, *op*是一元算符

  ③ x:=y　　复写语句

  ④ *goto* L　　无条件转移语句， L是下一步要执行的三地址语句

28

7.1.3 三地址码

- 三地址代码的类型
  - ⑤ if x *goto* L或if False x *goto* L

    有条件转移语句，如果x为true或false时执行语句L
  - ⑥ if x relop y *goto* L (relop = rel op)

    有条件转移语句，根据x与y的关系为true或false时执行语句L，关系包括：<, ==, >=, etc.

7.1.3 三地址码

- 三地址代码的类型
  - ⑦ param x和call p, n，以及返回语句return y

    过程和函数调用语句，param x是参数，call p, n 是过程调用，y = call p, n是函数调用。Return y是可选的返回值。

    p(x_1, x_2, … x_n), n是指参数的数量

    param x_1
    param x_2
    …
    param x_n
    call p, n

    call p(x_1 , x_2 , … , x_n)

7.1.3 三地址码

- 三地址代码的类型

⑧ x := y[i]及x[i] := y的赋值

数组引用赋值或索引赋值，把相对地址y后面第i个单元里的值赋值给x；后面同理。

⑨ x := &y, x := *y 和 *x := y的地址和指针赋值

地址和指针赋值，从左往右，依序把y的地址赋值给x；把y指向的地址存放的值赋值给x等等

7.1.3 三地址码

- 四元式

带有四个域的记录结构，四个域分别为op, arg1, arg2及result

输入串 a := b * – c + b * – c的四元式序列为：

	op	arg1	arg2	result
(0)	uminus	c		T_1
(1)	*	b	T_1	T_2
(2)	uminus	c		T_3
(3)	*	b	T_3	T_4
(4)	+	T_2	T_4	T_5
(5)	:=	T_5		a

7.1.3 三地址码

- 三元式

  通过计算临时变量值的语句的位置来引用这个临时变量

  三个域：op、arg1和arg2

  输入串a:=b*-c+b*-c的三元式序列为：

	op	arg1	arg2
(0)	uminus	c	
(1)	*	b	(0)
(2)	uminus	c	
(3)	*	b	(2)
(4)	+	(1)	(3)
(5)	assign	a	(4)

三元式：把四元式的结果域去掉，临时变量的值用所在的位置代替。

7.1.2 图表示法

- 间接三元式

  为了便于优化，用 三元式表+间接码表 表示中间代码

  间接码表:一张指示器表，按运算的先后次序列出有关三元式在三元式表中的位置。

  优点: 方便优化，节省空间

7.1.2 图表示法

- 间接三元式

  例如，语句　　X:=(A+B)*C;
  　　Y:=D↑(A+B)

  的间接三元式表示如下表所示。

间接代码

(1)
(2)
(3)
(1)←
(4)
(5)

直接计算A+B，不再引入新的标号

三元式表

	OP	ARG1	ARG2
(1)	+	A	B
(2)	*	(1)	C
(3)	:=	X	(2)
(4)	↑	D	(1) ← A+B
(5)	:=	Y	(4)

35

7.2 赋值语句的翻译

7.2 赋值语句的翻译

7.2.1简单算术表达式及赋值语句

Operations Within Expressions and Static Single-Assignment Form

7.2.2数组元素的引用

Translation of Array References

7.2.1 简单算术表达式及赋值语句

7.2.1 简单算术表达式及赋值语句

- 为赋值语句生成三地址代码的S-属性文法定义

产生式	语义规则
$S \rightarrow id := E$	S.code:=E.code || gen(id.place ':=' E.place)
$E \rightarrow E_1 + E_2$	E.place:=newtemp; $E.code := E_1.code \,\|\|\, E_2.code \,\|\|$ gen(E.place ':=' E_1.place '+' E_2.place)

非终结符号S有**综合属性**S.code，它代表赋值语句S的三地址代码。

非终结符号E有如下两个属性：

E.place表示存放E值的地址。

E.code表示对E求值的三地址语句序列。

7.2.1 简单算术表达式及赋值语句

- 为赋值语句生成三地址代码的S-属性文法定义

产生式	语义规则
$E \rightarrow -E_1$	E.place:=newtemp; $E.code := E_1.code \,\|\|$ gen(E.place ':=' 'uminus' E_1.place)

函数newtemp的功能是，每次调用它时，将返回一个不同的临时变量名字,如T_1,T_2,…。

Gen函数，产生三地址语句,例如，gen(x ':=' y '+' z)表示生成三地址语句x: = y + z。

7.2.1 简单算术表达式及赋值语句

- 为赋值语句生成三地址代码的S-属性文法定义

产生式	语义规则
E→id	{p:=lookup(id,name); if p <> nil then E.place : = p else error }

当id为某个标识符的时候，此时id.place是指向符号表中该标识符表项的指针。如果扫描到是标识符，首先通过top(tblptr)指针在当前符号表中查找，看name是否存在；若没有找到，则lookup用当前符号表的表头的指针找到该符号表的外围符号表，然后继续查找name。如果所有外围符号表都没有name，则返回nil，表明查找失败。

41

产生式	语义规则
S→id:=E	S.code:=E.code ‖ gen(id.place ‘:=’ E.place)
E→E_1+E_2	E.place:=newtemp; E.code:=E_1.code ‖ E_2.code ‖ gen(E.place ‘:=’ E_1.place ‘+’ E_2.place)
E→E_1*E_2	E.place:=newtemp; E.code:=E_1.code ‖ E_2.code ‖ gen(E.place ‘:=’ E_1.place ‘*’ E_2.place)
E→ – E_1	E.place:=newtemp; E.code:=E_1.code ‖ gen(E.place ‘:=’ ‘uminus’ E_1.place)
E→ (E_1)	E.place:=E_1.place; E.code:=E_1.code
E→id	E.place:=id.place; E.code=‘ ’

{p:=lookup(id,name);
if p <> nil then
E.place : = p
else error }

42

7.2.1 简单算术表达式及赋值语句

例： 只含整型变量的简单赋值句文法描述为：

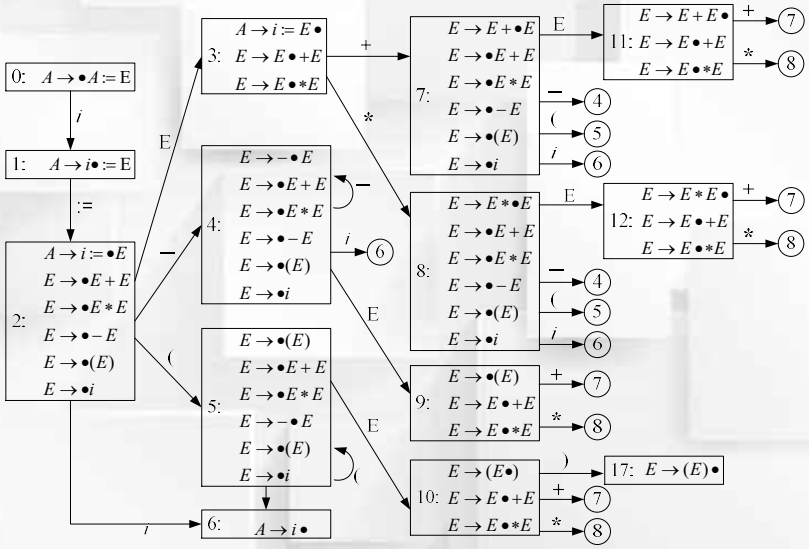
$A \rightarrow i := E$

$E \rightarrow E+E \mid E*E \mid (E) \mid -E \mid i$

用SLR(1)分析法（设*优先于+），分析A:=-B*(C+D)的三地址语句序列生成过程。

解：**1. 对上述文法，构造其活前缀识别自动机。**

7.2.1 简单算术表达式及赋值语句



7.2.1 简单算术表达式及赋值语句

例： 只含整型变量的简单赋值句文法描述为：

$A \rightarrow i := E$

$E \rightarrow E+E \mid E*E \mid (E) \mid -E \mid i$

用SLR(1)分析法（设*优先于+），分析A:= – B*(C+D)的三地址语句序列生成过程。

解：1. 对上述文法，构造其活前缀识别自动机。

2. 设符合① *高于+的优先关系; ②先左后右, 构造SLR(1)分析表

7.2.1 简单算术表达式及赋值语句

	ACTION		
S	:=	+	*
0			
1	S_2		
2			

7.2.1 简单算术表达式及赋值语句

例： 只含整型变量的简单赋值句文法描述为：

$A \rightarrow i := E$

$E \rightarrow E+E \mid E*E \mid (E) \mid -E \mid i$

用SLR(1)分析法（设*优先于+），分析A:= – B*(C+D)的三地址语句序列生成过程。

解：1. 对上述文法，构造其活前缀识别自动机。

2. 设符合① *高于+的优先关系; ②先左后右，构造SLR(1)分析表。

3. 用SRL(1)分析法分析输入串，同时生成三地址语句序列。

状态栈	符号栈
	#
	#
012	#i:=
0124	#i:=-
01246	#i:=-i
01249	#i:=-E

7.2.1 简单算术表达式及赋值语句

例： 只含整型变量的简单赋值句文法描述为：

$A \rightarrow i := E$

$E \rightarrow E+E \mid E*E \mid (E) \mid -E \mid i$

分析A:= – B*(C+D)的三地址语句序列生成过程。

$T_1 := -B$ //运用$E \rightarrow -E_1$的语义规则；

$T_2 := C+D$ //用$E \rightarrow E_1+E_2$；

$T_3 := T_1 * T_2$ //用$E \rightarrow E_1*E_2$；

$A := T_3$ //用$S \rightarrow id:=E$

第七章 小结

第七章 小结

- 7.1 中间语言
- 7.2 赋值语句的翻译