编译系统

# 第六章
# 中间代码生成

哈尔滨工业大学 陈鄞

# 第14讲（中间代码生成_4）要点

- 上讲知识回顾
  - 控制流语句翻译的一个关键是确定跳转指令的目标标号
  - 存在问题：生成跳转指令时，目标标号还不能确定
  - 解决办法：生成一些临时变量用来存放标号，将临时变量的地址作为继承属性传递到标号可以确定的地方。也就是说，当目标标号的值确定下来以后再赋给相应的变量
    - 缺点：需要进行两遍处理
      - 第一遍生成临时的指令
      - 第二遍将指令中的临时变量的地址改为具体的标号，从而得到最终的三地址指令

# 第14讲（中间代码生成_4）要点

- 回填(Backpatching)技术
  - 基本思想
    - 生成一个跳转指令时，暂时不指定该跳转指令的目标标号。这样的指令都被放入由跳转指令组成的列表中。同一个列表中的所有跳转指令具有相同的目标标号。等到能够确定正确的目标标号时，才去填充这些指令的目标标号

# 综合属性的设置

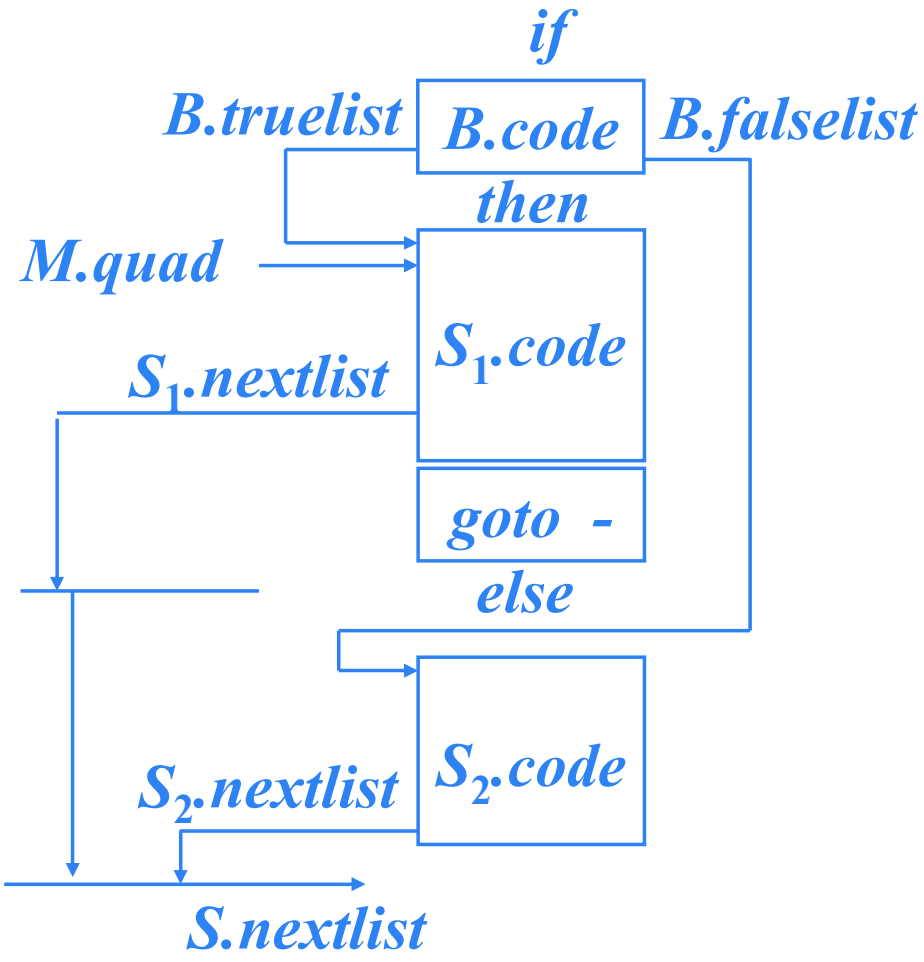
- ***B.truelist***：指向一个包含跳转指令的列表，这些指令最终获得的目标标号就是当**B**为真时控制流应该转向的指令的标号
- ***B.falselist***：指向一个包含跳转指令的列表，这些指令最终获得的目标标号就是当**B**为假时控制流应该转向的指令的标号
- ***S.next1ist***：指向一个包含跳转指令的列表，这些指令最终获得的目标标号就是按照运行顺序紧跟在**S**代码之后的指令的标号

# $S \rightarrow$ if $B$ then $S_1$ else $S_2$

$S \rightarrow$ if $B$ then $M$ $S_1$ else $S_2$

$M \rightarrow \varepsilon$

{ *M.quad = nextquad* ; }

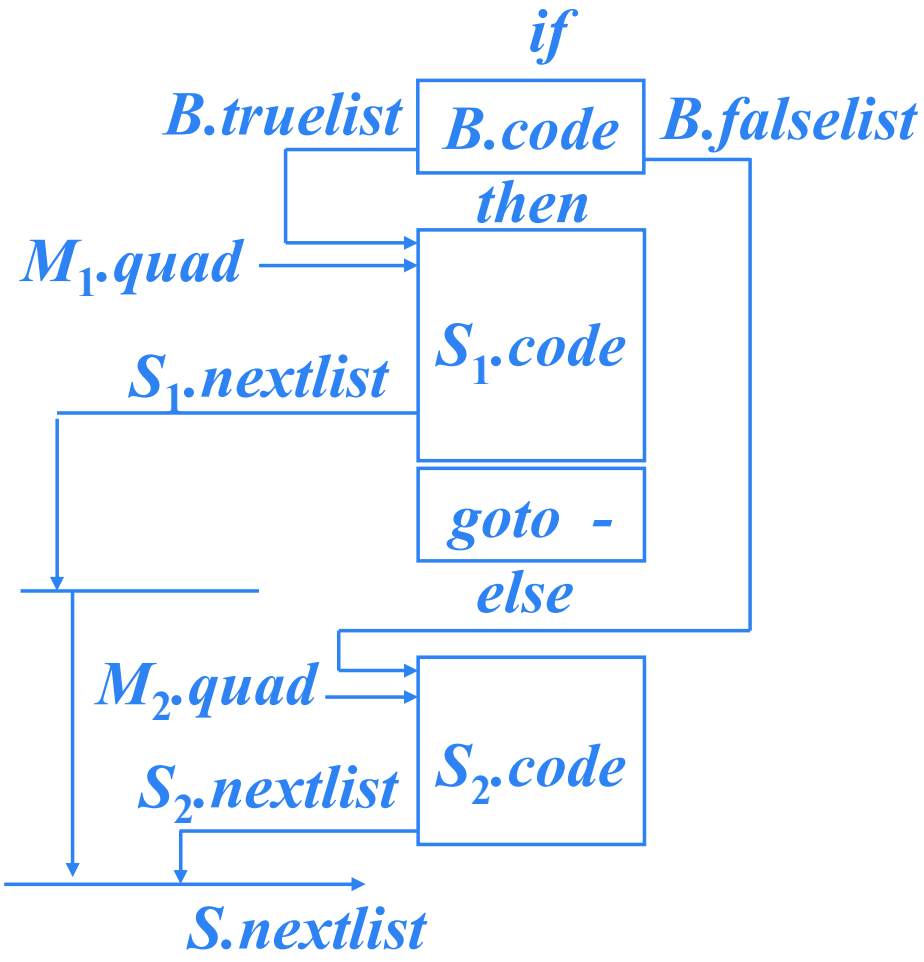


# $S \rightarrow$ if $B$ then $S_1$ else $S_2$

$S \rightarrow$ if $B$ then $M_1$ $S_1$ else $M_2$ $S_2$

$M \rightarrow \varepsilon$

{ *M.quad = nextquad* ; }



# $S \rightarrow$ if $B$ then $S_1$ else $S_2$

$S \rightarrow$ if $B$ then $M_1$ $S_1$ $N$ else $M_2$ $S_2$

{

$S.nextlist = merge( merge(S_1.nextlist, N.nextlist), S_2.nextlist )$;

$backpatch(B.truelist, M_1.quad)$;

$backpatch(B.falselist, M_2.quad)$;

}

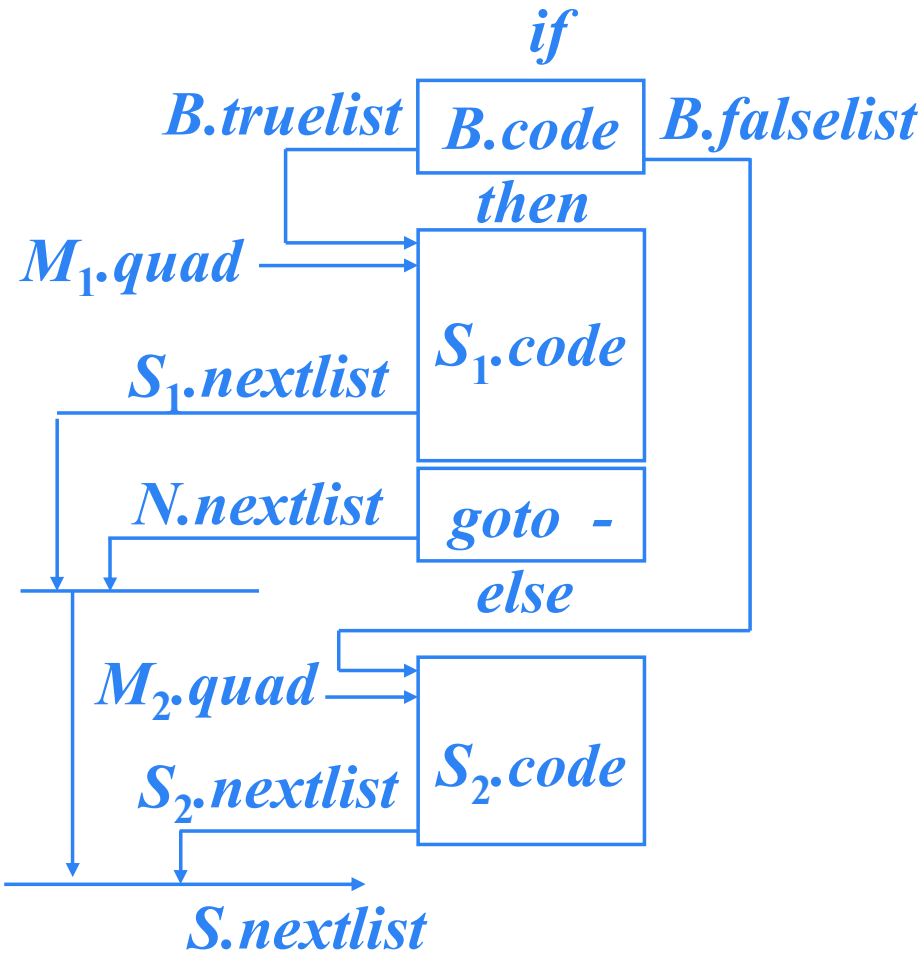
$N \rightarrow \varepsilon$

{

$N.nextlist = makelist(nextquad)$;

$gen(\text{'goto \_'})$;

}



# 回填技术SDT编写要点

- 文法改造
  - 在list箭头指向的位置设置标记非终结符M
- 在产生式末尾的语义动作中
  - 计算综合属性
  - 调用backpatch ( )函数回填各个list

# 语义分析中的错误检测

- 变量或过程未经声明就使用 （赋值/过程调用语句翻译）

```
S → id = E ;
{   p = lookup(id.lexeme); if p == nil then error ;
    gen( p '=' E.addr ); }
E → id
{   E.addr = lookup(id.lexeme); if E.addr == nil then error ; }
```

```
S → call id ( Elist )
{    n=0;
     for q中的每个t  do
     { gen('param' t );    n = n+1 ;  }
     gen('call' id.addr ',' n);
}
```

# 语义分析中的错误检测

- 变量或过程未经声明就使用 （赋值/过程调用语句翻译）
- 变量或过程名重复声明 （声明语句翻译）

$D \rightarrow T$ id;{ *enter*( id.*lexeme*, *T.type*, *offset* ); *offset* = *offset* + *T.width*; }$D$

# 语义分析中的错误检测

- 变量或过程未经声明就使用 （赋值/过程调用语句翻译）
- 变量或过程名重复声明 （声明语句翻译）
- 运算分量类型不匹配 （赋值语句翻译）

$E \rightarrow E_1 + E_2$

```
{   E.addr = newtemp()
    if E1.type == integer and E2.type == integer then
            { gen( E.addr '=' E1.addr 'int+' E2.addr ); E.type = integer ; }
    else if E1.type == integer and E2.type == real then
            {u = newtemp();
            gen( u '=' 'inttoreal' E1.addr );
            gen( E.addr '=' u 'real+' E2.addr );
            E.type = real ; }
    . . .
}
```

# 语义分析中的错误检测

- 变量或过程未经声明就使用（赋值/过程调用语句翻译）
- 变量或过程名重复声明（声明语句翻译）
- 运算分量类型不匹配（赋值语句翻译）
- 操作符与操作数之间的类型不匹配
  - 数组下标不是整数（赋值语句翻译）
  - 对非数组变量使用数组访问操作符（赋值语句翻译）

$$L \rightarrow \mathrm{id}\,[E] \mid L_1\,[E]$$

# 语义分析中的错误检测

- 变量或过程未经声明就使用（赋值/过程调用语句翻译）
- 变量或过程名重复声明（声明语句翻译）
- 运算分量类型不匹配（赋值语句翻译）
- 操作符与操作数之间的类型不匹配
  - 数组下标不是整数（赋值语句翻译）
  - 对非数组变量使用数组访问操作符（赋值语句翻译）
  - 对非过程名使用过程调用操作符（过程调用翻译）
  - 过程调用的参数类型或数目不匹配
  - 函数返回类型有误

$S \rightarrow$ call id ( *Elist* )
{ *n*=0;
*for q*中的每个*t do*
{ *gen*(‘*param*’ *t* ); *n* = *n*+1 ; }
*gen*(‘*call*’ id.*addr* ‘,’ *n*); }

# 结束