编译原理复习提纲

1 编译程序模型

输入：源程序

分析阶段：词法分析、语法分析、语义分析

综合阶段：中间代码优化、优化、目标代码生成器

两个阶段均有：错误处理、符号表管理

输出：目标代码

1.1 词法分析

内码：二元式

1.2 语法分析

语法规则用 BNF 表示

1.3 语义分析和中间代码生成

中间代码常用：逆波兰式、三元式、四元式、抽象机代码表示

1.4 代码优化

等价变换原则

1.5 目标代码生成

绝对指令代码→可重定位指令代码或汇编指令代码

1.6 表格管理

常见符号表：名字特性表、常数表、标号表、分程序入口表、中间代码表

1.7 前端和后端

前端：编译逻辑结构中的分析部分

后端：与目标机器有关的部分

2 文法和语言

2.1 产生式

形如$A \to \alpha$的(A, α)有序对

2.2 文法

$G[S]$可表示成四元式(V_N, V_T, P, S)

V_N非终结符号集，V_T终结符号集，P产生式集，$S \in V_N$起始符号

2.3 句型

α 是 G 的一个**句型** iff $S \overset{*}{\Rightarrow} \alpha$

当 α 只有终结符时，它是 G 的**句子**

G 产生的全部句子的集合为 G 产生的语言，记为 L(G)

2.4 规范推导

最右推导为规范推导，规范推导产生的句型为规范句型。

2.5 递归规则

2.5.1 直接左递归的产生式

$$A \to \beta A\delta, \beta = \epsilon, \delta \neq \epsilon$$

2.5.2 间接左递归的产生式

$$A \overset{+}{\Rightarrow} \beta A\delta, \beta = \epsilon, \delta \neq \epsilon$$

2.5.3 递归文法

至少含一个递归的非终结符号。

2.6 短语

有句型$w = xuy$：

$$Z \overset{*}{\Rightarrow} xUy且U \overset{+}{\Rightarrow} u，则u是相对于U的句型w的短语$$

2.6.1 简单（直接）短语

$$Z \overset{*}{\Rightarrow} xUy且U \Rightarrow u，则u是相对于U的句型w的简单(直接)短语$$

2.6.2 句柄

句型的**最左**简单短语

2.7 上下文无关文法（2 型、CFG）

$$V \to \alpha, V \in V_N, \alpha \in (V_N \cup V_T)^*$$

左边只有一个非终结符。

2.8 正则文法（3 型）

$$A \to aB或A \to a$$

2.9 CFG 的化简

2.9.1 消除无用符号

2.9.1.1 可达

$$X \in (V \cup U)，若起始符号S \overset{*}{\Rightarrow} \alpha X\beta，则X是可达的$$

2.9.1.2 产生

$$若有\alpha X\beta \overset{*}{\Rightarrow} w \ (w \in T^*)，则X是产生的$$

2.9.1.3 有用

若 X 同时是产生的和可达的，则称 X 是有用的，否则为无用符号。

2.9.1.4 消除算法

2.9.1.4.1 计算“产生的”符号集算法

1. 每个 T 中的符号都是产生的
2. 若有产生式A → α且α中符号都是产生的，则 A 是产生的

2.9.1.4.2 计算“可达的”符号集算法

1. 符号 S 是可达的
2. 若有产生式A → α且A是可达的，则α中的符号都是可达的

2.9.1.4.3 示例

$$S \to AB|\alpha$$
$$A \to b$$

消除非产生的

$$S \to \alpha$$
$$A \to b$$

消除非可达的

$$S \to \alpha$$

必须先消除非产生的，再消除非可达的。

2.9.2 消除ϵ − 产生式

形如$A \to \epsilon$的产生式为空产生式或ϵ − 产生式。

2.9.2.1 算法

先确定全部可空的变元：

1. 若$A \to \epsilon$，则 A 可空
2. 若$B \to \alpha$且α中每个符号都是可空的，则 B 可空

再替换带可空符号的产生式，若$A \to X_1X_2 \dots X_n$是产生式，那么用所有的$A \to Y_1Y_2 \dots Y_n$替代，其中：

1. 若X_i不是可空的，则$Y_i = X_i$
2. 若X_i是可空的，则$Y_i = X_i$或$Y_i = \epsilon$
3. 但Y_i不能全部为ϵ

2.9.2.2 示例

$$S \to AB$$
$$A \to AaA|\epsilon$$
$$B \to BbB|\epsilon$$

消除后

$$S \to AB|A|B$$
$$A \to AaA|Aa|aA|a$$
$$B \to BbB|Bb|bB|b$$

2.9.3 消除单元产生式

形如$A \to B$。

代入消除即可。

2.9.4 简化的可靠顺序

1. 消除ϵ − 产生式

2. 消除单元产生式
3. 消除非产生的无用符号
4. 消除非可达的无用符号

3 词法分析

3.1 有穷自动机

3.1.1 确定的有穷自动机（DFA）

$$M = (K, \Sigma, f, S_0, Z)$$

K：*有穷状态集*

Σ：*有穷字母表*

f：*映射*

S_0：*初始状态*

Z：*终止状态集*

3.1.2 非确定的有穷自动机（NFA）

$$M = (K, \Sigma, f, S_0, Z)$$

f*是多值函数*

3.1.3 NFA→DFA

3.1.3.1 $\epsilon - closure$

$\epsilon - closure(q)$：从状态 q 出发，仅经过 ϵ 弧，能达到的状态集合（包括 q 本身）

$$\epsilon - closure(I) = \{\epsilon - closure(q) | q \in I\}$$ (I 为状态集的子集)

3.1.3.2 move

$$move(T, a) = \{f(t, a) | t \in T\}$$

3.1.3.3 算法

计算起始状态的$\epsilon - closure$，记为T_0，依次计算T_i：$\epsilon - closure\big(move(T_0, a)\big)$ $(a \in \Sigma)$。

3.1.4 DFA 最小化

标记状态对(Q_i, Q_j)，$Q_i \in Z, Q_j \notin Z$。（即终点和非终点不能为一类）

若$(f(Q_i,\Sigma),f(Q_j,\Sigma))$被标记了，则标记$(Q_i,Q_j)$，重复该步骤，直到无可标记。（即目的地不在一类的点对不在一类）

3.1.4.1 定理

对于有同一接受集的 FA，与之等价且具有最小状态数的 DFA 在同构意义下是唯一的。

3.2 正则式（正规式）

$\cdot\ |\ *$ *三种基本运算*

与 FA 等价。

4 自顶向下语法分析

4.1 下推自动机

七元组$M=(Q,\Sigma,H,\delta,q_0,z_0,F)$

Q：*有限状态集*

Σ：*输入字母表*

H：*下推栈内字母表*

$\delta(q,a,z)$：*映射（非单值映射，当前状态*q*，输入符号*a*，下推栈栈顶符号*z*）*

$q_0\in Q$：*控制器的初始状态*

$z_0\in H$：*栈厨师符*

$F\subseteq Q$：*终态集*

4.2 LL(1)文法

4.2.1 构造

4.2.1.1 FIRST 集（开头集，α*可能的开头的终结符集合）*

$$FIRST(\alpha)=\left\{a\middle|\alpha\overset{*}{\Rightarrow}a\ldots,a\in V_T\right\}$$

若$\alpha\overset{*}{\Rightarrow}\epsilon$，则$\epsilon\in FIRST(\alpha)$

4.2.1.2 FOLLOW集（跟随集，A后面可能跟随的终结符集合）

$$FOLLOW(A) = \left\{a \middle| S \overset{*}{\Rightarrow} \dots Aa \dots, a \in V_T\right\}$$

若$S \overset{*}{\Rightarrow} \dots A$，则$\# \in FOLLOW(A)$

4.2.1.3 SELECT集（产生式预测集）

$$若\alpha \overset{*}{\nRightarrow} \epsilon，则SELECT(X \to \alpha) = FIRST(\alpha)$$

$$若\alpha \overset{*}{\Rightarrow} \epsilon，则SELECT(X \to \alpha) = (FIRST(\alpha) - \{\epsilon\}) \cup FOLLOW(X)$$

4.2.2 判别

相同左部的产生式的$SELECT$集的交集都为空，则是LL(1)文法。

4.2.3 非LL(1)转换为LL(1)

- 提取左公共因子
- 消除左递归
  - 直接左递归
  - 间接左递归

4.2.4 预测分析表

$$SELECT(E \to TE') = \{(, i\}$$
$$SELECT(E' \to +TE') = \{+\}$$
$$SELECT(E' \to \epsilon) = \{\#,)\}$$
$$SELECT(T \to FT') = \{(, i\}$$
$$SELECT(T' \to * FT') = \{*\}$$
$$SELECT(T' \to \epsilon) = \{+,), \#\}$$
$$SELECT\big(F \to (E)\big) = \{(\}$$
$$SELECT(F \to i) = \{i\}$$

	i	$+$	$*$	$($	$)$	$\#$
E	$\to TE'$			$\to TE'$		
E'		$\to +TE'$			$\to \epsilon$	$\to \epsilon$
T	$\to FT'$			$\to FT'$		
T'		$\to \epsilon$	$\to * FT'$		$\to \epsilon$	$\to \epsilon$
F	$\to i$			$\to (E)$		

4.2.4.1 分析过程

$$i + i * i\#$$

步骤	分析栈	剩余输入串	所用产生式或匹配
1	$\#E$	$i + i * i\#$	$E \to TE'$
2	$\#E'T$	$i + i * i\#$	$T \to FT'$
3	$\#E'T'F$	$i + i * i\#$	$F \to i$

4	$\#E'T'i$	$i+i*i\#$	*i匹配*
5	$\#E'T'$	$+i*i\#$	$\mathrm{T}'\to\epsilon$
…	…	…	…

5 自底向上优先分析

5.1 算符优先分析法

5.1.1 优先关系

5.1.1.1 $\boldsymbol{a\doteq b}$

$$iff\ \ 文法有\boldsymbol{U\to\cdots ab\ldots}或\boldsymbol{U\to\cdots aVb\ldots}的规则\ (a、b同时规约)$$

5.1.1.2 $\boldsymbol{a<b}$

$$iff\ \ 文法有\boldsymbol{U\to\cdots aW\ldots}，\ 其中\boldsymbol{W}\overset{+}{\Rightarrow}\boldsymbol{b\ldots}或\boldsymbol{W}\overset{+}{\Rightarrow}\boldsymbol{Vb\ldots}\ (b先规约)$$

5.1.1.3 $\boldsymbol{a>b}$

$$iff\ \ 文法有\boldsymbol{U\to\cdots Wb\ldots}，\ 其中\boldsymbol{W}\overset{+}{\Rightarrow}\ldots\boldsymbol{a}或\boldsymbol{W}\overset{+}{\Rightarrow}\ldots\boldsymbol{aV}\ \ (a先规约)$$

5.1.2 构造优先关系矩阵

5.1.2.1 ***FIRSTVT集***

$$FIRSTVT(U)=\left\{a\middle|U\overset{*}{\Rightarrow}a\ldots或U\overset{*}{\Rightarrow}Va\ldots\right\}$$

5.1.2.2 ***LASTVT集***

$$LASTVT(U)=\left\{a\middle|U\overset{*}{\Rightarrow}\ldots a或U\overset{*}{\Rightarrow}\ldots aV\right\}$$

5.1.2.3 算法

对每个产生式

$$U\to x_1x_2\ldots x_n$$

1. 若x_i和x_{i+1}都是终结符：$x_i\doteq x_{i+1}$
2. 若x_i是终结符x_{i+1}是非终结符：

$$x_i\doteq x_{i+2}$$

$$任意b\in FIRSTVT(x_{i+1}),\ \ x_i<b$$

3. 若x_i是非终结符x_{i+1}是终结符：

$$任意a \in FIRSTVT(x_i)，\ a \geqslant x_{i+1}$$

规定#优先级比相邻任何运算符都低。

不可以同时出现$a \lessdot b, a \gtrdot b, a \doteq b$任意两种。

5.1.3 素短语

- 至少包含一个终结符
- 除他自身，不再包含其他素短语

5.1.3.1 最左素短语

$$N_i a_i N_{i+1} \dots a_j N_j$$

满足

$$a_{j-1} \lessdot a_j$$
$$a_j \doteq a_{j+1} \doteq \cdots \doteq a_{i-1} \doteq a_i$$
$$a_i \gtrdot a_{i+1}$$

5.1.4 规约过程

$$i + i\#$$

栈	优先关系	当前符号	剩余输入串	移进或规约
#	$\lessdot$	i	$+i\#$	移进
$\#i$	$\gtrdot$	$+$	$i\#$	规约
$\#F$	$\lessdot$	$+$	i#	移进
$\#F+$	$\lessdot$	i	#	移进
$\#F+i$	$\gtrdot$	#		规约
$\#F+F$	$\gtrdot$	#		规约
$\#F$	$\doteq$	#		接受

$\doteq\lessdot$ 移进 $\gtrdot$ 规约

5.2 LR(0) 分析

5.2.1 拓广文法

增加产生式$S' \rightarrow S$

5.2.2 活前缀

规范推导（最右推导）：$S \Rightarrow \alpha_1 \Rightarrow \alpha_2 \dots \Rightarrow \alpha_{m-1} \Rightarrow \alpha_m = x$

其逆过程为最左规约（规范规约）。

每次规约前句型的前部，称为可归前缀。

把形成可归前缀之前，包括可归前缀的所有前缀，称为活前缀。

5.2.2.1 计算不包含句柄的活前缀

$$LC(A) = \left\{\alpha \middle| S' \overset{*}{\Rightarrow} \alpha A\omega, \alpha \in V_T^*\right\}$$

即，若有产生式$B \to \gamma A\delta$

则$LC(A) \supseteq LC(B) \cdot \{\gamma\}$

5.2.2.2 计算包含句柄的活前缀

$$\mathrm{LR(0)C(A} \to \beta) = \mathrm{LC(A)} \cdot \beta$$

5.2.2.3 例

$$S' \to E$$
$$E \to aA$$
$$E \to bB$$
$$A \to cA$$
$$A \to d$$
$$B \to cB$$
$$B \to d$$

求不包含句柄在内的活前缀方程组：

$$\begin{cases} LC(S') = \epsilon \\ LC(E) = LC(S') \cdot \epsilon = \epsilon \\ LC(A) = LC(E) \cdot a | LC(A) \cdot c = ac^* \\ LC(B) = LC(E) \cdot b | LC(B) \cdot c = bc^* \end{cases}$$

所以包含句柄的活前缀为：

$$LR(0)C(S' \to E) = E$$
$$LR(0)C(E \to aA) = LC(E) \cdot aA = aA$$
$$LR(0)(E \to bB) = LC(E) \cdot bB = bB$$
$$LR(0)(A \to cA) = LC(A) \cdot cA = ac^*ca$$
$$LR(0)(A \to d) = LC(A) \cdot d = ac^*d$$
$$LR(0)(B \to cB) = LC(B) \cdot cB = bc^*cB$$
$$LR(0)(B \to d) = LC(B) \cdot d = bc^*d$$

5.2.3 项目集规范族

5.2.3.1 LR(0) 项目

在产生式的右部每个空隙加一个圆点。

5.2.3.2 构造 NFA/DFA

$$A \to \alpha \cdot B\beta \text{属于} CLOSURE(I)，\ \text{则} B \to \cdot\gamma \text{也属于} CLOSURE(I)$$

5.2.3.3 LR(0) 分析表构造

*若*DFA 中，$f(I_i, a) = I_j$，*则*$ACTION[i, a] = S_j$

若DFA中，$f(I_i, A) = I_j, I_j$为规约项目，则$GOTO[i, A] = j$

若DFA中，I_i为规约项目，规约产生式为第j个产生式，则$ACTION[i,$*所有终结符和*$\#] = r_j$

最终表形如：

$$S' \to E$$
$$E \to aA|bB$$
$$A \to cA|d$$
$$B \to cB|d$$

状态	$ACTION$					$GOTO$		
	a	b	c	d	$\#$	E	A	B
0	S_2	S_3				1		
1					acc			
2			S_4	S_{10}			6	
3			S_5	S_{11}				7
4			S_4	S_{10}			8	
5			S_5	S_{11}				9
6	r_1	r_1	r_1	r_1	r_1			
7	r_2	r_2	r_2	r_2	r_2			
8	r_3	r_3	r_3	r_3	r_3			
9	r_5	r_5	r_5	r_5	r_5			
10	r_4	r_4	r_4	r_4	r_4			
11	r_6	r_6	r_6	r_6	r_6			

5.2.4 LR(0) 分析器工作

如上例中的分析表，分析$bccd\#$

状态栈	符号栈	输入串	$ACTION$	$GOTO$
0	$\#$	$bccd\#$	S_3	
03	$\#b$	$ccd\#$	S_5	
035	$\#bc$	$cd\#$	S_5	
0355	$\#bcc$	$d\#$	S_{11}	
0355(11)	$\#bccd$	$\#$	r_6	9
03559	$\#bccB$	$\#$	r_5	9
0359	$\#bcB$	$\#$	r_5	7
037	$\#bB$	$\#$	r_2	1
01	$\#E$	$\#$	acc	

遇到r_i规约后，状态栈弹出k个状态，k即第i个产生式右部符号数。

然后以此时栈顶状态为当前状态，当前输入为规约后的非终结符，找到对应的$GOTO$。

5.3 SLR(1) 分析

5.3.1 基本思路

利用非终结符号的 FOLLOW 集，判断“规约”还是“移进”。

5.3.2 解决冲突

若 LR(0)的规范族含有如下项目集

$$I = \{X \to \alpha \cdot b\beta, A \to \gamma \cdot, B \to \delta \cdot\}$$

存在移进-移进冲突和移进-规约冲突。

若

$$FOLLOW(A) \cap FOLLOW(B) = \emptyset$$
$$FOLLOW(A) \cap \{b\} = \emptyset$$
$$FOLLOW(B) \cap \{b\} = \emptyset$$

则当状态I面临输入符号a时，

若$a = b$，*则移进*

若$a \in FOLLOW(A)$，*则用*$A \to \gamma$*规约*

若$a \in FOLLOW(B)$，*则用*$B \to \delta$*规约*

此外，报错

5.4 LR(1) 分析

5.4.1 LR(1) 项目集族的构造

若$A \to \alpha \cdot B\beta, a$*属于*$CLOSURE(I)$

且$B \to \gamma$*是一个产生式*

则$B \to \cdot \gamma, b$*也属于*$CLOSURE(I), b \in FIRST(\beta a)$

起始项目为$S' \to \cdot S, \#$

5.4.2 LR(1) 分析表构造

项目$A \to \alpha \cdot, a$*属于*I_i，*则*$ACTION[i, a] = r_j$

其余不变

5.5 LALR(1) 分析

5.5.1 基本思路

合并同心集而不产生冲突。超前搜索符为之前的合集。

5.6 优先关系解决二义性文法

例如状态：

$$\mathrm{E} \to \mathrm{E} + \mathrm{E} \cdot$$
$$\mathrm{E} \to \mathrm{E} \cdot + \mathrm{E}$$
$$\mathrm{E} \to \mathrm{E} \cdot * \mathrm{E}$$

由于$*$> +，所以遇$*$移进，而+符合左结合，所以遇+规约。

又例如状态：

$$\mathrm{E} \to \mathrm{E} * \mathrm{E} \cdot$$
$$\mathrm{E} \to \mathrm{E} \cdot + \mathrm{E}$$
$$\mathrm{E} \to \mathrm{E} \cdot * \mathrm{E}$$

由于$*$> +，所以无论如何都规约。

6 语法制导翻译和中间代码生成

6.1 属性文法

对每个产生式$A \to \alpha$，有一套与之相关的语义规则。规则形式为：$b \coloneqq f(c_1, c_2, \dots, c_k)$

其中，f是一个函数，b和$c_1, c_2, \dots, c_k$都是该产生式文法符号的属性。

- *若b是A的属性，$c_1, c_2, \dots, c_k$是产生式右部文法符号的属性或A的其他属性，则b是A的**综合属性***
- *若b是产生式右部文法符号X的属性，并且$c_1, c_2, \dots, c_k$*

  *是A或产生式右边任何文法符号的属性，称b是X的**继承属性**。*

即综合属性是自下而上的，继承属性是自上而下的。

6.2 中间代码

6.2.1 逆波兰记号

运算对象在前，运算符号在后。

例：

$$a+b\text{写为}ab+$$

6.2.2 三元式

$$(\text{算符}\quad \text{运算对象 1},\quad \text{运算对象 2})$$

例：

$$a:=b*c+b*d$$

写为

$$(*\quad b,\quad c)$$
$$(*\quad b,\quad d)$$
$$(+\quad (1),\quad (2))$$
$$(:=\quad (3),\quad a)$$

6.2.2.1 间接三元式

操作表和三元式表。

操作表为三元式表的索引，表示执行顺序。

例：

$$\mathrm{A}:=\mathrm{B}+\mathrm{C}*\mathrm{D}/\mathrm{E}$$
$$\mathrm{F}:=\mathrm{C}*\mathrm{D}$$

写为

三元式表

1	$(*,\mathrm{C},\mathrm{D})$
2	$(/,(1),\mathrm{E})$
3	$(+,\mathrm{B},(2))$
4	$(:=,\mathrm{A},(3))$
5	$(:=,\mathrm{F},(1))$

操作表：(1)(2)(3)(4)(1)(5)

6.2.3 四元式

普遍采用。

$$(\text{算符},\quad \text{运算对象 1},\quad \text{运算对象 2},\quad \text{结果})$$

例：

$$a:=b*c+b*d$$

写为

$$(*,\quad b,\quad c,\quad t_1)$$
$$(*,\quad b,\quad d,\quad t_2)$$

$$(+, \quad t_1, \quad t_2, \quad t_3)$$
$$(:=, \quad t_3, \quad -, \quad a)$$

6.2.4 抽象语法树

节点序号为自底向上分析器创建该树的节点的次序。

6.3 翻译

newtemp 为生成临时变量。

6.3.1 简单赋值语句

(op, arg1, arg2, result)

```
S -> id:=E    {
                  p := lookup(id.name);
                  if p != null then emit(p ‘:=’ E.place) else error;
              }

E -> E¹+E²    {
                  E.place := newtemp;
                  emit(E.place ‘:=’ E¹.place ‘+’ E².place);
              }

E -> -E¹      {
                  E.place := newtemp;
                  emit(E.place ‘:=’ ‘uminus’ E1.place);
              }

E -> (E¹)     {
                  E.place := E¹.place;
              }

E -> id       {
                  E.place := newtemp;
                  p := lookup(id.name);
                  if p != nil then E.place := p else error;
              }
```

6.3.2 布尔表达式

```
E -> id₁ rop id₂     {
                         E.place := newtemp;
                         emit(‘if’ id₁.place ‘rop’ id₂.place ‘goto’ nextstat+3);
                         emit(E.place ‘:=’ 0);
                         emit(‘goto’ nextstat+2);
                         emit(E.place ‘:=’ ‘1’);
                     }

E -> true            {
                         E.place := newtemp;
                         emit(E.place ‘:=’ ‘1’);
                     }
```

```
E -> false          {
                        E.place := newtemp;
                        emit(E.place ‘:=’ ‘0’);
                    }
```

6.3.2.1 控制语句中的布尔表达式

```
E -> E¹ or E²       {
                        backpatch(E¹.false, E².codebegin);
                        E.codebegin := E¹.codebegin;
                        E.true := merge(E¹.true, E².true);
                        E.false := E².false;
                    }

E -> E¹ and E²      {
                        backpatch(E¹.true, E².codebegin);
                        E.codebegin := E¹.codebegin;
                        E.false := merge(E¹.false, E².false);
                        E.true := E².true;
                    }

E -> not E¹         {
                        E.true := E¹.false;
                        E.codebegin := E¹.codebegin
                        E.false := E².true;
                    }

E -> id₁ rop id₂    {
                        E.true := nextstat;
                        E.codebegin := nextstat;
                        E.false := nextstat+1;
                        emit(‘if’ id₁.place ‘rop’ id₂.place ‘goto’ ___);
                        emit(‘goto’ ___);
                    }

E -> true           {
                        E.true := nextstat;
                        E.codebegin := nextstat;
                        emit(‘goto’ ___);
                    }

E -> false          {
                        E.false := nextstat;
                        E.codebegin := nextstat;
                        emit(‘goto’ ___);
                    }
```

其中 backpatch 为回填，merge 为合并真/假链。

扫描到 then…else…再回填真/假链出口的空。

7 符号表

7.1 作用

- 收集符号属性
- 检查语义正确性
- 辅助生成代码

7.2 符号表的内容

7.2.1 标识符的名称

两种存储方法：

- 定长：标识符名称域规定最大长度
- 集中：开辟一个存放所有标识符的缓冲区，标识符名称域只存放偏移地址和长度

7.2.2 信息区

7.2.2.1 不同种属名字，建立不同的符号表

如常数表、变量名表、过程名表

7.2.2.2 最大单一符号表+

建立各类符号所有的属性项，形成一个单一的大符号表。

7.2.2.3 折中结构

将属性信息类似的符号分在一组，为每组建立单一符号表。

7.2.2.4 统一符号表的链接式结构

为多数符号的定长属性项确定“基本长度”，对需要空间多的符号，占用多个“基本长度”，用指针相连。

7.2.3 非分程序结构语言的符号表的组织

7.2.3.1 无序表

插入简单、查找效率低

7.2.3.2 有序表

插入需要附加查找

查找效率高于无序表

7.2.3.3 散列（Hash）表

平均查找次数本质上与表长无关。

7.2.4 分程序结构语言的符号表的组织（下推链）

7.2.4.1 分表结构

为每个分程序建立一张符号表。

7.2.4.2 单表结构

设一个属性域登录符号所在层次。

7.2.4.3 栈式符号表

遇变量声明，压入堆栈；

到达分程序结尾，弹出。

栈式符号表无序，查询效率低。

7.3 符号存储

7.3.1 类别

静态存储、寄存器存储

外部变量（公共存储变量）、内部变量（私有存储变量）

7.3.2 存储区

静态存储区

动态存储区

8 目标程序运行时的存储组织

8.1 分配方式

8.1.1 静态存储分配

编译时确定存储大小和位置。

8.1.2 动态存储分配

允许递归、可变数组、自由申请释放空间。

8.1.2.1 栈式动态存储分配

8.1.2.1.1 活动记录（AR：Activation Record）

存放过程的一次执行所需信息。

1. 临时工作单元：计算表达式临时存放中间结果
2. 局部变量
3. 机器状态信息：PC、寄存器
4. 存取链（非必需）：非局部变量

5. 控制链（非必需）：指向调用该过程的那个过程的活动记录

6. 实参

7. 返回地址

8.1.2.1.2 嵌套过程

- 增设存取链，指向包含该过程直接外层过程的最新活动记录的起始位置。
- display 栈
  - 指针数组，自顶向下存放现行层、直接外层，直至最外层 0 层的过程的最新活动记录的地址。

8.1.2.2 堆式动态存储分配

new、delete

9 代码优化

9.1 分类

9.1.1 按对象

- 中间代码优化
- 目标代码优化

9.1.2 按范围

- 局部优化
- 循环优化
- 全局优化

9.2 优化类型

9.2.1 删除多余运算

删除公共子表达式

9.2.2 代码外提

循环不变运算

9.2.3 强度削弱

乘法变加法等

9.2.4 变换循环控制条件

通过改变循环控制条件，减少使用的变量

9.2.5 合并已知量与复写传播

例如：T4=T1，T5=T4，则删除 T4，直接 T5=T1

9.2.6 删除无用赋值

对 T4 赋值但未引用

9.3 局部优化

9.3.1 基本块的划分

9.3.1.1 入口语句

- 程序的第一个语句
- 条件转移语句或无条件转移语句的转移目标语句
- 紧跟在条件转移语句后面的语句

9.3.1.2 基本快

1. 求各个入口语句
2. 对每个入口语句，构造所属的基本块。
   由该入口到下一入口，或到一转移语句、停语句之间的语句序列
3. 未被纳入某一基本块的语句，是不可到达的，可删

9.3.2 基本块的变换

9.3.2.1 保结构变换

- 删除公共子表达式
- 删除无用代码
- 重新命名临时变量
- 交换语句次序

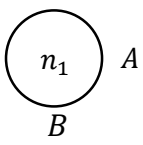
9.3.2.2 代数变换

- 简化表达式
- 较快运算替代较慢运算

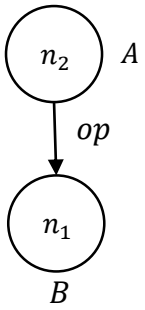
9.3.3 基本块的有向无环图（DAG）表示

节点下部是值，右部是附加节点，节点里面n_i是节点编号。

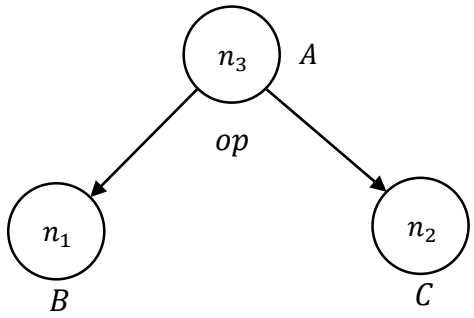
9.3.3.1 $\boldsymbol{A} \coloneqq \boldsymbol{B}$ $(\coloneqq, \boldsymbol{B}, -, \boldsymbol{A})$ 0 型



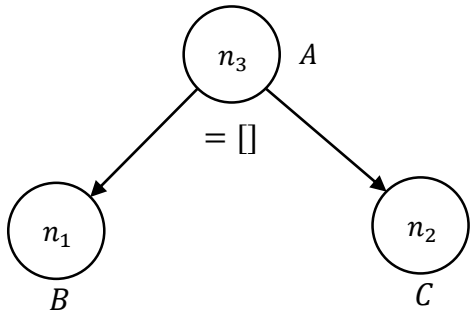
9.3.3.2 $\boldsymbol{A \coloneqq op\ B}$ $(\boldsymbol{op}, \boldsymbol{B}, -, \boldsymbol{A})$ 1 型



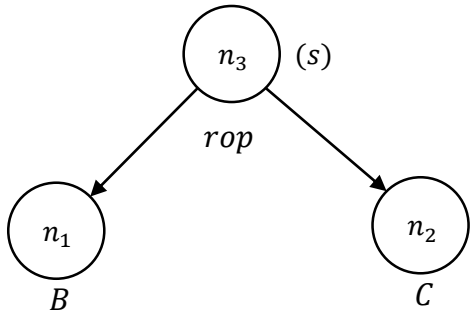
9.3.3.3 $\boldsymbol{A \coloneqq B\ op\ C}$ $(\boldsymbol{op}, \boldsymbol{B}, \boldsymbol{C}, \boldsymbol{A})$ 2 型



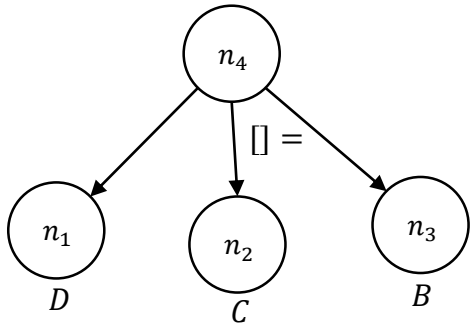
9.3.3.4 $\boldsymbol{A \coloneqq B[C](= [], B[C], -, A)}$



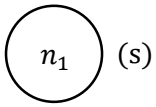
9.3.3.5 $\boldsymbol{if\ B\ rop\ C\ goto(s)}$ $(\boldsymbol{jrop}, \boldsymbol{B}, \boldsymbol{C}, (\boldsymbol{s}))$



9.3.3.6 $\boldsymbol{D[C] \coloneqq B}$ $([] =, \boldsymbol{B}, -, \boldsymbol{D[C]})$



9.3.3.7 $\boldsymbol{goto(s)(j, -, -, (s))}$



9.3.3.8 构造 0、1、2 型四元式的 DAG

- 若 NODE(B)无定义，则构造 NODE(B)
- 若是 0 型 $\boldsymbol{A \coloneqq B}$ $(\coloneqq, \boldsymbol{B}, -, \boldsymbol{A})$
  a) 令 NODE(B)=n
- 若是 1 型$\mathrm{A \coloneqq op\ B\ (op, B, -, A)}$
  a) 若 NODE(B)是常数叶节点
    i. 执行$op\ B$（合并已知量），令得到的新常数为 P
    ii. 若 NODE(B)是处理当前四元式才创建的，则删除它
    iii. 若 NODE(P)无定义
      1. 构造一个用 P 做标记的叶节点 n
      2. 置 NODE(P)=n
  b) 否则，不是常数叶节点
    i. 检查是否有一节点，其唯一后继为 NODE(B)，且标记为op（即公共子表达式）
    ii. 若没有，构造该节点 n
    iii. 若有，把已有的节点作为它的节点并设该节点为 n
- 若是 2 型$\mathrm{A \coloneqq B\ op\ C\ (op, B, C, A)}$
  a) 若 NODE(C)无定义
    i. 构造标记为 C 的叶节点，定义 NODE(C)为该节点

b) 若 NODE(B)和 NODE(C)都是标记为常数的叶节点

i. 执行$B\ \ op\ \ C$（合并已知量），令得到的新常数为 P

ii. 若 NODE(B)或 NODE(C)是处理当前四元式才创建的，删除

iii. 若 NODE(P)无定义

1. 构造一个用 P 做标记的叶节点 n

2. NODE(P)=n

c) 否则，其中之一不是常数叶节点

i. 检查是否有一节点，其左后继为 NODE(B)，右后继为 NODE(C)，且标记为op（公共子表达式）

ii. 若没有，构造该节点 n

iii. 若有，把已有节点作为它的节点并设该节点为 n

最后统一：

- 若 NODE(A)无定义
  i. A 附加在节点 n 上
  ii. 令 NODE(A)=n
- 否则，若 NODE(A)有定义
  i. 把 A 从 NODE(A)节点上的附加标识符集中删除（若 NODE(A)是叶节点则不删除）
  ii. 把 A 附加到新节点 n 上
  iii. 令 NODE(A)=n

9.4 控制流分析和循环优化

9.4.1 程序流（程）图（流图）

节点即基本块。

有向边，建立边$i \to j$，iff：

基本块 j 在程序中的位置紧跟在基本块 i 之后，且 i 的出口语句不是无条件转移语句或停语句

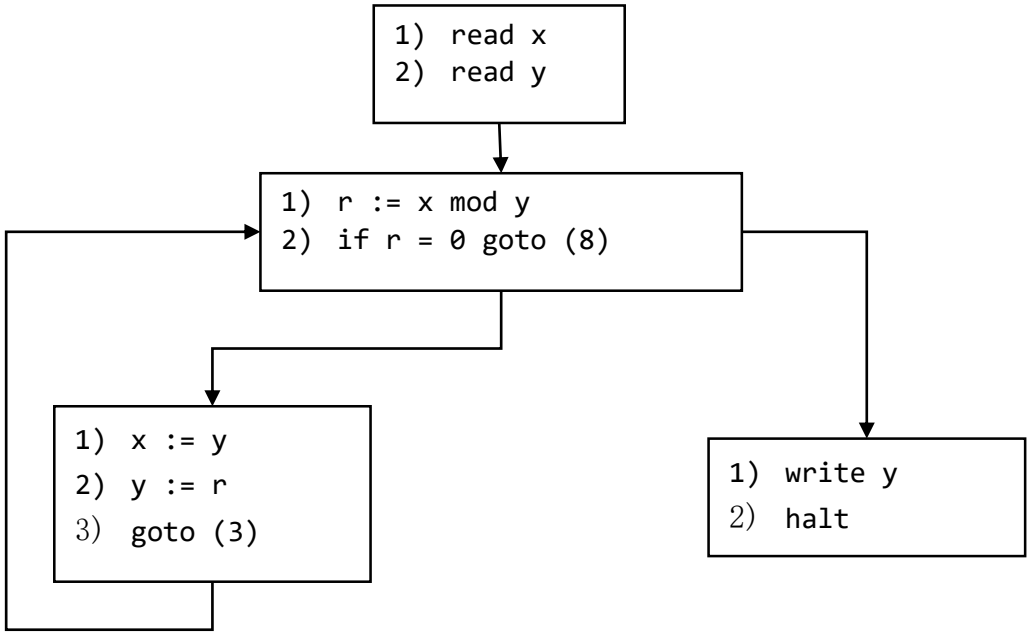
或

基本块 i 的出口语句是$goto(s)$或$if\ \ldots goto(s)$，且(s)是基本块 j 的入口语句

9.4.1.1 例

```
1) read x
2) read y
```

```
3)  r := x mod y
4)  if r = 0 goto (8)
5)  x := y
6)  y := r
7)  goto(3)
8)  write(y)
9)  halt
```



9.4.2 循环的查找

必经节点：若从流图首节点出发，到达 n 的任意通路，都经过 m，则 m 是 n 的必经节点。

记为 m DOM n。因此，有 a DOM a。

节点 n 的所有必经节点的集合，为 n 的必经节点集，记为 D(n)。

回边：a → b是一条有向边，若 b DOM a，则a → b是回边。

循环：a → b是回边，则节点 a、b，有通路到达 a 而不经过 b 的所有节点组成。

9.4.3 循环的优化

9.4.3.1 代码外提

循环所有出口节点的必经节点中的**循环不变运算**前置外提。

9.4.3.2 强度削弱与删除归纳变量

循环中对I*只有*$I := I \pm C$*的赋值*

$$J = C_1 \times I \pm C_2$$

I*为基本归纳变量，*J*为归纳变量，与*I*同族*

C_1*和*C_2*都是循环不变量*