

复习提纲

第1章 引论

1、什么是编译？

高级语言翻译成汇编语言或机器语言。

汇编器：把汇编语言转换为机器语言。

非终结符号是语法单位；终结符号是单词符号。

2、编译和解释的区别：

是否产生目标代码！

编译方式：先编译后执行。

解释方式：以源程序作为输入，但不产生目标代码，而是边解释边执行源程序本身。

3、编译过程包括哪些阶段，能够分析编译过程中出现的错误出现在编译的哪一个阶段，能举例说明：

- (1) 识别单词.....词法分析
- (2) 分析句子语法结构.....语法分析
- (3) 根据句子含义初步翻译.....语义分析与中间代码产生
- (4) 修饰译文.....优化
- (5) 写出最后译文.....目标代码生成

每一阶段都可能检测出错误，绝大多数错误可在前三阶段检测出来。

语法错误：源程序中不符合语法/词法规则的错误。词法/语法分析时检测

语义错误：源程序中不符合语义规则的错误。语义分析/运行时检测出来

4、编译前端和后端（概念，包含阶段）

(1) 前端： 由与源语言有关 但与目标机无关 的那些部分组成。包括——词法分析、语法分析、语义分析与中间代码产生、部分代码优化工作。

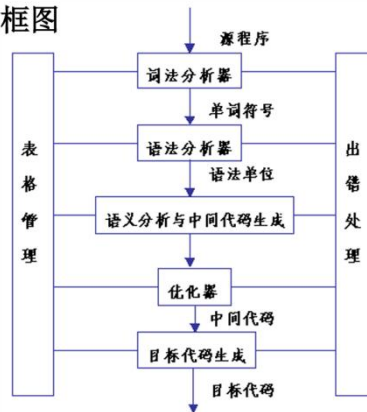
(2) 后端： 包括编译程序中与目标机有关的那些部分，如与目标机有关的代码优化和目标代码生成等。不依赖于源语言而仅仅依赖于中间语言。

5、遍（概念）

对源程序或源程序的中间结果从头到尾扫描一次，并作有关的加工处理，生成新的中间结果或目标程序。

1.3 编译程序的结构

一. 编译程序总框图



第2章 高级语言及其语法描述

1、高级语言的定义（语法和语义）

高级语言的定义通常包括两个主要方面：语法和语义。

语法：一组规则，使用它可以形成和产生一个合式的程序。

语义：一组规则，使用它可以定义一个程序的意义。

2、上下文无关文法定义：

文法：描述语言的语法结构的形式规则（即语法规则）

文法描述的语言是该文法的所有(句子)的集合。

上下文无关文法：所定义的语法范畴（或语法单位）是完全独立于这种范畴可能出现的环境的一种文法。

3、终结符/非终结符分别代表什么：

终结符号：描述单词符号，组成语言的基本符号，是一个语言的不可再分的基本符号。例如：基本字，标识符，常数，算符，界符等。

非终结符：代表语法范畴，一个非终结符代表一个一定的语法概念，每个非终结符表示一定符号串的集合。例如：算术表达式，布尔表达式，赋值句，分程序，

过程等。

“句子”：仅含终结符号的句型是一个句子。

句柄：一个句型的最左直接短语成为该句型的句柄。

短语：每棵子树的叶子。短语从上往下看非叶子节点的所有子叶子节点。

(1) **短语**: 若 $S \Rightarrow^* \alpha A \delta$ 且 $A \Rightarrow \beta$, 则称 β 是句型 $\alpha \beta \delta$ 相对于非终结符 A 的短语。

(2) **直接短语**: 若 $A \Rightarrow \beta$, 则称 β 是句型 $\alpha \beta \delta$ 相对于规则 $A \rightarrow \beta$ 的直接短语。

(3) **句柄**: 一个句型的最左直接短语成为该句型的句柄。

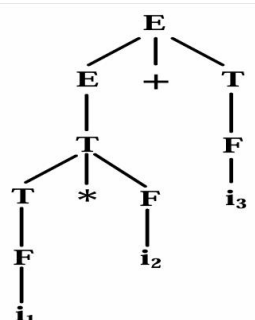
例1:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid i$

$i_1 * i_2 + i_3$



短语: $i_1, i_2, i_3, i_1 * i_2, i_1 * i_2 + i_3$

直接短语: i_1, i_2, i_3

句柄: i_1

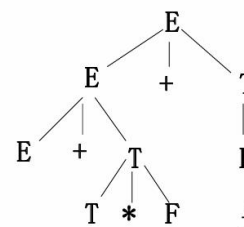
练习:

$E \rightarrow E + T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid i$

$E + T * F + i$



短语: $E + T * F + i, E + T * F, T * F, i$

直接短语: $T * F, i$

句柄: $T * F$

4、最左推导/最右推导/语法分析树（概念及应用）；

定义：任何一步 $\alpha \Rightarrow \beta$ 都是对 α 中的最左 / 最右非终结符 进行替换的。

语法分析树：用树的形式表示一个句型的推导生成。

二义性文法是指在某个文法中，某个句子存在不止一棵语法树。

5、乔姆斯基的文法分类。

2、乔姆斯基把文法分成4种类型，即0型，1型，2型和3型文法，这4类文法又分别称作()。

- ☐ A. 线性文法,上下文有关文法,上下文无关文法,正规文法
- ☐ B. 短语文法,上下文无关文法,上下文有关文法,正规文法
- ☒ C. 短语文法,上下文有关文法,上下文无关文法,正规文法
- ☐ D. 短语文法,单调文法,正则文法,上下文无关文法

第3章 词法分析

1、描述词法规则的工具（正规式，正规文法，有限自动机），要掌握这些工具的描述方式：

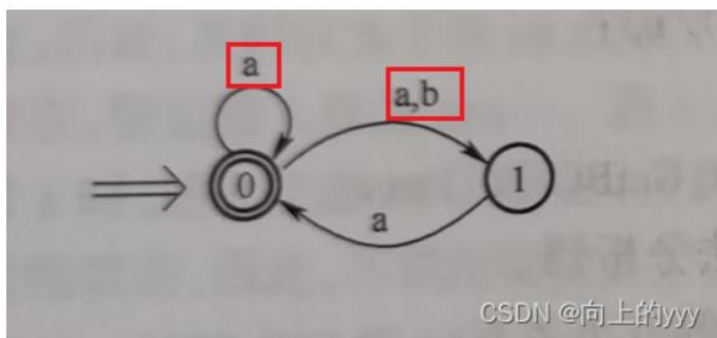
"*"表示匹配前一个字符零次或多次，"+"表示匹配前一个字符一次或多次等等。

3、同正规式 $(a|b)^*$ 等价的正规式是()。

- ☐ A. $(ab)^*$
- ☒ B. $(a^*|b^*)^+$
- ☐ C. $a^*|b^*$
- ☐ D. $(a|b)^+$

2、NFA 和 DFA 在定义上有何不同，给一个有限自动机，要能够知道是否是 DFA；
NFA=>DFA=>化简 DFA

NFA 在同一状态，可以有多条出边，DFA 在同一状态，只能有一条出边；



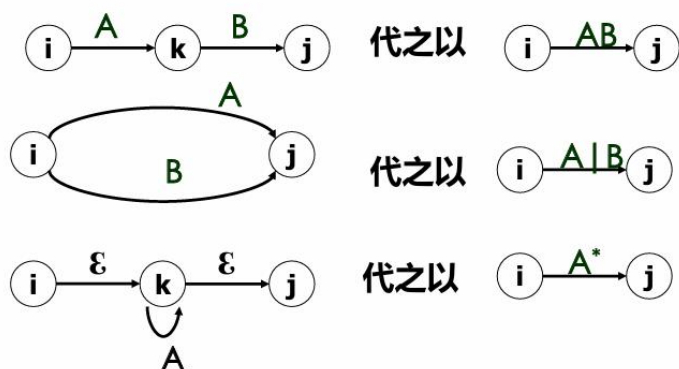
比如这个图就是NFA，因为0可以通过输入一个字符a到达本身，还可以通过a到达1，这就是在同一状态，有多条出边；

	DFA	NFA
S	有限状态集	同左
Σ	字母表	同左
δ	$S \times \Sigma \rightarrow S$	$S \times \Sigma^* \rightarrow 2^S$
初态	初态唯一	初态不一定唯一
终态F	可为空	同左

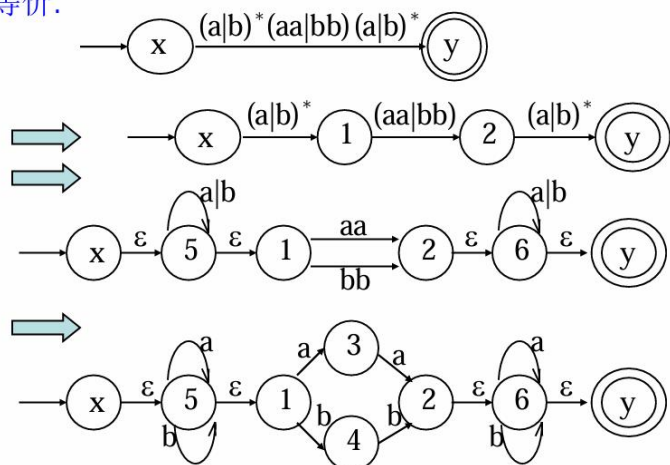
3、给定一个有限自动机，要能够写出其对应的正规式；

1、有限自动机=>正规式

替换规则：



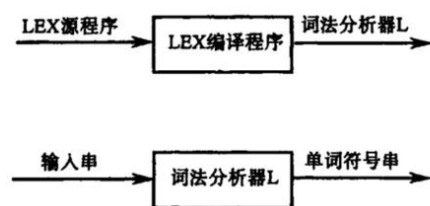
例：根据正规式 $(a|b)^*(aa|bb)(a|b)^*$ ，构造DFA M，使之等价。



4、要掌握 Lex 的工作流程。

一个描述词法分析器的语言：LEX

LEX 及其编译系统的作用如图 3.14 所示。



第 4 章 自上而下语法分析

1、语法分析的任务，自上而下语法分析的实质；

语法分析是编译过程的核心，其任务是在词法分析识别出正确的单词符号串的基础上，分析并判定程序的语法结构是否符合语法规则。

自上而下分析的主旨是：对任何输入串，试图用一切可能的办法，从文法的开始符号出发，自上而下地为输入串建立一个语法树。

自上而下分析法：从文法的开始符号出发，向下推导(使用最左推导)，尽可能使用各种产生式，推导出与输入串匹配的句子。

2. LL(1)文法的条件;

LL(1) 文法: (对文法进行不带回溯的确定的自顶向下分析的条件), 据此判别是否为LL(1)文法。

- (1) 文法不含左递归
- (2) 对文法中的任一个非终结符A的各个产生式的候选首终结符集两两不相交, 即: 若

$A \rightarrow \alpha_1 | \alpha_2 | \dots | \alpha_n$, 则

$$\text{First}(\alpha_i) \cap \text{First}(\alpha_j) = \varnothing \quad (i \neq j)$$

- (3) 对文法中的每个非终结符A, 若它存在某个首选符集含有 ϵ , 则

$$\text{First}(A) \cap \text{Follow}(A) = \varnothing$$

$T \rightarrow T, S$ 就不是 LL(1) 文法, 因为有左递归。

■ 1. 直接左递归的消除:

- 假定产生式为: $P \rightarrow P\alpha \mid \beta$, 将其替换为形式等价的产生式组:

$$\left\{ \begin{array}{l} P \rightarrow \beta P' \\ P' \rightarrow \alpha P' \mid \epsilon \end{array} \right.$$

例2: 文法

$E \rightarrow E+T \mid T$

$T \rightarrow T * F \mid F$

$F \rightarrow (E) \mid i$

消去左递归后为:

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \epsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \epsilon$

$F \rightarrow (E) \mid i$

3. 非终结符的 FIRST 和 FOLLOW 集构造。

6、文法G(S):

$S \rightarrow ABc$

$A \rightarrow a|\epsilon$

$B \rightarrow b|\epsilon$

则First(S)= .

☐ A. {a}

☐ B. {a,b}

☒ C. {a,b,c}

☐ D. {a,b,c, ϵ }

5、文法G(S):

$S \rightarrow ABc$

$A \rightarrow a|\epsilon$

$B \rightarrow b|\epsilon$

则Follow(A)=

☐ A. {B}

☐ B. {b, ϵ }

☒ C. {b,c}

☐ D. {b,c,#}

7、文法G(S):

$S \rightarrow ABBA$

$A \rightarrow a|\epsilon$

$B \rightarrow b|\epsilon$

则Follow(A)= .

☐ A. {b}

☐ B. {a,b}

☐ C. {a}

☒ D. {a,b,#}

3、预测分析表的构造

$\text{FIRST}(E) = \{ (, i \}$

$\text{FOLLOW}(E) = \{ \#,) \}$

$\text{FIRST}(E') = \{ +, \epsilon \}$

$\text{FOLLOW}(E') = \{ \#,) \}$

$\text{FIRST}(T) = \{ (, i \}$

$\text{FOLLOW}(T) = \{ +, \#,) \}$

$\text{FIRST}(T') = \{ *, \epsilon \}$

$\text{FOLLOW}(T') = \{ +, \#,) \}$

$\text{FIRST}(F) = \{ (, i \}$

$\text{FOLLOW}(F) = \{ *, +, \#,) \}$

例:

$E \rightarrow TE'$

$E' \rightarrow +TE' \mid \epsilon$

$T \rightarrow FT'$

$T' \rightarrow *FT' \mid \epsilon$

$F \rightarrow (E) \mid i$

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

1、FIRST集的求法

定义: $FIRST(\alpha) = \{a \mid \alpha \xRightarrow{*} a \dots\}$

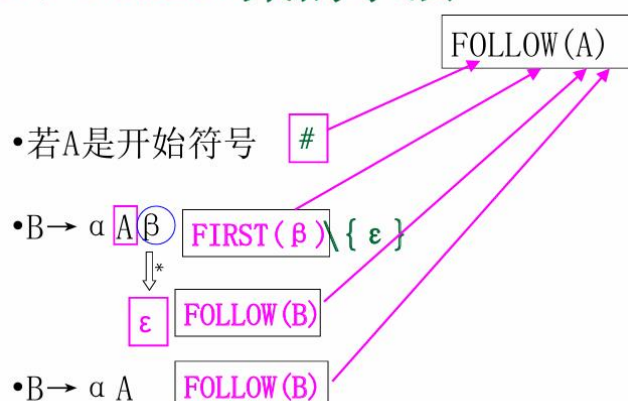
特别是, 若 $\alpha \xRightarrow{*} \epsilon$, 则规定 $\epsilon \in FIRST(\alpha)$

产生式左部是阿拉法的, 右部该元素第一个字母若是终结符, 则包含右部该元素第一个字母。

$FIRST(A)$ 是以 A 的开始符的集合, A 的所有可能推导的开头终结符或者是 ϵ

$FIRST$ 集就是 “箭头右侧首个终结符的集合”

2、FOLLOW集的求法



若有 $first()$ 注意去掉空。

$follow$ 的意思就是 “跟随”。所以求 $FOLLOW(A)$, 就是求出这个符号 A 后面能跟些什么。

注意, 并不是跟在 A 后面的所有元素都是它的 $FOLLOW$ 集, 我们只要 A 后面的且与 A 能够相邻的。注意这里的用词, 是 “能够相邻的”, 而不是 “相邻的”。

$B \rightarrow A$ $follow(B)$ 进入 $follow(A)$, $first(A)$ 进入 $first(B)$

如何求 $FIRST$ 集和 $FOLLOW$ 集 ($first$ 集看产生式左边 $follow$ 集看产生式右边)

① 文法开始符, 必有 $\#$

② $A \rightarrow \alpha B$ $FOLLOW(B)$ B 后为空, 将 $follow(A)$ 加入到 $follow(B)$ 中

③ $A \rightarrow \alpha B \beta$ $\begin{cases} \beta \text{ 是终结符, 直接写下来} \\ \beta \text{ 是非终结符, } first(\beta) \text{ 加入到 } follow(B) \text{ 中.} \end{cases}$
除 ϵ

如果 β 可以推出空, 那么需考虑包含情况②

第5章 自下而上语法分析

1、自下而上语法分析的实质：

即是从输入串开始，逐步进行“归约”，直至归约到文法的开始符号；或者说，从语法树的末端开始，步步向上“归约”，直到根结点。

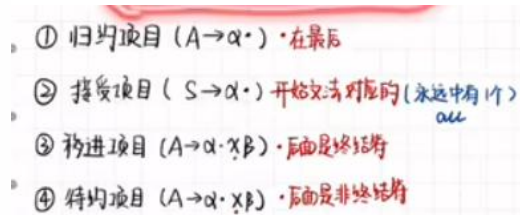
5.3 LR分析法

归约项目： $A \rightarrow \alpha \cdot$

接受项目(唯一)： $S' \rightarrow \alpha \cdot$

移进项目： $A \rightarrow \alpha \cdot a \beta$ ($a \in V_T$)

待约项目： $A \rightarrow \alpha \cdot B \beta$ ($B \in V_N$)

- 
- ① 归约项目 ($A \rightarrow \alpha \cdot$) · 在最后
 - ② 接受项目 ($S \rightarrow \alpha \cdot$) · 开始文法对应的 (永远有1个) *all*
 - ③ 移进项目 ($A \rightarrow \alpha \cdot a \beta$) · 后面是终结符
 - ④ 待约项目 ($A \rightarrow \alpha \cdot B \beta$) · 后面是非终结符

LR(0)文法的判别：LR(0)文法规范族的每个项目集不包含任何冲突项目。

假若一个文法G的拓广文法G'的活前缀识别自动机中的每个状态不存在

①既含**移进项目**又含**归约项目**；

②或者含有**多个归约项目**；

则称G是一个LR(0)文法。

2、基本概念：句柄：一个句型的**最左直接短语**成为该句型的句柄。

3、拓广文法是什么：

拓广文法是在原有的上下文无关文法基础上增加一个额外的起始符号和相应的产生式构成的文法。

4. 能写出拓广文法的所有 LR (0) 项目；

LL 分析是最左推导，LR 分析是最右推导的逆过程。

LR(0) 文法的判别：LR(0) 文法规范族的每个项目集不包含任何冲突项目。

LR(0) 项目，简称项目。

项目:文法产生式的右部添加一个圆点,称为项目.

例: $A \rightarrow \bullet XYZ$, $A \rightarrow X \bullet YZ$, $A \rightarrow XY \bullet Z$, $A \rightarrow XYZ \bullet$

含义:指明了在分析过程的某时刻我们看到产生式多大一部分.

例: $A \rightarrow \bullet XYZ$

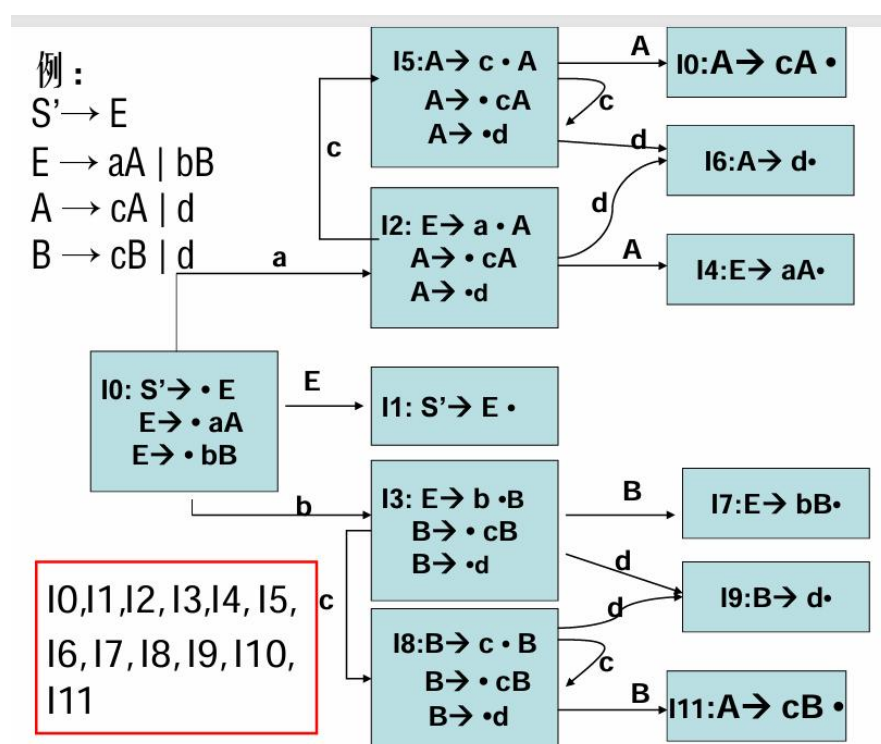
我们希望能从后面输入串中看到可以从XYZ推出的符号串.

$A \rightarrow X \bullet YZ$

我们已经从输入串中看到能从X推出的符号串,希望能进一步看到可以从YZ推出的符号串.

5.能写出识别活前缀的 LR(0)DFA (项目集规范族)。

LR(0) 项目集规范族: 识别活前缀的 DFA 的状态集。(其中, 该 DFA 的状态是 LR(0) 项目集)



5.3 LR分析法

(0) $S' \rightarrow E$	I0:	I4:	I7:
(1) $E \rightarrow E+T$	$S' \rightarrow \cdot E$	$F \rightarrow (\cdot E)$	$T \rightarrow T^* \cdot F$
(2) $E \rightarrow T$	$E \rightarrow \cdot E+T$	$E \rightarrow \cdot E+T$	$F \rightarrow \cdot (E)$
(3) $T \rightarrow T^*F$	$E \rightarrow \cdot T$	$E \rightarrow \cdot T$	$F \rightarrow \cdot i$
(4) $T \rightarrow F$	$T \rightarrow \cdot T^*F$	$T \rightarrow \cdot T^*F$	I8:
(5) $F \rightarrow (E)$	$T \rightarrow \cdot F$	$T \rightarrow \cdot F$	$F \rightarrow (E \cdot)$
(6) $F \rightarrow i$	$F \rightarrow \cdot (E)$	$F \rightarrow \cdot (E)$	$E \rightarrow E \cdot +T$
	$F \rightarrow \cdot i$	$F \rightarrow \cdot i$	I9:
	I1:	I5:	$E \rightarrow E+T \cdot$
	$S' \rightarrow E \cdot$	$F \rightarrow i \cdot$	$T \rightarrow T \cdot^*F$
	$E \rightarrow E \cdot +T$	I6:	I10:
	I2:	$E \rightarrow E+ \cdot T$	$T \rightarrow T^* F \cdot$
	$E \rightarrow T \cdot$	$T \rightarrow \cdot T^*F$	
	$T \rightarrow T \cdot^*F$	$T \rightarrow \cdot F$	I11:
	I3:	$F \rightarrow \cdot (E)$	$F \rightarrow (E) \cdot$
	$T \rightarrow F \cdot$	$F \rightarrow \cdot i$	

第6章 属性文法和语法制导翻译

1. 能够判定继承属性和综合属性，掌握其作用；

综合属性：用于“自下而上”传递信息。

继承属性：用于“自上而下”传递信息。

终结符只有综合属性，且是只读属性。

5、给定属性文法G(P)如下：

产生式	语义规则
$P \rightarrow xQR$	$R.b = Q.a; R.c = 1$
$Q \rightarrow u$	$Q.a = 1$
$R \rightarrow v$	$R.d = R.c; R.a = 2$

则对该文法属性的分类正确的是 ()。

- ☒ A. 综合属性: R.a, R.d, Q.a; 继承属性: R.b, R.c
- ☐ B. 综合属性: R.a, Q.a; 继承属性: R.b, R.c, R.d
- ☐ C. 综合属性: Q.a; 继承属性: R.a, R.d, R.b, R.c
- ☐ D. 综合属性: R.a, R.d, R.c, Q.a; 继承属性: R.b

看等式的左边在产生式中的位置。

若在产生式右边，则为继承属性；在左边，则为综合属性。

2、S-属性文法和 L-属性文法的概念。

L-属性文法——遍扫描的自上而下分析。L-属性文法可以含有综合与继承属性

S-属性文法——遍扫描的自下而上分析。S-属性文法只含有综合属性

S-属性文法一定是 L-属性文法。

第 7 章 语义分析和中间代码生成

1、采用中间代码的好处有哪些，中间代码的形式有哪些；

中间代码：即中间语言，独立于机器的，复杂性介于源语言和机器语言之间的一种表示形式。

采用中间语言的好处：

- (1) 便于进行与机器无关的代码优化工作；
- (2) 使编译程序改变目标机更容易；
- (3) 使编译程序的结构在逻辑上更为简单明确。

中间语言形式： 后缀式(逆波兰式)；三地址代码；图表示法

2、编译程序在对说明语句进行语义分析时，完成的主要任务是什么；

对一个过程或分程序的一系列说明语句,考察时,需要为局部于该过程的名字分配存储空间;对每个局部名字,都需在符号表中建立相应的表项,并填入有关的信息如类型、在存储器中的相对地址等。

3、针对简单的属性文法，能够对给定句子，结合语法分析树分析翻译结果；

4、根据赋值语句、布尔表达式、控制流语句的属性文法，对于给定程序段能够写出翻译后产生的四元式序列。

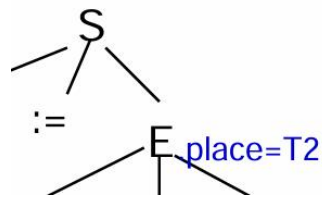
四元式：(操作符，左操作数，右操作数，结果)

与赋值语句 $a = b * (-c) + b * (-c)$ 等价的中间代码有 ()

四元式:

- (1) (uminus,c,,t1)
- (2) (*,b,t1,t2)
- (3) (uminus,c,,t3)
- (4) (*,b,t3,t4)
- (5) (+,t2,t4,t5)
- (6) (assign,t5,,a)

给出 $d:=a*(b+c)$ 的翻译结果



输出的四元式:

- (+,b,c,T1)
- (*,a,T1,T2)
- (:=,T2, ,d)

第 9 章 运行时存储空间的管理

1. 掌握编译程序对存储空间的分配策略及特点。

静态存储分配策略；栈式动态分配策略；堆式动态分配策略。

1、静态存储分配策略 在编译时对所有数据对象分配固定的存储单元，且在运行时始终保持不变。

2、栈式动态分配策略 在运行时把存储器作为一个栈进行管理，运行时，每当调用一个过程，它所需要的存储空间就动态地分配于栈顶，一旦退出，它所占空间就予以释放。

3、堆式动态分配策略 在运行时把存储器组织成堆结构，以使用户关于存储空间的申请与归还（回收），凡申请者从堆中分给一块，凡释放者退回给堆。

6、在运行时，程序代码，静态数据，堆区，栈区各自所在地址空间从低到高的顺序是（ ）。

☐ A. 静态数据区，程序代码，堆区，栈区

☒ B. 程序代码，静态数据区，堆区，栈区

第 10 章 优化

1. 掌握优化的目的和遵循的原则。

优化的目的是为了产生更高效的代码，需遵循以下的原则：

1、**等价**原则:经过优化后不应改变程序运行的结果。

2、**有效**原则:使优化后所产生的目标代码运行时间较短，占用的存储空间较小。

3、**合算**原则:应尽可能以较低的代价取得较好的优化效果。