



编译原理

第二章 高级语言 及其语法描述

内容简介：

本章概述高级语言的结构和主要的共同特征，并介绍程序语言的语法描述方法。要学习和构造编译程序，理解和定义高级语言是必不可少的。

2.1 程序语言的定义

任何语言实现的基础是语言的定义。在定义方面，编译程序研制者与一般用户有所不同，他们对那些构造允许出现更感兴趣。即使一时不能看出某种构造的实际应用，或者判断实现该结构会导致严重的困难，但仍必须严格根据语言的定义实现它。

程序语言主要由语法和语义两方面定义。

2.1.1 语法

语言的语法是指这样一组规则，用它可产生一个程序。

一. 词法规则

字母表就是一个有穷字符集。

C语言的字母表为：

$$\Sigma = \{a-z, A-Z, 0-9, (,),$$
$$[,], \rightarrow, .., !, \sim, +, -, *,$$
$$/, \&, \%, <, >, =, ^,$$
$$|, ?, ,, ; \}$$

词法规则是指单词符号的形成规则

C语言的标识符的构成规则：

字母、下划线打头的字母、数字和下划线构成的符号串。如：al、ave、_day

上述的定义是用文字来描述的，当设计编译程序时，就要把它用形式的方式描述出来，就要用到形式语言。

在现今多数程序设计语言中，单词符号一般包括：

标识符、 基本字、
各类型的常数、 算符和界符等

正规式和有穷自动机是描述词法结构和进行词法分析的有效工具

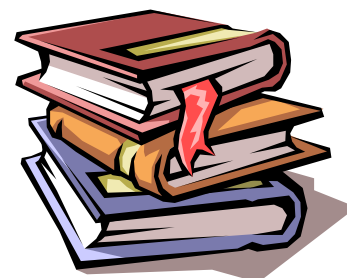
二. 语法规则

语法规则规定了如何从单词符号形成更大的结构（即语法单位），换言之，语法规则是语法单位的形成规则

一般的程序设计语言的语法单位有：

表达式、语句、分程序、函数、过程和程序等

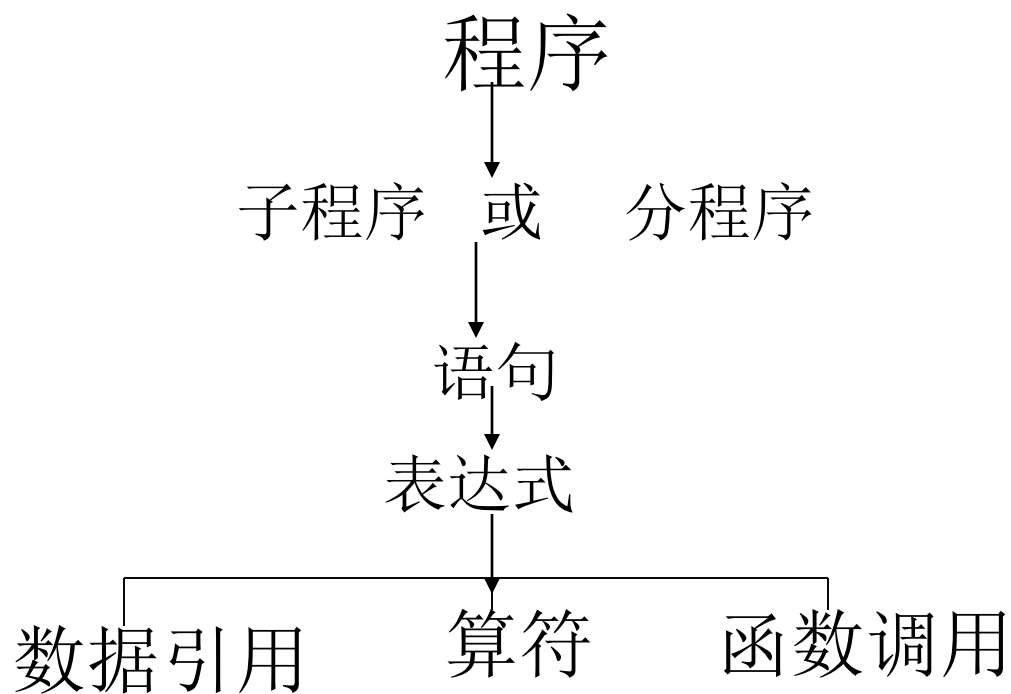
下推自动机理论和上下文无关文法
是我们讨论语法分析的理论基础

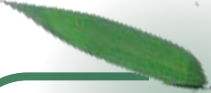


2.1.2 语义

- 对于一个语言来说，不仅要给出它的词法、语法规则，而且要定义它的单词符号和语法单位的意义。这就是语义问题。
- 语义是指这样的一组规则，使用它可以定义一个程序的意义。
- 我们采用的方法为：基于属性文法的语法制导翻译方法。

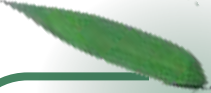
一个程序语言的基本功能是描述数据和对数据的运算。所谓程序，从本质上来说是描述一定数据的处理过程。在现今的程序语言中，一个程序大体可以视为下面所示的层次结构





自上而下看上述层次结构：顶端是程序本身，他是一个完整的执行单位。一个程序通常是由若干个子程序或分程序组成的，他们常常含有自己的数据（局部名）。

子程序或分程序是由于语句组成的。而组成语句的成分是个种类型的表达式。表达式是描述数据运算的基本结构，它通常含有数据引用、算符和函数调用。



自下而上看上述层次结构：我们希望通过了解下层成分的了解掌握上层成分，从而掌握整个程序。

在学习编译原理的过程中特别注意：程序语言的每个组成成分都有（抽象的）逻辑和计算机实现两方面的意义。当从数学上考虑每一个组成成分时，我们注重它的逻辑意义，当从计算机这个角度来看时，我们注重他在机内的表示和实现的可能性与效率。



2.2 高级语言的一般特性

2.2.1 高级语言的分类;

a.强制式语言 b.应用式语言 c.基于规则的语言 d.面向对象语言

2.2.2 几种程序的典型结构;

一. FORTRAN

一个FORTRAN 程序由一个主程序和若干个辅助程序段组成

PROGRAM MAIN

.

END

SUBROUTINE SUB1

.

. END

FUNCTION FUN1

.

END



它的定义是并列的

二. Pascal

Pascal 的 程序结构

Pascal 允许子程序
嵌套定义

也允许并列定义

```
program main
...
  procedure P1;
    procedure P11;
      begin
        end;
    begin
      end;
    procedure P2;
      begin
        end;
    begin
      end;
  end .
```

2.2.3 数据类型与操作

一. 名字

程序设计语言的数据对象： 数据、 函数、 过程

常用能反映其本质的、有助于记忆的名字来表示

特性：

一个名字对应一个对象 ， 普通变量

多个名字对应一个对象 ， common

一个名字对应多个对象 ， 数组、重载、
局部变量、
重写、



二. 数据类型

1. 初等数据类型

- ①数值数据：整形、实型、双精度等，可施行算术运算
- ②逻辑数据：可施行逻辑运算
- ③字符数据：
- ④指针类型：

三。数据结构

许多程序语言提供了一种由初级数据定义复杂数据的手段。下面我们将概述其中常见的定义方式：

1. 数组

从逻辑上说，一个数组是由同一类型数据所组成的某种 n 维矩形结构。沿着每一维的距离称为一个下标。数组的每个元素是矩形结构中的一个点，它的位置可通过给出每维的下标来确定。

数组的每个元素（也称为**下标变量**）是由数组名连同各维的下标值命名的。如 $A(i_1, i_2, \dots, i_n)$ 。根据数组的类型，每个数组元素在计算机中占有同样大小的存储空间。如果一个数组所需的存储空间大小在编译时就已知道则称此数组是一个**确定数组**；否则称为**可变数组**。



数组的存储表示有多种形式，最简单的一种是把整个数组按行（或按列）存放在一片连续存储区中。

数组元素的地址计算和数组的存储方式密切相关。关于数组元素的地址计算公式，数据结构教材中有详细的介绍。编译程序要做的就是实现地址计算公式，使数组元素得到正确的引用。

在编译过程中，当碰到数组说明时，必须把数组的有关的信息记录在一个“内情向量”之中，以便以后计算数组元素的地址时引用这些信息。每个数组的内情向量必须包括：维数，各维的上下限，首地址，以及数组元素的类型等。

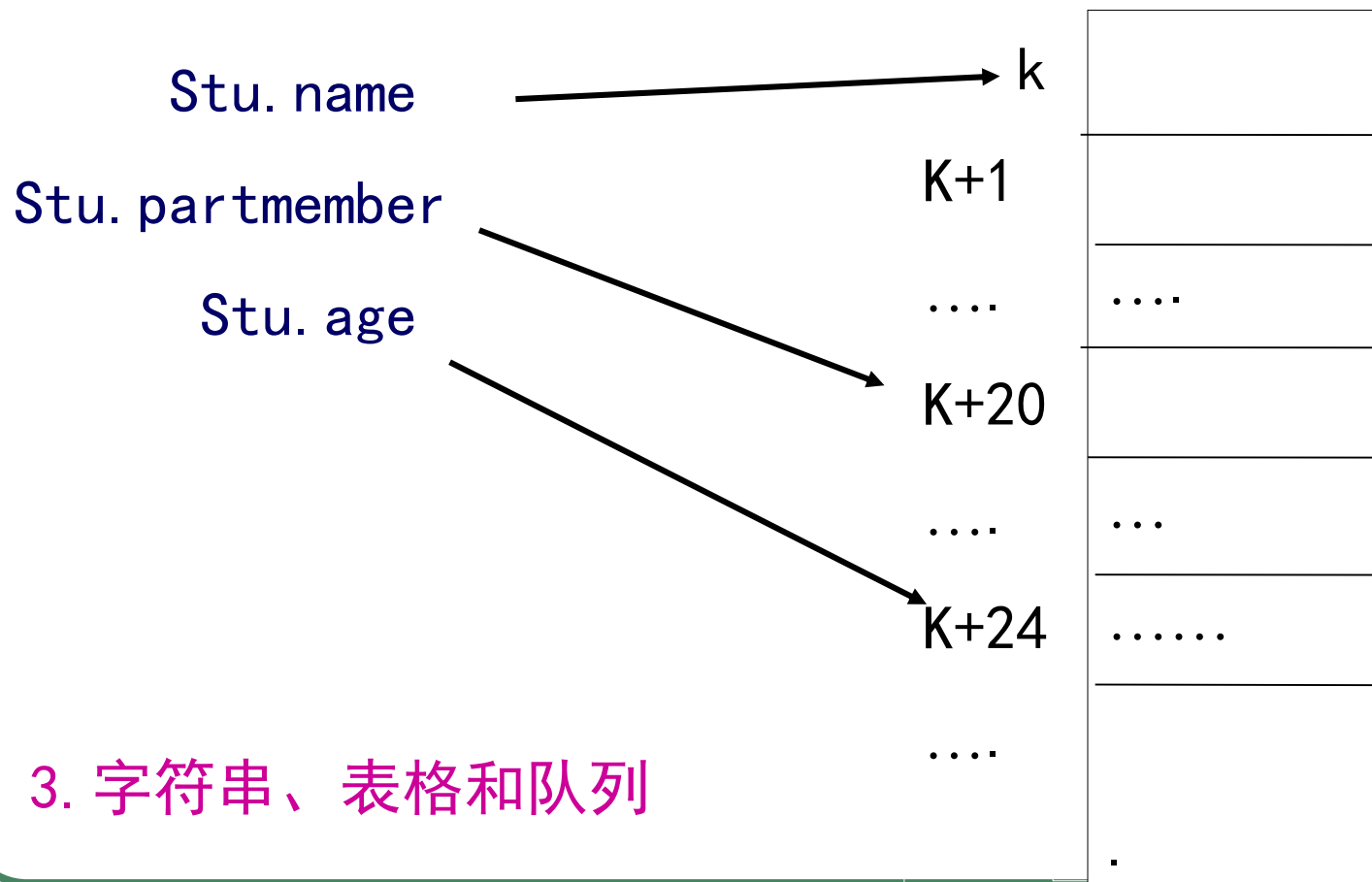
2. 记录（结构）

从逻辑上讲，记录是由已知的数据组合起来的一种结构

```
Struct student
{ char    name[20];
  boolean partmember;
  int age;
} stu;
```

记录结构最简单的存贮方式是连续存放

上述的变量stu共占7个字，共28个字节



四. 抽象数据类型

一个抽象数据类型包括：

- (1)数据对象的一个集合
- (2)作用于这些数据对象的抽象运算的集合
- (3)这种类型对象的封装

C++、Java语言通过类对抽象类型提供支持

五. 语句与控制结构

1. 表达式

要解决的问题：

- ① 优先级
- ② 结合率

2. 语句

语句可分为：

- ① 说明语句： 定义各种不同数据类型的变量和运算
- ② 可执行语句： 描述语句的动作

执行语句分为：赋值、控制和I/O语句

(1)赋值句

A=B

左值

右值

名字的左值指它所代表的存储单元地址

名字的右值指该单元的内容

(2)控制语句

无条件转移语句: Goto lable

条件语句: If B then S

 If B then S else S

循环语句: While B do S

 Repeat S until B

 For $l=e_1$ to e_2 step e_3

过程调用语句: Call P(x_1, x_2, \dots, x_n)

返回语句: Return(E)

(3)说明语句

说明语句用于定义名字的性质。

编译程序把这些性质登记在符号表中，并检查程序中名字的引用和说明是否一致。

许多说明语句不产生目标代码

但有的说明语句，如过程说明和可变数组说明，则要产生相应的目标代码



(4)简单句和复合句

简单句是指不包含其它语句成分的基本句。赋值、goto语句等

复合句则指那些句中有句的语句

```
If (x==0) then  x=1
```

```
{x=1;y=2;goto l1;}
```



2.3 程序语言的语法描述

对于高级程序语言及编译程序而言，语言的语法定义是非常重要的。本节将介绍语法结构的形式描述问题。首先引入几个概念：

设 Σ 是一个有穷字母表，它的每个元素称为一个符号。 Σ 上的一个符号串是指由 Σ 中的符号所构成的有穷序列。不包含符号的序列称为空字，记为 ε 。用 Σ^* 表示 Σ 上的所有符号串的全体，空字也包括在其中。如：若 $\Sigma = \{a, b\}$ 则 $\Sigma^* = \{\varepsilon, a, b, aa, ab, bb, aaa, \dots\}$ 。 ϕ 表示不含任何元素的空集 $\{\}$ 。这里要注意 ε 、 $\{\}$ 和 $\{\varepsilon\}$ 的区别。

Σ^* 的子集U和V中的（连接）积定义为：

$$UV = \{\alpha\beta \mid \alpha \in U \ \& \ \beta \in V \}$$

即集合UV中的符号串是由U和V的符号串连接而成的。

注意，一般 $UV \neq VU$ ，但 $(UV)W = U(VW)$ 。

V自身的n次（连接）积记为：

$$V_n = V \ V \dots V \quad (n \text{个} V)$$

规定 $V_0 = \{\varepsilon\}$ 。

$$\text{令： } V^* = V_0 \cup V_1 \cup V_2 \cup \dots$$

称 V^* 是V的闭包。

记 $V^+ = VV^*$ ，称 V^+ 是V的正则包。

闭包 V^* 中的每个符号都是由V中的符号串经有限次连接而成的。

2.3.1 上下文无关文法：

文法是描述语言的语法结构的形式规则（即语法规则）。

所谓上下文无关文法是这样一种文法，它所定义的语法范畴（或语法单位）是完全独立于这种范畴可能出现的环境的。

请仔细阅读课本P27页的英文分析的例句，定义英文句子的规则可以说是一个上下文无关文法。其中He, me, book, gave, a 等，称为终结符号；<句子>、<主语>、<谓语>等为非终结符号；这个文法最终是要定义<句子>的语法结构，所以<句子>在这里成为开始符号；<间接宾语>→<冠词><名词> 这种书写形式称为产生式。

归纳起来，一个上下文无关文法G包括四个组成部分：一组终结符号，一组非终结符，一个开始符号，以及一组产生式。

所谓终结符号乃是组成语言的基本符号，即在程序语言中以前屡次提到的单词符号，如基本字，标识符，常数，算符和界符等

所谓非终结符号（也称语法变量）用来代表语法范畴。如“算术表达式”、“布尔表达式”、“过程”等。一个非终结符代表一个一定的语法概念。因此非终结符是一个类（或集合）记号，而不是个体记号。

开始符号是一个特殊的非终结符号，它代表所定义的语言中我们最感兴趣的语法范畴，这个语法范畴通常称为“句子”。但在程序语言中我们最终感兴趣的是“程序”这个语法范畴，而其他的语法范畴都只不过是构造“程序”的一块块砖石。

产生式（也称为产生规则或简称规则）是定义语法范畴的一种书写规则。一个产生式的形式是 $A \rightarrow \alpha$ ，其中箭头左边的A是一个终结符，称为产生式的左部符号；箭头右边的 α 是终结符号或与非终结符号组成的一符号串，称为产生式的右部。

产生式是定义语法范畴的。

如要定义一类含+、*的算术表达式，这个定义可以这样给出：

“变量是一个算术表达式；

若E1和E2是算术表达式，则E1+E2、E1*E2和(E1)也是算术表达式。 对于这个定义，用产生式描述：

$$E \rightarrow i$$
$$E \rightarrow E + E$$
$$E \rightarrow E * E$$

$E \rightarrow (E)$ 其中E代表“算术表达式”，i代表“变量”

- 形式上定义一个上下文无关文法 G 是一个四元式 $(V_T, V_N, S, \mathcal{E})$ 其中
- V_T 是一个非空有限集，它的每一个元素称为终结符号；
- V_N 是一个非空有限集，它的每一个元素称为非终结符号， $V_T \cap V_N = \phi$ ；
- S 是一个非终结符号，称为开始符号；
- \mathcal{E} 是一个产生式（有限）集合，每个产生式形式是 $P \rightarrow \alpha$ ，其中， $P \in V_N$ ， $\alpha \in (V_T \cup V_N)^*$ 开始符号 S 至少必须在某一产生式的左部出现一次。

- 假定G是一个文法，S是它的开始符号。如果 $S \Rightarrow^* \alpha$ (表示从S出发，经0步或若干步可推出 α)，则称 α 是一个句型。仅含终结符号的句型是一个句子。文法G所产生的句子的全体是一个语言，将它记为 $L(G)$ 。

- $$L(G) = \{ \alpha \mid S \Rightarrow^+ \alpha \ \& \ \alpha \in VT \}$$

- 例如：终结符号串 $(i*i+i)$ 是文法

- $$E \rightarrow E+E \mid E * E \mid (E) \mid i \quad (2.1)$$

- 的一个句子。是因为有

- $$E \Rightarrow (E) \Rightarrow (E+E) \Rightarrow (E * E + E) \Rightarrow (i * E + E)$$

- $$\Rightarrow (i * i + E) \Rightarrow (i * i + i)$$
 从开始符号E至

- $(i * i + i)$ 的一个推导。而E, (E), $(E * E + E)$ 等是文法的句型。

- 下面介绍几个简单文法的例子：
- 例2。1考虑一个文法G1：
- $S \rightarrow bA$
- $A \rightarrow aA \mid a$ 它定义了一个什么语言呢？
- 从开始符S出发，我们可以推出如下句子：
- $S \Rightarrow bA \Rightarrow ba$
- $S \Rightarrow bA \Rightarrow baA \Rightarrow baa$
- $S \Rightarrow bA \Rightarrow baA \Rightarrow \dots \Rightarrow baa\dots a$
- 可以写为： $L(G1) = \{ban \mid n \geq 1\}$

例2.2 设有文法G

$$S \rightarrow P \mid aPPb$$
$$P \rightarrow ba \mid bQa$$
$$Q \rightarrow ab$$

求语言 $L(G)$.

解:

$$L(G) = \{ba, baba, abab, ababab, \dots\}$$

例2.3 构造一个文法G3使

$$L(G3) = \{anbn \mid n \geq 1\}$$

解: $S \rightarrow aSb \mid ab$

为了对句子结构进行确定性分析，我们往往只考虑最左推导或最右推导。所谓最左推导是指：任何一步 $\alpha \Rightarrow \beta$ 都是对 α 中的最左非终结符进行替换的。同样，可定义最右推导。

2.3.2 语法分析树与二义性

前面我们提到过可以用一张图表示一个句型的推导，这种表示称为**语法分析树**，或简称**语法树**。

语法树的根结由开始符号所标记。随着推导的展开，当某个非终结符被它的某个候选式所替换时，这个非终结符的相应结就产生了下一代新结。每个新结和其父亲结间都有一条连线。在一棵语法树生长过程中的任何时刻，所有那些没有后代的端末结自左至右排列起来就是一个句型。

例如对于文法 (2.1) 关于 $(i*i+i)$ 的推导形成语法树如图 2.2

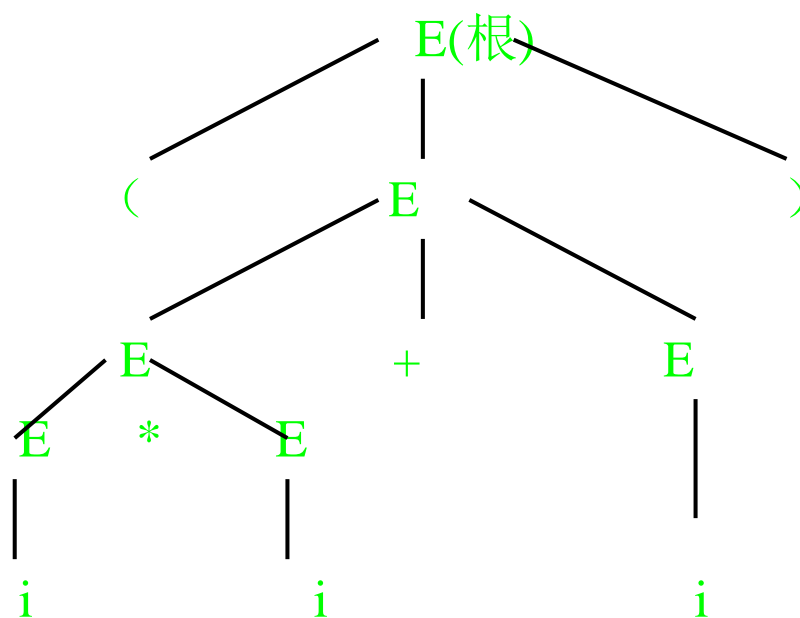


图 2.2 语法树

- 一个句型是否只对应唯一的一棵语法树呢？
也就是说它是否只有唯一的一个最左（最右）
推导呢？^{（根）}不尽然。
- 如文法2.1, 关于 $(i*i+i)$ 就存在一个与2。
2非常不同的推导：
- $E \Rightarrow (E) \Rightarrow (E * E) \Rightarrow (i * E) \Rightarrow (i * E + E)$
- $\Rightarrow (i * i + E) \Rightarrow (i * i + i)$ 其对应语法树：

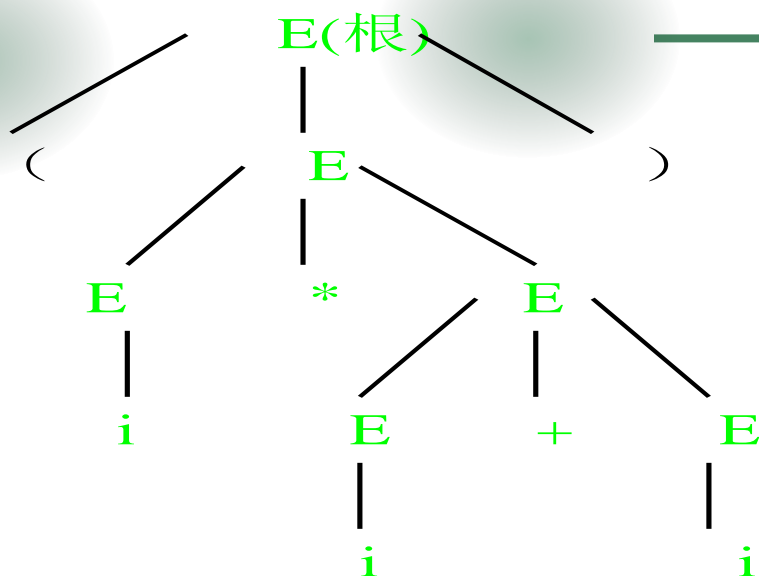


图 2.3 语法树

图2.2与图2.3的不同之处在于从第二代三代过渡开始。对于前者，我们选用规则 $E \rightarrow E + E$ ，而后者我们选用 $E \rightarrow E * E$ 。这里不是同代兄弟生儿子的先后次序的不同而是生什么儿子的不同，后面的这个不同是本质上的差异。这意味着我们可以用两种完全不同的办法产生同一个句子。

如果一个文法存在某个句子对应两棵不同的语法树，则称这个文法是二义的。也就是说，若一个文法存在某个句子，它有两个不同的最左（最右）推导，则这个文法是法是二义的。最后，作为描述程序语言的上下文无关文法，我们对它有几点限制。

(1) 文法中不含任何下面形式的产生式： $P \rightarrow P$ 因为这种产生式除了产生二义性外没有任何用处。

(2) 每个非终结符 P 必须有用处。这一方面意味着，必须存在含 P 的句型；也就是，从开始符号出发，存在推导 $S \Rightarrow^* \alpha P \beta$ 。

另一方面意味着，必须存在终结符串 $\gamma \in VT^*$ ，使得 $P \Rightarrow^+ \gamma$ ；也就是，对于 P 不存在永不终结的回路。

我们以后所讨论的文法均假定满足上述两条件。

2.3.3 形式语言鸟瞰：

乔姆斯基把文法分为四种类型即0型、1型、2型、3型。0行强于1型，1行强于2型，2型强于3型。这几文法的差别在于对产生式施加不同的限制。

我们说 $G=(VT, VN, S, \zeta)$ 是一个0型文法，如果它的每个产生式 $\alpha \rightarrow \beta$ 是这样的结构

$\alpha \in (VN \cup VT)^*$ 且至少有一个非终结符，而
 $\beta \in (VN \cup VT)^*$ 。0型文法也称短语文法。

如果对0型文法分别施加以下的第*i*条限制，则我们就得到第*i*型文法：

(1) *G*的任何产生式 $\alpha \rightarrow \beta$ 均满足

$|\alpha| \leq |\beta|$ (其中 $|\alpha|$ 和 $|\beta|$ 分别为 α 和 β 的长度；仅 $S \rightarrow \varepsilon$ 例外，但 S 不得出现在任何产生式的右部。

(2) *G*的任何产生式为 $A \rightarrow \beta$, $A \in VN$,

$\beta \in (VN \cup VT)^*$ 。

(3) *G*的任何产生式为 $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha$, 其中 $\alpha \in VT^*$, $A, B \in VN$ 。

其中1型文法也称上下文有关文法。这种文法意味着，对非终结符进行替换式务必考虑上下文并且一般不允许替换成空串 ε 。

2型文法也称上下文无关文法，注意其语言定义：

G的任何产生式为 $A \rightarrow \beta$ ， $A \in VN$ ， $\beta \in (VN \cup VT)^*$

表明凡出现在产生式左边的符号都是非终结符。

3型文法也称右线性文法。3型文法还有另一种形式，称左线性文法：一个文法G为左线性文法，如果G的任何产生式为

$A \rightarrow B\alpha$ 或 $A \rightarrow \alpha$ ，其中 $\alpha \in VT^*$ ，

$A, B \in VN$ 由于3型文法等价于正规式所以也称正规文法。