编译原理

——学习指导与典型题解析

刘春林　王挺　周会平　钟求喜　编著

国防工业出版社
http://www.ndip.cn

编译原理

学习指导与典型题解析

刘春林　王挺　周会平　钟求喜　编著

国防工业出版社

·北京·

图书在版编目(CIP)数据

编译原理:学习指导与典型题解析/刘春林等编著.
北京:国防工业出版社,2004.8
ISBN 7-118-03475-4

Ⅰ.编… Ⅱ.刘… Ⅲ.编译程序-程序设计-高等学校-习题 Ⅳ.TP314-44

中国版本图书馆 CIP 数据核字(2004)第 033184 号

国防工业出版社出版发行
(北京市海淀区紫竹院南路 23 号)
(邮政编码 100044)
腾飞胶印厂印刷
新华书店经售
*
开本 787×960 1/16 印张 19 390 千字
2004 年 8 月第 1 版 2004 年 8 月北京第 1 次印刷
印数:1—5000 册 定价:26.00 元

内容简介

本书依据中国计算机学会、全国高校计算机教育研究会制定的“计算机科学与技术教程(CCC2002)”对编译原理课程教学的基本要求,并以陈火旺院士等编写的《程序设计语言编译原理(第三版)》教材的结构和内容为主线编写而成,旨在帮助学生正确理解书中的概念和原理,把握重点和难点,掌握解题技巧。书中每一章均包括学习要点、典型题解析和习题与解答3部分。学习要点中简要归纳该部分的主要内容和需要重点掌握的知识点,着重理清其中的概念、原理和方法,为学生理解和掌握课程内容提供指导;典型题解析针对那些具有普适性的问题,特别是针对学生在学习中遇到的重点和疑难问题,详尽地进行了分析和讨论,旨在帮助学生拓宽思路,加深对课程内容的理解,提高分析和解决问题的能力;每一章都选编了适当数量的各类习题,提供给读者练习,所有习题均给出了参考解答。在附录中还收入了几所大学的考研全真试卷以供读者参考。

本书不仅是计算机专业编译原理课程的学习指导书,也是研究生入学考试的复习参考书,对于参加计算机专业自学考试和各类软件考试的考生以及其他需学习或了解编译原理的人员也有一定的参考价值。

前　言

程序设计语言编译原理是计算机专业的一门核心课程，在计算机专业教学中占有十分重要的地位。该课程具有很强的理论性、技术性和实践性，学生在学习时普遍感到内容抽象、不易理解、难以掌握。为了帮助学生及其他初学者正确理解概念和原理，把握重点和难点，掌握解题技巧，我们依据中国计算机学会、全国高校计算机教育研究会制定的“计算机科学与技术教程(CCC2002)”对编译原理课程教学的基本要求，并以陈火旺院士等编写的在国内使用广泛并荣获全国高校优秀教材一等奖的《程序设计语言编译原理(第三版)》教材的结构和内容为主线，编写了这本学习辅导书。

本书共分为8章。第1章包括编译程序的组成和高级语言及其语法描述；第2章是词法分析，主要涉及正规式与有限自动机的内容；第3章介绍自上而下语法分析，主要是LL(1)分析以及递归下降分析法和预测分析法；第4章介绍自下而上语法分析，主要包括算符优先分析和LR分析；第5章介绍属性文法和语法制导翻译的基本概念与方法；第6章是语义分析和中间代码产生；第7章包括运行时存储空间组织和符号表的内容；第8章包括优化与目标代码生成的内容。

书中每一章均包括学习要点、典型题解析和习题及解答。学习要点中简要归纳了该部分的主要内容和需要重点掌握的知识点，着重理清其中的概念、原理和方法，为学生理解和掌握课程内容提供指导；典型题解析针对那些具有普遍性的问题，特别是针对学生在学习中遇到的重点和疑难问题，详尽地进行分析和讨论，旨在帮助学生拓宽思路，加深对课程内容的理解，提高分析和解决问题的能力；习题及解答选编了适当数量的各类习题，提供给读者练习，所有习题均给出了参考解答。本书所选的例题和习题主要来源于两个方面：一是《程序设计语言编译原理(第三版)》教材中的习题，二是历年来全国计算机领域各著名大学和研究所的研究生入学考试试题。为了便于学生复习考试，我们还在附录中

选编了若干典型试卷，包括4套研究生入学考试全真试卷和2套模拟试卷。

本书不仅是计算机专业编译原理课程的学习指导书，也是研究生入学考试的复习参考书，对于计算机专业自学考试和各类软件考试的考生以及其他需学习或了解编译原理的人员也有一定的参考价值。

本书第1章、第2章由周会平编写，第3章、第4章由王挺编写，第7章、第8章由钟求喜编写。刘春林编写第5章、第6章，并负责最后统稿。在本书的编写过程中得到了陈火旺院士的鼓励和指导，得到了国防科技大学计算机学院领导和同事的支持和帮助，在此表示衷心感谢。

由于作者水平有限，书中疏漏之处恳请读者批评指正。

目　录

第 1 章　高级语言及编译程序概述

计算机语言是人类指导计算机工作的工具，为了便于操作，高级语言一般较接近于数学语言和工具语言，比较直观、自然和易于理解。但一般来说，计算机无法理解和直接执行高级语言，因此我们要为计算机构造编译程序，由编译程序将高级语言程序翻译成计算机能够识别的机器语言，然后交给计算机去执行。

这一章的内容主要是有关编译程序和高级语言及其语法描述的一些基本概念。

1.1　学习要点

1.1.1　程序语言的定义

一个程序语言是一个记号系统。如同自然语言一样，程序语言主要是由语法和语义两方面定义的。有时，语言定义也包含语用信息，语用主要是有关程序设计技术和语言成分的使用方法，它使语言的基本概念与语言的外界（如数学概念或计算机的对象和操作）联系起来。任何语言程序都可看作是一定字符集（称为字母表）上的一个字符串。合乎语法的字符串才算是一个合法的程序。

语言的语法是用来形成一个合法程序的一组规则。这些规则的一部分称为词法规则，另一部分称为语法规则（或产生规则）。语言的词法规则是指单词符号的形成规则。语法规则则规定了如何从单词符号形成更大的结构（即语法单位），因此，语法规则也可以说是语法单位的形成规则。一般程序语言的语法单位有：表达式、语句、分程序、函数、过程和程序等等。词法规则和语法规则定义了程序语言的形成规则，而程序的意义则由语言的语义规则来定义。语义规则规定了语言的单词符号和语法单位的意义，是定义一个程序意义的一组规则。

程序语言的语法规则是用文法来描述的，构成文法的规则必须是准确而且易于理解的，并具有相当强的描述能力，足以描述各种不同的结构。现今的程序语言一般用上下文无关文法来描述。

1.1.2　高级语言的分类

根据其功能和构成规则，高级语言可以分为以下几类：

（1）强制式语言（Imperative Language），也称过程式语言。其特点是命令驱动，面向

语句。一个强制式语言程序是由一系列的语句组成,每个语句的执行引起若干存储单元中的值的改变。如 FORTRAN、C、PASCAL、Ada 等等。

(2) 应用式语言(Applicative Language),也称函数式语言。这类语言更注重程序所表示的功能,而不是一个语句接一个语句地执行。程序的开发过程是从前面已有的函数出发构造出更复杂的函数,对初始数据集进行操作直至最终的函数可以用于从初始数据计算出最终的结果。如 LISP、ML 等。

(3) 基于规则的语言(Rule-based Language),也称逻辑程序设计语言。这类语言的程序执行过程是:检查一定的条件,当它满足值,则执行适当的动作。如 Prolog 等。

(4) 面向对象语言(Object-Oriented Language),是如今最流行、最重要的高级语言。它的主要的特征是支持封装性、继承性和多态性等。把复杂的数据和用于这些数据的操作封装在一起,构成类;通过类的继承和复合设计出更复杂的类。如 Smalltalk、Java、C + +、面向对象的 PASCAL 等。

1.1.3 数据类型与操作

程序的本质是描述一定数据的处理过程。因此对于大多数程序设计语言而言,数据的概念是最基本的。程序设计语言所提供的数据及其操作对语言的适用性有很大影响。一个数据类型通常包括以下 3 种要素:

(1) 用于区别这种类型数据对象的属性;

(2) 这种类型的数据对象可以具有的值;

(3) 可以作用于这种类型的数据对象的操作。

一个程序语言必须提供一定的初等数据类型,包括这些数据类型上能进行的运算的定义。不同的语言含有不同的初等数据成分。常见的初等数据类型有:

(1) 数值数据,如整数、实数、复数以及这些类型的双长(或多倍长)精度数。对它们可施行算术运算(+,-,*,/等)。

(2) 逻辑数据,对它们可施行逻辑运算(and、or、not 等)。

(3) 字符数据,包括字符型或字符串型的数据,这对于符号处理是必需的。

(4) 指针类型,指针是把内存地址作为其值的数据类型,通过指针可以操作内存空间。

程序语言中的各种名字都是用标识符表示的。标识符是指由字母、下划线和数字组成的,以字母或下划线为开头的一个字符串。名字和标识符在形式上难于区分,标识符是一个没有意义的字符序列,而名字则有明确的意义和属性。用计算机术语来说,每个名字可看成是代表一个抽象的存储单元,这个单元可含有一位、一字节或相继的多个字节。该单元的内容则被认为是名字的值。仅把名字看成代表一定的存储单元还是不够的,我们还必须同时指出它的属性(数据类型)。只有指定了属性的存储单元,其值才是可以理解的。

一个名字的属性包括类型和作用域。名字的类型决定了它能具有什么样的值,值在计算机内的表示方式,以及对它能施加什么运算。名字的作用域规定了它的值的存在范围。

除了初等数据类型外,有些语言还提供了由初等数据构造复杂数据的手段。常见的复杂数据有:

(1) 数组,一个数组是由同一类型数据所组成的 n 维矩形结构。数组在内存中占有一块连续的空间,系统采用基地址加偏移量的方式访问数组元素。

(2) 记录,记录是由已知类型的数据组合起来的一种结构。一个记录通常含有若干个分量,每个分量称为记录的一个栏(或域 field)。每个分量都是一个确定类型的数据,不同分量的数据类型可以不同。

(3) 字符串、表格、栈和队列。

(4) 抽象数据类型,抽象数据类型封装了数据和操作,在面向对象程序设计语言中,Ada 通过程序包(Package)提供了数据封装的支持,Smalltalk、C + + 和 Java 语言则通过类(Class)对抽象数据类型提供支持。

1.1.4 函数调用的方式

程序是由函数或过程构成的,程序的任务是通过函数或过程之间的协作(相互调用)来完成的,函数或过程的调用有以下 4 种方式:传地址(Call by Reference)、得结果(Call by Result)、传值(Call by Value)和传名(Call by Name)。以下面例子为例:

定义函数 Swap,其中 M、N 是形式参数,简称形参。

```
void  Swap (int M,int N) {
      int t;
      t = N;
      N = M;
      M = t;
}
```

函数调用:

Swap(I,J);其中 I 和 J 是实在参数,简称实参。下面我们分别讨论 4 种参数传递的方式。

(1) 传地址　今天的高级语言基本上都实现了传地址的参数传递方式。所谓传地址是指把实在参数的地址传递给相应的形式参数。在函数段中每个形式参数都有一个对应的单元,称为形式单元。形式单元将用来存放相应的实在参数的地址。当调用一个函数时,调用段必须预先把实在参数的地址传递到一个被调用段可以拿得到的地方。当程序控制转入被调用段之后,被调用段首先把实参地址抄进自己相应的形式单元中,函数体对形式参数的任何引用或赋值都被处理成对形式单元的间接访问。当调用段工作完毕返回

时,形式单元(它们都是指示器)所指的实在参数单元就持有所指望的值。对于传地址,上面函数调用的参数传递过程如图 1.1 所示。

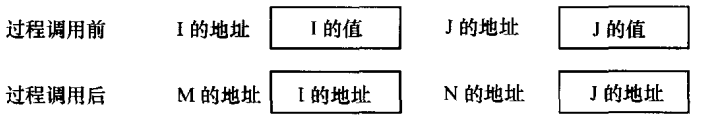

图 1.1 传地址的参数调用

图 1.1 中参数调用后,形式参数 M 的值是 I 的地址,N 的值是 J 的地址,这样通过 M 的值可以间接访问到 I 的值,通过 N 的值可以间接访问到 J 的值,函数调用返回后,I 和 J 的值被交换。

(2) 得结果 和传地址相似(但不等价)的另一种参数传递方法是所谓得结果。这种方法的实质是,每个形式参数对应有两个单元,第 1 个单元存放实参的地址,第 2 个单元存放实参的值。在函数体中对形参的任何引用或赋值都被看成是对它的第 2 个单元的直接访问。但在过程工作完毕返回前必须把第 2 个单元的内容存放到第 1 个单元所指的那个实参单元之中。对于得结果,上面函数调用的参数传递过程如图 1.2 所示。图中参数调用后,形式参数 M 的值是 I 的值,N 的值是 J 的值,但在函数中无法访问到 I 和 J 所在的内存单元,只有在函数返回前将 M 和 N 的值分别抄送到 I 和 J 所在的内存单元中。这种方式的函数调用,I 和 J 的值也被交换。

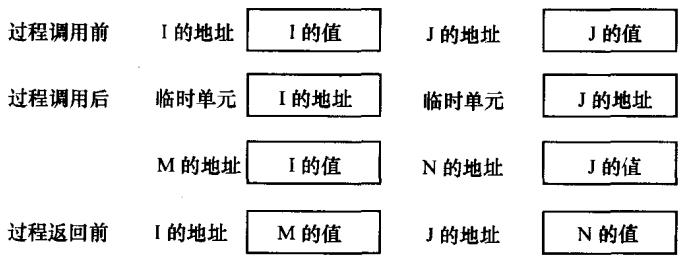

图 1.2 得结果的参数调用

(3) 传值 是一种简单的参数传递方法,也是今天高级语言中使用最多的参数传递方法之一,调用段把实在参数的值计算出来并存放在一个被调用段可以拿得到的地方。被调用段开始工作时,首先把这些值抄入到形式单元中,然后就好像使用局部名一样使用这些形式单元。如果实在参数不为指示器(指针),那么,在被调用段中无法改变实参的值。对于传值,上面函数调用的参数传递过程如图 1.3 所示。

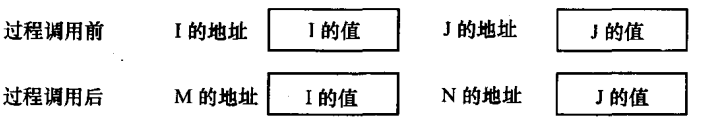

图 1.3 传值的参数调用

图 1.3 中参数调用后,形式参数 M 的值是 I 的值,N 的值是 J 的值,但在函数中无法改变 I 和 J 的值,函数调用结束后,I 和 J 的值也不会改变。

(4) 传名　传名是 ALGOL 60 所定义的一种特殊的形—实参数结合方式。过程调用的作用相当于把被调用段的过程体抄到调用出现的地方,但把其中任一出现的形式参数都替换成相应的实在参数(文字替换)。它与采用"传地址"或"传值"的方式所产生的结果均不相同。对于传名,上面函数调用的参数传递过程如图 1.4 所示。

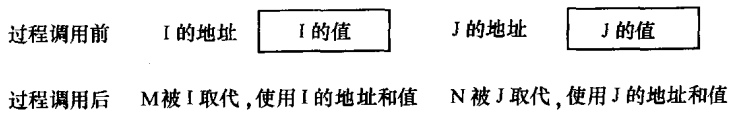

图 1.4　传名的参数调用

图 1.4 中参数调用后,形式参数 M 被 I 所取代,N 被 J 所取代,函数中对 M 和 N 的引用都是直接对 I 和 J 的引用。这种方式的函数调用实际上可以改写为:

```
t = J;
J = I;
I = t;
```

1.1.5　编译程序

通常所说的翻译程序是指这样的一个程序,它能够把某一种语言程序(称为源语言程序)转换成另一种语言程序(称为目标语言程序),而后者与前者在逻辑上是等价的。如果源语言是诸如 FORTRAN、PASCAL、C、Ada、Smalltalk 或 Java 这样的"高级语言",而目标语言是诸如汇编语言或机器语言之类的"低级语言",这样的一个翻译程序就称为编译程序。

编译程序的工作,即从输入源程序开始到输出目标程序为止的整个过程,是非常复杂的。通常,编译程序的工作过程可以划分为 5 个阶段:

第 1 阶段,词法分析。词法分析的任务是输入源程序,对构成源程序的字符串进行扫描和分解,识别出一个个的单词(亦称单词符号或简称符号),如保留字、标志符、算符、界符等。它依循语言的构词规则。

第 2 阶段,语法分析。语法分析的任务是在词法分析的基础上,根据语言的语法规则,把单词符号串分解成各类语法单位(语法范畴),如语句、程序块、函数等,判断整个输入串是否是一个语法上正确的"程序"。它依循语言的语法规则。

第 3 阶段,语义分析与中间代码产生。这一阶段的任务是对语法分析所识别出的各类语法范畴,分析其含义,并进行初步翻译(产生中间代码)。常用的中间代码有三元式、四元式、间接三元式、逆波兰式。本阶段依循的是语言的语义规则。

第 4 阶段,优化。优化的任务在于对前段产生的中间代码进行加工变换,以期在最后

阶段能产生出更为高效(省时间和空间)的目标代码,优化采用的主要方法有公共子表达式的提取、循环优化等。它依循程序的等价变换规则。

第 5 阶段,目标代码生成。这一阶段的的任务是把中间代码(或经优化处理之后的中间代码)变换成特定机器上的低级语言代码。

编译程序在分析源语言程序的过程中将源语言程序的各种信息都保留在各种表格中,同时,在源语言程序出错时,编译程序要帮助进行错误判断、错误定位等,因此,表格管理及出错处理和编译程序的各个阶段都有关联。

编译程序的结构图如图 1.5 所示。

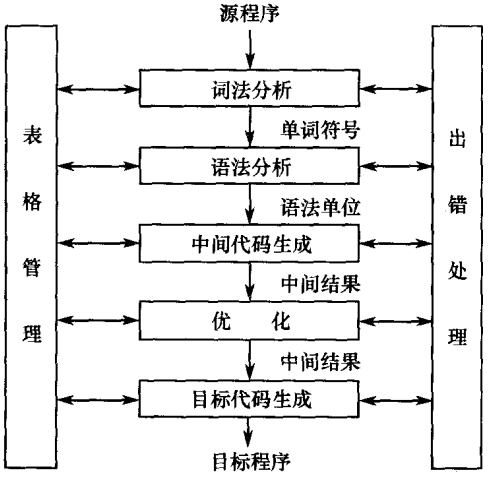

图 1.5　编译程序结构

1.1.6　程序语言的语法描述

1.基本概念

设∑是一个有穷字母表,它的每个元素称为一个符号。∑上的一个符号串是指由∑中的符号所构成的一个有穷序列。不包含任何符号的序列称为空字,记为 ε。∑ * 表示∑上的所有符号串的全体,空字 ε 也包括在其中。∅表示不含任何元素的空集。

$\sum^*$ 的子集 U 和 V 的(连接)积定义为

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

V 自身的 n 次(连接)积记为

$$V^n = \underbrace{VV\cdots V}_{n}$$

规定 $V^0 = \{\varepsilon\}$。令

$$V^* = V^0 \cup V^1 \cup V^2 \cup V^3 \cup \cdots$$

称 V^* 是 V 的闭包。记 $V^+ = VV^*$,称 V^+ 是 V 的正则闭包。

闭包 V^* 中的每个符号串都是由 V 中的符号串经有限次连接而成的。

2. 上下文无关文法

上下文无关文法是描述高级语言语法的基本工具，上下文无关文法有足够的能力描述现今多数程序设计语言的语法结构。

这里有许多重要的概念需要深入理解，包括：

1) 上下文无关文法严格的定义

形式上说，一个上下文无关文法 G 是一个四元式(V_T, V_N, S, $\mathcal{P}$)，其中

V_T是一个非空有限集，它的每个元素称为终结符号；

V_N是一个非空有限集，它的每个元素称为非终结符号，$V_T \cap V_N = \varnothing$；

S 是一个非终结符号，称为开始符号；

$\mathcal{P}$是一个产生式集合(有限)，每个产生式的形式是 $P \rightarrow \alpha$，其中，$P \in V_N$，$\alpha \in (V_T \cup V_N)^*$。开始符号 S 至少必须在某个产生式的左部出现一次。

2) 关于文法应用的若干基本概念

严格地说，我们称 $\alpha A \beta$ 直接推出 $\alpha\gamma\beta$，即

$$\alpha A \beta \Rightarrow \alpha\gamma\beta$$

仅当 $A \rightarrow \gamma$ 是一个产生式，且 α、$\beta \in (V_T \cup V_N)^*$。如果 $\alpha_1 \Rightarrow \alpha_2 \Rightarrow \cdots \Rightarrow \alpha_n$，我们则称这个序列是从 α_1 至 α_n 的一个推导。若存在一个从 α_1 至 α_n 的推导，则称 α_1 可推导出 α_n。我们用 $\alpha_1 \overset{+}{\Rightarrow} \alpha_n$ 表示从 α_1 出发，经一步或若干步，可推导出 α_n。而用 $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$ 表示从 α_1 出发，经 0 步或若干步，可推导出 α_n。换言之，$\alpha \overset{*}{\Rightarrow} \beta$ 意味着，或者 $\alpha = \beta$，或者 $\alpha \overset{+}{\Rightarrow} \beta$。

假定 G 是一个文法，S 是它的开始符号。如果 $S \overset{*}{\Rightarrow} \alpha$，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)。

$$L(G) = \{\alpha \mid S \overset{+}{\Rightarrow} \alpha \,\&\, \alpha \in V_T^*\}$$

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。类似地，可定义最右推导。

3) 语法分析树与二义性的概念

我们可以用一张图表示一个句型的推导，这种表示称为语法树。一棵语法树是一些不同推导过程的共性抽象。如果使用最左(右)推导，则一个最左(右)推导与语法树一一对应。

在文法中，一个句型不一定只对应惟一的一棵语法树。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。

需要特别指出的是，文法的二义性和语言的二义性是两个不同的概念。一个语言是二义性的，如果这个语言不存在无二义性的文法。可见，语言的二义性是一个更强的概念。对于某个语言 L 来说，可能存在两个文法 G 和 G'，其中一个为二义的，另一个为无

二义的，但 $L(G) = L(G') = L$，这时，语言 L 不是二义的。可以证明，二义性问题是不可判定问题，即不存在一个算法，它能在有限步骤内，确切地判定一个文法是否是二义的。但是我们可以找到一组无二义文法的充分条件。

3. 乔姆斯基(Chomsky)形式语言体系

乔姆斯基于 1956 年建立了形式语言体系，为语法的描述和分析提供了坚实的理论基础。乔姆斯基把文法分成 4 种类型，即 0 型、1 型、2 型和 3 型。就文法的描述能力来说，0 型强于 1 型，1 型强于 2 型，2 型强于 3 型。这几类文法的差别在于对产生式施加不同的限制。

我们说 $G = (V_T, V_N, S, \mathscr{P})$ 是一个 0 型文法，如果它的每个产生式

$$\alpha \rightarrow \beta$$

是这样的一种结构：$\alpha \in (V_N \cup V_T)^*$ 且至少含有一个非终结符，而 $\beta \in (V_N \cup V_T)^*$。0 型文法也称短语文法，其能力相当于图灵(Turing)机。或者说，任何 0 型语言都是递归可枚举的；反之，递归可枚举集必定是一个 0 型语言。

1 型文法也称上下文有关文法。该类文法的任何产生式 $\alpha \rightarrow \beta$ 均满足 $|\alpha| \leqslant |\beta|$(其中 $|\alpha|$ 和 $|\beta|$ 分别为 α 和 β 的长度)；仅仅 $S \rightarrow \varepsilon$ 例外，但 S 不得出现在任何产生式的右部。这种文法意味着，对非终结符进行替换时务必考虑上下文(上下文相关)，并且，一般不允许替换成空串 ε。例如，假若 $\alpha A \beta \rightarrow \alpha \gamma \beta$ 是 1 型文法 G 的一个产生式，α 和 β 都不空，则非终结符 A 只有在 α 和 β 这样的一个上下文环境中才可以把它替换为 γ。

2 型文法也称上下文无关文法。文法中任何产生式形为 $A \rightarrow \beta$，$A \in V_N$，$\beta \in (V_N \cup V_T)^*$。从产生式的形式可以看出，非终结符的替换是不用考虑上下文的。2 型文法对应非确定的下推自动机。因此，语法分析时，我们通常使用下推表(先进后出存区或栈)的有限自动机来分析上下文无关语言。

3 型文法也称正规文法。3 型文法有两种形式，一种称为右线性文法，其产生式形如 $A \rightarrow \alpha B$ 或 $A \rightarrow \alpha$，其中 $\alpha \in V_T^*$，$A、B \in V_N$。另一种称为左线性文法，其产生式形如 $A \rightarrow B\alpha$ 或 $A \rightarrow \alpha$，其中 $\alpha \in V_T^*$，$A、B \in V_N$。3 型文法等价于正规式(在第 2 章中介绍)，所以也称正规文法。

需要指出的是，程序设计语言不是上下文无关语言，甚至不是上下文有关语言。因此用上下文无关文法甚至上下文有关文法都不能完整地描述程序设计语言。例如，很多程序设计语言规定，当引用一个标识符时，要求这个标识符是被“说明”过的；又如，在函数或子程序的调用过程中，要实现“形—实参数的对应性”(如个数，顺序和类型一致性)。这些约束和规定都无法仅用上下文无关手段来实现。

但是，尽管如此，从编译程序设计的角度看，现今程序设计语言的语言结构，用上下文无关文法描述就足够了。因此，编译程序仍然以上下文有关文法作为描述程序设计语言语法的基本工具。也就是说，程序设计语言中能用上下文有关文法描述的部分我们采用上下文有关文法来描述，这样编译程序可以采用成熟、高效的分析方法；程序设计语言中