编译原理

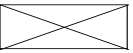
第二章 高级语言及其语法描述

颜波

复旦大学计算机科学技术学院
byan@fudan.edu.cn

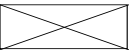
本章概述

- 高级语言的一般特性
- 程序语言的语法描述



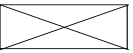
常用的高级语言

- 常用的高级语言
  - FORTRAN　　数值计算
  - COBOL　　事务处理
  - PASCAL　　结构程序设计
  - ADA　　大型程序、嵌入式实时系统
  - PROLOG　　逻辑程序设计
  - ALGOL　　算法语言
  - C/C++　　系统程序设计
  - Java　　Internet程序设计



高级语言的优点

- 与机器语言或汇编语言比较，高级语言的优点：
  - 较接近于数学语言和工程语言，面向应用，比较直观、自然和易于理解；
  - 便于验证其正确性，易于改错；
  - 对于程序员而言，编写效率高；
  - 易于移植。



2.1 程序语言的定义

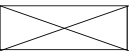
- 程序语言是一个记号系统，由两方面定义：
  - 语法：词法规则、语法规则；
  - 语义：单词符号和语法单位的意义；

- 语法：
  - 程序本质上是一定字符集上的字符串。
  - 语法：一组规则，用它可以形成和产生一个合式(well-formed)的程序。

语　法

- 词法规则：单词符号的形成规则。
  - 单词符号是语言中具有独立意义的最基本结构。一般包括：常数、标识符、基本字、算符、界符等。
  - 描述工具：有限自动机

- 语法规则：如何从单词符号形成更大的结构（语法单位），即语法单位的形成规则。
  - 语法单位通常包括：表达式、语句、分程序、过程、函数、程序等；
  - 描述工具：上下文无关文法



语法和语义

- E→i

  E→E+E

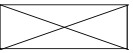
  E→E*E

  E→(E)

- 语法规则和词法规则定义了程序的形式结构。定义语法单位的意义属于语义问题。

语义

- 对于一个语言而言，不仅要给出其词法和语法规则，还要定义其单词符号和语法单位的意义。
- 离开了语义，语言只是一堆符号的集合。
- 各种语言中有形式上完全相同的语法单位，但是含义却不尽相同。
- 对某种语言，可以定义一个程序意义的一组规则称为语义规则。
- 大多数编译程序使用基于属性文法的语法制导翻译方法来分析语义。



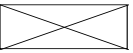
复旦大学 Fudan University

程序语言的基本功能和层次结构

- 程序语言的基本功能：
  - 描述数据和对数据的运算。
  - 所谓程序，本质上说是描述一定数据的处理过程。
- 程序的层次结构

逻辑和实现意义

- 程序语言每个组成成分的逻辑和实现意义
- 抽象的逻辑的意义
  - 数学意义
  - 比如：表示实数的名字，在逻辑上，是一个变量或用于保存实数的场所。
- 计算机实现的意义
  - 在计算机内实现的可能性和效率；
  - 比如：表示实数的名字，在计算机实现上，就是一个或若干个相继的存储单元。
    - 每位都有特殊的解释（符号位等等）
    - 可表示一定大小和精度的数值



2.2 高级语言的一般特性

- 高级语言的分类
  - 强制式语言(Imperative Languge)也称过程式语言：命令驱动，面向语句
    - FORTRAN、C、Pascal，Ada
  - 应用式语言（Applicative Language）：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
    - LISP、ML
  - 基于规则的语言（Rule-based Language）：检查一定的条件，当它满足值，则执行适当的动作
    - Prolog，逻辑程序设计语言
  - 面向对象语言（Object-Oriented Language）：封装性、继承性和多态性
    - C++，Java

2.2 高级语言的一般特性

程序结构

- FORTRAN
  - 一个程序由一个主程序段和若干辅程序段组成。
  - 辅程序段可以是子程序、函数段或数据块。
  - 每个程序段有一系列的说明语句和执行语句组成。各段可以独立编译。
  - 模块结构，没有嵌套和递归
  - 各程序段中的名字相互独立，同一个标识符在不同的程序段中代表不同的名字。
  - Common Area，全局性

```
主程序   PROGRAM ...
            ...
         end
辅程序1  SUBROUTINE ...
            ...
         end
辅程序2  SUBROUTINE ...
            ...
         end
```

程序结构 - PASCAL

- PASCAL
  - PASCAL程序本身可以看成是一个操作系统所调用的过程，过程可以嵌套和递归。
  - 一个PASCAL过程：

```
过程头;
    说明段（由一系列的说明语句组成）；
begin
    执行体（由一系列的执行语句组成）；
end
```

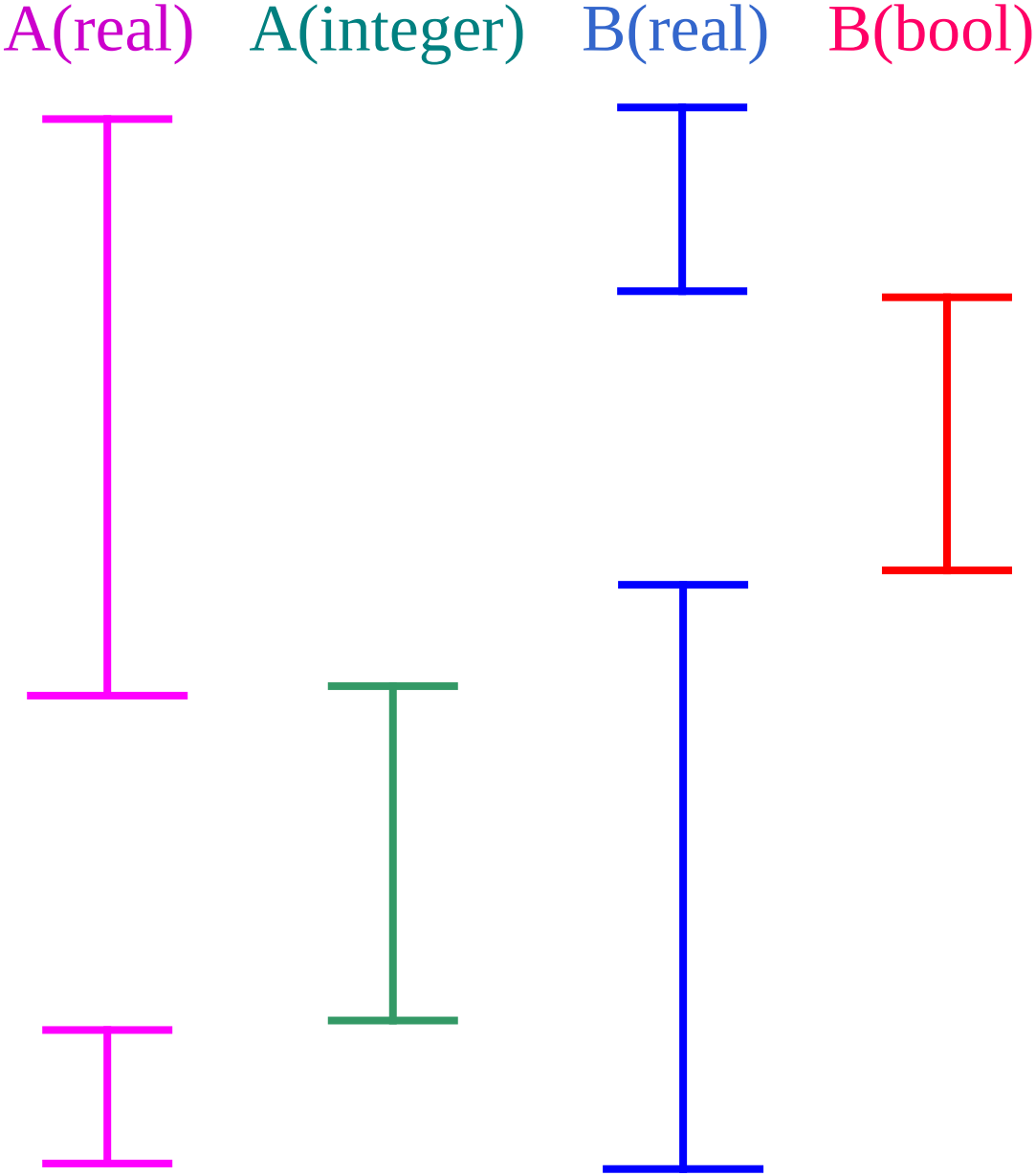
程序结构 - PASCAL

- 作用域：一个名字能被使用的区域范围称作这个名字的作用域。
- 允许同一个标识符在不同的过程中代表不同的名字。
- 名字作用域规则--"最近嵌套原则"
  - 一个在子程序B1中说明的名字X只在B1中有效（局部于B1）；
  - 如果B2是B1的一个内层子程序且B2中对标识符X没有新的说明，则原来的名字X在B2中仍然有效。如果B2对X重新作了说明，那么，B2对X的任何引用都是指重新说明过的这个X。
  - 标示符X的任一出现，都意味着引用某一说明句所说明的X，此说明句同所出现的X共处在一个最小子程序中。
- PASCAL提供了丰富的数据类型和运算方式，它允许用户动态地申请和退还存贮空间。

```
program main
   var  A, B : real;
   …
  procedure P1
    var B:boolean;
          …
  begin
          …

  end
  procedure P2
    var A:integer;
          …
  begin
          …

  end
begin
   …
end
```



程序结构 - JAVA

- JAVA
  - Java是一种面向对象的高级语言
    - 类（Class）
    - 继承(Inheritance)
    - 多态性(Polymorphism)和动态绑定(Dynamic binding)

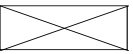
```
class Car{
  int color_number;
  int door_number;
  int speed;
  …
  push_break ( ) {
  …
  }
  add_oil ( ) {
  …
  }
}

class Trash_Car extends car {
  double amount;
  fill_trash ( ) {
  …
  }
}
```

2.2.3 数据类型与操作

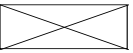
- 一个数据类型通常包括以下三种要素：
  - 用于区别这种类型数据对象的属性(类型)
  - 这种类型的数据对象可以具有的值
  - 可以作用于这种类型的数据对象的操作

- 初等数据类型
  - 数值类型：整型、实型、复数、双精度，可进行运算（+，−，*，/）
  - 逻辑类型：布尔运算：∨，∧，¬
  - 字符类型：字符型或字符串型的数据，用于符号处理
  - 指针类型：其值指向另一些数据。



标识符与名字

- 标识符：以字母开头的，由字母数字组成的字符串。
- 标识符与名字两者有本质区别：
  - 标识符是语法概念
  - 名字有确切的意义和属性
- 名字：
  - 值：单元中的内容
  - 属性：类型
- 名字的性质的说明方式：
  - 由说明语句来明确规定的
  - 隐含说明：FORTRAN 以I,J,K,…N为首的名字代表整型，否则为实型。
  - 动态确定：在程序运行时，动态地确定。



复旦大学 Fudan University

二 数据结构

数组

- 逻辑上，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，称为下标。
  - 确定数组和可变数组：编译时能否确定其存贮空间的大小。
  - 访问：给出数组名和下标值
  - 存放方式:
    - 按行存放(C，PASCAl)：扫描数组时，后面的下标变化的快。
    - 按列存放(FORTRAN)：扫描数组时，前面的下标变化的快。

数组元素地址计算

- 数组A[10,20]的A[1，1]为a，各维下标为1，按行存放，那么A[i，j]地址为：

$$a+(i-1)*20+(j-1)$$

- 数组元素地址计算公式

设A为$d_1 \times d_2 \times \cdots \times d_n$的n维数组，各维下限均为$l_i$，各维上限均为$u_i$，$d_i = u_i - l_i + 1$，按行存放，则数组元素$A(i_1, i_1, \cdots, i_n)$的地址D为：

$$D = a + (i_1 - 1)d_2 \cdots d_n + (i_2 - 1)d_3 \cdots d_n + \cdots + (i_{n-1} - 1)d_n + (i_n - 1)$$
$$= CONSPART + VARPART$$

其中：

$$CONSPART = a - C$$

$$C = d_2 \cdots d_n + d_3 \cdots d_n + \cdots + d_n + 1$$

$$VARPART = i_1 d_2 \cdots d_n + i_2 d_3 \cdots d_n + \cdots + i_{n-1} d_n + i_n$$
$$= (\cdots((i_1 d_2 + i_2)d_3 + i_3) \cdots i_{n-1})d_n + i_n$$

内情向量

- 把数组的有关信息记录在一个“内情向量”中，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。

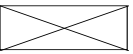
l_1	u_1	d_1
l_2	u_2	d_2
…	…	…
l_n	u_n	d_n
n	C	
$type$	a	

2 数据结构 - 记录

- 逻辑上说，记录结构由已知类型的数据组合在一起的一种结构。

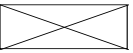
```
record {    char NAME[20];
            integer  AGE;
            bool  MARRIED;
        } CARD[1000]
```

- 访问：复合名 CARD[k].NAME
- 存储：连续存放
- 域的地址计算：相对于记录结构起点的相对数OFFSET。



3 字符串、表格、栈

- 字符串：符号处理、公式处理
- 表格：本质上是一种记录结构
- 线性表：一组顺序化的记录结构
- 栈：一种线性表，后进先出，POP, PUSH



三　抽象数据类型

- 抽象象数据类型(Abstract Data Type 简称ADT)是指一个数学模型以及定义在此数学模型上的一组操作。

- 作用：抽象数据类型可以使我们更容易描述现实世界。例：用线性表描述学生成绩表，用树或图描述遗传关系。

- 关键：使用它的人可以只关心它的逻辑特征，不需要了解它的存储方式。定义它的人同样不必要关心它如何存储。
  - 例：线性表这样的抽象数据类型，其数学模型是：数据元素的集合，该集合内的元素有这样的关系：除第一个和最后一个外，每个元素有唯一的前趋和唯一的后继。可以有这样一些操作：插入一个元素、删除一个元素等。

- 一个抽象数据类型包括：
  - 数据对象的一个集合；
  - 作用于这些数据对象的抽象运算的集合；
  - 这种类型对象的封装，即，除了使用类型中所定义的运算外，用户不能对这些对象进行操作。

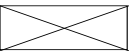
- 程序设计语言对抽象数据类型的支持
  - C++和Java语言则通过类（Class）对抽象数据类型提供支持。

复旦大学 Fudan University

2.2.4 语句与控制结构

- 表达式
  - 表达式由运算量（也称操作数，即数据引用或函数调用）和算符（操作符）组成。
  - 形式：中缀、前缀、后缀

    X*Y　　-A　　P↑
  - 表达式形成规则
    - 变量、常数是表达式；
    - 若E1和E2为表达式，α是一个二元运算符，则E1αE2是表达式（采用中缀形式）；
    - 若E是表达式，α是一元算符，则αE或Eα是表达式；
    - 若E是表达式，则(E)是表达式。



算符的优先次序

- 一般的规定
  - PASCAL：左结合A+B+C=(A+B)+C
  - FORTRAN：对于满足左、右结合的算符可任取一种，如A+B+C就可以处理成(A+B)+C，也可以处理成A+(B+C)。

- 注意两点：
  - 代数性质能引用到什么程度视具体的语言不同而不同；
  - 在数学上成立的代数性质在计算机上未必完全成立。如：交换率一般都成立，但结合率和分配率未必成立。

复旦大学 Fudan University

二. 语句

- 名字的两个特征：
  - 名字左值：该名字代表的那个单元（地址）称为该名字的左值。（所代表的存贮单元的地址）
  - 右值：一个名字的值称为该名字的右值。（所代表的存贮单元的内容）
- 赋值语句：　　A := B
  - 赋值号左右两边的变量名扮演着不同的角色；
  - 赋值号左边的表达式必须持有左值；
  - 赋值号右边的表达式必须持有右值。

```
#include "stdafx.h"

int main(int argc, char* argv[])
{
    int x, y;
    x+y = 10;

    return 0;
}
```

```
--------------------Configuration: test - Win32 Debug--------------------
Compiling...
test.cpp
D:\test\test.cpp(9) : error C2106: '=' : left operand must be l-value
Error executing cl.exe.

test.exe - 1 error(s), 0 warning(s)
```


复旦大学 Fudan University

语句

- 控制语句:
  - 无条件转移语句

    goto L
  - 条件语句

    if B then S

    if B then S1 else S2
  - 循环语句

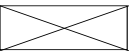
    while B do S

    repeat S until B

    for i:=E1 step E2 until E3 do S
  - 过程调用语句

    call P(X1, X2, ... ,Xn)
  - 返回语句

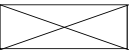
    return (E)



语句

- 说明语句：
  - 定义各种不同数据类型的变量或运算，定义名字的性质。
  - 编译程序将这些性质登记在符号表中。

- 简单句和复合句
  - 简单句：不包含其他语句成分的基本句
  - 复合句：句中有句的语句
    - If A then B else C;



复旦大学 Fudan University

2.3 程序语言的语法描述

- 几个概念:
  - 考虑一个有穷 字母表∑ 字符集
  - 其中每一个元素称为一个字符
  - ∑上的字(也叫字符串)是指由∑中的字符所构成的一个有穷序列

    例 某个字母表

    ∑={a,b,c,…z,if,then,else,main,1,2,3,4,…,9,0,=,==,>,<,;},

    则建立在∑上的符号串有：if (2+3==5) then a=6 else b=8;
  - 不包含任何字符的序列称为空字，记为ε
  - 用$\sum^*$表示∑上的所有字的全体,包含空字ε

例如: 设 ∑={a， b}，则 $\sum^*$={ε,a,b,aa,ab,ba,bb,aaa,...}

- $\sum^*$的子集U和V的连接（积）定义为

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

UV中的符号串是由U和V的符号串连接而成的。

- 设：$U = \{ a, aa \}$

$V = \{ b, bb \}$

那么：

$UV = \{ ab, abb, aab, aabb \}$

- UV 和VU之间的关系。

- 对于字母表上的任何符号串x，都有εx=xε=x

- $\sum^*$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次积记为

$$V^n = VV\ldots V$$

- 规定$V^0=\{\varepsilon\}$,
  令$V^*=V^0 \cup V^1 \cup V^2 \cup V^3 \cup \ldots$　称V^*是V的闭包;
  - 由V符号组成的所有串的集合。

- 记 $V^+ = VV^*$，称V^+是V的正则闭包。
  - 由V符号组成的所有串的集合（不包括空字ε）。

- 设:　　$U = \{ a, aa \}$
- 那么:

$$U^* = \{ \varepsilon, a, aa, aaa, aaaa, \ldots \}$$

$$U^+ = \{ a, aa, aaa, aaaa, \ldots \}$$

- 显然，$U^* = U^0 \cup U^+$ ，且$U^+ = UU^* = U^*U$。

2.3.1 上下文无关文法

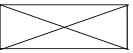
- 文法：
  - 描述语言的语法结构的形式规则（语法规则）
  - 文法作为一种工具，不仅严格地定义句子的结构，也是为了用适当条数的规则把语言的全部句子描述出来，是以有穷的集合刻划无穷的集合的工具。
- 上下文无关文法
  - 所定义的语法范畴（或语法单位）是完全独立于这种范畴可能出现的环境。
  - 比如：算术表达式处理，不必考虑上下文；
  - 自然语言中，词和句子的语法性质和上下文有密切关系；
  - 上下文无关文法不适于描述自然语言，但对于程序语言是够用的。
- He gave me a book.

〈句子〉 → 〈主语〉〈谓语〉〈间接宾语〉〈直接宾语〉　　（产生或定义为）
〈主语〉 → 〈代词〉
〈谓语〉 → 〈动词〉
〈间接宾语〉 → 〈代词〉
〈直接宾语〉 → 〈冠词〉〈名词〉
〈代词〉 → He
〈代词〉 → me
〈名词〉 → book
〈冠词〉 → a
〈动词〉 → gave

复旦大学 Fudan University

<句子> → <主语><谓语><间接宾语><直接宾语>
<主语> → <代词>
<谓语> → <动词>
<间接宾语> → <代词>
<直接宾语> → <冠词> <名词>
<代词> → He
<代词> → me
<名词> → book
<冠词> → a
<动词> → gave

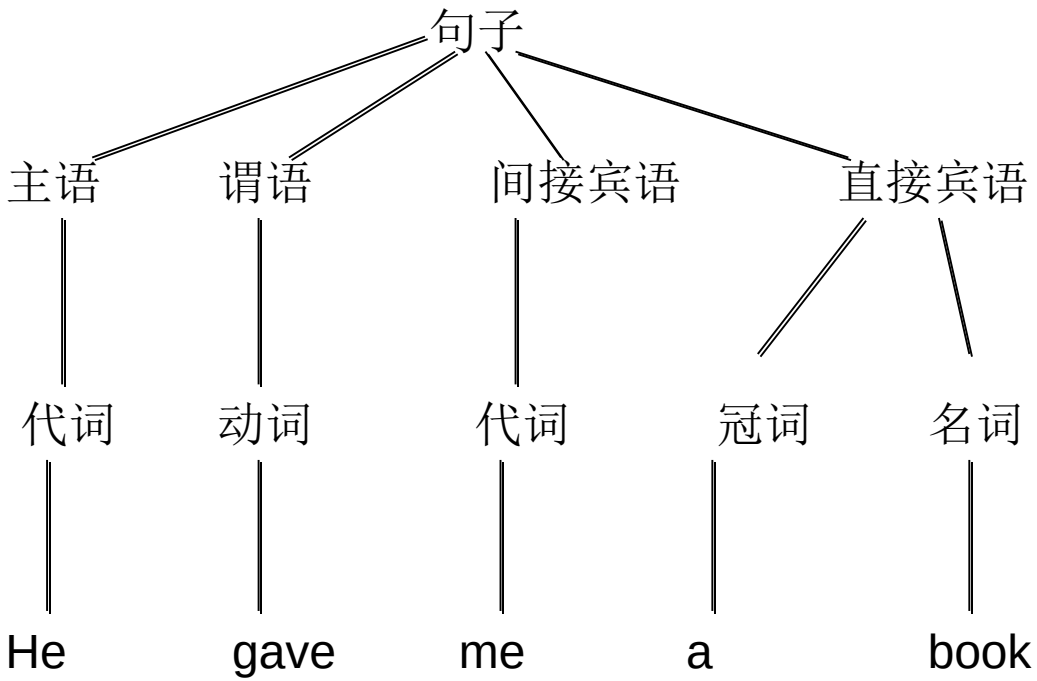
<句子>
⇒<主语><谓语><间接宾语><直接宾语>
⇒<代词><谓语><间接宾语><直接宾语>
⇒He <谓语><间接宾语><直接宾语>
⇒He <动词><间接宾语><直接宾语>
⇒He gave <间接宾语><直接宾语>
⇒He gave <代词><直接宾语>
⇒He gave me <直接宾语>
⇒He gave me <冠词><名词>
⇒He gave me a <名词>
⇒He gave me a book



复旦大学 Fudan University

语法分析

- 语法分析树: 用一种图示化的方法来表示这种推导



- 上下文无关文法的定义：

一个上下文无关文法G是一个四元式

$G=(V_T, V_N, S, P)$，其中

- V_T：终结符集合(非空)：语言中不可再分割的字符串。
- V_N：非终结符集合(非空)，且$V_T \cap V_N=\varnothing$
- S：文法的开始符号，$S\in V_N$
- P：产生式集合(有限)，每个产生式形式为

  $P\rightarrow\alpha$，$P\in V_N$，$\alpha \in (V_T \cup V_N)^*$
  - 是用来定义符号串之间关系的一组(语法)规则。
  - 形式：$A \rightarrow \alpha$（A产生α）
  - 如：<句子> → <主语><谓语>
- 开始符S至少必须在某个产生式的左部出现一次。

- 例，定义只含+，*的算术表达式的文法

  G=<{i，+，*，(，)}，{E}，E，　P>，　其中，P由下列产生式组成：

  $E \rightarrow i$

  $E \rightarrow E+E$

  $E \rightarrow E*E$

  $E \rightarrow (E)$

- 几点规定：
  - “ $\rightarrow$ ”也可以用“::="表示， 这种表示称为巴科斯范式(BNF)
  - $P \rightarrow \alpha_1$
    $P \rightarrow \alpha_2$ 可缩写为 $P \rightarrow \alpha_1|\alpha_2|\ldots|\alpha_n$
    …
    $P \rightarrow \alpha_n$
    其中，“|”读成“或”，称为P的一个候选式。
  - 一般，用大写字母A,B,C…或汉语词组（如，算术表达式）代表非终结符号，用小写字母a,b,c…代表终结符，用α、β、γ等代表由终结符和非终结符组成的符号串。
  - 表示一个文法时，通常只给出开始符号和产生式，如上例，可表示为：
    G(E)： $E \rightarrow i \mid E+E \mid E*E \mid (E)$

- **例:**在程序设计语言中，假设我们定义标识符的命名规则为字母a、b、c开头的，字母a、b、c和数字1、2、3的序列。命名规则为:

  <标识符>→<字母>

  <标识符>→<标识符><字母>

  <标识符>→<标识符><数字>

  <字母>→a

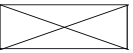
  <字母>→b

  <字母>→c

  <数字>→1

  <数字>→2

  <数字>→3



复旦大学 Fudan University

- 一般用大写字母代表左边的非终结符，设N 代表<标识符>，D代表<数字>，L代表<字母>，则定义标识符的文法是：

  G=(V_T， V_N，N， P)

  其中， V_N ={N，L，D}　V_T ={a,b,c,1,2,3}　N 是开始符号.

  P为产生式的规则：

  {N→L，　N→NL ，N→ND ，L→a ，L→b ，L→c ，D→1 ，D→2，D→3}

- 上面的产生式规则可以改写为：

  {N→L|NL|ND

  L→a|b| c

  D→1|2|3}

- 定义：称$\alpha A\beta$直接推出$\alpha\gamma\beta$，即

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

仅当$A \to \gamma$是一个产生式，

且α， $\beta \in (V_T \cup V_N)^*$。

- 如果$\alpha_1 \Rightarrow \alpha_2 \Rightarrow \ldots \Rightarrow \alpha_n$，则我们称这个序列是从$\alpha_1$到$\alpha_n$的一个推导。若存在一个从$\alpha_1$到$\alpha_n$的推导，则称$\alpha_1$可以推导出$\alpha_n$。

- 对文法G(E)： $E \to i \mid E+E \mid E*E \mid (E)$

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

- 通常，用 $\alpha_1 \overset{+}{\Rightarrow} \alpha_n$ 表示：从α_1出发，经过一步或若干步，可以推出α_n。

用 $\alpha_1 \overset{*}{\Rightarrow} \alpha_n$ 表示：从α_1出发，经过0步或若干步，可以推出α_n。

所以： $\alpha \overset{*}{\Rightarrow} \beta$ 即 $\alpha = \beta$ 或 $\alpha \overset{+}{\Rightarrow} \beta$

定义：假定G是一个文法，S 是它的开始符号。如果 $S \overset{*}{\Rightarrow} \alpha$ 则α称是一个句型。仅含终结符号的句型是一个句子。

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

实质上，句子是句型的特殊情况，句子是由终结符组成，而句型是有终结符和非终结符组成。

语言是由S开始通过1步或1步以上推导所得的句子的集合。将它记为 L(G)。

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

- 例： (i*i+i)是文法
  G(E)： $E \rightarrow i \mid E+E \mid E*E \mid (E)$
  的一个句子。

证明：

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

E，(E)，(E*E+E)，…，(i*i+E)是句型。

例：文法G1(A)：A → c|Ab，求G1(A)的语言?

L(G1)={c，cb，cbb，…}

以c开头，后继若干个b

例：文法G (S)：$S \rightarrow aS$,

$S \rightarrow a$,

$S \rightarrow b$,

求G (S)的语言?

$L(G)=\{a^i(a|b)|i\geq 0\}$

- 例：文法$G_2(S)$：

  $S \rightarrow AB$

  $A \rightarrow aA|a$

  $B \rightarrow bB|b$

  $G_2(S)$的语言?

$L(G_2)=\{a^m b^n | m，n>0\}$

- 例：给出产生语言为$\{a^n b^n | n \geq 1\}$的文法。

$G_3(S)$:

$S \rightarrow aSb$

$S \rightarrow ab$

例：设$L_1=\{a^{2n}b^n|n>=1$ 且$a,b \in V_T\}$试构造生成L_1的文法G_1

- 设n=1， L_1 =aab
- n=2， L_1 =aaaabb
- n=3， L_1 =aaaaaabbb
- ……
- 所以得：S $\rightarrow$ aaSb
- S $\rightarrow$ aab

例 构造一个上下文无关文法G，使其描述的语言L（G）是能够被5整除的无符号整数集合。

能够被5整除的整数其结构特点是，末位数一定是0或5。所以，只要保证生成的整数末位数字是0或5即可。据此，构造描述能被5整除的无符号整数集合的文法如下：

G[S]：

S→N0|N5

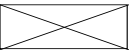
N→DN|ε

D→0|1|2|3|4|5|6|7|8|9

例 文法G[S]定义如下

S→if E then S| if E then S else S|while E do S|begin L end|A

该文法产生的语言就是程序设计语言中的单分支、双分支、循环语句和顺序语句，其中每个非终结符的意义是：S代表语句，L代表语句串、A代表赋值语句，E代表布尔表达式。



复旦大学 Fudan University

- 从一个句型到另一个句型的推导往往不唯一。

$E+E \Rightarrow i+E \Rightarrow i+i$　　$E+E \Rightarrow E+i \Rightarrow i+i$

- 最左推导：任何一步$\alpha \Rightarrow \beta$都是对α中的最左非终结符进行替换。

最右推导：任何一步$\alpha \Rightarrow \beta$都是对α中的最右非终结符进行替换。

最右推导和最左推导

例 给出了下列文法G

（1）<无正负号整数>→<数字序列>

（2）<数字序列>→<数字序列><数字>| <数字>

(3) <数字> → 0|1|2|3|4|5|6|7|8|9

VT ={0,1,2,3,4,5,6,7,8,9}　VN ={<无正负号整数> , <数字序列>,<数字>}

判断数据2634是否是C语言合法的数据。

【解】

（1）用最右推导，每次用产生式的规则替换最右边的非终结符，推导过程如下：

<无正负号整数> ⇒ <数字序列>⇒<数字序列><数字>⇒<数字序列>4

⇒<数字序列><数字>4⇒<数字序列>34⇒<数字序列><数字>34

⇒<数字序列>634⇒2634

（2）用最左推导，每次直接推导都替换最左边的非终结符，推导过程如下：

<无正负号整数> ⇒ <数字序列>⇒<数字序列><数字>⇒<数字序列><数字><数字>

⇒<数字序列><数字><数字><数字>⇒<数字><数字><数字><数字>

⇒2<数字><数字><数字>

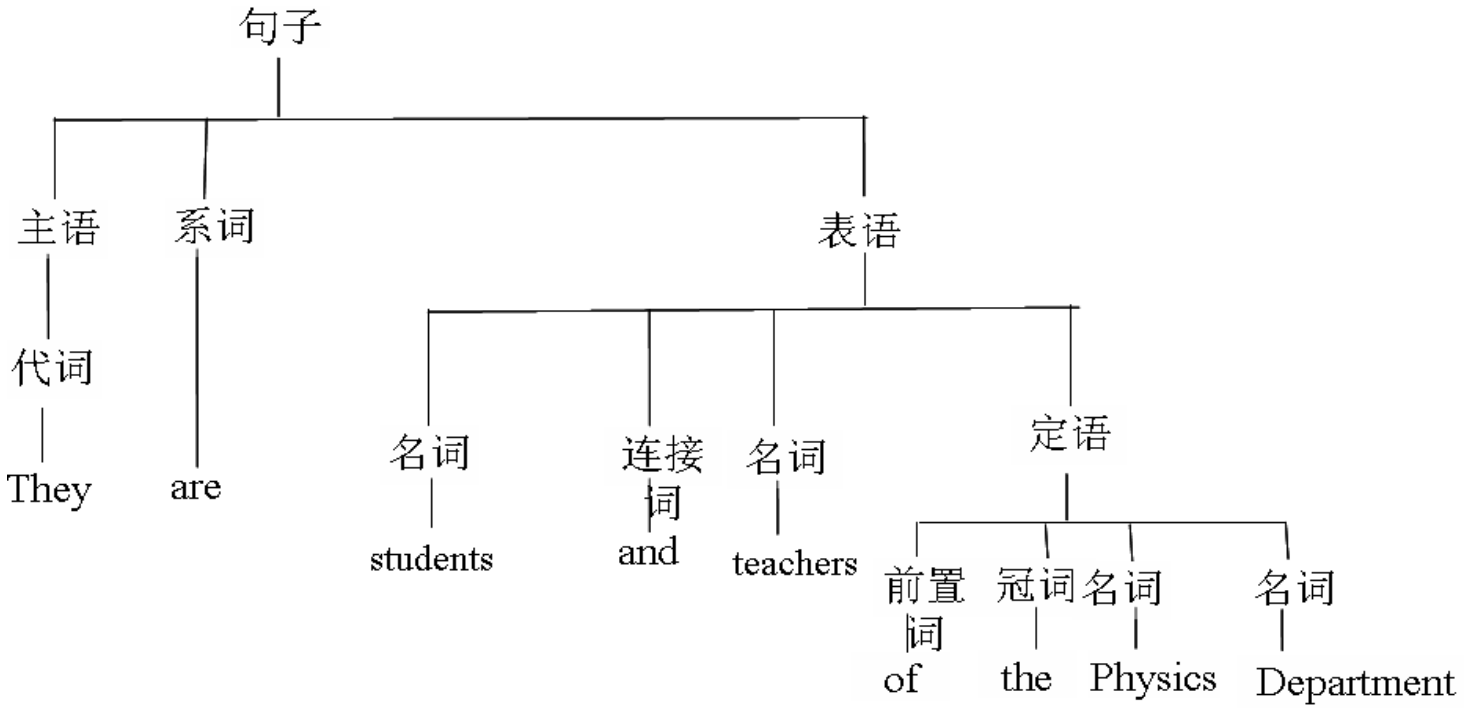
⇒26<数字><数字>

⇒263<数字>

⇒2634

2.3.2 语法树

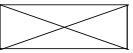
- 在自然语言中，句子结构可以借助一种树形表示进行分析。如下面的句子：
  - They are students and teachers of the Physics Department。
  - 对该句子的结构进行分析，其树型结构如图所示，由此可以看出，该句子是由主语、系词和表语组成，是一个语法正确的句子。



语法树

- 在自然语言中，可以通过树型表示直观地分析句子的结构；

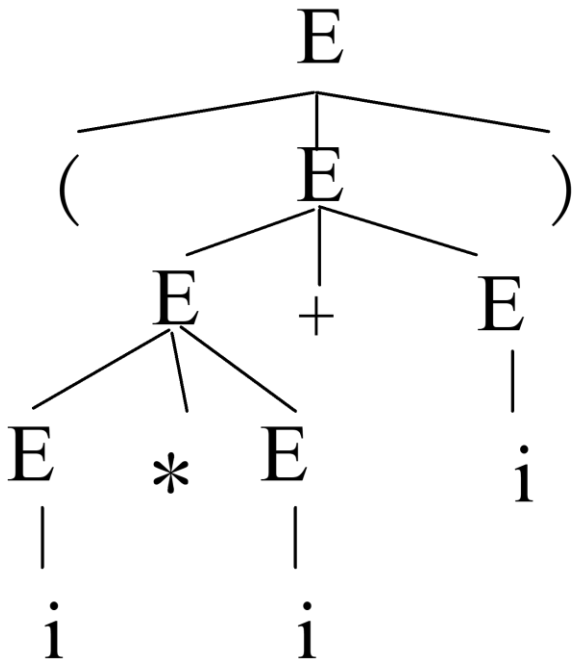
- 在形式语言中，我们提到了句型、推导的概念，在证明某个符号串是否是某个文法的句型时，采用从文法开始符号推导的方法，这个推导过程可以用语法树直观的表示出来。



2.3.2 语法树与二义性

- 用一张图表示一个句型的推导，称为语法树。
- (i*i+i)的语法树

G(E)： E → i | E+E | E*E | (E)
(i*i+i)

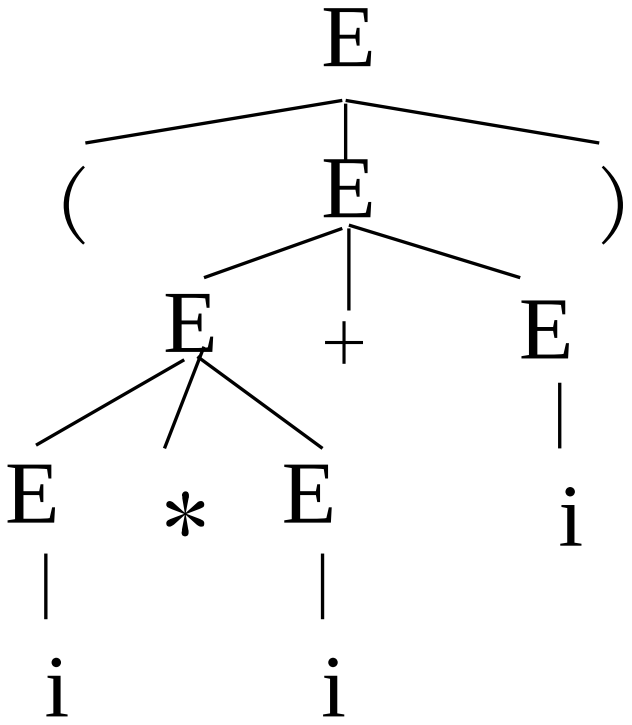


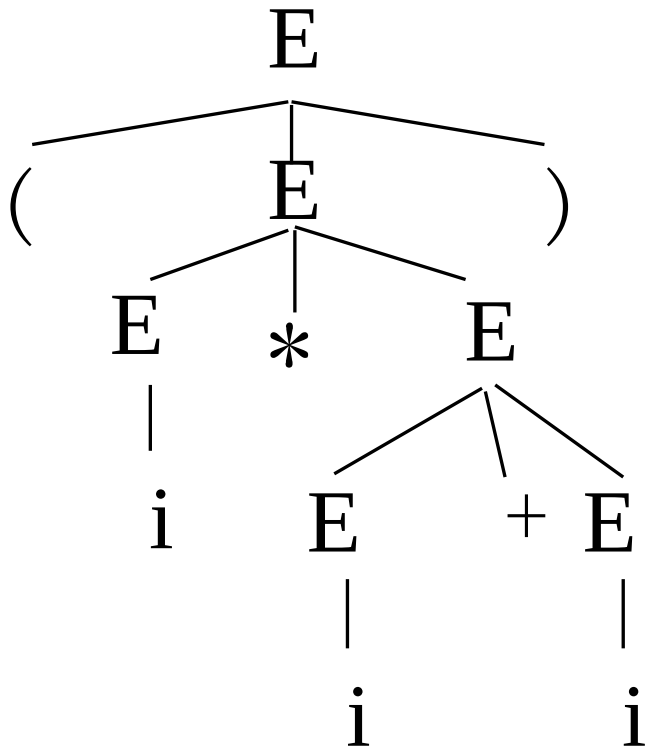
$$E \Rightarrow (E) \Rightarrow (E+E) \Rightarrow (E*E+E) \Rightarrow (i*E+E) \Rightarrow (i*i+E) \Rightarrow (i*i+i)$$

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

- 一棵语法树是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。
- 一个句型是否只对应唯一一棵语法树?





- 定义：如果一个文法存在某个句子对应两颗不同的语法树，则说这个文法是二义的。

  G(E)： $E \rightarrow i|E+E|E*E|(E)$ 是二义文法。

- 语言的二义性：如果对语言存在二义性的文法，该语言未必是二义性的。
  - 可能存在G和G’，一个为二义的，一个为无二义的。但L(G)=L(G’)，即语言相同。
- 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。
- 二义性文法的证明：我们要证明一个文法是否是一个二义性文法，就是找到该文法的一个句型特例，能够画出这个句型的两棵语法树，该文法就是二义性文法。

二义文法

- 假若规定了运算符‘十’与‘*’的优先顺序和结合规则．比方说，让‘*’的优先性高于‘+’，且它们都服从左结合，那么，就可以构造出一个无二义文法：

二义文法：

G(E)： E → i|E+E|E*E|(E)

表达式 →项|表达式+项

项 → 因子 | 项*因子

因子 → (表达式) | i

无二义文法：

G(E)： E → T | E+T

T → F | T*F

F → (E) | i

无二义文法：

G(E)：E → T | E+T

T → F | T*F

F → (E) | i

考虑句子(i*i+i)

E ⇒T

⇒F

⇒(E)

⇒(E+T)

⇒(T+T)

⇒(T*F+T)

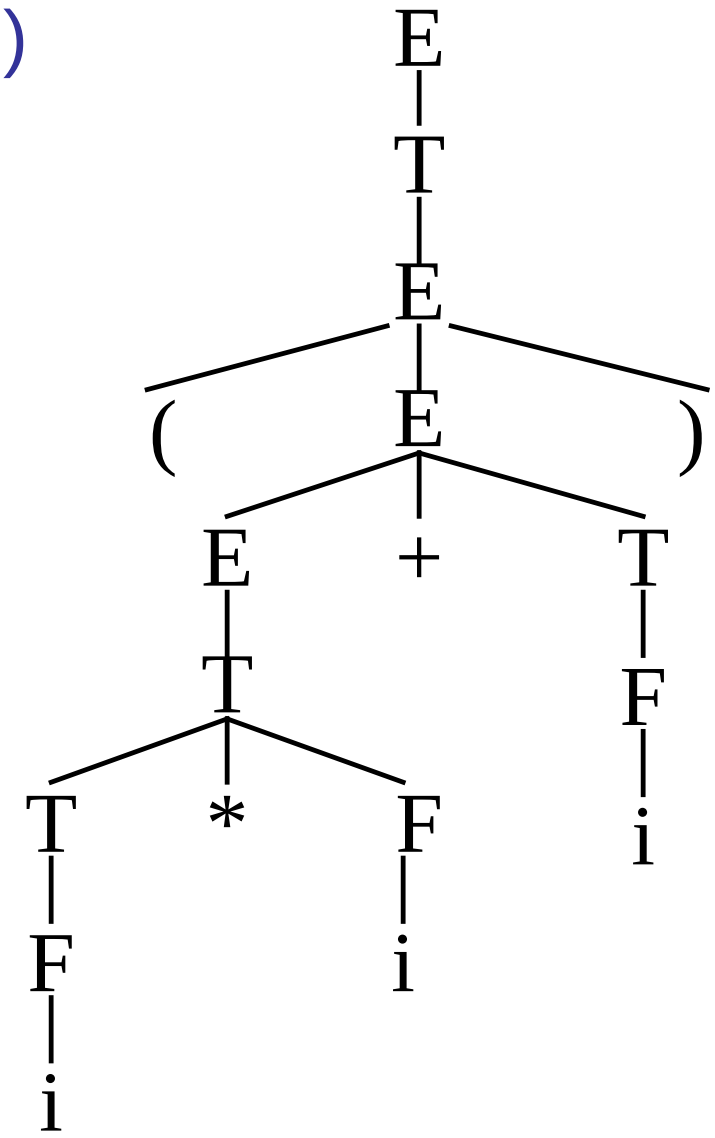
⇒(F*F+T)

⇒(i*F+T)

⇒(i*i+T)

⇒(i*i+F)

⇒(i*i+i)



二义文法

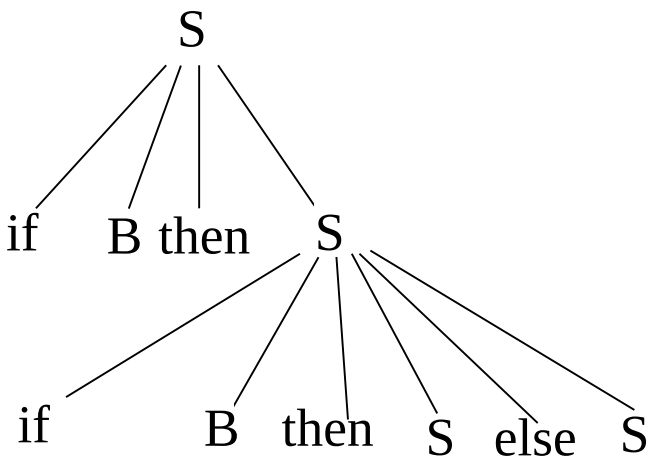
例 设文法G[S]:

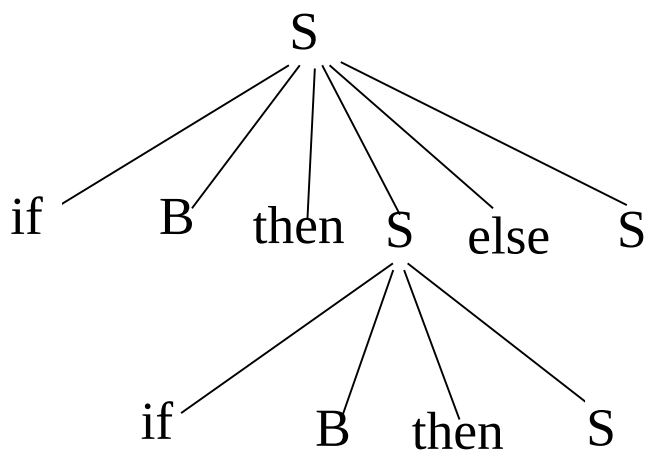
S→if B then S | if B then S else S

给出符号串if B then if B then S else S的语法树。

语法树的结构如图所示。

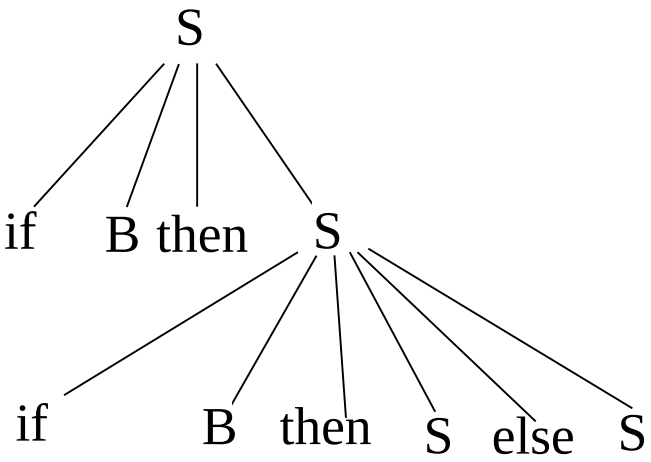
- 从上面的语法图我们可以看出，字符串if B then if B then S else S能够画出两棵语法树，所以该文法是一个二义性文法。

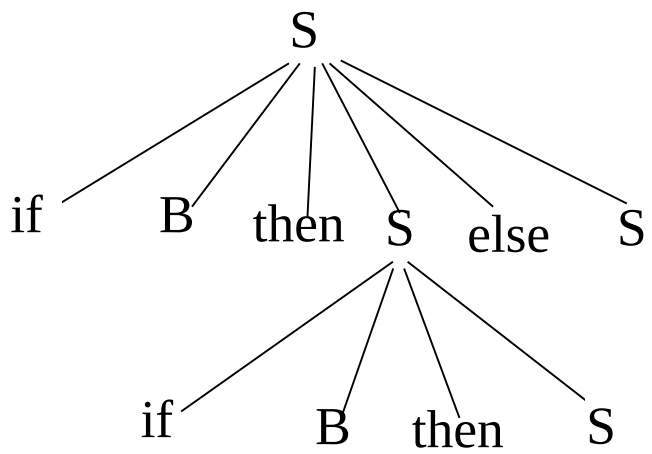




二义文法

- 在语言中，为了避免二义性的文法，往往对文法加以一定的限制，从语义解释方面限制条件语句中的else只能与其前面的、还没有和其他else配对的then配对。如此限制之后，符号串if B then if B then S else S就只有图左边的树形结构了。





- 描述程序设计语言时，对于上下文无关文法的限制：
  - 不含P→P形式的产生式，会引起二义性。
  - 每个非终结符P必须有用处
    - 必须存在含P的句型；$S \overset{*}{\Rightarrow} \alpha P \beta$
    - P不存在永不终结的回路。$P \overset{*}{\Rightarrow} r \qquad r \in V_T^*$

2.3.3 形式语言鸟瞰

- Chomsky于1956年建立形式语言体系，他把文法分成四种类型：0，1，2，3型。

- 与上下文无关文法一样，它们都由四部分组成，但对产生式的限制有所不同。

文法的分类

- 0型(短语文法，图灵机):
  - 产生式形如： $\alpha \rightarrow \beta$
  - 其中：$\alpha \in (V_T \cup V_N)^*$且至少含有一个非终结符；$\beta \in (V_T \cup V_N)^*$
    - 识别0型语言的自动机称为图灵机(TM);
    - 0型文法是对产生式限制最少的文法;
    - 对0型文法产生式的形式作某些限制，可得到其他类型文法的定义。

- 1型(上下文有关文法，线性界限自动机):
  - P中产生式$\alpha \rightarrow \beta$,除可能有$S \rightarrow \varepsilon$ 外均有$|\beta| >= |\alpha|$,若有$S \rightarrow \varepsilon$，规定S不得出现在产生式右部 .
    - 1型文法又称为长度增加文法、上下文有关文法;
    - 识别1型语言的自动机称为线性界限自动机(LBA) ;
    - 1型文法意味着，对非终结符进行替换时务必考虑上下文，并且，一般不允许替换成ε，除非是开始符号产生ε

文法的分类

- 2型(上下文无关文法，非确定下推自动机)：
  - P中产生式具有形式$A \to \beta$ 其中$A \in V_N$， $\beta \in (V_T \cup V_N)^*$.
  - 2型文法对产生式的要求是：产生式左部一定是单个非终结符，产生式右部可以是V_N、 V_T或ε； 非终结符的替换不必考虑上下文;
  - 识别2型语言的自动机称为下推自动机(PDA);
  - 2型文法也称为上下文无关文法，也就是当用β取代非终结符A时，与A所在的上下文无关。上下文无关文法有足够的能力描述现今的程序设计语言。

- 3型(正规文法，有限自动机)：
  - 右线性文法：

    产生式形如： $A \to \alpha B$ 或 $A \to \alpha$

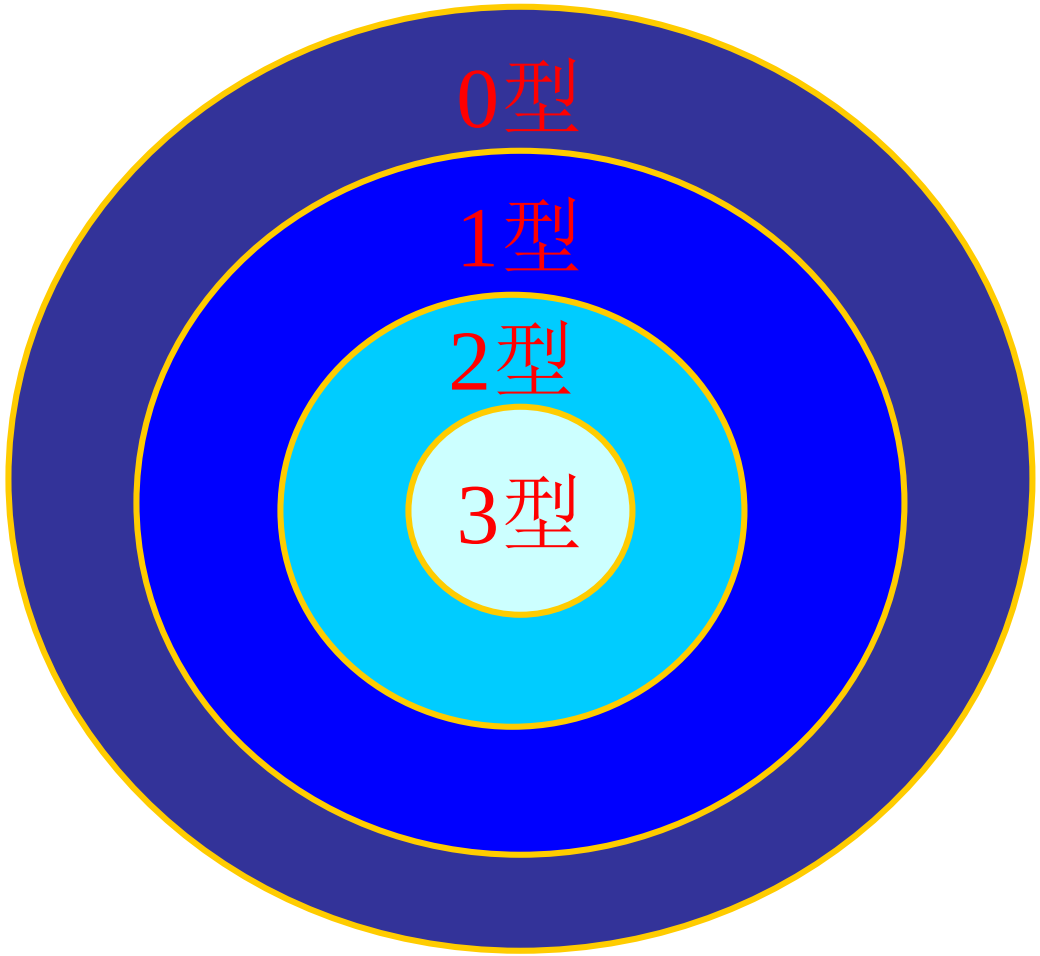
    其中： $\alpha \in V_T^*$； A，$B \in V_N$

    对2型文法的产生式做进一步的限制，限制产生式右部是终结符或终结符跟着单一非终结符
  - 左线性文法

    产生式形如： $A \to B\alpha$ 或 $A \to \alpha$

    其中： $\alpha \in V_T^*$； A，$B \in V_N$
  - 识别3型语言的自动机称为有限状态自动机;

四种类型描述能力比较



(最近原则)

- 正规文法的能力比上下文无关文法弱的多。

  例：$L_5=\{a^nb^n|n\geq 1\}$ 不能由正规文法产生，但可由上下文无关文法产生：

  $G_5(S)$：　　$S \rightarrow aSb|\ ab$

- $L6=\{a^nb^nc^n|n\geq 1\}$不能由上下文无关文法产生，但可由上下文有关文法产生：

  G6(S)：　$S \rightarrow aSBC|\ aBC$

  $CB \rightarrow BC$

  $aB \rightarrow ab$

  $bB \rightarrow bb$

  $bC \rightarrow bc$

  $cC \rightarrow cc$

文法的分类

- 例: 给出文法G(S):

S→aSb|P　P→bPc|bQc　Q→Qa|a

（1）问该文法是Chomsky哪一类型的文法?

（2）它生成的语言是什么?

答(1)它符合2型文法的定义，所以该文法是上下文无关文法。

(2)由S推出的串的形式是$a^iPb^i(i\geq 0)$,

由P推出的串的形式为$b^jQc^j(j\geq 1)$,

Q推出的串的形式为$a^k(k\geq 1)$,

所以它生成的语言为:

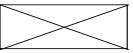
$$L=\{a^ib^ja^kc^jb^i|i\geq 0, j\geq 1, k\geq 1)$$

- $L_7=\{\alpha c\alpha \mid \alpha \in (a|b)^*\}$不能由上下文无关文法产生，甚至连上下文有关文法也不能产生，只能由0型文法产生。

- 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。

本章小结

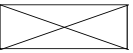
- 高级语言的一般特性
- 文法的书写
- 语法树
- 二义性
- 文法的分类



作业

- P36 - 6，7，8，9，10，11

- 第五周上课时间交



复旦大学 Fudan University