



编译原理与技术

--基本概念

刘爽

天津大学智算学部

Outline

- 程序语言的定义
 - 语法定义
 - 语义定义
- 高级语言的一般特性
 - 程序结构
 - 数据类型和操作
 - 语句与控制结构
- 程序语言的语法描述
 - 基本概念
 - 文法和文法的类型
 - 上下文无关文法及其语法树

程序语言的定义

- 程序语言的**语法定义**

- 所谓一个语言的**语法**是指这样一组规则，用它可以形成和产生一个**合式**（形式上正确）的程序。这些规则一部分称为**词法规则**，另一部分称为**语法规则**（或产生规则）

- **词法规则**：词法规则规定了字母表中什么样的字符串是一个单词符号，是**单词符号的形成规则**，描述工具为**有限自动机** 变量名，关键字，常量，算符，界符……

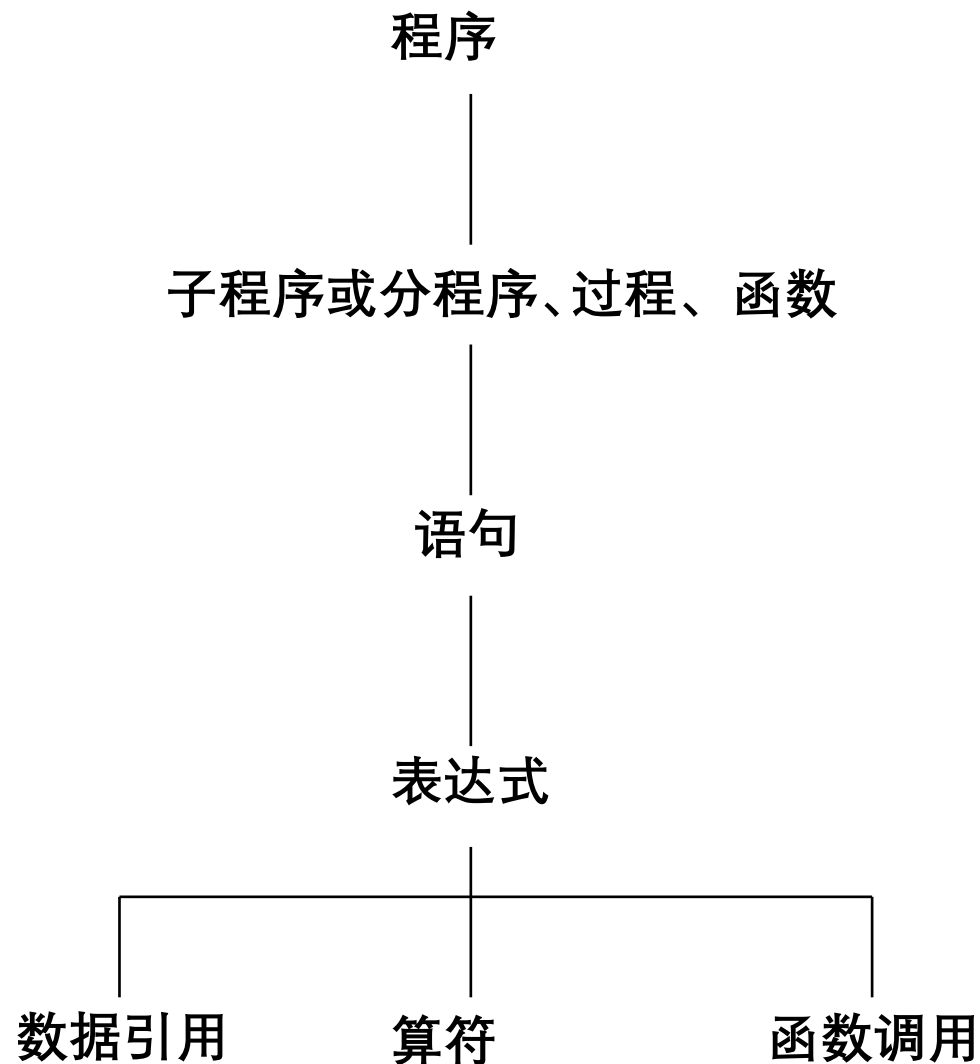
- **语法规则**：语言的语法规则规定了如何从单词符号形成更大的结构（即语法单位），换言之，语法规则是**语法单位（语法范畴）的形成规则**，描述工具为**上下文无关文法** 表达式，语句……

- 程序语言的**语义定义**

- 所谓一个语言的**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**。这些规则称为**语义规则**，可以用**形式语义描述**（操作语义，指称语义等）。

程序的一般结构

- 高级语言的程序结构
 - 程序：描述数据和数据运算
 - 表达式：描述数据运算的基本结构



Outline

- 程序语言的定义
 - 语法定义
 - 语义定义
- 高级语言的一般特性
 - 程序结构
 - 数据类型和操作
 - 语句与控制结构
- 程序语言的语法描述
 - 基本概念
 - 文法和文法的类型
 - 上下文无关文法及其语法树

高级程序语言结构

- 分类

- 强制式语言 (Imperative Language) /过程式语言
- 应用式语言 (Applicative Language) 或函数式语言 (Functional Language)
- 基于规则的语言 (Rule-based Language)
- 面向对象的语言 (Object-Oriented Language)

- 结构

- 程序, 子程序 (过程, 函数), 语句, 表达式
- 程序包, 类, 成员函数, 语句, 表达式

数据类型和操作

- 数据类型的要素：
 - 用于区别这种类型的数据对象的**属性**（类型，作用域等）；
 - 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**；
- 数据类型分类：
 - 初等数据类型：数值类型、逻辑类型、字符类型、指针类型
 - 数据结构：数组、记录、字符串、表格、栈、队列和抽象数据类型

语句与控制结构

- **表达式**：一个表达式是由**运算量**（操作数，即数据引用或函数调用）和**算符**组成的。
 - 例如：表达式 $X+Y$ 由二元算符‘+’和运算量‘ X ， Y ’组成。
- 形式
 - 前缀型：如 $-X$, $+XY$
 - 中缀型：如 $X+Y$
 - 后缀型：如 $P \uparrow$, $XY+$
- 形成规则
 - (1) **变量**，**常数**是表达式
 - (2) 若 E_1 , E_2 为表达式， Θ 是一个二元算符，则 $E_1 \Theta E_2$ 是表达式
 - (3) 若 E 是表达式， Θ 为一元算符，则 ΘE (或 $E \Theta$)是表达式
 - (4) 若 E 是表达式，则 **(E)** 是表达式

语句与控制结构

- **优先级**：大多数语言中，算术算符和逻辑算符的优先顺序符合下述规定（优先级由高到低，同级列在一行）
 - 乘幂（ \wedge 或者 $**$ 或者 \uparrow ）
 - 一元负（ $-$ ）
 - 乘、除（ $*$, $/$ ）
 - 加、减（ $+$, $-$ ）
 - 关系符（ $<$, $=$, $>$, $<=$, $<>$, $>=$ ）
 - 非(not, \neg)
 - 与（ \wedge , and, $\&$ ）
 - 或（ \vee , or, $|$ ）
 - 等值（ $=$, \sim ）

语句与控制结构

- **语句**：不同程序语言含有不同形式和功能的各种语句
 - 从形式上分，语句可以分为简单句、复合句和分程序等。
 - 从功能上分，语句可分为执行语句和说明性语句
 - **执行语句**：描述程序的动作
 - 赋值语句 $a := 5$
 - 控制语句（条件语句，循环语句，过程调用，返回语句）
 - 输入/输出语句；
 - **说明性语句**：定义各种不同数据类型的变量或运算

Outline

- 程序语言的定义
 - 语法定义
 - 语义定义
- 高级语言的一般特性
 - 程序结构
 - 数据类型和操作
 - 语句与控制结构
- 程序语言的语法描述
 - 基本概念
 - 文法和文法的类型
 - 上下文无关文法及其语法树

一些约定

- A, B, C, \dots 用来表示非终结符
- a, b, c, \dots 表示终结符
- \dots, X, Y, Z 可以用来表示终结符或者非终结符
- \dots, w, x, y, z 表示终结符号串
- $\alpha, \beta, \gamma, \delta, \dots$ 表示由终结符或非终结符构成的符号串
- 在产生式 $A \rightarrow \alpha$ 中,
 - A 是产生式的左边 (lefthand side, LHS)
 - α 是产生式的右边 (righthand side, RHS)
- $A \rightarrow \alpha_1 | \dots | \alpha_n$ 表示产生式 $A \rightarrow \alpha_1, \dots, A \rightarrow \alpha_n$

基本概念

- 字母表

- 字母表是符号的非空有限集，通常用 Σ 表示，例如 $\Sigma=\{a, b, c\}$ ，其中 a, b, c 是字母表中的元素，也称符号。

- 符号串

- 符号的有穷序列，例如 a, ab, aab, \dots 空符号串用 ϵ 表示。
- 形式定义：给定字母表 Σ ，
 - (1) ϵ 是 Σ 上的符号串；
 - (2) 若 x 是 Σ 上的符号串，且 $a \in \Sigma$ ，则 ax 或者 xa 是 Σ 上的符号串；
 - (3) y 是 Σ 上的符号串，当且仅当 y 可以由（1）和（2）产生。

- 符号串集合

- 由符号串构成的集合，例如 $\{ab, abc, \epsilon\}$
- Σ^* 表示 Σ 上的所有字的全体

基本概念 (cont) -- 符号串的运算

- 符号串**相等**

- 若 x, y 是集合 Σ 上的两个符号串, 则 $x=y$ 当且仅当组成 x 的每一个符号和组成 y 的每一个符号**依次相等**。

- 符号串的**长度**

- x 为符号串, 其长度 $|x|$ 等于组成该符号串的**符号个数**。

例如: $x=abcc, |x|=4$

- 符号串的**(连接) 积**

- 若 x, y 是定义在 Σ 上的符号串, 且 $x=XY, y=YZ$, 则 x 和 y 的连接积 $xy=XYYZ$ 也是 Σ 上的符号串。

注意: $\varepsilon x = x\varepsilon$

基本概念 (cont) -- 符号串集合的运算

- 符号串集合的 **乘积运算**

- 令 A, B 为符号串集合, 定义

$$AB = \{xy \mid x \in A, y \in B\}$$

- 例如: $A = \{s, t\}, B = \{u, v\}, AB = ?$ $\{su, sv, tu, tv\}$

- 注意: $\{\varepsilon\}A = A\{\varepsilon\}$, 如何证明?

Hint: $\varepsilon x = x\varepsilon$

- 符号串集合的 **幂运算**

- 有符号集合 A , 定义:

$$A^0 = \{\varepsilon\}, \quad A^1 = A, \quad A^2 = AA, \quad A^3 = AAA, \quad \dots \quad A^n = A^{n-1}A = AA^{n-1}, \quad n > 0$$

- 注意:** A^0 不是空集, $\varepsilon, \{\}, \{\varepsilon\}$ 的差别

基本概念 (cont) -- 符号串集合的运算

- 符号串集合的**闭包运算**:

- 设A是符号串集合, 定义

$$A^0 = \{\varepsilon\}$$

$$A^n = \underbrace{AA \dots A}_n$$

$A^+ = A^1 \cup A^2 \cup A^3 \dots \cup A^n \dots$ 为集合A的**正闭包 (正则闭包)**

$A^* = A^0 \cup A^+$ 为集合A的**闭包**

- 例: $A = \{x, y\}$

$$A^+ = \{x, y, xx, xy, yx, yy, xxx, xxy, xyx, xyy, yxx, yxy, yyx, yyy, \dots\}$$

$$A^* = \{\varepsilon, x, y, xx, xy, yx, yy, xxx, xxy, xyx, xyy, yxx, yxy, yyx, yyy, \dots\}$$

基本概念 (cont)

- 将字符看做**符号**，则单词就是**符号串**，单词集合就是**符号串的集合**
- 将单词看做**符号**，则句子就是**符号串**，而所有句子的集合（语言）就是**符号串的集合**
- 例如：若**A**为某语言的基本**字符集**（把字符看作符号）
 $A = \{a, b, \dots, z, 0, 1, \dots, 9, +, -, *, /, _, (,), =, \dots\}$
B为单词集
 $B = \{\text{if, else, while, for, switch, case, } \dots\}$
则 $B \subset A^*$

把单词看作符号，句子就是符号串。语言的句子是定义在B上的符号串，若令**C**为句子集合，则 $C \subset B^*$ ，**程序** $\subset C$

■ 思考

- $A=\{a, b\}$, $B=\{\epsilon, a, b\}$, $A^* A^+ B^* B^+$ 分别是什么?

- A^* 和 B^+ 有什么关系, 如何证明?

■ Outline

- 程序语言的定义
 - 语法定义
 - 语义定义
- 高级语言的一般特性
 - 程序结构
 - 数据类型和操作
 - 语句与控制结构
- 程序语言的语法描述
 - 基本概念
 - 文法和文法的类型
 - 上下文无关文法及其语法树

文法的直观概念

- **文法/语法**是描述语言的**语法结构的形式规则**（语法规则），即从形式上用于描述和规定语言结构的称为“**文法**”（或“**语法**”）。
- 例如：
 - “我是大学生”，是一个在语法上正确的句子，其句子结构（语法结构），本例为“主谓结构”，是由它的语法决定的。虽然该句子语义正确，但语法并未涉及语义信息。
 - “我是天大”，就是一个语法正确但语义错误的句子。

语法规则

- 一组用来描述句子语法结构的规则，规定：用“::=”表示“定义为”（或“由...组成”）。**注：有些规则用→表示“定义为”。**
 - $A::=\alpha$ ，A为非终结符，称为产生式的左边部分， α 是由终结符或/与非终结符组成的一串符号，称为产生式的右边部分。这种表示方法称为**巴科斯范式**（Backus Normal Form 缩写为**BNF**）。
 - 例如：
 - $\langle \text{句子} \rangle ::= \langle \text{主语} \rangle \langle \text{谓语} \rangle$
 - $\langle \text{主语} \rangle ::= \langle \text{代词} \rangle | \langle \text{名词} \rangle$
 - $\langle \text{代词} \rangle ::= \text{你} | \text{我} | \text{他}$
 - $\langle \text{名词} \rangle ::= \text{刘飞} | \text{程序员} | \text{大学生} | \text{法语}$
 - $\langle \text{谓语} \rangle ::= \langle \text{动词} \rangle \langle \text{直接宾语} \rangle$
 - $\langle \text{动词} \rangle ::= \text{是} | \text{学习}$
 - $\langle \text{直接宾语} \rangle ::= \langle \text{代词} \rangle | \langle \text{名词} \rangle$

基于规则的推导

<句子>::=<主语><谓语>
<主语>::=<代词>|<名词>
<代词>::=你|我|他
<名词>::=刘飞|程序员|大学生|法语
<谓语>::=<动词><直接宾语>
<动词>::=是|学习
<直接宾语>::=<代词>|<名词>

- 根据给定规则，推导产生句子
- **推导方法**：从一个**要识别的符号**开始推导，即用相应的规则**右边部分**来替代规则的**左边部分**，每次仅用一条规则进行推导。
- 例如：
 - <句子> 用 <主语><谓语> 代替
 - <主语><谓语> 用 <代词><谓语> 代替
 -
- 这种推导一直进行下去，直到所有带<>的符号都由**终结符号**替代为止。一般用“=>”表示**直接推出**（即一步推出）。

例子：“我是大学生”的推导

<句子> $\xRightarrow{1}$ <主语><谓语>
 $\xRightarrow{2}$ <代词><谓语>
 $\xRightarrow{3}$ 我<谓语>
 $\xRightarrow{5}$ 我<动词><直接宾语>
 $\xRightarrow{6}$ 我是<直接宾语>
 $\xRightarrow{7}$ 我是<名词>
 $\xRightarrow{4}$ 我是大学生

1 <句子>::=<主语><谓语>
2 <主语>::=<代词>|<名词>
3 <代词>::=你|我|他
4 <名词>::=刘飞|程序员|大学生|法语
5 <谓语>::=<动词><直接宾语>
6 <动词>::=是|学习
7 <直接宾语>::=<代词>|<名词>

练习

- 用上述文法练习推导句子“你学习法语”

1 $\langle \text{句子} \rangle ::= \langle \text{主语} \rangle \langle \text{谓语} \rangle$

2 $\langle \text{主语} \rangle ::= \langle \text{代词} \rangle | \langle \text{名词} \rangle$

3 $\langle \text{代词} \rangle ::= \text{你} | \text{我} | \text{他}$

4 $\langle \text{名词} \rangle ::= \text{刘飞} | \text{程序员} | \text{大学生} | \text{法语}$

5 $\langle \text{谓语} \rangle ::= \langle \text{动词} \rangle \langle \text{直接宾语} \rangle$

6 $\langle \text{动词} \rangle ::= \text{是} | \text{学习}$

7 $\langle \text{直接宾语} \rangle ::= \langle \text{代词} \rangle | \langle \text{名词} \rangle$

注意

- 语法/文法只在形式上对句子结构进行描述，未涉及语义
- 从一组语法规则可以推导出不同的句子，
例如：“他是法语”，“我学习工人”，等
这些句子语法正确，但语义未必正确

1 <句子>::=<主语><谓语>

2 <主语>::=<代词>|<名词>

3 <代词>::=你|我|他

4 <名词>::=刘飞|程序员|大学生|法语

5 <谓语>::=<动词><直接宾语>

6 <动词>::=是|学习

7 <直接宾语>::=<代词>|<名词>

文法的定义

- 文法 G 定义为四元组 (V_N, V_T, P, S)
 - 其中 V_N 为**非终结符号**（或语法实体，或变量）集；
 - V_T 为终结符号集； V_N 和 V_T 不含公共元素，即 $V_N \cap V_T = \emptyset$ 。通常 V 表示 $V_N \cup V_T$ ， V 称为文法 G 的**字母表**。
 - P 为产生式（也称规则）的集合； V_N , V_T 和 P 是**非空有穷集**。
 - S 称作**开始符号**(识别符号)，是一个非终结符 ($S \in V_N$)，至少要在一条规则中作为左部出现。

例： 文法 $G = (V_N, V_T, P, S)$

$V_N = \{ S \}$, $V_T = \{ 0, 1 \}$, $P = \{ S \rightarrow 0S1, S \rightarrow 01 \}$, S 为开始符号

文法可以简写，只需要指出开始符号和产生式即可，如 $G[S]: S \rightarrow 0S1, S \rightarrow 01$

注：上述文法的产生式集合 P ，若满足每个产生式的形式是 $A \rightarrow \alpha$ ，其中 $A \in V_N$ ， $\alpha \in (V_T \cup V_N)^*$ ，则称 G 为**上下文无关文法**。

关于文法的定义 (cont)

- 如 $\alpha \rightarrow \beta$ 是文法 $G=(V_N, V_T, P, S)$ 的规则 (即 P 中的一个产生式), γ 和 δ 是 V^* 中的任意符号串, 若有符号串 v, w 满足: $v=\gamma\alpha\delta, w=\gamma\beta\delta$, 则说 v (应用规则 $\alpha \rightarrow \beta$) 直接产生 w , 或说 w 是 v 的**直接推导**。 ($v \Rightarrow w$)

例: 文法 $G=(V_N, V_T, P, S)$

$V_N = \{ S \}, V_T = \{ 0, 1 \}, P = \{ S \rightarrow 0S1, S \rightarrow 01 \}, S$ 为开始符号

$S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 00001111$

$S \rightarrow 0S1, S \rightarrow 01$ 可以缩写成 $S \rightarrow 0S1 \mid 01$, 其中 $0S1, 01$ 称为 S 的一个**候选式**。

关于文法的定义 (cont)

- 如果存在直接推导的序列: $v=w_0 \Rightarrow w_1 \Rightarrow w_2 \cdots \Rightarrow w_n=w$, ($n>0$), 则称 v 推导出 (产生) w (推导长度为 n), 该序列是从 v 到 w 的一个**推导**, 记做 $v \stackrel{+}{\Rightarrow} w$ 。
- 若有 $v \stackrel{+}{\Rightarrow} w$, 或 $v=w$, 则记做 $v \stackrel{*}{\Rightarrow} w$ 。
- 规范推导 (最右推导)
 - **最左推导**: 若规则右端符号串中有两个以上的非终结符时, 先推导左边的。
 - **最右推导**: 若规则右端符号串中有两个以上的非终结符时, 先推导右边的。

关于文法的定义 (cont)

- 设 $G[S]$ 是一文法，如果符号串 x 是从识别符号推导出来的，即有 $S \xRightarrow{*} x$ ，则称 x 是文法 $G[S]$ 的**句型**。若 x 只由终结符号组成，则称 x 为 $G[S]$ 的**句子**。
- 文法 G 所产生的**语言**定义为集合 $\{x \mid S \xRightarrow{*} x, \text{ 其中 } S \text{ 为文法的开始符号, 且 } x \in V_T^*\}$ 。可用 $L(G)$ 表示该集合，即文法 G 产生的句子的全体。

例： $G[S]: S \rightarrow OS1, S \rightarrow 01$

$S \Rightarrow OS1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 00001111$

$L(G) = \{0^n 1^n \mid n \geq 1\}$

形式语言理论可以证明以下两点：

- (1) $G \rightarrow L(G)$; 已知文法，可以推导出语言；
- (2) $L(G) \rightarrow G1, G2, \dots, Gn$; 已知语言，构造文法，无形式化方法，凭经验。

关于文法的定义 (cont)

- 若 $L(G_1) = L(G_2)$, 则称文法 G_1 和 G_2 是等价的。
- 例1: 如文法 $G_1[A]$:
 $A \rightarrow 0R$
 $A \rightarrow 01$
 $R \rightarrow A1$ 与 $G_2[S]$:
 $S \rightarrow 0S1$
 $S \rightarrow 01$ 等价

编译感兴趣的问题是：

- 给定句子 x 以及文法 G ，求 $x \in L(G)$?
 - 即一个句子 x ，是不是属于文法 G 规定的文法范畴？
 - 例如 (1) x 为 “ $a>0?1:2$ ”， $G[S]: S \rightarrow 0S1, S \rightarrow 01, x \in L(G)$?
 - (2) x 为 “ 0011 ”， $G[S]: S \rightarrow 0S1, S \rightarrow 01, x \in L(G)$?
 - (3) x 为 “ 001101 ”， $G[S]: S \rightarrow 0S1, S \rightarrow 01, x \in L(G)$?

文法的类型

- Chomsky将文法分为四种类型：
 - 0型文法（短语文法）：对任一产生式 $\alpha \rightarrow \beta$ ，都有 $\alpha \in (V_N \cup V_T)^+$ ， $\beta \in (V_N \cup V_T)^*$
 - 1型文法（上下文有关文法）：对任一产生式 $\alpha \rightarrow \beta$ ，都有 $|\beta| \geq |\alpha|$ ，仅仅 $\alpha \rightarrow \varepsilon$ 除外
 - 2型文法（上下文无关文法）：对任一产生式 $\alpha \rightarrow \beta$ ，都有 $\alpha \in V_N$ ， $\beta \in (V_N \cup V_T)^*$
 - 3型文法（正规文法）：任一产生式 $\alpha \rightarrow \beta$ 的形式都为 $A \rightarrow aB$ 或 $A \rightarrow a$ ，其中 $A \in V_N$ ， $B \in V_N$ ， $a \in V_T$ 。上述叫做右线性文法，另有左线性文法，二者等价。

文法的类型举例

1型文法（上下文有关文法）：对任一产生式 $\alpha \rightarrow \beta$ ，都有 $|\beta| \geq |\alpha|$ ， 仅仅 $\alpha \rightarrow \varepsilon$ 除外

- 1型（上下文有关）文法

文法 $G[S]$:

$S \rightarrow CD$

$Ab \rightarrow bA$

$C \rightarrow aCA$

$Ba \rightarrow aB$

$C \rightarrow bCB$

$Bb \rightarrow bB$

$AD \rightarrow aD$

$C \rightarrow \varepsilon$

$BD \rightarrow bD$

$D \rightarrow \varepsilon$

$Aa \rightarrow aA$

$L(G) = \{ww \mid w \in \{a,b\}^*\}$

文法的类型举例

2型文法（上下文无关文法）：对任一产生式 $\alpha \rightarrow \beta$ ，都有 $\alpha \in V_N$ ， $\beta \in (V_N \cup V_T)^*$

- 2型（上下文无关）文法

文法G1[S]:
 $S \rightarrow aB | bA$
 $A \rightarrow a | aS | bAA$
 $B \rightarrow b | bS | aBB$

文法G2[S]:
 $S \rightarrow 0A | 1B | 0$
 $A \rightarrow 0A | 1B | 0S$
 $B \rightarrow 1B | 1 | 0$

文法的类型举例

3型文法（正规文法）：任一产生式 $\alpha \rightarrow \beta$ 的形式都为 $A \rightarrow aB$ 或 $A \rightarrow a$ ，其中 $A \in V_N$ ， $B \in V_N$ ， $a \in V_T$ 。

- 定义标识符的3型（正规）文法

文法 $G[S]$: $S \rightarrow aT$

$S \rightarrow a$

$T \rightarrow aT$

$T \rightarrow dT$

$T \rightarrow a$

$T \rightarrow d$

文法和语言

- 0型文法
 - 0型文法（短语文法）的能力相当于图灵机，可以表征任何递归可枚举集，而且任何0型语言都是递归可枚举的
- 1型文法（上下文有关文法）
 - 产生式的形式为 $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ ，即只有A出现在 α_1 和 α_2 的上下文中时，才允许 β 取代A。其识别系统是线性界限自动机。
- 2型文法（上下文无关文法）
 - 产生式的形式为 $A \rightarrow \beta$ ， β 取代A时与A的上下文无关。其识别系统是不确定的下推自动机。
- 3型文法（正规文法）
 - 产生的语言是有穷自动机（FA）所接受的集合

■ Outline

- 程序语言的定义
 - 语法定义
 - 语义定义
- 高级语言的一般特性
 - 程序结构
 - 数据类型和操作
 - 语句与控制结构
- 程序语言的语法描述
 - 基本概念
 - 文法和文法的类型
 - 上下文无关文法及其语法树

上下文无关文法

- **上下文无关文法**有足够的描述能力描述现今程序设计语言的语法结构
- **文法G** 定义为四元组 (V_N, V_T, P, S) ,其中文法的产生式集合P, 若满足每个产生式的形式是 $A \rightarrow \alpha$, 其中 $A \in V_N$, $\alpha \in (V_T \cup V_N)^*$, 则称G为**上下文无关文法**。
 - 算术表达式

• 文法 $G = (\{E\}, \{+, *, |, (,)\}, P, E)$
P:
 $E \rightarrow i$
 $E \rightarrow E + E$
 $E \rightarrow E * E$
 $E \rightarrow (E)$

- 语句
 - 赋值语句
 - 条件语句 $\langle \text{条件语句} \rangle \rightarrow \text{if} \langle \text{条件} \rangle \text{then} \langle \text{语句} \rangle \mid \text{if} \langle \text{条件} \rangle \text{then} \langle \text{语句} \rangle \text{else} \langle \text{语句} \rangle$
 -

上下文无关文法的语法树

- **语法树**：用于描述上下文无关文法的**句型推导**的直观方法

例: $G[S]$:

$S \rightarrow aAS$

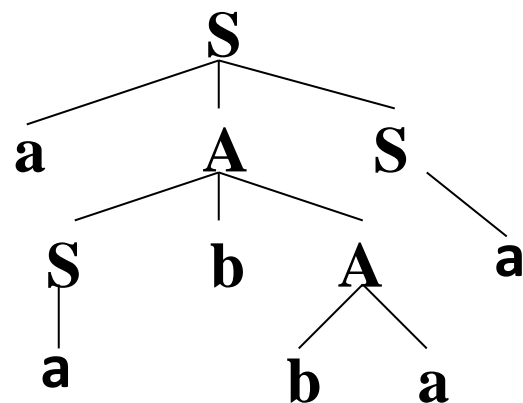
$A \rightarrow SbA$

$A \rightarrow SS$

$S \rightarrow a$

$A \rightarrow ba$

句子aabbbaa的语法树（推导树）



叶子结点：树中没有子孙的结点。

从左到右读出推导树的叶子标记，所得的句型为推导树的结果。也把该推导树称为该句型的**语法树**。

上下文无关文法的语法树

- 推导过程中施用产生式的顺序

例: $G[S]$:

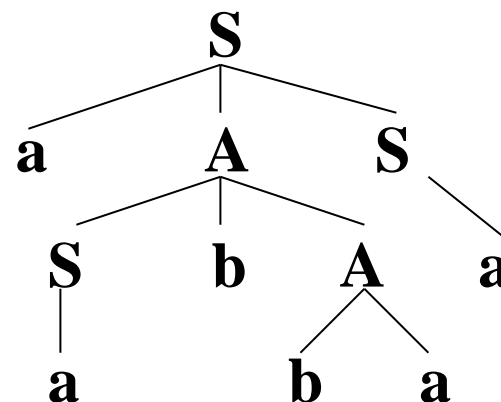
$S \rightarrow aAS$

$A \rightarrow SbA$

$A \rightarrow SS$

$S \rightarrow a$

$A \rightarrow ba$



$S \Rightarrow aAS \Rightarrow aAa \Rightarrow aSbAa \Rightarrow aSbbaa \Rightarrow aabbaa$

$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aabAS \Rightarrow aabbaS \Rightarrow aabbaa$

$S \Rightarrow aAS \Rightarrow aSbAS \Rightarrow aSbAa \Rightarrow aabAa \Rightarrow aabbaa$

文法的二义性

- **最左（最右）推导**：在推导的任何一步 $\alpha \Rightarrow \beta$ ，其中 α 、 β 是句型，都是对 α 中的最左（右）非终结符进行替换
- 最右推导被称为**规范推导**。
- 由规范推导所得的句型称为**规范句型**

文法的二义性

- 若一个文法存在某个句子对应两棵不同的语法树(或一个文法存在某个句子有两个不同的最左（右）推导), 则称这个文法是二义的。
- 部分二义文法可以改造为无二义文法

$G[E]: E \rightarrow i$

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$

$G[E]: E \rightarrow T | E + T$

$T \rightarrow F | T * F$

$F \rightarrow (E) | i$

规定优先顺序 (T) 和结合律 (左递归)

文法的二义性

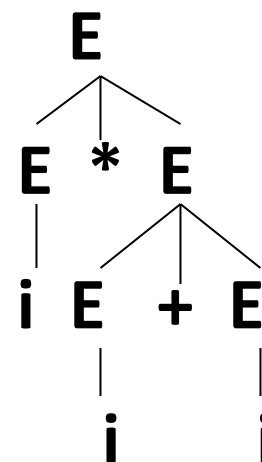
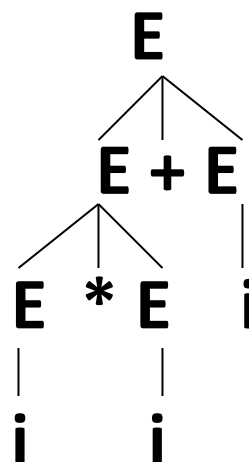
例: $G[E]$:

$E \rightarrow i$

$E \rightarrow E + E$

$E \rightarrow E * E$

$E \rightarrow (E)$



句型 $i*i+i$ 的两个不同的最左推导:

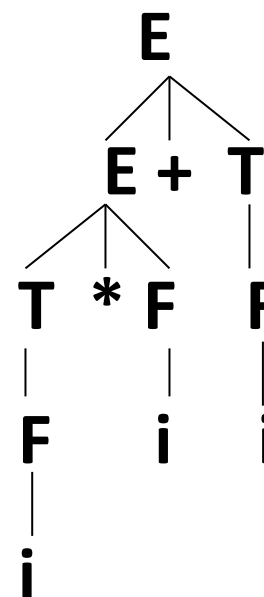
推导1: $E \Rightarrow E + E \Rightarrow E * E + E \Rightarrow i * E + E \Rightarrow i * i + E \Rightarrow i * i + i$

推导2: $E \Rightarrow E * E \Rightarrow i * E \Rightarrow i * E + E \Rightarrow i * i + E \Rightarrow i * i + i$

文法的二义性

- $G[E]: E \rightarrow T | E + T$
 $T \rightarrow F | T * F$
 $F \rightarrow (E) | i$

句型 $i*i+i$ 的最左推导:



Summary

- 程序语言的定义
 - 语法定义
 - 语义定义
- 高级语言的一般特性
 - 程序结构
 - 数据类型和操作
 - 语句与控制结构
- 程序语言的语法描述
 - 基本概念
 - 文法和文法的类型
 - 上下文无关文法及其语法树

阅读材料：《程序设计语言编译原理（第3版）》，
陈火旺等编著，国防工业出版社，2004年----第二章