中国人民大学
RENMIN UNIVERSITY OF CHINA

编译原理

--语言与文法基础

刘爽

中国人民大学信息学院

Outline

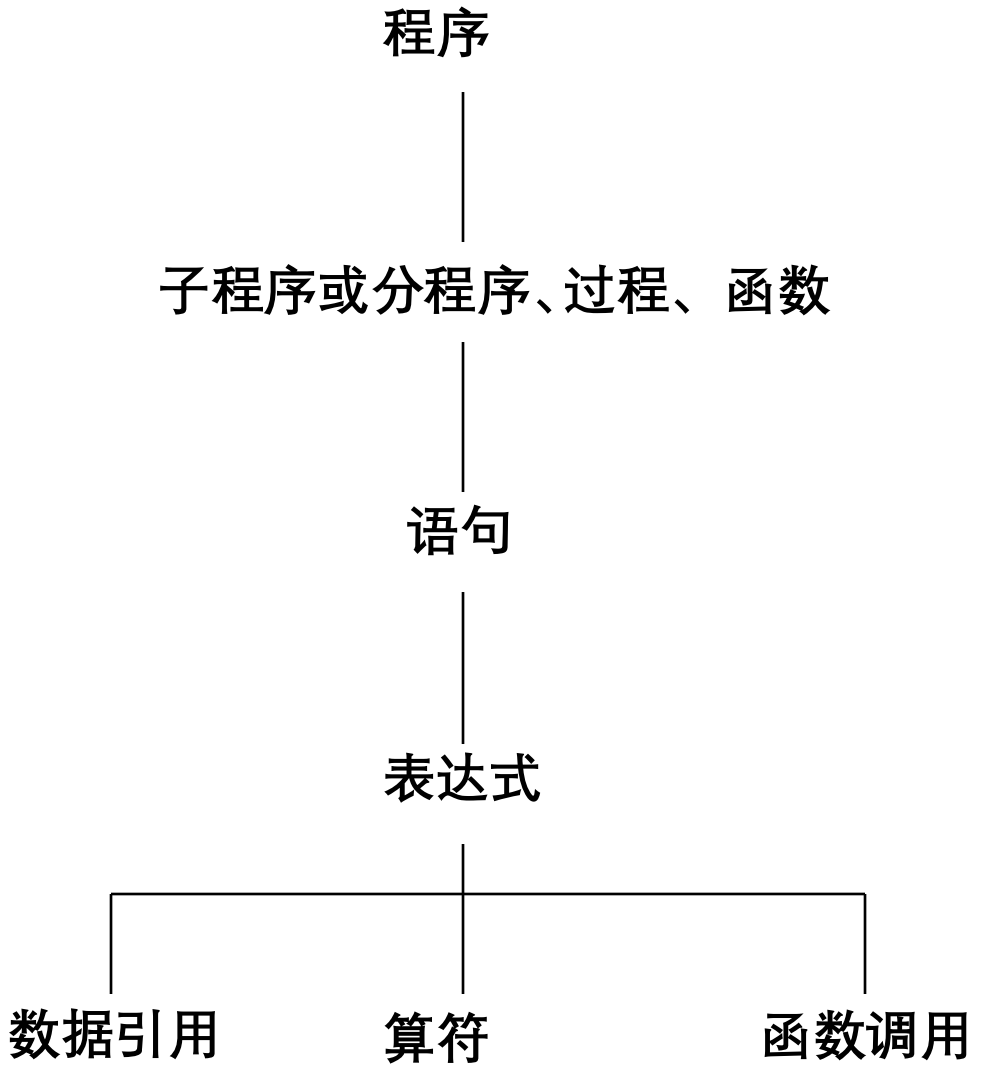
- 程序语言的定义
  - 语法定义
  - 语义定义
- 高级语言的一般特性
  - 程序结构
  - 数据类型和操作
  - 语句与控制结构
- 程序语言的语法描述
  - 符号和符号串
  - 文法和文法的类型
  - 上下文无关文法及其语法树

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



思考

- 下列哪些属于词法规则、语法规则，哪些属于语义规则?

  A、标识符是由字母开头的字母和数字组成的字符串

  B、循环体内有多条语句时，需要用大括号阔起

  C、C语言中符号 &在变量名前表示取地址操作

  D、变量名不可以与关键字重复

  E、if后的括号中必须有逻辑表达式

  F、int类型的变量占4个byte的空间

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
  - 符号和符号串
  - 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↑, XY+
- 形成规则
  - (1) **变量，常数**是表达式
  - (2) 若$\mathbf{E_1}$，$\mathbf{E_2}$为表达式，Θ是一个二元算符，则$\mathbf{E_1}$ Θ $\mathbf{E_2}$是表达式
  - (3) 若E是表达式，Θ为一元算符，则Θ **E** (或**E** Θ)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 （^或者 ** 或者 ↑）
  - 一元负 （-）
  - 乘、除 （*， /）
  - 加、减 （+, -）
  - 关系符 (<, =, >, <=, <>, >=)
  - 非(not, ¬)
  - 与 (∧, and, &)
  - 或 (∨, or, |)
  - 等值 (≡, ~)

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

符号和符号串

- **字母表**
  - 字母表是符号的**非空有限集**，通常用∑表示，例如∑={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，…空符号串用ε表示。
  - 形式定义：给定字母表∑，

    （1）ε 是∑上的符号串；

    （2）若x是∑上的符号串，且a ∈∑, 则ax或者xa是∑上的符号串；

    （3）y是∑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, ε}
  - $\sum^*$表示∑上的所有字的全体

符号串的运算

- 符号串**相等**
  - 若x, y是集合∑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∑上的符号串，且x=XY, y=YZ, 则x和y的连接积xy=XYYZ也是∑上的符号串。
    注意： ε x=xε

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|x∈A, y∈B}
  - 例如：A={s, t}, B={u, v}, AB=?

    {su, sv, tu, tv}
  - 注意：{ε}A=A{ε}, 如何证明?

    Hint: εx=xε
- 符号串集合的**幂运算**
  - 有符号集合A，定义:
    $A^0=\{\varepsilon\}$，　$A^1=A$，　$A^2=AA$，$A^3=AAA$, … $A^n=A^{n-1}A=AA^{n-1}$, n>0
  - 注意：A^0 不是空集，　ε，{}，{ε} 的差别

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

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

    $A^n=\underbrace{AA\ldots A}_{n}$

    $A^+=A^1 \cup A^2 \cup A^3 \ldots \cup A^n \ldots$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, ...}

    A^*={ε , x, y, xx, xy, yx, yy, xxx, xxy, xyx, xyy, yxx, yxy, yyx, yyy, ...}

符号和符号串

- 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, ..., z, 0, 1, ..., 9, +, -, *, /, _, (,), =, ...}

  B为**单词集**

  B={if, else, while, for, switch, case, ...}

  则**B⊂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 ⊂ B***, ***程序 ⊂ C***

思考

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

- $\mathbf{A^*}$和$\mathbf{B^+}$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
  - 符号和符号串
  - 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。
  - “我是人大”，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::=α，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**）。
    - 对于具有相同左部的那些规则，如U ::= x, U ::= y, U ::= z,可以将其缩写为：U ::= x|y|…|z
    - 符号 | 所表示的意思为“或”。因此上式表示的意思为：U定义为x或y或 … 或z
    - 符号“ ::=”、|、< 和 > 称为元符号。由元符号构成的语言称为元语言，所谓元语言即是用以描述其他语言的语言
    - 由元符号组成的巴科斯范式 < 元语言公式 > 是用以描述算法语言的元语言

文法的形式定义举例

- 例题：文法G[＜无符号整数＞]由如下13条规则组成

(1) ＜无符号整数＞::=＜数字串＞

(2) ＜数字串＞::=＜数字串＞＜数字＞

(3) ＜数字串＞::=＜数字＞

(4) ＜数字＞::= 0

(5) ＜数字＞::= 1

(6) ＜数字＞::= 2

(7) ＜数字＞::= 3

(8) ＜数字＞::= 4

(9) ＜数字＞::= 5

(10) ＜数字＞::= 6

(11) ＜数字＞::= 7

(12) ＜数字＞::= 8

(13) ＜数字＞::= 9

＜无符号整数＞是文法*G*[＜无符号整数＞]的识别符号。该文法的字汇表*V*为：*V* = {0, 1, ..., 9, ＜数字串＞, ＜数字＞, ＜无符号整数＞}

将该文法改写成BNF范式

＜无符号整数＞ ::= ＜数字串＞
＜数字串＞ ::= ＜数字串＞ ＜数字＞ | ＜数字＞
＜数字＞ ::= 0 | 1 |…| 9

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> =1=> <主语><谓语>

=2=> <代词><谓语>

=3=> 我<谓语>

=5=> 我<动词><直接宾语>

=6=> 我是<直接宾语>

=7=> 我是<名词>

=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>

文法推导练习

- 例题：文法G[＜无符号整数＞]由如下13条规则组成

(1) ＜无符号整数＞::=＜数字串＞

(2) ＜数字串＞::=＜数字串＞＜数字＞

(3) ＜数字串＞::=＜数字＞

(4) ＜数字＞::= 0

(5) ＜数字＞::= 1

(6) ＜数字＞::= 2

(7) ＜数字＞::= 3

(8) ＜数字＞::= 4

(9) ＜数字＞::= 5

(10) ＜数字＞::= 6

(11) ＜数字＞::= 7

(12) ＜数字＞::= 8

(13) ＜数字＞::= 9

如何从＜无符号整数＞推出数字1937?

＜无符号整数＞是文法*G*[＜无符号整数＞]的识别符号。该文法的字汇表*V*为：*V* = {0, 1, ..., 9, ＜数字串＞, ＜数字＞, ＜无符号整数＞}

练习

- 用上述文法练习推导句子“你学习法语”

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>

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$V_N \cap V_T = \Phi$。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→0S1，S→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→0S1, S→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α→β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=γαδ，w=γβδ，则说v（应用规则α→β）直接产生w，或说w是v的**直接推导**。（v=>w）

例：文法G=（V_N，V_T，P，S）

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

S ⇒0S1 ⇒00S11 ⇒000S111 ⇒00001111

S→0S1，S→01 可以缩写成 S→0S1 | 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：$v=w_0=>w_1=>w_2\cdots=>w_n=w$，$(n>0)$，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$v\overset{+}{=>}w$。
- 若有$v\overset{+}{=>}w$，或v=w，则记做$v\overset{*}{=>}w$。
- 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$S \overset{*}{=}>x$，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 | $S \overset{*}{=}>x$，其中S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→0S1， S→01

S ⇒0S1 ⇒00S11 ⇒000S111 ⇒00001111

$L(G) = \{0^n1^n \mid n \geqslant 1\}$

形式语言理论可以证明以下两点：
(1) G →L(G)；已知文法，可以推导出语言；
(2) L(G) →G1, G2, … , 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(G1) = L(G2)，则称文法G1和G2是等价的。
- 例1：如文法$G_1[A]$：A→0R　与　$G_2[S]$：S→0S1 等价

  A→01　　S→01

  R→A1

编译感兴趣的问题是：

- 给定句子x以及文法G，求x∈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→0S1, S→01，x∈L(G)？

    （2）x为 “0011”， G[S]: S→0S1, S→01，x∈L(G)？

    （3）x为 “001101”， G[S]: S→0S1, S→01，x∈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$|\beta| \geqslant |\alpha|$，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α→β，都有|β|≥|α|，仅仅 α→ε除外

- 1型（上下文有关）文法

  文法G[S]:

  S→CD
  C→aCA
  C→bCB
  AD→aD
  BD→bD
  Aa→aA
  Ab→bA
  Ba→aB
  Bb→bB
  C→ε
  D→ε

L(G)={ww|w∈{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$\alpha \to \beta$，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→aB|bA

  A→a|aS|bAA

  B→b|bS|aBB

  文法G2[S]:　　S→0A|1B|0

  A→0A|1B|0S

  B→1B|1|0

文法的类型举例

3型文法（正规文法）：任一产生式α→β的形式都为A→aB或A→a，其中A∈V_N，B∈V_N，a∈V_T。

- 定义标识符的3型（正规）文法

  文法G[S]：

  S → aT

  S → a

  T → aT

  T → dT

  T → a

  T → 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$\alpha_1$和$\alpha_2$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
  - 符号和符号串
  - 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 → α，其中A ∈ V_N， α ∈$(V_T U V_N)^*$， 则称G为上**下文无关文法**。
  - 算术表达式

> - 文法G=({E}, {+,*,I,(,)}, P, E}
>
> P: E → i
>
> E → E+E
>
> E → E*E
>
> E → (E)

  - 语句
    - 赋值语句
    - 条件语句 <条件语句>→if<条件>then<语句> | if<条件>then<语句>else <语句>
    - ……

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→aAS

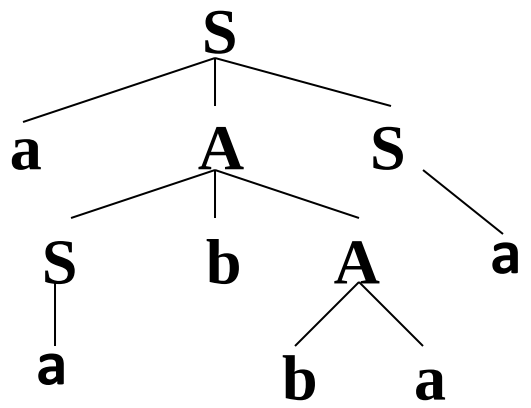
A→SbA

A→SS

S→a

A→ba

句子aabbaa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。

推导的定义及语法树的生成

- 语法树通常表示成一棵倒置的树，其根在上，枝叶在下
- 随着推导的展开，当树枝上标记的是非终结符，并用它的某个候选式进行替换时，这个非终结符标记的相应结点就产生下一代新结点，候选式中自左至右的每一个符号对应标记一个新结点，每个新结点和其父结点之间均有一连线
- 在一棵语法树生长过程中的任何一个时刻，所有那些没有后代的末端结点的标记自左至右的排列就是一个句型
- 如果自左至右末端结点的标记均为终结符，那么，这棵语法树代表了一个句子的各种推导过程。此时不存在新的推导，推导终止。

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]:

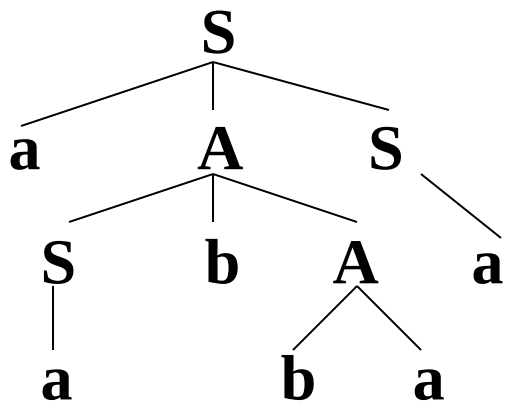
S→aAS

A→SbA

A→SS

S→a

A→ba



S⇒aAS⇒aAa⇒aSbAa⇒aSbbaa⇒aabbaa

S⇒aAS⇒aSbAS⇒aabAS⇒aabbaS⇒aabbaa

S⇒aAS⇒aSbAS⇒aSbAa⇒aabAa⇒aabbaa

文法的二义性

- 最左（最右）推导：在推导的任何一步α⇒β，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 → i
E → E+E
E → E*E
E → (E)

G[E]: E → T|E+T
T → F|T*F
F → (E) |i

规定优先顺序和结合律

文法的二义性

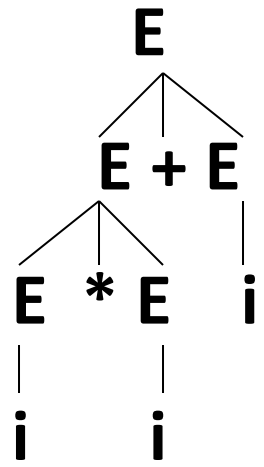
例：G[E]:

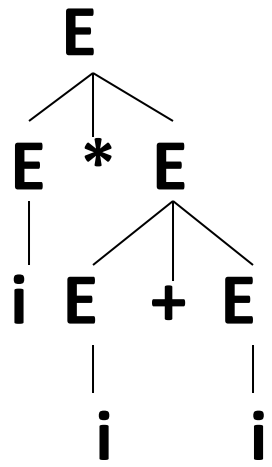
E → i

E → E+E

E → E*E

E → (E)





句型 i*i+i 的两个不同的最左推导：

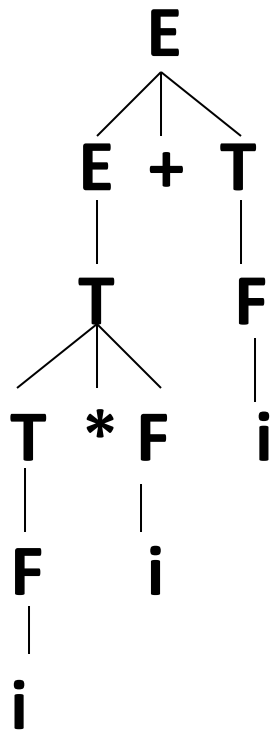
推导1：E ⇒ E+E ⇒ E*E+E ⇒ i*E+E ⇒ i*i+E ⇒ i*i+i

推导2：E ⇒ E*E ⇒ i*E ⇒ i*E+E ⇒ i*i+E ⇒i*i+i

文法的二义性

- G[E]: E → T|E+T
  T → F|T*F
  F → (E) |i

句型 i*i+i 的最左推导：



- 遗憾的是，已经证明：文法二义性的性质是不可判定的
- 这就意味着不存在（或不能给出）一种算法，它接收任意BNF文法，并能在有穷步骤内确切判定出该文法是否是二义性的。人们能做的就是找到某些简单条件，当文法满足这些条件时，就确信该文法是无二义性的。它们是无二义性的充分条件，但并不是必要条件

有关文法的实用限制

对文法的实用性限制有两条：

- 不能有U ::= U这样的规则
- 不能有多余规则。在文法中，有两种规则是多余规则：
  - 在推导文法的所有句子中始终都用不到的规则
  - 在推导句子的过程中，一旦使用了此规则，将无法再推出任何终结符号串来

有关文法的实用限制

- 例题：有文法G[<z>]:

  <z> ::= <b>e

  <a> ::= <a>e | e

  <b> ::= <c>e | <a>f

  <c> ::= <c>f

  <d> ::= f

- 在该文法中，由于非终结符号<d>不出现在任何规则的右部，而句子的推导总是从文法的识别符号<z>开始，所以在句子的推导中始终不可能用到规则<d> ::= f，因此是多余规则。另外，规则<c> ::= <c>f也是多余规则。因为一旦使用了这条规则以后，将使推导无限制地进行下去，如： <c>⇒<c>f⇒<c>ff⇒<c>fff…

  就再也无法推出任何终结符号串来了。由于同样的原因，规则<b> ::= <c>e也是多余的规则，因为在该规则中包含有非终结符号<c>

有关文法的实用限制

- 根据上述分析，可以断言：如果程序设计语言的文法包含有多余规则，其中必定有错误存在
- 要检查文法中每一条规则左部的每个非终结符号U是否满足下述两个条件：
  - 对任何$U \in V_N, U \neq \mathbf{S}$（识别符号），则U必须出现在某个句型中，即有：

$$\mathbf{S} \overset{*}{\Rightarrow} xUy，\ 其中，\ x, y \in V_T^*$$

  - 对任何$U \in V_N$，必须能够从U推导出终结符号串t来，即：

$$U \overset{+}{\Rightarrow} t，\ 其中，\ t \in V_T^*$$

- 显而易见，如果非终结符U不满足上述两个条件，称U为无用符号，而且，包含有U的规则即是多余规则，要从文法中除去

有关文法的实用限制

- 上述文法在去掉多余规则以后可压缩为：

  < z > ::= < b > e

  < a > ::= < a > e | e

  < b > ::= < a > f

- 压缩文法：如果文法G中的每一个非终结符号U都满足上述条件一和条件二，则称该文法为压缩过的或化简过的

扩充的BNF表示

- 扩充的BNF表示
  - 扩充的BNF是在BNF基础上发展起来的，它与BNF表示具有相同的表达能力，但在结构上更为简单和清晰
  - 在BNF中所使用的元语言符号只有＜、＞、::=、|共4种，在消除公共左因子时，引入了圆括号，现在引入方括号和花括号，使其定义语言的表达能力更强，通常称为花括号法或扩充的BNF
  - {a}表示a的0次到任意多次重复，即（a*）
  - [a]表示a可有可无（即a | ε）
  - 利用扩充BNF，标识符的定义可写为

I ::= L{L|D}

L ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

D ::= 0|1|2|3|4|5|6|7|8|9

- 用这种定义系统不仅能增强表达能力，而且直观易懂，有许多语言采用这一系统进行定义

扩充的BNF表示

- 这种定义系统也便于消除左递归和提取公共左因子
  - 例如

    E ::= T | E + T
  - 改写成:

    E ::= T{+ T}
  - 利用这个定义系统，文法G1可改写成:

    E ::= T{+ T}

    T ::= F{ * F}

    F ::= i | ' ('E') '

文法G1:

E ::= T | E + T

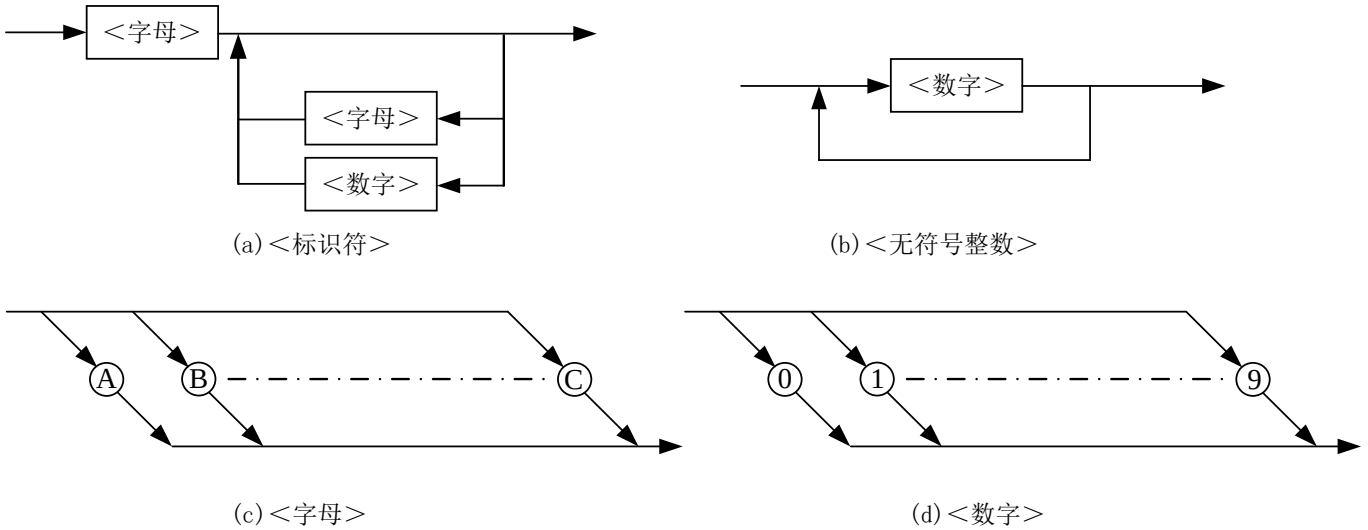
T ::= F | T * F

F ::=i | ' ('E') '

被定义的圆括号与元语言符号冲突，特别用单引号括起来

语法图

- 语法图
  - 任何以产生式给定的文法，均可以构造一个等效的语法图，用来识别该文法所产生的合法句子，即用所谓识别图（Recognition Graph），或称语法图（Syntax Graph）来定义给定的语言
  - 例如表示 < 标识符 > 、< 无符号整数 > 以及 < 字母 > 和 < 数字 > 的语法



语法图

- 在语法图中，终结符用圆框标记，非终结符用方框标记
- 任何一个非终结符均可由仅有的一个入口边和以一个出口边的语法图来定义
- 若一个终结符序列是合法的，那么必须从语法图的入口边通过语法图而达到出口边，且在通过的过程中，恰恰能识别该终结符序列。在通过标识终结符的圆框时，标记的终结符与被识别的终结符正好符合，则该终结符被识别；若通过标记为非终结符的方框，那么由通过该非终结符的语法图来识别；若遇到分支，可以经由任一边来识别，若经由这个边识别不成功，则返回另一边来识别，这种情况称为回溯，直到所有的边都识别不成功，则该终结符序列是不合法的，不属于该语法图定义的语言
- 该语法图能识别的所有终结符序列的集合即为该语法图定义的语言

语法图

- 早期的FORTRAN语言的语法定义是采用自然语言描述的
- ALGOL 60首次采用BNF对程序设计语言的语法进行形式描述，为语言定义做出了重要贡献
- Pascal首次采用语法图来定义语言，给出了较为直观的语法结构
- BNF和语法图是语言文法的等价表示，语法图从识别的观点来定义语言，它更直观、更简洁、更清晰地给出了语言的语法结构图像

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
《编译原理与技术》张莉等编著， 第二章