复　习

什么是编译程序

编译过程（六个阶段）

编译程序的结构（六个模块、表格、出错处理）

编译阶段的组合（前端、后端）

解释程序（编译和解释的区别）

一些软件工具

程序设计语言范型

让系统知道
源程序语言的“组成规则”
并按其分析

语　言

语言是一定的群体用来进行信息交流的工具。

什么是语言？

语　言

- 信息交流的基础是什么？
  - 按照共同约定的生成规则和理解规则去生成“句子”和理解“句子”
  - 例：
    - “今节日上课始开译第一编”
    - “今日开始上第一节编译课”

语　言

- 语言的特征
  - 自然语言(Natural Language)
    - 是人与人的通讯工具
    - 语义(semantics):环境、背景、语气、二义性——难以形式化
  - 计算机语言(Computer Language)
    - 计算机系统间、人机间通讯工具
    - 严格的语法(Grammar)、语义(semantics) ——易于形式化：严格

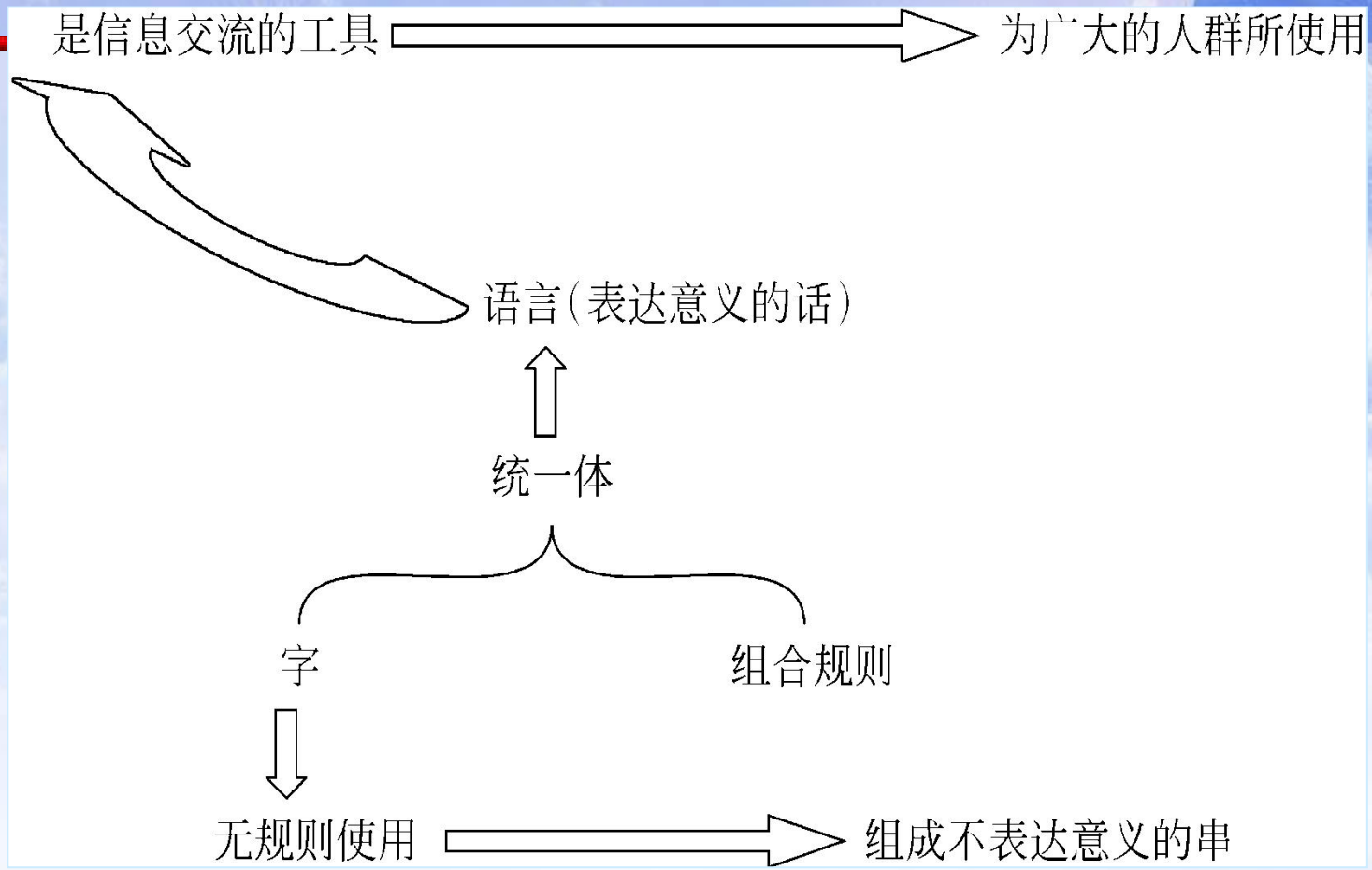
语　言

- 语言的描述方法——现状
  - 自然语言：自然、方便-非形式化
  - 数学语言（符号）：严格、准确-形式化
  - 形式化描述
    - 高度的抽象、严格的理论基础和方便的计算机表示

语　言

- 语言——形式化的内容提取
  - 语言(Language)：满足一定条件的句子集合
  - 句子(Sentence)：满足一定规则的单词序列
  - 单词(Token)：满足一定规则的字符(Character)串
- 语言是字和组合字的规则
  - 例（自然语言：第译始二天课今开编上节）
  - 今天开始上第二节编译课

语　言



语言是字及其组合规则的统一体

程序设计语言

- 程序设计语言——形式化的内容提取
  - 程序设计语言(Programming Language)：组成程序的所有语句的集合。
  - 程序(Program)：满足语法规则的语句序列。
  - 语句(Sentence)：满足语法规则的单词序列。
  - 单词(Token)：满足词法规则的字符串。
- 例：变量:=表达式
  - if 条件表达式 then 语句
  - while 条件表达式 do 语句
  - call 过程名(参数表)

程序设计语言

- 描述形式——文法
  - 语法——语句
    - 语句的组成规则
    - 描述方法：BNF范式、语法(描述)图
  - 词法——单词
    - 单词的组成规则
    - 描述方法：BNF范式、正规式

第二章 文法和语言

主要内容

- 文法的直观概念
- 符号和符号串
- 文法和语言的形式定义
- 文法的类型
- 上下文无关文法及其语法树
- 句型的分析
- 有关文法实用中的一些说明

2.1 文法的直观概念

- 当我们表述一种语言时，就是要说明这种语言的句子。
  - 如果语言只含有有穷多个句子，则只需列出句子的有穷集。
  - 如果语言含有无穷多个句子，存在着如何给出它的有穷表示的问题。这需要一种规则，用这些规则来描述语言的结构，可以把这些规则看成一种元语言，这些规则(或语言)就称为文法。

汉语文法举例

- <句子>::=<主语><谓语>
- <主语>::=<代词>|<名词>
- <代词>::=我|你|他
- <名词>::=王明|大学生|工人|英语
- <谓语>::=<动词><直接宾语>
- <动词>::=是|学习
- <直接宾语>::=<代词>|<名词>

“我是大学生”的动作过程

<句子>⇒<主语><谓语>

⇒<代词><谓语>

⇒我<谓语>

⇒我<动词><直接宾语>

⇒我是<直接宾语>

⇒我是<名词>

⇒我是大学生

<句子>::=<主语><谓语>
<主语>::=<代词>|<名词>
<代词>::=我|你|他
<名词>::=王明|大学生|工人|英语
<谓语>::=<动词><直接宾语>
<动词>::=是|学习
<直接宾语>::=<代词>|<名词>

2.2 符号和符号串

- 字母表(字符集)：元素的非空集合，也称为符号集
  - 例：

    $\sum=\{0，1\}$

    $A=\{a, b, c\}$

    $B=\{a，b，c，……，z\}$

    ASCII字母表

∑={0，1}
A={a, b, c}

2.2 符号和符号串

- 符号：字母表中的元素
  - 例：0，1都是∑的符号，a、b、c是A的符号
- 符号串：由字母表中的符号组成的任何有穷序列
  - 例：01、1001是∑的符号串
  - a、b、c、abc、ab是A的符号串

长度：如果某符号串x有m个符号，称其长度为m，表示为|x|=m。如001110的长度为6

空符号串：不包含任何符号的符号串，用| ε |=0表示

符号和符号串

- 符号串的头尾（前缀、后缀）：

  如果z=xy是一符号串，

  那么x是z的头，y是z的尾。

  如果x是非空的，那么y是固有尾；

  若y非空，x是固有头。

  – 如：符号串abc

    - 头： ε , a, ab, abc
    - 尾：abc，bc, c, ε
    - 固有头： ε , a, ab
    - 固有尾： bc, c, ε

符号和符号串

- 符号串的连接：设x和y是符号串，它们的连接xy是把y的符号写在x的符号之后得到的符号串
  - 如：x=ST，y=abu，xy=STabu
- 符号串的方幂：设x是符号串，把x自身连接n次得到符号串z，即z=xx…xx，成为符号串x的方幂，写作$z=x^n$
  - 如：x=AB，$x^0=\varepsilon$，x^1=AB，x^2=ABAB
  - 对于n>0，$x^n=xx^{n-1}=x^{n-1}x$

∑={0，1}

符号和符号串

- 符号串集合：若符号串集合A中的一切元素都是某字母表上的符号串，则称A为该字母表上的符号串集合。
  - 如{0，1，01，001，0001}是∑上的符号串的集合

区别于笛卡尔积

符号串集合的乘积：$AB=\{xy|x\in A且y\in B\}$。即AB是满足$x\in A$，$y\in B$的所有符号串xy所组成的集合

- 如：A={a, b}，B={c, d}，AB={ac, ad, bc, bd}
- 对于任意符号串，有 ε x=x ε =x

符号和符号串

- 集合的闭包（克林闭包）：指定字母表Σ之后，用Σ*表示Σ上的所有有穷长的串的集合。Σ*称为集合Σ的闭包。$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots \cup \Sigma^n \ldots$
  - 如Σ={0，1}，Σ*={ε, 0, 1, 00, 01, 10, 11, 000, 001, 010, …}
- 集合的正闭包：$\Sigma^+ = \Sigma^1 \cup \Sigma^2 \cup \ldots \cup \Sigma^n \ldots$称为Σ的正闭包。
- Σ*具有无穷数量的元素，如果x是Σ*中的元素，则表示为$x \in \Sigma^*$，否则$x \in \Sigma^* \backslash$。对于所有的Σ，有$\varepsilon \in \Sigma^*$。

ε、{ε}、{}

设∑是一个字母表，

$\forall L \subseteq \sum^*$，$L$称为字母表∑上的一个语言？

$\forall x \in L$，x叫做L的一个句子？

- 例：字母表是所有汉字
  - 语言？（第译始二天课今开编上节）
  - 今天开始上第二节编译课

怎样有效刻画语言中的句子？

如何实现语言结构的形式化描述？

2.3 文法和语言的形式定义

- 规则（重写规则，产生式或生成式）的定义：
  - 形如 α → β 或 α ::= β 的(α， β)有序对，
  - α 称为规则的左部， β 称作规则的右部
  - 箭头符号读作：“定义为”。

如何实现语言结构的形式化描述？

考虑赋值语句的形式：

左部量 = 右部表达式

a = a+a

b = m[3]+b

m[1] = a+m[2]

句子的组成规则

- <赋值语句>→<左部量>=<右部表达式>
- <左部量>→<简单变量>
- <左部量>→ <下标变量>
- <简单变量>→a
- <简单变量>→b
- <简单变量>→ c
- <下标变量>→m[1]
- <下标变量>→m[2]
- <下标变量>→m[3]
- <右部表达式>→<简单变量><运算符><简单变量>
- <右部表达式>→<简单变量><运算符><下标变量>
- <右部表达式>→<下标变量><运算符><简单变量>
- <右部表达式>→<下标变量><运算符><下标变量>
- <运算符>→ +
- <运算符>→ -

问题：如何用符号来描述？即如何形式化？

定义句子的规则的语法组成

——终结符号集，非终结符号集，语法规则，开始符号

非终结符号集V＝

{<赋值语句>，<左部量>，<右部表达式>，

<简单变量>，<下标变量>，<运算符>}

终结符号集T＝

{a，b, c,m[1],m[2],m[3], +, -}

语法规则集P＝

{<赋值语句>→<左部量>=<右部表达式>，……}

开始符号S＝ <赋值语句>

文法G的形式定义

定义： 文法G为一个四元组:

$$G=(V，T，P，S)$$

- V（Vn）：非终结符(Terminal)集
  - 语法变量（成分）——代表某个语言的各种子结构
- T（V_T）：终结符(Variable)集
  - 语言的句子中出现的字符， $Vn \cap V_T = \Phi$
- S：开始符号(Start Symbol)，$S \in Vn$
  - 代表文法所定义的语言，至少在产生式左侧出现一次

文法G的形式定义

- P：产生式(Product)集合

α → β，被称为产生式（定义式），读作：α 定义为 β。

其中 α ∈(V ∪T)$^+$，且 α 中至少有 V 中元素的一个出现。β ∈(V ∪T)*。

α 称为产生式 α → β 的左部(Left Part)，

β 称为产生式 α → β 的右部(Right Part)。

产生式定义各个语法成分的结构（组成规则）

文法定义举例1

例1：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

文法定义举例2

例2：文法G=（V_N，V_T，P，S）

V_N ={标识符，字母，数字}

V_T ={a,b,c,…x,y,z,0,1,…,9}

P={<标识符>➞<字母>

<标识符>➞<标识符><字母>

<标识符>➞<标识符><数字>

<字母>➞a,…, z

<数字>➞0, …, 9 }

S=<标识符>

文法的写法

- 一般约定：
  - 第一条产生式的左部是识别符号
  - 用尖括号括起来的是非终结符，不用尖括号括起来的是终结符
  - 或者用大写字母表示非终结符，小写字母表示终结符。
- 常用符号：→　　::=　　|　　<　>
- 各种写法：
  - G：S→0S1　S→01
  - G[S]：S→0S1　S→01
  - G[S]: S→0S1 | 01

赋值语句的文法

- V_N={<赋值语句>，<左部量>，<右部表达式>，<简单变量>，<下标变量>，<运算符>}
- V_T ={a，b，c，m[1]，m[2]，m[3]，+，-}
- *P*={<赋值语句>→<左部量>=<右部表达式>，<左部量>→<简单变量>，<左部量>→ <下标变量>，<简单变量>→a，<简单变量>→b，<简单变量>→c，<下标变量>→m[1]，<下标变量>→m[2]，<下标变量>→m[3]，<右部表达式>→<简单变量><运算符><简单变量>，<右部表达式>→<简单变量><运算符><下标变量>，<右部表达式>→<下标变量><运算符><简单变量>，<右部表达式>→<下标变量><运算符><下标变量>，<运算符>→ +，<运算符>→ -}
- *S*=<赋值语句>

赋值语句的文法（续）

符号化之后：

G=({A，B，E，C，D，O}，{a，b，c，m[1]，m[2]，m[3]，+，-}，P，A)，

其中，P={$A\rightarrow B=E$，$B\rightarrow C$，$B\rightarrow D$，$C\rightarrow$a，$C\rightarrow$b，$C\rightarrow$ c，$D\rightarrow$m[1]，$D\rightarrow$m[2]，$D\rightarrow$m[3]，$E\rightarrow COC$，$E\rightarrow COD$，$E\rightarrow DOC$，$E\rightarrow DOD$，$O\rightarrow$ +，$O\rightarrow$ -}

产生式的简写

- 对一组有相同左部的产生式

$\alpha \rightarrow \beta_1$，$\alpha \rightarrow \beta_2 \ldots$，$\alpha \rightarrow \beta_n$

可以简单地记为：

$\alpha \rightarrow \beta_1 | \beta_2 | \ldots | \beta_n$

读作：α 定义为或者 β_1，或者 β_2，…，或者 β_n。并且称它们为 α 产生式。β_1，β_2，…，β_n称为候选式。

- 对一个文法，只列出该文法的所有产生式。

问题：有了定义句子的规则，如何判定某一句子是否属于某语言？

句子的派生(推导)-从产生语言的角度

<赋值语句>

⇒ <左部量> = <右部表达式>

⇒ <简单变量> = <右部表达式>

⇒ a = <右部表达式>

⇒ a = <简单变量><运算符><简单变量>

⇒ a = a <运算符><简单变量>

⇒ a = a + <简单变量>

⇒ a = a + a

句子的归约

-从识别语言的角度

a = a + a

⇐ a = a + <简单变量>

⇐ a = a <运算符><简单变量>

⇐ a = <简单变量><运算符><简单变量>

⇐ a = <右部表达式>

⇐ <简单变量> = <右部表达式>

⇐ <左部量> = <右部表达式>

⇐ <赋值语句>

- 派生与推导均根据规则

直接推导

- 定义：如α→β是文法G[V_N，V_T, P, S]的产生式，γ和δ是V^*中的任意符号，若有符号串v, w满足：v=γαδ,w=γβδ,则称v(应用规则α→β)直接产生w,或说，w是v的直接推导，或说，也称w直接归约到v，记作v⇒w

直接推导举例

- 例1：G[S]：S→0S1， S→01

  0S1⇒00S11　应用规则S→0S1

  00S11⇒000S111　应用规则S→0S1

  000S111⇒00001111　应用规则S→01

  S⇒0S1　应用规则S→0S1

- 例2：

  <标识符>⇒<标识符><字母>

  <标识符><字母><数字>⇒<字母><字母><数字>

  abc<数字>⇒abc5

推导

- 若存在直接推导的序列：

  $v = w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n = w, (n>0)$

  则记为$v \overset{\pm}{\Rightarrow} w$，v推导出w，或w归约到v，推导长度为n

- 若有$v \overset{\pm}{\Rightarrow} w$，或v=w，则记为$v \overset{*}{\Rightarrow} w$

记为 $\alpha_0 \overset{n}{\Rightarrow} \alpha_n$ （恰用n步）

$\alpha_0 \overset{+}{\Rightarrow} \alpha_n$ （至少一步）

$\alpha_0 \overset{*}{\Rightarrow} \alpha_n$ （若干步：零步或多步）

推导举例

例：G[S]： S→0S1， S→01

$0S1 \Rightarrow 00S11$

$00S11 \Rightarrow 000S111$

$000S111 \Rightarrow 00001111$

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

$S \overset{+}{\Rightarrow} 00001111$

$S \overset{*}{\Rightarrow} S$　　$00S11 \overset{*}{\Rightarrow} 00S11$

- 推导的实质：

  从开始符号依据产生式对所得串的特定部位进行变换，不断获得新的串，最终得到目标串。

句型和句子

- 定义：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$S \overset{*}{\Rightarrow} x$，则称x是文法G的句型。若x仅由终结符号组成，即$S \overset{*}{\Rightarrow} x$，$x \in V_T^*$，则称x是文法G的句子。
- 例：G[S]： $S \rightarrow 0S1$， $S \rightarrow 01$

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

  G的句型S,0S1 ,00S11 ,000S111,00001111

  G的句子00001111, 01

语言

- 定义：文法G所产生的语言定义为集合$\{x|S\overset{*}{\Rightarrow}x$，其中S为文法识别符号，且$x\in V_T^*\}$。用L(G)表示。
- 从定义可以看出两点：
  - X是句子
  - X仅由终结符号组成
- 例： G[S]： $S\rightarrow 0S1$， $S\rightarrow 01$

  $L(G)=\{0^n1^n|n\geq 1\}$

句子组成语言。
句子是根据文法推导的结果。

语言举例

- 例 文法G[S]：

（1）S→aSBE

（2）S→aBE

（3）EB→BE

（4）aB→ab

（5）bB→bb

（6）bE→be

（7）eE→ee

$L(G)=\{ a^n b^n e^n \mid n\geq 1 \}$

文法的等价

- 若L（G_1）=L（G_2），则称文法G_1和G_2是等价的。

- 如文法G_1[A]：A→0R 与G_2[S]：S→0S1 等价

  A→01　　S→01

  R→A1

总 结

- 语言：字和组合字的规则——结构型描述
- 文法的直观概念

  以有穷的集合刻画无穷的集合的工具
- 符号和符号串

  字母表、符号串、头尾、连接、方幂、闭包
- 文法和语言的形式定义

  四元组（非终结符、终结符、产生式、开始符）
- 推导（归约）

  从开始符号依据产生式对所得串的特定部位进行变换，不断获得新的串，最终得到目标串。
- 句型、句子、语言

作　业

- P33:1、2、3、4、5 题思考

复 习

- 语言：字和组合字的规则——结构型描述
- 文法的直观概念

  以有穷的集合刻画无穷的集合的工具
- 符号和符号串

  字母表、符号串、头尾、连接、方幂、闭包
- 文法和语言的形式定义

  四元组（非终结符、终结符、产生式、开始符）
- 推导（归约）

  从开始符号依据产生式对所得串的特定部位进行变换，不断获得新的串，最终得到目标串。
- 句型、句子、语言

2.4 文法的类型

- 根据语言结构的复杂程度（形式语言）
  - 涉及文法的复杂程度、分析方法的选择
  - 反映文法描述语言的能力
  - 描述不同、分析方法不同。

2.4 文法的类型

- 通过对产生式施加不同的限制，乔姆斯基(Chomsky)把文法分成四种类型：
  - 0型文法
  - 1型文法
  - 2型文法
  - 3型文法

0型文法（短语文法）

- 设文法G=（V_N，V_T，P，S），若P中任一产生式 $\alpha \rightarrow \beta$，都有 $\alpha \in (V_N \cup V_T)^*$ 且至少含有一个非终结符，而 $\beta \in (V_N \cup V_T)^*$，则G是一个0型文法。
- 0型文法也称短语文法。0型文法的能力相当于图灵机。
- 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（上下文有关文法）

- 设文法G=（V_N，V_T，P，S），若P中任一产生式 α → β，都有| β |≥| α |，仅仅S → ε 除外，则G是一个1型文法或上下文有关文法。

- 例：文法G[S]：S→CD　　C→aCA

  CA→Ca　　CaD→daD

  dAc→dec

2型文法（上下文无关文法）

- 设文法G=（V_N，V_T，P，S），若P中任一产生式α→β，满足：α是一个非终结符，β$\in(V_N \cup V_T)^*$，则G是一个2型文法或上下文无关文法。
- 能描述程序设计语言的多数语法成分

- 例：文法G[S]：

  S→AB　A→BS|0　B→SA|1

3型文法（正规文法）

- 设文法G=（V_N，V_T，P，S），若P中每一个产生式都是A→aB或A→a，其中A和B都是非终结符，a∈V_T∪{ε}，则G是一个3型文法或正规文法。
- 例1：G[S]：S→0A|1B|0

  A→0A|1B|0S

  B→1B|1|0
- 例2：G[L]：L → 1T　　L→ 1　　T → 1T

  T → dT　　T → 1　　T → d

乔姆斯基体系——总结

$G=(V，T，P，S)$是一个文法, $\alpha \rightarrow \beta \in P$

* G是0型文法，L(G)是0型语言；

  ---其能力相当于图灵机

* $|\alpha| \leq |\beta|$:G是1型文法，L(G)是1型语言(除$S \rightarrow \varepsilon$)；

  ---其识别系统是线性界限自动机

* $\alpha \in V_N$: G是2型文法，L(G)是2型语言；

  ---其识别系统是不确定的下推自动机

* A→aB或A→a: G是右线性文法，L(G)是3型语言

  A→Ba或A→ a : G是左线性文法，L(G)是3型语言

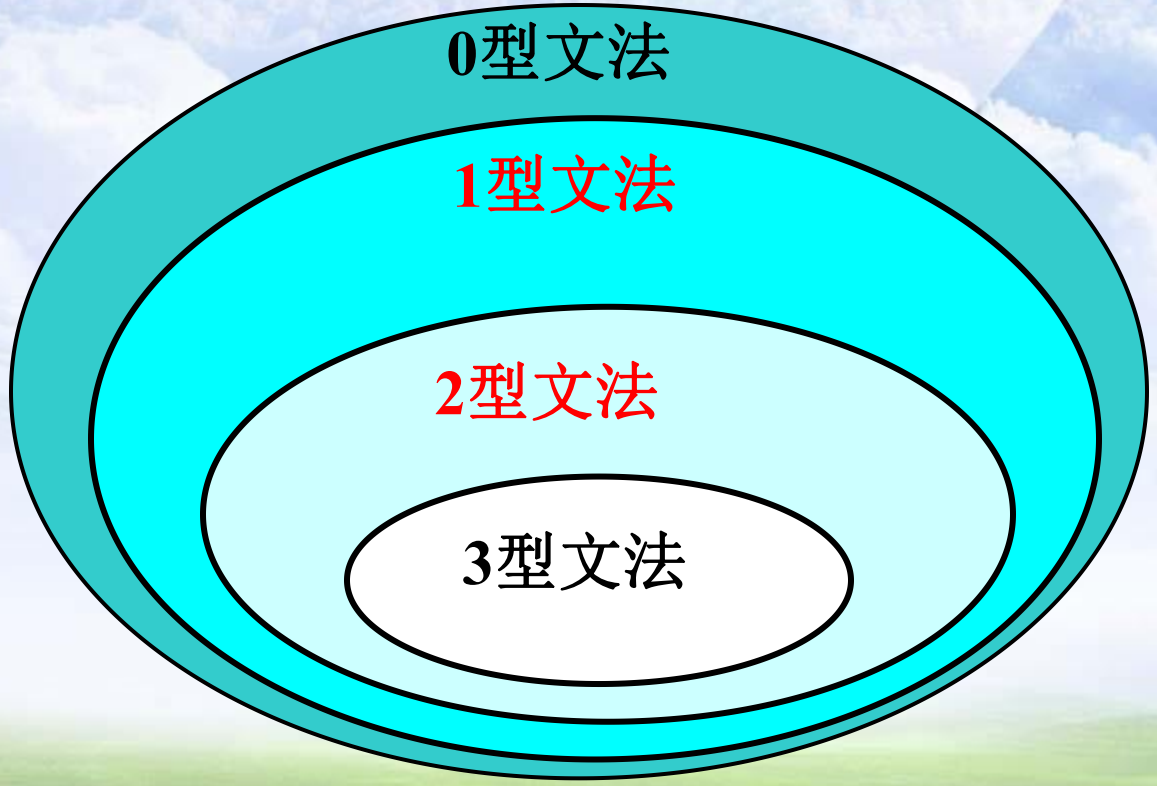
  ---其识别系统是有穷自动机

0型文法 (PSG)	1型文法 (CSG)	2型文法 (CFG)	3型文法 (RG)	3型文法 (RG)
S→aBC	S→aBC	E→E+E	S→a|b	S→a|b
S→aSBC	S→aSBC	E→E*E	S→aT|bT	S→Ha|Hb
CB→BC	CB→BC	E→(E)	T→a|b	S→H1|H2
aB→d	aB→ab	E→id	T→1|2	H→Ha|Hb
bB→bb	bB→bb	E→E-E	T→aT|bT	H→H1|H2
bC→b	bC→bc	E→E/E	T→1T|2T	H→a|b
cC→ cc	cC→ cc			

文法的类型

四种文法之间的关系是将产生式作进一步限制而定义的。

四种文法之间的逐级“包含”关系如下：



2.5 上下文无关文法及其语法树

- 上下文无关文法有足够的能力描述程序设计语言的语法结构
  - 文法G=({E} , {+, *, i, (,) } ,P,E)其中P为：

    {E→i ， E→E+E ， E→E*E， E→(E) }。

    E表示算术表达式, i表示程序的“变量”，

    该文法定义了由变量，+，*，（ 和 ）组成的算术表达式的语法结构，即：

    变量是算术表达式；若E_1和E_2是算术表达式，则E_1+E_2，E_1*E_2和(E_1)也是算术表达式

上下文无关文法描述语法结构

- 描述一种简单赋值语句的产生式：

  〈赋值语句〉→i：=E

- 描述条件语句的产生式：

  〈条件语句〉→if〈条件〉then〈语句〉 |

  if〈条件〉then〈语句〉else〈语句〉

2.5 上下文无关文法及其语法树

- 用树的形式表示句型的生成
  - 树根：　开始符号
  - 中间结点：　非终结符
  - 叶结点：　终结符或者非终结符

文法G=({E} , {+,*,i,(,)} ,P,E)其中P为：

{E→i ， E→E+E ， E→E*E， E→(E) }

句子i*(i+i)

语法树(推导树)

- 设G=(V_N,V_T,P,S)为一上下文无关文法，若对于G的任何句型都能构造与之关联的语法树(推导树)。这棵树满足下面四个条件：
  - 每个结点都有一个标记，此标记是V的一个符号
  - 根的标记是S
  - 若一结点n至少有一个它自己除外的子孙，并且有标记A，则肯定A∈V_N（中间结点是非终结符（变量））
  - 如果结点n有标记A，其直接子孙结点从左到右的次序是n_1，n_2，…，n_k，其标记分别为A_1，A_2，…，A_k，那么A→A_1A_2，…，A_k一定是P中的一个产生式（子树）
- 语法树的结果：从左至右读出推导树的叶子标记

语法树的构造举例

- 文法G=({S,A}, {a,b}, P, S)，其中P为：

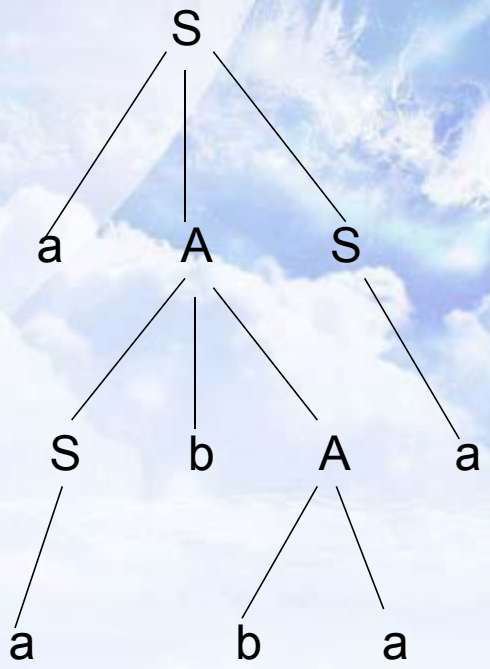
  S→aAS

  A→SbA

  A →SS

  S →a

  A →ba



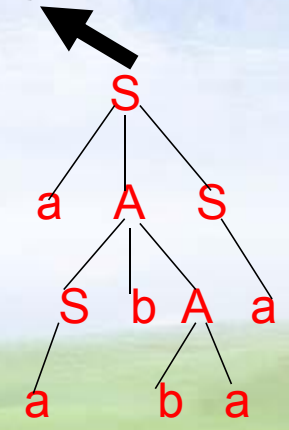
句子**aabbaa**

语法树的构造举例

- 语法树表示了在推导过程中施用了哪个产生式和在哪个非终结符上，它并没有标明使用产生式的顺序。
- 句型aabbaa的推导过程可以列举为：
  - S ⇒aAS ⇒aAa ⇒aSbAa ⇒aSbbaa ⇒aabbaa最右
  - S ⇒aAS ⇒aSbAS ⇒aabAS ⇒aabbaS ⇒aabbaa最左
  - S ⇒aAS ⇒aSbAS ⇒aSbAa ⇒aabAa ⇒aabbaa
  - S ⇒aAS ⇒aAa ⇒aSbAa ⇒aabAa ⇒aabbaa

S→aAS　　S→a

A→SbA　　A→SS　　A→ba



最左推导和最右推导

- 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

举例

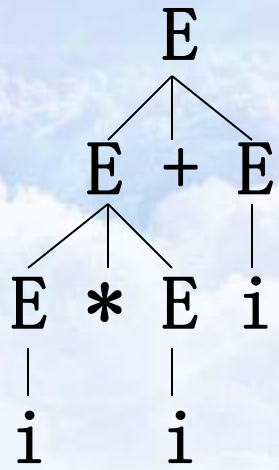
G[E]:

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

- 若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
  或者，若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

文法的二义性

- 判定任给的一个上下文无关文法是否二义，或它是否产生一个先天二义的上下文无关语言，这两个问题是递归不可解的，但可以为无二义性寻找一组充分条件
- 文法可以经过改写，从而构造出一个等价的无二义文法：
- G[E]: E → T | E+T

  T → F | T*F

  F → (E) | i

原文法G[E]:

E → i

E → E+E

E → E*E

E → (E)

2.6 句型的分析

- 句型的分析是识别一个符号串是否为某文法的句型，是某个推导的构造过程。
- 当给定一个符号串时，试图按照某文法的规则为该符号串构造推导或语法树，依此识别出它是该文法的一个句型或句子。
- 在语言的编译实现中，把完成句型分析的程序称为分析程序或识别程序，分析算法又称识别算法。

句型的分析

- 从左到右的分析算法：
- 总是从左到右地识别输入符号串，首先识别符号串中的最左符号，进而识别右边的一个符号。分为两大类：
  - 自上向下：从文法的开始符号出发，反复使用各种产生式，寻找“匹配”于输入符号串的推导。
  - 自下而上：从输入符号串开始，逐步进行“归约”，直至归约到文法的开始符号

两种句型分析算法

- 两种方法反映了两种语法树的构造过程。
  - 自上而下方法是从文法符号开始，将它做为语法树的根，向下逐步建立语法树，使语法树的结果正好是输入符号串.
  - 自下而上方法则是从输入符号串开始，以它做为语法树的结果，自底向上地构造语法树.

2.6.1 自上而下的分析方法

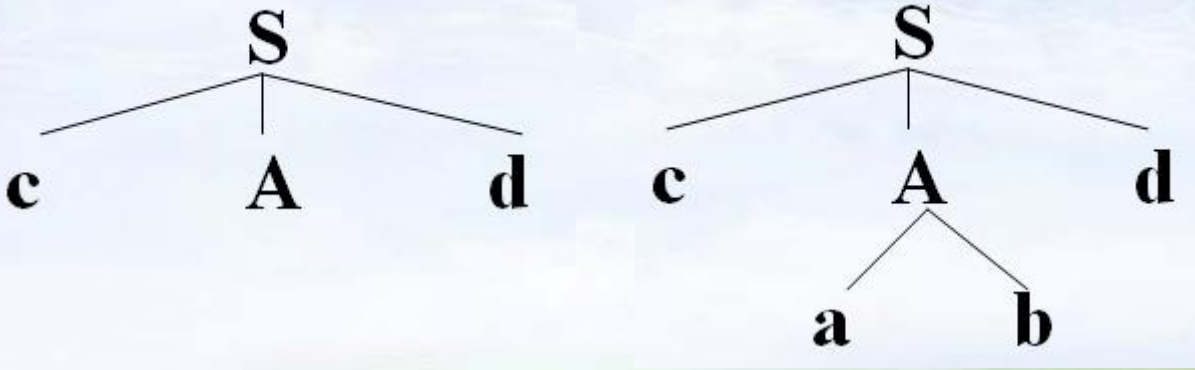
例：文法G：S → cAd

A → ab

A → a

识别输入串w=cabd是否为该文法的句子

S



推导过程：S ⇒ cAd　　cAd ⇒ cabd

2.6.2 自下而上的分析方法

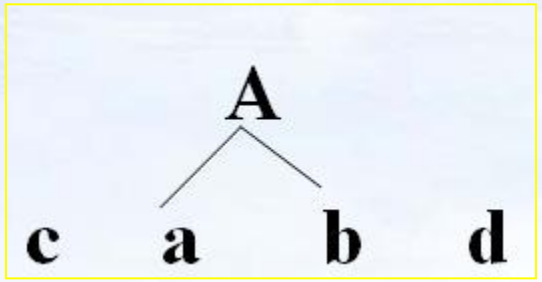
例：文法G：　S → cAd

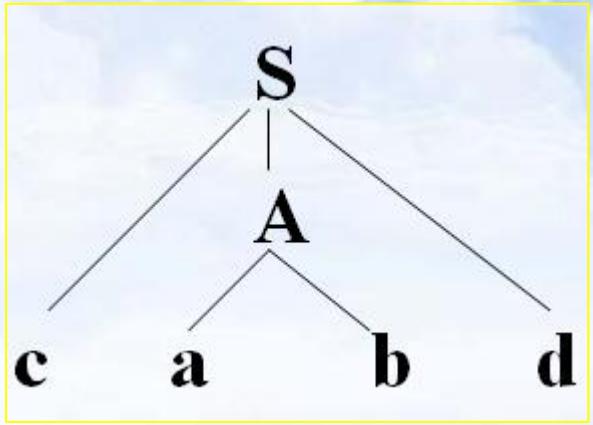
A → ab

A → a

识别输入串w=cabd是否该文法的句子

c　a　b　d



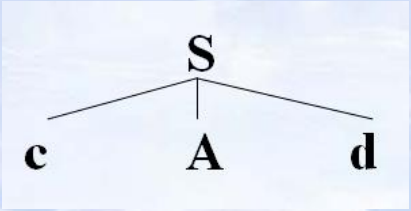


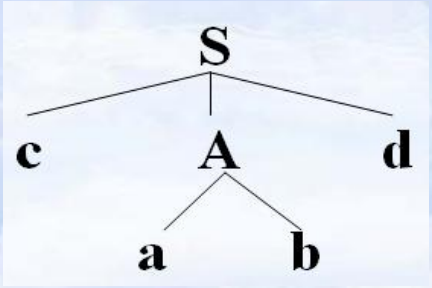
归约过程构造的推导：　cAd ⇒ cabd　　S ⇒ cAd

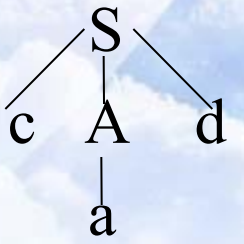
自上而下的语法分析的有关问题

文法G：S → cAd　A → ab　　A → a　自上而下对句子cabd构造语法树

S







宣告分析失败（其意味着，识别程序不能为串cabd构造语法树，即cabd不是句子）

-显然是错误的结论。

导致失败的原因是在分析中对A的选择不是正确的。

自下而上的语法分析中的有关问题

文法G：S → cAd　A → ab　　A → a　自下而上对句子cabd构造语法树

达不到归约到S的结果，因而也无从知道cabd是一个合法的句子

？？？？

1）在自上而下的分析方法中如何选择使用哪个产生式进行推导？

假定要被代换的最左非终结符号是B，且有n条规则：B→A1|A2|…|An，那么如何确定用哪个右部去替代B？

2）在自下而上的分析方法中如何识别可归约串？

在分析程序工作的每一步，都是从当前串中选择一个子串，将它归约到某个非终结符号，该子串称为“可归约串”

刻画“可归约串”

文法G[S]，α β δ是该文法的一个句型

句型的短语

一棵子树的所有叶子自左至右排列起来形成一个相对于子树根的短语。

$S \overset{*}{\Rightarrow} \alpha A \delta$ 且 $A \overset{+}{\Rightarrow} \beta$，则称β是句型α β δ相对于非终结符A的短语

句型的直接短语

仅有父子两代的一棵子树，它的所有叶子自左至右排列起来所形成的符号串。

若有$A \Rightarrow \beta$，则称β是句型α β δ相对于非终结符A的直接短语

句型的句柄

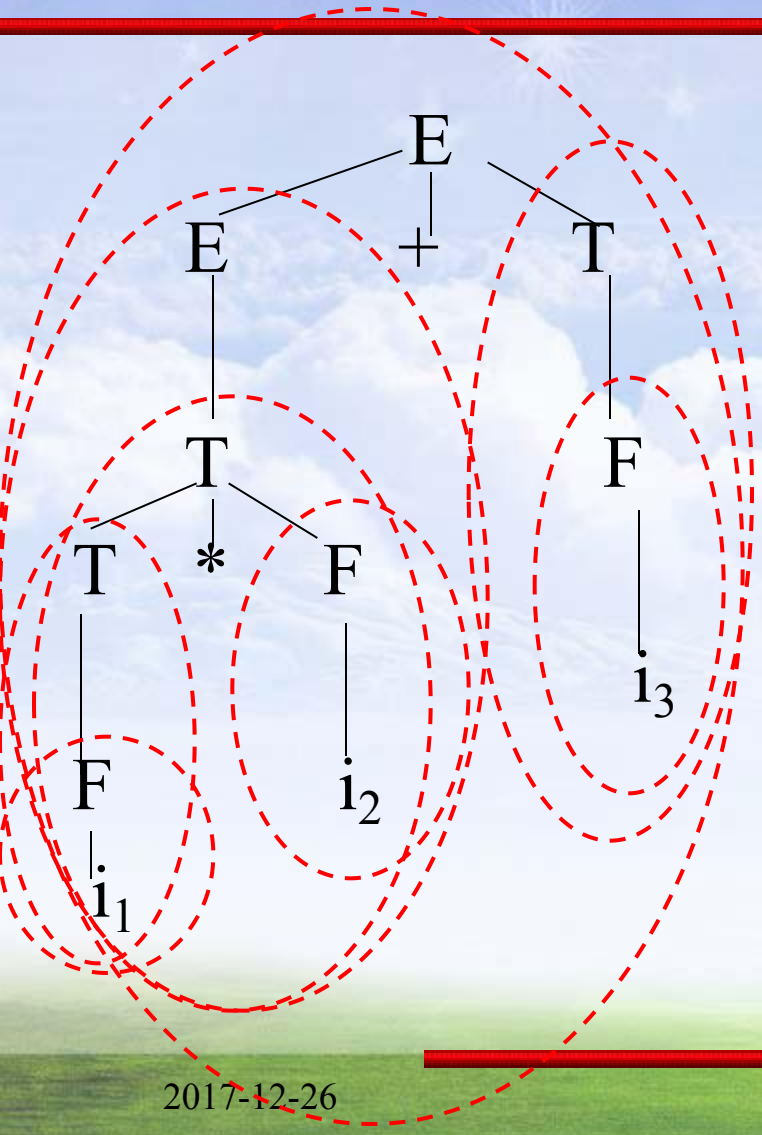
一个句型的分析树中最左那棵只有父子两代的子树的所有叶子的自左至右排列

一个句型的最左直接短语称为该句型的句柄

用子树解释短语，直接短语，句柄

- 短语：一棵子树的所有叶子自左至右排列起来形成一个相对于子树根的短语。
- 直接短语：仅有父子两代的一棵子树，它的所有叶子自左至右排列起来所形成的符号串。
- 句柄：一个句型的分析树中最左那棵只有父子两代的子树的所有叶子的自左至右排列。

例 ： i*i+i 的短语、直接短语和句柄



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

短语： i_1* i_2+ i_3， i_1* i_2 ， i_1 ， i_2， i_3

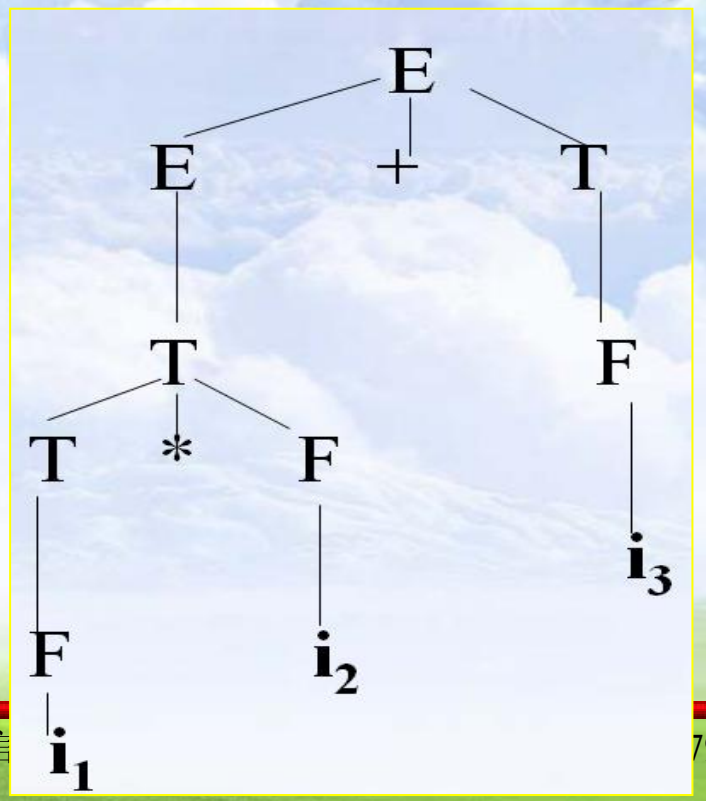
直接短语： i_1 ， i_2 ， i_3

句柄： i_1

句柄举例

G[E]：E→E+T|T
T→T*F|F
F→(E)|i
句型：i*i+i

$\underline{i_1} * i_2 + i_3$
$<= \underline{F} * i_2 + i_3$
$<= T * \underline{i_2} + i_3$
$<= \underline{T * F} + i_3$
$<= \underline{T} + i_3$
$<= E + \underline{i_3}$
$<= E + \underline{F}$
$<= \underline{E + T}$
$<= E$



2.7 文法实用中的一些说明

文法中不含有*有害规则*和*多余规则*

- 有害规则：形如U→U的产生式。会引起文法的二义性
- 多余规则：指文法中任何句子的推导都不会用到的规则

文法中不含有不可到达和不可终止的非终结符

1）文法中某些非终结符不在任何规则的右部出现，该非终结符称为不可到达。

2）文法中某些非终结符，由它不能推出终结符号串，该非终结符称为不可终止。

化简文法举例

- 例：G[S]：

1) S→Be
2) B→Ce
3) B→Af
4) A→Ae
5) A→e
6) C→Cf
7) D→f

D为不可到达

C为不可终止

产生式 2），6），7）为多余规则应去掉。

非终结符在推导中出现的保证条件

对于文法G[S]，为了保证任一非终结符A在句子推导中出现，必须满足如下两个条件：

1. A必须在某句型中出现

即有$S \overset{*}{\Rightarrow} \alpha A \beta$，其中α，β属于$V^*$

2. 必须能够从A推出终结符号串t来

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

上下文无关文法中的ε规则

- 上下文无关文法中某些规则可具有形式A→ε，称这种规则为ε规则
- 因为ε规则会使得有关文法的一些讨论和证明变得复杂,有时会限制这种规则的出现
- 两种定义的唯一差别是ε句子在不在语言中
- 如果语言L有一个有穷的描述，则L_1=L∪{ε}也同样有一个有穷的描述。
- 若L是上下文有关语言、上下文无关语言或正规语言，则L∪{ε}和L-{ε}分别是上下文有关语言、上下文无关语言和正规语言。

复 习

- 文法的直观概念
- 符号和符号串
- 文法和语言的形式定义
- 文法的类型(0、1、2、3)
- 上下文无关文法及其语法树（最左/右推导、二义性）
- 句型的分析（自上而下/自下而上、短语、直接短语、句柄）
- 有关文法实用中的一些说明

作业题

- P34：10，11，12（1）（2），13（1）