第 2 章 文法和语言

程序设计语言

- 程序设计语言
  - 语法：一种规则，用它可以形成和产生一个合适的程序。阐明语法的工具是文法。
  - 语义
    - 静态语义：一系列的限定规则，并确定哪些合乎语法的程序是合适的。
    - 动态语义：运行语义或执行语义，表明程序要做什么，要计算什么。

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：元素的非空集合，也称为符号集
  - 例：∑={0，1} A={a, b, c}
- 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，固有头和固有尾：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，$x^n=xx^{n-1}=x^{n-1}x$

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
- 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$\varepsilon x=x\varepsilon=x$

符号和符号串

- 集合的闭包：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$x \notin \Sigma^*$。对于所有的Σ，有$\varepsilon \in \Sigma^*$。

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$\alpha \rightarrow \beta$或$\alpha ::= \beta$的(α, β)有序对，α称为规则的左部，β称作规则的右部

文法的定义

- 定义2.1：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；
  - P为规则($\alpha \rightarrow \beta$)的集合，$\alpha \in (V_N \cup V_T)^*$且至少包含一个非终结符，$\beta \in (V_N \cup V_T)^*$；
  - V_N，V_T和P是非空有穷集。
  - S称作识别符号或开始符号，它是一个非终结符，至少要在一条产生式中作为左部出现。
  - V_N和V_T不含公共的元素，即$V_N \cap V_T = \Phi$
  - 用V表示$V_N \cup V_T$，称为文法G的字母表或字汇表

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

直接推导

- 定义2.2：设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$v \overset{+}{\Rightarrow} w$，v推导出w，或w归约到v，推导长度为n

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

推导举例

例：G[S]： $S \rightarrow 0S1$， $S \rightarrow 01$

$0S1 \Rightarrow 00S11$

$00S11 \Rightarrow 000S111$

$000S111 \Rightarrow 00001111$

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

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

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

句型和句子

- 定义3.5：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3.6：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3.6可以看出两点：
  - X是句型
  - X仅由终结符号组成
- 例： G[S]： $S\rightarrow 0S1$， $S\rightarrow 01$

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

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

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

- 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

- 设文法G=（V_N，V_T，P，S），若P中任一产生式$\alpha\rightarrow\beta$，都有$|\beta|\geq|\alpha|$，仅仅$S\rightarrow\varepsilon$除外，则G是一个1型文法或上下文有关文法。
- 其他定义方法：设文法G[S]，若P中任一产生式$\alpha\rightarrow\beta$的形式为$\alpha_1 A\ \alpha_2\rightarrow\alpha_1\ \beta\ \alpha_2$，其中$\alpha_1$，$\beta$，$\alpha_2\in(V_N\cup V_T)^*$，$\beta\neq\varepsilon$，$A\in V_N$
- 例：文法G[S]：S→CD　　C→aCA

  CA→Ca　　CaD→daD

  dAc→dec

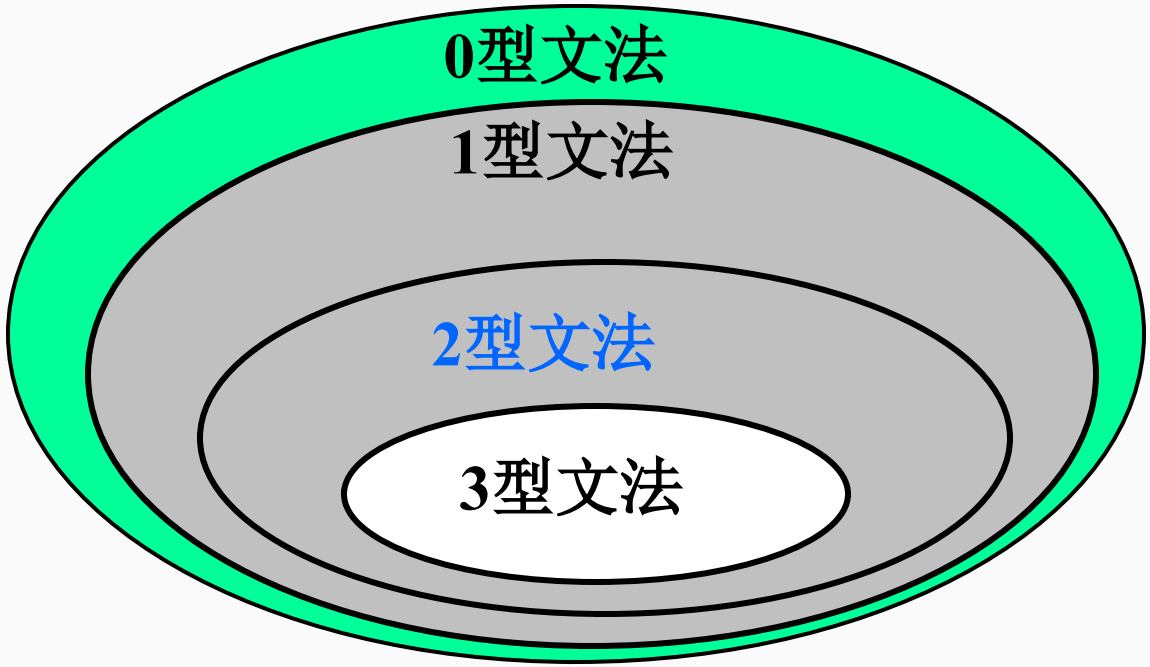
2型文法

- 设文法G=（V_N，V_T，P，S），若P中任一产生式$\alpha\rightarrow\beta$，满足：α是一个非终结符，$\beta\in(V_N\cup V_T)^*$，则G是一个2型文法或上下文无关文法。
- 有时将2型文法的产生式表示为$A\rightarrow\beta$的形式，其中$A\in V_N$，也就是说用β代替非终结符A时，与A所在的上下文无关，因此取名为上下文无关
- 例：文法G[S]：$S\rightarrow AB$　　$A\rightarrow BS|0$
  $B\rightarrow SA|1$

3型文法

- 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，$a \in V_T \cup \{\varepsilon\}$，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

四种文法之间的关系



文法和语言

- 0型文法产生的语言称为0型语言
- 1型文法或上下文有关文法产生的语言称为1型语言或上下文有关语言
- 2型文法或上下文无关文法产生的语言称为2型语言或上下文无关语言
- 3型文法或正则（正规）文法产生的语言称为3型语言或正则（正规）语言

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〉

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$\in V_N$
  - 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$\rightarrow A_1A_2$，…，A_k一定是P中的一个产生式
- 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：

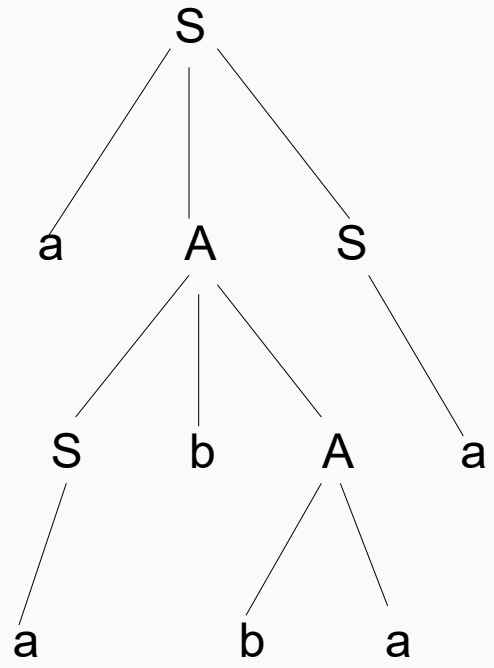
  S→aAS

  A→SbA

  A →SS

  S →a

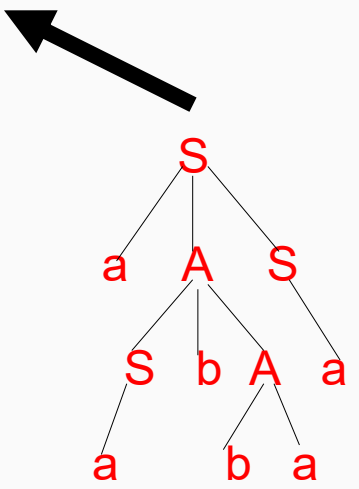
  A →ba



G的一棵语法树

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
  - S ⇒aAS ⇒aSbAS ⇒aabAS ⇒aabbaS ⇒aabbaa
  - S ⇒aAS ⇒aSbAS ⇒aSbAa ⇒aabAa ⇒aabbaa



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

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
- 自然语言是二义性语言。二义性句子举例：

  I was told to read a book by Tom.

二义性文法举例

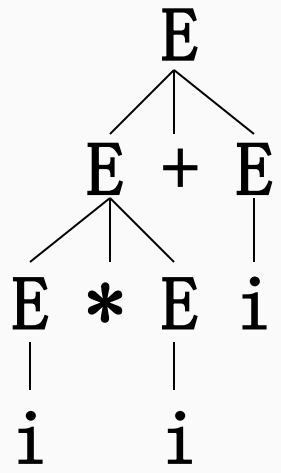
G[E]:

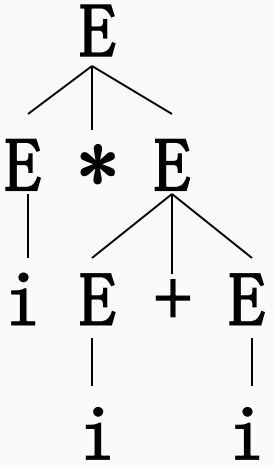
$E \rightarrow i$

$E \rightarrow E+E$

$E \rightarrow E*E$

$E \rightarrow (E)$





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

- 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]: $E \rightarrow T \mid E+T$

  $T \rightarrow F \mid T*F$

  $F \rightarrow (E) \mid i$

原文法G[E]:

$E \rightarrow i$

$E \rightarrow E+E$

$E \rightarrow E*E$

$E \rightarrow (E)$

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。进一步说，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，当符号串全部由终结符号组成时，就是识别它是不是某文法的句子。
- 对于程序设计语言来说，我们要识别程序设计语言的程序，程序是定义程序设计语言的句子。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

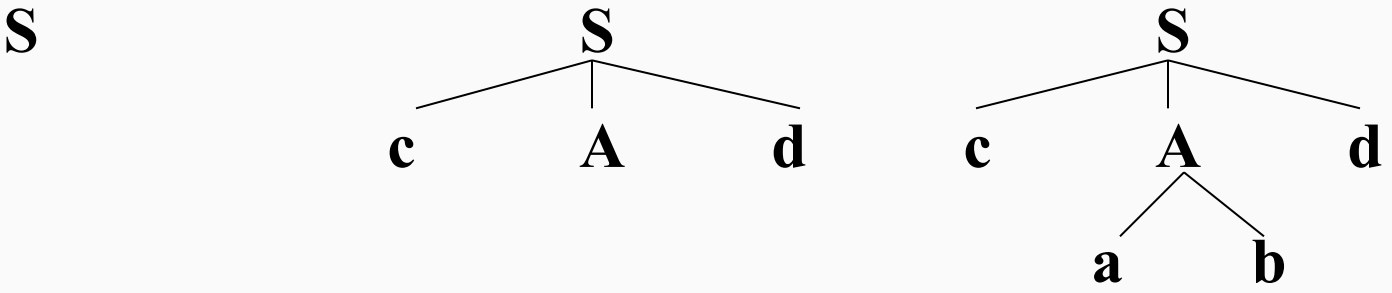
3.6.1 自上而下的分析方法

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



推导过程：S ⇒ cAd　　cAd ⇒ cabd

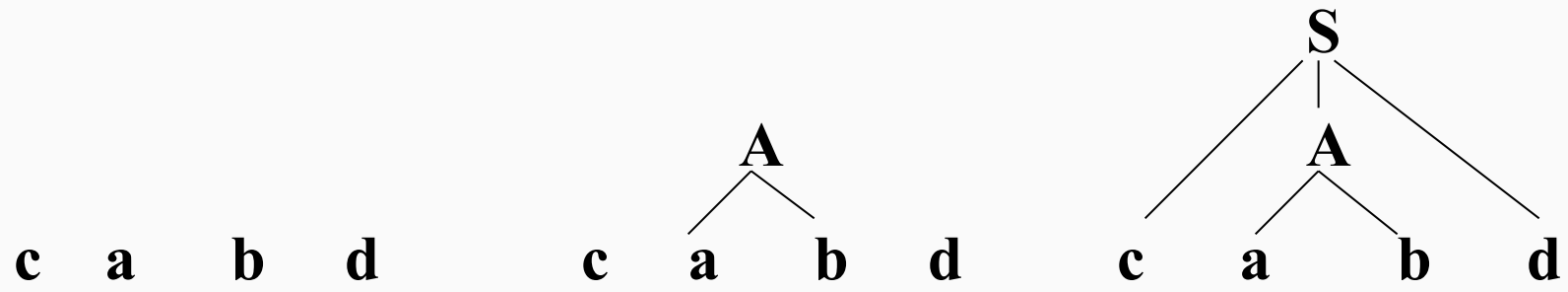
3.6.2 自下而上的分析方法

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



归约过程构造的推导：　cAd ⇒ c<u>ab</u>d　　S ⇒ cAd

自上而下的语法分析的有关问题

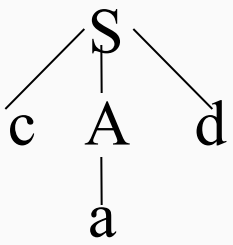
若S ⇒ cAd 后选择**A → a**扩展A，
S ⇒ cAd ⇒ cad

则w的第二个符号可以与叶子结点a得以匹配，但第三个符号却不能与下一叶子结点d匹配

宣告分析失败（其意味着，识别程序不能为串cad构造语法树，即cad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。

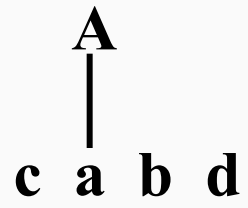


文法**G**：**S → cAd**
A → ab
A → a

自下而上的语法分析中的有关问题

对串cabd的分析中，如果不是选择ab用产生式A→ab,而是选择a用产生式A→a将a归约到了A

那么最终就达不到归约到S的结果，因而也无从知道cabd是一个合法的句子



文法G：S → cAd
A → ab
A → a

句型分析的有关问题

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

$S \overset{*}{\Rightarrow} \alpha A \delta$ 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

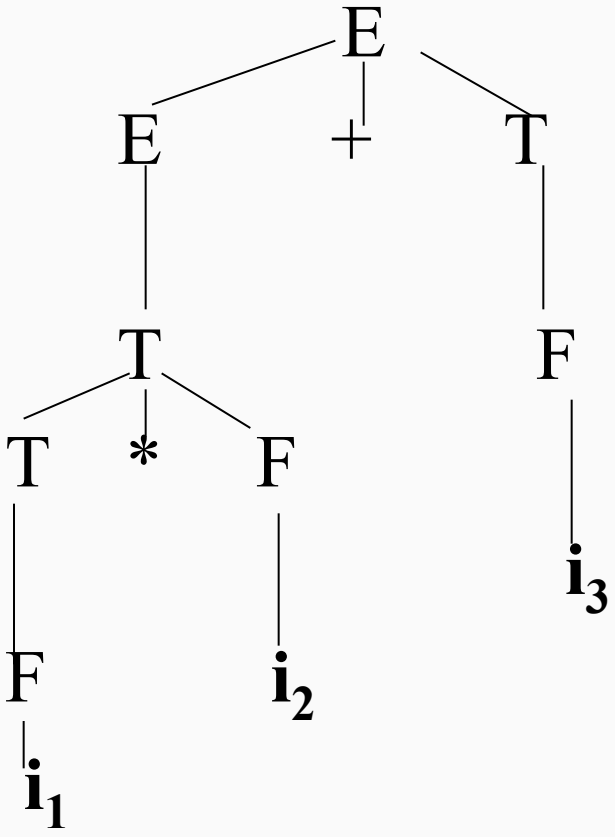
句型的直接短语

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

一个句型的最左直接短语称为该句型的句柄

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

$\underline{i_1} * i_2 + i_3$

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

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

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

$<= \underline{T} + i_3$

$<= E + \underline{i_3}$

$<= E + \underline{F}$

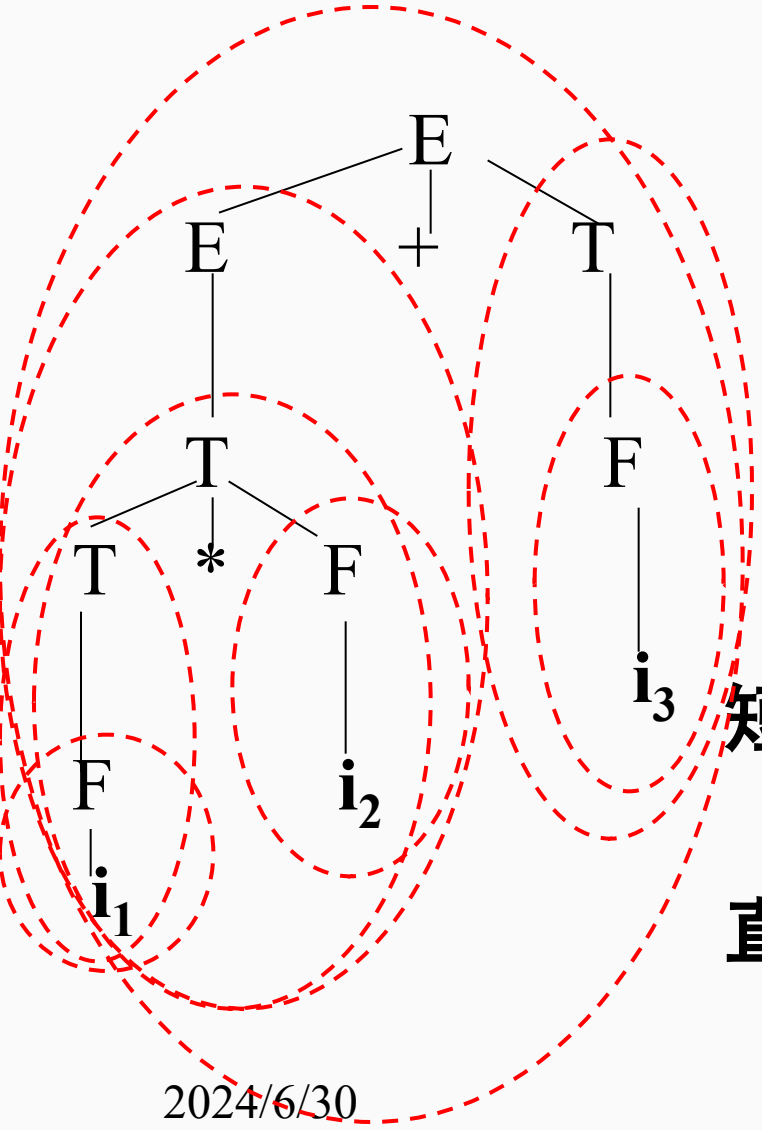
$<= \underline{E + T}$

$<= E$

通过语法树查找短语和直接短语

- 短语：找到某语法树的所有非叶结点为根的子树，该子树的叶结点从左向右组成的符号串是对应于该子树的根结点的短语
- 直接短语：在查找短语过程中，如果该子树为两层，则该短语为对应于该子树的根结点的直接短语。

举例查找短语和直接短语



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

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$A \rightarrow \varepsilon$，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

文法构思的启示是要找出语言的有穷描述，而如果语言L有一个有穷的描述，则$L_1=L \cup \{\varepsilon\}$ 也同样有一个有穷的描述，并且可以证明，若L是上下文有关语言、上下文无关语言或正规语言，则$L \cup \{\varepsilon\}$ 和$L-\{\varepsilon\}$ 分别是上下文有关语言、上下文无关语言和正规语言。

定理3.1

- 若L是由文法G=(V_N,V_T,P,S)产生的语言，P中的每一个产生式的形式均为A→**α**，**其中A∈**V_N，**α** **∈**(V_N∪V_T)*，则L能由这样一种文法产生，即每一个产生式或者为A→B形式，其中**A**□**一非**□□**符**，**即A∈**V_N，β **∈**(V_N∪V_T)$^+$，或者为S→**ε形式，且S不出现在任何产生式的右边。**

定理3.2

- **如果**$G=(V_N,V_T,P,S)$是一个上下文有关文法，则存在另外一个上下文有关文法G1，它所产生的语言与G产生的相同，其中G1的开始符号不出现在任何产生式的右边。又如果G是一个上下文无关文法，也能找到一个上下文无关文法G1，如果G是一个正规文法，则也能找到这样一个正规文法G1。

作业题

- 1，2，11
- 12（1）（2）
- 13（1）