第2章　文法和语言

目前形式语言与自动机理论已成为计算机科学和软件工程学科中的一个重要分枝。

Hopcroft 曾说：“在不了解语言及自动机理论的技术和结果的情况下，就不能对计算机科学进行严肃的研究。”

返回章
返回节
退出

- 一个程序设计语言是一个记号系统，包括语法和语义两个方面。一个语言的语法是指一组规则，用它可以形成和产生一个合适的程序。
- 阐明语法的一个工具是文法，这是形式语言理论的基本概念之一。
- 阐明语义比阐明语法困难得多，目前仍无一种公认的形式语义系统来自动构造编译系统。本章将介绍文法和语言的概念，重点讨论上下文无关文法及其句型分析中的有关问题。

本章内容

- 2.1 符号和符号串
- 2.2 文法和语言的形式定义
- 2.3 文法的类型
- 2.4 上下文无关文法及其语法树
- 2.5 句型的分析
- 2.6 有关文法实用中的一些说明
- 2.7 扩展的BNF

返回章
返回节

2.1 符号和符号串

- 结合语言的形式定义，首先讨论符号和符号串的有关概念。

1、字母表

元素的有穷非空集合。

如PASCAL语言的字符是由字母、数字、若干专用符号及BEGIN、IF之类的保留字组成。

2、符号串

由字母表中的符号组成的任何有穷序列。

如：PASCAL语言程序是PASCAL语言字母表上的一个符号串，不含任何符号的符号串为空符号串，记为ε。

返回章
返回节
退出

3、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；同样如果y非空，那么x是固有头。

如:设z=abc，那么z的头是ε，a, ab, abc, 除abc外，其它都是固有头；z的尾是ε，c, bc, abc, z的固有尾是ε，c, bc。

4、符号串的运算

（1）符号串的连接

设x和y是符号串

x和y的连接xy是把y的符号写在x的符号ε后得的符号串。

如：x=ST, y=abu,

则xy=STabu 显然有εx=xε=x。

（2）符号串的方幂

设x是符号串，把x自身连接n次得x的几次方幂x^n。

如：设x=ab

则$x^0=\varepsilon$　$x^1=ab$　$x^2=abab$　$x^3=ababab$

（3）符号串集合的乘积

设A和B为符号串集合，则A和B的乘积定义为

$AB=\{xy|x\in A$且$y\in B\}$

如：a={a, b}, B={00, 11}

则AB={a00, a11, b00, b11}

显然：$\{\varepsilon\}A=A\{\varepsilon\}=A$

返回章
返回节
退出

（4）符号串集合的方幂

设A为符号串集，则A的n次方幂A^n定义为:

$$A^n = \underbrace{AA\ldots\ldots A}_{n个A} = AA^{n-1} = A^{n-1}A$$

（5）符号串集合的正闭包A^+

$A^+ = A^1 \cup A^2 \cup \ldots \cup A^n \cup \ldots$

（6）符号串集合的闭包A*

$A^* = A^0 \cup A^+ = \{\varepsilon\} \cup A^+$

如：设有正字母表Σ={0，1}

则$\Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \cup \ldots \cup \Sigma^n \cup \ldots$

$= \{\varepsilon, 1, 1, 00, 01, 10, 11, 000, 001, \ldots\ldots\}$

2.2 文法和语言的形式定义

文法是描述语言的语法结构的形式规则。

1、文法的形式定义

2、推导的概念

3、句型、句子的定义

4、语言的定义

5、文法等价定义

1、文法的形式定义

- 文法G定义为四元组（V_N，V_T，P，S）　其中:

（1）V_N为非终结符号集

非终结符号表示一个语言短语（或语法成分、语法单位）。　如 程序、语句、表达式等。一般用大写字母或用〈 〉括起表示非终结符号。

（2）V_T为终结符号集

终结符号：组成语言的基本符号。是文法中不属于非终结符号集合的符号。一般用小写字母或不带〈 〉的符号表示。如程序设计语言的单词符号。

设$V=V_N \cup V_T$，称V为文法G的字母表。

（3）P 为产生式（也称规则）的集合。

产生式的形式：$\alpha \to \beta$或$\alpha ::= \beta$，其中$\alpha \in V^+$，$\beta \in V^*$

（4）S 称作识别符号或开始符号，是一个非终结符号。

一般表示此文法定义的最大语法短语，至少要在一条产生式 中作为左部出现。

（例2_2_1）(例2_2_2)

退出

返回章
返回节

2、推导的概念

（1）直接推导的定义

如$\alpha \to \beta$是文法G=（V_N，V_T，P，S）的规则（或说是P中的一产生式），γ和δ是V^*中的任意符号，若有符号串v,w满足：

v=$\gamma\alpha\delta$, w=$\gamma\beta\delta$，则说v（应用规则$\alpha \to \beta$）直接产生 w，或说，w是v的直接推导，或说，w直接归约到v，记作$v \Rightarrow w$。

即$\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ 使用产生式$\alpha \to \beta$

如：对上面例1中的文法G：$S \to 0S1$ $S \to 01$设v=0S1, w=0011,

直接推导：$0S1 \Rightarrow 0011$

（使用的规则：$S \to 01$，这里γ=0，δ=1)。

v=S，w=0S1,直接推导：$S \Rightarrow 0S1$

（使用的规则：$S \to 0S1$，这里$\gamma=\varepsilon$，$\delta=\varepsilon$)。

（2）推导的定义

如果存在直接推导的序列：$v=w0 \Rightarrow w1 \Rightarrow w2 \ldots \Rightarrow wn=w$, $(n>0)$ 则称v推导出（产生）w（推导长度为n），或称w归约到v。记作 $v \overset{+}{\Rightarrow} w$ 。即一步或多步推导。

若有 $v \overset{+}{\Rightarrow} w$ 或 $v=w$ 则记作 $v \overset{*}{\Rightarrow} w$，即0步或多步推导。

(例2_2_3)

3、句型、句子的定义

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[S]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为G[S]的句子。

例如：S，0S1，000111都是上例文法G的句型，

其中000111是G的句子。

返回章
返回节
退出

4、语言的定义

文法G产生的语言记为L（G），它是文法G产生的全部句子的集合。　　即：

$L(G)=\{X|S\overset{*}{\Rightarrow}X$，其中S为文法识别符号且$x\in V_T{}^*\}$

(例2_2_4)(例2_2_5)

5、文法等价定义

若$L(G_1)=L(G_2)$则称文法G_1和G_2是等价的。

如:文法G[A]：$A\rightarrow 0R$　与文法G[S]：$S\rightarrow 0S1$

$A\rightarrow 01$　　　$S\rightarrow 01$

$R\rightarrow A1$

是等价的。

退出
返回章
返回节

2.3 文法的类型

文法的定义和记号

$G=(V_N, V_T, P, S)$ （ $V_N \cup V_T = V$， $V_N \cap V_T=\phi$ ）

是 N.Chomsky (乔姆斯基)在1956年描述形式语言时首先提出来的。Chomsky把文法分成四种类型，每种类型的文法对应一类语言，可分别构造四类自动机来接受(识别)它们。

- 0型文法：定义0型语言，对应Turing机；
- 1型文法：定义1型语言，对应线性限界自动机；
- 2型文法：定义2型语言，对应非确定下推自动机；
- 3型文法：定义3型语言，对应有限自动机。

这几类文法的差别在于对产生式施加不同的限制

返回章
返回节

2.3 文法的类型

文法类别	产生式形式	产生的语言	说　明
0型文法（短语文法）	$\alpha\to\beta$，其中$\alpha\in V^+$且至少含一非终结符, $\beta\in V^*$	0型语言（递归可枚举）	对产生式基本无限制
1型文法（上下有关文法CSG）	$\alpha\to\beta$，其中再限制$\|\beta\|\geq\|\alpha\|$，仅$S\to\varepsilon$除外，但S不得出现在任何产生式右部	1型语言（上下文有关语言）	有些产生式形式为$\alpha_1A\alpha_2\to\alpha_1\beta\alpha_2$其中$\alpha_1$、$\alpha_2\in V^*$，$A\in V_N$ $\beta\in V^+$，即:将A替换成β时，必须考虑A的上下文
2型文法(上下文无关)	$\alpha\to\beta$其中$\alpha\in V_N$，$\beta\in V^*$	2型语言(上下文无关)	将α替换成β时，无需考虑α的上下文(CFG)
3型文法（正规文法）	（1）$A\to aB$或$A\to a$ 或者： （2）$A\to Ba$或$A\to a$ 其中A、$B\in V_N$，$a\in V_T$	3型语言（正规语言）	文法形式 （1）为右线性 （2）为左线性

用L_0，L_1，L_2，L_3分别表示0型、1型、2型、3型语言，则有$L_0\supset L_1\supset L_2\supset L_3$。

返回章
返回节
退出

Eg. 1. 下列文法 G[S] 是一个0型文法：

G[S]=（{S，A，B，C，D，E}，{a}，P，S}）

其中P由如下产生式组成

$S \to ACaB$, $Ca \to aaC$, $CB \to DB$,

$CB \to E$, $aD \to Da$, $AD \to AC$, $aE \to Ea$, $AE \to$

ε

G[S] 产生的语言为 $L_0=\{a^i \mid i$是2的正整数次方$\}$

$=\{aa, aaaa, aaaaaaaa, \ldots\}$

返回章
返回节

退出

Eg. 2. 下列文法 G[S] 是一个不严格的1型文法：

G[S]=（{S， A， B， C}，{a, b, c}， P， S})

其中P由如下产生式组成

$S \rightarrow \varepsilon$, $S \rightarrow A$, $A \rightarrow aABC$, $A \rightarrow abC$,

$CB \rightarrow BC$, $bB \rightarrow bb$, $bC \rightarrow bc$, $cC \rightarrow cc$

G[S] 产生的语言为 $L_g=\{a^i b^i c^i \mid i>=0\}$

但去掉G[S] 中的$S \rightarrow \varepsilon$ 得到G'[S] 则

$L_1=\{a^i b^i c^i \mid i>=1\}$是一个1型文法产生的语言。

Eg. 3. 下列文法 G[S] 是一个2型文法：

G[S]=（{S}，{a，b}，{$S\rightarrow aSb$, $S\rightarrow ab$}，S})

G[S] 产生的语言为 $L_2=\{a^i b^i \mid i>=1\}$

注意：通常使用的BNF表示法等价于2型文法。因此，凡是能用BNF定义的语言都是上下文无关语言。

Eg. 4. 下列文法 G[S] 是一个3型文法：

G[S]=（{S，A，B}，{a, b}，P，S})

其中P由如下产生式组成

$S\rightarrow aA$, $A\rightarrow bA \mid aB \mid b$, $B\rightarrow aA$

G[S] 产生的语言为 $L_3=\{a(b \mid aa)^i b \mid i>=0\}$

返回章
返回节

2.4 上下文无关文法及其语法树

- 1、用上下文无关文法描述程序设计语言的语法结构
- 2、语法树定义
- 3、由语法树定义句型
- 4、最左、最右推导
- 5、文法的二义性

返回章
返回节
退出

1、用上下文无关文法描述程序设计语言的语法结构

- 例如，文法G=（{E}，{+，*，i，（，）}，P，E）其中P为：

  E→E+E

  E→E*E

  E→(E)

  E→i

  这里的非终结符E表示一类算术表达式。i表示程序设计语言中的“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E1和E2是算术表达式，则E1+E2，E1*E2和（E1）也是算术表达式。

返回章
返回节
退出

• 又如 描述一种简单赋值语句的产生式为：

<赋值语句>→i:=E

描述条件语句的文法片断为：

<条件语句>→if<条件>then<语句>

|if<条件>then

<语句>else<语句>。

2、语法树定义

- 给定文法G=（V_N，V_T，P，S），对于G的任何句型都能构造与之关联的语法树（推导树）。
- 这棵树满足下列4个条件：

  1．每个结点都有一个标记，此标记是V的一个符号。

  2．根的标记是S。

  3．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A肯定在V_N中。

  4．如果结点n的直接子孙，从左到右的次序是结点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。

- 语法树是描述上下文无关文法的句型推导的直观方法。

(例2_4_1)

返回章

返回节

退出

3、由语法树定义句型

- 句型：在一棵树生长过程的任何时刻，所有那些端末结点自左至右的排列，就是一个句型。

4、最左、最右推导

- 在推导的任何一步$\alpha \Rightarrow \beta$,

  如果都是对α中最左非终结符进行，则称为最左推导。

  如果都是对α中最右非终结符进行替换，则称为最右推导。

  最右推导也称为**规范推导**，所得句型称为规范句型。

  如前例(例2_4_1)，第一种推导是最左推导，另一种推导是最右推导。

返回章
返回节
退出

5、文法的二义性

- 定义：如果一个文法存在某个句型对应两棵不同的语法树，则说这个文法是二义的。
- 或者说，若一个文法中存在某个句型，它有两个不同的最左（最右）推导，则这个文法是二义的。

  如文法G： E→E+E|E*E|i

- eg. 句型 i*i+i 对应两个不同的最左推导及相应的语法树如下：

返回章
返回节
退出

- 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：

  $E \Rightarrow E*E \Rightarrow E*E+E$

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

因此文法G是二义的。

排除文法二义性通常有两种方法:

（1）在语义上加些限制

（2）重新构造一个无二义性的文法

返回章
返回节

2.5 句型的分析

- 句型的分析：就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。
- 分析方法分两大类：自上而下分析法和自下而上分析法

- 1、自上而下的分析方法
- 2、自下而上的分析法
- 3、句型分析的有关问题
- 4、有关文法的实用限制

返回章
返回节
退出

1、自上而下的分析方法

- 自上而下的分析方法就是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。
- 如：文法G（此例为2_5_1）

（1）S→cAd

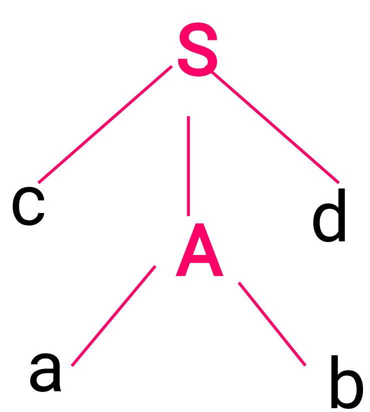
（2）A→a

（3）A→ab

分析输入串cabd是否该文法的句子

现根据推导构造语法树.

演示/继续



匹配输入串cabd

返回章
返回节
退出

2、自下而上的分析法

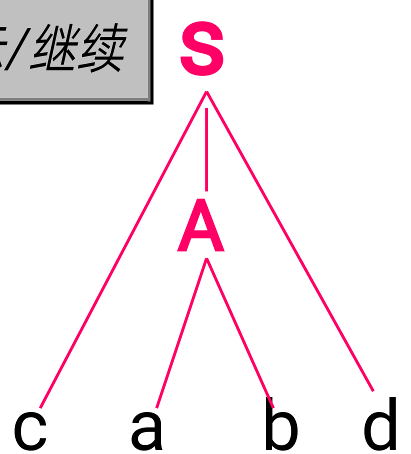
- 就是从输入符号串开始，逐步进行"归约"，直至归约到文法的开始符号。
- 如: 对文法G：（1）S→cAd

  （2）A→a

  （3）A→ab

  分析串Cabd（此例为2_5_2)

演示/继续



S已归约

采用产生式S→cAd

采用产生式A→ab

3、句型分析的有关问题

- 由上例，对句子cabd归约,

  若用产生式$A\rightarrow a$归约，得：cAbd无法归约到S。

  因此，并不是句型中的“串”只要是产生式的右部就可进行归约.

  满足什么条件 件的串才可进行归约呢？为此引入下面的概念。

（1）短语、句柄的定义

- 令G是一文法，S是文法的开始符号，$\alpha\beta\delta$是文法G的一个句型。

  若$S\overset{*}{\Rightarrow}\alpha A\delta\overset{+}{\Rightarrow}\alpha\beta\delta$（由$A\overset{+}{\Rightarrow}\beta$得）则称$\beta$是句型$\alpha\beta\delta$相对于非终结符A的短语。

  若$S\overset{*}{\Rightarrow}\alpha A\delta\Rightarrow\alpha\beta\delta$(由$A\rightarrow\beta$得)则称$\beta$是句型$\alpha\beta\delta$相对于$A\rightarrow\beta$的*直接短语*（也称简单短语）。

- 一个句型的最左直接短语称为该句型的**句柄**。

返回章
返回节
退出

- 由语法树定义短语：

  一棵子树（至少要有父子两代）的所有端末结点自左至右排列起来形成相对于子树根的短语。

- 若子树只有父子两代，则得到直接短语。

  (例2_5_3)

（2）规范归约

最右推导的逆过程称为规范归约。

- 自下而上分析过程通常采用规范归约。

即寻找句柄对句子进行归约。

(例2_5_4)

返回章
返回节
退出

2.6 有关文法实用中的一些说明

1. 有关文法的实用限制

- 在实用中应限制文法中含有如下规则：

（1）有害规则 文法中含形如U→U的产生式。它对描述语言没有必要，且会引起文法的二义性。

（2）多余规则 文法中任何一个句子的推导都用不到的规则。

（3）无用规则 文法中含形如U→V的产生式，即单产生式。

- 为保证文法G的任一非终结符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，αAδ。

（2）必须能够从A推导出终结符号串t。

返回章
返回节
退出

2.6 有关文法实用中的一些说明

2. 有关文法的二义性

（1）无二义性文法 如果一个文法所产生的每一句子都仅有一棵语法树，则称此文法为无二义性的。

（2）二义性的判定 1962—1963年 Floyd, Contor和Chomsky证明：上下文无关文法是否具有二义性是不可判定的。

- 但有些特殊的2型文法[例如LL（1）、LR（0）、LR（1）等文法]是无二义性的。
- 一个文法兼有左递归和右递归是导致二义性的常见原因。
- Eg. 文法G[E] E→E+E | E*E |(E) | i

是一个二义性文法。

2. 有关文法的二义性

（3）解决二义性　可将二义性文法G 改写为等价的无二义性文法G’。

Eg. 上述文法 G[E] 可改写为 G’ [E]:

$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid i$

则 G’ [E] 是无二义性的。

2.6 有关文法实用中的一些说明

3. 有关文法的化简和改造

- 包括以下几项工作：

（1）无用符号和无用产生式的删除。

（2）ε - 产生式的消除。

（3）单产生式的消除。

（4）左递归的消除。

2.7 扩展的BNF

- 前面叙述的表示语法规则的形式是Backus和Naur在ALGOL60报告中引入的，称为Backus—Naur Form，(也称Backus—Normal Form)，简化为BNF。

- 为了增加可续性和避免递归形式，引入了扩展的BNF，改为EBNF。

返回章
返回节
退出

2.7 扩展的BNF

- EBNF表示的符号说明如下。
- ‘< >’：用左右尖括号括起来的中文字表示语法构造成分，或称语法单位，为非终结符。
- ‘::=’：该符号的左部由右部定义，可读作‘定义为’。
- ‘|’：表示‘或’，为左部可由多个右部定义。
- ‘{ }’：表示花括号内的语法成分可以重复。在不加上下界时可重复0到任意次数，有上下界时为可重复次数的限制。
- ‘[ ]’：表示方括号内的成分为任选项。
- ‘( )’：表示圆括号内的成分优先。

例：PL/0语言文法的EBNF表示为(例2_6_1)

返回章
返回节
退出

例：PL/0语言文法的EBNF表示为：

<程序>::=<分程序>

<分程序>::=[<常量说明部分>][<变量说明部分>]

[<过程说明部分>]<语句>

<常量说明部分>::=CONST<常量定义>{，<常量定义>}；

<常量定义>::=<标识符>=<无符号整数>

<无符号整数>::=<数字>{<数字>}

PL/0语言文法的EBNF表示

- <变量说明部分>::=VAR<标识符>{，<标识符>};

  <标识符>::=<字母>{<字母>|<数字>}

  <过程说明部分>::=<过程首部><分程序>{；<过程说明部分>};

  <过程首部>::=PROCEDURE<标识符>;

  <语句>::=<赋值语句>|<条件语句>|<当型循环语句>
  |<过程调用语句>|<读语句>|<写语句>
  |<复合语句>|<空>

<赋值语句>::=<标识符>： =<表达式>

<复合语句>::=BEGIN<语句>{； <语句>}END

<条件>::=<表达式><关系运算符><表达式>|ODD<表达式>

<表达式>::=[+|-]<项>{<加法运算符><项>}

<项>::=<因子>{<乘法运算符><因子>}

<因子>::=<标识符>|<无符号整数>|‘（’<表达式>‘）’

<加法运算符>::=+|-

<乘法运算符>::=*|/

- <关系运算符>::==|#|<|<=|>|>=

  <条件语句>::=IF<条件>THEN<语句>

  <过程调用语句>::=CALL<标识符>

  <当型循环语句>::=WHILE<条件>DO<语句>

  <读语句>::READ‘（’<标识符>{，<标识符>}‘）’

  <写语句>::=WRITE‘（’<表达式>{，<表达式>}‘）’

  <字母>::=a|b|…|X|Y|Z

  <数字>::=0|1|2|…|8|9

作业

2-1 设英文小写字母集合L={a,b, …, z}，数字集合D={0,1, …,9},试问L(L ∪ D)*中长度不大于3的符号串共有多少个？请列出其中5个有代表性的符号串。

2-2 文法G=({U,V,S},{a,b,c},P,S)，其中产生式集合P为:

S→Uc|aV

U→ab

V→bc

试写出L(G[S])的全部元素。

2-3 文法G[N]为：

N→D|ND

D→0|1|2|3|4|5|6|7|8|9|

G[N]的语言是什么？

退出
返回章
返回节

作业

2-4 试确定下面文法的类型：

$G=(\{A,B,T,S\},\{a,b,c\},P,S)$

其中，$P=\{ S\to aTB|aB,\ T\to aTA|aA,\ B\to bc,\ Ab\to bA,\ Ac\to bcc\}$

2-5 试构造正规文法以生成以下语言：

(1) $\{a^m b^n \mid m > n > 0\}$；

(2) $\{a^n b^m c^m d^n \mid n,m\geq 1\}$。

2-6 考虑文法$G=(\{T, S\}, \{a, b, (,)\}, P, S)$，其中P为:

$S\to (T) \mid a \mid b$

$T\to T , S \mid S$

（1）给出 (a,(b)) 的最左推导和最右推导。

（2）给出 ((a))和(b,a) 的语法树。

（3）句子 (a,b,a) 是二义的吗？为什么？

2-7 考虑文法G=({T, S}, {a, b, (,)}, P, S)，其中P为:

$S \to (T) \mid a \mid b$

$T \to T , S \mid S$

（1）给出 (a,(b)) 的最左推导和最右推导。

（2）给出 ((a))和(b,a) 的语法树。

（3）句子 (a,b,a) 是二义的吗？为什么？

2-8 设文法G[S]为：

$S \to$ if (E) S else S | if (E) S | S:=a

试证明文法G[S]是二义性文法。

2-9 对于习题2-7的文法G，证明 (S,(T,S)) 是该文法的一个句型，指出这个句型的所有短语、直接短语和句柄。

本章要点

- 上下文无关文法及其语法树

  (1)用上下文无关文法描述程序设计语言的语法结构

  (2)最左、最右推导

  (3)文法的二义性

- 句型的分析

  (1) 自上而下的分析方法

  (2) 自下而上的分析法

  (3) 短语、句柄、规范归约

返回章
返回节
退出

谢谢

例2_2_1

文法G=（VN，VT，P，S）

其中VN={S}

VT={0，1}

P={S→0S1，S→01}

也可表示为如下形式：

G：S→0S1

S→01

约定第一条产生式的左部是文法的开始符号。

或 G[S]：S→0S1

S→01

例2_2_2

文法G=（VN，VT，P，S）

其中VN={标识符，字母，数字}

VT={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<字母>→b...<字母>→z<数字>→0<数字>→1...<数字>→9}

例2_2_3

- 对文法G：$S \rightarrow 0S1|01$
- 存在直接推导序列：

$V=0S1 \Rightarrow 00S11 \Rightarrow 000S111 \Rightarrow 00001111=W$

即：　$0S1 \stackrel{+}{\Rightarrow} 00001111$

也可记作：　$0S1 \stackrel{*}{\Rightarrow} 00001111$

返回

例2_4_1

- 设有文法G。

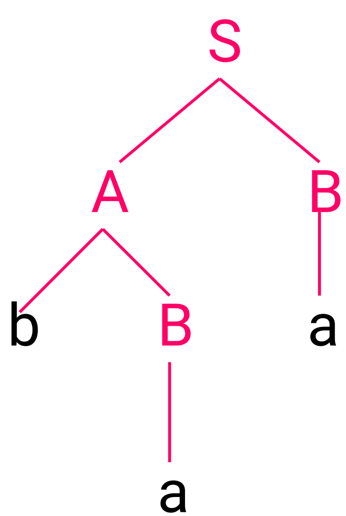
  S→AB

  A→Aa|Bb

  B→a|sb

推导句子baa， S⇒AB⇒bBB⇒baB⇒baa

另一种推导：S⇒AB⇒Aa⇒bBa⇒baa



演示/继续

此时可替换A或者B

返回

返回章
返回节

退出

例2_5_3

一棵子树（至少要有父子两代）的所有端末结点自左至右排列起来形成相对于子树根的短语。若子树只有父子两代，则得到直接短语。

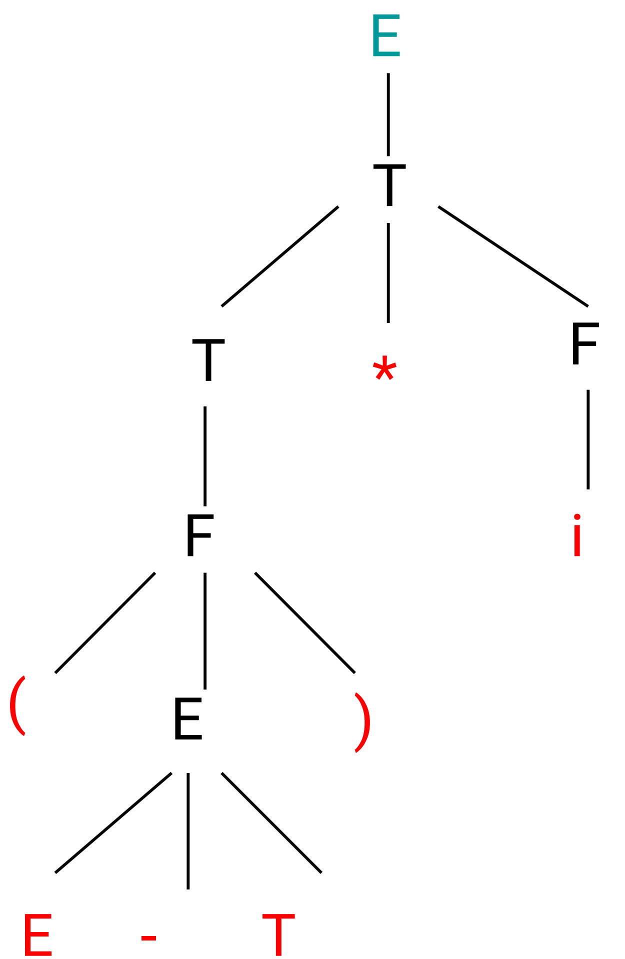
已知文法G为：

E -> E+T | E-T | T

T -> T*F | T/F | F

F -> (E) | i

要求：给出句型（E-T）*i 的所有短语和句柄。



短语为：

(E-T)*i

(E-T)

E-T

i

直接短语：

E-T

i

句柄：

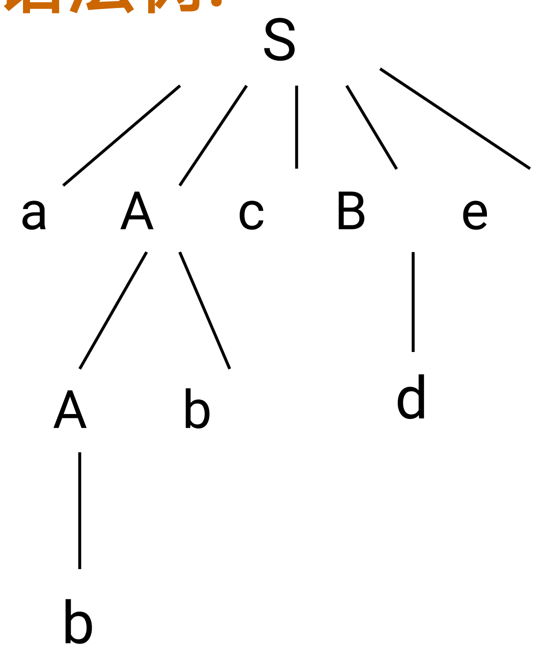
E-T

返回
返回章
返回节
退出

例2_5_4

已知文法G：（1）S→aAcBe
（2）A→b
（3）A→Ab
（4）B→d
给出句子abbcde的规范归约过程。

解: 语法树:



修剪语法树

规范归约过程

句型归约	归约产生式
abbcde	(2) A→b
aAbcde	(3) A→Ab
aAcde	(4) B→d
aAcBe	(1) S→aAcBe
S	

返回

返回章
返回节
退出