F5 上下文无关文法 2024

F5 上下文无关文法

- 介绍另一类语言，上下文无关语言（CFL），使用了我们熟悉的术语：句子、句型、语法、语法树等，这容易让人产生联想而不感到陌生。
- 在Chomsky体系中，上下文无关语言是比正则语言范围更大的一类语言。
- CFL有两种语言识别器，一种是上下文无关文法（CFG）类似于正则表达式但更强，另一种是堆栈自动机（PDA）类似于有穷自动机但也更强。

- 对应于教材第五章，我们从CFG开始，介绍与CFG有关的定义、推导、语法树、歧义性、CFG化简等概念。目的是为语法分析做必要准备。

5.1 文法如何定义语言

- 注意到一些非正则语言它们可能是上下文无关语言。
  - $L = \{0^n1^n \mid n \geq 1\}$
  - 回文$= \{w \in \{0, 1\} \mid w = w^R\}$
  - $L^{DUP} = \{ww \mid w \in L, L \subseteq \{0, 1\}^*\}$
  - ...
- 如何识别上下文无关语言？
  - 上下文无关文法定义CFL，是CFL的语言识别器。

定义语言的思路

- 例，给定$L = \{ 0^n1^n \mid n \geq 1\}$
- 对于什么样的串x属于L，有一个自然的递归定义：
  - 基础：01属于L;
  - 归纳：若w属于L，那么0w1属于L。

- 从上述定义中产生出两条规则：
  - $01 \in L$;
  - 若$w \in L$，那么$0w1 \in L$。
- 则L是满足这两条（约束）规则的串的集合。

- 形式化这种递归定义方式得到CFG。

变元指代语言

- 我们用变元A指代语言L，那么L(A)=L，借此建立构造L的若干规则。
- 例 借用变元定义$L=\{\,0^n1^n\,|\,n\geq 1\}$的两条规则：
  - $\mathrm{L}(A)\supseteq\{01\}$
  - $\mathrm{L}(A)\supseteq\{0\}\,\mathrm{L}(A)\{1\}$
- 在CFG中，上述规则被表示成产生式规则形如：
  - $A\rightarrow 01$
  - $A\rightarrow 0A1$
- 左部是单个变元，右部为一个符号串，变元也可以出现在产生式规则右部。

利用产生式规则产生语言

- 对于由产生式规则构成的CFG，
  - $A \rightarrow 01$
  - $A \rightarrow 0A1$

- A 被称为变元，0和1被称为终结符（除了变元以外的符号）

- 如果我们将变元A的语言记为L(A)，那么，
  - 根据第一个产生式规则知道 $01 \in L(A)$；
  - 根据第二个产生式规则，如果 $x \in L(A)$ 那么 $0x1 \in L(A)$。

- 利用归纳法得到 $L(A) = \{ 0^n 1^n \mid n \geq 1 \}$

CFG允许多个变元

- 当CFG中有多个变元时，通过产生式规则，一些变元的语言借助于另一些变元的语言得到定义。
- $B \rightarrow AA$
- $A \rightarrow 01$
- $A \rightarrow 0A1$

- 第一条规则解释为：若$x,y \in L(A)$，那么$xy \in L(B)$
- 容易证明：$L(B) = \{xy \mid x,y \in L(A)\}$，$L(A)=\ldots$

- CFG中有一个变元是文法开始符号，它的语言也就是这个文法所定义的语言。
- 因此，我们说CFG识别的语言是它的文法开始符号所指代的语言，这是上下文无关语言。

CFG的形式定义

- 至此我们得到CFG的关键元素：变元、终结符、产生式规则、文法开始符号。
- 形式定义：CFG $G = (V, T, \mathcal{P}, S)$，其中，
  - G为文法标识（可省略），
  - V是变元的有穷集合，
  - T是终结符的有穷集合（字母表，也习惯用Σ），
  - $\mathcal{P}$是产生式规则的有穷集合，
  - S是一个文法开始符号，$S \in V$。
- 文法G所定义的语言，$L(G) = L(S) \subseteq T^*$，是T上的语言。

CFG形式定义举例

- 语言$\{ 0^n1^n \mid n \geq 1\}$ 的CFG：
- $A \rightarrow 01$
- $A \rightarrow 0A1$
- 写成代数形式，CFG $G=(\{A\}, \{0,1\}, \{(A ,01), (A, 0A1)\}, A)$
- 有$L(G) = L(A) = \{ 0^n1^n \mid n \geq 1\}$

- 语言$\{ xy \mid x, y\in\{ 0^n1^n \mid n \geq 1\}\}$的CFG G'：
- $B \rightarrow AA$
- $A \rightarrow 01$
- $A \rightarrow 0A1$
- 代数形式：CFG $(\{B, A\}, \{0, 1\}, \{(B, AA), (A, 01), (A, 0A1)\}, B)$
- $L(G') = L(B) = \{ xy \mid x, y\in L(G)\}$

对产生式规则的解释

- CFG $G = (V, T, \mathcal{P}, S)$
- 对于$\mathcal{P}$中产生式规则

  $A \to \alpha$，

  其中$A\in V$，$\alpha\in(V\cup T)^*$，有：
- $L(A)\supseteq L(\alpha)$
- 如果$\alpha \in T^*$那么有$L(\alpha)=\{\alpha\}$;
- 如果$\alpha=\beta N\gamma$, $\beta,\gamma\in(V\cup T)^*$, $N\in V$，则$L(\alpha)=L(\beta)L(N)L(\gamma)$

- 再考察：
- $B \to AA$ $L(B) \supseteq L(A)L(A)$ 若$x,y\in L(A)$则$xy\in L(B)$
- $A \to 01$ $L(A) \supseteq L(01)$ 若$x\in L(01)$则$x\in L(A)$
- $A \to 0A1$ $L(A) \supseteq L(0)L(A)L(1)$ 若$x\in L(A)$则$0x1\in L(A)$

CFG形式定义（续）

- 对于任意CFG $G = (V, T, \mathcal{P}, S)$，有，
- $V \cap T = \varphi$
- 若$(A, \alpha) \in \mathcal{P}$, 那么$A \in V$, 并且$\alpha \in (V \cup T)^*$
- 一般直观地写成：$A \rightarrow \alpha$
- A是产生式左部，α是右部，为了方便α又称为A的候选式
- 如果A的全部候选式为$\alpha_1 \ldots \alpha_k$，$k>1$，那么可简写为，
  - $A \rightarrow \alpha_1 | \ldots | \alpha_k$
- $S \in V$（恰有一个变元是文法开始符号）
- $L(G) = L(S) \subseteq T^*$

符号使用习惯：

A，B，C，… 为变元

a，b，c，… 为终结符

…，X，Y，Z 为终结符或者变元

符号使用习惯：

…，w，x，y，z 只是终结符串

α，β，γ，… 终结符和变元组成的串

第一个产生式的左部变元为文法开始符号

例

- $E \longrightarrow E + E \mid E * E \mid (E) \mid d$
- 变元集合{E}
- 终结符集合{+, *, (,), d}
- 产生式规则集合，4个如上所示
- 文法开始符号E

- $L(E) \supseteq L(E)\ \{+\}\ L(E)$
- $L(E) \supseteq L(E)\ \{*\}\ L(E)$
- $L(E) \supseteq \{(\}\ L(E)\ \{)\}$
- $L(E) \supseteq \{d\}$

- CFG ({E}, {+, *, (,), d}, {(E, E+E), (E, E*E), (E, \(E\)), (E, d)}, E)

例

- Ď ⟶ ε | D; Ď
- D ⟶ T d
- T ⟶ int | float
- 回顾词法记号与词法单元概念
  - d, int, float和;构成文法的终结符集合。
- 用到的语言？
  - Ď, D, T构成文法的变元集合，其中Ď为文法开始符号。
- 代数表示为

  CFG ({Ď, D, T},

  {d, int, float, ;},

  {(Ď, ε),(Ď, D;Ď),(D, Td),(T, int),(T, float)},

  Ď)
- L(T)=?；L(D)=?；L(Ď)=?
  - RE：((int d + float d);)*；采用推导Ď ⇒ ...

推导

- 对于CFG $G = (V, T, \mathcal{P}, S)$，
- 符号串η推导δ写为，$\eta \Rightarrow \delta$，其中$\eta, \delta \in (V \cup T)^*$
- 基本思想是将产生式规则看作重写规则，如果串η中某个变元的一次出现被它的某个候选式替换，得到串δ，就说η推导δ
- 形式地，如果$\eta = \alpha A \beta$，$\delta = \alpha \gamma \beta$，$(A, \gamma) \in \mathcal{P}$，那么$\eta \Rightarrow \delta$

- 例：CFG $G = (\{A, B\}, \{0,1\}, \{(B, AA), (A, 01), (A, 0A1)\}, B)$
- $00A11 \Rightarrow 000111$
- $00A11 \Rightarrow 000A111$
- $A \Rightarrow 0A1 \Rightarrow 00A11 \Rightarrow 000A111 \Rightarrow 00001111$
- $B \Rightarrow AA \Rightarrow 0A1A \Rightarrow 0011A \Rightarrow 00110A1 \Rightarrow 00110011$

推导

- 这种推导可以一直进行下去，串中任何变元的一次出现都可以用它的某个候选式来替换，直到串中只剩下终结符为止。
- $B \Rightarrow AA \Rightarrow 0A1A \Rightarrow 0011A \Rightarrow 00110A1 \Rightarrow 00110011$
- $E \Rightarrow E+E \Rightarrow E+E*E \Rightarrow E+d*E \Rightarrow E+d*d \Rightarrow d+d*d$
- Ď ⇒ D;Ď ⇒ D;D;Ď ⇒ Td;D;Ď ⇒ Td;Td;Ď ⇒ Td;Td; ⇒ int d;Td; ⇒ int d;float d;

- 如果从文法开始符号开始这个推导过程，终将得到该文法定义的语言的每一个成员。

推导与直接推导

- 对于CFG $G=(V, T, \mathcal{P}, S)$，
- 如果$\eta=\alpha A\beta$，$\delta=\alpha\gamma\beta$，$(A, \gamma)\in\mathcal{P}$，$\alpha,\beta\in(V\cup T)^*$，那么$\eta\Rightarrow\delta$。
- $\Rightarrow$称为直接推导。

- 推导$\Rightarrow^*$是指“0到多个直接推导步骤”
- 基础：$\eta\Rightarrow^*\eta$，对任意$\eta\in(V\cup T)^*$
- 归纳：若 $\eta\Rightarrow^*\alpha$，$\alpha\Rightarrow\delta$，那么 $\eta\Rightarrow^*\delta$，对任意$\eta,\alpha,\delta\in(V\cup T)^*$

例子：推导与直接推导

- $S \rightarrow 01$; $S \rightarrow 0S1$。
  - $S \Rightarrow 0S1 \Rightarrow 00S11 \Rightarrow 000111$ 。
- 所以
  - $S \Rightarrow^* S$;
  - $S \Rightarrow^* 0S1$;
  - $S \Rightarrow^* 00S11$;
  - $S \Rightarrow^* 000111$。

- $B \Rightarrow AA \Rightarrow 0A1A \Rightarrow 0011A \Rightarrow 00110A1 \Rightarrow 00110011$
- $E \Rightarrow E+E \Rightarrow E+E*E \Rightarrow E+d*E \Rightarrow E+d*d \Rightarrow d+d*d$
- $\check{D} \Rightarrow D;\check{D} \Rightarrow D;D;\check{D} \Rightarrow Td;D;\check{D} \Rightarrow Td;Td;\check{D} \Rightarrow Td;Td; \Rightarrow$ int d;Td; $\Rightarrow$ int d;float d;

推导与直接推导的性质

- 推导$\Rightarrow^*$
  - 自反性
  - 传递性

- 直接推导$\Rightarrow$
  - 重写性质， 即，

对任意文法符号串α和β，$\alpha A\beta \Rightarrow \alpha\gamma\beta$当且仅当$(A, \gamma) \in \mathcal{P}$

归约与直接归约（递归推理）

- 从产生式的右部到左部来运用，一步一步地进行，并最终将给定的终结符号串转变为文法开始符号，或者因所有子串都不是候选式而结束。
- 在过程的每一步，当前符号串中的某个子串与某个候选式相同，则可以将这个子串用那个候选式的变元替代，从而当前符号串发生改变。

- 对于CFG G = (V, T, $\mathcal{P}$, S)，如果(A, γ)∈$\mathcal{P}$，α,β∈(V∪T)*，那么有αγβ⇐αAβ。⇐称为直接归约，是直接推导的逆过程。

- 连续归约⇐*是指“0到多个直接归约步骤”，定义为，
  - 基础：η⇐*η，对任意η∈(V∪T)*
  - 归纳: 若 η⇐*α，α⇐δ，那么η⇐*δ，对任意η,α,δ∈(V∪T)*

例子：归约与直接归约

- $S \rightarrow 01; S \rightarrow 0S1$。
  - $000111 \Leftarrow 00S11 \Leftarrow 0S1 \Leftarrow S$。
- 所以
  - $S \Leftarrow^* S$;
  - $0S1 \Leftarrow^* S$;
  - $00S11 \Leftarrow^* S$;
  - $000111 \Leftarrow^* S$ 。

推导、归约与语言

- 除了自反性，推导关系的传递性还包括以下内容：
  - $\alpha \Rightarrow^* \gamma$如果有$\beta$使得$\alpha \Rightarrow \beta$且$\beta \Rightarrow \gamma$
  - $\alpha \Rightarrow^* \gamma$如果有$\beta$使得$\alpha \Rightarrow^* \beta$且$\beta \Rightarrow \gamma$
  - $\alpha \Rightarrow^* \gamma$如果有$\beta$使得$\alpha \Rightarrow^* \beta$且$\beta \Rightarrow^* \gamma$

- 借助于推导关系的语言定义。
- 定义：给定CFG $G=(V, \Sigma, \mathcal{P}, S)$，
  该文法定义的**语言**$L(G) = \{w \in T^* \mid S \Rightarrow^* w\}$。

- 借助于归约关系的语言定义。
- 定义：给定CFG $G=(V,T,\mathcal{P},S)$，
  该文法定义的**语言**$L(G) = \{w \in T^* \mid w \Leftarrow^* S\}$。

例：文法生成符号串

- 示例：文法通过推导来生成符号串aabbbcc
- 替换策略：随机选取任意变元的任意一次出现替换之（推导）；固定选择最左（右）边的变元用它的某个候选式替换之（最左、最右推导）

$$
\begin{aligned}
T &\rightarrow R\\
T &\rightarrow \mathrm{a}T\mathrm{c}\\
R &\rightarrow \varepsilon\\
R &\rightarrow R\mathrm{b}R
\end{aligned}
$$

$$
\begin{aligned}
& \underline{T}\\
\Rightarrow\ & \mathrm{a}\underline{T}\mathrm{c}\\
\Rightarrow\ & \mathrm{aa}\underline{T}\mathrm{cc}\\
\Rightarrow\ & \mathrm{aa}\underline{R}\mathrm{cc}\\
\Rightarrow\ & \mathrm{aa}R\mathrm{b}\underline{R}\mathrm{cc}\\
\Rightarrow\ & \mathrm{aa}\underline{R}\mathrm{bcc}\\
\Rightarrow\ & \mathrm{aa}R\mathrm{b}\underline{R}\mathrm{bcc}\\
\Rightarrow\ & \mathrm{aa}R\mathrm{b}\underline{R}\mathrm{b}R\mathrm{bcc}\\
\Rightarrow\ & \mathrm{aa}\underline{R}\mathrm{bb}R\mathrm{bcc}\\
\Rightarrow\ & \mathrm{aabb}\underline{R}\mathrm{bcc}\\
\Rightarrow\ & \mathrm{aabbbcc}
\end{aligned}
$$

$$
\begin{aligned}
& \underline{T}\\
\Rightarrow\ & \mathrm{a}\underline{T}\mathrm{c}\\
\Rightarrow\ & \mathrm{aa}\underline{T}\mathrm{cc}\\
\Rightarrow\ & \mathrm{aa}\underline{R}\mathrm{cc}\\
\Rightarrow\ & \mathrm{aa}\underline{R}\mathrm{b}R\mathrm{cc}\\
\Rightarrow\ & \mathrm{aa}\underline{R}\mathrm{b}R\mathrm{b}R\mathrm{cc}\\
\Rightarrow\ & \mathrm{aab}\underline{R}\mathrm{b}R\mathrm{cc}\\
\Rightarrow\ & \mathrm{aab}\underline{R}\mathrm{b}R\mathrm{b}R\mathrm{cc}\\
\Rightarrow\ & \mathrm{aabb}\underline{R}\mathrm{b}R\mathrm{cc}\\
\Rightarrow\ & \mathrm{aabbb}\underline{R}\mathrm{cc}\\
\Rightarrow\ & \mathrm{aabbbcc}
\end{aligned}
$$

句型与句子

- 这里借用了自然语言的术语。
- 句型：从文法开始符号推导出来的任意串。
- 句子：从文法开始符号推导出来的终结符串。
- 句子是句型的一个特例。
- 形式地：
  - 对于文法$G=(V, T, \mathcal{P}, S)$，
  - α 是一个句型，当且仅当 $S \Rightarrow^* \alpha$
  - w是一个句子，当且仅当 $S \Rightarrow^* w$ 且 $w \in T^*$

- 上页例子中每一步推导得到的都是句型，只有最后一步是句子。
- 思考：CFG的语言是所有句子的集合。

最左（右）推导的符号表示

- 推导允许我们替换串中的任意变元，并允许使用它的任意候选式。
- 这种重写策略会导致同一个句型有多个推导存在。
- 最左（右）推导确定化了推导过程采用的重写策略。

- 最左推导：$\Rightarrow_{lm}$、$\Rightarrow^{*}_{lm}$
- 最右推导：$\Rightarrow_{rm}$、$\Rightarrow^{*}_{rm}$

例：平衡括号文法

- 平衡括号文法：

  $S \rightarrow SS \mid (S) \mid ()$
- $S \Rightarrow_{lm} SS \Rightarrow_{lm} (S)S \Rightarrow_{lm} (())S \Rightarrow_{lm} (())()$
- 因此，$S \Rightarrow^{*}_{lm} (())()$
- $S \Rightarrow SS \Rightarrow S() \Rightarrow (S)() \Rightarrow (())()$ 是一个推导，但不是最左推导

- $S \Rightarrow_{rm} SS \Rightarrow_{rm} S() \Rightarrow_{rm} (S)() \Rightarrow_{rm} (())()$
- 因此, $S \Rightarrow^{*}_{rm} (())()$
- $S \Rightarrow SS \Rightarrow SSS \Rightarrow S()S \Rightarrow ()()S \Rightarrow ()()()$ 既不是最左推导也不是最右推导

最左、最右推导

$E \rightarrow E+E \mid E*E \mid (E) \mid F$

$F \rightarrow aF \mid bF \mid 0F \mid 1F \mid \varepsilon$

问句子 a*(ab+10)的最左、最右推导？

E
$\Rightarrow_{lm} E*E$
$\Rightarrow_{lm} F*E$
$\Rightarrow_{lm} a*E$
$\Rightarrow_{lm} a*(E)$
$\Rightarrow_{lm} a*(E+E)$
$\Rightarrow_{lm} a*(F+E)$
$\Rightarrow_{lm} a*(aF+E)$
$\Rightarrow_{lm} a*(abF+E)$
$\Rightarrow_{lm} a*(ab+E)$
$\Rightarrow_{lm} a*(ab+F)$
$\Rightarrow_{lm} a*(ab+1F)$
$\Rightarrow_{lm} a*(ab+10F)$
$\Rightarrow_{lm} a*(ab+10)$

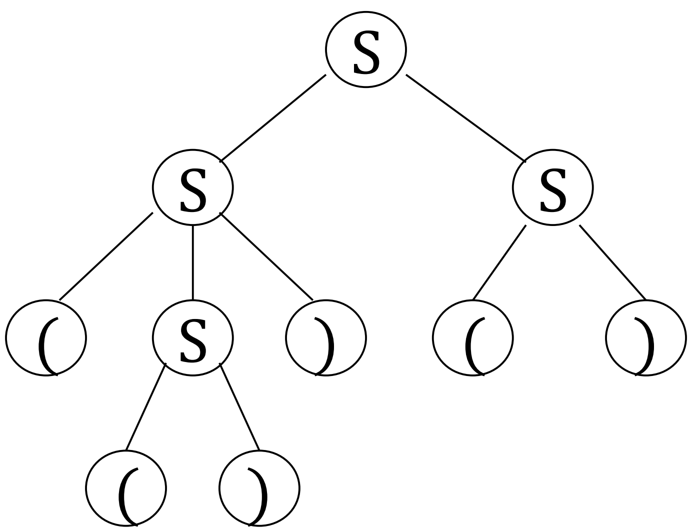
E
$\Rightarrow_{rm} E*E$
$\Rightarrow_{rm} E*(E)$
$\Rightarrow_{rm} E*(E+E)$
$\Rightarrow_{rm} E*(E+F)$
$\Rightarrow_{rm} E*(E+1F)$
$\Rightarrow_{rm} E*(E+10F)$
$\Rightarrow_{rm} E*(E+10)$
$\Rightarrow_{rm} E*(F+10)$
$\Rightarrow_{rm} E*(aF+10)$
$\Rightarrow_{rm} E*(abF+0)$
$\Rightarrow_{rm} E*(ab+10)$
$\Rightarrow_{rm} F*(ab+10)$
$\Rightarrow_{rm} aF*(ab+10)$
$\Rightarrow_{rm} a*(ab+10)$

5.2 语法分析树

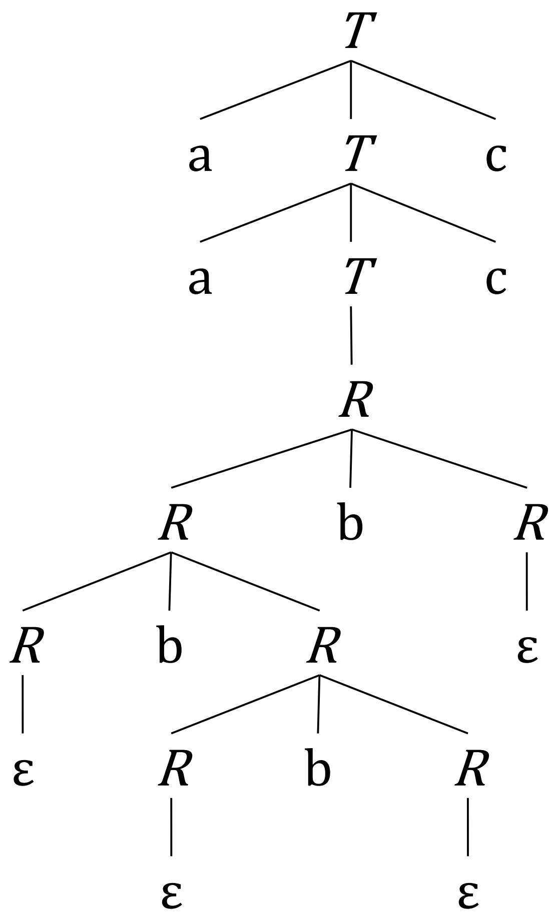
- 可以将推导写成树形表示。树根是文法开始符号。每一次直接推导所替换的变元，就是在树中给它加上孩子结点，即对于所用候选式，依序一个符号一个结点，这些结点都作为孩子。

对于文法：$S \rightarrow SS \mid (S) \mid ()$，

推导$S \Rightarrow SS \Rightarrow (S)S \Rightarrow (())S \Rightarrow (())()$的语法树



语法树示例



$$
\begin{aligned}
T &\rightarrow R \\
T &\rightarrow \mathrm{a}T\mathrm{c} \\
R &\rightarrow \varepsilon \\
R &\rightarrow R\mathrm{b}R
\end{aligned}
$$

$$
\begin{aligned}
& \underline{T} \\
\Rightarrow\ & \mathrm{a}\underline{T}\mathrm{c} \\
\Rightarrow\ & \mathrm{aa}\underline{T}\mathrm{cc} \\
\Rightarrow\ & \mathrm{aa}\underline{R}\mathrm{cc} \\
\Rightarrow\ & \mathrm{aa}R\mathrm{b}\underline{R}\mathrm{cc} \\
\Rightarrow\ & \mathrm{aa}\underline{R}\mathrm{bcc} \\
\Rightarrow\ & \mathrm{aa}R\mathrm{b}\underline{R}\mathrm{bcc} \\
\Rightarrow\ & \mathrm{aa}R\mathrm{b}\underline{R}\mathrm{b}R\mathrm{bcc} \\
\Rightarrow\ & \mathrm{aa}\underline{R}\mathrm{bb}R\mathrm{bcc} \\
\Rightarrow\ & \mathrm{aabb}\underline{R}\mathrm{bcc} \\
\Rightarrow\ & \mathrm{aabbbcc}
\end{aligned}
$$

$$
\begin{aligned}
& \underline{T} \\
\Rightarrow\ & \mathrm{a}\underline{T}\mathrm{c} \\
\Rightarrow\ & \mathrm{aa}\underline{T}\mathrm{cc} \\
\Rightarrow\ & \mathrm{aa}\underline{R}\mathrm{cc} \\
\Rightarrow\ & \mathrm{aa}\underline{R}\mathrm{b}R\mathrm{cc} \\
\Rightarrow\ & \mathrm{aa}\underline{R}\mathrm{b}R\mathrm{b}R\mathrm{cc} \\
\Rightarrow\ & \mathrm{aab}\underline{R}\mathrm{b}R\mathrm{cc} \\
\Rightarrow\ & \mathrm{aab}\underline{R}\mathrm{b}R\mathrm{b}R\mathrm{cc} \\
\Rightarrow\ & \mathrm{aabb}\underline{R}\mathrm{b}R\mathrm{cc} \\
\Rightarrow\ & \mathrm{aabbb}\underline{R}\mathrm{cc} \\
\Rightarrow\ & \mathrm{aabbbcc}
\end{aligned}
$$

语法分析树

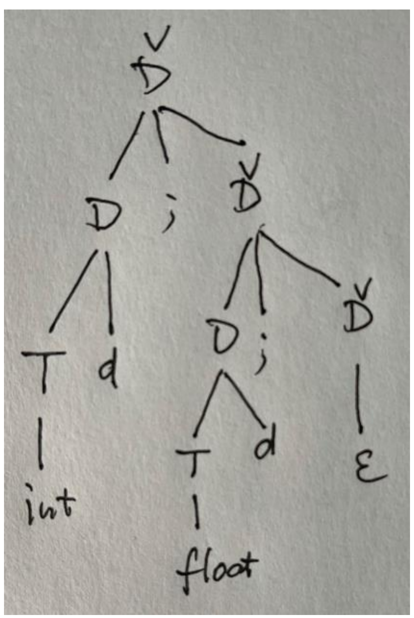
- 语法树的叶子是终结符或ε，将语法树的叶子从左往右依序连接成串，该串是句子。

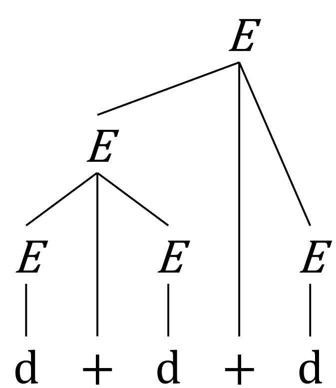
- 结论：CFL的句子都有自己的语法树，是有序树。
  - 叶子：用终结符或ε标记
  - 内结点：用变元标记
  - 内结点的孩子对应变元的候选式（从左往右组成串）
  - 根：用文法开始符号标记

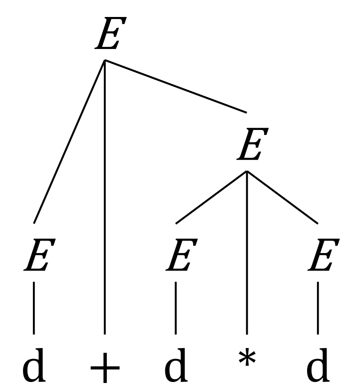
- 若允许语法树的叶子还可以是变元的话，我们就能得到句型的语法树。句型的语法树的叶子是变元、终结符或ε。

语法树的产物

- 句子的语法树的产物就是该句子（每个句子都有它的语法树，它的语法树的产物就是它）。
- 句型的语法树的产物就是该句型。
- 语法树给它的产物增加了结构信息，将为编译所使用。
- Ď ⇒ D;Ď ⇒ D;D;Ď ⇒ Td;D;Ď ⇒ Td;Td;Ď ⇒ Td;Td; ⇒ int d;Td; ⇒ int d;float d;







与CFG判定性质有关的等价性

- CFG G=(V, T, P, S)的判定过程的等价性。

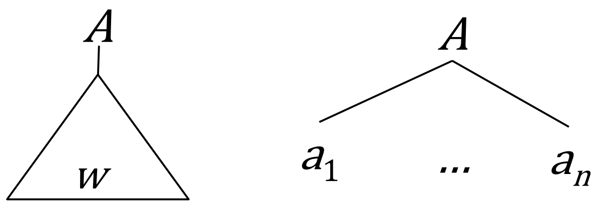
- 结论：对于任意A∈V，w∈T*，以下彼此是等价的。

1. （归约）$w \Leftarrow^{*} A$ 根据归约知道w∈L(A)。
2. （推导）$A \Rightarrow^{*} w$ 根据推导知道w∈L(A)。
3. （最左推导）$A \Rightarrow^{*}_{lm} w$ 根据最左推导知道w∈L(A)。
4. （最右推导）$A \Rightarrow^{*}_{rm} w$ 根据最右推导知道w∈L(A)。
5. （语法树）存在根结点为A，产物为w的语法树。

从判定性1（归约）到5（语法树）

- 定理5.1 设文法G = (V, T, P, S)是一个CFG，对任意终结符串w，如果w归约为A，则一定存在一颗以A为根、产物为w的语法分析树。
- 证明，对归约步数进行归纳。
- 基础：若该归约步数只有一步，那么是直接归约，即存在产生式规则A→w，w⇐*A。那么有一颗以w个符号标记的叶子，以A标记的根，组成树，即是w的语法树，产物为w。

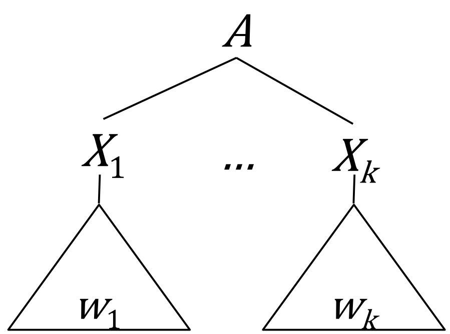
A→w , w⇐*A



从归约到语法树

- 归纳：假设对于步数小于n的归约$w \Leftarrow^* A$，都存在以A为根以w为产物的语法树。那么归约步数为n时，观察最后一步归约，即归约为A，一定使用了A的产生式，不失一般性假设为$A \to X_1 \ldots X_k$，其中$X_i \in V$或者$X_i \in T$，对于$i=1,\ldots,k$。那么w一定可以分成k段，即$w=w_1 \ldots w_k$，并且$w_i \Leftarrow^* X_i$，对于$i=1,\ldots,k$。因此存在根为A，A的孩子为$X_1 \ldots X_k$，产物为w的语法树。

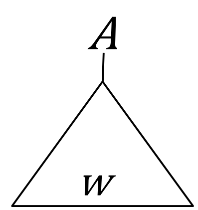
$A \to X_1 \ldots X_k$
$X_1 \ldots X_k \Leftarrow^* A$
$w=w_1 \ldots w_k$
$w_i \Leftarrow^* X_i$，$i=1,\ldots,k$



从判定性5（语法树）到3（推导）

- 定理5.2。设文法$G = (V, T, \mathcal{P}, S)$是一个CFG，若有一个语法树它的根标记为变元A并且产物是w，$w \in T^*$，那么一定存在一个G中的最左推导 $A \Rightarrow^*_{lm} w$。
- 证明，对语法树的高度进行归纳。
- 基础：树高为1，那么树的叶子只能是终结符，树根为A。那么树的叶子从左往右连接起来为w。由于这是一个语法树，所以$A \rightarrow w$一定是$\mathcal{P}$中产生式，因此有最左推导$A \Rightarrow^*_{lm} w$。

树高为1



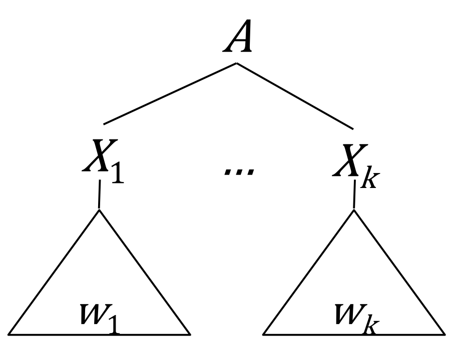
有$A \rightarrow w$，

$A \Rightarrow^*_{lm} w$

从语法树到推导

- 归纳：假设对于高度小于n的语法树，其中根为X产物为x，都存在$X \Rightarrow^*_{lm} x$。那么，当树高为n时，我们总可以把它分解为树根及其孩子，以及每个以孩子为根的子树。不失一般性，假设树根为A，孩子为$X_1 \ldots X_k$，由于这是语法树，所以必定有产生式$A \rightarrow X_1 \ldots X_k$，故$A \Rightarrow^*_{lm} X_1 \ldots X_k$。另每个以A的孩子为根的子树的高度都小于n，根据归纳假设，它们对应有最左推导$X_i \Rightarrow^* w_i$，对于i=1,…,k。其中$w=w_1 \ldots w_k$。
- 因此, $A \Rightarrow_{lm} X_1 \ldots X_k \Rightarrow^*_{lm} w_1 X_2 \ldots X_k \Rightarrow^*_{lm} w_1 w_2 X_3 \ldots X_k \Rightarrow^*_{lm} \ldots \Rightarrow^*_{lm} w_1 \ldots w_k$

子树高≥0



$A \rightarrow X_1 \ldots X_k$

$X_1 \ldots X_k \Leftarrow^* A$

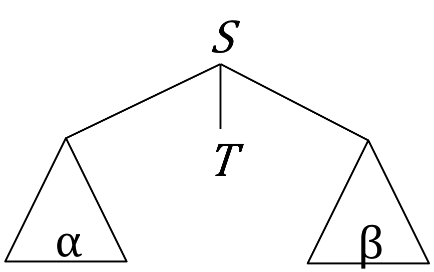
$w=w_1 \ldots w_k$

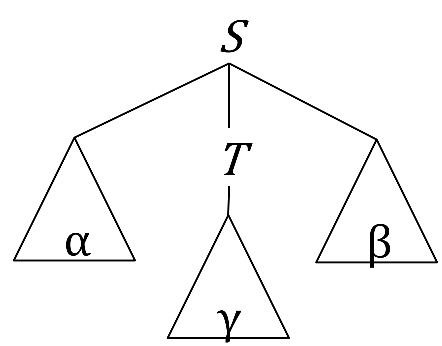
$w_i \Leftarrow^* X_i$，i=1,…,k

推导步骤与扩展语法树步骤——对应

- 在过程步骤上，推导一一对应于扩展语法树。
- 从根S扩展产物为w的语法树，初始时S为可扩展结点，
  - 对于当前可扩展结点，用它的候选式作为孩子结点；
  - 若当前结点为未扩展过的变元则它为可扩展结点；
  - 扩展语法树直到没有可扩展结点时结束。
  - 当语法树与推导策略无关时，给定推导得到语法树扩展的对应关系

$$\alpha T\beta \Rightarrow \alpha\gamma\beta \text{ 如果有产生式 } T \to \gamma$$





归约步骤与构建语法树步骤——对应

- w 归约为S，对应语法树根S和产物w
- 符号串$w \in T^*$的归约是一个序列$\alpha_1, \ldots, \alpha_n$, $n>1$。其中$\alpha_1 = w$，$\alpha_n = S$，$\alpha_{i+1} \Rightarrow \alpha_i$, $1 \le i < n$。记为$w \Leftarrow^* \alpha_n$。

- 语法树的生成过程：
- 从根向叶子方向扩展语法树：对应推导。
- 从叶子往根方向构建语法树：对应归约。

推导、归约与语法树对应关系

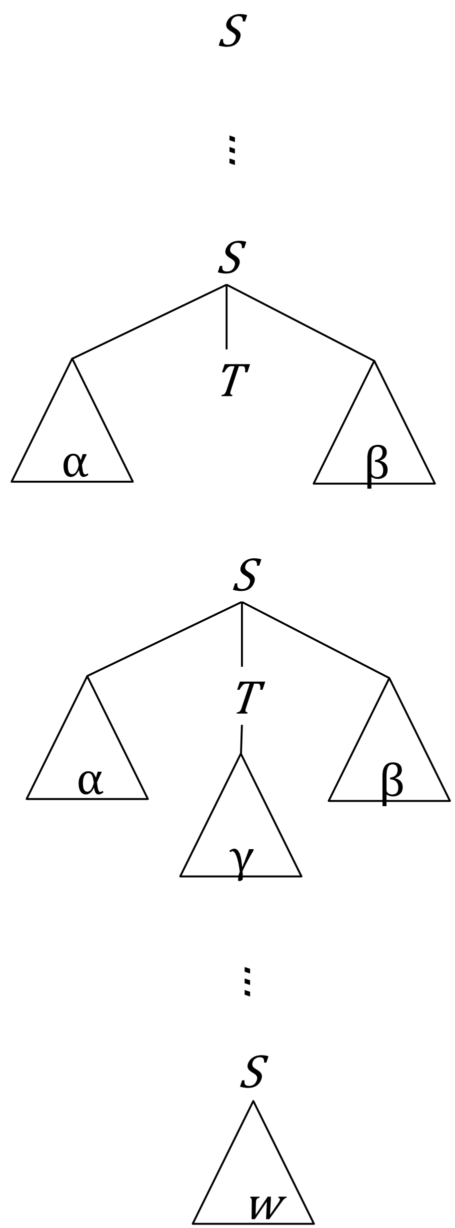
$S \Rightarrow^* w$

$S \Rightarrow^* \alpha T \beta$

$\Rightarrow \alpha\gamma\beta$

$\Rightarrow^* w$

自上而下地扩展



$w \Leftarrow^* S$

$w \Leftarrow^* \alpha\gamma\beta$

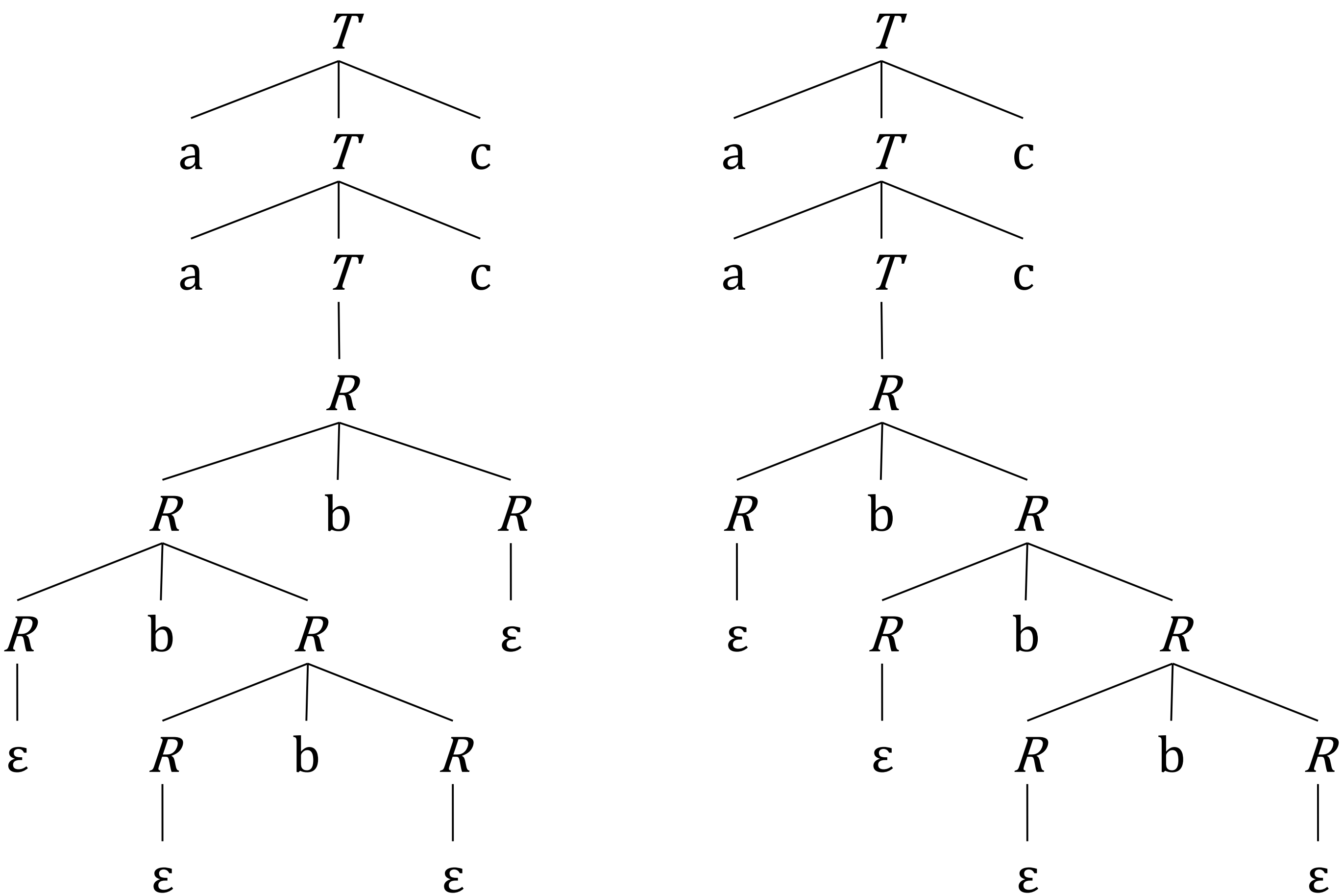
$\Leftarrow \alpha T \beta$

$\Leftarrow^* S$

自下而上地构建

5.4 歧义性文法

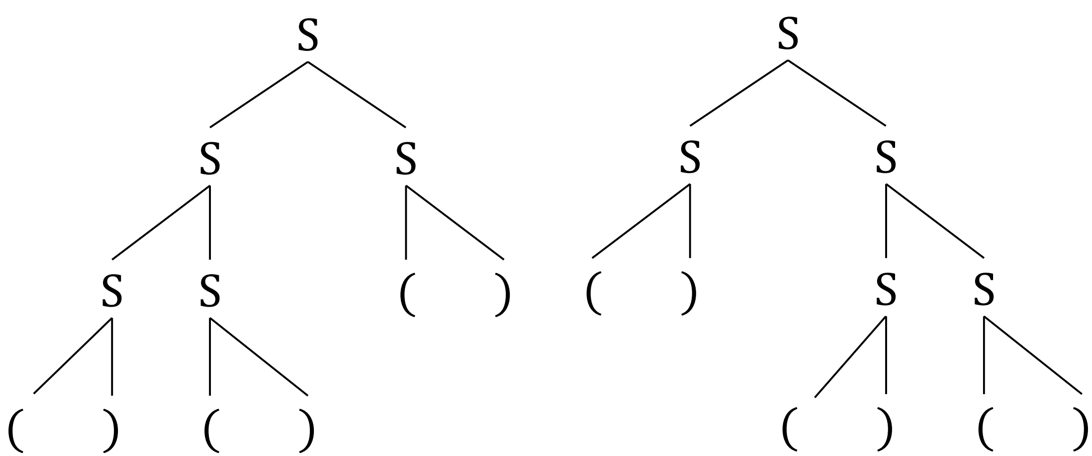
- 选择不同的候选式常常会推导出不同的句子，但有时不尽然。如果真出现同一个句子有多个语法树的情况，表明文法是有歧义的。



$$T \rightarrow R$$
$$T \rightarrow aTc$$
$$R \rightarrow \varepsilon$$
$$R \rightarrow RbR$$

歧义性文法

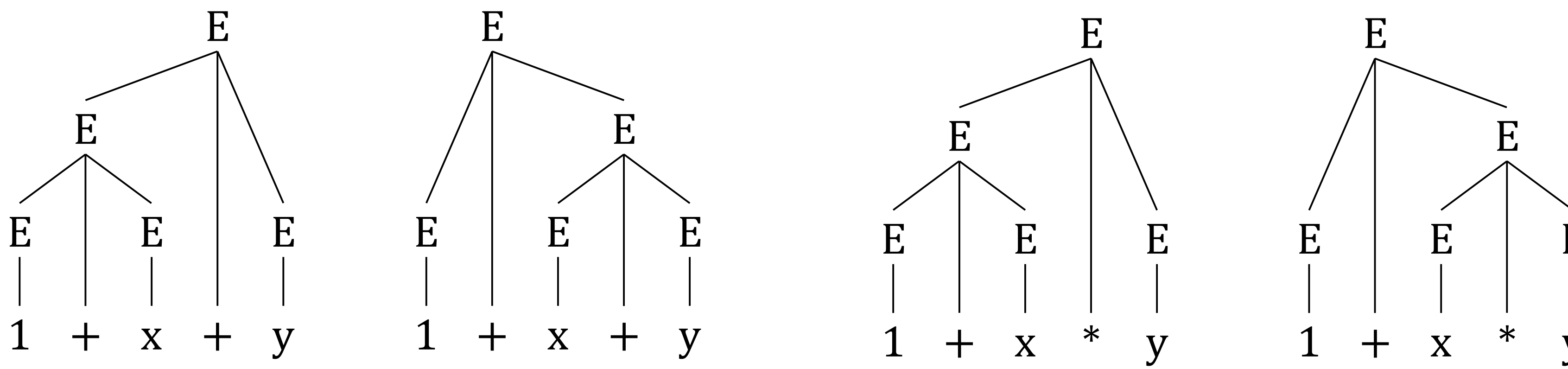
- 一个 CFG 是歧义的 若它的语言有元素至少是两个语法树的产物。
- 例：平衡括号语言 $S \rightarrow SS \mid (S) \mid ()$
- 容易找到句子它有两个语法树



(a) (b)

歧义性文法

- $E \rightarrow E + E \mid E * E \mid (E) \mid d \mid i$



(a) (b) (d) (e)

歧义性与最左/最右推导

- 定理5.5 对于CFG的句子w有两个语法树当且仅当有两个最左推导。
- 如果w有两个语法树，根据定理证明中的构造方法必定存在两个不同的最左推导。
- 相反地, 根据证明中的另一部分，两个不同的最左推导产生不同的语法树。
- 最右推导类似。

歧义性的来源

- 2+3*4
  ① 2+(3*4)
  ② (2+4)*4
- 操作符的优先级
- 2+3+4
  ① (2+3)+4
  ② 2+(3+4)
- 中缀操作符的结合性
- if e_1 then if e_2 then s_1 else s_2
  ① if e_1 then {if e_2 then s_1 else s_2}
  ② if e_1 then {if e_2 then s_1} else s_2

歧义性是文法的性质

- 歧义性是文法的性质不是语言的性质
- 对于平衡括号语言，有另一个CFG，这个没有歧义性。

$$B \rightarrow (RB \mid \varepsilon$$

$$R \rightarrow \,) \mid (RR$$

  - B 文法开始符号，从它推导出平衡括号串。
  - R 生成右括号比左括号多一个的串。

例：把歧义文法转换为无歧义文法

- 歧义文法：
  - $B \rightarrow BB \mid (B) \mid ()$
- 无歧义文法：
  - $B \rightarrow (RB \mid \varepsilon$
  - $R \rightarrow) \mid (RR$
- 对于给定的平衡的括号的串通过从左到右扫描该串，构造出唯一最左推导。
- 如果我们需要扩展B：
  - 若下一个符号是“(”使用$B \rightarrow (RB$扩展
  - 若输入串为空使用ε扩展
- 如果我们需要扩展R：
  - 若下一个符号是“)”使用$R \rightarrow)$ 扩展
  - 若下一个符号是“(”使用$R \rightarrow (RR$ 扩展

分析过程

最左推导步骤:

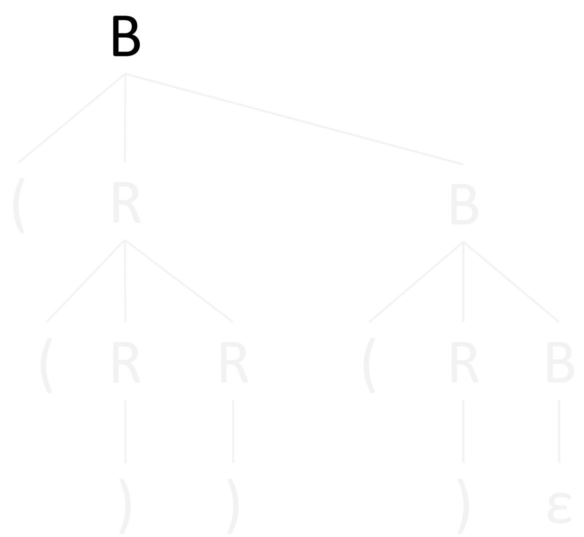
B

剩余输入串：

(( )) ( )

^

当前输入符号



$$B \rightarrow (RB \mid \varepsilon \qquad R \rightarrow) \mid (RR$$

The Parsing Process

剩余输入串：

( )) ( )

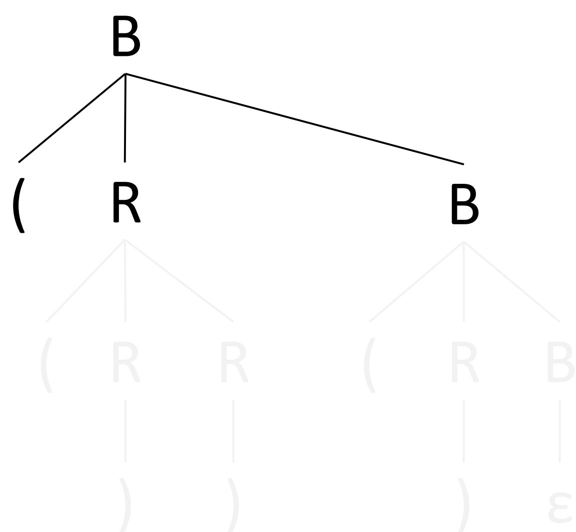
^

当前输入符号

最左推导步骤:

B

(RB



$B \rightarrow (RB \mid \varepsilon \qquad R \rightarrow) \mid (RR$

The Parsing Process

最左推导步骤:

B

(RB

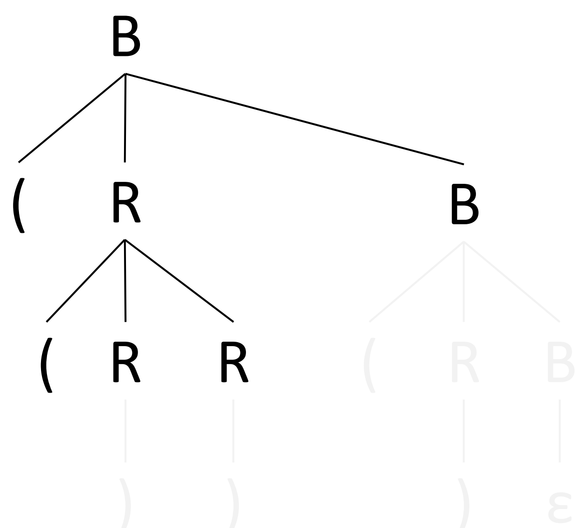
((RRB

剩余输入串：

)) ()

^

当前输入符号



$B \rightarrow (RB \mid \varepsilon \qquad R \rightarrow)\mid (RR$

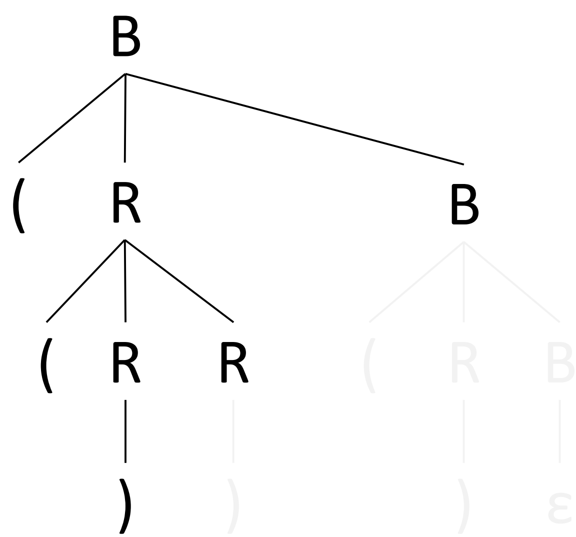
The Parsing Process

剩余输入串：

) ( )

^

当前输入符号



最左推导步骤:

B

(RB

((RRB

(()RB

$B \rightarrow (RB \mid \varepsilon \qquad R \rightarrow) \mid (RR$

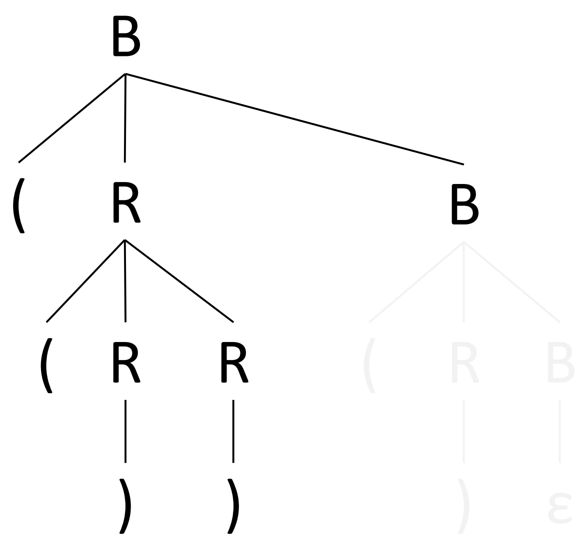
The Parsing Process

剩余输入串：

( )

^

当前输入符号



最左推导步骤:

B

(RB

((RRB

(( ) RB

(( )) B

$B \rightarrow (RB \mid \varepsilon \qquad R \rightarrow) \mid (RR$

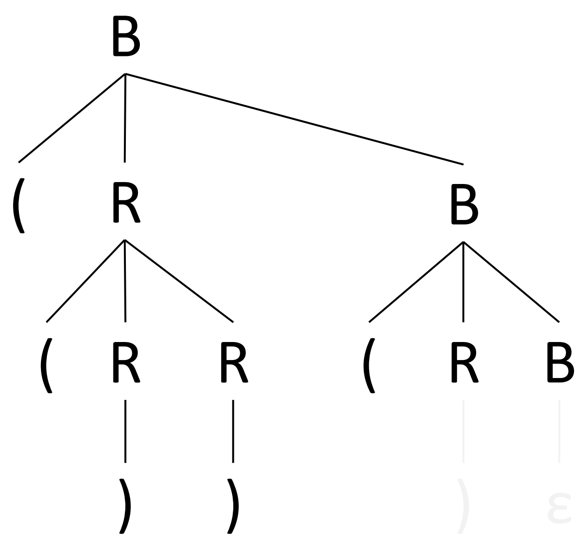
The Parsing Process

剩余输入串：

)

^

当前输入符号



最左推导步骤:

B

(RB

((RRB

(( ) RB

(( )) B

(( )) (RB

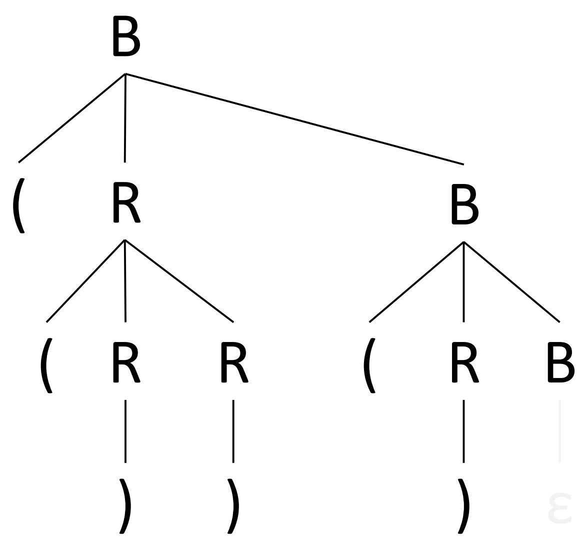
$B \rightarrow (RB \mid \varepsilon \qquad R \rightarrow) \mid (RR$

The Parsing Process

剩余输入串：

^

当前输入符号



最左推导步骤:

B

(RB

((RRB

(()RB

(())B

(())(RB

(())()B

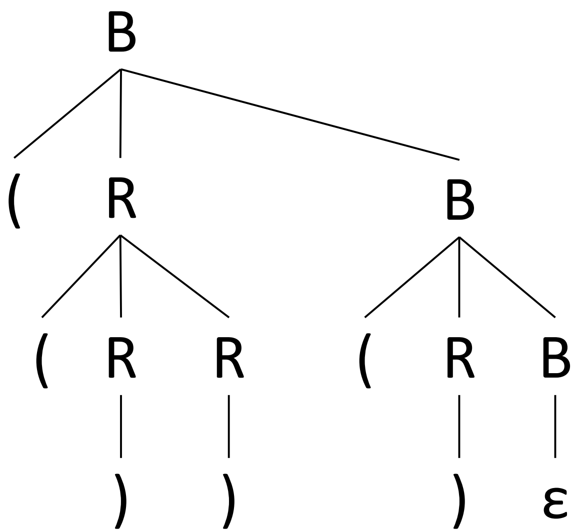
$$B \rightarrow (RB \mid \varepsilon \qquad R \rightarrow \,) \mid (RR$$

The Parsing Process

剩余输入串：

^

当前
输入
符号



最左推导步骤:

B

(RB

((RRB

(( ) RB

(( )) B

(( )) (RB

(( )) ( ) B

(( )) ( )

$B \rightarrow (RB \mid \varepsilon \qquad R \rightarrow) \mid (RR$

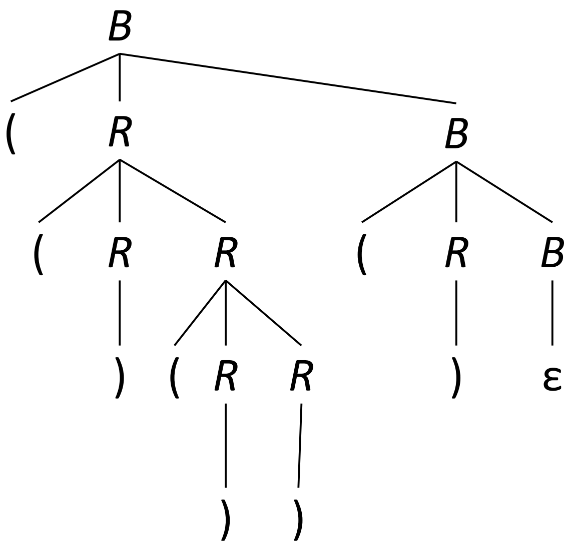
平衡括号文法的另一无歧义文法

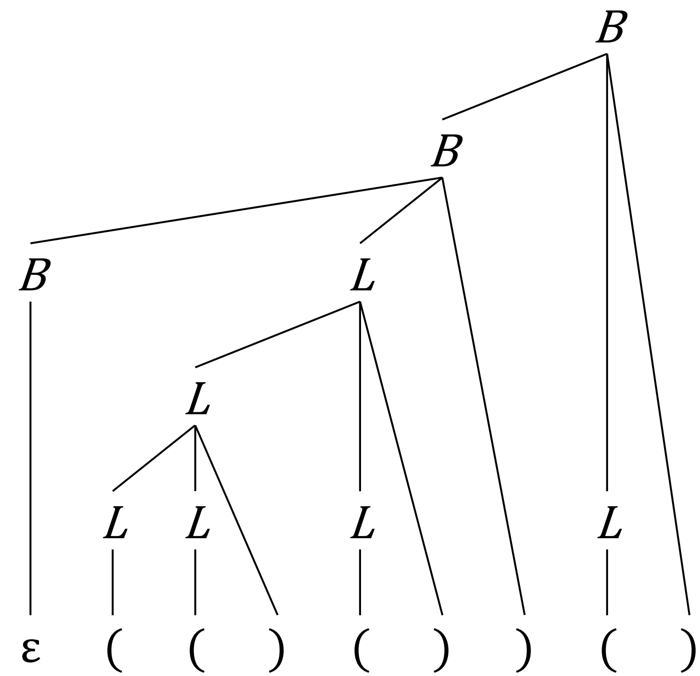
$B \rightarrow (RB \mid \varepsilon$

$R \rightarrow) \mid (RR$

$B \rightarrow BL) \mid \varepsilon$

$L \rightarrow (\mid LL)$





固有的歧义性

- 消除歧义性：通过分析产生歧义性的原因，针对性地修剪文法得到无歧义性文法。
- 分析定义CFL的多个文法：
  ① 都是无歧义的
  ② 有一些是无歧义的，其余都是歧义的
  ③ 都是歧义的
- 选择无歧义的文法来进行CFL的语法分析，但不幸的是，有一些CFL是固有的歧义的，即对它不存在无歧义文法。
- 歧义性文法是语法分析需要考虑的情形。

Example:固有的歧义性

- 语言$\{0^i1^j2^k \mid i = j \text{ or } j = k\}$ 是固有的歧义的。
- 直觉地，至少一些串形如 $0^n1^n2^n$ 必定是由两个不同语法树产生的，一个基于对0和1的个数进行检测，而另一个基于对1和2的个数进行检测。

- -是左结合+是右结合的表达式2-3+4（同一优先级运算符的结合性不一致）

小结及作业

- CFG，变元与终结符
- 产生式左部与右部、候选式，符号串替换
- 推导、直接推导、最左（右）推导，归约
- 语言、句子、句型、语法树、产物
- 推导与语法树扩展对应，归约与语法树构建对应
- 歧义性文法
- 消除文法歧义性（运算符优先级原因）

- 作业：p116：习题5.1a&b; 5.4; 5.5; 5.6; 5.8

中缀运算符$\oplus$的结合性

- $\oplus$是左结合的，如果a $\oplus$ b $\oplus$ c=(a $\oplus$ b) $\oplus$ c
- $\oplus$是右结合的，如果a $\oplus$ b $\oplus$ c=a $\oplus$ (b $\oplus$ c)
- $\oplus$是无结合性的，如果a $\oplus$ b $\oplus$ c则为非法
- 数学上，-和/是左结合的，+和-是结合的，左右都可以
- 计算上，因为满足结合律的运算符它的左、右结合性表现在精度、是否溢出等方面会有不同，所以左、右结合性只能二者选一。
- 一些程序设计语言如C选则左结合的+和*，这样就与数学上的-和/的左结合性保持一致，否则就会顶牛，对消除歧义性没有帮助。
- 但是C中的赋值操作是右结合的，a=b=c被解释为a=(b=c)
- 另一些程序设计语言如SML有一些右结合操作
- PASCAL语言的<和>都是无结合性的。即a<b<c出错

消除文法歧义性

- $E \rightarrow E \oplus E \mid n$
- $\oplus$是左结合的,
  - $E \rightarrow E \oplus E' \mid E'$
  - $E' \rightarrow n$
- $\oplus$是右结合的,
  - $E \rightarrow E' \oplus E \mid E'$
  - $E' \rightarrow n$
- 扩展到多个运算符时保持结合性一致
  - $E \rightarrow E + E' \mid E - E' \mid E'$
  - $E' \rightarrow n$

解决不同优先级操作符带来的歧义性

- 每个优先级对应一个非终结符
- 如果一个表达式使用了某优先级的运算符，那么它的子表达式就不能使用较低优先级的运算符（除非出现在括号里）。
- 因此一个优先级的变元的候选式中不能出现那些对应于比它优先级低的变元（除非出现在括号里）。
- $E \rightarrow E + T \mid T$
- $T \rightarrow T * F \mid F$
- $F \rightarrow (E) \mid n \mid i$

一个歧义文法

$S \rightarrow AB \mid CD$
$A \rightarrow 0A1 \mid 01$
$B \rightarrow 2B \mid 2$
$C \rightarrow 0C \mid 0$
$D \rightarrow 1D2 \mid 12$

A 产生相等个数0和1

B 产生任意个数的 2

C 产生任意数目的 0

D 产生相等个数1和2

每个有相等数目的0, 1, 和2的串都有两个最左推导.如:

$S \Rightarrow AB \Rightarrow 01B \Rightarrow 012$

$S \Rightarrow CD \Rightarrow 0D \Rightarrow 012$

5.5 CFG化简

- 对文法进行哪方面的化简？

  F7.1.1 去除无用符号

  F7.1.2 去除ε-产生式

  F7.1.3 去除单位产生式

5.5.1 有用与无用的文法符号

- 从文法开始符号推导句子的步骤中出现的符号都是有用的。
- 给定CFG (V, T, P, S)，如果有$S \Rightarrow^* \alpha X\beta \Rightarrow^* w$, $\alpha,\beta\in(V\cup T)^*$, $w\in T^*$，那么X是有用符号并且$X\in(V\cup T)$。
- 一个文法符号不是有用的便是无用的。

- 可达符号：由文法开始符号推导出的任意串中的文法符号。
- 如果$S \Rightarrow^* \alpha X\beta$, $\alpha,\beta\in(V\cup T)^*$，那么X是可达的。
- 有产出的符号：能推导出终结符串的文法符号。
- 如果$X\Rightarrow^*w$, $X\in(V\cup T)^*$, $w\in T^*$，那么X是产出的。

- 例：终结符，一定是产出的，但有可能是不可达的；
变元，4种情形都有可能。

例：有用、无用的文法符号

- 若$S \Rightarrow^* \alpha X\beta$，则X是可达的。
- 若$X \Rightarrow^* w$，则X是产出的。

- 例：$S \longrightarrow AB$；$A \longrightarrow aA \mid a$；$B \longrightarrow AB$
- 可达符号：S，A，B，a
- 不可达符号：无
- 有产出的符号：A，a
- 无产出符号：S，B
- 因为文法开始符号S是无产出的，所以该语言是空

去除无用符号

- 对于CFG $G=(V, T, P, S)$，$L(G) \neq \varphi$
- 从G中去除无产出变元得到CFG $G_g=(V_g, T, P_g, S)$，其中：
  V_g是V中去掉无产出成员所得；
  P_g是P中去掉含有无产出变元的成员。
- 从G_g中去除不可达符号得到CFG $G_u=(V_u, T_u, P_u, S)$，其中：
  T_u是T中去掉不可达成员；V_u是V_g中去掉不可达成员；
  P_u是P_g去掉含有不可达符号的成员。

- 例G：$S \longrightarrow AB \mid a$；$A \longrightarrow b$
  - V_g=？； P_g=？
- 则G_g：$S \longrightarrow a$；$A \longrightarrow b$
- 那么G_u：$S \longrightarrow a$
- 对于G'：$S \longrightarrow AB \mid B$；$A \longrightarrow b$ 会怎样？ V_g中不能没有S！

例：去除无用符号

- 从文法中先去除无产出变元，进而去除不可达符号。
- 例G：$S \longrightarrow AB \mid C$；$A \longrightarrow aA \mid a$；$B \longrightarrow bB$，$C \longrightarrow c$
  - $V=\{S,A,B,C\}$；$T=\{a,b,c\}$；P；S
- G_g：$S \longrightarrow C$；$A \longrightarrow aA \mid a$；$C \longrightarrow c$
  - $V_g=\{S,A,C\}$；$T=\{a,c\}$；P_g；S
- G_u：$S \longrightarrow C$；$C \longrightarrow c$
  - $V_u=\{S,C\}$；$T_u=\{c\}$；P_u；S
- 思考：从G中去除不可达符号，进而去除无产出变元，如何？

定理证明

- 定理5.6 对于CFG $G = (V, T, P, S)$，$L(G) \neq \varphi$，从G中去除无产出变元得到CFG $G_g = (V, T_g, P_g, S)$，从G_g中去除不可达符号得到CFG $G_u = (T_u, V_u, P_u, S)$，则G_u中没有无用符号，且$L(G_u)=L(G)$。

- 证明：(1) G_u中没有无用符号。
- 对于每一个$X \in T \cup V_g$，则X也属于$T \cup V$，如果$X \Rightarrow^*_G w$，$w \in T^*$，那么这个推导中所有符号都是产出的，所以$X \Rightarrow^*_{Gg} w$。
- 对于每一个$X \in T \cup V_g$，如果$S \Rightarrow^*_{Gg} \alpha X \beta$, $\alpha, \beta \in (V_g \cup T)^*$，那么这个推导中所有符号都是可达的，所以有$S \Rightarrow^*_{Gu} \alpha X \beta$, 并且$X \in V_u \cup T_u$，$\alpha, \beta \in (V_u \cup T_u)^*$。

- (2) $L(G_u)=L(G)$

定理证明（续）

- 定理5.6 对于CFG $G=(V, T, P, S)$，$L(G)\neq\varphi$，从G中去除无产出变元得到CFG $G_g=(V, T_g, P_g, S)$，从G_g中去除不可达符号得到CFG $G_u=(T_u, V_u, P_u, S)$，则G_u中没有无用符号，且$L(G_u)=L(G)$。

- 证明：(2) $L(G_u)=L(G)$
- 由于$T_u\subseteq T$，$V_u\subseteq V$，并且 $P_u\subseteq P$，所以$L(G_u)\subseteq L(G)$。
- 又由于$S\Rightarrow^*_G w, w\in T^*$，中所有出现的符号都是有用的，所以$S\Rightarrow^*_{Gu} w$，$w\in T_u{}^*$因此$L(G)\subseteq L(G_u)$。

发现有产出变元的算法

- 若$X \Rightarrow^* w$，则X是产出的。

- 根据CFG的产生式判断变元A是否是产出的：
  - 基础：若存在产生式$A \longrightarrow w$，其中w不含变元，那么A是产出的。
  - 归纳：若存在产生式$A \longrightarrow \alpha$，其中α仅由终结符和有产出变元组成，那么A是产出的。

- 证明：根据推导$X \Rightarrow^* w$的步数归纳，0步成立；假定n-1步成立；那么n步时，$X \Rightarrow \alpha \Rightarrow^* w$中第一步对应产生式$A \longrightarrow \alpha$，且$\alpha$中每个符号都推导出w的一个子串，并且推导步数小于n，那么这些符号都是产出的，因此算法能发现A是产出的。

例：发现有产出变元

- $S \longrightarrow AB \mid C$，$A \longrightarrow aA \mid a$，$B \longrightarrow bB$，$C \longrightarrow c$

- 基础：A 和C 可以被确定有产出，因为$A \longrightarrow a$ 和$C \longrightarrow c$。
- 归纳：S 可被确定有产出，因为$S \longrightarrow C$。
- 没有别的可以被发现的了。

发现可达符号的算法

- 若$S \Rightarrow^* \alpha X\beta$，则X是可达的。

- 根据CFG的产生式判断某变元A是否是可达的：
  - 基础：文法开始符号S可达。
  - 归纳：如果A可达，并且存在产生式$A \longrightarrow \alpha$，那么所有α中的符号均可达。

- 证明：对从S开始推导的步数归纳。0步成立；设小于n步成立；那么n步时如果有$S \Rightarrow^* \alpha A\beta \Rightarrow \alpha\gamma\beta$，其中$A \longrightarrow \gamma$。除最后一步直接推导，前面的步数小于n，所以A是可以被发现的，那么γ中的符号也能被发现。

例：发现可达符号

- 例：$S \longrightarrow AB$；$A \longrightarrow C \mid \varepsilon$；$C \longrightarrow c$；$B \longrightarrow bB$；$D \longrightarrow \varepsilon$
- 基础：S被发现，因为文法开始符号是可达的。
- 归纳：A和B被发现，因为有产生式$S \longrightarrow AB$，即有$S \Rightarrow^* AB$；
- 接着C被发现，因为$A \longrightarrow C$；
- b被发现因为$B \longrightarrow bB$；
- 接着c被发现因为$C \longrightarrow c$；
- 再没有别的可发现的了。

- 思考：ε是不是可达的？

5.5.2 去除ε-产生式

- ε-产生式A⟶ε是有用的但不是必须的。如果语言中不考虑空串的话我们定义它时就不需要ε-产生式了，或者含有ε-产生式的CFG可以等价转化为无ε-产生式的CFG。
- 发现可空变元。
- 可空变元在候选式中不出现的话，事实上就反映了它推导ε这种情况，如此就已经起到了ε-产生式的作用。
- 去除ε-产生式。

- 定义：CFG (V,T,P,S)，A∈V是可空变元，当且仅当$A \Rightarrow^* \varepsilon$。

- 例：S⟶ AcB, A⟶ aA | ε, B⟶ bB | A
- 可空变元：A； B

发现可空变元的算法

- 定义：CFG G=(V,T,P,S)，$A \in V$是可空变元，当且仅当$A \Rightarrow^* \varepsilon$。

- 基础：若$A \longrightarrow \varepsilon$是G的产生式，那么A 可空。
- 归纳：若G有产生式$A \longrightarrow C_1 \ldots C_k$, 其中$C_1,\ldots,C_k \in V$, $k>0$都是可空的，那么A 可空。

- 例：$S \longrightarrow AB$, $A \longrightarrow aA \mid \varepsilon$, $B \longrightarrow bB \mid A$
- 基础：A 可空因为$A \longrightarrow \varepsilon$。
- 归纳：B 可空因为$B \longrightarrow A$;
- 再者，S 可空因为$S \longrightarrow AB$。

定理证明

- 定理5.7 在任何文法G中，上面的算法生成的恰好是全部的可空符号。
- 变元A 被算法找出当且仅当 $A \Rightarrow^* \varepsilon$
- 证明：根据推导$A \Rightarrow^* \varepsilon$的步数归纳，1步成立即算法发现A可空；假定小于n步成立；那么n步时，$A \Rightarrow C_1 \ldots C_k \Rightarrow^* \varepsilon$中每个$C_i$小于n步推导出ε，根据归纳假设$C_i$可空，因此A可空。
- 若算法发现A可空，那么$A \Rightarrow^* \varepsilon$，证明对应算法中归纳过程（留作练习）。

去除ε-产生式

- 变形产生式：若产生式$A\longrightarrow\alpha$ 的候选式α中有m个符号是可空变元，$m\geq 0$，那么这m个变元的每一个出现和不出现，就形成了α的2^m个变形版本。其中α又称为原形版本。
- $m=0$时？只有原形版本
- $m=|\alpha|$时？ 有个$A\longrightarrow\varepsilon$版本
- 如果使用变形候选式推导出w，
  那么，使用原形候选式亦推导出w。

- 例：若A 可空，那么产生式 $B\longrightarrow CAD$ 的变形版本为：
- $B\longrightarrow CAD$； $B\longrightarrow CD$
- 例：若A和B可空，那么产生式$S\longrightarrow AaBDbAbC$的变形版本：
- $S\longrightarrow aDbbC|\ aDbAbC|\ aBDbbC|\ aBDbAbC|\ AaDbbC|\ AaDbAbC|\ AaBDbbC|\ AaBDbAbC$

算法：去除ε-产生式

- 输入：$G = (V, T, P, S)$
- 输出：$G_a = (V, T, P_a, S)$

1. P_a初始化为空集合；
2. 找出G的所有可空变元；
3. 对P中每个非ε产生式，将它的变形全部加入P_a；
4. P_a 中去掉以ε为候选式的所有产生式。

例：去除ε-产生式

- G： $S \longrightarrow ABC$； $A \longrightarrow aA \mid \varepsilon$； $B \longrightarrow bB \mid \varepsilon$； $C \longrightarrow \varepsilon$

- A，B，C，和S 都是可空的。
- G_a:

  $S \longrightarrow ABC \mid AB \mid AC \mid A \mid BC \mid B \mid C$

  $A \longrightarrow aA \mid a$

  $B \longrightarrow bB \mid b$

例：去除ε-产生式

- G：$S \longrightarrow AB$；$A \longrightarrow aAA \mid \varepsilon$；$B \longrightarrow bBB \mid \varepsilon$

- A，B 是可空的。
- G_a:

  $S \longrightarrow AB \mid A \mid B$

  $A \longrightarrow aAA \mid aA \mid a$

  $B \longrightarrow bBB \mid bB \mid b$

- $L(G)=L(G_a)\cup\{\varepsilon\}$

定理5.8

- 定理7.9 如果文法G_a是利用去除ε-产生式的算法从G得出的，则$L(G_a) = L(G) - \{\varepsilon\}$。
- 令$G=(V, T, P, S)$
- 令$G_a=(V, T, P_a, S)$

- 证明对任意变元$A \in V$：
- $A \Rightarrow^*_{Ga} w$ 当且仅当$A \Rightarrow^*_G w$且$w \neq \varepsilon$。

- 对推导步数进行归纳。

定理证明

- （当且）若$w \neq \varepsilon$且$A \Rightarrow^*_G w$，则$A \Rightarrow^*_{Ga} w$。
- 基础：1步时，$(A, w) \in P$，由于$w \neq \varepsilon$，所以$(A, w) \in P_a$。
- 归纳：假设小于n步时结论成立，那么n步时，$A \Rightarrow_G X_1 \ldots X_k \Rightarrow^*_G w$。$w$可被分成$w = w_1 \ldots w_k$，其中$X_i \Rightarrow^*_G w_i$，$i=1,\ldots,k$，对所有i都小于n步。那么根据归纳假设，若$w_i \neq \varepsilon$，那么$X_i \Rightarrow^*_{Ga} w_i$。
- 我们把$w_j=\varepsilon$的X_j去掉，对于所有j，结果记为α。那么α是$X_1 \ldots X_k$的一个变形，$\alpha \Rightarrow^*_{Ga} w$，同时$A \rightarrow X_1 \ldots X_k$和$A \rightarrow \alpha$都在$G_a$中，因此，$A \Rightarrow_{Ga} \alpha \Rightarrow^*_{Ga} w$。

定理证明

- （仅当）若$A \Rightarrow^{*}_{Ga} w$则$A \Rightarrow^{*}_{G} w$且$w \neq \varepsilon$。
- 基础：推导步数为1时，必有$(A,w) \in P_a$，这是原形产生式所以$(A,w) \in P$而且$w \neq \varepsilon$。
- 归纳：假设推导步数小于n时结论成立，那么推导步数为n时，该推导一定是$A \Rightarrow_{Ga} \alpha \Rightarrow^{*}_{Ga} w$，
- 由于$\alpha \Rightarrow^{*}_{Ga} w$的步数小于n所以根据归纳假设$\alpha \Rightarrow^{*}_{G} w$。
- 我们找出$(A,\alpha) \in P_a$的原形产生式$(A,\delta) \in P_a$那么$(A,\delta) \in P$并且，
- $A \Rightarrow_{G} \delta \Rightarrow^{*}_{G} w$。

5.5.3 去除单位产生式

- 单位产生式：右部为单个变元的产生式。

- 思路：若$A \Rightarrow^* B$是通过一系列单位产生式完成，并且$B \longrightarrow \alpha$是一个非单位产生式，那么给文法中增加产生式$A \longrightarrow \alpha$。
- 此项工作完成后，删掉所有单位产生式。

- $S \longrightarrow A|a$
- $A \longrightarrow B|b|Cc$
- $B \longrightarrow d|e$

- $S \longrightarrow d|e|b|Cc|a$
- $A \longrightarrow d|e|b|Cc$
- $B \longrightarrow d|e$

发现单位产生式的算法

- 关键点：找出所有单位对(A, B) 使得$A \Rightarrow^* B$ 仅通过一系列单位产生式得到。
- 基础：对于任何变元，(A, A)是单位对。
- 归纳：若已知(A, B)是单位对且$B \longrightarrow C$ 是一个单位产生式，那么(A, C)是单位对。

- 例：α，β，γ，δ，η都不是单个变元，
- $A \longrightarrow B \mid \alpha$
- $B \longrightarrow C \mid \beta$
- $C \longrightarrow D \mid \gamma$
- $D \longrightarrow \delta \mid \eta$
- 单位对：(A, B)，(A, C)，(A, D)，(B, C)，(B, D)，(C, D)

证明算法正确工作

- 基础： 对于任何变元，(A, A)是单位对。
- 归纳： 若已知(A, B)是单位对且B⟶C 是一个单位产生式，那么(A, C)是单位对。

- 证明算法能发现给定文法的所有单位对。
- 按照找出单位对 (A, B)的次序归纳，能表明$A \Rightarrow^* B$ 是通过单位产生式得到。
- 相反地，对经过单位产生式得到的$A \Rightarrow^* B$ 的推导步数进行归纳，能够表明 单位对(A, B) 被发现。

去除单位产生式的算法

- 输入：CFG $G=(V, T, P, S)$
- 输出：CFG $G_r=(V, T, P_r, S)$

1. 求出G的所有单位对；
2. 对每个单位对(A, B)，把所有的$(A, \delta)\in P$加入P_r，其中δ不是单个变元且$(B, \delta)\in P$。

- 证明：对任意$w\in T^*$，$w\in L(G)$当且仅当$w\in L(G_r)$。
- （当且）$S\Rightarrow^*_{G_r} w$为1步时有$(S, w)\in P$或者存在单位对(S, A)且$(A, w)\in P$。假设小于n步时结论成立，那么n步时，$S\Rightarrow_{G_r} X_1\ldots X_k\Rightarrow^*_{G_r} w$，其中$X_i\Rightarrow^*_{G_r} w_i$，根据归纳假设有$X_i\Rightarrow^*_{G} w_i$。对于$(S, X_1\ldots X_k)\in P_r$，或者$(S, X_1\ldots X_k)\in P$，或者存在单位对(S, A)且$(A, X_1\ldots X_k)\in P$。
- （仅当）仿照上面思考。

例：去除单位产生式

1. 求出G的所有单位对；
2. 对每个单位对(A, B)，把所有的(A, δ)∈P加入P_r，其中δ不是单个变元且(B, δ)∈P。

- 例：α，β，γ，δ，η都不是单个变元，
- A⟶B | α；B⟶C | β；C⟶D | γ；D⟶δ | η
- (A, B)，(A, C)，(A, D)，(B, C)，(B, D)，(C, D)
- A⟶ δ | η | γ | β | α
- B⟶ δ | η | γ | β
- C⟶ δ | η | γ
- D⟶ δ | η

例：去除单位产生式

- 例：
- $E \longrightarrow E+T \mid T$
- $T \longrightarrow T*F \mid F$
- $F \longrightarrow (E) \mid d$

- (E, T)，(E, F)，(T, F)

- $E \longrightarrow E+T \mid T*F \mid (E) \mid d$
- $T \longrightarrow T*F \mid (E) \mid d$
- $F \longrightarrow (E) \mid d$

清理文法

- 定理5.10 若L是一个CFL，那么有一个L-{ε}的CFG有如下特征：
  - 不含无用符号。
  - 不含ε-产生式。
  - 不含单位产生式。

- 即文法符号都是有用的，并且每个候选式或者是单个终结符或者是长度大于1的串。
- 文法清理步骤：
  1. 消除ε-产生式。
  2. 消除单位产生式。
  3. 消除无产出变元。
  4. 消除不可达变元。

小结

- 计算G_a，该文法的变元都不是可空变元；(actual/solid)
- 计算G_r，该文法没有单位产生式。(retrenched/direct)
- 计算G_g，该文法的变元都是产出的。(generative)
- 计算G_u，该文法的文法符号都是有用的。(useful/reachable)

$$G \Rightarrow G_a \equiv G_r \equiv G_g \equiv G_u$$

- P116： 习题1.1a&b； 习题5.4； 习题5.5；
- 习题5.6； 习题5.7； 习题5.8
- P117：习题5.9； 习题5.10； 习题5.13