Chapter 5

上下文无关文法和上下文无关语言

5.1 上下文无关文法

上下文无关文法在程序设计语言的设计, 编译器的实现等方面有重要应用, 也应用在可扩展标记语言 (XML) 的格式类型定义 (DTD) 中等. 上下文无关文法重要的原因，在于它们拥有足够强的表达能力，可以表示大多数程序设计语言的语法; 实际上, 几乎所有程序设计语言都是通过上下文无关文法来定义的. 另一方面, 上下文无关文法又足够简单, 使得我们可以构造有效的分析算法来检验一个给定字串是否是由某个上下文无关文法产生的. (如 LR 分析器和 LL 分析器)

5.1.1 文法的示例

自然语言

上下文无关文法最初是用来描述自然语言的, 比如如下文法规则:

$$\begin{aligned}
&\langle \textit{sentence} \rangle \to \langle \textit{noun phrase} \rangle \ \langle \textit{verb phrase} \rangle \\
&\langle \textit{noun phrase} \rangle \to \langle \textit{adjective} \rangle \ \langle \textit{noun phrase} \rangle \\
&\langle \textit{noun phrase} \rangle \to \langle \textit{noun} \rangle \\
&\langle \textit{noun} \rangle \to \text{boy} \\
&\langle \textit{adjective} \rangle \to \text{little} \\
&\cdots
\end{aligned}$$

算数表达式

用于产生具有 $+, *$ 的表达式的规则, 用 **id** 表示操作数

$$\begin{aligned}
&\langle \text{expression} \rangle \to \langle \text{expression} \rangle + \langle \text{expression} \rangle \\
&\langle \text{expression} \rangle \to \langle \text{expression} \rangle * \langle \text{expression} \rangle \\
&\langle \text{expression} \rangle \to (\ \langle \text{expression} \rangle\ ) \\
&\langle \text{expression} \rangle \to \textbf{id}
\end{aligned}$$

这种表示方法也称为 BNF(Backus-Naur Form).

示例

串 w 称为“回文 (*palindrome*)”, 当且仅当 $w = w^R$, 比如 *drawkward* (Dr. Awkward), “僧游云隐寺, 寺隐云游僧”. 那么 $\{0, 1\}$ 上的回文语言 L_{pal} 可定义为:

$$L_{pal} = \{w \in \{0, 1\}^* \mid w = w^R\}$$

并且, 很容易证明是 L_{pal} 是非正则的. 显然 ε,0,1 都是回文, 而且如果 w 是回文, $0w0$ 和 $1w1$ 也是回文. 如果我们把这种递归的定义表示为文法, 可以如下定义:

$$P \to \varepsilon \qquad P \to 0P0$$
$$P \to 0 \qquad P \to 1P1$$
$$P \to 1$$

使用上面的文法, 可以通过 P 产生 $\{0,1\}$ 上全部的回文串, 比如串 0010100 可以通过先使用 $P \to 0P0$ 两次, 再使用 $P \to 1P1$ 一次, 再使用 $P \to 0$ 一次得到.

5.1.2 上下文无关文法的形式定义

上下文无关文法 (也称文法, *Context-Free Grammar*, CFG) G 是一个四元组 $G = (V, T, P, S)$, 其中:

(1) V: 变元 (*Variable*) 的有穷集, 变元也称为非终结符或语法范畴;

(2) T: 终结符 (*Terminal*) 的有穷集, 用来构成语言的串, 这里 $V \cap T = \emptyset$;

(3) P: 产生式 (*Production*) 的有穷集, 表示语言的递归定义, 其中每个产生式都包括 3 部分:

a) 一个变元, 称为产生式的头 (*head*) 或左部, 是产生式中被定义的部分;

b) 一个产生式符号 $\to$;

c) 一个 $(V \cup T)^*$ 中的串, 称为产生式的体 (*body*) 或右部, 表示如何定义产生式的头;

(4) S: 初始符号 (*Start symbol*), $S \in V$, 表示文法开始的地方.

产生式 $A \to \alpha$, 读作 A 定义为 α. 如果有多个 A 的产生式 $A \to \alpha_1$, $A \to \alpha_2$, $\cdots A \to \alpha_n$, 可以简写为 $A \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n$. 文法中定义 A 的所有产生式, 称为 A 产生式.

字符使用的一般约定: a, b 等表示终结符; A, B 等表示非终结符; x, y, z 等表示终结符串; X, Y 等表示终结符或非终结符; α, β 等表示终结符或非终结符的串.

示例

例 1. 用于产生 $\{0,1\}$ 上回文的 CFG G_{pal} 的产生式:

$$P \to \varepsilon \mid 0 \mid 1 \mid 0P0 \mid 1P1$$

例 2. 用 **id** 表示数, 产生具有 $+$ 和 $*$ 的算数表达式的文法产生式:

$G = (\{E\}, \{\mathbf{id}, +, *, (,)\}, P, E)$, P 如下

$$E \to E + E \qquad E \to E * E \qquad E \to (E) \qquad E \to \mathbf{id}$$

例 3. $L = \{0^n 1^m \mid n \neq m\}$.

$$S \to AC \mid CB \qquad C \to 0C1 \mid \varepsilon \qquad A \to A0 \mid 0 \qquad B \to 1B \mid 1$$

例 4. $L_{eq} = \{w \in \{0,1\}^* \mid w$ 中 0 和 1 个数相等$\}$.

$$S \to 0S1 \mid 1S0 \mid SS \mid \varepsilon$$

例 5. $L_{j \geq 2i} = \{a^i b^j \mid j \geq 2i\}$.

$$S \to aSbb \mid B \qquad B \to \varepsilon \mid bB$$

5.1.3 文法的派生

分析文法和串的关系, 可以由产生式的头向产生式的体分析, 称为派生或推导 (*derivation*), 也可以由产生式的体向头分析, 称为递归推理 (*recursive inference*) 或归约 (*reduction*).

CFG $G=(V,T,P,S)$, 设 $\alpha,\beta,\gamma\in(V\cup T)^*$ 且 $A\in V$, 如果 $A\to\gamma$ 是 G 的产生式, 那么称在文法 G 中由 $\alpha A\beta$ 可派生出 $\alpha\gamma\beta$, 记为

$$\alpha A\beta \underset{G}{\Rightarrow} \alpha\gamma\beta$$

如果 G 已知, 可记为 $\alpha A\beta\Rightarrow\alpha\gamma\beta$. 表示产生式 $A\to\gamma$ 应用到串 $\alpha A\beta$ 的变元 A, 得到 $\alpha\gamma\beta$. 相应的, 称在文法 G 中 $\alpha\gamma\beta$ 可归约为 $\alpha A\beta$.

设 $\alpha_1,\alpha_2,\cdots,\alpha_m\in(V\cup T)^*$, 这里 $m\geq 1$, 对 $i=1,2,\cdots,m-1$ 有 $\alpha_i \underset{G}{\Rightarrow} \alpha_{i+1}$ 成立, 即

$$\alpha_1 \underset{G}{\Rightarrow} \alpha_2 \underset{G}{\Rightarrow} \cdots \underset{G}{\Rightarrow} \alpha_{m-1} \underset{G}{\Rightarrow} \alpha_m$$

则记为 $\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m$, 表示 α_1 经过零步或多步派生得到 α_m, 如果 G 已知, 可记为 $\alpha_1\overset{*}{\Rightarrow}\alpha_m$. (类似 FA 中的 δ 与 $\hat{\delta}$.) 若 α 经过 i 步派生出 β, 可记为 $\alpha\overset{i}{\Rightarrow}\beta$.

示例

算数表达式 $\mathbf{id}*(\mathbf{id}+\mathbf{id})$ 的派生过程如下:

$$\begin{aligned}E&\Rightarrow E*E\Rightarrow E*(E)\Rightarrow \mathbf{id}*(E)\Rightarrow \mathbf{id}*(E+E)\\&\Rightarrow \mathbf{id}*(E+\mathbf{id})\Rightarrow \mathbf{id}*(\mathbf{id}+\mathbf{id})\end{aligned}$$

5.1.4 最左派生和最右派生

为限制派生过程的随意性, 要求派生的每一步, 只能替换最左边变元的派生过程, 称为最左派生 (或最左推导), 记为 $\underset{\text{lm}}{\Rightarrow}$ 和 $\overset{*}{\underset{\text{lm}}{\Rightarrow}}$; 而只能替换最右变元的派生过程, 称为最右派生 (或最右推导), 记为 $\underset{\text{rm}}{\Rightarrow}$ 和 $\overset{*}{\underset{\text{rm}}{\Rightarrow}}$.

$\mathbf{id}*(\mathbf{id}+\mathbf{id})$ 的最左派生和最右派生分别为:

$$\begin{aligned}E&\underset{\text{lm}}{\Rightarrow} E*E\underset{\text{lm}}{\Rightarrow}\mathbf{id}*E\underset{\text{lm}}{\Rightarrow}\mathbf{id}*(E)\underset{\text{lm}}{\Rightarrow}\mathbf{id}*(E+E)\\&\underset{\text{lm}}{\Rightarrow}\mathbf{id}*(\mathbf{id}+E)\underset{\text{lm}}{\Rightarrow}\mathbf{id}*(\mathbf{id}+\mathbf{id})\end{aligned}$$

$$\begin{aligned}E&\underset{\text{rm}}{\Rightarrow} E*E\underset{\text{rm}}{\Rightarrow}E*(E)\underset{\text{rm}}{\Rightarrow}E*(E+E)\underset{\text{rm}}{\Rightarrow}E*(E+\mathbf{id})\\&\underset{\text{rm}}{\Rightarrow}E*(\mathbf{id}+\mathbf{id})\underset{\text{rm}}{\Rightarrow}\mathbf{id}*(\mathbf{id}+\mathbf{id})\end{aligned}$$

任何派生都有等价的最左派生和最右派生, 即 $A\overset{*}{\Rightarrow}w$ 当且仅当 $A\overset{*}{\underset{\text{lm}}{\Rightarrow}}w$ 当且仅当 $A\overset{*}{\underset{\text{rm}}{\Rightarrow}}w$.

5.1.5 文法产生的语言

由 CFG $G=(V,T,P,S)$ 产生的语言定义为

$$\mathbf{L}(G)=\{w\mid w\in T^*,\ S\overset{*}{\underset{G}{\Rightarrow}}w\}.$$

那么串 w 在 $\mathbf{L}(G)$ 中, 要满足:

(1) w 仅能由终结符组成;
(2) w 能由初始符号 S 派生出来.

语言 $L = \mathbf{L}(G)$ 称为**上下文无关语言** (*Context-Free Language*, CFL). 这里的上下文无关, 就是指文法派生的每一步

$$\alpha A \beta \Rightarrow \alpha \gamma \beta$$

中, 串 γ 仅根据 A 的产生式派生, 而无需依赖上下文的 α 和 β. 如果有两个 CFG G_1 和 G_2, $\mathbf{L}(G_1) = \mathbf{L}(G_2)$, 则称 G_1 和 G_2 等价.

示例

描述 CFG $G = (\{S\}, \{a, b\}, \{S \to aSb, S \to ab\}, S)$ 产生的语言?

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

$$S \Rightarrow aSb \Rightarrow aaSbb \Rightarrow aaaSbbb \Rightarrow \cdots \Rightarrow a^{n-1}Sb^{n-1} \Rightarrow a^n b^n$$

示例

CFG $G = (V, T, P, S)$, 其中 $V = \{S, A, B\}$, $T = \{a, b\}$, P 如下

$$\begin{aligned} S &\to bA \mid aB \\ A &\to bAA \mid aS \mid a \\ B &\to aBB \mid bS \mid b \end{aligned}$$

能够产生 a, b 数目相等串的集合, 可以对 w 长度归纳证明.

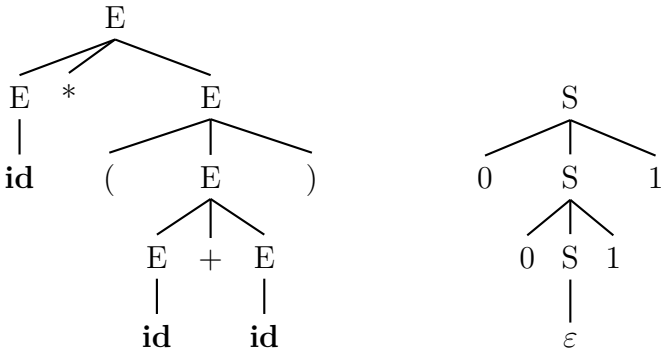
5.1.6 句型

在文法 G 中, 能由初始符号派生出来的串, 对文法本身有重要意义, 称为 G 的**句型** (*sentential form*). 即任何 $\alpha \in (V \cup T)^*$, 如果 $S \overset{*}{\Rightarrow} \alpha$, 则称为句型. 如果 $S \overset{*}{\underset{\text{lm}}{\Rightarrow}} \alpha$ 的 α 称为**左句型**, $S \overset{*}{\underset{\text{rm}}{\Rightarrow}} \alpha$ 的 α 称为**右句型**. 只含有终结符的句型, 也称为 G 的**句子**. 而 $\mathbf{L}(G)$ 就是文法 G 全部的句子.

5.2 语法分析树

语法分析树用来表示派生 (或递归推理), 可以从树中看出整个派生过程和最终产生的符号串. 在编译器设计中, 语法分析树是表示源程序的重要数据结构, 用来指导由程序到可执行代码的翻译.

例如, 产生表达式 $\mathbf{id} * (\mathbf{id} + \mathbf{id})$ 和 L_{eq} 中 0011 的过程, 可以分别用语法分析树表示为



CFG $G = (V, T, P, S)$ 的**语法分析树** (*parse tree*), 也称为**派生树** (*derivation tree*), 形式定义为:

(1) 每个内节点的标记是 V 中的变元符号;

(2) 每个叶节点的标记是 $V \cup T \cup \{\varepsilon\}$ 中的符号;

(3) 如果内节点的标记是 A, 其子节点从左至右分别为

$$X_1,\ X_2,\ \cdots,\ X_n$$

那么, $A \to X_1X_2\cdots X_n$ 是 P 的一个产生式; 若某个 X_i 是 ε, 则 X_i 一定是 A 唯一的子节点, 且 $A \to \varepsilon$ 是一个产生式.

语法分析树的全部叶节点从左到右连接起来, 称为该树的*产物* (*yield*) 或*结果*, 并且总是能由根节点变元派生出来. 如果树根节点是初始符号 S, 叶节点是终结符或 ε, 那么该树的产物属于 $\mathbf{L}(G)$.

语法分析树中标记为 A 的内节点及其全部子孙节点构成的子树, 称为 A *子树*.

5.2.1 语法树和派生的等价性

定理 1. *CFG $G=(V,T,P,S)$, 那么文法 G 中 $S \overset{*}{\Rightarrow} \alpha$ 当且仅当在文法 G 中存在以 S 为根节点产物为 α 的语法分析树.*

证明. 对任意 $A \in V$, 如果有 $A \overset{*}{\Rightarrow} \alpha$, 当且仅当存在以 A 为根且产物为 α 的语法分析树.

($\Rightarrow$) 对派生的步骤数做归纳: (1) 当仅需 1 次时, 那么一定有产生式 $A \to \alpha$, 根据语法树定义, 显然命题成立; (2) 假设派生小于 n 步时命题都成立, 当需要 $n+1$ 步时的第 1 步派生, 一定有 $A \to X_1X_2\cdots X_n$, 其中 X_i 的派生都不超过 n 步, 根据归纳假设, 都有一棵语法分析树, 再构造得 A 的语法分析树.

($\Leftarrow$) 对树的内部节点数做归纳: (1) 当只有 1 个内节点时, 一定是 A, 产物是 α, 所以 $A \to \alpha$ 是产生式, 所以有 $A \overset{*}{\Rightarrow} \alpha$; (2) 若假设内节点数量 $\le n$ 时命题成立, 当内节点数为 $n+1$ 时, 根节点 A 的子节点 X_i 或者是叶子, 或者是 X_i 子树, 而且每个的内节点数都 $\le n$, 所以有 $X_i \overset{*}{\Rightarrow} \alpha_i$, 而因为 $A \to X_1X_2\cdots X_n$, 所以 $A \Rightarrow X_1X_2\cdots X_n \overset{*}{\Rightarrow} \alpha_1X_2\cdots X_n \overset{*}{\Rightarrow} \alpha_1\alpha_2\cdots\alpha_n = \alpha$. □

给定 CFG $G=(V,T,P,S)$, $A \in V$, 下面 5 个命题等价:

(1) 通过递归推理, 可以确定终结符串 w 在变元 A 的语言中;

(2) $A \overset{*}{\Rightarrow} w$;

(3) $A \overset{*}{\underset{\text{lm}}{\Rightarrow}} w$;

(4) $A \overset{*}{\underset{\text{rm}}{\Rightarrow}} w$;

(5) 存在以 A 为根节点, 产物为 w 的语法分析树;

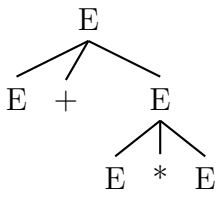
每棵语法分析树都有唯一的最左派生和唯一的最右派生. 因在 $A \Rightarrow \alpha$ 过程中, 第一步使用产生式 $A \to X_1X_2\cdots X_n$, 再 (递归的) 使用 $X_i \overset{*}{\Rightarrow} \alpha_i$ 的最左派生, 从左至右依次派生 X_i, 就得到唯一的最左派生; 同理, 若从右至左的依次使用 $X_i \overset{*}{\Rightarrow} \alpha_i$ 的最右派生, 则得到唯一的最右派生.

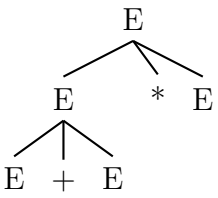
5.3 文法和语言的歧义性

如果 CFG G 使某些串有两棵不同的语法分析树, 则称 G 是*歧义* (*ambiguity*) 的, 也称*二义性*的.

示例

文法 $G=(\{E\},\{id,*,+,(,)\},\{E \to E+E \mid E*E \mid (E) \mid id\},E)$, 产生句型 $E+E*E$ 时, 会有下面两棵语法分析树:





(1) $E \Rightarrow E+E \Rightarrow E+E*E$

(2) $E \Rightarrow E*E \Rightarrow E+E*E$

5.3.1 文法歧义性的消除

有些文法的歧义性, 可以通过重新设计来文法消除. 比如考虑了运算的优先级表达式文法:

$$E \to E+T \mid T \qquad T \to T*F \mid F \qquad F \to (E) \mid \mathbf{id}$$

5.3.2 文法的固有歧义性

产生同样的语言可以有多种文法, 如果上下文无关的语言 L 的所有文法都是歧义的, 那么称 L 是**固有歧义** (*Inherent Ambiguity*) 的. 这样的语言是确实存在的.

比如语言 $L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$ 是固有歧义的.

“判定任何给定 CFG G 是否是歧义的”是一个不可判定问题.

5.4 上下文无关文法的化简

典型的分析问题: 给定 CFG G 和串 w, 判断是否 $w \in \mathbf{L}(G)$. 这是编译器设计和自然语言处理的基本问题之一. 由于文法的形式非常自由, 为了便于分析和解决问题, 我们希望在不改变文法能力的前提下, 化简文法和限制文法的格式.

文法的化简主要包括:

(1) 消除**无用符号** (*useless symbols*): 不在 $S \overset{*}{\Rightarrow} w (w \in T^*)$ 的派生过程中出现的变元和终结符;

(2) 消除 ε **产生式** (ε-*productions*): 形式为 $A \to \varepsilon$, A 是变元 (得到语言 $L - \{\varepsilon\}$);

(3) 消除**单元产生式** (*unit productions*): 形式为 $A \to B$, A 和 B 都是变元.

5.4.1 消除无用符号

符号 $X \in (V \cup U)$, 如果由初始符号 $S \overset{*}{\Rightarrow} \alpha X \beta$, 称 X **是可达的** (*reachable*); 如果有 $\alpha X \beta \overset{*}{\Rightarrow} w$ $(w \in T^*)$, 称 X **是产生的** (*generating*). 如果 X 同时是产生的和可达的, 即 $S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w$ $(w \in T^*)$, 称 X **是有用的**, 否则称为**无用符号**. 从文法中消除无用符号, 同时不改变文法产生的语言.

计算“产生的”符号集的算法

(1) 每个 T 中的符号都是产生的;
(2) 如果有产生式 $A \to \alpha$ 且 α 中符号都是产生的, 则 A 是产生的.

计算“可达的”符号集的算法

(1) 符号 S 是可达的;
(2) 如果有产生式 $A \to \alpha$ 且 A 是可达的, 则 α 中的符号都是可达的.

定理 2. *每个非空的 CFL 都能被一个不带无用符号的 CFG 产生.*

示例

例如文法	消除非产生的	消除非可达的
$S \to AB \mid a$	$S \to a$	$S \to a$
$A \to b$	$A \to b$	

注意: 要先消除非产生的, 再消除非可达的.

5.4.2 消除 ε-产生式

形如 $A \to \varepsilon$ 的产生式称为*空产生式*或 ε-*产生式*. 某个 CFL L, 消除其中全部的 ε-产生式后得到语言 $L - \{\varepsilon\}$ 也是 CFL. 如果变元 $A \overset{*}{\Rightarrow} \varepsilon$, 称 A *是可空的*. 则消除 ε 产生式的算法如下:

先确定全部可空的变元:

(1) 如果 $A \to \varepsilon$, 则 A 是可空的;

(2) 如果 $B \to \alpha$ 且 α 中的每个符号都是可空的, 则 B 是可空的.

再替换全部带有可空符号的产生式, 如果 $A \to X_1X_2\cdots X_n$ 是产生式, 那么用所有的 $A \to Y_1Y_2\cdots Y_n$ 产生式代替, 其中:

(1) 若 X_i 不是可空的, 则 $Y_i = X_i$;

(2) 若 X_i 是可空的, 则 Y_i 是 X_i 或 ε;

(3) 但 Y_i 不能全部为 ε.

定理 3. *对于某个 CFG G, 有一个不带无用符号和 ε-产生式的 CFG G', 使 $\mathbf{L}(G') = \mathbf{L}(G) - \{\varepsilon\}$.*

示例

给定 CFG $G = (\{S, A, B\}, \{a, b\}, P, S)$	则 CFG $G' = (\{S, A, B\}, \{a, B\}, P', S)$
$S \to AB$	$S \to AB \mid A \mid B$
$A \to AaA \mid \varepsilon$	$A \to AaA \mid Aa \mid aA \mid a$
$B \to BbB \mid \varepsilon$	$B \to BbB \mid Bb \mid bB \mid b$

5.4.3 消除单元产生式

消除单元产生式, 先确定全部的 $A \overset{*}{\Rightarrow} B$ 的变元对 A 和 B, 如果有 $B \to \alpha$ 不是单元产生式, 就增加一条产生式 $A \to \alpha$, 再删除全部单元产生式.

定理 4. *每个不带 ε 的 CFL 都可由一个不带无用符号, ε 产生式和单元产生式的文法来定义.*

示例

消除文法的单元产生式

$S \to A \mid B \mid 0S1$ $A \to 0A \mid 0$ $B \to 1B \mid 1$

先确定非单元产生式, 在确定单位对 (S,A) 和 (S,B), 再带入非单元产生式

$S \to 0S1$ $A \to 0A \mid 0$ $B \to 1B \mid 1$

$S \to 0A \mid 0$ $S \to 1B \mid 1$

5.4.4 文法简化的顺序

文法简化步骤的顺序是重要的, 一个可靠的顺序是:

(1) 消除 ε 产生式;
(2) 消除单元产生式;
(3) 消除非产生的无用符号;
(4) 消除非可达的无用符号.

5.5 乔姆斯基范式和格雷巴赫范式

文法格式的限制通过两个范式定理给出: 乔姆斯基范式 (*Chomsky Normal Form*, CNF) 定理和格雷巴赫范式 (*Greibach Normal Form*, GNF) 定理.

5.5.1 乔姆斯基范式

定理 5 (乔姆斯基范式定理)**.** *每个不带 ε 的 CFL 都可以由这样的 CFG G 产生, G 中所有产生式的形式或为 $A \to BC$, 或为 $A \to a$, 这里的 A, B 和 C 是变元, a 是终结符.*

设文法不带无用符号, ε 产生式和单元产生式. 则考虑文法每个形式为

$$A \to X_1 X_2 \cdots X_m \quad (m \geq 2)$$

的产生式, 若 X_i 是终结符 a, 则引进变元 C_a 替换 X_i 并新增产生式 $C_a \to a$. 此时, 对每个形式为

$$A \to B_1 B_2 \cdots B_m \quad (m \geq 3)$$

的产生式, 引进变元 $D_1, D_2, \cdots, D_{m-2}$ 并用一组产生式

$$A \to B_1 D_1,\ D_1 \to B_2 D_2,\ \cdots,\ D_{m-2} \to B_{m-1} B_m$$

来替换, 就完成了到 CNF 的转换.

利用 CNF 派生长度为 n 的串, 刚好需要 $2n-1$ 步, 因此可以设计算法判断串是否在 CFL 中. 此外利用 CNF 可以实现多项式时间的解析算法 (CYK 算法).

示例

CFG $G = (\{S, A, B\}, \{a, b\}, P, S)$, 产生式集合 P 为:

$$\begin{aligned} S &\to bA \mid aB \\ A &\to bAA \mid aS \mid a \\ B &\to aBB \mid bS \mid b \end{aligned}$$

构造等价的 CNF 文法.

$$\begin{aligned} S &\to C_b A \mid C_a B \\ A &\to C_a S \mid C_b D_1 \mid a \\ B &\to C_b S \mid C_a D_2 \mid b \\ D_1 &\to AA \\ D_2 &\to BB \\ C_a &\to a \\ C_b &\to b \end{aligned}$$

5.5.2 格雷巴赫范式

定理 6 (格雷巴赫范式定理)**.** *每个不带 ε 的 CFL 都可以由这样的 CFG G 产生, G 中的每个产生式的形式为 $A \to a\alpha$, 这里 A 是变元, a 是终结符, α 是零个或多个变元的串.*

因为格雷巴赫范式的每个产生式都会引入一个终结符, 所以长度为 n 的串的派生恰好是 n 步.

示例

将文法 $S \to AB,\ A \to aA \mid bB \mid b,\ B \to b$ 转换为 GNF.

$$S \to aAB \mid bBB \mid bB \qquad A \to aA \mid bB \mid b \qquad B \to b$$

将文法 $S \to 01S1 \mid 00$ 转换为 GNF.

$$S \to 0ASA \mid 0B \qquad A \to 1 \qquad B \to 0$$

消除直接左递归

直接左递归 (*immediate left-recursion*) 形式为 $A \to A\alpha$, 若有 A 产生式

$$A \to A\alpha_1 \mid A\alpha_2 \mid \cdots \mid A\alpha_n \mid \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m$$

其中 $\alpha_i \neq \varepsilon$, β_j 不以 A 开始; 消除方法是: 引入新的变元 B, 并用如下产生式替换

$$A \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m \mid \beta_1 B \mid \beta_2 B \mid \cdots \mid \beta_m B$$

$$B \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n \mid \alpha_1 B \mid \alpha_2 B \mid \cdots \mid \alpha_n B$$

消除间接左递归

间接左递归 (*indirect left-recursion*) 形式为

$$A \to B\alpha \mid C \qquad B \to A\beta \mid D$$

派生过程可能产生 $A \Rightarrow B\alpha \Rightarrow A\beta\alpha$.

为变元重命名为 $A_1, A_2, \cdots, A_n$, 通过替换变元, 使每个 A_i 产生式, $A_i \to \alpha$ 的右半部, 或以终结符开始, 或以 A_j $(j \geq i)$ 开始; 然后消除产生的直接左递归.