Chapter 5

上下文无关文法

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而又足够简单, 使得我们可以设计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)

自然语言的文法

$$
\begin{aligned}
\langle \textit{sentence} \rangle &\to \langle \textit{noun-phrase} \rangle \langle \textit{verb-phrase} \rangle \\
\langle \textit{noun-phrase} \rangle &\to \langle \textit{article} \rangle \langle \textit{noun} \rangle \mid \langle \textit{article} \rangle \langle \textit{adjective} \rangle \langle \textit{noun} \rangle \\
\langle \textit{verb-phrase} \rangle &\to \langle \textit{verb} \rangle \mid \langle \textit{verb} \rangle \langle \textit{noun-phrase} \rangle \\
\langle \textit{article} \rangle &\to \texttt{a} \mid \texttt{the} \\
\langle \textit{noun} \rangle &\to \texttt{boy} \mid \texttt{girl} \mid \texttt{cat} \\
\langle \textit{adjective} \rangle &\to \texttt{big} \mid \texttt{small} \mid \texttt{blue} \\
\langle \textit{verb} \rangle &\to \texttt{sees} \mid \texttt{likes} \\
&\cdots
\end{aligned}
$$

使用文法规则产生句子:

$$
\begin{aligned}
\langle \textit{sentence} \rangle &\Rightarrow \langle \textit{noun-phrase} \rangle \langle \textit{verb-phrase} \rangle \\
&\Rightarrow \langle \textit{article} \rangle \langle \textit{noun} \rangle \langle \textit{verb-phrase} \rangle \\
&\Rightarrow \langle \textit{article} \rangle \langle \textit{noun} \rangle \langle \textit{verb} \rangle \langle \textit{noun-phrase} \rangle \\
&\Rightarrow \langle \textit{article} \rangle \langle \textit{noun} \rangle \langle \textit{verb} \rangle \langle \textit{article} \rangle \langle \textit{adjective} \rangle \langle \textit{noun} \rangle \\
&\Rightarrow \texttt{the}\ \langle \textit{noun} \rangle \langle \textit{verb} \rangle \langle \textit{article} \rangle \langle \textit{adjective} \rangle \langle \textit{noun} \rangle \\
&\Rightarrow \cdots \\
&\Rightarrow \texttt{the girl sees a blue cat}
\end{aligned}
$$

如果把式子中表示定义的符号 $\to$ 写为 ::= 时, 也称为 BNF(Backus-Naur Form, 巴科斯范式), 程序设计语言的语法规范常使用 BNF 描述.

定义. 如果字符串 $w \in \Sigma^*$ 满足

$$w = w^R,$$

则称字符串 w 为回文 *(palindrome).*

定义. 如果语言 L 中的字符串都是回文, 则称 L 为回文语言

$$L = \{w \in \Sigma^* \mid w = w^R\}.$$

- ε, 010, 0000, *radar*, *racecar*, *drawkward*
- A man, a plan, a canal — Panama
- 僧游云隐寺, 寺隐云游僧

例 1. 字母表 $\Sigma = \{0, 1\}$ 上的回文语言

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

- 很容易证明是 L_{pal} 是非正则的. 但如何表示呢?
- 可使用递归的方式来定义:

  1. 首先 ε, 0, 1 都是回文
  2. 如果 w 是回文, $0w0$ 和 $1w1$ 也是回文

- 使用嵌套定义表示这种递归结构:

$$\begin{array}{ll} A \to \varepsilon & A \to 0A0 \\ A \to 0 & A \to 1A1 \\ A \to 1 & \end{array}$$

使用上面的文法, 可以通过 A 产生 $\{0, 1\}$ 上全部的回文串, 比如串 0010100 可以通过先使用 $A \to 0A0$ 两次, 再使用 $A \to 1A1$ 一次, 再使用 $A \to 0$ 一次得到.

5.1.1 形式定义

定义. 上下文无关文法 *(CFG, Context-Free Grammar,* 简称文法*)* G 是一个四元组

$$G = (V, T, P, S),$$

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, 且 $V \cap T = \varnothing$*;*

3. P*:* 产生式 *(Production)* 的有穷集, 每个产生式包括*:*

i 一个变元, 称为产生式的头 *(head)* 或左部 *(left-hand side, LHS);*

ii 一个产生式符号 $\to$, 读作定义为;

iii 一个 $(V \cup T)^*$ 中的符号串, 称为体 *(body)* 或右部 *(right-hand side, RHS);*

4. $S \in V$: 初始符号 *(Start symbol)*, 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

$$A \to \alpha_1,\ A \to \alpha_2,\ \cdots,\ A \to \alpha_n$$

可简写为

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

- 文法中变元 A 的全体产生式, 称为 A 产生式

续例 1. 回文语言 $L_{\text{pal}} = \{w \in \{0,1\}^* \mid w = w^R\}$ 的文法可设计为

$$G = (\{A\}, \{0,1\}, \{A \to \varepsilon \mid 0 \mid 1 \mid 0A0 \mid 1A1\}, A).$$

在 $(V \cup T)^*$ 的串上使用 *(apply)* 产生式的过程, 就是将串中产生式左部的变元替换为产生式右部的串. 即如果有串 $\alpha, \beta, \gamma \in (V \cup T)^*$ 和变元 $A \in V$, 在串 $\alpha A \beta$ 上使用产生式 $A \to \gamma$ 后, 可得串 $\alpha\gamma\beta$.

- 上下文无关文法是形式化描述语言的又一种工具;
- 它表示语言的能力强于有穷自动机和正则表达式, 但也有无法它表示的语言;
- 基本思想是用变量表示串的集合;
- 递归的使用变量去定义变量, 因此善于表示嵌套的结构, 比如匹配的括号等;
- 变量之间仅使用了“连接”, 同一变量的不同定义使用了“并”.

字符使用的一般约定

- 终结符: $0, 1, \ldots,\ a, b, \ldots$
- 终结符串: $\ldots, w, x, y, z$
- 非终结符: $S, A, B, \ldots$
- 终结符或非终结符: $\ldots, X, Y, Z$
- 终结符或非终结符组成的串: $\alpha, \beta, \gamma, \ldots$

例 2. 简化版的算数表达式:

- 运算只有“加”和“乘”$(+, *)$, 参数仅为标识符;

- 标识符: 以 $\{a, b\}$ 开头由 $\{a, b, 0, 1\}$ 组成的字符串.

这样的表达式集合可用文法 G_{exp} 表示

$$G_{\text{exp}} = (\{E, I\}, \{a, b, 0, 1, +, *, (,)\}, P, E),$$

其中产生式集 P 中有 10 条产生式

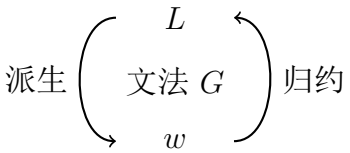
1. $E \to I$	5. $I \to a$	9. $I \to I0$
2. $E \to E + E$	6. $I \to b$	10. $I \to I1$
3. $E \to E * E$	7. $I \to Ia$	
4. $E \to (E)$	8. $I \to Ib$	

注意, 变元 I 所定义的标识符集合, 刚好是 $(\mathbf{a} + \mathbf{b})(\mathbf{a} + \mathbf{b} + \mathbf{0} + \mathbf{1})^*$.

5.1.2 归约和派生

非形式定义

从字符串到文法变元的分析过程, 称为**递归推理** (*recursive inference*) 或**归约** (*reduction*) ;
从文法变元到字符串的分析过程, 称为**推导**或**派生** (*derivation*).



- 归约: 自底向上 (*bottom up*), 由产生式的体向头的分析
- 派生: 自顶向下 (*top down*), 由产生式的头向体分析

续例 2. 用算数表达式文法 G_{exp}, 将 $a * (a + b00)$ 归约的过程.

1. $E \to I$
2. $E \to E + E$
3. $E \to E * E$
4. $E \to (E)$
5. $I \to a$
6. $I \to b$
7. $I \to Ia$
8. $I \to Ib$
9. $I \to I0$
10. $I \to I1$

	串归约到变元		应用产生式	重用结果
(1)	a	I	5. $I \to a$	–
(2)	b	I	5. $I \to b$	–
(3)	$b0$	I	9. $I \to I0$	(2)
(4)	$b00$	I	9. $I \to I0$	(3)
(5)	a	E	1. $E \to I$	(1)
(6)	$b00$	E	1. $E \to I$	(4)
(7)	$a + b00$	E	2. $E \to E + E$	(5), (6)
(8)	$(a + b00)$	E	4. $E \to (E)$	(7)
(9)	$a * (a + b00)$	E	3. $E \to E * E$	(5), (8)

派生和归约的形式定义

定义. *若 CFG $G=(V,T,P,S)$, 设 $\alpha,\beta,\gamma\in(V\cup T)^*$, $A\in V$, $A\to\gamma\in P$, 那么称在 G 中由 $\alpha A\beta$ 可派生出 $\alpha\gamma\beta$, 记为*

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

相应的, 称 $\alpha\gamma\beta$ 可归约为 $\alpha A\beta$.

- $\alpha A\beta \underset{G}{\Rightarrow} \alpha\gamma\beta$, 即用 $A\to\gamma$ 的右部 γ 替换串 $\alpha A\beta$ 中变元 A 得到串 $\alpha\gamma\beta$
- 如果语境中 G 是已知的, 可省略, 记为 $\alpha A\beta\Rightarrow\alpha\gamma\beta$
- 设 $\alpha_1,\alpha_2,\ldots,\alpha_m\in(V\cup T)^*$, $m\geq 1$, 对 $i=1,2,\ldots,m-1$ 如果有

  $$\alpha_i \underset{G}{\Rightarrow} \alpha_{i+1}$$

  成立, 即 α_1 经过零步或多步派生可得到 α_m

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

  那么, 记为

  $$\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m.$$

- 若 α 派生出 β 刚好经过了 i 步, 可记为

  $$\alpha \overset{i}{\underset{G}{\Rightarrow}} \beta.$$

续例 2. 算数表达式 $a*(a+b00)$ 在文法 $G_{\exp}$ 中的派生过程.

$$\begin{aligned}
&E\Rightarrow E*E\Rightarrow E*(E)\Rightarrow I*(E)\\
&\Rightarrow I*(E+E)\Rightarrow I*(E+I)\Rightarrow I*(I+I)\\
&\Rightarrow I*(a+I)\Rightarrow a*(a+I)\Rightarrow a*(a+I0)\\
&\Rightarrow a*(a+I00)\Rightarrow a*(a+b00)
\end{aligned}$$

5.1.3 最左派生和最右派生

定义. *为限制派生的随意性, 要求只替换符号串中最左边变元的派生过程, 称为最左派生 (left-most derivation), 记为*

$$\underset{\text{lm}}{\Rightarrow},\quad \overset{*}{\underset{\text{lm}}{\Rightarrow}},$$

只替换最右的, 称为最右派生 (right-most derivation), 记为

$$\underset{\text{rm}}{\Rightarrow},\quad \overset{*}{\underset{\text{rm}}{\Rightarrow}}.$$

- 任何派生都有等价的最左派生和最右派生

  $$A\overset{*}{\Rightarrow} w \text{ 当且仅当 } A\overset{*}{\underset{\text{lm}}{\Rightarrow}} w \text{ 当且仅当 } A\overset{*}{\underset{\text{rm}}{\Rightarrow}} w.$$

续例 2. 表达式 $a*(a+a)$ 在 G_{exp} 中的最左派生和最右派生分别为:

$$
\begin{aligned}
E &\underset{\text{lm}}{\Rightarrow} E*E \\
&\underset{\text{lm}}{\Rightarrow} I*E \\
&\underset{\text{lm}}{\Rightarrow} a*E \\
&\underset{\text{lm}}{\Rightarrow} a*(E) \\
&\underset{\text{lm}}{\Rightarrow} a*(E+E) \\
&\underset{\text{lm}}{\Rightarrow} a*(I+E) \\
&\underset{\text{lm}}{\Rightarrow} a*(a+E) \\
&\underset{\text{lm}}{\Rightarrow} a*(a+I) \\
&\underset{\text{lm}}{\Rightarrow} a*(a+a)
\end{aligned}
\qquad
\begin{aligned}
E &\underset{\text{rm}}{\Rightarrow} E*E \\
&\underset{\text{rm}}{\Rightarrow} E*(E) \\
&\underset{\text{rm}}{\Rightarrow} E*(E+E) \\
&\underset{\text{rm}}{\Rightarrow} E*(E+I) \\
&\underset{\text{rm}}{\Rightarrow} E*(E+a) \\
&\underset{\text{rm}}{\Rightarrow} E*(I+a) \\
&\underset{\text{rm}}{\Rightarrow} E*(a+a) \\
&\underset{\text{rm}}{\Rightarrow} I*(a+a) \\
&\underset{\text{rm}}{\Rightarrow} a*(a+a)
\end{aligned}
$$

5.1.4 文法的语言

定义. *CFG* $G=(V,T,P,S)$ 的语言定义为

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

那么符号串 w 在 $\mathbf{L}(G)$ 中, 要满足:

1. w 仅由终结符组成;
2. 初始符号 S 能派生出 w.

上下文无关语言

定义. 语言 L 是某个 *CFG* G 定义的语言, 即 $L=\mathbf{L}(G)$, 则称 L 为上下文无关语言 *(CFL, Context-Free Language).*

- 上下文无关是指在文法派生的每一步

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

符号串 γ 仅根据 A 的产生式派生, 而无需依赖 A 的上下文 α 和 β.

文法的等价性

定义. 如果有两个文法 *CFG* G_1 和 *CFG* G_2, 满足

$$\mathbf{L}(G_1)=\mathbf{L}(G_2),$$

则称 G_1 和 G_2 是等价的.

句型

定义. *若 CFG $G = (V, T, P, S)$, 初始符号 S 派生出来的符号串, 称为 G 的句型 (sentential form), 即*

$$\alpha \in (V \cup T)^* \text{ 且 } S \overset{*}{\Rightarrow} \alpha.$$

如果 $S \overset{*}{\underset{\text{lm}}{\Rightarrow}} \alpha$, 称 α 为左句型. 如果 $S \overset{*}{\underset{\text{rm}}{\Rightarrow}} \alpha$, 称 α 为右句型.

- 只含有终结符的句型, 也称为 G 的句子 (*sentence*)
- 而 $\mathbf{L}(G)$ 就是文法 G 全部的句子

5.1.5 文法设计举例

例 3. 给出语言 $L = \{w \in \{0,1\}^* \mid w \text{ contains at least three 1s}\}$ 的文法.

解: $S \to A1A1A1A$, $A \to 0A \mid 1A \mid \varepsilon$

例 4. 描述 CFG $G = (\{S\}, \{a, b\}, \{S \to aSb, S \to ab\}, S)$ 定义的语言?

解: $\mathbf{L}(G) = \{a^n b^n \mid n \geq 1\}$, 因为 $S \Rightarrow aSb \Rightarrow \cdots \Rightarrow a^{n-1}Sb^{n-1} \Rightarrow a^n b^n$.

例 5. 请为语言 $L = \{0^n 1^m \mid n \neq m\}$ 设计文法.

解:
$$\begin{array}{ll} S \to AC \mid CB & A \to A0 \mid 0 \\ C \to 0C1 \mid \varepsilon & B \to 1B \mid 1 \end{array}$$

例 6. 设计 $L_{\text{eq}} = \{w \in \{0,1\}^* \mid w \text{ 中 0 和 1 个数相等}\}$ 的文法.

解 1: $S \to 0S1 \mid 1S0 \mid SS \mid \varepsilon$, 寻找递归结构, 用变量构造递归结构;

解 2: $S \to S0S1S \mid S1S0S \mid \varepsilon$, “目标串”这样构成, 由变量定义变量.

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

解:
$$\begin{array}{l} S \to AB \\ A \to aAbb \mid \varepsilon \\ B \to bB \mid \varepsilon \end{array} \quad \text{或} \quad \begin{array}{l} S \to aSbb \mid B \\ B \to \varepsilon \mid bB \end{array}$$

程序设计语言的文法定义

- C — ISO C 1999 definition

...
selection-statement:
`if (` *expression* `)` *statement*
`if (` *expression* `)` *statement* `else` *statement*
`switch (` *expression* `)` *statement*
...

- Python — Full Grammar specification — https://docs.python.org/3/reference/grammar.html

...

```
try_stmt: ('try' ':' suite
           ((except_clause ':' suite)+
            ['else' ':' suite]
            ['finally' ':' suite] |
           'finally' ':' suite))
```

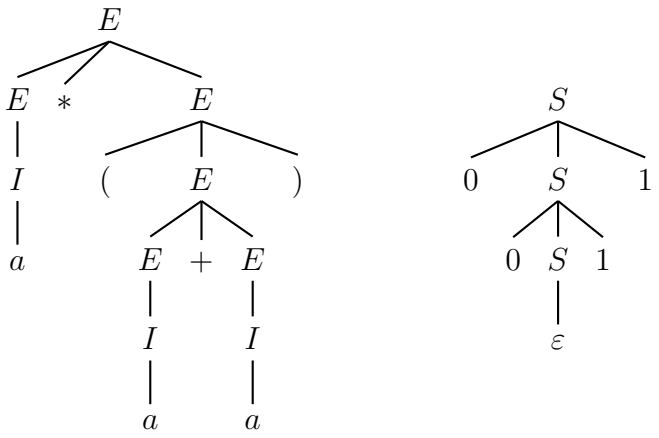
...

5.2 语法分析树

语法分析树与派生 (或归约) 紧密关联, 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. 语法分析树的一个重要应用, 语法的歧义性研究, 研究有关不适合用来作为程序设计语言的语法.

派生或归约的过程可以表示成树形结构.

- 例 2 文法 G_{exp} 中推导算数表达式 $a*(a+a)$ 的过程
- 例 6 中语言 L_{eq} 的文法中推导 0011 的过程



5.2.1 形式定义

定义. *CFG $G=(V,T,P,S)$ 的语法分析树 (parse tree, 简称语法树或派生树) 为:*

1. *每个内节点标记为 V 中的变元符号;*
2. *每个叶节点标记为 $V\cup T\cup\{\varepsilon\}$ 中的符号;*
3. *如果某内节点标记是 A, 其子节点从左至右分别为*

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

那么

$$A \to X_1 X_2 \cdots X_n \in P,$$

若有 $X_i = \varepsilon$, 则 ε 是 A 唯一子节点, 且 $A \to \varepsilon \in P$.

定义. 语法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. 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 $\mathbf{L}(G)$.

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

5.2.2 语法树和派生的等价性

定理 17. *CFG* $G = (V, T, P, S)$ 且 $A \in V$, 那么文法 G 中

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

当且仅当 G 中存在以 A 为根节点产物为 α 的语法树.

证明:
[充分性 $\Rightarrow$] 对派生 $A \overset{j}{\Rightarrow} \alpha$ 的步骤 j 数做归纳证明.

1. 归纳基础: 当 $j = 1$ 时, 即 $A \Rightarrow \alpha$, 由派生的定义, 一定有产生式 $A \to \alpha \in P$, 再由语法树定义, 可构造以 A 为根、产物为 α 的语法树 $\overset{A}{\underset{\alpha}{\uparrow}}$, 因此命题成立;

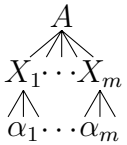
2. 归纳递推: 假设 $j \le n$ 时命题成立. 当 $i = n + 1$ 时, 则 $A \overset{n+1}{\Longrightarrow} \alpha$ 的派生过程为

   $$A \Rightarrow X_1 X_2 \cdots X_m \overset{n}{\Rightarrow} \alpha_1 \alpha_2 \cdots \alpha_m = \alpha,$$

   即派生的第 1 步, 一定首先由某产生式 $A \to X_1 X_2 \cdots X_m \in P$ 推导出, 其中每个 X_i 或为终结符或为变元, 而且每个变元的派生

   $$X_i \overset{*}{\Rightarrow} \alpha_i$$

   都不超过 n 步. 根据归纳假设, 每个 $X_i \overset{*}{\Rightarrow} \alpha_i$ 都有一棵以 X_i 为根、以 α_i 为产物的语法分析树 $\overset{X_i}{\underset{\alpha_i}{\uparrow}}$. 那么, 可以构造以 A 为根, 以 X_i 为子树 (或叶子) 的语法树, 其产物刚好为 $\alpha_1 \alpha_2 \cdots \alpha_m = \alpha$.

   因此命题成立.

[必要性 $\Leftarrow$] 对语法分析树的内部节点数 j 做归纳证明.

1. 归纳基础: 当 $j = 1$ 时, 该树一定是以 A 为根、α 为产物 $\overset{A}{\underset{\alpha}{\uparrow}}$, 由语法树定义, 产生式 $A \to \alpha \in P$, 那么 $A \overset{*}{\Rightarrow} \alpha$;

2. 归纳递推: 假设 $j \le n$ 时命题成立. 当 $j = n+1$ 时, 设根节点 A 的全部子节点从左至右分别为 $X_1, X_2, \ldots, X_m$, 那么显然 $A \to X_1X_2\cdots X_m \in P$, 且

$$A \Rightarrow X_1X_2\cdots X_m.$$

而每个 X_i 或为叶子或为 X_i 子树, 且 X_i 子树的内节点数都不超过 n, 根据归纳假设

$$X_i \overset{*}{\Rightarrow} \alpha_i.$$

又因为每个 X_i 子树的产物 α_i(或为叶子的 X_i 自身), 从左至右连接起来刚好为树的产物 α, 所以有

$$X_1X_2\cdots X_m \overset{*}{\Rightarrow} \alpha_1X_2\cdots X_m \overset{*}{\Rightarrow} \cdots \overset{*}{\Rightarrow} \alpha_1\alpha_2\cdots\alpha_m = \alpha.$$

因此 $A \Rightarrow X_1X_2\cdots X_m \overset{*}{\Rightarrow} \alpha$ 因此命题成立. □

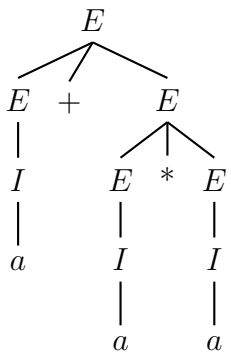
语法树唯一确定最左 (右) 派生

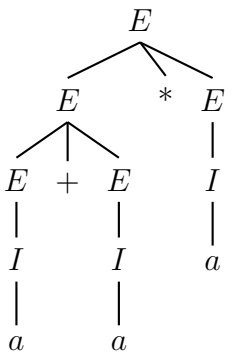
- 每棵语法分析树都有唯一的最左 (右) 派生
- 给定 CFG $G = (V, T, P, S)$, $A \in V$, 以下命题等价:
  1. 通过递归推理, 确定串 w 在变元 A 的语言中.
  2. 存在以 A 为根节点, 产物为 w 的语法分析树.
  3. $A \overset{*}{\Rightarrow} w$.
  4. $A \overset{*}{\underset{\text{lm}}{\Rightarrow}} w$.
  5. $A \overset{*}{\underset{\text{rm}}{\Rightarrow}} w$.

5.3 文法和语言的歧义性

定义. 如果 *CFG* G 使某些符号串有两棵不同的语法分析树, 则称文法 G 是歧义 *(ambiguity)* 的.

续例 2. 算数表达式的文法 G_{exp} 中, 对句型 $a + a * a$ 有下面两棵语法分析树:





(1) $E \Rightarrow E + E$
$\Rightarrow E + E * E$
$\overset{*}{\Rightarrow} a + a * a$

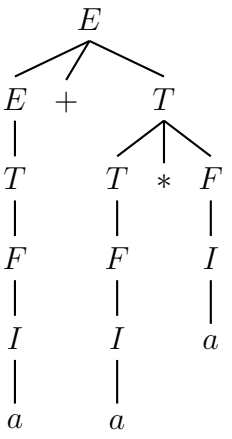
(2) $E \Rightarrow E * E$
$\Rightarrow E + E * E$
$\overset{*}{\Rightarrow} a + a * a$

5.3.1 文法歧义性的消除

有些文法的歧义性, 可以通过重新设计文法来消除.

续例 2. 文法 G_{exp} 重新设计为文法 $G_{\text{exp}'}$ 可消除歧义.

$$
\begin{aligned}
&E \to E + T \mid T\\
&T \to T * F \mid F\\
&F \to (E) \mid I\\
&I \to a\\
&I \to b\\
&I \to Ia\\
&I \to Ib\\
&I \to I0\\
&I \to I1
\end{aligned}
$$



5.3.2 语言的固有歧义性

定义. *定义同样的语言可以有多个文法, 如果 CFL L 的所有文法都是歧义的, 那么称语言 L 是固有歧义 (Inherent Ambiguity) 的.*

- 固有歧义的语言确实存在, 如语言

  $$L = \{a^i b^j c^k \mid i = j \text{ or } j = k\}$$

  中任何形为 $a^n b^n c^n$ 的串, 总会有两棵语法树.

- “判定任何给定 CFG G 是否歧义”是一个不可判定问题.

5.4 文法的化简与范式

为什么要化简

- 典型问题: 给定 CFG G 和串 w, 判断 $w \in \mathbf{L}(G)$?
- 编译器设计和自然语言处理的基本问题之一
- 但文法的形式非常自由, 过于复杂不易于自动处理
- 以不改变语言为前提, 化简文法和限制文法的格式

例 7. 如下文法中, 有无意义的变元和产生式

$$
\begin{aligned}
&S \to 0DS1D \mid B \mid \varepsilon \\
&B \to BC1 \mid 0CBC \\
&A \to A0 \mid A1 \mid \varepsilon \\
&C \to D \\
&D \to \varepsilon
\end{aligned}
$$

在这个文法中: B 无法生成全部为终结符的串; A 无法从 S 派生出来; $C \to D$ 在推导过程中仅增加了推导的长度; C 和 D 能派生出空串, 为其他串 w 的归约增加了难度. 所有这些对文法定义的语言没有贡献, 需要删除掉来简化文法.

文法的化简

1. 消除无用符号 (*useless symbols*): 对文法定义语言没有贡献的符号
2. 消除 ε 产生式 (ε-*productions*): $A \to \varepsilon$ (得到语言 $L - \{\varepsilon\}$)
3. 消除单元产生式 (*unit productions*): $A \to B$

无用符号, 从文法开始符号派生到终结符串的过程中用不到; ε-产生式, 除了空串 ε 自身, 没有贡献语言中其他的串; 单元产生式, 仅仅增加了推导 (或归约) 的步骤. 这三者对语言的定义都没有实质的作用.

5.4.1 消除无用符号

无用符号

定义. *CFG* $G = (V, T, P, S)$, *符号* $X \in (V \cup T)$*:*

1. *如果* $S \overset{*}{\Rightarrow} \alpha X \beta$, *称* X *是可达的 (reachable);*
2. *如果* $\alpha X \beta \overset{*}{\Rightarrow} w$ $(w \in T^*)$, *称* X *是产生的 (generating);*
3. *如果* X *同时是产生的和可达的, 即*

$$S \overset{*}{\Rightarrow} \alpha X \beta \overset{*}{\Rightarrow} w \quad (w \in T^*),$$

则称 X *是有用的, 否则称* X *为无用符号.*

消除无用符号

消除无用符号: 删除全部含有 “非产生的” 和 “非可达的” 符号的产生式

计算 “产生的” 符号集

1. 每个 T 中的符号都是产生的;
2. $A \to \alpha \in P$ 且 α 中符号都是产生的, 则 A 是产生的.

计算“可达的”符号集

1. 符号 S 是可达的;
2. $A \to \alpha \in P$ 且 A 是可达的, 则 α 中符号都是可达的.

定理 18. *每个非空的 CFL 都能被一个不带无用符号的 CFG 定义.*

注意

- 先寻找并消除全部非“产生的”符号
- 再寻找并消除全部非“可达的”符号
- 否则可能消除不完整

例 8. 消除如下文法无用符号

$S \to AB \mid a$
$A \to b$

先消除非产生的
$S \to a$
$A \to b$

再消除非可达的
$S \to a$

5.4.2 消除 ε-产生式

定义. *文法中形如 $A \to \varepsilon$ 的产生式称为ε-产生式.*

如果变元 $A \overset{*}{\Rightarrow} \varepsilon$, 称 A 是可空的.

- ε-产生式在文法定义语言时, 除产生空串外没有其他帮助
- 对于 CFL L, 消除其文法中全部的 ε-产生式后, 得到语言 $L - \{\varepsilon\}$

确定“可空变元”

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.

替换带有可空变元的产生式

将含有可空变元的一条产生式 $A \to X_1X_2\cdots X_n$, 用一组产生式 $A \to Y_1Y_2\cdots Y_n$ 代替, 其中:

1. 若 X_i 不是可空的, Y_i 为 X_i;
2. 若 X_i 是可空的, Y_i 为 X_i 或 ε;
3. 但 Y_i 不能全为 ε.

定理 19. *任何 CFG G, 都存在一个不带无用符号和 ε-产生式的 CFG G', 使 $\mathbf{L}(G') = \mathbf{L}(G) - \{\varepsilon\}$.*

例 9. 消除 CFG $G = (\{S, A, B\}, \{a, b\}, P, S)$ 的 ε-产生式.

$$
\begin{aligned}
&S \to AB\\
&A \to AaA \mid \varepsilon\\
&B \to BbB \mid \varepsilon
\end{aligned}
$$

解: CFG G' 为

$$
\begin{aligned}
&S \to AB \mid A \mid B\\
&A \to AaA \mid Aa \mid aA \mid a\\
&B \to BbB \mid Bb \mid bB \mid b
\end{aligned}
$$

5.4.3 消除单元产生式

确定“单元对”

如果有 $A \overset{*}{\Rightarrow} B$, 则称 $[A, B]$ 为单元对.

1. $A \to B \in P$, 则 $[A, B]$ 是单元对;
2. 若 $[A, B]$ 和 $[B, C]$ 都是单元对, 则 $[A, C]$ 是单元对.

消除单元产生式

1. 删除全部形为 $A \to B$ 的单元产生式;
2. 对每个单元对 $[A, B]$, 将 B 的产生式复制给 A.

定理 20. *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.*

例 10. 消除文法的单元产生式

$$
\begin{aligned}
&S \to A \mid B \mid 0S1\\
&A \to 0A \mid 0\\
&B \to 1B \mid 1
\end{aligned}
$$

解: 单位对为 $[S, A]$ 和 $[S, B]$, 带入得:

$$\begin{array}{ll} S \to 0S1 & A \to 0A \mid 0 \\ S \to 0A \mid 0 & B \to 1B \mid 1 \\ S \to 1B \mid 1 & \end{array}$$

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.

限制文法格式

将任意形式的文法转换为:

1. 乔姆斯基范式 (CNF, Chomsky Normal Form)
2. 格雷巴赫范式 (GNF, Greibach Normal Form)

5.4.4 乔姆斯基范式

定理 21 (乔姆斯基范式 CNF)**.** *每个不带 ε 的 CFL 都可以由这样的 CFG G 定义, G 中每个产生式的形式都为*

$$A \to BC \quad 或 \quad A \to a$$

这里的 A, B 和 C 是变元, a 是终结符.

- 利用 CNF 派生长度为 n 的串, 刚好需要 $2n-1$ 步
- 因此存在算法判断任意字符串 w 是否在给定的 CFL 中
- 利用 CNF 的多项式时间解析算法 — CYK 算法

CFG 转为 CNF 的方法

1. 将产生式

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

中每个终结符 a 替换为新变元 C_a,

2. 增加新产生式

$$C_a \to a,$$

3. 引入新变元 $D_1, D_2, \cdots, D_{m-2}$, 将产生式

$$A \to B_1 B_2 \cdots B_m \ \ (m \geq 2)$$

替换为一组级联产生式

$$\begin{aligned} A &\to B_1 D_1 \\ D_1 &\to B_2 D_2 \\ &\cdots \\ D_{m-2} &\to B_{m-1} B_m. \end{aligned}$$

例 11. CFG $G = (\{S, A, B\}, \{a, b\}, P, S)$, 产生式集合 P 为:

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

请设计等价的 CNF 文法.

解: CNF 为

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

5.4.5 格雷巴赫范式

定理 22 (格雷巴赫范式 GNF)**.** *每个不带 ε 的 CFL 都可以由这样的 CFG G 定义, G 中每个产生式的形式都为*

$$A \to a\alpha$$

其中 A 是变元, a 是终结符, α 是零或多个变元的串.

- GNF 每个产生式都会引入一个终结符
- 长度为 n 的串的派生恰好是 n 步

例 12. 将以下文法转换为 GNF.

$$\begin{aligned} S &\to AB \\ A &\to aA \mid bB \mid b \\ B &\to b \end{aligned}$$

解: GNF 为

$$\begin{aligned} S &\to aAB \mid bBB \mid bB \\ A &\to aA \mid bB \mid b \\ B &\to b \end{aligned}$$

直接左递归

定义. 文法中形式为 $A \to A\alpha$ 的产生式, 称为直接左递归 *(immediate left-recursion).*

消除直接左递归

1. 若 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;
2. 引入新变元 B, 并用如下产生式替换
$$A \to \beta_1 \mid \beta_2 \mid \cdots \mid \beta_m \mid \beta_1 B \mid \beta_2 B \mid \cdots \mid \beta_m B$$
$$B \to \alpha_1 \mid \alpha_2 \mid \cdots \mid \alpha_n \mid \alpha_1 B \mid \alpha_2 B \mid \cdots \mid \alpha_n B$$

间接左递归

定义. 文法中如果有形式为
$$A \to B\alpha \mid \ldots$$
$$B \to A\beta \mid \ldots$$
的产生式, 称为间接左递归 *(indirect left-recursion).*

- 会有 $A \Rightarrow B\alpha \Rightarrow A\beta\alpha$, 无法通过代换消除递归

消除间接左递归

1. 将文法中变元重命名为 $A_1, A_2, \cdots, A_n$;
2. 通过代入, 使产生式都形如
$$A_i \to A_j\alpha$$
$$A_i \to a\alpha$$
但要求 $i \leq j$;
3. 消除直接左递归 $A_i \to A_i\beta$, 再代入其他产生式.

例 13. Convert the following grammar to GNF.

$$S \to AB$$
$$A \to BS \mid b$$
$$B \to SA \mid a$$

解: 1. 重命名变元, 代换 $i > j$ 的 A_j
$$A_1 \to A_2A_3$$

$$A_2 \to A_3A_1 \mid b$$
$$A_3 \to a \mid \cancel{A_1A_2} \mid \cancel{A_2A_3A_2} \mid A_3A_1A_3A_2 \mid bA_3A_2$$

2. 消除直接左递归

$$A_1 \to A_2A_3$$
$$A_2 \to A_3A_1 \mid b$$
$$A_3 \to bA_3A_2 \mid a \mid bA_3A_2B_1 \mid aB_1$$
$$B_1 \to A_1A_3A_2 \mid A_1A_3A_2B_1$$

3. A_3 代入到 A_2, A_2 代入到 A_1, A_1 代入 B_1

$$A_3 \to bA_3A_2 \mid a \mid bA_3A_2B_1 \mid aB_1$$
$$A_2 \to bA_3A_2A_1 \mid aA_1 \mid bA_3A_2B_1A_1 \mid aB_1A_1 \mid b$$
$$A_1 \to bA_3A_2A_1A_3 \mid aA_1A_3 \mid bA_3A_2B_1A_1A_3 \mid aB_1A_1A_3 \mid bA_3$$
$$B_1 \to bA_3A_2A_1A_3A_3A_2 \mid aA_1A_3A_3A_2 \mid bA_3A_2B_1A_1A_3A_3A_2 \mid aB_1A_1A_3A_3A_2 \mid bA_3A_3A_2 \mid bA_3A_2A_1A_3A_3A_2B_1 \mid aA_1A_3A_3A_2B_1 \mid bA_3A_2B_1A_1A_3A_3A_2B_1 \mid aB_1A_1A_3A_3A_2B_1 \mid bA_3A_3A_2B_1$$

例. Convert the following grammar to GNF.

$$A_1 \to A_2bA_3 \mid aA_1$$
$$A_2 \to A_3cA_3 \mid b$$
$$A_3 \to A_1cA_3 \mid A_2bb \mid a$$

例. [习题 7.1.1] 消除无用符号

$$S \to AB \mid CA$$
$$A \to a$$
$$B \to AB$$
$$C \to ab \mid b$$

例. [习题 7.1.2] Begin with the grammar:

$$S \to ASB \mid \varepsilon$$
$$A \to aAS \mid a$$
$$B \to SbS \mid A \mid bb$$

a) Eliminate $\varepsilon-$productions.

b) Eliminate any unit productions in the resulting grammar.

c) Eliminate any useless symbols in the resulting grammar.

d) Put the resulting grammar into Chomsky Normal Form.