

语言的定义

推导和归约

- 给定文法 $G=(V_T, V_N, P, S)$, 如果 $\alpha \rightarrow \beta \in P$, 那么可以将符号串 $\gamma\alpha\delta$ 中的 α 替换为 β , 也就是说, 将 $\gamma\alpha\delta$ 重写(rewrite)为 $\gamma\beta\delta$, 记作 $\gamma\alpha\delta \Rightarrow \gamma\beta\delta$ 。此时, 称文法中的符号串 $\gamma\alpha\delta$ 直接推导(directly derive)出 $\gamma\beta\delta$
- 简而言之, 就是用产生式的右部替换产生式的左部

➤ 如果 $\alpha_0 \Rightarrow \alpha_1, \alpha_1 \Rightarrow \alpha_2, \alpha_2 \Rightarrow \alpha_3, \dots, \alpha_{n-1} \Rightarrow \alpha_n$, 则可以记作 $\alpha_0 \Rightarrow \alpha_1 \Rightarrow \alpha_2 \Rightarrow \alpha_3 \Rightarrow \dots \Rightarrow \alpha_{n-1} \Rightarrow \alpha_n$, 称符号串 α_0 经过 n 步推导出 α_n , 可简记为 $\alpha_0 \Rightarrow^n \alpha_n$

➤ $\alpha \Rightarrow^0 \alpha$

➤ \Rightarrow^+ 表示 “经过正数步推导”

➤ \Rightarrow^* 表示 “经过若干 (可以是0) 步推导”

句型 and 句子

➤ 如果 $S \Rightarrow^* \alpha, \alpha \in (V_T \cup V_N)^*$, 则称 α 是 G 的一个句型(sentential form)

➤ 一个句型中既可以包含终结符, 又可以包含非终结符, 也可能是空串

➤ 如果 $S \Rightarrow^* w, w \in V_T^*$, 则称 w 是 G 的一个句子(sentence)

➤ 句子是不包含非终结符的句型

语言的形式化定义

➤ 由文法 G 的开始符号 S 推导出的所有句子构成的集合称为文法 G 生成的语言, 记为 $L(G)$ 。
即

$$L(G) = \{w \mid S \Rightarrow^* w, w \in V_T^*\}$$

语言上的运算

运算	定义和表示
L 和 M 的并	$L \cup M = \{s \mid s \text{ 属于 } L \text{ 或者 } s \text{ 属于 } M\}$
L 和 M 的连接	$LM = \{st \mid s \text{ 属于 } L \text{ 且 } t \text{ 属于 } M\}$
L 的幂	$\begin{cases} L^0 = \{\varepsilon\} \\ L^n = L^{n-1}L, n \geq 1 \end{cases}$
L 的Kleene闭包	$L^* = \bigcup_{i=0}^{\infty} L^i$
L 的正闭包	$L^+ = \bigcup_{i=1}^{\infty} L^i$

例：令 $L = \{A, B, \dots, Z, a, b, \dots, z\}$, $D = \{0, 1, \dots, 9\}$ 。则 $L(L \cup D)^*$ 表示的语言是标识符

文法的分类

0 型文法 (Type-0 Grammar)

$$\alpha \rightarrow \beta$$

- 无限制文法(Unrestricted Grammar)/短语结构文法(Phrase Structure Grammar, PSG)
- $\forall \alpha \rightarrow \beta \in P$, α 中至少包含1个非终结符
- 0型语言
- 由0型文法 G 生成的语言 $L(G)$

1 型文法 (Type-1 Grammar)

$$\alpha \rightarrow \beta$$

- 上下文有关文法(Context-Sensitive Grammar, CSG)
- $\forall \alpha \rightarrow \beta \in P$, $|\alpha| \leq |\beta|$
- 产生式的一般形式: $\alpha_1 A \alpha_2 \rightarrow \alpha_1 \beta \alpha_2$ ($\beta \neq \varepsilon$)
- 上下文有关语言 (1型语言)
- 由上下文有关文法(1型文法) G 生成的语言 $L(G)$

2 型文法 (Type-2 Grammar)

$$\alpha \rightarrow \beta$$

上下文无关文法 (Context-Free Grammar, CFG)

➤ $\forall \alpha \rightarrow \beta \in P, \alpha \in V_N$

➤ 产生式的一般形式: $A \rightarrow \beta$

上下文无关语言 (2型语言)

➤ 由上下文无关文法(2型文法) G 生成的语言 $L(G)$

3 型文法 (Type-3 Grammar)

$$\alpha \rightarrow \beta$$

➤ 正则文法 (Regular Grammar, RG)

➤ 右线性 (Right Linear) 文法: $A \rightarrow wB$ 或 $A \rightarrow w$

➤ 左线性 (Left Linear) 文法: $A \rightarrow Bw$ 或 $A \rightarrow w$

➤ 左线性文法和右线性文法都称为正则文法

例 (右线性文法)

① $S \rightarrow a | b | c | d$

② $S \rightarrow aT | bT | cT | dT$

③ $T \rightarrow a | b | c | d | 0 | 1 | 2 | 3 | 4 | 5$

④ $T \rightarrow aT | bT | cT | dT | 0T | 1T | 2T | 3T | 4T | 5T$

文法 G (上下文无关文法)

① $S \rightarrow L | LT$

② $T \rightarrow L | D | TL | TD$

③ $L \rightarrow a | b | c | d$

④ $D \rightarrow 0 | 1 | 2 | 3 | 4 | 5$

CFG的分析树

➤ 根节点的标号为文法开始符号

➤ 内部结点表示对一个产生式 $A \rightarrow \beta$ 的应用, 该结点的标号是此产生式左部 A 。该结点的子结点的标号从左到右构成了产生式的右部 β

➤ 叶结点的标号既可以是非终结符, 也可以是终结符。从左到右排列叶节点得到的符号串称为是这棵树的产出 (yield) 或边缘 (frontier)

分析树是推导的图形化表示

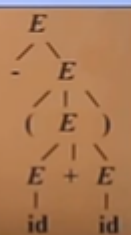
➤ 给定一个推导 $S \Rightarrow a_1 \Rightarrow a_2 \Rightarrow \dots \Rightarrow a_n$ ，对于推导过程中得到的每一个句型 a_i ，都可以构造出一个边缘为 a_i 的分析树

推导过程: $E \Rightarrow -E \Rightarrow -(E) \Rightarrow -(E+E) \Rightarrow -(id+E) \Rightarrow -(id+id)$

文法:

- ① $E \rightarrow E + E$
- ② $E \rightarrow E * E$
- ③ $E \rightarrow -E$
- ④ $E \rightarrow (E)$
- ⑤ $E \rightarrow id$

分析树:



(句型的) 短语

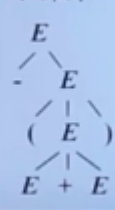
➤ 给定一个句型，其分析树中的每一棵子树的边缘称为该句型的一个短语 (phrase)

➤ 如果子树只有父子两代结点，那么这棵子树的边缘称为该句型的一个直接短语 (immediate phrase)

文法:

- ① $E \rightarrow E + E$
- ② $E \rightarrow E * E$
- ③ $E \rightarrow -E$
- ④ $E \rightarrow (E)$
- ⑤ $E \rightarrow id$

分析树:



短语:

- $-(E+E)$
- $(E+E)$
- $E+E$

直接短语:

- $E+E$

直接短语一定是某产生式的右部
但产生式的右部不一定是给定句型的直接短语

二义性文法 (Ambiguous Grammar)

- 如果一个文法可以为某个句子生成多棵分析树，则称这个文法是二义性的

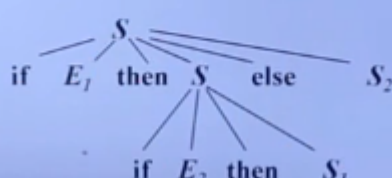
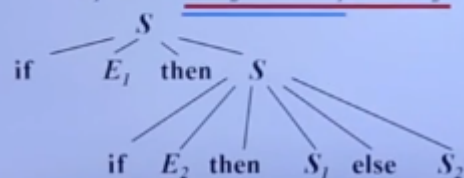
例

➤ 文法

- $S \rightarrow$ if E then S
- | if E then S else S
- | other

➤ 句型

- if E_1 then if E_2 then S_1 else S_2



二义性文法的判定

- 对于任意一个上下文无关文法，不存在一个算法，判定它是无二义性的；但能给出一组充分条件，满足这组充分条件的文法是无二义性的
 - 满足，肯定无二义性
 - 不满足，也未必就是有二义性的

