

形式语言与自动机理论

上下文无关文法

王春宇

计算机科学与技术学院
哈尔滨工业大学

上下文无关文法

- 上下文无关文法
 - 形式定义
 - 归约和派生
 - 最左派生和最右派生
 - 文法的语言
- 语法分析树
- 文法和语言的歧义性
- 文法的化简与范式

自然语言的文法

$\langle \text{sentence} \rangle \rightarrow \langle \text{noun-phrase} \rangle \langle \text{verb-phrase} \rangle$

$\langle \text{noun-phrase} \rangle \rightarrow \langle \text{article} \rangle \langle \text{noun} \rangle \mid \langle \text{article} \rangle \langle \text{adjective} \rangle \langle \text{noun} \rangle$

$\langle \text{verb-phrase} \rangle \rightarrow \langle \text{verb} \rangle \mid \langle \text{verb} \rangle \langle \text{noun-phrase} \rangle$

$\langle \text{article} \rangle \rightarrow \text{a} \mid \text{the}$

$\langle \text{noun} \rangle \rightarrow \text{boy} \mid \text{girl} \mid \text{cat}$

$\langle \text{adjective} \rangle \rightarrow \text{big} \mid \text{small} \mid \text{blue}$

$\langle \text{verb} \rangle \rightarrow \text{sees} \mid \text{likes}$

...

自然语言的文法

使用语法规则产生句子:

$\langle sentence \rangle \Rightarrow \langle noun\text{-}phrase \rangle \langle verb\text{-}phrase \rangle$

$\Rightarrow \langle article \rangle \langle noun \rangle \langle verb\text{-}phrase \rangle$

$\Rightarrow \langle article \rangle \langle noun \rangle \langle verb \rangle \langle noun\text{-}phrase \rangle$

$\Rightarrow \langle article \rangle \langle noun \rangle \langle verb \rangle \langle article \rangle \langle adjective \rangle \langle noun \rangle$

$\Rightarrow \text{the} \langle noun \rangle \langle verb \rangle \langle article \rangle \langle adjective \rangle \langle noun \rangle$

$\Rightarrow \text{the girl} \langle verb \rangle \langle article \rangle \langle adjective \rangle \langle noun \rangle$

$\Rightarrow \dots$

$\Rightarrow \underline{\text{the girl sees a blue cat}}$

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:

- 首先 $\varepsilon, 0, 1$ 都是回文
- 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:

$$A \rightarrow \varepsilon \qquad A \rightarrow 0A0$$

$$A \rightarrow 0 \qquad A \rightarrow 1A1$$

$$A \rightarrow 1$$

0110

0A0

01A10

01ε10 ⇒ 0110

$$\begin{cases} i=0 \\ i=i+1 \end{cases} \Rightarrow 3$$

$$\Rightarrow i$$

$$i+1$$

$$i+1+1$$

$$i+1+1+1=3$$

文法

上下文无关文法的形式定义

定义

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

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

$$A = (\overset{Q}{\Sigma}, \overset{I}{\Gamma}, \overset{J}{\Delta}, \overset{F}{S})$$

① V : 变元的有穷集, 变元也称为非终结符或语法范畴;

② T : 终结符的有穷集, 且 $V \cap T = \emptyset$;

③ P : 产生式的有穷集, 每个产生式包括:

i 一个变元, 称为产生式的头或左部;

ii 一个产生式符号 \rightarrow , 读作定义为;

iii 一个 $(V \cup T)^*$ 中的符号串, 称为体或右部;

④ $S \in V$: 初始符号, 文法开始的地方.

V, T, P, S

- 产生式 $A \rightarrow \alpha$, 读作 A 定义为 α
- 如果有多个 A 的产生式

$$A \rightarrow \alpha_1, A \rightarrow \alpha_2, \dots, A \rightarrow \alpha_n$$

可简写为

$$A \rightarrow \alpha_1 \mid \alpha_2 \mid \dots \mid \alpha_n$$

- 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

$$\underline{G} = (\overset{\vee}{\{A\}}, \overset{\top}{\{0, 1\}}, \overset{P}{\{A \rightarrow \underline{\varepsilon} \mid \underline{0} \mid \underline{1} \mid \underline{0A0} \mid \underline{1A1}\}}, \overset{S}{A}).$$

字符使用的一般约定

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

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

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

其中产生式集 P 中有 10 条产生式

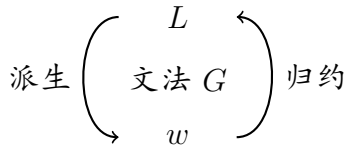
- | | | |
|--------------------------|-----------------------|------------------------|
| 1. $E \rightarrow I$ | 5. $I \rightarrow a$ | 9. $I \rightarrow I0$ |
| 2. $E \rightarrow E + E$ | 6. $I \rightarrow b$ | 10. $I \rightarrow I1$ |
| 3. $E \rightarrow E * E$ | 7. $I \rightarrow Ia$ | |
| 4. $E \rightarrow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

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

归约和派生

非形式定义

从字符串到文法变元的分析过程, 称为递归推理或归约;
从文法变元到字符串的分析过程, 称为推导或派生.



- 归约: 自底向上, 由产生式的体向头的分析
- 派生: 自顶向下, 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.

	串归约到变元		应用产生式	重用结果	
1. $E \rightarrow I$	(1)	a	I	5. $I \rightarrow a$	—
2. $E \rightarrow E + E$	(2)	b	I	<u>5.</u> $I \rightarrow b$	— $I \neq (I + I00)$
3. $E \rightarrow E * E$	(3)	<u>$b0$</u>	<u>I</u>	<u>9.</u> $I \rightarrow I0$	(2)
4. $E \rightarrow (E)$	(4)	<u>$b00$</u>	I	9. $I \rightarrow I0$	(3) $I \neq (I + I)$
5. $I \rightarrow a$	(5)	<u>a</u>	E	1. $E \rightarrow I$	(1) $E \neq (E + E)$
6. $I \rightarrow b$	(6)	$b00$	E	1. $E \rightarrow I$	(4)
7. $I \rightarrow Ia$	(7)	$a + b00$	E	2. $E \rightarrow E + E$	(5), (6) $\downarrow E$
8. $I \rightarrow Ib$	(8)	$(a + b00)$	E	4. $E \rightarrow (E)$	(7)
9. $I \rightarrow I0$	(9)	$a * (a + b00)$	E	3. $E \rightarrow E * E$	(5), (8)
10. $I \rightarrow I1$					

派生和归约的形式定义

定义

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

$$\alpha A \beta \xRightarrow{G} \alpha \gamma \beta.$$

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

- $\alpha A \beta \xRightarrow{G} \alpha \gamma \beta$, 即用 $A \rightarrow \gamma$ 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 $\alpha_1, \dots, \alpha_m \in (V \cup T)^*$, $m \geq 1$, 对 $i = 1, \dots, m-1$ 如果有

$$\alpha_i \xRightarrow{G} \alpha_{i+1}$$

成立, 即 α_1 经过零步或多步派生可得到 α_m

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

那么, 记为

$$\alpha_1 \xRightarrow[G]{*} \alpha_m.$$

- 若 α 派生出 β 刚好经过了 i 步, 可记为

$$\alpha \xRightarrow[G]{i} \beta.$$

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

$$E \Rightarrow E * E \Rightarrow \underline{E * (E)} \Rightarrow I * (E)$$

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

$$\Rightarrow I * (a + I) \Rightarrow a * (a + I) \Rightarrow a * (a + I0)$$

$$\Rightarrow a * (a + I00) \Rightarrow a * (a + b00)$$

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, 记为

$$\underline{\Rightarrow_{lm}}, \quad \underline{\overset{*}{\Rightarrow_{lm}}},$$

只替换最右的, 称为最右派生, 记为

$$\underline{\Rightarrow_{rm}}, \quad \underline{\overset{*}{\Rightarrow_{rm}}}.$$

- 任何派生都有等价的最左派生和最右派生

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

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:

1. $E \rightarrow I$	$E \xRightarrow{\text{lm}} E * E$	$E \xRightarrow{\text{rm}} E * E$
2. $E \rightarrow E + E$	$\xRightarrow{\text{lm}} I * E$	$\xRightarrow{\text{rm}} E * (E)$
3. $E \rightarrow E * E$	$\xRightarrow{\text{lm}} a * E$	$\xRightarrow{\text{rm}} E * (E + E)$
4. $E \rightarrow (E)$	$\xRightarrow{\text{lm}} a * (E)$	$\xRightarrow{\text{rm}} E * (E + I)$
5. $I \rightarrow a$	$\xRightarrow{\text{lm}} a * (E + E)$	$\xRightarrow{\text{rm}} E * (E + a)$
6. $I \rightarrow b$	$\xRightarrow{\text{lm}} a * (I + E)$	$\xRightarrow{\text{rm}} E * (I + a)$
7. $I \rightarrow Ia$	$\xRightarrow{\text{lm}} a * (a + E)$	$\xRightarrow{\text{rm}} E * (a + a)$
8. $I \rightarrow Ib$	$\xRightarrow{\text{lm}} a * (a + I)$	$\xRightarrow{\text{rm}} I * (a + a)$
9. $I \rightarrow I0$	$\xRightarrow{\text{lm}} a * (a + a)$	$\xRightarrow{\text{rm}} a * (a + a)$
10. $I \rightarrow I1$		

文法的语言

定义

CFG $G = (V, T, P, S)$ 的语言定义为

$$\underline{L(G)} = \{w \mid \underline{w \in T^*}, \underline{S \xRightarrow{*}_G w}\}.$$

那么符号串 w 在 $L(G)$ 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 $L = L(G)$, 则称 L 为 上下文无关语言 (CFL, Context-Free Language). CFL

- 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

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

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, 即

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

如果 $S \xRightarrow[\text{lm}]{*} \alpha$, 称 α 为左句型.

如果 $S \xRightarrow[\text{rm}]{*} \alpha$, 称 α 为右句型.

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

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

$$A \mid A1A \mid A1, A \rightarrow 0A \mid 1A \mid \varepsilon$$

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

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

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

$$V = S, T = ab$$

$$aSb, ab,$$

{ left multiple
right multiple }

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

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

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

① $n > m$

00011


0A, A

00 0011


$S \rightarrow AC, A \rightarrow A0 \mid 0, C \rightarrow 0C \mid \varepsilon$

解:

$\begin{cases} S \rightarrow AC \mid CB \\ C \rightarrow 0C1 \mid \varepsilon \end{cases}$

$A \rightarrow A0 \mid 0$

$B \rightarrow 1B \mid 1$

② $n < m$

B 1B11


$C \rightarrow 0C1 \mid \varepsilon$
 CB

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

$$\checkmark S \rightarrow 0S1 \mid 1S0 \mid \varepsilon \mid SS$$

$$\checkmark S \rightarrow S0S1S \mid S1S0S \mid \varepsilon$$

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

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

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

...

A → if(B)C | if(B)C else C | switch(C)

- Python — Full Grammar specification

...

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

...