计算理论导论

习题四: 上下文无关文法

中国人民大学 信息学院 崔冠宇 2018202147

1. Find the language described by the following context-free grammars:

(a) $\mathrm{S} \to \mathtt{abS} \mid \mathtt{a}$

(b)

- $\mathrm{S} \to \mathtt{bSb} \mid \mathrm{A}$
- $\mathrm{A} \to \mathtt{aA} \mid \varepsilon$

(c)

- $\mathrm{S} \to \mathtt{aSa} \mid \mathtt{bSb} \mid \mathrm{L}$
- $\mathrm{L} \to \mathrm{a} \mid \mathrm{b} \mid \varepsilon$

(d)

- $\mathrm{S} \to \mathrm{AS} \mid \mathrm{B}$
- $\mathrm{A} \to \mathrm{aAc} \mid \mathrm{Aa} \mid \varepsilon$
- $\mathrm{B} \to \mathrm{bBb} \mid \varepsilon$

解:

(a) 思路: 注意到推导任意字符串时，先任意次使用第一个产生式，最后使用一次第二个产生式。

任意次使用第一个产生式推导，使得中间串具有 $(\mathtt{ab})^n\mathrm{S}$ 的形式，最后使用第二个产生式将 S 替换为 $\mathtt{a}$。于是 $L(G) = \{w|w = (\mathtt{ab})^n\mathtt{a}, n \geq 0\}$（进一步地，这是一个正则语言）。

(b) 思路: 分层次自顶向下看。推导一个字符串时，先任意次使用第一个产生式，然后使用一次第二个产生式，再任意次使用第三个产生式，最后使用一次第四个产生式。

反复使用第一个产生式，然后使用第二个产生式将使中间串具有 $\mathtt{b}^n\mathrm{A}\mathtt{b}^n (n \geq 0)$ 的形式；再反复使用第三个产生式，最后使用第四个产生式将使串具有 $\mathtt{b}^n a^m \mathtt{b}^n (m, n \geq 0)$ 的形式。所以 $L(G) = \{w | w = \mathtt{b}^n\mathtt{a}^m\mathtt{b}^n, m, n \geq 0\}$ （进一步地，可以用泵引理证明这不是一个正则语言）。

(c) 思路: 仍然是分层次自顶向下看。推导一个字符串时，先任意次使用第一个和第二个产生式，然后使用一次第三个产生式，最后使用一次第四、五或六个产生式中的一个。

使用任意次第一、二个产生式，然后使用一次第三个产生式，此时得到的串具有 $w\mathrm{L}w^{\mathcal{R}}$ 的形式（其中 $w \in \{\mathtt{a}, \mathtt{b}\}^*$，$w^{\mathcal{R}}$ 是字符串 w 的反转）；最后使用第四、五或六个产生式，得到

$L(G) = \{s | s = waw^{\mathcal{R}}$ 或 $wbw^{\mathcal{R}}$ 或 $ww^{\mathcal{R}}, w \in \{\mathtt{a}, \mathtt{b}\}^*\}$。

或更简单地，$L(G) = \{w | w = w^{\mathcal{R}}, w \in \{a, b\}^*\}$。

(d) 思路: 分层次自顶向下看。推导一个字符串时，先任意次使用第一个产生式，然后使用一次第二个产生式，然后再对 A 任意次使用第三、四个产生式，最后使用一次第五个产生式；对 B 任意次使用第六个产生式，最后使用一次第七个产生式。

反复使用第一个产生式，然后使用第二个产生式，这将使中间串具有 A$\cdots$AB 的形式，所以只要分别分析 A 和 B 能产生的语言。B 产生的字符串很简单，具有 $\mathtt{b}^{2n}$ 的形式。下面讨论 A 产生的字符串。

首先，每个 A 在推导时，产生式右侧最多只有一个非终结符 A，在反复使用第三、四个产生式延长字符串时，若使用的是第三个产生式，则会在 A 的左右分别增加一个 a 与 c（即每个 c 的前面都至少有一个 a 在解析树的同一层同时产生）；若使用的是第四个产生式，则会在 A 的右边增加一个 a。这两个产生式使得 A 产生的字符串 w 具有性质：对 w 的任意前缀，都有 a 的数量总是不少于 c。

上面的性质对于任意个 A 产生的字符串相连得到的新字符串成立，于是

$L(G) = \{w\mathtt{b}^{2n} |$ w 的任意前缀中 a 的数量不少于 c 的数量$, n \geq 0\}$。

2. Find a context-free grammar to describe the following languages.

(a) $L = \{0^n 1^{2n} | n \geq 0\}$.

(b) $L = \{w|$ the length of w is odd$\}$, $L \subseteq \{0, 1\}^*$.

(c) The set of all strings of a and b that include the substring baa.

(d) The set of strings over the alphabet $\{a, b\}$ with an equal number of a's and b's.

(e) The set of strings over the alphabet $\{a, b\}$ with at least as many a's as b's.

(f) The set of strings over the alphabet $\{a, b\}$ with twice as many a's as b's.

解:

(a) 思路: 需要每次在左边增加一个 0 的同时，在右边增加两个 1。

一种 CFG 的产生式如下： S → 0S11 | ε。（进一步地，这不是一个正则语言）

(b) 思路: 需要每次在左侧增加任意两个字符，最后再加上一个字符。

一种 CFG 的产生式如下： S → 00S | 01S | 10S | 11S | 0 | 1。（进一步地，这是一个正则语言）

(c) 思路: 满足条件的字符串 w 可以分为三部分，$w = w_1(\mathtt{baa})w_2$，其中 $w_1, w_2 \in \{a, b\}^*$。

一种 CFG 的产生式如下：

- S → ATA
- A → aA | bA | ε
- T → baa

（进一步地，这是一个正则语言）

(d) 思路: 因为 a 与 b 的数量相等，所以每使用一次产生式增加相同数目的 a 与 b。因此，只需要考虑 a 和 b 的排列，然后在空隙中插入 S 以保持结构。

一种 CFG 的产生式如下： S → SaSbS | SbSaS | ε。（进一步地，这不是一个正则语言）

(e) 思路: 类似于 (d)，引入新变量 A，保证中间串中 A 的数量等于 b 的数量，然后将每一个 A 替换为至少一个 a，还要考虑可能不含 b 的情况。

一种 CFG 的产生式如下：

- S → SASbS | SbSAS | SAS | ε
- A → aA | a

（进一步地，这不是一个正则语言）

(f) 思路: 类似于 (d)，考虑两个 a 和一个 b 的排列。

一种 CFG 的产生式如下：S → SaSaSbS | SaSbSaS | SbSaSaS | ε。（进一步地，这不是一个正则语言）

3. Consider the CFG G defined by productions: S → aS | Sb | a | b

(a) Prove by induction on the string length that no string in $L(G)$ has ba as a substring.

(b) Describe $L(G)$ informally.

解:

(a) 用归纳法。注意到当字符串 w 长度为 n 时，意味着使用了 $n-1$ 次第一或第二个产生式，最后使用了一次第三或第四个产生式。

1. (归纳基础) S 产生的长度为 1 的字符串仅有两个：a 和 b，它们都满足条件。
2. (归纳步骤) 假设 S 产生的长度小于 n 的字符串不含 ba。S 应用第一或第二个产生式，产生的长度为 n 的字符串应该具有 aw 或 wb 的形式。其中 w 是 S 可以产生的长度为 $n-1$ 的字符串，根据归纳假设它不含 ba，则 aw 是在 w 最左边增加一个 a，不可能含有 ba；同理，wb 也不会含有 ba。

根据数学归纳法，结论成立。

(b) 容易看出每次使用第一个产生式，将在左半部分增加一个 a；每次使用第二个产生式，将在右半部分增加一个 b，于是最后 $L(G) = \{\mathtt{a}^m\mathtt{b}^n, m, m \geq 0\}$。

4. Consider the following context-free grammar that tries to capture if-then-else statements in a regular programming language:

- S → PRINT hello | T | U

- T $\rightarrow$ IF condition THEN S
- U $\rightarrow$ IF condition THEN S ELSE S

Prove by example that the grammar is ambiguous. Make it unambiguous.

解:

(1) 考虑如下字符串:

IF condition THEN IF condition THEN PRINT hello ELSE PRINT hello

它有下列两种推导方式:

- 方式一（下划线表示替换部分）:

$\underline{\text{S}} \Rightarrow \underline{\text{U}}$

$\Rightarrow$ IF condition THEN $\underline{\text{S}}$ ELSE S

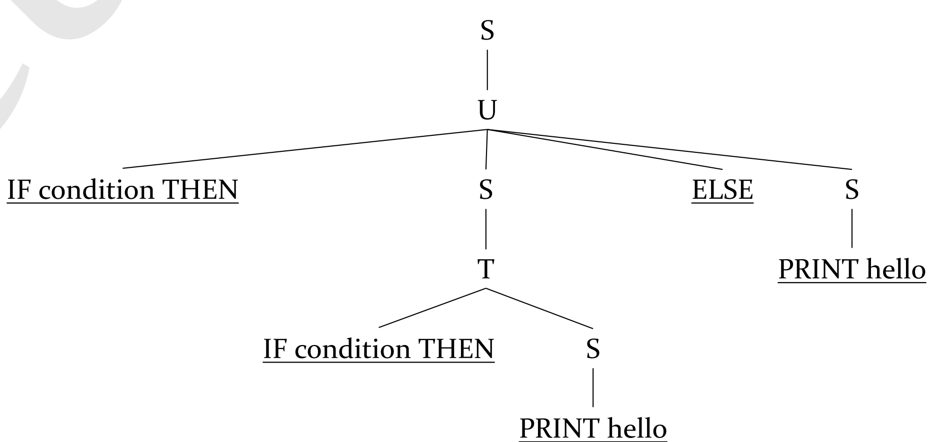
$\Rightarrow$ IF condition THEN $\underline{\text{T}}$ ELSE S

$\Rightarrow$ IF condition THEN IF condition THEN $\underline{\text{S}}$ ELSE S

$\Rightarrow$ IF condition THEN IF condition THEN PRINT hello ELSE $\underline{\text{S}}$

$\Rightarrow$ IF condition THEN IF condition THEN PRINT hello ELSE PRINT hello

此时解析树为（下划线表示终结符）:



- 方式二：

$\underline{S} \Rightarrow \underline{T}$

$\Rightarrow$ IF condition THEN $\underline{S}$

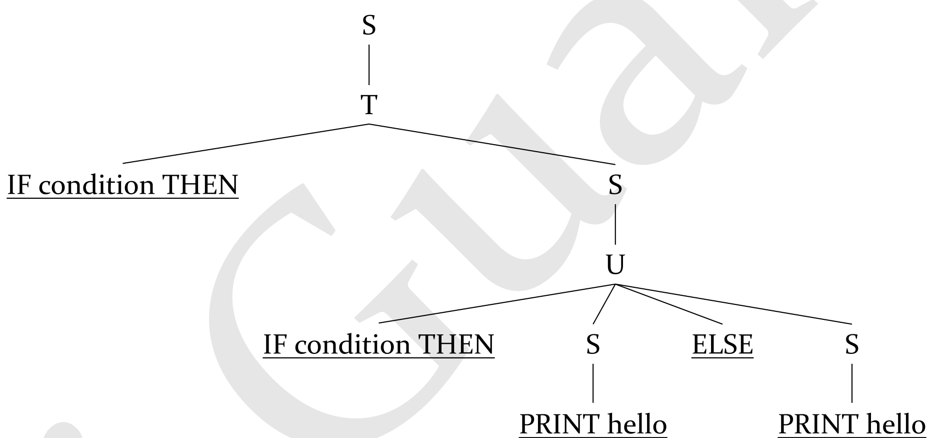
$\Rightarrow$ IF condition THEN $\underline{U}$

$\Rightarrow$ IF condition THEN IF condition THEN $\underline{S}$ ELSE S

$\Rightarrow$ IF condition THEN IF condition THEN PRINT hello ELSE $\underline{S}$

$\Rightarrow$ IF condition THEN IF condition THEN PRINT hello ELSE PRINT hello

此时解析树为：



(2) 上面的语法出现歧义的原因是没有处理好 IF-THEN-ELSE 语句的嵌套问题，遇到嵌套时不知道 ELSE 与前面哪个 IF 配对。可以模仿 C 语言处理方式，当遇到这种情况时，认为 ELSE 与前面最近的 IF 配对，或者说在 IF-THEN-ELSE 语句 THEN 与 ELSE 之间不能有没有 ELSE 的 IF-THEN 语句。于是消歧义后的语法为：

- S → <ABT>　　// 起始符号
- <ABT> → A | B | T　　// 三类，IF-THEN、IF-THEN-ELSE 以及 PRINT
- A → IF condition THEN <ABT>　　// IF-THEN 后面可以接三种类型

- B → IF condition THEN <BT> ELSE <ABT>   // THEN 后不能嵌套 IF-THEN，但可以嵌套 IF-THEN-ELSE，ELSE 后无限制
- <BT> → B | T   // 两类，IF-THEN-ELSE 以及 PRINT
- T → PRINT hello   // 只产生终结符的 PRINT