编译原理作业 - 2.2 节练习题解答

姓名：吴晨昊
学号：102201113

2025 年 3 月 23 日

1 练习题解答

练习 2.2.1

考虑下面的上下文无关文法：

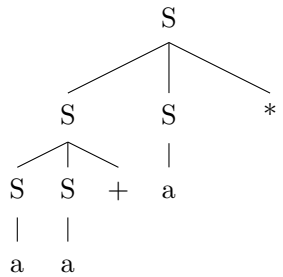
$$S \to SS+ \mid SS* \mid a$$

1. 说明如何使用该文法生成串 $aa+a*$。

   推导过程如下：

$$\begin{aligned} S &\Rightarrow SS* \\ &\Rightarrow SS+S* \\ &\Rightarrow aS+S* \\ &\Rightarrow aa+S* \\ &\Rightarrow aa+a* \end{aligned}$$

2. 为这个串构造一棵语法分析树。



3. 该文法生成的语言是什么？证明你的答案。

   该文法生成的是后缀表达式（逆波兰表达式）的语言。在后缀表达式中，操作符放在操作数之后。

   证明：

   (a) 首先，我们证明该文法生成的任何字符串都是合法的后缀表达式。使用归纳法：

   - 基本情况：$S \to a$ 生成单个操作数，这是一个合法的后缀表达式。
   - 归纳步骤：假设 S 可以推导出合法的后缀表达式 α 和 β，那么：
     - $S \to SS+$ 生成 $\alpha\beta+$，这是一个合法的后缀表达式，表示 α 和 β 的加法。
     - $S \to SS*$ 生成 $\alpha\beta*$，这是一个合法的后缀表达式，表示 α 和 β 的乘法。

   (b) 其次，任何合法的后缀表达式都可以由该文法生成：

   - 单个操作数 a 可以直接由 $S \to a$ 生成。
   - 对于形如 $\alpha\beta+$ 的后缀表达式，可以通过规则 $S \to SS+$ 生成，其中 S 分别推导出 α 和 β。
   - 对于形如 $\alpha\beta*$ 的后缀表达式，可以通过规则 $S \to SS*$ 生成，其中 S 分别推导出 α 和 β。

   因此，该文法生成的语言是只包含单个变量 a 和两个操作符 $+$、$*$ 的后缀表达式语言。

练习 2.2.2

下面的各个文法生成什么语言？证明你的每一个答案。

1. $S \to 0S1 \mid 01$

   该文法生成的语言是 $\{0^n1^n \mid n \geq 1\}$，即等量的 0 和 1 所组成的串，且所有的 0 在所有的 1 之前。

   证明：

- 首先证明 S 只能生成形如 0^n1^n 的串，其中 $n \geq 1$：
  - 基本情况：$S \to 01$ 生成串 01，符合 $0^1 1^1$。
  - 归纳步骤：假设 S 可以生成 0^k1^k，那么通过规则 $S \to 0S1$，可以生成 $0 \cdot 0^k1^k \cdot 1 = 0^{k+1}1^{k+1}$。
- 其次，证明任何形如 $0^n1^n (n \geq 1)$ 的串都可以由该文法生成：
  - 对于 $n = 1$，直接通过 $S \to 01$ 生成 01。
  - 对于 $n > 1$，可以通过规则 $S \to 0S1$ 应用 $n-1$ 次，再使用 $S \to 01$ 生成 0^n1^n。

2. $S \to +SS \mid -SS \mid a$

   该文法生成的是前缀表达式（波兰表达式）的语言，其中只包含单个操作数 a 和两个操作符 $+$、$-$。

   完整证明：

   我们需要证明两点：(1) 该文法生成的任何串都是合法的前缀表达式；(2) 任何合法的前缀表达式都可以由该文法生成。

   - 第一部分：证明该文法生成的任何串都是合法的前缀表达式。

     使用归纳法对推导步骤数量进行归纳：

     (a) 基本情况：$S \Rightarrow a$ 生成单个操作数 a，这是一个合法的前缀表达式。

     (b) 归纳假设：假设通过不超过 k 步推导得到的所有串都是合法的前缀表达式。

     (c) 归纳步骤：考虑通过 $k+1$ 步推导得到的串。最后一步必然是应用以下两条规则之一：
     - $S \to +SS$：根据归纳假设，两个 S 分别能推导出合法的前缀表达式 α 和 β。因此，该规则生成 $+\alpha\beta$，这是一个合法的前缀表达式，表示 α 和 β 的加法运算。
     - $S \to -SS$：同理，该规则生成 $-\alpha\beta$，这是一个合法的前缀表达式，表示 α 和 β 的减法运算。

     因此，任何由该文法生成的串都是合法的前缀表达式。
   - 第二部分：证明任何合法的前缀表达式都可以由该文法生成。
     - 单个操作数 a 可以直接由 $S \to a$ 生成。

- 对于形如 $+\alpha\beta$ 的前缀表达式，可以通过规则 $S \to +SS$ 生成，其中 S 分别推导出 α 和 β。
- 对于形如 $-\alpha\beta$ 的前缀表达式，可以通过规则 $S \to -SS$ 生成，其中 S 分别推导出 α 和 β。

综上所述，该文法生成的语言正是所有由单个操作数 a 和两个操作符 $+$、$-$ 组成的前缀表达式的集合。

3. $S \to S(S)S \mid \varepsilon$

该文法生成的语言是所有包含匹配括号的串，其中括号可以嵌套，并且括号之间可以为空。简单来说，这个语言包含所有由平衡括号组成的串。

这个语言可以表示为集合：

$$L = \{w \in \{(,)\}^* \mid w \text{ 中的括号是匹配的}\}$$

简洁证明如下：

- 第一部分：证明该文法生成的所有串都具有匹配的括号。

  采用对推导步骤数的归纳法：

  (a) 基本情况：$S \Rightarrow \varepsilon$ 生成空串，不含括号，自然满足括号匹配。

  (b) 归纳假设：假设通过不超过 k 步推导得到的所有串的括号都是匹配的。

  (c) 归纳步骤：对于 $k+1$ 步推导，最后一步必须应用规则 $S \to S(S)S$，即将某个 S 替换为 $S(S)S$。由归纳假设，替换前所有括号都匹配，替换后仅增加了一对匹配括号，因此结果串的括号仍然匹配。

- 第二部分：证明任何具有匹配括号的串都可以由该文法生成。

  我们使用归纳法，对串中括号对的数量进行归纳：

  (a) 基本情况：如果串中没有括号对，即空串 ε，可以直接由规则 $S \to \varepsilon$ 生成。

  (b) 归纳假设：假设任何包含不超过 n 个括号对的串都可以由该文法生成。

(c) 归纳步骤：考虑包含 $n+1$ 个括号对的串 w。有两种情况：
a) 如果 w 可以分解为 $w = w_1w_2$，其中 w_1 和 w_2 都是合法的匹配括号串，且含有的括号对数量都小于 $n+1$，那么根据归纳假设，存在推导 $S \Rightarrow^* w_1$ 和 $S \Rightarrow^* w_2$。因此 $S \Rightarrow S(S)S \Rightarrow^* w_1(\varepsilon)w_2 = w_1()w_2$。若某个 w_i 为空，则使用 $S \to \varepsilon$ 规则。
b) 如果 w 的形式为 $w = w_1(w_2)w_3$，其中 w_1、w_2 和 w_3 都是包含匹配括号的串，且括号对数量都小于 $n+1$，那么根据归纳假设，存在推导 $S \Rightarrow^* w_1$、$S \Rightarrow^* w_2$ 和 $S \Rightarrow^* w_3$。因此 $S \Rightarrow S(S)S \Rightarrow^* w_1(w_2)w_3 = w$。
通过分析可知，任何匹配括号的串都可以通过上述两种情况之一来构造，因此任何具有匹配括号的串都可以由该文法生成。

综上所述，该文法生成的语言正是所有具有匹配括号的串的集合。

4. $S \to aSbS \mid bSaS \mid \varepsilon$

该文法生成的语言是所有含有等量 a 和 b 的串，即 $\{w \in \{a,b\}^* \mid |w|_a = |w|_b\}$。

完整证明分为两部分：

- 第一部分：证明该文法生成的任何串中 a 和 b 的数量相等。
使用归纳法对推导步骤数量进行归纳：
(a) 基本情况：$S \Rightarrow \varepsilon$ 产生空串，其中 a 和 b 的数量均为 0，因此相等。
(b) 归纳假设：假设通过不超过 k 步推导得到的所有串中，a 和 b 的数量相等。
(c) 归纳步骤：考虑通过 $k+1$ 步推导得到的串。最后一步必然是应用以下规则之一：
  - $S \to aSbS$：若规则应用后得到串 w，则 $w = a\alpha b\beta$，其中 α 和 β 是由 S 经过不超过 k 步推导得到的串。根据归纳假设，$|\alpha|_a = |\alpha|_b$ 且 $|\beta|_a = |\beta|_b$。因此，$|w|_a = 1 + |\alpha|_a + |\beta|_a = 1 + |\alpha|_b + |\beta|_b = |w|_b$。
  - $S \to bSaS$：类似地，若规则应用后得到串 $w = b\alpha a\beta$，则 $|w|_a = |\alpha|_a + 1 + |\beta|_a = |\alpha|_b + 1 + |\beta|_b = |w|_b$。

因此，由该文法生成的任何串中，a 和 b 的数量都相等。

- 第二部分：证明任何含有等量 a 和 b 的串都可以由该文法生成。
  使用归纳法对串的长度进行归纳：
  (a) 基本情况：长度为 0 的串只有 ε，可以直接由规则 $S \to \varepsilon$ 生成。
  (b) 归纳假设：假设任何长度不超过 $2n$ 且含有等量 a 和 b 的串都可以由该文法生成。
  (c) 归纳步骤：考虑长度为 $2(n+1)$ 且含有等量 a 和 b 的串 w。
  令 $w = w_1 w_2 ... w_{2(n+1)}$，考虑函数 $f(i) = |w_1 ... w_i|_a - |w_1 ... w_i|_b$，表示前 i 个字符中 a 的数量减去 b 的数量。显然 $f(0) = 0$ 且 $f(2(n+1)) = 0$（因为整个串中 a 和 b 数量相等）。
  在 i 从 1 到 $2(n+1)$ 的遍历过程中，$f(i)$ 的值会发生变化：当遇到 a 时增加 1，遇到 b 时减少 1。因此，存在某个位置 j 使得 $f(j) = 0$ 且 $1 \le j < 2(n+1)$。
  我们可以将 w 分解为 $w = uv$，其中 $u = w_1 ... w_j$ 且 $v = w_{j+1} ... w_{2(n+1)}$。由于 $f(j) = 0$，所以 u 中 a 和 b 的数量相等；同理，v 中 a 和 b 的数量也相等。
  现在，u 和 v 都是更短的等量 a 和 b 的串。根据归纳假设，存在推导 $S \Rightarrow^* u$ 和 $S \Rightarrow^* v$。
  如果 u 的第一个字符是 a 且最后一个字符是 b，那么我们可以将 u 表示为 $u = au'b$，其中 u' 中 a 和 b 的数量相等。根据归纳假设，存在推导 $S \Rightarrow^* u'$ 和 $S \Rightarrow^* v$。因此，$S \Rightarrow aSbS \Rightarrow au'bS \Rightarrow^* au'bv = uv = w$。
  类似地，如果 u 的第一个字符是 b 且最后一个字符是 a，我们可以构造 $S \Rightarrow bSaS \Rightarrow^* buav = w$。
  如果 u 不是以上两种情况，我们可以找到另一个分解点 j' 使得分解后的 u' 符合以上两种情况之一。

综上所述，该文法生成的语言正是所有含有等量 a 和 b 的串的集合。

5. $S \to a \mid S + S \mid SS \mid S* \mid (S)$

该文法生成的是包含单个变量 a、加法操作 $+$、连接操作、后缀乘法操作 $*$ 和带括号表达式的混合表达式语言。它生成的是由 a、$+$、$*$、(和) 组成的表达式，其中 $*$ 是后缀运算符。

完整证明通过归纳法进行：

- 第一部分：证明该文法生成的所有串都是合法的表达式。

  使用对推导步骤数量的归纳：

  (a) 基本情况：$S \Rightarrow a$ 产生单个操作数 a，这是一个合法的表达式。

  (b) 归纳假设：假设通过不超过 k 步推导得到的所有串都是合法的表达式。

  (c) 归纳步骤：考虑通过 $k+1$ 步推导得到的串。最后一步一定是应用以下规则之一：

  - $S \to S+S$：将一个非终结符 S 替换为 $S+S$。根据归纳假设，S 可以推导出合法的表达式 α 和 β，那么 $\alpha+\beta$ 也是一个合法的表达式，表示两个表达式的加法。
  - $S \to SS$：将一个非终结符 S 替换为 SS。根据归纳假设，S 可以推导出合法的表达式 α 和 β，那么 $\alpha\beta$ 表示两个表达式的连接。
  - $S \to S*$：将一个非终结符 S 替换为 $S*$。根据归纳假设，S 可以推导出合法的表达式 α，那么 $\alpha*$ 也是一个合法的表达式，表示对 α 进行后缀乘法运算。
  - $S \to (S)$：将一个非终结符 S 替换为 (S)。根据归纳假设，S 可以推导出合法的表达式 α，那么 (α) 也是一个合法的表达式，表示带括号的表达式。

  在所有情况下，替换后的串仍然是合法的表达式。

- 第二部分：证明任何合法的表达式都可以由该文法生成。

  我们对表达式的复杂度进行归纳：

  (a) 基本情况：最简单的合法表达式是单个操作数 a，可以直接由规则 $S \to a$ 生成。

  (b) 归纳假设：假设任何复杂度不超过 n 的合法表达式都可以由该文法生成。

  (c) 归纳步骤：考虑复杂度为 $n+1$ 的合法表达式 E。根据表达式的形式，有以下几种情况：

- 如果 E 是形如 $E_1 + E_2$ 的表达式，其中 E_1 和 E_2 是复杂度较小的合法表达式，那么根据归纳假设，存在推导 $S \Rightarrow^* E_1$ 和 $S \Rightarrow^* E_2$。因此 $S \Rightarrow S+S \Rightarrow^* E_1+E_2 = E$。
- 如果 E 是形如 E_1E_2 的表达式（表示连接），其中 E_1 和 E_2 是复杂度较小的合法表达式，那么根据归纳假设，存在推导 $S \Rightarrow^* E_1$ 和 $S \Rightarrow^* E_2$。因此 $S \Rightarrow SS \Rightarrow^* E_1E_2 = E$。
- 如果 E 是形如 E_1* 的表达式，其中 E_1 是复杂度较小的合法表达式，那么根据归纳假设，存在推导 $S \Rightarrow^* E_1$。因此 $S \Rightarrow S* \Rightarrow^* E_1* = E$。
- 如果 E 是形如 (E_1) 的表达式，其中 E_1 是复杂度较小的合法表达式，那么根据归纳假设，存在推导 $S \Rightarrow^* E_1$。因此 $S \Rightarrow (S) \Rightarrow^* (E_1) = E$。

通过分析可知，任何合法的表达式都可以通过上述情况之一来构造，因此任何合法的表达式都可以由该文法生成。

综上所述，该文法生成的语言是由单个变量 a、加法操作 $+$、连接操作、后缀乘法操作 $*$ 和括号组成的表达式语言。

练习 2.2.3

练习 2.2.2 中哪些文法具有二义性?

- 文法 1: $S \to 0S1 \mid 01$ 不具有二义性。对于任何给定的串，都只有一种唯一的派生方式。
- 文法 2: $S \to +SS \mid -SS \mid a$ 不具有二义性。前缀表达式本身就是无歧义的，而且该文法的构造方式保证了对每个前缀表达式只有一种唯一的派生方式。
- 文法 3: $S \to S(S)S \mid \varepsilon$ 具有二义性。例如，对于串 $()(())$，有多种不

同的派生方式：

第一种派生方式：

$$S \Rightarrow S(S)S \Rightarrow S(S)S(S)S$$
$$\Rightarrow S(S)S(S) \Rightarrow S(S)S(()) \Rightarrow S(S)(()) \Rightarrow (S)(())$$
$$\Rightarrow ()(())$$

第二种派生方式：

$$S \Rightarrow S(S)S \Rightarrow S(S)S(S)S \Rightarrow S(S)S(S) \Rightarrow S(S)(S)$$
$$\Rightarrow S()(S) \Rightarrow ()(S) \Rightarrow ()(S(S)S) \Rightarrow ()(S(S))$$
$$\Rightarrow ()(S()) \Rightarrow ()(())$$

这两种不同的派生方式对应于不同的语法分析树，因此该文法具有二义性。

- 文法 4: $S \to aSbS \mid bSaS \mid \varepsilon$ 具有二义性。例如，对于串 $abab$，可以通过以下两种方式派生：

$$S \Rightarrow aSbS \Rightarrow abS \Rightarrow abaSbS \Rightarrow ababS \Rightarrow abab$$
$$S \Rightarrow aSbS \Rightarrow aSb \Rightarrow abSaSb \Rightarrow abSab \Rightarrow abab$$

- 文法 5: $S \to a \mid S + S \mid SS \mid S* \mid (S)$ 具有二义性。例如，对于串 $a + a*$，可以有以下两种不同的派生方式：

第一种派生方式（解释为 $(a + a)*$）：

$$S \Rightarrow S* \Rightarrow (S)* \Rightarrow (S + S)* \Rightarrow (a + S)* \Rightarrow (a + a)*$$

第二种派生方式（解释为 $a + (a*)$）：

$$S \Rightarrow S + S \Rightarrow a + S \Rightarrow a + S* \Rightarrow a + a*$$

这表明该文法在处理表达式优先级时是有二义性的。

因此，文法 3、4 和 5 具有二义性。

练习 2.2.4

为下面的各个语言构建无二义性的上下文无关文法。证明你的文法都是正确的。

1. 用后缀方式表示的算术表达式。

   无二义性的文法：

   $$E \to id \mid EEop$$

   其中，id 表示标识符或数字，op 表示操作符（如 $+$、$-$、$*$、$/$）。

   证明：该文法生成的每一个后缀表达式都只有一种解析方式，因为后缀表达式本身就是无歧义的。对于任何合法的后缀表达式，都可以构建唯一的语法分析树。

2. 由逗号分隔开的左结合的标识符列表。

   无二义性的文法：

   $$L \to id \mid L, id$$

   证明：该文法强制了左结合性，因为列表的扩展只能是在右侧添加元素。例如，对于列表 a, b, c，唯一的解析方式是 $((a, b), c)$。

3. 由逗号分隔开的右结合的标识符列表。

   无二义性的文法：

   $$L \to id \mid id, L$$

   证明：该文法强制了右结合性，因为列表的扩展只能是在左侧添加元素。例如，对于列表 a, b, c，唯一的解析方式是 $(a, (b, c))$。

4. 由整数、标识符、四个二目运算符 $+$、$-$、$*$、$/$ 构成的算术表达式。

   无二义性的文法（考虑运算符优先级和左结合性）：

   $$\begin{aligned} E &\to E + T \mid E - T \mid T \\ T &\to T * F \mid T/F \mid F \\ F &\to id \mid num \mid (E) \end{aligned}$$

   证明：该文法通过分层结构处理了运算符的优先级（乘除高于加减），通过左递归处理了左结合性。对于任何表达式，都只有一种唯一的解析方式。

5. 在 (4) 的运算符中增加单目 $+$ 和单目 $-$，构成新的算术表达式。

   无二义性的文法：

$$
\begin{aligned}
E &\to E + T \mid E - T \mid T \\
T &\to T * T1 \mid T/T1 \mid T1 \\
T1 &\to +T1 \mid -T1 \mid F \\
F &\to id \mid num \mid (E)
\end{aligned}
$$

   证明：该文法除了处理二目运算符的优先级和结合性外，还将单目运算符放在最高优先级层，确保了表达式的唯一解析方式。

练习 2.2.5

1. 证明：用下面的文法生成的所有二进制串的值都能被 3 整除。

$$num \to 11 \mid 1001 \mid num0 \mid num\ num$$

   证明使用数学归纳法：

   基本情况：

   - $num \to 11$：十进制值为 3，可被 3 整除。
   - $num \to 1001$：十进制值为 9，可被 3 整除。

   归纳步骤：

   - 假设 num 可以推导出值为 $3k$ 的二进制串（即被 3 整除）。
   - 对于规则 $num \to num\ 0$：将二进制数左移一位相当于乘以 2，因此得到的值为 $3k \times 2 = 6k$，仍能被 3 整除。
   - 对于规则 $num \to num\ num$：如果两个串的值分别为 $3k_1$ 和 $3k_2$，则它们的串联值为 $3k_1 \times 2^n + 3k_2 = 3(k_1 \times 2^n + k_2)$，其中 n 是第二个串的长度。这个值仍能被 3 整除。

   因此，用该文法生成的所有二进制串的值都能被 3 整除。

2. 上述文法是否能够生成所有能被 3 整除的二进制串?

   不能。例如，二进制串 1111（十进制值为 15）能被 3 整除，但不能由该文法生成。

练习 2.2.6

为罗马数字构建一个上下文无关文法。
罗马数字的规则如下:

- 基本符号: I(1), V(5), X(10), L(50), C(100), D(500), M(1000)
- 一般规则是从左到右由大到小排列符号, 如 VII 表示 7(5+1+1)
- 特殊规则是当小符号放在大符号左边时, 表示大符号减去小符号的值, 如 IV 表示 4(5-1)
- 减法规则只使用于特定组合: IV(4), IX(9), XL(40), XC(90), CD(400), CM(900)
- 符号重复不超过三次, 如 III 表示 3, 但不使用 IIII 表示 4
- 符号 V, L, D 不重复

注：该文法不考虑罗马数字的上下划线表示，故只能产生小于 4000 的数字。

根据观察罗马数字的结构，可以发现每个数位（个位、十位、百位、千位）的表示都遵循类似的模式，可分为四类:

- 1-3: 重复基本符号 (I, X, C, M)1-3 次
- 4: 小符号在大符号前 (IV, XL, CD)
- 5-8: 大符号 (V, L, D) 加 0-3 个小符号
- 9: 小符号在更大符号前 (IX, XC, CM)

罗马数字的上下文无关文法如下：

$$
\begin{aligned}
romanNum &\to thousand\ hundred\ ten\ digit \\
thousand &\to M \mid MM \mid MMM \mid \varepsilon \\
hundred &\to smallHundred \mid CD \mid D\ smallHundred \mid CM \\
smallHundred &\to C \mid CC \mid CCC \mid \varepsilon \\
ten &\to smallTen \mid XL \mid L\ smallTen \mid XC \\
smallTen &\to X \mid XX \mid XXX \mid \varepsilon \\
digit &\to smallDigit \mid IV \mid V\ smallDigit \mid IX \\
smallDigit &\to I \mid II \mid III \mid \varepsilon
\end{aligned}
$$

这个文法直接对应了罗马数字的数位结构，每个数位都有四种可能的表示形式，对应数字 1-3、4、5-8 和 9。文法中：

- $romanNum$ 表示整个罗马数字
- $thousand$ 表示千位部分 (1000-3000 或 0)
- $hundred$ 表示百位部分 (100-900 或 0)
- ten 表示十位部分 (10-90 或 0)
- $digit$ 表示个位部分 (1-9 或 0)
- $smallHundred$, $smallTen$, $smallDigit$ 分别表示百位、十位、个位中的 1-3 或 0