第 1 章

基础知识

本章主要讨论本书中用到的相关基础知识，为后续章节内容打下基础。这些基础知识包括集合、关系与映射、上下文无关文法、数学归纳法和结构归纳法，以及文法的实现。

1.1 集合

本小节，我们首先介绍本书中需要用到的一些集合论基础概念以及它们的符号表示。

集合论是研究集合（由一堆抽象物件构成的整体）的数学理论，包含了集合、元素、成员关系等最基本的数学概念。我们用大写字母表示集合，小写字母表示集合元素；符号 ϕ 表示空集，即不包含任何元素，且空集是任何集合的子集。

元素 a 是集合 A 的元素，记作 $a \in A$；集合 A 是集合 B 的子集，记作 $A \subseteq B$， 表示集合 A 的元素都是集合 B 的元素；集合 A 是集合 B 的真子集，记作 $A \subset B$，表示凡是 A 的元素都是 B 的元素，但是集合 A 和 B 不相等。例如集合 $\{1,2\}$ 的子集包含 $\{\{1\},\{2\},\{1,2\},\phi\}$， 而它的真子集只包

含 $\{\{1\},\{2\},\phi\}$。

集合 A 和集合 B 相等，记作 $A=B$，即满足

$$A\subseteq B \text{ 且 } B\supseteq A。$$

集合 A 和集合 B 不相等，记作 $A\neq B$。

集合 A 的幂集，记作

$$\mathcal{P}(A)=\{a\mid a\subseteq A\},$$

它是由集合 A 的全体子集所形成的集合。例如，集合

$$A=\{1,2,3\},$$

则

$$\mathcal{P}(A)=\{\{1\},\{2\},\{3\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\},\phi\}。$$

记 $\mid A\mid$ 为集合 A 中元素的个数；则对幂集，我们有

$$\mid\mathcal{P}(A)\mid=2^{\mid A\mid}。$$

可通过数学归纳法证明该结论，我们把该证明作为练习留给读者。

集合 A 和集合 B 的并，记作

$$A\cup B=\{x\mid x\in A\text{或}x\in B\}。$$

集合 A 和集合 B 的交，记作

$$A\cap B=\{x\mid x\in A\text{且}x\in B\}。$$

集合的并和交都满足交换律、结合率和分配率。以后，我们把有限次的集合并

$$A_1\cup A_2\cup\ldots\cup A_n$$

记作

$$\bigcup_{i=1}^{n} A_i;$$

把有限次的集合交

$$A_1 \cap A_2 \cap \ldots \cap A_n$$

记作

$$\bigcap_{i=1}^{n} A_i。$$

集合 A 与集合 B 的差集，记作

$$A - B = \{x \mid x \in A \text{ 且 } x \notin B\}。$$

1.2 关系与映射

本小节，我们主要讨论关系与映射。

两元素 a 和 b 按一定次序组成的二元组 (a, b)，称为**有序对**，并满足基本性质

$$(a_1, b_1) = (a_2, b_2) \quad \Longleftrightarrow \quad a_1 = a_2 \text{ 且 } b_1 = b_2。$$

集合 A 与集合 B 的**笛卡尔积**$A \times B$，是由如下有序对构成的集合

$$A \times B = \{(a, b) \mid a \in A \text{ 且 } b \in B\}。$$

有序对和笛卡尔积可推广到 n 个元素或集合的情况。即 n 个集合 A_1，…，A_n 的笛卡尔积，可记作

$$\begin{aligned}\prod_{i=1}^{n} A_i &= A_1 \times A_2 \times \ldots \times A_n \\ &= \{(a_1, \ldots, a_n) \mid a_1 \in A_1, \ldots, a_n \in A_n\}。\end{aligned}$$

特别的，我们用 A^n 表示 n、$n \geq 1$ 个集合 A 的笛卡尔积

$$\underbrace{A \times A \times \ldots \times A}_{n},$$

并规定 $A^0 = \phi$。

对于任意两个集合 A 和集合 B，我们称其笛卡尔积 $A \times B$ 的任意子集

$$R \subseteq A \times B,$$

为在集合 A 和 B 上的**二元关系**，简称**关系**。特别的，若 $A = B$，则称集合 R 为集合 A 上的二元关系。

对于集合 A 和 B 上的关系 R，我们称集合 A 是关系 R 的**前域**，集合 B 是关系 R 的**后域**。记

$$C = \{a \mid a \in A，\text{存在} b \in B，\text{使得} (a, b) \in R\}$$
$$D = \{b \mid b \in B，\text{存在} a \in A，\text{使得} (a, b) \in R\}$$

我们称集合 C 为关系 R 的**定义域**，记作 $dom(R)$；称集合 D 为关系 R 的**值域**，记作 $ran(R)$；记

$$fld(R) = dom(R) \cup ran(R)$$

为关系 R 的**域**。

若集合 A 上的二元关系 R 满足如下三条性质：

1. 自反性：对于任意 $x \in A$，有 $(x, x) \in R$；
2. 对称性：对于任意 x，$y \in A$，若 $(x, y) \in R$，则 $(y, x) \in R$；
3. 传递性：对于任意 x，y，$z \in A$，若 $(x, y) \in R$ 且 $(y, z) \in R$，则 $(x, z) \in R$；

则把关系 R 称作集合 A 上的**等价关系**。

若关系 R 是集合 A 上的等价关系，且 $(a,b) \in R$，则称元素 a 和 b 等价，记作 $a \sim b$。集合 A 中与元素 a 等价的所有元素形成的集合，称为 a 由关系 R 生成的等价类，记作

$$[a]_R = \{x \mid x \in A \text{且} x \sim a\}。$$

若关系 R 是集合 A 上的等价关系，我们把关系 R 定义的所有等价类的集合称为**商集**，记作 A/R。

接下来，我们讨论映射。

设 $f \subseteq A \times B$ 是集合 A 和 B 上的一个关系，如果关系 f 还满足：对任意 $x \in A$，有且仅有一个 $y \in B$ 使得 $(x,y) \in f$，那么我们称关系 f 是从集合 A 到 B 的**映射**，并记作

$$f : A \to B。$$

其中，集合 A 称作映射 f 的**定义域**，集合 B 称作映射 f 的**值域**。对于 $(x,y) \in f$，我们称 y 为 x 的**象**，称 x 为 y 的**原象**，并记作

$$x \mapsto y \text{ 或 } y = f(x)。$$

对于映射 $f : A \to B$，若在集合 B 中的任意元素 y，在集合 A 中都存在至少一个原象 x，使得 $y = f(x)$，则我们把这类映射 f 称作为集合 A 到 B 的**满射**。

如果映射 $f : A \to B$ 对任意 $x_1, x_2 \in A$ 都满足

$$x_1 \neq x_2 \Rightarrow f(x_1) \neq f(x_2),$$

则我们把这类映射 f 称作从集合 A 到 B 的**单射**。

如果映射 $f : A \to B$ 既是单射又是满射，则我们把这类映射 f 称作从集合 A 到 B 的**双射**。此时，我们也可以说集合 A 和集合 B 之间存在一一对应关系。

如果映射 $f : A \to B$ 是双射，则 f^{-1} 称为映射 f 的**逆映射**；即

$$f^{-1}(x) = y \iff f(y) = x。$$

如果映射 $f : A \to B$ 的值域是映射 $g : B \to C$ 的定义域，则我们可以对映射 f 和 g 进行复合，得到复合映射

$$g \circ f : A \to C,$$

它满足

$$(g \circ f)(x) = g(f(x))。$$

1.3 上下文无关文法

本小节，我们将介绍**上下文无法文法**（Context-free Grammer），简称**文法**。

定义 1.1 (上下文无关文法) **上下文无关文法** G **由四元组**

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

构成，其中

1. N **是非终结符的有限集合**；
2. T **是终结符的有限集合，且** $N \cap T = \phi$；
3. S **是开始符号，即** $S \in N$；
4. P **是产生式的有限集合，每个产生式具有形式**

$$A ::= \alpha,$$

其中 $A \in N$ 称为产生式的左部；而 $\alpha \in (N \cup T)^*$ 称为产生式的右部。

在本书中，我们约定用小写字母代表终结符，用大写字母代表非终结符，用小写希腊字母代表由终结符或非终结符构成的串。

例 1.2　给定文法 G:

$$S ::= A\ B\ C$$
$$A ::= u$$
$$A ::= v$$
$$B ::= w$$
$$B ::= x$$
$$C ::= y$$

其中，终结符集合 $N = \{u, v, w, x, y\}$，非终结符集合 $T = \{S, A, B, C\}$，开始符号为非终结符 S，文法 G 中包括 6 条产生式。

在接下来，我们将文法的产生式写成紧凑形式，即将左部相同的表达式写成一条，并用单竖线将右边进行隔离。例如，可将产生式

$$A ::= u$$
$$A ::= v$$

写成

$$A ::= u \mid v。$$

从文法的开始符号 S 出发，反复用特定产生式右部替换其左部的非终结符，最终生成一个不包含非终结符的串，我

们称该串为**句子**（Sentence）；并且，我们把这个反复替换的过程称作**推导**（Derivation）。

根据例（1.2）给出的文法定义，我们对句子 $u\ w\ y$ 进行推导，过程如下：

$$\begin{aligned} S &\to A\ B\ C \\ &\to u\ B\ C \\ &\to u\ w\ C \\ &\to u\ w\ y \end{aligned}$$

接下来，我们举一个含有加减乘除的表达式的文法：

例 1.3　给定文法 G

$$E ::= E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid n$$

其中，终结符集合 $N = \{+, -, *, /, n, ()\}$；非终结符集合 $T = \{E\}$；开始符号为非终结符 E。

根据例（1.3）给出的文法定义，对句子 $1 + 2 * 3$ 的一个可能的推导过程如下：

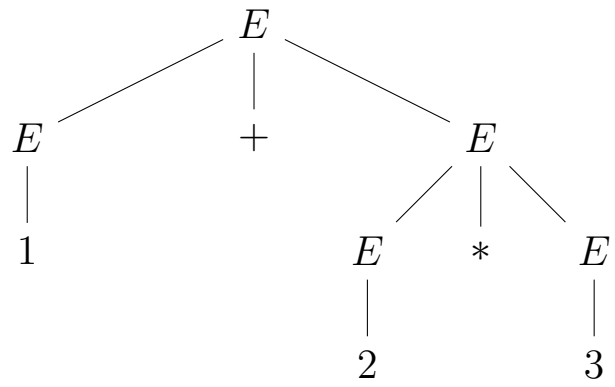
$$\begin{aligned} E &\to E + E \\ &\to E + E * E \\ &\to 1 + E * E \\ &\to 1 + 2 * E \\ &\to 1 + 2 * 3 \end{aligned}$$

需要注意，对该句子的推导过程并不是唯一的；作为练习，请读者自行尝试给出其它可能的推导。

我们可以把推导的过程，以“树”的图形形式展现出来，这种图形形式称为**推导树** （Derivation tree）（在编译原理领域

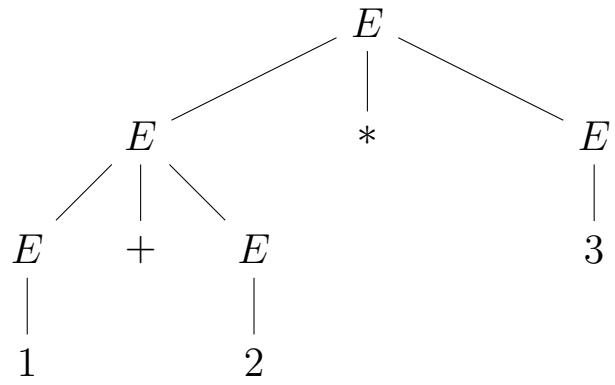
的术语中，由于这类树经常被用来表示语法分析过程，也经常被称为**语法分析树**（Parsing tree））。

对例（1.3）给出的文法定义，对句子 $1+2*3$ 的推导树表示如下：



推导树具有一些非常明确的性质，例如：1）根节点是文法的开始符号；2）每个非叶子节点为一个非终结符；3）每个叶子节点为一个终结符；4）所有叶子节点，从左向右排列，构成了目标句子。

对于给定一个的文法，如果其某个句子能用多棵推导树来表示，那么我们称该文法具有**二义性**（Ambiguity）。如例（1.3）给出的文法定义，其句子 $1+2*3$ 可以用另一颗推导树来表示：



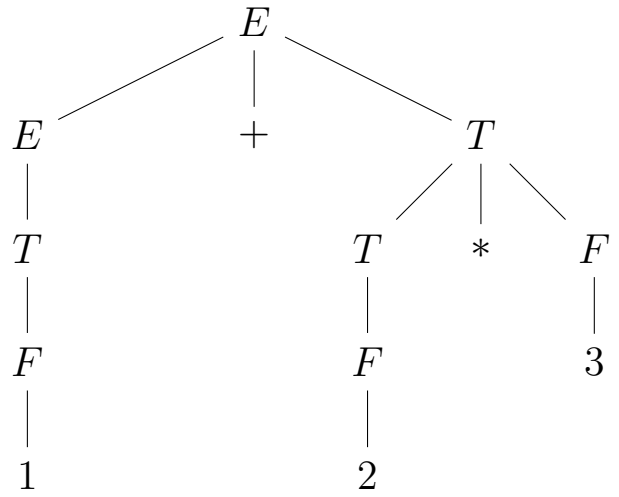
直观上看，这两棵推导树分别对应两种带括号的表达式

$1+(2*3)$ 和 $(1+2)*3$，计算得到的值分别为 7 和 9。显然表示式 $1+(2*3)$ 符合我们对式 $1+2*3$ 的直观理解。但是，例（1.3）给出的文法无法支持确定的推导，所以我们需要对该文法的产生式规则进行修改，避免出现这类二义性，修改后的文法规则如下：

$$
\begin{aligned}
E &::= E+T \mid E-T \mid T \\
T &::= T*F \mid T/F \mid F \\
F &::= (E) \mid n
\end{aligned}
$$

其中，终结符集合 T、开始符号和文法 G 的句子集合不变，非终结符 N 的集合变为 $N=\{E,T,F\}$。

根据修改后的文法，我们继续对句子 $1+2*3$ 进行推导，可以得到唯一一棵推导树:



1.4 归纳法

本小节，我们将讨论在本书中常用的几种归纳法原理。

我们先从基本的数学归纳法开始，再推广到更一般的结构化归纳法。数学归纳法遵循如下的步骤：要证明命题 $P(n)$

对任意的自然数 $n \in \mathbb{N}$ 都成立，我们只需要证明：

1. 基础步：证明命题 P 对自然数 0 成立，即 $P(0)$ 成立；
2. 归纳步：假定命题 P 对自然数 k 成立，证明命题 $P(k+1)$ 仍然成立。

作为数学归纳法的示例，我们证明

定理 1.4 *对任意自然数 n，有*

$$\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}。$$

证明. 分成两个步骤来证：

1. 基础步：当 $n = 0$ 时，等式两侧都等于 0，命题成立；
2. 归纳步：假设命题对 $n = k$ 成立，即

$$\sum_{i=0}^{k} i^2 = \frac{k(k+1)(2k+1)}{6};$$

则当 $n = k+1$ 时，我们有

$$\begin{aligned}
\text{左部} &= \sum_{i=0}^{k+1} i^2 \\
&= \sum_{i=0}^{k} i^2 + (k+1)^2 \\
&= \frac{k(k+1)(2k+1)}{6} + (k+1)^2 \\
&= \frac{(k+1)(2k^2+7k+7)}{6} \\
&= \frac{(k+1)((k+1)+1)(2(k+1)+1)}{6} \\
&= \text{右部};
\end{aligned}$$

故命题对 $n = k + 1$ 成立。

综上两个步骤，命题得证。 证毕

基于上下文无关文法，我们可以给出自然数的另外一种表示形式，即

定义 1.5 (Church 数) *自然数 N 由如下文法定义：*

$$N ::= z \mid s\, N$$

直观上，Church 数是由如下元素构成的集合

$$\{z, s\, z, s\, s\, z, s\, s\, s\, z, \ldots\}\text{。}$$

自然数及其 Church 数表示是等价的，因为可以在两者间建立如下的双射。映射 f 把自然数映射到 Church 数：

$$\begin{aligned} f(0) &= z \\ f(n) &= s\,(f(n-1)) \end{aligned}$$

以及逆映射 f^{-1}，把 Church 数映射到自然数：

$$\begin{aligned} f^{-1}(z) &= 0 \\ f^{-1}(s\, N) &= 1 + (f^{-1}(N)) \end{aligned}$$

我们可以把数学归纳法，重新表述为在 Church 数上的等价形式；即要证明命题 $P(N)$ 对任意的 Church 数 N 都成立，我们只需要证明：

1. 基础步：证明命题 P 对 z 成立，即 $P(z)$ 成立；
2. 归纳步：假定命题 P 对 Church 数 N 成立，即 $P(N)$；证明命题 $P(s\, N)$ 成立。

注意到上述归纳证明的原理，本质上是对定义（1.5）产生式右侧的两种不同结构进行归纳，因此这种归纳法被称为**结构归纳法**（Structural induction）。

结构归纳法是对数学归纳法的一种推广，它可以适用于一般的产生式结构。对于产生式

$$A ::= \alpha_1 \mid \ldots \mid \alpha_m \mid \beta_1 A \gamma_1 \mid \ldots \mid \beta_n A \gamma_n ,$$

其中非终结符 A 在 α_i、$1 \leq i \leq m$ 中不出现。要证明某个命题 $P(A)$ 成立，只要证明如下两个步骤：

1. 基础步：证明命题 P 对 α_i、$1 \leq i \leq m$ 都成立；
2. 归纳步：分别假设命题 P 对 $\beta_i A \gamma_i$、$1 \leq i \leq n$ 中的 A 成立，分别证明 $P(\beta_i A \gamma_i)$ 成立。

综合上述两个步骤，可证明命题 $P(A)$ 成立。

对例（1.3）中给定的文法 G，我们记非终结符 E 产生的句子中的左、右圆括号的数量分别为 $L(E)$ 和 $R(E)$，则我们可以证明

定理 1.6 *对 E 产生的任意句子，有 $L(E) = R(E)$。*

证明. 用结构归纳法。

1. 基础步：对于终结符 n，显然有 $L(n) = R(n) = 0$，故命题成立；
2. 归纳步：分别考虑产生式的每个不同的包含非终结符 E 的右部：
   (a) 对于 $E_1 + E_2$（注意，为了对其中的两个 E 进行区分，我们加入了显式下标），假设命题对于 E_1 和 E_2 都成

立，即

$$L(E_1) = R(E_1)$$
$$L(E_2) = R(E_2);$$

则

$$\begin{aligned} L(E_1 + E_2) &= L(E_1) + L(E_2) \\ &= R(E_1) + R(E_2) \\ &= R(E_1 + E_2); \end{aligned}$$

因此原命题成立；类似的，我们可以证明命题对于 $E_1 - E_2$、$E_1 * E_2$、E_1/E_2 亦成立；

(b) 对于 (E_1)，假设命题对于 E_1 成立，即 $L(E_1) = R(E_1)$；则

$$\begin{aligned} L((E_1)) &= 1 + L(E_1) \\ &= 1 + R(E_1) \\ &= R((E_1)); \end{aligned}$$

因此原命题成立；

因此，命题对产生式的所有右部都分别成立；

综合上述两个步骤，原命题成立。 证毕

对数学归纳法的一个重要推广是**第二数学归纳法**（The second principle of mathematical induction），也称为**完全归纳法**（The principle of complete induction）。在第二数学归纳法中，欲证明某个命题 $P(n)$ 成立，只需要执行以下步骤：

1. 基础步：证明 $P(0)$、...、$P(m)$ 命题成立（其中 m 是某个自然数）；

2. 归纳步：假设命题对于任意自然数 i、$i \leq k$ 都成立，证明命题对于 $k+1$ 成立。

由于第二数学归纳法使用了更多的证明前提，因此证明的过程更加灵活。

定理 1.7 *任意给定的自然数 n、$n \geq 12$ 都可以写成*

$$n = 4p + 5q$$

的形式，其中 p、$q \in \mathbb{N}$。

证明. 使用第二数学归纳法。

1. 基础步：不难证明命题对于 $n = 12$、13、14 分别都成立；
2. 归纳步：假设命题对于任意自然数 $n \leq k$ 都成立，则

$$\begin{aligned} n &= k + 1 \\ &= (k-3) + 4 \\ &= (4p + 5q) + 4 \qquad \text{（由归纳假设）} \\ &= 4(p+1) + 5q \end{aligned}$$

命题得证；

综合以上两个步骤，命题得证。 证毕

对第二数学归纳法的推广是**良基归纳法**（Well-founded induction）。为此，我们首先需要

定义 1.8 (良基关系) *集合 S 上的二元关系 $\succ \subseteq S \times S$ 被称为良基关系；当且仅当，在集合 S 上不存在形如*

$$s_1 \succ s_2 \succ s_3 \succ \dots$$

的无限序列。

直观上，良基关系意味着由二元关系 $\succ$ 定义的递减序列一定是有限的。

例 1.9 *在自然数集合 $\mathbb{N}$ 上的二元关系 $>$ 是良基关系。*

例如，从自然数 65536 开始的序列：

$$65536 > 1024 > 128 > 37 > 0。$$

不难验证，在 $\mathbb{N}$ 上的二元关系 $>$ 最多下降到元素 0。

例 1.10 *在有理数集合 $\mathbb{Q}$ 上的二元关系 $>$ 不是良基关系。*

不难举出一个在序列 $\frac{1}{i}$、$i \geq 1$ 上无限下降的反例

$$\frac{1}{1} > \frac{1}{2} > \frac{1}{3} > \frac{1}{4} > \ldots。$$

在良基归纳法中，欲证明某个命题 $P(n)$ 成立，只需要执行以下步骤：

1. 基础步：证明 $P(0)$、...、$P(m)$ 命题成立（其中 m 是若干个具体命题相关的基础情形）；
2. 归纳步：假设命题对于任意元素 i、$i \prec k$ 都成立，证明命题对于 k 成立。

良基归纳法的重要意义在于：它把数学归纳法、第二数学归纳法和结构归纳法统一都一个框架中。表 1.1给出了不同归纳法所使用的良基关系：对于数学归纳法，良基关系定义为元素 n 的前驱元素；对于第二数学归纳法，良基关系定义为比 n 小的所有元素；而对于结构归纳，良基关系定义为表达式的所有子表达式。

良基关系可进一步推广到集合的笛卡尔积上：给定分别具有良基关系 $\succ_A$ 和 $\succ_B$ 的集合 A 和 B，定义笛卡尔积 $A \times B$

表 1.1: 良基归纳法的不同表现形式

归纳法	良基关系的定义
数学归纳法	$m \prec n$ 当且仅当 $m+1=n$
第二数学归纳法	$m \prec n$ 当且仅当 $m<n$
结构归纳法	$m \prec n$ 当且仅当 m 是 n 的子表达式

上的良基关系 $\succ_{A\times B}$ 为:

$$(a_1,b_1)\succ_{A\times B}(a_2,b_2)\triangleq\begin{cases}a_1\succ_A a_2;\\ a_1=a_2 \text{ 且 } b_1\succ_B b_2。\end{cases}$$

直观上，上述定义表明二元组满足良基关系 $\succ_{A\times B}$，要么第一元满足良基关系 $\succ_A$；要么在第一元相等的情况下，第二元满足良基关系 $\succ_B$。

例 1.11　给定自然数集合 $\mathbb{N}$ 上的良基关系 $>$；下面的序列定义了在笛卡尔积 $\mathbb{N}\times\mathbb{N}$ 上的良基关系 $\succ$:

$$\begin{aligned}(4,30)&\succ(4,20)\\ &\succ(3,80)\succ(3,60)\succ(3,10)\\ &\qquad\qquad\succ(2,90)\succ(2,80)\end{aligned}$$

显然，我们可以把上述笛卡尔积上的良基关系，推广到 n 个集合的情况；我们把这个推广的细节作为练习留给读者。

进一步,对于两个不一定等长的元组 $(a_1,\ldots,a_m)$ 和 $(b_1,\ldots,b_n)$，我们仍可以定义其良基关系:

$$(a_1,\ldots,a_m)\succ(b_1,\ldots,b_n)\triangleq\begin{cases}m>n;\\ a_1\succ b_1;\\ a_1=b_1 \quad 且 \quad (a_2,\ldots,a_m)\succ(b_2,\ldots,b_n)。\end{cases}$$

特别的，如果元素 a_i、$1\le i\le m$ 和 b_j、$1\le j\le n$ 都属于集合 $\{1,\ldots,9\}$ 的话；上述良基关系本质上定义了任意两个自然

数间的大小排序，因此经常被称为**字典序良基关系**（Lexicographic well-founded order）。

需要注意，字符串之间通常的字典序并不满足良基关系，我们把对这个结论的证明作为练习留给读者。

1.5 文法实现

在后续章节，我们经常需要对给定的文法进行编程处理。本小节，我们讨论对文法进行实现的一般技术。为统一起见，本书都使用 Python 程序设计语言给出示例实现，但需要注意的是，本书任何理论和技术都不依赖于 Python 语言，读者也可以根据本书讨论的基本原理和技术，使用其它任何合适的程序设计语言进行实践。

对文法的实现需要用到**抽象语法树**（Abstract syntax tree）的概念，它是编译器内部对文法的编码实现。一般的，对于一般形式的产生式

$$\begin{aligned} A_1 &::= \alpha_{11} \mid \ldots \mid \alpha_{1m} \\ &\vdots ::= \vdots \\ A_n &::= \alpha_{n1} \mid \ldots \mid \alpha_{nk} \end{aligned}$$

在用 Python 语言进行实现时，我们对每个非终结符 A_i、$1 \leq i \leq n$ 都需要定义一个父类 `A_i`；而对每个非终结符的右部中的每种情况，我们都需要从其左部的父类中继承一个子类，对该右部的相应信息进行合理编码实现。

作为示例，我们重新考虑上一小节给出的文法：

$$E ::= E + T \mid E - T \mid T$$
$$T ::= T * F \mid T/F \mid F$$
$$F ::= (E) \mid n$$

为了对文法进行实现，我们对三个左侧的非终结符，分别定义了三个类 `E`、`T` 和 `F`；这些类并没有实际的功能，只是作为父类让子类来进行继承：

```
class E:
  pass

class T:
  pass

class F:
  pass
```

需要注意，本小节给出的示例代码，主要强调代码的完整性和精确性，并不追求代码量最小。

接着，我们为产生式的每个右侧定义一个类，该类继承相应的左部。例如，考虑非终结符 E，它有三个右部，因此我们定义三个类 `EAdd`、`ESub` 和 `ETerm`：

```
from dataclasses import dataclass

# E ::= E+T
@dataclass
class EAdd(E):
  e: E
  t: T

  def __str__(self):
    return f"{self.e} + {self.t}"

```

```
# E ::= E-T
@dataclass
class ESub(E):
  e: E
  t: T

  def __str__(self):
    return f"{self.e} - {self.t}"

# E ::= T
@dataclass
class ETerm(E):
  t: T

  def __str__(self):
    return str(self.t)
```

这三个类都继承自同一个父类 `E`；并且每个类都包含了默认构造方法和字符串方法 `__str()__`。需要注意的是，我们使用了 Python 3.7 之后引入的精简的构造方法的语法 `dataclass`，它等价于普通的构造方法的写法。例如，对于 `EAdd` 类，其等价的构造方法 `__init()__` 是

```
# E ::= E+T
class EAdd(E):
  def __init__(self, e, t)
    self.e = e
    self.t = t
```

使用 `dataclass`，而不是像如上的显式构造方法，有好几个主要优势：首先，每个类的字段及其类型都显式化了，有助于 Python 编译器进行错误检查；其次，有效减少了代码量。

另外，还需要注意：上面示例代码还使用了 Python 3.8 之后引入的新的 `f` 格式化字符串。因此，读者需要安装至少 Python 3.8 或更高的版本的编译器来编译运行上述代码。

类似的，我们可以为非终结符 T 的右部进行编码实现：

```
# T ::= T*F
@dataclass
class TMul(T):
  t: T
  f: F

  def __str__(self):
    return f"{self.t} * {self.f}"

# T ::= T/F
@dataclass
class TDiv(T):
  t: T
  f: F

  def __str__(self):
    return f"{self.t} / {self.f}"

# T ::= F
@dataclass
class TFactor(T):
  f: F

  def __str__(self):
    return str(self.f)
```

我们用三个类 `TMul`、`TDiv` 和 `TFactor` 分别表示产生式右部的三种不同情况，它们也都包含默认的构造方法和字符串方法。

最后，我们给出对非终结符 F 的产生式右侧的实现：

```
# "(E)"
@dataclass
class FExp(F):
  e: E

```

```
  def __str__(self):
    return f"({self.e})"

# "n"
@dataclass
class FNum(F):
  n: int

  def __str__(self):
    return str(self.n)
```

这两个类 FExp 和 FNum，分别编码了非终结符 F 的两个右部。

根据这些 Python 类，我们可以构造具体表达式的数据结构表示。例如，表达式 $1+2*3$，我们可以用如下数据结构表示：

```
e = EAdd(ETerm(TFactor(FNum(1))),
        TMul(TFactor(FNum(2)),
            FNum(3)))
```

基于对文法的 Python 数据结构表示，文法上的操作可表达成对应 Python 数据结构上的操作或者函数。例如，我们可以实现对上述文法生成的表达式的求值操作；为此，我们分别在三个非终结符 E、T 和 F 对应的类 E、T 和 F 上分别定义一个求值函数：

```
int eval_E(e: E):
  if isinstance(e, EAdd):
    return eval_E(e.e) + eval_T(e.t)
  if isinstance(e, ESub):
    return eval_E(e.e) - eval_T(e.t)
  if isinstance(e, ETerm):
    return eval_T(e.t)

int eval_T(t: T):
```

```
10   if isinstance(t, TMul):
11     return eval_T(t.t) * eval_F(t.f)
12   if isinstance(e, TDiv):
13     return eval_T(t.t) / eval_F(t.f)
14   if isinstance(e, TFactor):
15     return eval_F(t.f)
16
17 int eval_F(f: F):
18   if isinstance(f, FExp):
19     return eval_E(f.e)
20   if isinstance(f, FNum):
21     return f.n
```

这三个函数 `eval_E`、`eval_T` 和 `eval_F` 分别接受类 `E`、`T` 和 `F` 类型的值作为参数，它们的函数体对参数可能的类型进行分情况讨论，最后返回其计算得到的整型值。

1.6 本章小结

本章，我们主要介绍形式化方式所涉及到的一些基础知识，包括集合、关系与映射、上下文无关文法和归纳法；并结合 Python 程序设计语言讨论了文法实现的一般技术。

1.7 深入阅读

Herbert B. Enderton[1] 系统讲述了集合论相关理论；Hopcroft[2] 讨论了自动机和文法；Mitchell[3] 讨论计算机科学中常用的各种归纳法；Appel[4] 给出了对文法实现的编译器实现技术；Python 的官方网站上 [5] 包括安装包及丰富的学习材料；Matthes[6][7] 给出了 Python 语言的详细介绍。

1.8 思考题

1. 列举集合 $\{1，3，5，7，9\}$ 的所有子集和真子集。

2. 已知集合 A 满足

$$\{a，b，c\} \subseteq A \subset \{a，b，c，d，e\}，$$

请列出所有满足该条件的集合 A。

3. 设集合 $A = \{a，b，c\}$ 和集合 $B = \{0，1\}$，请写出所有从集合 A 到集合 B 的映射。

4. 设集合 $M = \{0，-1，1\}$ 和集合 $N = \{-2，-1，0，1，2\}$ 存在映射 $f : M \to N$，且满足条件：对任意 $x \in M$，$x + f(x)$ 是奇数，列出所有满足该条件的映射。

5. 请根据例（1.3）中的文法，给出 $1 + 2 * 3$ 的另外一种不同的推导。

6. 请用数学归纳法证明，对任意集合 A

$$|\mathcal{P}(A)| = 2^{|A|}。$$

7. 给定英文字母的集合 $\Sigma = \{a, \ldots, z\}$，定义 Σ 上的字符串是若干个英文字母的连接；特别的，不包含任何字母的空串记为 ϵ。

1). 定义字符串上的二元关系 $\succ$ 如下：

$$s \succ t \quad \text{当且仅当} \quad |s| > |t|,$$

其中 $|s|$ 表示字符串 s 的长度。试证明二元关系 $\succ$ 是良基关系。

2). 定义字符串上的二元关系 $\succ$ 如下：

$s \succ \epsilon$ 当且仅当 $s \neq \epsilon$

$a\,s \succ b\,t$ 当且仅当 a比b在Σ中靠前；或$a = b$且$s \succ t$。

试证明二元关系 $\succ$ 不是良基关系。