第3章　Z的类型与构造单元

本章介绍Z的表示法，叙述在Z中是如何定义有关元素和结构的数学与逻辑的表示以及如何应用这些表示的。由第2章可知，一个集合中的所有元素一般是同一“类”对象。在Z中，这种“类”可以用类型系统来形式定义。Z的一个重要特征是类型化，也即，Z是一个类型化的语言。引入类型有几个优点。对规格说明的书写者来讲，最明显的优点是能够使用一个类型检查的程序查出规格说明中关于数据定义方面的错误。另一个优点是类型系统对规格说明施加了结构，并要求规格说明的书写者使用规定的写法来写规格说明。引入类型的一个理论方面的理由是：可以避免关于集合论的一些“悖论”。Z规格说明中各种表达式的类型是可以自动计算的，目前，已有一些系统实现了Z的规格说明的类型检查。

3.1　Z的类型系统

类型是一个受到限定的“类”的表达式，它表示了对象的一个聚集。一个类型是一个给定集合的名字，或是由简单的类型使用了一个或多个类型构造符构造出来的复合类型。对于类型，了解程序设计语言的人都不陌生。几乎所有的程序设计语言都有类型的概念。类型系统要求，在Z中，任何一个新的变量在被使用以前必须以一个类型对其进行声明，类型决定了变量可取值的范围。Z有幂集类型、笛卡儿积类型、序列、包和模式类型等复合类型。在Z中，不仅变量有类型，每一个表达式，如集合等也一定有一个类型。如果定义了一个表达式，则表达式的值就是这个表达式的类型的一个成员，或叫元素。

变量声明给变量指定了一个名字和一个类型。例如Book是一个类型，则声明：

$$x : Book$$

引入了类型Book的一个变量x。在一个Z文档中的许多地方都会出现变量声明，例如当写一个谓词时会用到变量声明：

$$\forall c : city \bullet c \text{ is in Europe} \Rightarrow \text{SaoPaulo is bigger than } c$$

点符号“•”用来分隔量词约束变量和谓词表达式。$\forall c : city \bullet c$ is in Europe读作：“对所有的city c，使得c is in Europe”。

又例如，当定义集合时也会使用变量声明：

$$Bigcountries == \{a : country \mid a \text{ has more than 40 million inhabitants}\}$$

在上述例子中使用了类型city和country，并以c和a分别表示类型city和country的变量。

3.1.1　基本类型

每一个Z规格说明都引入并使用了基本类型(Basic Type)。这些基本类型有时被称为给定集合(Given Sets)，可由一对方括号括起来的名所组成的基本类型定义引入。基本类型定义是一个类型声明，它引入了一个或多个基本类型。如

$$[Book, Car]$$

引入了基本类型 Book 和 Car。通常对这些集合的内部结构并不关心，而感兴趣的是在一个规格说明引入它们以后，就有了可使用的类型。也可以用这些类型来声明变量。同一个规格说明中不同的基本类型没有相同的元素。例如，若 x 被声明为类型 Book 的变量，则 x 就不能成为类型 Car 的变量。一个重要的基本类型是整数类型。

$$\cdots -3, -2, -1, 0, 1, 2, 3, \cdots$$

它表示为$\mathbb{Z}$。基本类型$\mathbb{Z}$是内定义类型，不需再声明。这样，给出以下声明：

$$x : \mathbb{Z}$$

则表达式如 $x=150$、$3 * x > x$、$x^2 - 3 * x + 4 = 0$ 的类型都是正确的。应该注意到：$x = x + 2$ 和 $x \neq x$ 的类型是正确的，尽管没有一个 x 的值能使这两个等式成立。

除了上面表达式中用到的“+”、“−”、“*”等算术运算符和“=”、“>”等关系运算符外，在 **Z** 中还可以使用“整除”(div)、“模”(mod)算术运算符和“<”、“≤”、“≥”关系运算符。下述定律描述了这些运算符的一些性质：

$b > 0 \Rightarrow 0 \leqslant a \bmod b < b$　　$b \neq 0 \Rightarrow a = (a \operatorname{div} b) * b + a \bmod b$

$b \neq 0 \wedge c \neq 0 \Rightarrow (a * c) \operatorname{div} (b * c) = a \operatorname{div} b$

基本类型是 **Z** 的类型系统中的重要组成部分，对基本类型重复使用类型构造符，可构造出其他的类型。现介绍两个复合类型：幂集类型和笛卡儿积类型。另外还有两个复合类型：序列和包，将在第 6 章中进行专门介绍。

3.1.2 幂集类型

可先考虑集合，给出以下一些例子。

[**例 3.1**]

$$single == \{2\}$$
$$smallodd == \{1, 3, 5, 7\}$$
$$\mathbb{N} == \{n : \mathbb{Z} \mid n \geqslant o\}$$
$$\mathbb{N}_1 == \{n : \mathbb{Z} \mid n > o\}$$

上述每一个集合中的每一个元素都是类型$\mathbb{Z}$的元素，因此 single、smallodd、$\mathbb{N}$ 和$\mathbb{N}_1$都是$\mathbb{Z}$的子集。那么要问：上述每一个集合本身的类型是什么？显然集合本身不是一个数，因此也就不是类型$\mathbb{Z}$。在这样的情况下，使用 2.4 节中所介绍的幂集构造符 $\mathbb{P}$ 来定义上述集合的类型。如果 S 是一个集合，则 $\mathbb{P}S$ 也是一个集合，称为幂集，它含有集合 S 的所有子集。

使用一个规格说明中的基本类型和幂集构造符，就可以产生一个新的类型，称为该基本类型的幂集类型。如果有基本类型 S 和 T，则$\mathbb{P}S$ 和$\mathbb{P}T$ 都是合法的类型，$\mathbb{P}\mathbb{P}S$ 和$\mathbb{P}\mathbb{P}T$ 等也都是合法的类型。任何具有同一类型 T 的对象组成的集合本身就是集合类型$\mathbb{P}T$ 的对象。

对于集合 smallodd，其幂集类型可见例 3.2。

[**例 3.2**]

$$\mathbb{P}\ smallodd = \{\varnothing, \{1\}, \{3\}, \{5\}, \{7\}, \{1,3\}, \{1,5\}, \{1,7\}, \{3,5\}, \{3,7\}, \{5,7\},$$
$$\{1,3,5\}, \{1,3,7\}, \{1,5,7\}, \{3,5,7\}, \{1,3,5,7\}\}$$

幂集 $\mathbb{P}\mathbb{Z}$ 是 $\mathbb{Z}$ 的所有子集的集合。有些集合显然太长而不可能将其所有元素显式列出，但尽管如此，还是容易看出一个集合是否为 $\mathbb{P}\mathbb{Z}$ 的元素，因为由幂集的定义可知有：

$$X \in \mathbb{P}\mathbb{Z} \Leftrightarrow x \subseteq \mathbb{Z}$$

即一个集合是$\mathbb{P}\mathbb{Z}$ 的元素当且仅当它是$\mathbb{Z}$ 的子集。因此，所有元素都是整数的任一集合，本身也是$\mathbb{P}\mathbb{Z}$ 的元素。这样就可引出下述的例子。

［**例 3.3**］

$$\text{single} \in \mathbb{P}\mathbb{Z}$$
$$\text{smallodd} \in \mathbb{P}\mathbb{Z}$$
$$\{2,4,6,8,\cdots\} \in \mathbb{P}\mathbb{Z}$$
$$\mathbb{N} \in \mathbb{P}\mathbb{Z}$$
$$\mathbb{Z} \in \mathbb{P}\mathbb{Z}$$

有了幂集类型，就可用它来声明变量。当欲声明一个变量，且其值为对象的集合时，就可用幂集构造符来形成该变量的类型。这样，对由所有的质数所构成的集合 Primes 可作以下声明：

$$\text{Primes} : \mathbb{P}\mathbb{Z}$$

它是整数集的一个子集。

对于基本类型 Book，若要定义一个对象 library，则可以声明：

$$\text{library} : \mathbb{P}\ \text{Book}$$

它声明 library 是集合 Book 的一个子集。

可以将幂集构造符应用于任何集合来构成另外一个集合。这样，可由一个类型产生另一个新的类型。集合和类型有如下关系：一方面，每一个类型是一个集合。若引入了一个基本类型或者使用$\mathbb{P}$ 构造了一个类型，则可以把这个类型看成是一个集合，还可引用它的元素，并可使用有关的集合操作；另一方面，当某个元素是多个集合的元素时，它只能属于一个类型。例如整数 2 是$\{2\}$、$\{-2,2\}$、$\mathbb{N}$、$\mathbb{N}_1$、$\mathbb{Z}$ 和很多集合中的一个元素，但只能以它们中的某一个作为类型，这里应是类型$\mathbb{Z}$。这样，类型就是一种特定的类的集合，它被看成是一个“最大集合”，因为它代表了一个元素可以属于的最大的可能的集合。现在就可以明确地回答：single、smallodd、$\mathbb{N}$ 和 $\mathbb{N}_1$ 的类型是$\mathbb{P}\mathbb{Z}$。

使用基本类型和幂集类型，就可确定一个关于集合的语句是否正确。如对于$\{1\} \subseteq$ smallodd，类型检查的结果是正确的。符号$\subseteq$如果出现在两个具有相同类型的集合之间，则是有意义的。也就是说，$\subseteq$左边的项和右边的项必须都是某个 X 的幂集类型$\mathbb{P}X$。在本例中，$\{1\}$和 smallodd 的类型都是 $\mathbb{P}\mathbb{Z}$，因而表达式的类型是正确的。

3.1.3　笛卡儿积类型

在第 2 章里，已经引入了有序元组的概念。这里将使用笛卡儿积作为类型构造符，由已知的类型构造出新的类型。

［**例 3.4**］　定义两个元组：

(1,1)

($\{6,3\}$,green,Shanghai)

前者定义一个二元组(数,数)，二元组一般称为序偶。后者是一个(数的集合，颜色，城

市)的三元组。现在可给出 **Z** 中描述元组类型的形式表示。给定任何两个集合 R 和 S,其笛卡儿积 R×S 是所有这样的序偶的集合,这些序偶的第一个元素是 R 中的一个元素,其第二个元素是 S 中的一个元素。

[**例 3.5**] 定义 R==={a,b},S==={1,2,3},则

$$R \times R = \{(a,a),(a,b),(b,a),(b,b)\}$$

$$R \times S = \{(a,1),(a,2),(a,3),(b,1),(b,2),(b,3)\}$$

$$R \times \mathbb{N} = \{(a,1),(a,2),(a,3),\cdots,(b,1),(b,2),(b,3),\cdots\}$$

笛卡儿积还可定义于三个或更多的集合。例如三个集合 R,S 和 T 的笛卡儿积 R×S×T 含有所有的有序的三元组的集合,第一个元素来自 R,第二个元素来自 S,而第三个元素来自 T。

当已有了若干个集合,它们都已有类型,则它们的笛卡儿积就是一个新的类型:笛卡儿积类型。给定了一个有序的元组,只要检查一下它的每个元素的类型就可确定该元组应属于哪一个笛卡儿积类型。例如

$$(1,1) \in \mathbb{Z} \times \mathbb{Z}$$

$$(\text{Beijing},\text{red}) \in \text{cities} \times \text{colour}$$

这里 cities 和 colour 都是已定义的给定类型。

笛卡儿积和幂集类型构造符可以一起使用,由基本类型构造新的类型。例如,如果一个规格说明中有基本类型 R 和 S,已知 $\mathbb{P}R$,$\mathbb{P}S$,$\mathbb{P}\mathbb{P}R$,$\mathbb{P}\mathbb{P}S$…都是类型,现在又知 R×R,R×S,S×R 和 S×S 也都是类型。将两个类型构造结合在一起使用,可得到$(\mathbb{P}R)\times(\mathbb{P}R)$,$\mathbb{P}(R\times S)$,$(\mathbb{P}R)\times S$ 等新的类型。例如,由 cities 和 colour 则可有$(\mathbb{P}\,\text{cities})\times(\mathbb{P}\,\text{colour})$、$\mathbb{P}(\text{cities}\times\text{colour})$和$(\mathbb{P}\,\text{cities})\times\text{colour}$等类型。

现在就可将这些类型使用于对象声明了。假定在一个图书馆系统中,一本书的每一个复本可由书和其复本号来识别。(因为一本书可能有几个复本,为了区别,所以引入了复本号。)如果要写该系统的一个规格说明,则可将每一个复本表示为类型 Book×ℤ 的序偶。因此,若以一个特定的变量 newestbook 来表示最近刚进入到书库的书,则可以写成声明:

$$\text{newestbook}: \text{Book}\times\mathbb{Z}$$

一个图书馆可看成是书的复本的集合,因此就有

$$\text{library}: \mathbb{P}(\text{Book}\times\mathbb{Z})$$

除了已经介绍的两种构造类型——幂集类型和笛卡儿积类型以外,还有三个构造类型:模式类型在第 5 章中介绍,而序列和包则在第 6 章中介绍。

3.1.4 对象声明

前面提到一个对象可能是多个集合中的一个元素,但它只能是唯一的一个类型的元素。在 **Z** 中,每一个对象(变量或值)都被赋予一个类型。每一个变量的类型是在对象声明时确定的。这样,如果 Book 是一个类型,就可以声明一本书 x 为

$$x: \text{Book}$$

迄今,所知道的对象声明的形式为 x:T,这里 T 是一个类型。当已经定义了一个集合:

$$\text{EvenNum} == \{x:\mathbb{Z} \mid \exists y:\mathbb{Z} \bullet y > 0 \wedge x = 2 * y\}$$

并且希望引入一个数 oneeven,该数是偶数。于是,就可以有声明:

$$oneeven:\mathbb{Z}$$

然后再用谓词写出“oneeven 是偶数”的表达要求：

$$\exists y:\mathbb{Z} \bullet y>o \wedge oneeven=2*y$$

这种表达方法显然是不简洁的。因此可引入一个对象声明的缩写形式，上述的表达可写成对象声明的表达形式如下：

$$oneeven:EvenNum$$

这个声明不能理解为 oneeven 的类型是 EvenNum，事实上，oneeven 的类型是$\mathbb{Z}$。从这样的声明中如何知道 oneeven 的类型是$\mathbb{Z}$呢？该行声明表明：oneeven 是集合 EvenNum 中的一个元素，而 EvenNum 的定义告诉了我们，它的每一个元素的类型都是$\mathbb{Z}$，所以 oneeven 的类型一定是$\mathbb{Z}$。

由此可知：一个声明可以是 x：T 的形式，其中 T 是一个类型，或是 x：S 的形式，其中 S 是某一个类型的子集。显然 y：$\mathbb{N}$ 是后一种形式，y 是$\mathbb{N}$ 的一个对象，其类型是$\mathbb{Z}$，因为$\mathbb{N}$ 是类型$\mathbb{Z}$ 的一个子集。对象声明的缩写方法允许使用已经引入的集合的定义来写声明。这种声明缩写方法使得写出来的规格说明简短而且易读。

对于 y：S 这种形式的声明，必须强调两点：①通过检查 S 的定义可知它的元素必须具有的类型，从而确定 y 的类型；②限制了 y 必须满足 S 所定义的性质。

若欲声明某一个无穷类型或无穷集合的有穷子集，就可以使用声明缩写的表示方法。现已知$\mathbb{F}$S 表示了一个给定集合 S 的所有有穷子集的集合。因为$\mathbb{P}$S 是集合 S 的所有子集的集合，因此有

$$\mathbb{F}\ S \subseteq \mathbb{P}\ S$$

应该注意到，如果 T 是一个类型，则$\mathbb{P}$T 也还是一个类型，而一般来说$\mathbb{F}$T 不是一个类型。这样，如果声明

$$fset:\mathbb{F}\ \mathbb{Z}$$

则这是一个缩写形式的声明，因为 fset 的类型是$\mathbb{P}\ \mathbb{Z}$，而不是$\mathbb{F}\ \mathbb{Z}$。

具有相同类型的对象可组成一个对象表由该类型来声明，如

$$x:\mathbb{Z}$$
$$y:\mathbb{Z}$$

可写成：

$$x,y:\mathbb{Z}$$

而声明：

$$thiscopy:Book \times \mathbb{Z}$$
$$thatcopy:Book \times \mathbb{Z}$$
$$newestcopy:Book \times \mathbb{Z}$$

可写成：

$$thiscopy,thatcopy,\ newestcopy:Book \times \mathbb{Z}$$

声明部分是 **Z** 规格说明的一个重要部分。

3.1.5　枚举类型

有时想要定义一个只含有几个元素的类型，并对这几个元素给定名称。首先定义一个

叫做 YesNo 的类型声明的例子，该类型仅有两个值：yes 和 no，该类型定义可写为：

$$YesNo ::= yes \mid no$$

这种形式的定义称为枚举类型(Enumerated Types)定义，有时也称为简单类型定义。符号“$::=$”是数据类型定义符，符号“|”是分隔符。实际上，这个数据类型定义是一种简写形式，它等价于下述声明和谓词：

$$[YesNo]$$

$$yes, no : YesNo$$

$$yes \neq no$$

$$YesNo = \{yes, no\}$$

即该数据类型定义的含义是，YesNo 是一个给定集合类型，yes 和 no 是它的两个不同的值，并且 yes 和 no 是该类型仅有的两个值。

显然采用枚举类型定义的写法比其等价的一串声明和谓词要简洁得多，这就使得书写规格说明更方便。

3.2 扩充表示法

3.2.1 量词化扩充表示法

前述已介绍可以用声明和有关的性质来描述谓词，例如：

$$\forall y : \mathbb{Z}; x : \mathbb{Z} \bullet y - x \in \mathbb{Z}$$

$$\exists t : \mathbb{Z} \bullet t < 0$$

若要声明多个具有相同类型的变量，则可将这些变量放在一起而不必重复写它们的类型名。这样前一个命题就可写成

$$\forall y, x : \mathbb{Z} \bullet y - x \in \mathbb{Z}$$

对于量词化的这种谓词还可扩充其表示法。当写 **Z** 的谓词时，可以用一个另外的性质与声明部分一起来构成谓词，这个附加的性质进一步限制了声明的变量。例如，假定希望写一个谓词：“对$\mathbb{Z}$ 的某一个子集 S 和一个整数 x，x 是 S 的一个元素，x 等于 S 中元素的个数”。为了写出这个谓词，须有一个能表示“集合中元素的个数”的操作符，在 **Z** 中，使用操作符“#”可得到集合中的元素个数。

下述两条定律描述了“#”的有关性质。

$$\#(S \cup T) = \#S + \#T - \#(S \cap T) \qquad \mathbb{F}_1 X = \{S : \mathbb{F}\ X \mid \#S > 0\}$$

[**例 3.6**]

$$\#\{a, b, c\} = 3$$

$$\#\,smallodd = 4$$

上述谓词：“对 $\mathbb{Z}$ 的某一个子集 S 和一个整数 x，x 是 S 的一个元素，x 等于 S 中元素的个数”就可写成：

$$\exists x : \mathbb{Z}; S : \mathbb{P}\,\mathbb{Z} \mid x \in S \bullet x = \# S$$

可读作“存在一个整数 x 和一个整数的集合 S，并有 $x \in S$，使得 $x = \#S$”，附加的限制 x

$\in S$ 是 x 和 S 必须满足的条件。对于全称量词化的谓词，也有类似的扩充表示。比如

$$\forall x, y: \mathbb{Z} \mid x > y \bullet x - y \in \mathbb{N}_1$$

读作："对所有的满足 x>y 的 x 和 y，x－y 属于$\mathbb{N}_1$"。这里，附加的 x>y 就是 x 和 y 必须满足的条件。这个谓词是使用了限制类型的谓词。在声明中使用更有限制的集合可使谓词更简洁，更易理解。

考虑公式：

$$\forall x: \mathbb{Z} \mid x \in \mathbb{N}_1 \bullet x - 1 \in \mathbb{N}$$

该式等价于使用更有限制的集合作为类型的谓词：

$$\forall x: \mathbb{N}_1 \bullet x - 1 \in \mathbb{N}$$

事实上，虽然表示法已被扩充为可允许使用更方便的表达式，但在语言的功能上并未增加什么。同一个表达式有时可以用几种不同的方式书写。一般的谓词表达式

$$\forall Decs \mid Constr \bullet Pred$$

等价于更常规的形式表示：

$$\forall Decs \bullet Constr \Rightarrow Pred$$

其含义为：对所有的 Decs，如果 Constr 被满足，则 Pred 一定被满足。类似的表达式：

$$\exists Decs \mid Constr \bullet Pred$$

等价于更常规的形式表示：

$$\exists Decs \bullet Constr \wedge Pred$$

其含义为：存在 Decs，它满足 Constr，而且它也满足 Pred。

［**例 3.7**］　对以下陈述：

SaoPaulo is bigger than any other city in the same country，将其用谓词形式来描述。

根据该描述的含义，可有如下谓词表示：

$$\begin{aligned}&\exists Co: Country \bullet \\ &\quad SaoPaulo\ is\ in\ Co \wedge \\ &\quad \forall ci: city \bullet \\ &\quad ci\ is\ in\ Co \wedge \neg ci\ is\ SaoPaulo \\ &\quad \Rightarrow SaoPaulo\ is\ bigger\ than\ ci\end{aligned}$$

上述谓词可被重写为：

$$\begin{aligned}&\exists Co: Country \mid SaoPaulo\ is\ in\ Co \bullet \\ &\quad \forall ci: city \bullet ci\ is\ in\ Co \wedge \neg ci\ is\ SaoPaulo \\ &\quad \Rightarrow SaoPaulo\ is\ bigger\ than\ ci\end{aligned}$$

再进一步可重写为：

$$\begin{aligned}&\exists Co: Country \mid SaoPaulo\ is\ in\ Co \bullet \\ &\quad \forall ci: city \mid ci\ is\ in\ Co \wedge \neg ci\ is\ SaoPaulo \bullet \\ &\quad SaoPaulo\ is\ bigger\ than\ ci\end{aligned}$$

使用这种表示，长表达式可以写得清楚、简短，并可删去逻辑连接词如"∧"和"⇒"。

3.2.2　集合表达式扩充表示法

当构成一个集合时，可以使用声明加限制的组合表示，如

$$\{x:\mathbb{N} \mid x<5\}$$

$$\{n:\mathbb{Z} \mid n>o\}$$

这里只有一个声明，但实际上可使用一个声明的表。若给出的是声明的表，则集合的元素就是满足集合谓词的元组，元组的元素顺序是由声明确定的。如：

$$\{x:\mathbb{N};\ S:\mathbb{P}\,\mathbb{N} \mid S=\{x+1\} \wedge x<5\}$$

是集合

$$\{(0,\{1\}),(1,\{2\}),(2,\{3\}),(3,\{4\}),(4,\{5\})\}$$

而

$$\{x,y,z:\mathbb{Z} \mid z=x+y\}$$

是一个无穷的集合，该集合含有(0,0,0),(0,1,1),(1,2,3),(2,1,3),(3,2,5)等元组。

用声明加限制的组合表示集合，其一般形式为：

$$\{Decls \mid Pred\}$$

有时并不希望集合的元素是由声明 Decls 的表所确定的元组，而想自由地选择某些元素。**Z** 扩充了集合表达式原有的表示，增加了一个指明集合元素形式的部分。扩充后的集合表达式定义形如：

$$\{Decls \mid Pred \bullet ExPr\}$$

它表示了所有这样的值的集合，这些值是由 Decls 中满足限制 Pred 的变量以 ExPr 项的形式确定的。竖线“|”将声明和限制分隔开来，称为“限制条”(ConstraintBar)，黑圆点将限制与表达式项分隔开来。整个表示以一对花括号括起。

［**例 3.8**］ 给出一些例子：

(1) $\{x:\mathbb{N} \mid x\leqslant 5 \bullet x^2\}$

(2) $\{i,j:\mathbb{Z} \bullet i+j\}$

(3) $\{i:\mathbb{N} \mid 1\leqslant i\leqslant 8 \bullet (i,i)\}$

(1) 式所表示的集合是这样构成的，对 $n:\mathbb{N}$，满足 $n\leqslant 5$，取其平方，该集合为

$$\{0,1,4,9,16,25\}$$

注意到，集合中元素的类型就是集合定义中项表达式的类型。

(2) 式所表示的集合考虑了整数 i 和 j，没有限制谓词，集合的元素由 i 加上 j 得到的结果组成，因此也是整数，这个集合也就是整个$\mathbb{Z}$。

(3) 式考虑的是国际象棋棋盘上由 1 到 8 的序偶所组成的位置，可以用这个集合来表示某一条对角线。由于 i 的取值范围为 1～8，对每一个可能的值 i，可取序偶(i,i)。

虽然没有在语言中增加新的功能，但新的集合表达式形式使得在书写集合定义时更方便，其形式更简明紧凑。

在读这种形式的集合定义时可不必读出限制条和黑圆点，只要易于理解即可。集合

$$\{x:\mathbb{N} \mid x\leqslant 5 \bullet x^2\}$$

可读作：小于等于 5 的自然数平方的集合。

3.2.3 Z 的基本库

为了方便地使用各种数学表示，在 **Z** 中建立坚实的数学基础并使 **Z** 的使用者易于书写软件规格说明，我们考虑为 **Z** 的表示方法建立一个基本库，基本库中包括各种数学定义。

迄今，库中可含有整数的基本类型$\mathbb{Z}$，非负整数的集合$\mathbb{N}$和正整数的集合$\mathbb{N}_1$的定义部分，这些部分包括各种有用的操作和关系，如＋、－、＊、/、＜、≤、＞、≥等。当然还有许多定义和定律，随后的几章将逐步介绍。它们都将包含在基本库中。当需要时，就可直接在规格说明中使用这些定义和定律，而不必再进行声明了。

下面给出一些 **Z** 中简写某些定义的例子，若想定义 a 到 b 之间（包括 a 和 b）的所有整数的集合，就可以直接使用 a..b 的表示。“..”称做数域。

［**例 3.9**］ $3..9=\{3,4,5,6,7,8,9\}$

$4..4=\{4\}$

$9..7=\varnothing$

$-1..1=\{-1,0,1\}$

根据 3.2.1 节中“#”的定义，因此有：

$$\#(3..9)=7$$

$$\#(4..4)=1$$

$$\#(9..7)=0$$

$$\#(-1..1)=3$$

关于数域有下述三条定律。

$a>b \Rightarrow a..b=\varnothing$　　$a..a=\{a\}$

$a\leqslant b \wedge c\leqslant d \Rightarrow b..c \subseteq a..d$

3.3　Z 规格说明的构造单元

前面已经引入了规格说明语言 **Z** 的一些类型成分，介绍了如何建立类型定义和声明，如何写谓词以及如何构造集合。

一个 **Z** 的文档是由形式的数学描述和非形式（如同散文性）的解释所组成。形式的描述就是大家所关心的形式规格说明，它由若干个被称为段落（Paragraph）的单位所组成。段落是 **Z** 语言表示的核心，它引入了规格说明的基本类型、全程变量和模式。这一节介绍 **Z** 的符号和 **Z** 的构造单元：公理定义，模式和通用式定义。**Z** 的详细的语法规则描述在附录 A 中给出。

3.3.1　Z 的符号

标识符是 **Z** 规格说明中最简单的语法单位，它是一个非空的以字母开头并由若干个（零个或多个）大小写字母、数字或下划线组成的符号序列，或是一个特殊符号。**Z** 除了含有一般的程序设计语言中常用的 ASCII 码符号外，还含有其他专用的数学符号，这些数学符号或是运算符，或是具有特殊含义的符号，如$\mathbb{P}$、$\mathbb{N}$、$\mathbb{N}_1$、$\mathbb{F}$、$\mathbb{Z}$ 等。

Z 的标准的运算符分为前缀、中缀和后缀三种形式。下面给出这些运算符。

中缀函数符（按优先级从高到低的顺序列出）：

$\lhd$，$\rhd$，$\lhd\!\!\!\!-$，$\rhd\!\!\!\!-$

$\oplus$，#

$*$,div,mod,$\bigcap$,$\upharpoonright$,$\restriction$,$\oplus$,$\otimes$,。

$+$,$-$,$\cup$,$\setminus$,$\frown$,$\uplus$,$\dot{\cup}$

$..$

$\mapsto$

后缀函数符：

$\_^{\sim}$,$\_^{*}$,$\_^{+}$

"_"是形式的自变量。

中缀关系符：

$\neq$,$\in$,$\notin$,$\subseteq$,$\subset$,$<$,$\leqslant$,$>$,$\geqslant$

前缀关系符：

disjoint

中缀通用符：

$\leftrightarrow$,$\rightarrow$,$\nrightarrow$,$\rightarrowtail$,$\rightarrowtail$,$\twoheadrightarrow$,$\twoheadrightarrow$,$\twoheadrightarrowtail$,$\twoheadrightarrowtail$

后缀通用符：

$\mathbb{P}$,$\mathbb{P}_1$,**id**,$\mathbb{F}$,$\mathbb{F}_1$,**seq**,**seq**$_1$,**iseq**,**bag**

此外还有各种逻辑符号和其他符号：

$\forall$,#,$\neg$,$\wedge$,$\vee$,$\Rightarrow$,$\Leftrightarrow$,$\triangleq$,:,|,,,$\bullet$,$=$$=$,$::=$,《,》,$\times$,

$[\![$,$]\!]$,(,),λ,μ,θ,Δ,Ξ

在上述符号中，有些符号的含义已经介绍，另一些符号的含义在后面出现时再介绍。

3.3.2 公理定义

本书中重点介绍一个例子。在操作系统中，有一个可由用户访问的空闲存储块的管理程序。用户的集合可用基本类型定义：

$$[U]$$

系统中有 n 个连续的编了号的存储块。而系统中的所有空闲存储块的集合以 B 来表示。显然这两个变量将在整个规格说明中起作用，我们希望它们成为全程变量。对它们进行声明是很容易的，可以有

$$n:\mathbb{N}$$

$$B:\mathbb{P}\,\mathbb{N}$$

并且可给一个限制：

$$B=1..n$$

但它们应如何写在规格说明中？**Z** 为此类问题提供了一个结构，叫公理定义（Axiomatic Definition）。公理定义引入了一个或多个全程变量和关于它们的谓词，这些谓词给出了进一步的信息。这些变量不能是前面已经定义的变量。它们的作用域为它们的声明出现处至整个规格说明的结束处。而谓词也具有全程的性质。

一个公理定义具有如图 3-1 所示的形式。声明部分和谓词部分都可以不出现。上述例子可写成如图 3-2 所示的公理定义。