形 式 化 方 法:

基于 B 方法的严格软件开发

(3) 集合论与逻辑

裘宗燕

北京大学数学学院信息科学系

2010年春季

概要

B 方法的基础是集合论和逻辑，集合和逻辑有严格的形式和精确的意义

用 B 语言写规范，需要了解 B 语言里集合和逻辑公式的写法和意义

本节介绍这方面的情况，并通过一些实例，说明如何用集合和逻辑的方式表述软件系统的问题

本部分的主要内容包括：

- 集合记法（第二章）
- 谓词逻辑（第一章，部分）
- 序对，关系和函数（第二章）

注意：B 语言有一种数学表示形式，还有一种工具使用的正文表示形式（不同的 B 工具采用的形式差不多）。下面都有说明

集合 (1)

一个集合是某种事物的一些实例的一个汇集

例如：自然数的集合，1 到 100 之间的自然数集合，选本课程的学生的集合，本教室里的椅子的集合，等等

要说一个集合，应该给它取一个名字，例如 S

对集合，最经常要考虑的一个问题就是某事物是否在某集合里

- $e \in S$ 表示 e 是 S 的成员　　`e : S`
- $e \notin S$ 表示 S 里没有 e　　`e /: S`

集合的成员称为集合的元素

对于元素个数很少的集合，可以用列举元素的方式说明。例如

$$SQUARES = \{1, 4, 9, 16, 25\}$$

$$Category = \{vip, normal, dobiuos\}$$

正文写法与此类似。在 B 里，上面第二种方式定义的集合称为枚举集合

理论上称列举元素的集合表示为集合的外延表示

集合 (2)：元素和内涵表示

集合完全由其元素确定，枚举元素的顺序并不重要

空集是没有元素的集合，用 $\varnothing$ 表示　　`{}`

如果要描述有很多元素的集合，可以采用集合的内涵（comprehension）表示方式，用一个谓词描述集合元素满足的条件。如

$$SQUARES = \{x \mid n \in \mathsf{NAT}_1 \wedge n \leq 5 \wedge x = n \times n\}$$

$$SQUARES = \{n \times n \mid n \in \mathsf{NAT}_1 \wedge n \leq 5\}$$

正文形式

```
SQUARES = {x | n : NAT1 & n <= 5 & x = n * n}
```

内涵表示的一般形式是：

$$S = \{e \mid P\}$$

其中 P 是一个谓词，表示选取满足条件的东西，而 e 是个表达式，说明如何基于谓词选取的东西来构造集合元素

集合 (3)：几个基本集合运算

基本集合运算是：

$S \cup T$	并集	`S \/ T`
$S \cap T$	交集	`S /\ T`
$S - T$	差集	`S - T`

对于集合的集合可以做广义交集和广义并集：

$$\bigcap SS \mathrel{\widehat{=}} \{e \mid \text{for all } s \in SS \cdot e \in s\} \qquad \texttt{inter SS}$$

$$\bigcup SS \mathrel{\widehat{=}} \{e \mid \text{for some } s \in SS \cdot e \in s\} \qquad \texttt{union SS}$$

设 VIP 是所有重点客户的集合，Normal 是所有普通客户的集合，Dubious 是所有可疑客户的集合，{VIP, Normal, Dubious} 就是集合的集合。
$\bigcup \{\mathrm{VIP}, \mathrm{Normal}, \mathrm{Dubious}\}$ 就是所有客户的集合

假设 D_i 是本月 i 日刷卡坐公交的市民集合，DS 是本月各日刷卡乘公交的市民集合的集合（是集合的集合），$\bigcap DS$ 就是每天坐公交的市民，$\bigcup DS$ 是本月曾经坐过公交的市民

集合 (4)：幂集，笛卡儿积

一个集合的幂集是它的所有子集的集合

$\mathbb{P}\, S \mathrel{\widehat{=}} \{T \mid T \subseteq S\}$	幂集，所有子集的集合	`POW S`
$\mathbb{P}_1\, S \mathrel{\widehat{=}} \{T \mid T \subseteq S \wedge T \neq \varnothing\}$	所有非空子集的集合	`POW1 S`

对任何集合 S 都有

$$\varnothing \in \mathbb{P}\, S$$
$$S \in \mathbb{P}\, S$$

两个集合 S 和 T 的笛卡儿积是二元有序对 (s, t) 的集合

$$S \times T \mathrel{\widehat{=}} \{(s, t) \mid s \in S \wedge t \in T\} \qquad \text{笛卡儿积} \qquad \texttt{S * T}$$

集合 (5)：基本集合

B 的基本集合包括：

数学表示	正文表示	
$\mathbb{Z}$	INTEGER	整数集合，看作抽象集合
$\mathbb{N}$	NATURAL	自然数集合
INT	INT	系统可实现的整数集合
NAT	NAT	系统可实现的自然数集合
$\mathbf{NAT}_1$	NAT1	**NAT** 的正子集
BOOL	BOOL	布尔类型，只包括**TRUE, FALSE**

实际的整数只能是 **INT**的元素，有两个预定义整数 **MININT**和 **MAXINT**， 是 **INT**集合里的最小和最大整数。

还有几个只能用于实现的类型，后面再讨论

集合 (6)：整数区间，集合包含

整数的区间是一类特殊集合，有专门表示形式：

$$m..n \triangleq \{i \mid i \in \mathbb{Z} \wedge m \leq i \wedge i \leq n\}$$

例子已经见过多次

几个基本集合的定义是：

$$\begin{array}{l} \mathtt{NAT} = 0..\mathtt{MAXINT} \\ \mathtt{NAT1} = \mathtt{NAT} - \{0\} \\ \mathtt{INT} = \mathtt{MININT}..\mathtt{MAXINT} \\ \mathtt{BOOL} = \{\mathtt{FALSE}, \mathtt{TRUE}\} \end{array}$$

集合 (7)

集合的元素个数称为其序数，表示为 card(S)。例如

$$\begin{aligned} \mathrm{card}(\{2,4,6\}) &= 3 \\ \mathrm{card}(\mathsf{NAT}) &= \mathsf{MAXINT} \end{aligned}$$

集合之间的关系有

$S = T$	S 等于 T	`S = T`
$S \neq T$	S 不等于 T	`S /= T`
$S \subseteq T$	S 是 T的子集	`S <: T`
$S \subset T$	S 是 T的真子集	`S <<: T`
$S \nsubseteq T$	S 不是 T的子集	`S /<: T`
$S \not\subset T$	S 不是 T的真子集	`S /<<: T`

显然，对于任何 S，都有：

$$\begin{aligned} S &\subseteq S \\ \varnothing &\subseteq S \end{aligned}$$

谓词逻辑

集合关系是阐述了一个事实，集合元素关系 $e \in S$ 也阐述了一个事实。此外我们也可以写出参数基本数据层面的事实的表达式

这些对基本事实的表述都是简单断言（assertion，或称谓词，predicate）

为了描述软件系统中的事实，就需要写各种各样的谓词， 不但需要描述一些简单的事实，还可能需要描述非常复杂的事实。为此就需要谓词逻辑表示

为描述软件的性质，B 语言也提供了谓词逻辑的表达形式

基本谓词包括永真的谓词和永假的谓词，表示为：

TRUE	永真	`TRUE`
FALSE	永假	`FALSE`

设 P 和 Q 是谓词，下面表达式也是谓词（真值关系大家都清楚）

$\neg P$	否定	`not P`
$P \wedge Q$	合取	`P & Q`
$P \vee Q$	析取	`P or Q`
$P \Rightarrow Q$	蕴涵	`P => Q`
$P \Leftrightarrow Q$	等价	`P <=> Q`

谓词演算

有些关系属于逻辑之外，例如对于任何集合 S 和 T，都有

$$T \subseteq Q \;\Leftrightarrow\; T \in \mathbb{P}\, S$$

不同的逻辑连接词有一些相互关系。例如

$$\begin{array}{rcl} P \Rightarrow Q & \text{等价于} & \neg P \vee Q \\ P \wedge Q & \text{等价于} & \neg(\neg P \vee \neg Q) \\ P \vee Q & \text{等价于} & \neg(\neg P \wedge \neg Q) \\ P \Leftrightarrow Q & \text{等价于} & (P \Rightarrow Q) \wedge (Q \Rightarrow P) \end{array}$$

说"等价于"的意思是说，对于任意的谓词 P 和 Q，两边基于 P 和 Q 写出的谓词（公式）的真假值总是相同

如果两个谓词 P 和 Q 等价，我们将这一事实记为 $P = Q$。 在用谓词做推理时，经常需要用到这些逻辑等价关系

注意：$P = Q$ 不是逻辑公式，而是为了方便说逻辑公式之间的关系引进的一种写法

谓词逻辑

在谓词逻辑（和命题逻辑）里可以证明大量的等价公式，我们可以用这些公式做谓词的等价变换， 这种操作称为谓词演算。例如（还有大量）

$$\begin{array}{rcll} P \wedge P & = & P & \wedge\ \text{幂等律} \\ P \vee P & = & P & \vee\ \text{幂等律} \\ P \wedge \neg P & = & \mathsf{FALSE} & \text{矛盾律} \\ P \vee \neg P & = & \mathsf{TRUE} & \text{排中律} \\ (P \wedge Q) \wedge R & = & P \wedge (Q \wedge R) & \wedge\ \text{结合律} \\ (P \vee Q) \vee R & = & P \vee (Q \vee R) & \vee\ \text{结合律} \\ P \wedge Q & = & Q \wedge P & \wedge\ \text{交换律} \\ P \vee Q & = & Q \vee P & \vee\ \text{交换律} \\ P \wedge \mathsf{TRUE} & = & P & \wedge\ \text{单位元} \\ p \vee \mathsf{FALSE} & = & P & \vee\ \text{单位元} \\ P \vee \mathsf{TRUE} & = & \mathsf{TRUE} & \wedge\ \text{零元} \\ p \wedge \mathsf{FALSE} & = & \mathsf{FALSE} & \vee\ \text{零元} \\ P \wedge (Q \vee R) & = & (P \vee Q) \wedge (P \vee R) & \vee\ \text{对}\ \ \wedge\ \ \text{分配律} \\ P \vee (Q \wedge R) & = & (P \wedge Q) \vee (P \wedge R) & \wedge\ \text{对}\ \ \vee\ \ \text{分配律} \end{array}$$

谓词逻辑：量词

谓词逻辑和 B 规范里还可以写全称和存在量词，形式是：

$\forall x.P$	全称	`! x . P`
$\exists x.P$	存在	`# x . P`

写软件规范时，实际写出的全称和存在量化公式都是下面形式：

$$\forall x.(x \in T \Rightarrow P)$$

$$\exists x.(x \in T \wedge P)$$

这两个公式说的是：对于属于类型 T 的每个 x 都有 P，或者存在属于类型 T 的某个 x 有 P

例如：

$$\forall x.(x \in \mathbb{N} \Rightarrow x + 1 > 0)$$

$$\exists x.(x \in \mathbb{N} \Rightarrow x * x \leq x)$$

$$\forall (x, y).(x \in \mathbb{N} \wedge y \in \mathbb{N}_1 \Rightarrow x * y \geq x)$$

这里 $\forall(x, y).(\ldots)$ 是 $\forall x.\forall y.(\ldots)$ 的简写

谓词逻辑：量词

常用量词公式形式有如下对偶关系：

$$\neg\forall x.(x \in T \Rightarrow P) \quad = \quad \exists x.(x \in T \wedge \neg P)$$

$$\neg\exists x.(x \in T \wedge P) \quad = \quad \forall x.(x \in T \Rightarrow \neg P)$$

直观地看：

- 不是所有 T 类型的 x 都使 P 成立，iff 存在 T 类型的 x 使 P 不成立
- 不存在 T 类型的 x 使 P 成立，iff 对每一个 T 类型的 x，P 都不成立

一个量化公式有三个成分：一个量词，一个变量（称为量化变量）， 圆点后面是一个谓词，是这个量词的作用范围（作用域，scope，也称辖域）

量词只对它作用域（是一个谓词公式）里的量化变量的起作用， 说的是它们的可能取值导致的整个量化谓词的真值情况

要理解量词的意义，以及对于谓词公式的代换操作， 需要定义谓词中变量的自由出现和约束出现

谓词逻辑：量词

用 freev P 表示谓词 P 里所有自由变量的集合，它可根据 P 的结构定义：

- 基本谓词里的变量都是自由变量（有意义的名字不是变量，如 NAT）
- 对各种逻辑连接词，其自由变量集合就是其成分谓词的变量集合的并集，例如 $\text{freev}(P \wedge Q) = \text{freev}\, P \cup \text{freev}\, Q$，$\text{freev}\, \neg P = \text{freev}\, P$
- 对于量化公式，有：

$$\begin{aligned} \text{freev}\, \forall x.P &= \text{freev}\, P - \{x\} \\ \text{freev}\, \exists x.P &= \text{freev}\, P - \{x\} \end{aligned}$$

如果变量 x 出现在公式 P 里，但 x 又不是 P 里的自由变量，那么它就是 P 里的一个约束变量

我们还可以说变量 x 没有在一个表达式或谓词里自由出现（也就是说，x 或者根本没出现，或者只有约束出现）。在 B 方法书中，用记号 $x \setminus E$ 或 $x \setminus P$ 表示变量 x 在表达式 E 或谓词 P 里非自由（非自由出现）

按逻辑公式的定义，同一个变量名完全可以出现在谓词 P 里不同的量词下面，但这种做法很容易造成理解错误。 就像我们在编程中也不赞成在嵌套的作用域里使用同样的名字一样

谓词逻辑：量词

在实践中，上述情况完全可以避免

例如 P 和 Q 里都有量化变量 x，但无论在 P 里还是 Q 里，x 都只出现在一个量词下面。现在要构造公式 $P \wedge Q$

- 这时选一个没出现在 P 或 Q 里的新变量，设 y 是这样的变量
- 设把 Q 中 x 的所有出现都换成 y（包括作为量化变量）得到的公式是 Q'
- 公式 $P \wedge Q'$ 与直接得到的 $P \wedge Q$ 等价，而且其中没有任何一个变量出现在两个量词下面

这说明，在构造公式的过程中完全可以避免量化变量重名的情况

这样的操作称为量化变量的重命名（renaming）

如果有同一变量出现在两个量词下面，可以通过适当的重命名消除。例如

$$\forall (n, m).(n \in \mathbb{N} \wedge m \in \mathbb{N} \wedge m = n * n \Rightarrow \exists n.(n \in \mathbb{N} \wedge n > m))$$

可以等价地改写为

$$\forall (n, m).(n \in \mathbb{N} \wedge m \in \mathbb{N} \wedge m = n * n \Rightarrow \exists k.(k \in \mathbb{N} \wedge k > m))$$

谓词逻辑：量词

还可能有一个变量既作为某个量词的量化变量，又出现在该量词的辖域之外的情况， 这种情况很容易造成误解，也可以通过重命名消除

例如

$$\forall x.(x \in \mathbb{N}_1 \Rightarrow (\exists y.(y \in \mathbb{N} \wedge y < x)) \wedge y = x * x)$$

这里最后的子公式 $y = x * x$ 里的 y 与前面那个量化变量 y 毫不相关，它是本公式的一个自由变量。这种写法很容易引起误解

通过重命名局部量词的量化变量，得到的公式意义不变，但更清晰了：

$$\forall x.(x \in \mathbb{N}_1 \Rightarrow (\exists v.(v \in \mathbb{N} \wedge v < x)) \wedge y = x * x)$$

B 谓词公式

前面讨论了 B 语言中谓词公式的构造方法，现在总结一下其基本形式

B 规范里的谓词公式包括：

- 等于表达式，用 $=, \neq$ 构造，在任何类型里都能用
- 整数类型的比较表达式，用 $>, <, \geq, \leq$ 构造
- 元素关系表达式，用 $\in, \notin$ 构造
- 集合包含表达式，用 $\subset, \subseteq, \not\subset, \not\subseteq$
- 谓词组合式和量化式

还有一种基于“代换”的形式，下面讨论

集合、谓词和类型

使用集合论的优点很明显，重要的有理论意义

- 用一阶逻辑可以描述各种复杂对象。如果只用逻辑，一阶逻辑就不够用
- 很多情况的表达可以不用量词（量词的自动证明困难，代价高）
- 在集合论里比较容易处理否定

从实践的角度看， 就是比较容易描述许多复杂的结构

朴素集合论有著名的罗素悖论，其根源是集合的内涵表示，如：

$$\{x \mid x \in x\}$$

$$\{x \mid x \notin x\}$$

$$\exists x.(x \in x)$$

如果 B 方法里的描述也有本质性的悖论，我们就无法保证用 B 方法开发并严格证明的软件系统内部没有问题

B 方法解决这一问题的基础是类型和类型检查： 一个合乎语法的谓词（合式公式）需要先经过类型检查。只有通过类型检查的 （良类型的）谓词才能作为证明的候选

集合、谓词和类型

B 语言里的每个表达式有一个类型，其定义基于集合的包含关系

- 集合的包含关系有一个上界
- 如果某个表达式的值属于一个集合，其类型就是这个集合的最大超集

例如，我们知道有

$$1..2 \subseteq 1..10 \subseteq 0..20 \subseteq \mathsf{NAT} \subseteq \mathbb{Z}$$

一个取值为 1 或 2 表达式的类型就是 $\mathbb{Z}$

显然，**TRUE** 和 **FALSE** 的类型是 **BOOL**

集合也有类型，其类型是它的元素的类型的幂集。$1..2$ 的类型是 $\mathbb{P}\,\mathbb{Z}$

朴素地说，在证明 $x \in y$ 之前，需要先检查 x 和 y 的类型合适，能保证 $x \in y$ 是类型良好的谓词表述，而这实际上要求 y 的类型是 x 的类型的幂集，否则就不用考虑元素关系是否成立

集合、谓词和类型

回想一下 B 语言中的集合的来源，有如下几类：

- 系统预定义的几个基本集合，其中只有两个类型，**BOOL** 和 $\mathbb{Z}$
- 枚举集合。定义出的枚举集合与其他集合无关，每个枚举定义了一个类型
- 待定集合。无法确定它与其他集合有关，看作一个类型

类型构造，只有两种语法结构（假设 T_1 和 T_2 是类型）：

$\mathbb{P}\, T_1$	幂集，是 T 的子集的类型
$T_1 \times T_2$	有序对 (x_1, x_2) 的类型，其中 $x_1 \in T_1$ 且 $x_2 \in T_2$

抽象机描述的许多地方需要类型断言，用元素关系描述。包括：

- 在不变式子句 **INVARIANT** 里说明变量的类型
- 在性质子句 **PROPERTIES** 里说明常量的类型
- 在操作的定义里说明参数的类型（一般用 **IF** 或者 **PRE** 结构）

集合、谓词和类型

类型的例子。假设北大学生的学号用自然数表示，那么一个变量如果：

- 取值为学号，其类型是 $\mathbb{Z}$
- 取值为某班级学生的学号集合，其类型为 $\mathbb{P}\,\mathbb{Z}$
- 取值为一个集合，其中包含三个学号集合作为元素， 这三个集合分别包含北大学生中人数最多的三个姓的所有学生学号，其类型是 $\mathbb{P}\,\mathbb{P}\,\mathbb{Z}$
- 取值为表示一个学生是否取得学位的二元组 (n, ok)，其中 ok 的类型是 **BOOL**，这个变量的类型是 $\mathbb{Z} \times$ **BOOL**
- 取值为一个班的学生是否取得学位的二元组集合，其类型为 $\mathbb{P}(\mathbb{Z} \times \textbf{BOOL})$

在规范中定义变量和表达式时，都应该考虑清楚它的类型

在写谓词时（不变式/性质/条件等），也应关注谓词是否类型正确，如

- 等于或不等于判断两边表达式的类型是否一样
- 集合成员关系右边表达式的类型是否左边表达式类型的幂集
- 各种子集关系左右表达式的类型是否一样，是否都为某个类型的幂集？

有序对

一个有序对也一个表达式，其写法有两种：

$$e \mapsto e'$$

$$e, e'$$

许多地方可能用到有序对，例如

$\forall(x, y).P$	作为一种简写形式
$A_1, B_1 = A_2, B_2$	要求分别相等
$ok, seats := \textsf{TRUE}, seats - 1$	多重代换（“赋值”）

如果 e_1 的类型是 T_1， e_2 的类型是 T_2，那么 e_1, e_2 的类型是 $T_1 \times T_2$

其中的多重代换在 B 规范中使用的非常普遍

构造有序对的 , 和 $\mapsto$ 都是左结合的运算符，也就是说，e_1, e_2, e_3 表示 $(e_1, e_2), e_3$，它是一个有序对，其第一个元本身又是一个有序对

我们还经常需要讨论有序对的集合集合，下面是这方面的问题

二元关系

一个二元关系就是就是有序对的一个集合，但要求集合中所有有序对的第一个元同属一个类型，所有第二个元同属一个类型

已有集合 S_1 和 S_2 上的二元关系集合的表示形式是

$$S_1 \leftrightarrow S_2 \quad \text{二元关系} \quad \texttt{S1 <-> S2}$$

其定义是：

$$S_1 \leftrightarrow S_2 \;\widehat{=}\; \mathbb{P}(S_1 \times S_2)$$

实际上，规范中常用的是二元关系集合的元素，即具体的二元关系实例

例如，某学期学生学号和相应同学所选的课程，形成了一个具体的二元关系 $courseSel$，如果学生 n 选了课程 c_1 和 c_2，那么就一定有

$$(n, c_1) \in courseSel \qquad (n, c_2) \in courseSel$$

由于软件中经常需要描述各种对象之间的关系，所以在规范中， 二元关系以及它的一些受限的子集合使用非常普遍。有关重要子集下面介绍

二元关系：关系操作(1)

假设 $p \in S_1 \leftrightarrow S_2$, $q \in S_2 \leftrightarrow S_3$, $s \subseteq S_1$, $t \subseteq S_2$，有如下定义：

数学形式	正文形式	定义
p^{-1}	`~p`	$\{b, a \mid b \in S_2 \wedge a \in S_1 \wedge (a, b) \in p\}$
$\mathrm{dom}(p)$	`dom(p)`	$\{a \mid a \in S_1 \wedge \exists b.(b \in S_2 \wedge (a, b) \in p)\}$
$\mathrm{ran}(p)$	`ran(p)`	$\mathrm{dom}(p^{-1})$
$\mathrm{id}(S_1)$	`id(S_1)`	$\{a, b \mid (a, b) \in S_1 \times S_1 \wedge a = b\}$
$p \,;\, q$	`p;q`	$\{a, c \mid (a, c) \in (S_1 \times S_3) \wedge \exists b.(b \in S_2 \wedge (a, b) \in p \wedge (b, c) \in q)\}$
$s \lhd p$	`s <| p`	$\{a, b \mid (a, b) \in p \wedge a \in s\}$
$p \rhd t$	`p |> t`	$\{a, b \mid (a, b) \in p \wedge b \in t\}$
$s ⩤ p$	`s <<| p`	$\{a, b \mid (a, b) \in p \wedge a \notin s\}$
$p ⩥ t$	`p |>> t`	$\{a, b \mid (a, b) \in p \wedge b \notin t\}$

显然有下面等价关系：

$$\begin{aligned} s ⩤ p &= (\mathrm{dom}(p) - s) \lhd p \\ p ⩥ t &= p \rhd (\mathrm{ran}(p) - t) \end{aligned}$$

关系操作的解释

上面各种关系操作的解释

- 关系 p^{-1} 是 p 的逆关系
- 集合 $\mathrm{dom}(p)$ 称为 p 的定义域
- 集合 $\mathrm{ran}(p)$ 称为 p 的值域
- 关系 $\mathrm{id}(S_1)$ 称为集合 S_1 上的恒等关系
- $p \,;\, q$ 是 p 与 q 复合而成的复合关系
- $s \lhd p$ 是 p 被集合 s 做定义域限制而成的关系
- $p \rhd t$ 是 p 被集合 t 做值域限制而成的关系
- $s ⩤ p$ 是 p 被集合 s 做定义域减而成的关系
- $p ⩥ t$ 是 p 被集合 t 做值域减而成的关系

二元关系：操作应用实例

设有学生学号集合 $Student$，课程集合 $Course$ 和教学楼编号集合 $Building$

本学期学生选课情况是二元关系 $courseSel \in Student \leftrightarrow Course$，课程所安排的教学楼是二元关系 $coursePos \in Course \leftrightarrow Building$

设 $math \in Student$ 是数学学院学生的学号集合，$general \in Course$ 是本学期全校的所有选修课的集合。我们有

$math \lhd courseSel$	数学学院学生的选课情况
$math ⩤ courseSel$	非数学学院学生的选课情况
$courseSel ⩥ general$	学生选选修课的情况
$courseSel ⩥ genaral$	学生选非选修课的情况
$courseSel^{-1}$	各课程的选课学生情况
$courseSel \mathbin{;} coursePos$	学生与上课教学楼的关系
$math - \mathrm{dom}(courseSel)$	没选课的数学学院学生
$\mathrm{dom}((courseSel \mathbin{;} coursePos) \rhd \{2\})$	在二教有课的学生
$\mathrm{dom}(math \lhd (courseSel \mathbin{;} coursePos) \rhd \{2\})$	在二教有课的数学学院学生

关系操作(2)

假设 $p \in s_1 \leftrightarrow s_2$, $r \in s_1 \leftrightarrow s_2$, $w \subseteq s_1$,
$t \in s_2$，$q \in s_1 \leftrightarrow s_3$，$h \in s_3 \leftrightarrow s_4$。下面是另外几个关系操作

数学形式	正文形式	定义
$p[w]$	`p[w]`	$\mathrm{ran}(w \lhd p)$
$p \Lleftarrow r$	`p <+ r`	$(\mathrm{dom}\, r ⩤ p) \cup r$
$p \otimes q$	`p >< q`	$\{a, (b, c) \mid (a, (b, c)) \in s_1 \times (s_2 \times s_3) \wedge (a, b) \in p \wedge (a, c) \in q\}$
$\mathrm{prj}_1(s_1, s_2)$	`prj1(s1,s2)`	$\{a, b, c \mid (a, b, c) \in s_1 \times s_2 \times s_1 \wedge a = c\}$
$\mathrm{prj}_2(s_1, s_2)$	`prj2(s1,s2)`	$\{a, b, c \mid (a, b, c) \in s_1 \times s_2 \times s_1 \wedge b = c\}$
$p \parallel h$	`p || h`	$\{(a, b), (c, d) \mid ((a, b), (c, d)) \in (s_1 \times s_2) \times (s_3 \times s_4) \wedge (a, c) \in p \wedge (c, d) \in h\}$

上述操作构造出集合或关系。$p[w]$ 称为集合 w 在关系 p 下的像；$p \Lleftarrow r$ 是 r 对 p 覆盖形成的关系，$p \otimes q$ 是 p 与 q 的直积，$p \parallel q$ 是 p 与 q 的平行积，另外两个是投影关系

二元关系操作的实例

$$
\begin{array}{lcl}
p & = & \{3 \mapsto 5, 3 \mapsto 9, 6 \mapsto 3, 9 \mapsto 2\} \\
w & = & \{1, 2, 3\} \\
p[w] & = & \{5, 9\} \\
q & = & \{2 \mapsto 7, 3 \mapsto 4, 5 \mapsto 1, 9 \mapsto 5\} \\
q \Lleftarrow p & = & \{3 \mapsto 5, 3 \mapsto 9, 6 \mapsto 3, 9 \mapsto 2, 2 \mapsto 7, 5 \mapsto 1\} \\
f & = & \{8 \mapsto 10, 7 \mapsto 11, 2 \mapsto 11, 6 \mapsto 12\} \\
g & = & \{1 \mapsto 20, 7 \mapsto 20, 2 \mapsto 21, 1 \mapsto 22\} \\
f \otimes g & = & \{7 \mapsto (11, 20), 2 \mapsto (11, 21)\} \\
s & = & \{1, 4\} \\
t & = & \{2, 3\} \\
\mathrm{prj}_1(s, t) & = & \{(1, 2) \mapsto 1, (1, 3) \mapsto 1, (4, 2) \mapsto 4, (4, 3) \mapsto 4\} \\
\mathrm{prj}_2(s, t) & = & \{(1, 2) \mapsto 2, (1, 3) \mapsto 3, (4, 2) \mapsto 2, (4, 3) \mapsto 3\} \\
h & = & \{1 \mapsto 11, 4 \mapsto 12\} \\
k & = & \{2 \mapsto 21, 7 \mapsto 22\} \\
h \parallel k & = & \{(1, 2) \mapsto (11, 21), (1, 7) \mapsto (11, 22), \\
 & & \quad (4, 2) \mapsto (12, 21), (4, 7) \mapsto (12, 22)\}
\end{array}
$$

二元关系的类型检查

定义（描述）二元关系的记法也是表达式。要保证二元关系表达式有意义，需要表达式的组成满足类型的求。《B Book》第 2.4 节给出了二元关系类型检查的定义，其实基本情况不难理解，这里用几个例子帮助建立直观认识：

- 对于 $p\,;q$，要求 $\mathrm{ran}(p)$ 与 $\mathrm{dom}(q)$ 类型相同。我们可以说是要求它们的元素的类型相同，但明显，对于 $e_1 \in s_1$ 和 $e_2 \in s_2$，如果 e_1 和 e_2 类型相同，当且仅当 s_1 和 s_2 类型相同
- $p \Lleftarrow q$ 要求两个关系的定义域和值域的类型分别相同
- $p[w]$ 要求 $\mathrm{dom}(p)$ 和 w 类型相同
- $f \otimes g$ 要求 $\mathrm{dom}(f)$ 和 $\mathrm{dom}(g)$ 的类型相同
- $s \lhd p$ 要求 s 和 $\mathrm{dom}(p)$ 的类型相同
- $p \rhd t$ 要求 t 和 $\mathrm{ran}(p)$ 的类型相同

其他情况的要求与此类似

根据操作对象的类型，很容易确定得到的结果集合或关系的类型

函数

函数是关系的特殊情况，一个函数也是一个关系， 但它对其定义域里的任何一个点，值域里只有最多一个对应的值。例如前面定义关系

$$p \quad = \quad \{3 \mapsto 5, 3 \mapsto 9, 6 \mapsto 3, 9 \mapsto 2\}$$

不是函数，因为对于 p 的定义域里的 3，值域有两个对应的值。而

$$f \quad = \quad \{2 \mapsto 5, 3 \mapsto 9, 6 \mapsto 3, 9 \mapsto 2\}$$

就是一个函数

一个关系从其定义域类型的子集确定值域的一个子集（像集）。例如

$$p[\{2,3,6\}] = \{5,9,3\}$$

函数对其定义域的每个单点集合，像集也一定是单点集合：

$$f[\{3\}] = \{9\}$$

一般关系没有这种性质。把这个值域单点集的元素看作函数对定义域相应元素的值，记为

数学形式	正文形式	解释
$f(a) = b$	`f(a)`	函数的值，当 $f[\{a\}] = \{b\}$

不同类的函数

下面定义若干类函数，包括部分函数和全函数，部分内射和全内射，部分满射和全满射，以及全双射

数学形式	正文形式	定义	名称
$s \pfun t$	`s +-> t`	$\{r \mid s \leftrightarrow t \wedge (r^{-1}; r) \in \mathrm{id}(t)\}$	部分函数
$s \longrightarrow t$	`s --> t`	$\{f \mid f \in s \pfun t \wedge \mathrm{dom}(f) = s\}$	全函数
$s \pinj t$	`s >+> t`	$\{f \mid f \in s \pfun t \wedge f^{-1} \in t \pfun s\}$	部分内射
$s \rightarrowtail t$	`s >-> t`	$s \pinj t \cap s \longrightarrow t$	全内射
$s \psurj t$	`s +->> t`	$\{f \mid f \in s \pfun t \wedge \mathrm{ran}(f) = t\}$	部分满射
$s \twoheadrightarrow t$	`s -->> t`	$s \psurj t \cap s \longrightarrow t$	全满射
$s \bij t$	`s >->> t`	$s \rightarrowtail t \cap s \twoheadrightarrow t$	全双射

对部分函数的定义的解释：考虑任意 $a \in t$，由于 $(r^{-1}; r) \in \mathrm{id}(t)$，对任何 $b \in r^{-1}[\{a\}]$，像集 $r[\{b\}]$ 只能是单点集合 $\{a\}$，所以 r 是函数

全函数是定义域为 s 的部分函数；内射把两个不同点映射到不同值，所以其逆也是函数；全内射是定义域等于 s 的内射；满射的值域等于 t；全满射是定义域为 s 的满射；全双射既是全内射又是全满射，又称“一一对应”

函数抽象

我们已经有一些方法来描述具体函数的函数了

- 对于很小的有限函数，可以通过直接写序对集合的方式描述，例如

$$discount = \{vip \mapsto \mathbf{90}, normal \mapsto \mathbf{100}, dubious \mapsto \mathbf{100}\}$$

- 可以用集合内涵表示生成序对的集合，作为函数描述，如

$$func = \{\boldsymbol{x} \mapsto \boldsymbol{y} \mid \boldsymbol{x} \in \mathbf{1..10} \wedge \boldsymbol{y} \in \mathbb{Z} \wedge \boldsymbol{y} = \boldsymbol{x} * \boldsymbol{x} + \mathbf{1}\}$$

- B 语言还从计算理论引进了函数的 $\boldsymbol{\lambda}$ 表达方式

数学形式	正文形式
$\boldsymbol{\lambda x.(x \in s \mid e)}$	`% x . (x IN s | e)` 定义： $\{\boldsymbol{x}, \boldsymbol{y} \mid (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{s} \times \boldsymbol{t} \wedge \boldsymbol{y} = \boldsymbol{e}\}$
$\boldsymbol{\lambda x.(x \in s \wedge P \mid e)}$	`% x . (x IN s & P | e)` 定义： $\{\boldsymbol{x}, \boldsymbol{y} \mid (\boldsymbol{x}, \boldsymbol{y}) \in \boldsymbol{s} \times \boldsymbol{t} \wedge \boldsymbol{P} \wedge \boldsymbol{y} = \boldsymbol{e}\}$

要求：$\boldsymbol{s}, \boldsymbol{t}$ 是集合，$\boldsymbol{P}$ 是谓词，$\boldsymbol{x}, \boldsymbol{y}$ 是不同变量，它们在 $\boldsymbol{s}, \boldsymbol{t}$ 里不自由出现，$\forall \boldsymbol{x}.(\boldsymbol{x} \in \boldsymbol{s} \Rightarrow \boldsymbol{e} \in \boldsymbol{t})$。这样的 $\boldsymbol{\lambda}$ 表达式表示了一个从 $\boldsymbol{s}$ 到 $\boldsymbol{t}$ 的函数

函数构造和使用实例

以前面客户管理系统为例，看一些函数类型描述和函数使用的例子

- *category* 给出客户分类，其类型写为：

$$category \in client \rightarrow Category$$

  这是一个全函数，意味着每个客户都有确定的分类

- *allowance* 表示客户允许的一次消费额度，是从当前客户集合 *cliant* 到某整数区间的全函数：

$$allowance \in client \rightarrow \mathbf{1..2000}$$

- 从函数 *category* 可以求出各类客户的集合

$$category^{-1}[\{\boldsymbol{normal}\}]$$
$$category^{-1}[\{\boldsymbol{vip}\}]$$
$$category^{-1}[\{\boldsymbol{dubious}\}]$$

- 根据系统的要求，系统的不变式应该包括（普通客户消费额度为1000）

$$allowance[category^{-1}[\{\boldsymbol{normal}\}]] = \{\mathbf{1000}\}$$

关系和函数实例：家庭关系

下面看《B Book》里的一个例子（77页）：用形式化方式描述社会里的家庭关系

假定这是一个具有非常严格的规则的社会，用一套法律规定了人与人的关系（当然，实际社会情况的情况复杂得多），有如下规定：

- 一个人或为男人，或为女人，必为两者之一
- 只有女人可以有丈夫，其丈夫必须是男人
- 一个女人只能有一个丈夫，反之亦然
- 母亲必须是已婚女人
- 还可以增加许多关系

下面考虑如何将这些"法律"形式化

家庭关系，几个基本集合

首先假定抽象集合（待定集合） $PERSON$，包含我们关注的所有可能的人

在定义几个变量表示关心的集合：$people$, men, $women$, $husband$, $mother$。它们具有如下的约束关系（类型等）

$$people \subseteq PERSON$$

$$men \subseteq people$$

$$women \subseteq PERSON \wedge women = people - men$$

$$husband \in women \rightarrowtail men$$

$$mother \in person \rightarrowtail \mathrm{dom}(husband)$$

这里将 $husband$ 和 $mother$ 定义为部分函数，因为

- 不是每个女人都有丈夫
- 有些人的母亲已经不在了

在写软件规范时，应该注意我们写出的每个表述所蕴涵的意义

把软件规范写正确、写准确，需要学习和不断实践

家庭关系：派生概念

基于集合基本集合，可以定义许多派生概念：

$wife$	$\widehat{=}$	$busband^{-1}$	妻子
$spouse$	$\widehat{=}$	$husband \cup wife$	配偶
$father$	$\widehat{=}$	$mother \mathbin{;} husband$	父亲
$parents$	$\widehat{=}$	$mother \cup father$	双亲
$children$	$\widehat{=}$	$(mother \cup father)^{-1}$	子女
$daughter$	$\widehat{=}$	$children \rhd women$	女儿
boy	$\widehat{=}$	$children \rhd men$	儿子
$sibling$	$\widehat{=}$	$(children^{-1} \mathbin{;} children) - \mathrm{id}(PERSON)$	兄弟姐妹
$brother$	$\widehat{=}$	$sibling \mathbin{\rhd\!\!\!-} women$	兄弟
$sibInLaw$	$\widehat{=}$	$(sibling \mathbin{;} spouse) \cup (spouse \mathbin{;} sibling) \cup$ $(spouse \mathbin{;} sibling \mathbin{;} spouse)$	同辈亲戚
$nephew\_niece$	$\widehat{=}$	$(sibling \cup sibInLaw) \mathbin{;} children$	下辈
$uncle\_aunt$	$\widehat{=}$	$nephew\_niece^{-1}$	上辈
$cousin$	$\widehat{=}$	$uncle\_aunt \mathbin{;} children$	

家庭关系：性质

基于上面定义的家庭关系，可以证明许多性质，例如

$$father \mathbin{;} father^{-1} \quad = \quad mother \mathbin{;} mother^{-1}$$

证明：

$$\begin{array}{lll}
& father \mathbin{;} father^{-1} & \\
= & (mother \mathbin{;} husband) \mathbin{;} (mother \mathbin{;} husband)^{-1} & \text{(定义)} \\
= & (mother \mathbin{;} husband) \mathbin{;} (husband^{-1} \mathbin{;} mother^{-1}) & \text{(逆的性质)} \\
= & mother \mathbin{;} (husband \mathbin{;} husband^{-1}) \mathbin{;} mother^{-1} & \text{(“;” 的结合律)} \\
= & mother \mathbin{;} \mathrm{id}(\mathrm{dom}(husband)) \mathbin{;} mother^{-1} & \text{(互逆函数)} \\
= & (mother \rhd \mathrm{dom}(husband)) \mathbin{;} mother^{-1} & \text{(“}\rhd\text{”的定义)} \\
= & mother \mathbin{;} mother^{-1} & \text{(值域限制，这里没有限制)}
\end{array}$$

《B Book》78页最下面有几个性质证明

请选两个做一下，作为本周练习的一部分

抽象机实例：打印机访问管理

考虑一个软件，它管理一个系统里的用户与打印机的关系

一些用户被允许使用一些打印机，这种允许性质可以用一个关系来表示。在下面讨论的规范抽象机里， 用一个关系类型的变量表示它

系统里应该记录现有的用户集合和打印机集合，假设对于打印机还有一组可能的使用选项（如彩色、双面等），也用一个待定集合表示

在上述考虑下，抽象机的开头可以定义为：

MACHINE $Access$
SETS $USER; PRINTER; OPTION; PERMISSION = \{ok, noaccess\}$
CONSTANTS $options$
PROPERTIES $options \in PRINTER \leftrightarrow OPTION$
$\mathrm{dom}(options) = PRINTER \wedge \mathrm{ran}(options) = OPTION$

这里基于简单考虑，直接用集合 $PRINTER$ 表示系统里的所有打印机。如果还想允许加入和删除打印机，那么可以用 $PRINTER$ 作为可能打印机的集合，另外用一个变量 $printer$ 表示系统里现有的打印机集合

集合 $USER$ 也可以同样考虑。如果实际打印机集合用变量表示，$options$ 也必须用变量表示（因其也会在系统运行中变化）

打印机访问控制

要维护的是用户与打印机之间的"允许访问"关系，用一个变量表示：

VARIABLES $access$
INVARIANT $access \in USER \leftrightarrow PRINTER$
INITIALISATION $access := \varnothing$

下面考虑抽象机操作，根据需要定义

首先是给一个用户访问某打印机的权力，或取消其权力：

OPERATIONS
$add(uu, pp) =$
PRE $uu \in USER \wedge pp \in PRINTER$
THEN $access := access \cup \{uu \mapsto pp\}$
END;

$erase(uu, pp) =$
PRE $uu \in USER \wedge pp \in PRINTER$
THEN $access := access - \{uu \mapsto pp\}$
END;

打印机访问控制

增加/取消权力都可以用简单的关系操作（集合操作）描述

下面考虑其他操作。首先是两个查询操作，第一个查询具体用户可否做某种方式的打印，返回一个许可情况报告

OPERATIONS

$ans \longleftarrow queryOption(uu, oo) =$
PRE $uu \in USER \wedge oo \in OPTION$
THEN IF $uu \mapsto oo \in (access \mathbin{;} options)$
THEN $ans := ok$
ELSE $ans := noaccess$
END
END;

我们可以考虑扩充这一查询，在允许访问具有所希望打印选项的打印机时，（非确定地）返回一台该用户可以访问的具有指定选项功能的打印机

$pri :\in \mathrm{ran}(\{uu\} \lhd access \rhd options^{-1}[\{oo\}])$

显然，只有得到的集合不同时这一选取才有意义

可以证明如果 $uu \mapsto oo \in (access \mathbin{;} options)$ 成立，这个集合不空

打印机访问控制

考虑另一查询：一个用户可以使用的打印机的数目

写这个操作需要集合计数，用操作 card 表示：

$nn \longleftarrow queryPNumber(uu) =$
PRE $uu \in USER$
THEN $nn := \mathrm{card}(\mathrm{ran}(\{uu\} \lhd access))$
END;

有权使用具有某种选项打印功能的用户个数：

$nn \longleftarrow queryOpUsers(oo) =$
PRE $oo \in OPTION$
THEN $nn := \mathrm{card}(\ \mathrm{dom}(\ (access \mathbin{;} options) \rhd \{oo\}\ )\ )$
END;

各种查询都可以用类似方式描述。例如允许使用打印选项 oo 的用户集合：

$uus := (access \mathbin{;} options)^{-1}[\{oo\}]$

打印机访问控制

现在考虑两个用户访问权限的合一，也就是说，让两个用户都可以用原本被他们二人之一使用的计算机。操作定义如下

$$
\begin{array}{l}
unify(uu1, uu2) = \\
\quad \textbf{PRE } uu1 \in USER \wedge uu2 \in USER \\
\quad \textbf{THEN } access := access \cup \{uu1\} \times access[\{uu2\}] \\
\qquad\qquad\qquad\qquad\qquad \cup \{uu2\} \times access[\{uu1\}] \\
\quad \textbf{END;}
\end{array}
$$

再考虑另一操作：完全禁止一个用户使用打印机：

$$
\begin{array}{l}
ban(uu) = \\
\quad \textbf{PRE } uu \in USER \\
\quad \textbf{THEN } access := \{uu\} \lhd\!\!\!- access \\
\quad \textbf{END;}
\end{array}
$$

我们还可以定义许多其他操作

有些同学熟悉"基于角色的访问控制系统"，其基础就是维护两个集合: 用户到角色的映射和角色到资源的映射。提供一些操作维护修改这些关系

总结

本节讨论了 B 方法的一些基础问题：集合表述法和逻辑， 特别是讨论了一大类重要的集合：关系以及它的一些重要子集（各种函数）

集合、关系和逻辑我们写软件规范的基本手段

描述不变式、集合内涵表示和函数的 $\boldsymbol{\lambda}$ 记法定义时，都要用到谓词逻辑，谓词还出现在规范的各种条件里，包括前条件、**IF** 的条件等等

集合和关系用于描述系统的状态和状态变换。前面以及看到许多实例， 今后我们还会写很多实例

要保证写出的规范合乎基本要求，需要注意规范中表达式的类型，例如：

- 只能比较两个类型相同的表达式
- 考虑元素关系时，右边表达式的类型应是左边表达式的类型的幂集
- 等等

还需注意各种表达形式里变量的使用，保证一些"无自由出现"条件