形 式 化 方 法:

基于 B 方法的严格软件开发

(5) 非确定性

裘宗燕

北京大学数学学院信息科学系

2010年春季

概述

本节首先讨论抽象机记法（AMN）中的非确定性。对写规范而言，非确定性扮演着非常重要的角色，因为非确定性使我们：

- 可以把一些灵活性留在规范里，为以后的精化和实现留下变动的余地
- 推迟做出某些设计决策的时间，直到我们对问题有了足够的认识

这一点和程序里的情况很不一样，顺序程序的行为都是确定性的

下面将

- 先介绍规范中非确定性的作用
- 介绍 AMN 提供的各种非确定性结构
- 介绍这些非确定性结构的应用

非确定性

前面介绍过的 AMN 代换（命令）都是确定性的（除了选择赋值），因为

- 无论从什么状态出发执行这些代换，操作都保证结束
- 代换描述的工作完成时，只有一个可能的最后状态（确定性）

一个代换究竟做什么，完全可以准确地预料

如果在一个操作的规范里用的都是这种代换，该操作实施后达到什么状态，产生什么输出，就完全由操作执行前的状态和操作的输入确定

回想一下使用 B 方法的初衷，是想用它写软件系统和部件的规范，希望能在某个抽象层次上表述系统或组件的期望行为，或者说，允许表现的行为

正因为这样，我们允许规范中使用各种抽象的数学描述，其实许多数学描述并没有与之直接对应的计算机实现

对于一个抽象规范，应该有多种不同的可能方式去实现它，所有这些实现都正确反映了规范所规定的系统行为

在这种情况下，写规范的人实际上需要考虑一个问题：自己写出的规范不应给满足 “实际需要” 的实现强加任何不必要的限制

非确定性

举一个实际工程的例子：假设我们要设计一部越野车，开始可能是做整车的结构设计。如果在做这样抽象层设计时就确定了所用轮胎的尺寸，并根据这一选择继续设计，就可能造成一些不良后果，例如：

- 不适当地限制了后面设计的选择空间（这一设计决策将影响后面工作）
- 后来可能发现这一早期选择并不是最优，为了将这类过早做出的不够慎重的选择，给整车的各方面设计留下许多遗憾
- 后来发现这一选择不对，推翻重来造成极大时间和金钱损失，等等

一个抽象机应该描述期望开发的软件模块的可接受行为，也就是说，为未来的软件模块划一个框（未必是一种特定行为）， 既描述了该模块应该做什么、不应该做什么，也给后续开发和实现留下足够多灵活变化的空间

由于这些原因，规范语言中的非确定性结构就非常重要了

要描述一个 “背单词” 软件，在规范中，选下一单词的操作可能写成

$$newwd :\in WROD - recognized$$

后续开发中实际使用的精细选词策略都是这一规范的 “正确” 实现

非确定性

在写一个操作的规范时，我们应该考虑它的可能行为，或允许的行为。这时就需要非确定性的语言结构了

在实际情况中，一个操作可能有数种都可以接受的行为。这种情况下，写抽象规范的人应该考虑这些变化和可能，写出非确定性的规范，让操作的实现者去（根据实际情况）考虑最后实现那种行为

用 $newwd :\in WROD - recognized$ 作为选单词操作的规范，不仅给后续设计者提供了采用任何具体选词策略的可能性（因为都满足这个规范），而且允许系统里的不同具体操作实例采用不同的选词策略（因为都满足规范）

显然，确定性的规格描述结构无法满足这种需要。因为确定性操作规范只允许一种行为、一种可能性

进一步说，仅有上面这种从一个集合里的非确定性 “选择赋值” 也还不够，因为实际需要描述的非确定性未必都是简单地从一个集合选值

B 语言里提供了一批非确定性的规范描述结构，基于它们可以描述操作的多种行为可能性，不同的可能操作进程

非确定性

一个抽象机的变量和不变式确定了一个 “可能状态空间”（一个集合）

- 一个确定性操作，是从这一状态空间到它自身的一个 “全函数”：任给一个状态，该操作将它映射到另一特定状态
- 一个非确定性操作，是从这一状态空间到它自身的一个 “关系”：任给一个状态，该操作将系统转换到若干个可能状态之一

一般而言，一个抽象机操作就是这样的一个关系

非确定性的规范，实际上是一种 “弱规范”

- 在英文里，specify 表示把某些事情说得清清楚楚，确确切切
- 而非确定性规范，是把各种可能的变化的选择说得清清楚楚，而不是描述唯一的一样事物（一种行为）

非确定性的规范，既是精确的，又是不精确的

- 其精确性在于确实明确说明了这一系统有什么样的可能行为
- 却没有描述清楚在这些可能性的范围里系统的特定具体行为

约束选择代换 CHOICE

在一些情况下，我们知道操作可以以两种或者多种不同方式进行，不同方式可能改变不同的变量，做不同的事情，以不同方式得到输出。这种情况可以用 “约束选择代换” **CHOICE** 结构描述

约束选择代换表示在两个（或多个）代换之间的任意选择。其描述方式借用了 Dijkstra 的卫式命令的形式。在 S 和 T 之间选择描述为

$$S \parallel T$$

如前所述，规范需要给实现者留下必要的灵活性。约束选择代换的用途也在这个方面。规范 $S \parallel T$ 的实现并不需要同时提供 S 和 T 的实现并提供某种运行时的随机选择机制。实际上，实现者在这一点可以根据情况自由选择实现 S 或者实现 T，以其作为规范 $S \parallel T$ 的实现

B 语言允许任意多个分支的选择结构

$$S_1 \parallel S_2 \parallel \ldots \parallel S_n$$

这里的 S_1, S_2, ..., S_n 都是代换。做 **CHOICE** 代换的意义是：在其所有分支代换中任意选择一个，执行该代换

约束选择代换

如果我们在写规范时看到了一些情况，其中多种行为都可以接受，就可以考虑用 **CHOICE** 结构，其中罗列出各种可接受的行为

约束选择的 B 语言语法形式如下：

语法	定义
CHOICE S_1 **OR** ... **OR** S_n **END**	$S_1 \parallel \ldots \parallel S_n$

显然，约束选择必须在实现中被替代掉，因为它本身不是可执行的

CHOICE 的分支可以是任意代换，也可以又是 **CHOICE** 结构

由于选择结构 $S \parallel T$ 的行为可以像 S 或者像 T，要保证做了代换 $S \parallel T$ 之后，后条件 Q 成立，实际上既要求 S 能保证得到 Q，也要求 T 能保证得到 Q。由此得到选择结构的定义：

$$[S \parallel T]Q \;\widehat{=}\; [S]Q \wedge [T]Q$$

一般而言，是

$$[S_1 \parallel S_2 \parallel \ldots \parallel S_n]Q \;\widehat{=}\; [S_1]Q \wedge [S_2]Q \wedge \ldots \wedge [S_n]Q$$

约束选择代换

设要描述一次商业交易，我们可能买 2 吨；也可能买 1 吨；最后一种可能是交易失败。这些可以描述为

CHOICE
$trade := success || balance := balance - price \times \mathbf{2}$
OR $trade := success || balance := balance - price$
OR $trade := fail$
END

假设我们要保证购买货品后的帐户余额不低于 M 元，有计算

$$\left[\begin{array}{l} trade := success || \\ balance := balance - price \times \mathbf{2} \\ [] \quad trade := success || \\ balance := balance - price \\ [] \quad trade := fail \end{array}\right] (balance \geq \boldsymbol{M})$$

$= balance - price \times \mathbf{2} \geq \boldsymbol{M} \wedge balance - price \geq \boldsymbol{M}$

$= balance - price \times \mathbf{2} \geq \boldsymbol{M}$

这样就得到了要保证后条件，此规范的最弱前条件

约束选择代换

假定抽象机里有序列 $vseq \in \mathsf{seq}(VALUE)$，但将来抽象机的实现对序列实现会有某种限制。在写抽象规范时，不可能知道最后的限制是什么

要为这一抽象机描述抽象操作 $push\_back$，由于上述限制，这一操作可能成功将元素压入，也可能失败

可以让操作返回一个报告，将其描述为

$report \longleftarrow push\_back(\boldsymbol{val}) =$
PRE
$val \in VALUE$
THEN
CHOICE
$report := ok \;||\; vseq := vseq \leftarrow val$
OR
$report := fail$
END
END

无约束选择代换 ANY

B 语言里的无约束选择代换 **ANY** 使人可以基于一个值的任意选择来描述操作，使后面执行的动作依赖于这个任意值

无约束选择代换的描述形式是

ANY x **WHERE** P **THEN** S **END**

其中包含三个成分

- x 是一个新变量，应该与本代换出现环境中的变量不相交。也就是说，它不应与所在抽象机的状态变量，所在操作的输入或输出变量重名
- P 是一个有关 x 的谓词，它必须定义 x 的类型，还可以给 x 加以其他限制。它也可以引用其他变量，描述 x 与它们的关系
- S 是一个代换，称为 **ANY** 的体。其中可以引用 x，而如果引用了 x，其行为就会与 x 的值有关

从操作的角度看，**ANY** 建立了一个满足谓词 P 的新局部变量 x（P 确定了 x 的类型和可能值），在这个环境里做代换 S。**ANY** 的非确定性就在于 P 不必为 x 提供确定的初始值

代换 ANY

下面是一个例子：

ANY x **WHERE** $x \in \mathbb{N} \wedge x \leq total \wedge 2 \times x \geq total$ **THEN** $total := x$ **END**

这里希望把 $total$ 的值减小，但不减到一半一下

可以看到：

$total$ 原值	变换后的值	$total$ 原值	变换后的值
0	0	1	1
2	1, 2	3	2, 3
4	2, 3, 4	8	4, 5, 6, 7, 8

例：顾客选了一批商品 $choices \subseteq ITEM$，准备用信用卡付款。但信用卡有付款限额 $limit \in \mathbb{N}_1$。如果商品总价超过限额。顾客就需要去掉一些商品，只买余下的。这一操作可以描述为

ANY p **WHERE** $p \in \mathbb{P}(ITEM) \wedge p \subseteq choices \wedge cost(p) \leq limit$
THEN $purchases := p$ **END**

这里 p 的值是集合，**ANY** 的条件谓词要求 p 是原有商品选择的子集。在实际中，这种情况下顾客可能做出各种可能选择，也可能不买商品就离开。最后这种情况在这里的实际表现就是 $p = \varnothing$

代换 ANY 的最弱前条件

考虑什么条件能保证下面谓词成立

$$[\textsf{ANY } x \textsf{ WHERE } P \textsf{ THEN } S \textsf{ END}]Q$$

也就是说，做了 **ANY** 代换后 Q 成立

根据前面解释，这实际上要求，对任何 x，只要它满足谓词 P，做了 S 代换后 Q 都成立。因此有下面定义

$$[\textsf{ANY } x \textsf{ WHERE } P \textsf{ THEN } S \textsf{ END}]Q \ \widehat{=}\ \forall x.(P \Rightarrow [S]Q)$$

假设要证明做了前面的值减小操作之后 $total$ 的值不小于 2，有

$$\left[\begin{array}{l}\textsf{ANY } x \textsf{ WHERE } x \in \mathbb{N} \wedge x \leq total \wedge 2 \times x \geq total \\ \textsf{THEN } total := x \textsf{ END}\end{array}\right](total \geq 2)$$

$$= \forall x.(x \in \mathbb{N} \wedge x \leq total \wedge 2 \times x \geq total \Rightarrow [total := x](total \geq 2))$$

$$= \forall x.(x \in \mathbb{N} \wedge x \leq total \wedge 2 \times x \geq total \Rightarrow x \geq 2)$$

通过简单的算术运算和推理，可知前条件是

$$total \geq 2$$

ANY 和 :∈

实际上，B 语言里的 :∈ 只是一类特殊形式的 **ANY** 结果的缩写

$$x :\in T \ \widehat{=}\ \textsf{ANY } e \textsf{ WHERE } e \in T \textsf{ THEN } x := e \textsf{ END}$$

表示的就是从一个集合中任意（非确定性）取值

把这种特殊形式代换到 **ANY** 结构里，可以得到相应的前条件

$$[x :\in T]P \ = \ \forall z.(z \in T \Rightarrow P[z/x])$$

其中的 $P[z/x]$ 表示把 P 里所有自由出现的 x 都换为 z，这里要求 z 是一个新变量，不是 x 也不在 T, P 里出现

如果 x 不在 T 里出现，上面公式可以简化为

$$[x :\in T]P \ = \ \forall x.(x \in T \Rightarrow P)$$

例：考虑下面推导（假定 x 不在 T 里出现）

$$\begin{array}{rcl}[x :\in T](x > 10) & = & \forall x.(x \in T \Rightarrow x > 10) \\ & = & \mathsf{min}(T) > 10\end{array}$$

LET 和 ANY

LET 结构用于引进新的局部变量，并给它特定的初值。**LET** 的形式是

$$\textbf{LET } x \textbf{ BE } x = E \textbf{ IN } S \textbf{ END}$$

类似的 let 结构出现在许多函数式语言和模型语言里

实际上 **LET** 只是 **ANY** 的一种特殊形式的语法包装

$$\textbf{LET } x \textbf{ BE } x = E \textbf{ IN } S \textbf{ END} \quad \widehat{=} \quad \textbf{ANY } x \textbf{ WHERE } x = E \textbf{ THEN } S \textbf{ END}$$

这里的 **WHERE** 之后的谓词具有特定形式，是用等式给定变量初值

一般而言，允许用一个 **LET** 结构引进一组局部变量：

$$\textbf{LET } x_1, \ldots, x_n \textbf{ BE } x_1 = E_1 \wedge \ldots \wedge x_n = E_n \textbf{ IN } S \textbf{ END}$$

一个 **ANY** 结构也可以同时引入一组变量，谓词 P 说明它们的性质

$$\textbf{ANY } x_1, \ldots, x_n \textbf{ WHERE } P \textbf{ THEN } S \textbf{ END}$$

这些代换形式的例子后面可以看到

定义子句

在继续讨论其他代换结构之前，先介绍 B 语言里的显式定义特征。通过显式定义，可以把复杂的描述分解为一些部分，有助于规范的描述

DEFINITIONS 用于引进定义，其效果类似 C 语言的宏定义。与 C 的宏定义类似，Atelier B 支持定义单独的名字和带参数的名字，还可以在 **DEFINITIONS** 子句里引用内容都是定义的 "定义文件"。详情见 Atelier B 语言手册

这里用一个例子 **DEFINITIONS** 定义：

DEFINITIONS

$ADDRESS == \mathbf{0..}(memorysize - \mathbf{1})$
$PAGE == \mathbf{0..}((memorysize/pagesize) - \mathbf{1})$
$INDEX == \mathbf{0..}(pagesize - \mathbf{1})$
$addr(pn, ind) == (pn \times pagesize) + ind$

前三个定义的是简单的名字，最后一个定义的是带参数的名字。这样，写

$$addr(\mathbf{25}, \mathbf{36})$$

就相当于写：

$$(\mathbf{25} \times pagesize) + \mathbf{36}$$

完整例子见《B Book》4.19 节

定义和例子

B 抽象机里各种主要子句的排列顺序很重要，但 **DEFINITIONS** 子句的位置比较灵活。显然的要求是被引用名字应该已有定义。例如， 对上页的几个定义而言，在这些定义前，*memorysize* 和 *pagesize* 应该已定义

例（《B Book》4.17 和 4.19节）：现在要描述一部数据库机器，其中保存着某个人群的个人信息

- 假定每个人有一个性别（famale 或 male）
- 某个人有一个生存状态（living 或 dead）
- 机器提供出生、死亡、结婚等基本操作，以及一些查询操作

MACHINE $Person$
SETS
  $PERSON; SEX = \{male, female\}; STATUS = \{living, dead\}$
CONSTANTS
  max_person
PROPERTIES
  $max\_person \in \mathbb{N}_1 \wedge \mathsf{card}(PERSON) = max_p erson$

实例

定义了变量之后，我们引进一批定义：

VARIABLES
  $person, sex, status, mother, husband, wife$
DEFINITIONS
  $MAN == sex^{-1}[\{male\}];$
  $WOMAN == sex^{-1}[\{female\}];$
  $LIVING == status^{-1}[\{living\}];$
  $DEAD == status^{-1}[\{dead\}];$
  $MARRIED == \mathsf{dom}(husband \cup wife);$
  $SINGLE == person - MARRIED;$
  $ANGLE == PERSON - person$
INVARIANT
  $person \subseteq PERSON \ \wedge \ sex \in person \rightarrow SEX \ \wedge$
  $status \in person \rightarrow STATUS \ \wedge$
  $husband \in WOMAN \rightarrowtail\!\!\!\!\!\!\!+ \ MAN \ \wedge$
  $wife = husband^{-1} \ \wedge$
  $mother \in person \twoheadrightarrow\!\!\!\!\!\!\!+ \ (MARRIED \cap WOMAN)$

实例

初始化和部分操作

INITIALISATION

$person, sex, status, mother, husband, wife := \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing$

OPERATIONS

$death(psn) =$

PRE $psn \in PERSON \wedge psn \in LIVING$

THEN

$status(psn) := dead$

END;

$marriage(bride, groom) =$

PRE

$bride \in PERSON \wedge groom \in PERSON \wedge$
$bride \in SINGLE \cap WOMAN \wedge$
$groom \in SINGLE \cap MAN$

THEN

$husband(bride) := groom \parallel wife(groom) := bride$

END;

实例

其他操作

$baby \longleftarrow newborn(sx, mo) =$

PRE

$sx \in SEX \wedge mo \in PERSON \wedge$
$PERSON \notin person \wedge mo \in MARRIED \cap WOMAN$

THEN

ANY $angle$**WHERE** $angle \in ANGLE$ **THEN**

$person := person \cup \{angle\} \parallel sex(angle) := sx \parallel baby := angle \parallel$
$status(angle) := living \parallel mother(angle) := mo$

END

END;

$report \longleftarrow is\_present(psn) =$

PRE $psn \in PERSON$ **THEN** $report :=$ bool$(psn : person)$ **END;**

$report \longleftarrow is\_living(psn) = \cdots\cdots$

$report \longleftarrow is\_woman(psn) = \cdots\cdots$

$report \longleftarrow is\_married(psn) = \cdots\cdots$

END