形 式 化 方 法:

基于 B 方法的严格软件开发

(1) B 方法概述

裘宗燕

北京大学数学学院信息科学系

2010年春季

软件开发方法

作为一种软件设计技术，需要：

- 一套描述软件系统的形式化记法
- 一套设计实现软件系统的开发技术
- 一组软件开发支持工具

例如，支持面向对象的开发，一种可能的技术：

- 用 UML（建模语言）和 Java（编程语言）
- 采用面向对象的软件系统建模、开发和程序设计技术
- 支持 UML 的建模工具集和支持 Java 开发的工具集

B 方法和 B 语言

B 方法是一种描述、设计软件系统直至生成软件系统代码的形式化方法， 提供了一套描述软件规范的语言，B 语言

B 方法支持：

- 在抽象到具体的不同层面上描述软件系统的模型（软件规范）
- 基于抽象模型建立更具体的软件模型（从抽象规范到更具体规范的精化）
- 描述软件系统的性质（系统模块的不变式）
- 生成保证软件系统的完整性而必须证明的不变式定理（证明义务）
- 生成保证软件系统精化的正确性而必须证明的精化定理（证明义务）

B 方法的相关理论保证：

- 不变式定理可以自动生成
- 精化定理可以自动生成
- 一部分证明义务有可能自动完成
- 由最终具体规范自动生成实际代码

基本概念

用 B 描述软件的基本单元是抽象机 (Abstract Machine)

抽象机类似于我们常说的抽象数据结构，包括

- 数据描述（常量，变量等）
- 操作描述（数据上的一组操作）
- 不变式（数据状态必须满足的一组关系）

要保证一个抽象机 M 的描述是完整的无矛盾的，需要证明

- M 的所有可能初始状态都满足它的不变式
- 从任何满足 M 的不变式的 M 抽象机状态出发，执行 M 的任何操作，可能达到的状态必定满足 M 的不变式

这些是由抽象机生成的不变式定理

不变式定理和其他要证明的定理称为“证明义务”（Proof Obligation, PO）

如能证明由抽象机 M 生成所有不变式定理，M 就是一致的（无矛盾的）

将（软件）系统看作抽象机—概念

将（软件）系统看作抽象机，基本想法是一个系统或部件：

- 有自己的状态
- 其状态由一组常量和变量的取值表示 （所有可能取值确定了这一抽象机的状态空间，一组具体取值确定了抽象机的一个具体状态）
- 提供了一组操作
- 这些操作的执行可能改变系统或部件的状态

例：一台电视机：

- 内部状态：
  - 当前处于工作状态还是休眠状态
  - 频道设置状态
  - 当前频道，当前音量，当前显示状态 (亮度/对比度/色彩等)
  - ……
- 操作：由电视机操作键或遥控器提供的各种操作

B 方法和抽象机

学习 B 方法，需要学习：

- 如何用抽象机描述软件
- 如何保证一个抽象机的一致性
- 如何基于已有的抽象机去构造大型抽象机
- 如何基于抽象规范去开发更具体的规范， 并保证开发出的更具体的规范不打破抽象规范已证明的性质
- 如何从规范最终得到软件的实现

B 方法的基本想法（也是目前形式化软件开发的基本想法）是：

- 在不同抽象开发层次上证明系统的性质，证明所开发的规范满足需要
- 基于上述证明，保证最终的软件系统满足我们的需要

B 抽象机的基本结构

一个 B 抽象机从关键字 **MACHINE** 开始到对应的 **END** 结束，其中可以包含许多成分（子句）。最基本的成分包括：

1. 抽象机子句，抽象机的头部，**MACHINE** 后面写抽象机名
2. 状态描述
   - 变量子句，以关键字 **VARIABLES** 标识，其中列举本抽象机的变量，可以声明任意多的变量，变量之间用逗号分隔
   - 不变式子句，以关键字 **INVARIANT** 标识，其内容是一个逻辑公式，描述变量之间的关系。多个不变式用合取连接
   - 初始化子句，以关键字 **INITIALISATION** 标识，给定变量初值
3. 操作描述；可以包含一系列操作的描述，每个操作描述其：
   - 输入
   - 输出
   - 对抽象机状态的影响

还可以有许多其他成分

一个简单的抽象机

```
MACHINE counter

VARIABLES ii

INVARIANT ii : INT & ii >= 0

INITIALISATION ii := 0

OPERATIONS
  inc = BEGIN ii := ii+1 END;
  reset = BEGIN ii := 0 END;
  nn <-- get = BEGIN
                 nn := ii
               END
END
```

MACHINE $counter$

VARIABLES ii

INVARIANT $ii \in \mathbf{INT} \wedge ii \geq 0$

INITIALISATION $ii := 0$

OPERATIONS

$inc =$ **BEGIN** $ii := ii + 1$ **END;**

$reset =$ **BEGIN** $ii := 0$ **END;**

$nn \longleftarrow get =$ **BEGIN**

$nn := ii$

END

END

左边是计算机形式（字符正文），右边是数学形式。下面主要用数学形式

注意这里的证明义务：初始化定理和操作的不变式维持定理

变量和初始化

下面简单介绍抽象机的几个最基本部分的情况

变量子句声明本抽象机里的变量，变量的取值表示抽象机的状态

变量子句的例子：

VARIABLES $top, num, volumn$

初始化子句给本抽象机里的所有的变量指定初始值

初始化子句的例子：

INITIALISATION

$top := 0;$
$num := 0;$
$volumn := 10$

实际上变量的初始值还可以是非确定性的值，说明从一个集合里取值

INITIALISATION

$vip :\in VIP;$
$cond :\in BOOL$

基本类型

B 语言提供了如下基本类型：

BOOL	布尔类型，包括值 **TRUE** 和 **FALSE**
INT	实现确定的整数类型
NAT	实现确定的自然数类型
$\mathbb{Z}$	整数类型，在正文形式中的名字是 INTEGER

类型的构造和类型描述下面讨论

例如，可以写整数的子界类型：

$$num \in \mathbf{1..100}$$

$$max :\in \mathbf{1000..2000}$$

每个变量和常量都有类型

对于写出的 B 抽象机，需要首先通过类型检查，检查其是否良类型

B 工具都先做类型检查

不变式子句

不变式子句的内容是一个逻辑公式，通常是一个合取式， 其中每个合取项是一个谓词，描述抽象机的变量、常量等之间必须保持的关系

不变式子句的合取式里的一个谓词称为一个不变式

B 语言中用谓词说明变量的类型，也写在不变式里

不变式子句的例：

INVARIANT

$ii \in NAT \wedge$

$ii < \mathbf{20} \wedge$

$volumn \in \mathbf{NAT} \wedge$

$num \in \mathbf{NAT} \wedge$

$num \leq volumn \wedge$

$found \in \mathbf{BOOL}$

操作的描述

操作可以没有返回值，也可以有一个或几个返回值， 有一个操作名和一个参数表（可以没有），随后的等号后面是操作体。例如：

$reset$ = **BEGIN** ii := 0 **END;**

$nn \longleftarrow get$ = **BEGIN** nn := ii **END**

书写形式不影响语义

$nn \longleftarrow get$ =
BEGIN
nn := ii
END

返回值是操作的输出，参数是操作的输入

操作体描述对抽象机状态的改变，以及输出值的获得等

上面操作的体都是块结构，还可以有其他结构（下面介绍）

一些 B 工具不允许单个字母的名字（包括Atelier B），所以用 ii, nn 等