形式化方法

Formal Methods (1)

•裘宗燕

•北京大学数学学院

•2010年3－6月

1. 引 言

- 软件开发的过程和问题
  - 设计和制造
- 形式化方法
- 形式化方法的实践
- 对本课程的设想
- **B** 方法和相关技术简介

软件开发过程

- 软件开发的主要步骤（大致，有不同说法）：
- 问题和需求分析
- 设计
  - 功能设计
  - 结构设计
- 编码（构建）
- 调试
- 运行和维护、升级

上面是比较公认的软件开发活动。实践中提出了许多开发模型：如传统瀑布模型，新近的快速原型/迭代式开发/模型驱动的开发/测试驱动的开发等

软件开发

软件开发中考虑的关键问题：

- 软件的功能保证
- 软件的其他质量保证（非功能性保证）
- 开发成本
- 开发效率

有关计算机软件的研究工作都围绕着下面一些问题：

- 提高软件系统的质量和可靠性
- 提高开发的效率
- 降低开发的成本

问题：如何做到这些？

软件的本质

- 软件是联系计算机硬件与应用的桥梁，其本质特征是复杂性和多样性
- 本质特性之一：极端的复杂性
  - 规模巨大，可能包含成百万、成千万行代码，成万代码和数据文件
  - 各组成部分之间异常复杂的直接或间接相互作用方式
  - 静态结构与动态性质之间难以把握的复杂关系
- 本质特征之二：多变性
  - 异常丰富多彩的应用需求（无穷无尽的新应用问题）
  - 需求的不断提升和变化（总有新需要，总要不断修改）
- 需要较低的开发成本和较短的开发周期
- 不可能长期集中一大批最优秀的技术人员（对航天飞机软件，有可能；对手机控制软件，不可能），与**CPU**设计的情况不同

软件开发的问题：应对

- 计算机软件是人类开发的最复杂的制品。任何有助于缓和系统复杂性的影响的技术和方法都有价值

人们已经在许多层面上努力：

- 对软件开发活动的良好管理和组织
- 建立严格的工作规范，设法严格深入地控制软件开发过程
- 安排丰富的人际信息交流活动，希望能尽早发现软件需求、设计等各阶段引入的错误和不良决策。以上是软件工程研究的重要内容
- 采用多层次抽象技术，将复杂系统分解为相对独立的层次或部分，提供封装和清晰界面。如系统开发的层次模型，功能分解，模块化，**OO**等
- 研究程序开发模型及其支持技术，屏蔽软件开发难点和共性问题。如图形用户界面技术，客户端/服务器模型，中间件技术，**web**服务模型等
- 研究软件组件形式，设法尽可能重用已有开发成果，提高开发的基础层次，降低开发的复杂性和代价。如子程序库，**OO**类库，组件库等

软件开发的问题：应对

- 设计各种性质良好的描述形式，特别是在设计阶段建立和检验软件的模型的适当描述形式，以满足在各个层次上描述软件及其相关事物的需要。如数据流图、控制流图、状态机模型、**UML**等
- 设计合适的开发语言（包括编程语言），研究编程活动中的规律性和适用的模型，以支持有效的良好构建过程。例如结构化程序设计及结构化语言，模块化语言，面向对象的技术及相关语言等
- 开发适用的软件工具，支持软件开发中各个阶段的不同活动的需要，支持开发人员对开发结果做深入的分析和检查。如编辑器、调试器、各种程序分析工具、集成程序开发环境、**UML**支持工具等
- 培训人员，提高软件开发参与者的认识能力和工作能力
- 等等

软件开发：实际情况

但软件开发的实际情况并不乐观：

- 项目经常延误，不能按期完成
- 经常超出预算
- 软件开发的后续阶段常常发现许多前期设计错误，更正错误的代价高昂
- 发布运行的软件中常常存在着许多错误，时常崩溃
- 软件维护和更新的代价非常高
- 典型现象：无人愿意给软件做保险（现在人们开始考虑这一问题）

怎样解决这些问题？

- 没有万灵的神药。任何不幼稚的计算机工作者都不应期望有朝一日能发明某种新技术或新方法，一劳永逸地解决软件开发领域里的难题
- 任何有助于解决问题的技术和方法都是有价值的
- 形式化方法可能（也已经）在软件开发的各阶段和各种活动中起作用

软件：设计和制造

- 传统的认识是软件没有制造过程，只有设计。在中文里，软件的实现阶段也被称作“程序设计”
- 在传统的软件开发过程中，前期各阶段的工作成果都是自然语言描述的“文档”，只有最后开发阶段的成果是形式化的程序
- 非形式的前期工作成果（文档）只能由人阅读、理解和讨论，无法严格的分析和推理（这方面情况也正在变化）
- 原则上说，程序是形式化的描述，有“清晰”定义的形式和语义
- 程序的所有静态和动态性质都蕴藏在程序正文中，但是：
  - 程序里除包含了实际应用的功能逻辑外，还包含大量具体语言和具体实现的细节，这些使程序的逻辑分析变得非常困难
  - 大量实现细节大大增加了程序的复杂度，从而极大地提高了分析和推理的成本（通常不是规模的线性函数，可能是指数的）
- 由非形式的软件前期开发成果到复杂的程序，这一跨越的距离太大，相互之间的关系难以把握，无法保证

软件：设计和制造

- 严格描述出现得太晚，形式太复杂，与前期成果的距离太远
- 由于程序的复杂性，对其进行分析和推理非常困难，不可能完全（对程序的推理的分析也是形式化方法研究的课题）
- 可以把软件的前期工作看成设计，把编程工作看作制造
- 需要加强软件的设计过程：
  - 更可控的设计过程。
  - 给设计过程更好的基础性支持
  - 在设计过程中深入分析前期制品的内在性质，尽早发现并更正设计中隐藏的错误和缺陷
  - 需要清晰而无歧义的制品描述形式（包括设计阶段的产出品），要在设计和实现之间建立严格定义的关系（什么是正确的实现）
- 在软件设计实现的各阶段引进严格的形式化描述，就可以对软件工作的早期成果进行分析和推理，为后续开发打下更好的基础

朴素方法、形式化方法和半形式化方法

- 传统的软件设计方法：
  - 基于自然语言思考、设计和描述
  - 语义含糊，可能有歧义，依赖于使用者的理解
  - 无法进行严格的检查，只能通过人的交流活动进行分析
- 半形式化的方法（例如基于 **UML** 等）
  - 有较清晰定义的形式和部分的语义定义
  - 一些描述的语义较清晰，可能开发工具进行检查和分析
- 形式化方法：
  - 基于严格定义的数学概念和语言
  - 语义清晰，无歧义。可以开发自动化工具进行检查和分析
- 目前情况：软件的开发正在从朴素的、非形式的设计方法，向着更加严格、更加形式化的方向转变（特别是在大型/复杂/关键系统方面）

形式化方法

- 形式化方法研究如何把（具有清晰的数学基础的）严格性（描述形式、技术和过程等）融入软件开发的各个阶段：
- **Specification**（规范）：采用具有严格定义的形式和语义的记法形式，描述软件设计（和实现）
- **Reasoning and Analysis**（推理和分析）：对形式化规范进行分析和推理，确定它们的各种静态和动态性质
  - 规范是否具有一致性和完整性，其中有没有矛盾，有没有遗漏？
  - 运行中不会出现某些不能容忍的状态（死锁、活锁等）
  - 找出其中的错误和缺陷等
- **Refinement**（精化）：从抽象的高层描述出发，严格保语义地推导出更接近实现的包含更多细节的规范，最终得到正确实现了高层规范的可运行程序（逐步求精技术的严格化，另一些软件技术也可以看作精化）
- 等等

形式化方法的实践

形式化方法早期的最成功范例是编译程序的词法、语法分析和语义处理

- 早期的编译实践完全靠朴素理解，被认为是一类非常复杂和困难的工作，需要许多人花很多时间才能完成
- 研究中认识到：
  - 语言与语言处理器的关系就是形式语言类（正则语言、上下文无关语言等）与自动机类（有穷自动机、下推自动机等）的关系。对一些结构较简单的类（如上面的语言和机器类），可以从语言的形式化描述（文法）自动生成能够识别该语言的有效的自动机
  - 语义处理可以通过属性文法模型等很自然地形式化描述
- 可以根据相关理论构造分析和检查工具，自动检查文法描述中的许多问题（如歧义性，在某种分析中体现为“移入-归约冲突”）
- 现在语言处理程序已变成了简单工作，只要用**BNF**等描述好文法，就可以“程序化地”手工写出语言的词法/语法分析器，或自动生成分析器

形式化方法的实践

形式化方法在软件实践中已经有了许多实际应用：

- 常规软件实践已经融入了许多形式化方法的元素
  - 许多面向实际开发的教材和著作都在讨论断言、前/后条件、循环不变式和数据不变式等形式化术语和技术
  - 人们提出了基于契约的编程，净室技术等，并用于软件实践
  - 在 **UML** 等被广泛采用的建模和描述工具里，包含大量形式化方法研究成果，如：断言、状态、状态机、状态转换、约束和约束语言**OCL**、不变式（不变量）、前/后条件等等
- 一些有关计算机系统开发的规范性文件，强调了在某些类软件的开发过程中应用形式化方法的必要性（有的强调必须采用形式化方法）
  - 特别是各种安全攸关的（**safety-critical**）和生命攸关的（**life-critical**）的系统的开发，例如核电站监控系统、武器系统等
  - 社会生活中常见的许多系统也有这种性质，如：交通调度、铁路信号、交通工具控制、自动驾驶、医疗设备控制等许多系统

形式化方法的实践

形式化方法在实际的软件开发中也有了一些应用（虽然还不广泛）

- 《**Using Z**》介绍了 **1980** 年代末期**IBM**在做**CICS**（**Customer Information Control System**）系统更新和升级的过程中，利用 **Z** 语言做再工程（**Re-Engineering**）的工作，取得了影响广泛的成果
- **J-R Abrial**在**1980**年代后期开发**B**方法的过程中，法国**GEC-ALSTHOM Transport**公司利用**B**方法开发的法国高速铁路控制系统，开发取得了非常好的效果，成为形式化开发的典范
- 近些年人们报告了许多采用形式化方法支持的软件系统开发实例，特别是各种安全攸关和生命攸关的系统。例如：
  - 交通工具（汽车、飞机等）的控制系统
  - 交通指挥调度控制系统，自动化运输系统
  - 核电站控制系统
  - 从有关形式化方法，以及有关 **B** 和 **Z** 的网页可以找到许多开发实例

形式化方法的实践

- 形式化方法在硬件领域有许多应用，例如：
  - **90**年代**Moore**等人受**AMD**公司委托，证明了**ABD K5**浮点除法运算的的正确性，其形式化验证公司为各种硬件厂商做了许多形式化验证工作
  - 模型检查技术在硬件领域得到广泛应用，已成为硬件设计过程中一种标准技术
  - 从**1990**年代末开始大硬件公司都招募形式化方法、模型检查等领域的专业人员（博士/博士后等），开展硬件形式化分析和验证（**1990**年代**Intel** 的奔腾芯片浮点错误造成很大经济损失，硬件公司从中汲取经验和教训）
- 另一事实：微软近年在若干研究院建立了研究软件理论的队伍
  - 吸收了软件理论方面的一批重要人物，如：**Hoare**，**Milner**，**Lamport** 等
  - 开发了一些实用工具，例如 **SLIM**（支持设备驱动开发）。要求工程开发的软件都要用一些基于理论的工具检查
  - 开发了一些实验性的基于程序理论的系统，如 **Spec#**。组织队伍验证自己的几个关键性软件（部分）
  - 组织了 **Verified Software** 研讨会（**2009.10**，北京）

形式化方法的实践

- 一些软件和计算机规范中采用形式化方法的描述。例如：
  - **RBAC 2004** 规范中采用了形式化规范语言**Z**描述
  - **W3C**的**WSDL 2.0**规范（**Z**描述，**2005**），**W3C**的**XQuery 1.0 and XPath 2.0 Formal Semantics**（结构化操作语义，**2005**）
  - 原因：采用朴素的非形式方式，规范中常常存在许多漏洞，存在许多不一致的或者不准确的地方
- 用形式化方法研究各种复杂的系统问题
  - 研究各种网络规范和安全规范
  - 研究程序设计语言和其他复杂的软件规范的形式化模型和语义
  - 基于形式化模型的程序分析工具在**Windows 2000**发布版本的源代码中找出了数以万计的错误和漏洞
  - 等等

对一些问题的回答

人们对形式化方法提出了许多置疑。现在考虑其中的一些问题

1. 没有形式化方法，人们不也做出了许许多多的软件吗？
   - 做得不好！
   - 社会越来越依赖计算机系统的安全运转，目前这种设计者也没有清楚理解，没有良好质量保证的软件还能（应该）继续存在下去吗？
2. 我没学过形式化方法，也做了许多程序和软件！
   - 如果掌握了形式化方法的思想和技术，可能做得更好
   - 没学过建筑科学的人也能搭出很好的窝棚，但能允许他们去设计和建造摩天大厦、跨海大桥、高速铁路吗？
   - 如果不了解一些形式化方法，将来会发现许多东西看不懂了（如各种规范），许多工作没有资格参加了（大型/重要/安全攸关的系统），许多工具不会用了，……。应该早做准备！

对一些问题的回答

3. 形式化方法太难了
   - 程序也是形式化描述。对 “同一”问题的程序通常比在抽象层次上的形式化描述复杂得多（例：描述或创建不大于**20**的偶数的集合）。后边会看到许多实例
   - 既然投身这个领域，要学如何做出“正确的系统”，就要学习严格的形式化的思考和描述技术（原来用自然语言思考软件的设计问题，现在用**UML**思考，将来用更严格更形式化的方式思考）
   - 形式化方法可能没有想象的那么难，只是因为不习惯，没掌握。新型方法和新工具的开发将进一步降低应用形式化方法的难度
4. 形式化方法有什么用？
   - 前面讨论中包含了许多有关这个问题的内容
   - 在常规开发中，形式化方法的思想和技术也很有用（如程序中的断言和不变式，基于契约的开发等，都在实际中发挥着作用）

形式化方法

- 软件是最复杂的人工制品，难设计/分析/理解，难以做对。形式化描述和技术至少是增加了一种观点，因此会对系统的开发工作有所帮助
- 我在《**B**方法》的译者序里写
  - “计算机已被看作……（一种）最强有力的工具，是‘用之万事而皆灵’的魔杖。计算机的介入改造了所有行业和部门，不断改变着我们的工作和生活方式，是否应用计算机已经成为评判一个行业是否现代化的最重要标志。”
  - 为什么？
- “计算机化”就是应用领域的“形式化方法”：把不严格不清晰的操作和过程严格化、计算机化，就是一种形式化

  行业的现代化，其中的一个重要方面也就是“形式化”
- 这一论断同样适合计算机领域，这是一个特别需要形式化的领域
- 总之：形式化方法，对我们（个人、领域）的未来都非常重要

形式化方法：不是万灵神药

形式化方法并不是解决软件开发问题的万灵神药：

- 在非形式的客观需求与形式化规范之间的关系，不可能形式化地处理
- 形式化规范可能较难读，因为具有良好形式化训练的专业人员较缺乏
- 虽然理论上说可能对形式化规范进行深入分析，证明所需性质（或证明没有某种不良性质），但实际证明还是比较复杂，需要强有力工具的支持
- 形式化方法的支持工具仍然比较缺乏，其功能不够丰富，界面和交互方式不够友好，使用不够方便
- 基于形式化方法开发大型软件的过程还缺乏深入研究
- 软件的形式化描述技术本身也还不够成熟，特别是：
  - 大型系统的描述，描述的组织
  - 如何应对需求的变化
- 形式化方法不能保证不出错，但能有效地帮助人减少错误

课程计划

- 形式化方法已取得了许多成果，形成了一个内容丰富领域
- 国外许多重要大学有形式化方法课程，课程内容差别很大，以讨论软件的形式化理论的为多，如：系统建模，模型检查，等。这个课程还不成熟
- 国内少数学校开设这个课，情况与国外类似（我以前上的课也类似）
- 这次课想调整重点，更关注基于形式化方法的软件系统开发：
  - 介绍在实际软件系统开发中应用比较成功的**B**形式化开发方法
  - 研究基于**B**方法描述计算系统的技术和方法（ **Specification**问题，如何严格地写规范，描述要开发的软件系统或者其他系统）。希望大家能学会严格地描述实际的软件系统（设计和写规范）
  - 考虑一些进一步内容，如性质证明、功能精化和数据精化
- 希望这个课能更面向实际，反映形式化方法在实际软件开发中应用的情况
- 希望大家共同努力，通过学习讨论初步掌握描述软件系统的严格技术

课程计划

课程计划分为**4**个部分，交织进行

1. 介绍 **B** 方法的一些基础知识，用于写软件规范的 **B** 语言结构和细节，并适当介绍 **B** 方法的理论基础
2. 研究如何用 **B** 方法开发软件。个人做一些小的软件开发课题
3. 课程参与者交流使用 **B** 方法的经验，讨论相关问题
   - 有哪些主要的启发
   - 遇到哪些难点，是 **B** 语言和 **B** 方法的问题，还是我们的问题
   - 形式化方法开发
   - 我们还需要形式化方法和工具提供什么
4. 最后每人写一篇课程论文，报告自己的工作，讨论一些有关的问题。最后安排几次课，参加学习的人报告

本课程没有期终考试，以课堂参与、工作成果和最后论文评分

教学参考书和参考材料

- 课程网页：
- **http://www.math.pku.edu.cn/teachers/qiuzy/fm_B/**
- 教学参考书：**B**方法， **J-R Abrial**
  - 只作为参考，课程 不准备按参考书讲，而是想更面向实际
- 参考材料：
  - 有关参考材料见网页
- **B** 方法开发工具：
  - 用法国 **Clearsy** 公司发布的 **Atelier B** 系统
  - 这是目前在 **Windows** 平台上可用的 **B** 系统
  - 有较好的图形用户界面，类型检查，交互式证明器

B 方法及相关技术简介

- 法国计算机科学家 **J-R Abrial** 于 **1980** 年代在牛津大学程序研究组（**PRG**，**Hoare**长期领导 **PRG** 的工作）访问时提出 **Z** 语言，回国后希望形式化方法的研究更加面向实际软件开发过程，后开发了 **B** 方法
- **B** 语言和 **B** 方法基于一阶逻辑和集合论，作为其基础的抽象数学对象包括数据元素、集合、映射和代换等
- **B** 方法有严格的数学理论基础，写出的规范简明、精确，无歧义性
- **B** 方法支持的形式化方法也称为面向模型的方法，或基于状态的方法。其基本思想是从一些已知的简单的抽象数学对象出发，构造（描述）目标（软件）系统的状态特征和行为特征模型
- 与 **B** 类似的面向模型的方法还有著名的 **VDM**、 **Z** 等
- 除了面向模型的方法，还有一些其他的方法类，如代数方法等
- 特别的是，**B** 方法是在实际使用中逐渐发展完善起来的， **J-R Abrial** 刚开发出 **B** 方法，就与当时要开发铁路信号系统等的公司一起工作

B 方法简介

- **B** 方法是在开发巴黎地铁信号系统的过程中发展起来的，后来又用于开发巴黎的若干地铁/机场自动车的驾驶系统
- **J-R Abrial** 在 **2006** 年国际软件工程大会上的特邀报告“工业开发中的形式化方法，成就、问题和未来”中介绍了两个用 **B** 开发的实际系统：
  - 巴黎地铁**14**号线自动驾驶系统，**1998**投入运行
  - 巴黎**Roissy**机场自动穿梭车，**2006**年投入运行

线路全长	**8.5**公里		线路全长	**3.3**公里
停站数	**8**		停站数	**5**
列车间距时间	**115**秒		列车间距时间	**105**秒
速度	**40**公里/小时		速度	**26**公里/小时
列车数	**17**		列车数	**14**
每天旅客数	**350000**		每小时旅客数	**2000**

巴黎地铁**14**号线参数　　巴黎**Roissy**机场穿梭车参数

B 方法简介

两个项目软件的一些情况：

项目	巴黎地铁14号线	Roissy机场穿梭车
ADA代码行	86000	158000
证明定理个数	27800	43610
交互式证明的百分比	8.1	3.3
交互式证明所用人月	7.1	4.6

- 正在采用类似的方式进行开发的系统：纽约城市地铁，巴塞罗那地铁，布拉格地铁，巴黎地铁1号线 等
- 国内这方面工作开展较少。但一些关键部门已开始关注。国家近几年在“可信软件”领域启动了多个大型研究项目，形式化方法是最重要部分
- 国内引进的高速铁路控制系统，里面的软件开发中使用了B方法。消化吸收或提升这样的系统，需要 B 方法的基本知识和经验

用 B 方法开发软件系统的过程(1)

在 **B** 语言里的开发过程经过下述三个阶段

第一阶段：根据软件需求文档构造系统的抽象模型

- 基于需求文档，构造初步的抽象模型
- 向抽象模型中加入必要的不变式，基于工具提取并证明不变式定理（证明义务）。一些定理可以自动证明，有些需要人工交互证明
- 扩充已有模型，加入更多细节，证明不变式定理
- 不断重复上面几步，直至得到完整的抽象模型并证明所有定理

可能发现需求文档中不完全/不一致，需要交流/调整文档和模型

- 建立和扩充模型，加入不变式等完全是人工的工作
- 不变式定理（证明义务）的抽取将自动完成
- 工具可以自动证明许多定理，会有一些定理需要交互式证明

工作结果：待开发软件系统的一个完整的抽象模型

用 B 方法开发软件系统的过程(2)

第二阶段：开发软件系统的具体模型

- 在已有模型中加入更多细节，使更多的操作具体化，用更接近计算机实现的数据结构代替抽象结构，得到一个更为“具体”的模式
- 证明新的具体模型是原有模型的“精化”。使用工具自动生成一组“精化关系定理”，并证明这些定理
- 不断重复上面两步，最终做出一个可以直接对应到实现的模型

精化工作主要由人做，人们也开发了做精化的辅助工具

精化定理由工具自动生成，定理证明自动或交互式完成

第三阶段：生成系统的可执行代码

- 用工具检查模型是否已能翻译到编程语言代码
- 用工具把最终模型翻译到编程语言，用编译器生成可执行程序

这一阶段完全是自动的

学习 B 方法

学习 **B** 方法，要关注下面问题

- 首先理解其基本想法
  - 构造模型，用抽象的严格的数学描述方式刻画待开发的软件
  - 通过严格控制的精化过程，最终得到可执行的程序
- 在 **B** 方法里开发软件的思想、方法和技术
  - 如何思考和写出软件规范，通过实践掌握这些技术
  - 如何做精化
  - 如何做证明，特别是交互式证明
- 特别注意
  - 怎样在较高的抽象层面上描述软件的模型和性质
  - 怎样写不变式

好规范的特性

- 形式化开发的第一步是用形式化规范语言（如 **B** 语言）写出软件的规范
- 这也是本课程学习中需要第一步学习的
- 规范同样可以写的好或者不好，可能有用或者无用
- 好是质量，好规范就是质量高的规范，它应该：
  - 位于适当的抽象层次，完全地描述待开发的目标系统
  - 清晰且无歧义性
  - 简洁而易于理解
  - 容易维护修改，以满足系统需求变化和扩充的需要
  - 开发代价适当
  - 对软件开发工作真正有用