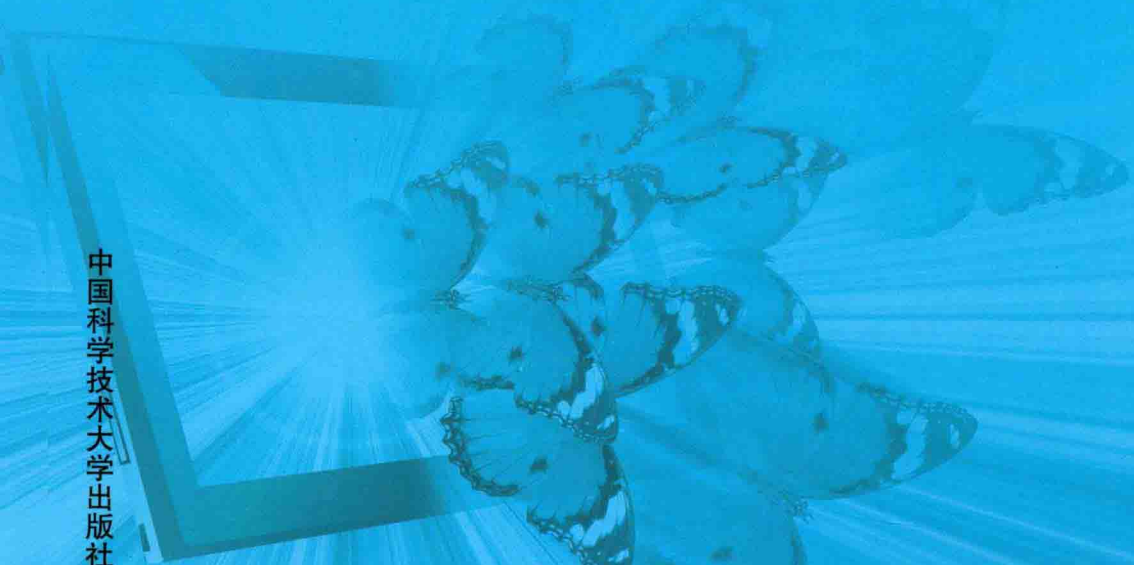
基于环境的软件近似正确性

Jiyu Huanjing de Ruanjian Jinsi Zhengquexing

马艳芳／著

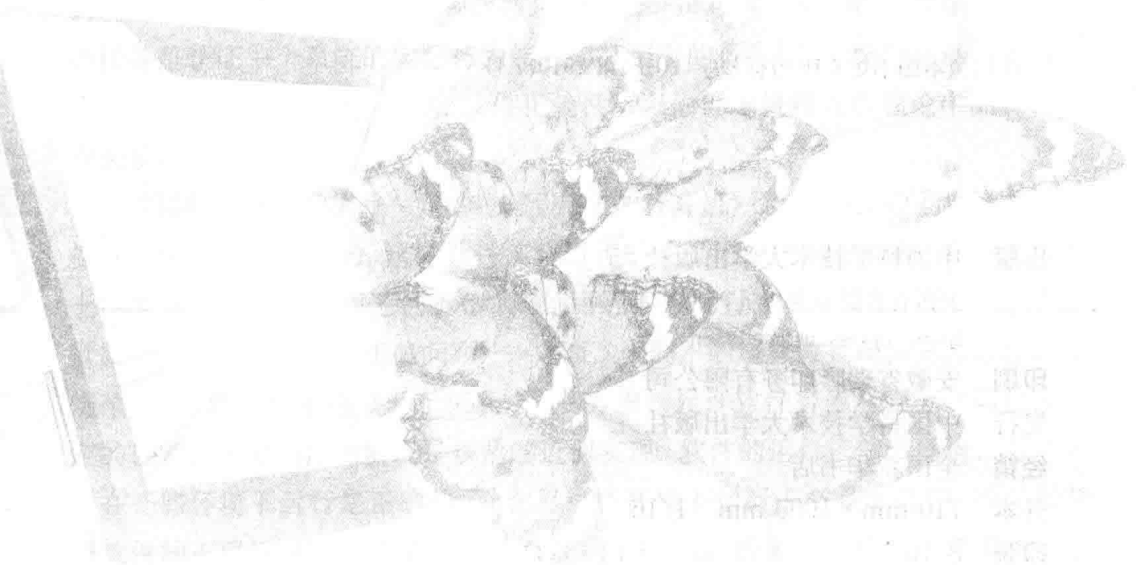


中国科学技术大学出版社

◎ 国家自然科学基金（61300048）资助
◎ 2014年安徽省高校优秀青年人才支持计划资助
◎ 淮北师范大学学术著作出版基金资助

基于环境的软件近似正确性

马艳芳／著



中国科学技术大学出版社

内 容 简 介

本书基于进程代数理论中的通信系统演算(CCS)模型对软件的近似正确性进行了形式化描述和度量. 从接受和拒绝环境角度，以参数化互模拟和三分之二互模拟为基础，建立了参数化互模拟和三分之二互模拟的无限演化理论及拓扑理论，建立了近似的参数化互模拟和近似的三分之二互模拟理论. 为描述带有概率信息软件系统的近似正确性，建立了概率化参数互模拟及确定性概率进程近似互模拟的无限演化及其拓扑结构.

图书在版编目(CIP)数据

基于环境的软件近似正确性/马艳芳著. —合肥:中国科学技术大学出版社，2017.1

ISBN 978-7-312-04102-0

Ⅰ. 基… Ⅱ. 马… Ⅲ. 程序正确性证明 Ⅳ. TP311.1

中国版本图书馆 CIP 数据核字(2016)第 291077 号

出版 中国科学技术大学出版社
安徽省合肥市金寨路 96 号，230026
http://press.ustc.edu.cn
印刷 安徽省瑞隆印务有限公司
发行 中国科学技术大学出版社
经销 全国新华书店
开本 710 mm×1000 mm 1/16
印张 8.75
字数 157 千
版次 2017 年 1 月第 1 版
印次 2017 年 1 月第 1 次印刷
定价 24.00 元

前　言

随着信息系统的发展, 软件作为信息系统的灵魂已经渗透到社会生活的各个方面. 由于人们对软件功能需求的不断增加, 软件系统变得日趋庞大和难以驾驭, 缺陷和漏洞难以避免, 系统越来越脆弱, 很多时候不以人们期望的方式工作. 由于软件并不总是能让人信任的, 从而软件可信性问题越来越得到人们的关注. 软件的可信性是在软件的正确性、可靠性、安全性等众多属性基础上发展起来的一个新概念. 软件的正确性是软件可信研究中一个重要的内容. 软件的正确性主要体现在软件执行结果是否符合人们的预期上. 在抽象层次上, 可以把软件执行看作软件实现, 把人们的预期看作软件规范, 这样软件的正确性就可以用软件实现与其规范之间的关系来表示. 进程代数理论是研究进程的一种理论，其中一个重要的任务是验证一个系统的实际行为能否与其想要的行为相一致, 即验证一个系统的实现是否满足其规范. 由此, 软件正确性可以借助于进程代数理论中的各种等价关系来刻画.

软件的实现都是依赖一定的物理设备 (如计算机) 而运行的, 而物理设备不能保证一定是完全可靠的, 这样软件实现就很难达到完全满足其规范, 在很大程度上一个软件实现是其规范的近似实现, 而近似程度直接影响着软件的正确程度, 这就产生了软件的近似正确问题. 一方面，人们不断地修改实现使其越来越满足规范, 进而软件越来越接近于正确. 另一方面，允许软件实现与规范之间存在一定的误差，由此给定的实现是规范的近似实现. 软件的运行依赖于环境，有些软件在一些环境下运行是正确的, 而在另一些环境下运行未必是正确的, 所以在考虑软件的正确性时需要考虑环境的因素. 由 Larsen 提出的参数化互模拟和三分之二互模拟分别从接受环境和拒绝环境角度反映了环境的变化.

为了刻画实际问题中软件的近似正确性，需要建立在参数化互模拟和三分之二互模拟下软件实现越来越满足规范的动态刻画以及软件实现与规范之间存在多大误差的静态描述. 本书基于进程代数理论中的通信系统演算（CCS）模型对

软件的近似正确性进行了形式化描述和度量. 从接受和拒绝环境角度, 以参数化互模拟和三分之二互模拟为基础, 建立了参数化互模拟和三分之二互模拟的无限演化理论及拓扑理论, 建立了近似的参数化互模拟和近似的三分之二互模拟理论. 为描述带有概率信息软件系统的近似正确性, 建立了概率化参数互模拟及确定性概率进程近似互模拟的无限演化及其拓扑结构.

本书在写作过程中得到了国内外许多专家学者的关心和帮助. 衷心感谢我的导师陈仪香教授, 他阅读了本书的部分书稿, 对本书做了充分的肯定和热情的推荐. 衷心感谢奥尔堡大学计算机学院 K. G. Larsen 教授, 北京大学信息科学技术学院金芝教授, 中科院软件所蒋颖研究员、柳新新研究员等对本书第 2 章、第 3 章、第 4 章内容方面给予的宝贵意见和建议. 同师好友吴恒洋教授、周洁博士、陶红伟教授、吴新星博士、潘海玉博士、姚兴华博士、陈艳文博士等都曾给予我极大的帮助和支持. 在此一并致谢!

最后, 向给予我巨大支持的丈夫陈亮、儿子陈志一等我的家人表示最诚挚的感谢, 他们的挚爱支持使我顺利地完成了本书的写作.

本书中的不少内容是作者最近完成的科研成果. 但限于作者的水平, 书中存在不妥乃至谬误之处都在可能之列, 希望各位专家与读者不吝赐教.

作 者

目　录

第 1 章 引 言

1.1 软件正确性及其意义

随着信息系统的发展, 以通信、存储和计算为核心的信息基础设施已经渗透到政治、经济、军事、文化等社会生活的各个方面. 软件是信息基础设施的灵魂, 由于人们对软件功能需求的不断增加, 软件系统变得日趋庞大和难以驾驭, 缺陷和漏洞难以避免, 系统越来越脆弱, 很多时候不以人们期望的方式工作. 软件并不总是让人信任的, 从而软件可信性问题越来越得到人们的关注 [1]. 软件的可信性是在软件的正确性、可靠性、安全性等众多属性基础上发展起来的一个新概念. 而软件的正确性是软件可信研究中一个重要的内容. 软件的正确性是指软件执行结果能否符合人们的预期.

随着各种计算机系统设计复杂性的提高, 软件设计的正确性已成为问题的关键. 目前在系统的开发过程中, 软件的验证在整个系统的开发时间和经费上所占的比重越来越大, 而在设计阶段的后期才发现软件的错误会造成时间和经费的巨大浪费. 由于软件故障造成的重大事件层出不穷, 例如:

(1) 奔腾 586 芯片中的除法错误是在大量进入市场后被一位用户发现的, 虽然这种错误发生的概率为几亿分之一, 但这个设计错误给 Intel 公司造成了 4.8 亿美元的经济损失.

(2) 1990 年 1 月 15 日, 由于软件问题, 美国长途电话中断 9 小时.

(3) 1991 年 2 月海湾战争期间, 又是软件问题, “爱国者” 未能成功拦截 “飞毛腿” 导弹, 造成 28 人被炸死.

(4) 1992 年 6 月 16 日, 利用阿利亚娜火箭发射的欧洲遥感卫星 1 号, 由于星载软件故障, 导致轨道上的卫星与地面失去联系.

(5) 1996 年 6 月, 阿里亚娜运载火箭爆炸, 导致直接经济损失 5 亿美元, 原因是某些参数值超过预期值, 数据发生了溢出.

(6) 1999 年, 美国国家宇航局 (NASA) 的火星气象卫星失踪, 原因是软件系统在英制单位和公制单位转换上出了问题.

保证软件正确性的途径是多方面的. 首先对软件的需求分析要全面而准确, 不能有任何实质性的遗漏, 从而产生正确的软件规格说明 (specification), 也即软件的规范, 建立软件准确的模型. 目前确认软件实现正确与否的主要途径是通过反复的模拟.

形式化方法 (formal methods) 提供了保证软件设计正确性的一条重要途径. 它用数学方法表达软件的规范或软件的性质, 并根据数学理论来证明所设计的软件满足软件的规范或所期望的性质. 在不能证明所期望的性质时, 则可能发现设计错误. 在软件的设计中, 从软件的设计规格说明 (软件规范) 到软件的最后实现, 在设计的每个阶段, 都用形式化方法对该设计阶段的设计进行验证, 尽可能地在每个设计阶段发现错误并及时更正, 避免将设计错误传播到下一个设计阶段. 实践证明, 形式化方法确实能通过形式规格说明和证明, 增强对软件的理解从而发现设计错误. 形式化方法可以根据它们的形式化程度、具体的策略等进行分类, 形式化程度的加强可以使得规格说明和假设减少对主观因素的依赖性. 其中一类形式化方法是使用具有严格语义定义以及相应机械化、形式化验证工具支持的规格说明语言. 状态遍历、模型检查技术在这类方法中得到广泛的使用. 这类方法具有最高的形式化程度, 并且使用更加复杂的规格说明语言来对软件的动态属性进行描述, 同时提供了相应的验证方法, 其中一个重要的验证方法是模拟. 在这种方法中, 对软件模型和软件的规格说明均用状态转换系统表示. 直觉上讲, 给定一个行为模型 A 和一个规格说明 B, 要判定 A 是否模拟了 B, 要看 A 的每一个行为是否同时也是 B 的一个行为. 若答案是肯定的, 同时 B 的每一个行为也是 A 的一个行为, 这种模型 A 与模型 B 之间的关系称为互模拟. 若一个行为模型和其规格说明之间是互模拟的, 则行为模型满足了规格说明, 同时规格说明是行为模型的规格说明, 此时这个行为模型是完全正确的.

在抽象层次上, 可以把软件执行看作软件实现, 把人们的预期抽象为规格说明即软件规范, 这样软件的正确性可以用软件实现与其规范之间的关系来表示 [2]. 进程代数理论中的一个重要的任务是验证一个系统的实际行为能否与其

想要的行为相一致, 即验证一个系统的实现是否满足其规范. 因此我们可以借助于进程代数理论中的各种等价关系来验证软件的正确性. 当然这种验证需要在一定的标准下进行, 从而需要确定两个进程的行为是否一致的标准. 这些标准构成了进程代数理论的语义. 根据不同的应用需要, 有不同的语义模型, 例如迹语义 (trace semantics)[3,4]、失败语义 (failure semantics)[5~10]、模拟语义 (simulation semantics)[11,12]、预备模拟语义 (ready simulation semantics) [13] 和互模拟语义 (bisimulation semantics)[11,14~16] 等. 为了能够更好地通过计算机来确定两个进程的行为是否一致, 已经出现了很多逻辑系统、算法和工具 [10,17~25]. 同时在一些复杂的系统中, 交互和并发是非常必要的, 为了讨论这种复杂系统的行为特征, 也发展了很多进程代数理论, 如英国学者 R. Milner 和 C. A. R Hoare 分别提出了通信系统演算 (CCS)[14,26~29] 和通信顺序进程 (CSP)[3~5]. 这两种代数理论都使用通信作为进程之间相互作用的基本手段, 表现出面向分布式系统的特征. 在语法上, 进程代数用一组算子作为进程的构件. 算子的语义通常用结构化操作语义方法定义, 这样进程就可看成是带标号的转换系统. 标号迁移系统中的结点被解释为进程的可能状态, 标号迁移系统中的一元关系被解释为状态性质, 标号迁移系统中的二元关系被解释为进程可能执行的原子行为. CSP 是描述交互式并发的原始模型, 为程序设计者提供了很好的帮助. 而 CCS 更加侧重于从数学的角度来建立并发理论的语义描述, 其中最重要的内容是提出了互模拟的概念, 互模拟及其变体可被看作标号迁移系统上的等价关系, 两个互模拟的标号迁移系统描述了相同的进程. 若系统的实现与其规范之间存在互模拟等价或观测等价关系, 则在一定程度上认为这个系统是正确的. 基于 CCS 中的数学特征, 本书主要在 CCS 框架下进行讨论, 在讨论中对软件与进程不加以区分.

然而软件的实现都是在一定物理设备 (如计算机) 上运行的, 而物理设备不能保证一定是完全可靠的, 这样软件实现很难完全满足其规范, 在很大程度上一个软件实现是其规范的近似实现, 而近似程度直接影响着软件的正确程度, 这就产生了软件的近似正确问题. 应明生教授指出软件近似正确性问题至少可以分为两种: 动态近似正确性和静态近似正确性. 其中动态近似正确性主要表现为软件实现越来越满足规范, 静态近似正确性体现了软件实现与规范之间存在一定的误差.

假设我们将设计一个复杂的系统 $S = f(R_1, \cdots, R_n)$, 这个系统由一些子系统 $R_1, \cdots, R_n$ 组成. 由于资源的有限性等原因, 我们只能一步一步地设计这个系统. 自然的, 我们希望通过模块化或结构化设计方法来设计这个系统. 第一步我们可以开发一个系统 $S_1 = f(R_{11}, \cdots, R_{1n})$, 然后逐步修改 $R_{1i}(1 \leqslant i \leqslant n)$, 获得

一个序列 $\{R_{mi}:m=1,2,\cdots\}$, 其中 R_{mi} 是对 R_{1i} 经过 m 次修改后得到的子系统. 这样整个的初始系统 S_1 也逐步地被修改了, 由此我们得到了相应的修改序列 $S_m:m=1,2,\cdots$, 且对每一个 $m=1,2,\cdots$, $S_m=f(R_{m1},\cdots,R_{mn})$. 由此一个非常重要的问题需要讨论: 对每一个 $i\leqslant n$, 当 $\{R_{mi}:m=1,2,\cdots\}$ 收敛于 R_i 时, $\{S_m:m=1,2,\cdots\}$ 是否收敛于 S? 换句话说 f 是否是连续的? 在进程演算中, f 通常由各种各样的算子组成, 因此为了说明程序进化的复合性特征, 我们需要考察这些算子是否是连续的. 这个问题将在本书中给予一些研究. 众所周知, 极限在程序语言的语义方面具有非常重要的意义. 特别的, 极限经常被用来描述语义论域中递归方程的解. 互模拟极限在进程演算的理论方面是非常重要的. 应明生教授在其著作 [2] 中描述了规范和实现之间的收敛关系. 系统 S 表示系统的规范,$\{S_m:m=1,2,\cdots\}$ 是其一系列修改的实现, 若这些实现从某次修改后所得实现与规范之间都是互模拟关系, 则不需要再对实现进行修改, 因此 S 是 $\{S_m:m=1,2,\cdots\}$ 的极限. 同时还证明了 f 在各种算子下的连续性. 这种实现与规范之间的动态近似关系表现为系统的动态近似正确性.

在进程演算中, 一个进程的行为由一些基本的动作所组成. 例如一个卖可口可乐的自动售货机, 若想从机器上买一杯可口可乐, 先要投币 1 美元, 然后选择 "可口可乐" 按钮, 最后从出口处取出. 假设这个售货机用 C 表示, 利用进程演算中的通信系统演算 (CCS) 语言可以把这一过程表示为 C=1d.CocaCola.Collect.C. 下面假设有另外两个自动售货机, 分别出售芬达和肥皂, 用 F,W 表示为: F=1d.Fanta.Collect.F,W=1d.Soap.Collect.W. 如果一个人真正的是想买一杯可口可乐, 但是附近只有卖芬达和肥皂的机器, 则这个人将买一杯芬达的可能性比买一块肥皂的可能性要大. 再例如, 又有两个卖可口可乐的自动售货机 C^+ 和 C^-, 假设其定义为 C^+= 1.5d.CocaCola.Collect.C^+, C^-=0.8d.CocaCola.Collect.C^-. 如果一个人想用 0.8 美元买一杯可口可乐, 且身边恰好没有机器 C^-, 则这个人从机器 C 买可口可乐的可能性比从 C^+ 上买可口可乐的可能性大. 为什么第一个人买芬达, 第二个人买 1 美元的可口可乐呢? 直觉告诉我们,F 比 W 更加接近于 C, C 比 C^+ 更加接近于 C^-. 然而这种表述不是很规范, 我们需要建立一套严格的数学方法来描述它. 我们可以定义一个度量 ρ_1, 其中 ρ_1(CocaCola, Fanta)=1,ρ_1(CocaCola,Soap)=ρ_1(Fanta,Soap)=$+\infty$; 定义另一个度量 ρ_2, 其中 ρ_2(0.8d, 1d)=0.2, ρ_2(0.8d, 1.5d)=0.7 且 ρ_2(1d,1.5d)=0.5. 如果动作上的度量 ρ_1 和 ρ_2 能诱导一个进程上的度量 ρ, 使得 $\rho(C,F)=1$, $\rho(C,W)=\rho(F,W)=+\infty$,$\rho(C^-,C)=0.2$,$\rho(C^-,C^+)=0.7$ 且 $\rho(C,C^+)=0.5$, 则

$\rho(C,F)<\rho(C,W)$, 且 $\rho(C^-,C)<\rho(C^-,C^+)$. 这样我们就定义了一种非常严格的数学证明.

从这个例子中我们注意到, 当我们考察进程之间的行为时, 这些行为之间的关系不能被忽略. 特别是当把系统设计的规范和实现都用进程来描述时, 它们的行为之间可能会存在一定的近似关系.

从软件实现来看, 软件的近似正确问题可以分为动态和静态两类. 动态近似体现在人们可以不断地修改实现使其越来越满足规范, 在修改过程中得到实现的一个不断进化的序列, 在这种情况下, 实现序列的最终目的是完全满足其规范, 进而认为软件是越来越接近于正确. 静态近似表现为给定的软件实现与规范之间存在一定的误差, 在这种情况下, 给定的实现可以看作是规范的近似实现, 进而认为软件是近似正确的. 从而我们注意到一个软件的运行依赖于环境, 有些软件在一些环境下运行是正确的, 而在另一些环境下运行未必是正确的, 所以在考虑软件的正确性时需要考虑环境的因素. CCS 模型所提供的强互模拟等价、弱互模拟等价以及观测等价等没有反映环境对软件正确性的影响, 所以我们需要在 CCS 模型中寻找一种体现环境的进程之间关系的特性来验证软件的正确性.

1.2 研究现状

软件的动态近似性主要体现在实现在不断修改过程中越来越接近于其规范, 在不断修改的过程中得到实现的一个进化序列, 更加一般的情况是这些实现构成了偏序. 例如, 一些软件需要很多团队一起合作, 在某一时刻, 可能有几个团队同时修改了实现, 由此修改过程中所得到的实现不再形成一个序列, 而是一个偏序结构.

在文献 [2, 30, 31] 中, 作者基于 CCS 模型建立了进程的无限演化理论以及拓扑理论. 在无限演化理论中, 借助于拓扑学中的极限 [32,33] 概念, 作者建立了一个系统规范是其实现的极限形式, 其主要基于强互模拟等价、弱互模拟等价以及迹等价三种规范与实现之间的抽象来刻画系统的动态近似. 在文献中作者定义了强互模拟极限、弱互模拟极限和迹极限, 给出了这三种极限的拓扑结构. 然而基于软件自身的特点, 在讨论软件实现与其规范之间的关系时需要考虑环境的影响,

文献 [2] 中的强互模拟极限、弱互模拟极限和迹极限没有反映出环境的因素, 所以我们需要在 CCS 模型中寻找一种与环境有关的等价关系来抽象基于环境的软件实现与其规范之间的关系. 幸运的是, K. G. Larsen 在文献 [34~37] 中提出的参数化互模拟以及在文献 [38~40] 中提出的三分之二互模拟符合我们的要求.

K. G. Larsen 于 1986 年在其博士论文中首次提出了参数化互模拟等价, 其目的是为了能够更好地使用逐步加细的观点来证明系统的规范与其实现之间的关系. 对于一个给定的规范, 一般人们希望能够通过对规范的逐步加细来获得实现, 而每个加细步骤都是由更加小的加细组成, 如图 1.1 所示: P 对于实现来说太抽象了, 所以用更加具体的 Q 来代替. 但是, 在代替过程中其他方面不应该受到影响, 所以需要证明在任何上下文环境 $\mathcal{C}$ 下都有 $\mathcal{C}[P]=\mathcal{C}[Q]$. 然而直接证明和通过同余 (congruence) 的证明都存在一定的难度, 为了能够更好地证明规范的加细在任何上下文环境 $\mathcal{C}$ 下等价, K. G. Larsen 提出了参数化互模拟等价, 以一些和上下文 $\mathcal{C}$ 有关的信息为参数, 一般化了强互模拟等价.

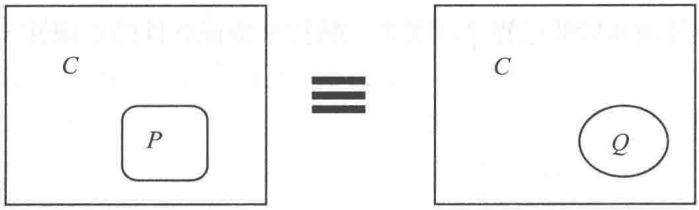


图 1.1 加细过程

由于软件实现满足其规范也需要在任意上下文环境中保持, 更加抽象的, 可以把软件的一些物理设备或者软件支持等环境抽象为实现与其规范的上下文环境, 这样我们可以通过参数化互模拟等价来验证在环境影响下软件实现是否满足其规范. 在参数化互模拟等价中, 把与上下文有关的信息抽象为一个标号转换系统 (labeled transition system), 这个转换系统显式地给出了环境之间的转换关系, 规范和实现都抽象为进程. 若环境在一定标号下允许一个转换, 则规范和实现在该标号下也要有相应的转换, 此时规范所要求做的事情实现能够完成, 同时实现完成的事情都是规范所要求的, 即在此标号下, 实现与规范之间是互模拟的. 若规范或实现有一个标号转换, 但环境在此标号下没有转换, 则规范或实现在此标号下的转换将不被允许, 也就是说, 规范或实现所能做的事情都是环境所允许的. 这样, 当把环境看作一个标号转换系统时, 参数化互模拟体现了软件在接受环境下的正确性. 当我们选择了参数化互模拟等价来验证一个软件在环境影响下的正确

性时, 从动态近似方面来看需要建立实现在逐渐修改过程中能够越来越满足其规范的形式化描述. 为此在本文的第 5 章中建立参数化互模拟的无限演化理论, 提出参数化互模拟极限的概念, 并构造参数化互模拟拓扑.

K. G. Larsen 于 1991 年首先在概率转换系统的基础上提出了三分之二互模拟 [39], 目的是为了刻画两个概率进程对于所有测试拥有相同的可观测集合时, 这两个进程是不可区分的. 在文献 [41~43] 中, R. J.van Glabbeek 指出: K. G. Larsen 的三分之二互模拟等价和失败模拟等价是一致的. 在文献 [40] 中, 何积丰基于 CCS 模型考虑了三分之二互模拟, 目的是证明模拟等价、相互加细和测试等价是一致的. 三分之二互模拟是进程之间的一个二元关系且满足: 如果一个进程 P 能执行一个可观测动作, 则另一个进程 Q 也能执行此动作, 并且如果 P 拒绝一些动作, 则 Q 也拒绝这些动作. 如果我们把环境看成是一些动作的集合, 软件实现和规范都看作进程, 则三分之二互模拟从拒绝环境的角度体现了规范与实现之间的关系. 在实际应用中, 对一些特别的用户来说, 使用进程的互模拟来描述软件的正确性可能太严格了, 这样需要寻找一种较宽松的规范与实现之间的关系来验证正确性. 由于三分之二互模拟中只要求规范所要求做的事情实现都能够完成, 而规范拒绝的事情, 实现也能够拒绝, 所以三分之二互模拟不是真正的互模拟, 而是一种比互模拟宽松的互模拟, 因此我们可以选择三分之二互模拟作为验证软件实现与其规范之间关系的标准. 同样, 当选择这个标准来验证软件的正确性时, 从动态近似方面也需要建立软件实现在逐渐修改过程中能够越来越满足其规范的形式化描述. 为此在本文的第 3 章中建立三分之二互模拟的无限演化理论以及三分之二互模拟极限, 并构造三分之二互模拟拓扑.

软件的静态近似性表现为软件实现与其规范之间在一定程度上允许存在一定的误差, 这个误差反映了规范与实现之间的近似程度, 而这种近似程度可以通过建立进程之间的度量关系来刻画. 对进程的度量模型已经出现了很多理论. 文献 [2, 44, 45] 中, 在预先给定的动作之间度量的基础上, 根据两个进程所执行动作之间的距离建立了强互模拟索引和弱互模拟索引. 这些索引刻画了进程集合上的一个二元关系在多大程度上是强互模拟和弱互模拟的, 进一步, 作者根据互模拟索引定义了 λ- 强 (弱) 互模拟. 若在两个进程之间存在一个 λ- 强 (弱) 互模拟关系, 则它们至少在 "1-λ" 程度上是近似的. 这种 λ- 强 (弱) 互模拟关系在一定程度上反映出两个进程的近似程度. Smolka 等在文献 [46, 47] 中基于概率转换系统, 根据确定性概率进程之间的概率关系, 提出了近似度为 "1-ϵ" 的 ϵ- 互模拟等价, 并在此基础上建立了两个概率进程之间近似程度的度量模型. 这个度量模型

是以两个概率进程执行相同动作概率的差距而定义的. 文献 [48] 在概率 CSP 模型基础上, 根据两个概率进程执行相同迹的概率以及一个折扣因子建立了两个概率进程之间的度量. 这个折扣因子反映了当它们执行相同迹的深度越大时, 它们之间的距离就越小的特性. 在文献 [49] 中, 作者根据进程的可观测行为建立了进程之间的量化关系. 在文献 [50] 中, 作者基于量化标号转换系统 (action-labeled quantitative transition system) 定义了系统中两个状态之间的度量. 文献 [51] 中在标号马尔科夫概率进程 [52~54] 模型基础上建立了进程之间的度量模型.

然而由于软件的运行依赖于环境, 在建立软件实现与其规范之间的近似度量模型时, 需要考虑环境的因素. 以上描述的进程之间的度量关系或者概率进程之间的度量关系没有反映出环境的影响. 为了刻画在环境影响下的软件实现与其规范之间的近似程度, 我们需要寻找一种与环境有关的度量模型. 由于三分之二互模拟从拒绝环境的角度体现了软件实现与其规范之间的关系, 所以在本文的第 4 章将以三分之二互模拟为基础, 建立软件实现与其规范之间近似程度的度量模型.

另一方面, 在一些实际应用中, 由于软件系统的复杂性越来越高, 软件中经常出现一些概率信息需要描述, 如一个通信协议丢失信息的概率为 0.02. 为了描述这些带有概率信息的系统, 已经发展了很多的理论. 例如文献 [55 ~ 60] 中提出了各种概率进程模型. 文献 [55, 61] 中对 SCCS 模型进行了概率化, 在进程和算子上增加了概率信息, 提出了三种概率进程的模型, 同时还建立了相应的互模拟等价关系. 文献 [62] 中将概率引入 CCS 模型, 不仅在和算子上增加了概率信息, 同时对并发算子也增加了概率信息, 所得到的模型称为可和的概率模型 (additive model of probabilistic processes), 同时也建立了进程之间的互模拟等价关系. 文献 [48] 中以概率 CSP [63,64] 为基础建立了有限概率进程的迹等价语义. 伴随着概率进程代数理论的建立, 概率进程的公理化表示及其语义解释也发展起来, 验证概率互模拟等价的算法及其实现的工具也不断出现 [65~71]. 基于测试的观点, 描述这种带有概率信息的复杂进程与环境的交互也有了很多成果, 如文献 [72, 73] 等. 然而基于环境影响, 如何建立软件实现与其规范之间的近似正确性需要进一步进行研究. 为此在本文的第 6、7 章中分别建立了在接受环境下软件实现与规范之间近似程度的度量描述; 同时在近似概率互模拟基础上, 建立了带有概率信息的软件系统实现逐渐接近于其规范的动态描述及其拓扑结构.

1.3 内容安排

本书作者自 2004 年师从于陈仪香教授攻读硕士和博士学位以来, 系统地研究了在环境基础上的软件近似正确性模型. 本书基于作者的博士论文, 依据最新研究成果, 系统地介绍了基于环境的软件近似正确性理论. 其内容是这样安排的: 在第 1 章中从软件正确性的定义出发, 介绍软件正确性的意义以及当前的研究现状. 在第 2 章中从通信系统演算 (CCS) 的基本概念入手, 介绍了 CCS 的语法和语义、参数化互模拟和三分之二模拟的基本概念以及在研究动态和静态近似正确性时用到的定向集及一些相关概念和性质. 这些内容是本书的基础. 在第 3 章中首先介绍了三分之二互模拟的背景, 然后建立了三分之二互模拟的无限演化机制, 并给出了三分之二互模拟极限以及三分之二互模拟的拓扑理论. 第 4 章中利用动作集合上的度量理论, 对三分之二互模拟做近似化, 建立了 λ- 三分之二互模拟, 利用模态逻辑语言对其进行刻画; 同时根据近似的三分之二互模拟给出了进程之间近似程度的量化模型, 证明了量化模型在各种复合算子下的可替换性. 第 5 章首先介绍了在参数化互模拟基础上研究动态近似正确性的背景, 然后介绍了参数化互模拟的无限演化机制, 并在此基础上给出了参数化互模拟极限的刻画, 最后建立了参数化互模拟的拓扑理论. 在第 6 章中研究了近似的参数化互模拟. 第 7 章对参数化互模拟进行了概率化. 第 8 章对概率进程代数中的近似互模拟建立了其无限演化理论和拓扑结构.