国外经典教材·计算机科学与技术

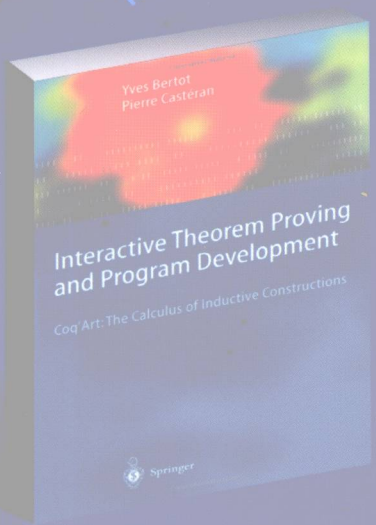
Springer

交互式定理证明与程序开发

Coq归纳构造演算的艺术

Interactive Theorem Proving and Program Development

Coq'Art: The Calculus of Inductive Constructions



Yves Bertot
Pierre Casteran 著

顾 明 等译

清华大学出版社

国外经典教材·计算机科学与技术

- 过程感知的信息系统
- 软件测试：跨越整个软件开发生
- 软件测试实践：成为一个高效能
- 机器视觉算法与应用
- 数值方法(C++描述)
- Web技术
- UML安全系统开发

内容简介

Coq是一个用于验证定理的证明是否正确的计算机工具。在推理和编程方面，Coq的语言都拥有足够强大的能力和表达能力，可以构造简单的项，执行简单的证明，直到建立完整的理论，学习复杂的算法。

本书的主要目标是从实践的角度来理解Coq系统及其基本理论，即归纳构造演算。这本书给出了大量的例子，所有这些例子都可以在计算机上执行。从本书配套网站www.labri.fr/Perso/~casteran/CoqArt可以下载并执行所有证明的例子，而且还提供了书中200个练习的答案。

这本书是一本很有价值的教材，它为初学者提供基础训练，为有经验的人提供必要的专业知识，帮助学习者开发有实用价值的数学证明。

本书的配套网站
www.labri.fr/Perso/~casteran/CoqArt

ISBN 978-7-302-20813-6
9 787302 208136
定价:59.00元
Springer

国外经典教材 • 计算机科学与技术

Interactive Theorem Proving and Program Development

Coq'Art: The Calculus of Inductive Constructions

交互式定理证明与程序开发

Coq 归纳构造演算的艺术

Yves Bertot

Pierre Casteran 著

顾 明 等译

清华大学出版社

北 京

版权登记号：图字 01-2009-3553 号

图书在版编目（CIP）数据

交互式定理证明与程序开发——Coq 归纳构造演算的艺术 /（德）伯托特（Bertot, Y.），（德）卡斯特兰（Casteran, P.）著；顾明等译. —北京：清华大学出版社，2010.1
书名原文：Interactive Theorem Proving and Program Development Coq'Art: The Calculus of Inductive Constructions
（国外经典教材·计算机科学与技术）
ISBN 978-7-302-20813-6

Ⅰ. 交… Ⅱ. ①伯… ②卡… ③顾… Ⅲ. 定理证明—软件工具，Coq—教材 Ⅳ. 0141-39

中国版本图书馆 CIP 数据核字（2009）第 156693 号

责任编辑：龙啟铭
责任校对：徐俊伟
责任印制：李红英

出版发行：清华大学出版社　　**地　　址**：北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　　编**：100084
社　总　机：010-62770175　　**邮　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质　量　反　馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：北京市清华园胶印厂
装 订 者：三河市新茂装订有限公司
经　　销：全国新华书店
开　　本：185×260　　**印　张**：28.5　　**字　数**：690 千字
版　　次：2010 年 1 月第 1 版　　**印　次**：2010 年 1 月第 1 次印刷
印　　数：1～2500
定　　价：59.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题，请与清华大学出版社出版部联系调换。
联系电话：010-62770177 转 3103　　产品编号：029058-01

译者序

定理证明是数学领域中一个古老的分支，它从公理出发，利用推理规则为定理寻找证明过程。但是，当我们把数学定理的手工证明和日常生活中的演绎推理变成一系列能在计算机上自动进行的符号演算的过程和技术时，定理证明就成为当今软件工程领域中一种非常重要的形式化技术，即定理证明系统。今天，定理证明系统已经被广泛应用于数学定理证明、协议验证以及软硬件的安全特性验证等方面，成为人们解决关键软件系统正确性、可信性的重要方法，也是继模型检测技术之后未来软件工程领域的一个重要发展方向。

为了帮助我国广大科技工作者和学生更好地学习、掌握和应用定理证明系统，我们CEREUS 小组翻译了《交互式定理证明和程序开发——*Coq* 的艺术: 归纳构造演算》一书。选择翻译这本书主要基于如下几点考虑：

1. *Coq* 是目前国际上交互式定理证明领域的主流工具，它基于归纳构造演算，有着强大的数学模型基础和很好的扩展性，并有完整的工具集。
2. *Coq* 有一支强大的全职研发队伍，支持开源，这对于想学习和使用该系统的读者非常有益。
3. 本书的作者一直从事 *Coq* 的研发，在书中提供了大量的示例和习题，可以帮助读者更快地掌握 *Coq*，并理解 *Coq* 背后的基础理论。

翻译从来就不是一件轻松的事，尤其是这种既有很深的理论高度，又有很强的实践要求的书，翻译难度就更大了。CEREUS 小组能够在一年内完成本书的翻译工作，不仅饱含了小组全体成员的辛勤劳动，也得到了国际学术界许多研究人员的帮助。参与本书翻译工作的是清华大学软件学院 CEREUS 小组，其中第 1～8 章由刘柳、周旻、张连怡负责；第 9 章由王瑞负责；第 10 章和第 12 章由张荷花负责；第 11 章、第 13～14 章由万海负责；第 15～16 章由陈钢负责；顾明、陈钢、宋晓宇、荔建琦负责全书的校稿工作。此外，在翻译这本书的过程中，我们得到了本书作者的全力支持，澳门科技大学张昱，中国科技大学“中科大-耶鲁”高可信软件联合研究中心的郭宇、李兆鹏、李勇、王僖和庄重，伦敦大学皇家霍洛威学院的冯扬悦，INRIA的罗正钦等做了本书的审稿工作，在此一并表示感谢。

我们希望这本书作为我国研究定理证明系统的一个新起点，能推动中国定理证明系统的发展，推动我国软件工程的健康发展。我们也希望读者在阅读本书的过程中，能够给我们多提意见。我们的联系方式是：cereus@tsinghua.edu.cn。

最后，我们要感谢国家自然科学基金委，本书的翻译工作得到“可信软件基础研究”重大研究计划的支持。

前言

Coq 是一个用于验证定理的证明是否正确的计算机工具。这些定理可能涉及到普通数学、证明理论或程序验证。

我们的主要目标是从实践的角度来理解 *Coq* 系统及其基本理论：归纳构造演算（the Calculus of Inductive Constructions）。因此，这本书中包含了大量的例子，所有这些例子都可以在计算机上执行。为了教学目的，一些例子解释了错误或笨拙的用法以及避免这些问题的准则。我们也尽量分解对话（dialogues）以便读者能够通过纸笔或直接在 *Coq* 上对其进行重现。有时，我们会给出一些综合表达式；它们乍看起来让人生畏，但事实上也是在 *Coq* 证明辅助工具的帮助下得到的。读者应该在试验时对它们进行分解、修改、了解其结构，并获得一种实际的感受。

本书有一个相关网站[1]，读者可以从该网站下载并执行所有证明的例子。我们也提供了书中 200 个练习的答案，以备不时之需。这本书及其网站使用的工具都是 2004 年初发布的*Coq V8*[2]。

用户对 *Coq* 中已证明的定理的信心来自于构造演算（Calculus of Inductive Constructions）的性质。构造演算是一个形式系统。以 λ 演算和类型（typing）的观点来看，它结合了逻辑中的一些最新进展。这个演算的主要性质已在此处给出，因为我们相信结合理论和实践的知识是使用 *Coq* 全部表达能力的一条最好的路径。

在推理和编程方面，*Coq* 的语言都拥有足够强大的能力和表达能力。从构造简单的项，执行简单的证明，到建立完整的理论，学习复杂的算法，对读者能力有着不同层次的需求。按照所需的能力层次，我们对章节进行了标注：

（没有标注） 初次阅读即可理解，
* 中等程度的实践者可以阅读，
** 有能力掌握复杂的推理和证明程序者可读，
*** 为有兴趣探索 *Coq* 形式系统所有可能性的专家预留。

练习也有着相同的标注，从基础的练习（可以在几分钟内解决）到非常困难的练习（可能需要几天的思考）。大多数的练习都是我们研究工作中遇到问题的简化版本。

在这本书的编写期间，许多人都给了我们热情的帮助。尤其要感谢 Laurence Rideau，她总是非常友好地为我们提供帮助，并且认真地阅读了从最初草稿到最终版本中的每一个版本。Gérard Huet 和 Janet Bertot 在帮助我们改进技术准确性和写作风格

[1] `www.labri.fr/Perso/~casteran/CoqArt/`。
[2] `coq.inria.fr`。

方面也投入了大量的时间和精力。另外，我们还要特别感谢 Gérard Huet 和 Christine Paulin-Mohring 为这本书撰写前言。

感谢整个 *Coq* 开发小组，研制了这么强大的工具。特别是要感谢：Christine Paulin-Mohring、Jean-Christophe Filliatre、Eduardo Gimenez、Jacek Chrząszcz 和 Pierre Letouzey，在归纳类型的内在一致性、命令式程序表达、co-归纳类型、模块、抽取方面给我们提供的宝贵见解，此外，他们还为本书编写了几页内容和一些例子。除此以外，Hugo Herbelin 和 Bruno Barras 和我们一起合作，帮助确保这本书中描述的所有例子都能被实现。

这里，还要感谢我们教过和一起工作过的学生，在与他们共同进行实验的过程中，我们的知识领域也得到了增长。尤其要指出的是，在里昂高师和波尔多第一大学执教时，和 Davy Rouillard，Antonia Balaa，Nicolas Magaud，Kuntal Das Barman 和 Guillaume Dufay 等合作研究解决一些问题后，才逐渐理解本书中描述的一些观点。

许多学生和研究人员花费了时间来阅读本书的初稿，并把本书作为教学资料使用。他们给我们提供了改进意见，并给出了一些候选的解决方法。我们想在这里感谢那些为我们宝贵的反馈意见的朋友，他们是：Hugo Herbelin、Jean-François Monin、Jean Duprat Philippe Narbel、Laurent Théry、Gilles Kahn、David Pichardie、Jan Cederquist、Frédérique Guilhot、James McKinna、Iris Loeb、Milad Niqui、Julien Narboux、Solange Coupet-Grimal、Sébastien Hinderer、Areski Nait-Abdallah、Simão Melo de Sousa。

除了以上的朋友外，我们各自的科研环境发挥了关键作用，感谢他们的支持让这个项目通过。特别要感谢 INRIA 的 Lemme 与 Signes 小组和波尔多第一大学的支持，以及欧洲的 Types 工作组为我们提供机会，让我们能与富有创新精神的年轻研究人员，如 Ana Bove，Venanzio Capretta，Conor McBride 见面交流，本书中一些例子的灵感就来源于他们。

非常感谢 Springer-Verlag 出版社的工作人员，他们的帮助使这本书最终能够完成，尤其是 Ingeborg Mayer、Alfred Hofman、Ronan Nugent、Nicolas Puech、Petra Treiber 和 Frank Holzwarth。他们的鼓励，以及在内容、编排、编辑和排版等方面的建议是必不可少的。此外，还要感谢 KünkelLopka GmbH 的 Julia Merz 所设计的艺术品般的封面。

Sophia Antipolis
Talence

Yves Bertot
Pierre Castéran

序言

Don Knuth 为充实计算机科学基础而写下数卷程序设计名著时，并没有把他的著作名选为《计算机程序设计科学》，而是叫做《计算机程序设计艺术》。此后，经过 30 余年的研究，程序设计和算法才成为一门严格的科学。类似地，在形式化证明设计领域，目前正在构建一个严格的基础。尽管证明论的主要概念可以追溯到 20 世纪 30 年代的 Gentzen、Gödel 和 Herbrand，图灵本人就已表现出对自动构造数学证明的兴趣。然而，直到 1960 年才首次出现进行一阶逻辑自动证明的实验，即系统地枚举 Herbrand 域。40 年之后，*Coq* 证明环境已成为计算逻辑方面一系列探索中的一个最新成果，在某种意义上，它代表了这一领域的当今水平。然而，*Coq* 的实际使用依然处于一种艺术状态，难以掌握、难以改进。Yves Bertot 和 Pierre Castéran 的这本书是一本很有价值的教材，它为初学者提供基础训练，为有经验的人提供必要的专业知识，帮助学习者开发有实用价值的数学证明。

一个简短的关于 *Coq* 系统历史的介绍将帮助读者学习这个软件以及它所实现的数学概念。关于基础概念起源的知识将有助于读者了解用户必须掌握的工作机制，构造系统模型时的各种观点，以及在出现问题时的各种可能的处理方案。

Gérard Huet 在 1970 年开始在自动定理证明方面进行工作，他使用 LISP 语言实现了带等式的一阶逻辑证明器 SAM。当时的研究水平只是把所有的逻辑命题翻译成由文字表（带符号原子公式的析取）组成的表（合取式），量化则由 Skolem 函数代替。在这样的表示方法之下，推理过程被归结为基于例化的互补原子公式配对原理（通常称为合一消解原理），等式则产生出相对于合一（modulo unification）的单方向重写。重写的顺序由人为方法决定，既不保证收敛性，也不保证完备性。证明器是黑箱，它们产生出大量的不可读的逻辑推理结论。当时的常见情形是输入一个假设，然后是等待，直到存储空间用尽。只有在罕见的简单情形下，证明器才会给出答案。这一灾难性状况当时并没有得到充分的理解，人们把它看成是不完全性定理带来的恶魔。然而，复杂性研究很快显示出，哪怕是在可判定的领域，比如命题演算，自动定理证明也注定要撞上组合爆炸的墙。

一个决定性的突破出现在 20 世纪 70 年代，那是一个系统化方法的实现，它以终结序指导重写。这一进展的基础是 Knuth 和 Bendix 的奠基性研究工作。Jean-Marie Hullot 和 Gérard Huet 在 1980 年完成了一个 KB 软件，它以一种自然的方式实现代数结构的自动判定过程和半可判定过程。同时，归纳证明领域也取得了稳步的进展，最著名的工作是 Boyer 和 Moore 的 NQTHM/ACL 系统。另一项有重要意义的进展是把消解技术推广到高阶逻辑，方法是使用 Gérard Huet 在 1972 年所设计的基于简单类型理

论的合一算法。该算法同 Gordon Plotkin 独立研究的一个方程理论上的通用合一技术本质上一致。

同时，逻辑学家（Dana Scott）和理论计算机科学家（Gordon Plotkin、Gilles Kahn、Gérard Berry）正在研究可计算函数的一种逻辑理论（可计算论域），以及有效可用公理化（可计算归纳），目的是为了定义程序语言的语义。当时人们希望用这个理论严格地处理可信软件的设计问题，这样的设计将采用形式化方法。一个程序相对于它的逻辑规范的正确性可以用数学理论中的定理来表达，算法的数据和控制结构在数学理论中进行描述。在爱丁堡大学，Robin Milner 领导的小组在这方面做出了引人注目的工作。他们的一项重要的成就是在 1980 年实现了 LCF 系统。该系统颇具智慧地引入了用于辅助证明的策略，后者可以用元语言 ML 进行编程。公式不再被归结到无法理解的子句，用户可以使用他们的直觉和知识指导系统进行证明，自动证明（预定义的证明策略和用户使用 ML 语言编写的特定证明策略的结合）和手工证明混合在一起。

哲学家 Per MartinLöf 探索了另一条研究路线。这一方向从 Brower 开创数学的构造性基础开始，经由 Bishop 的构造性分析而扩展深化。以此为基础，Per MartinLöf 在 80 年代设计了直觉主义类型理论，为数学结构的构造性公理化提供了一个优美的通用框架，而且适合于用作函数式程序设计的基础。康乃尔大学的 Bob Constable 教授认真地继续了这一方向的研究，他实现了 Nuprl 软件，用于从形式证明中进行软件设计。同时，在 Gothenburg 的 Chalmers 大学的 Brengt Nordström 领导的“程序设计方法组”也进行了类似的研究。

所有这些研究都依赖于最初由逻辑学家 Alongzo Church 所设计的 λ 演算记号，这一演算的纯粹形式相当于一个用于定义递归泛函的语言，它的带类型的版本相当于高阶谓词演算（即简单类型理论，它是最初在 Whitehead 和 Russell 的《**数学原理**》中提出的元数学系统的一个简单变种）。更进一步，λ 演算可用于表示自然推理中的证明，由此产生了著名的“Curry-Howard 对应关系”，它表示了证明结构的空间与函数空间的同构。λ 演算的这两个方面实际上已被用于 Automath 系统，该系统是在 1970 年由 Eindhoven 大学的 Niklaus de Bruijn 所设计，目的是为了实现数学的表示。在这个系统中，λ 表达式的类型不再是函数空间中的简单的层次结构。实际上，它们能够表达函数的输入变元同函数的输出结果之间的依赖关系。这可以类比于命题演算扩展到一阶谓词演算，在后一种情况下，谓词的输入项是它的定义域中的元素。

λ 演算的确是证明论中的主要工具。在 1970 年，Jean-Yves Girard 证明了分析的一致性，他的证明使用了称为 System-F 的多态 λ 演算中证明的终止性。这一系统被推广到一个表示多态泛函的 $F\omega$ 系统，因而可为一类超出了传统序数层次的算法进行编码。1974 年，John Reynolds 在推广 ML 语言的受限制的多态结构时，又重新发现了这一系统。

20 世纪 80 年代早期，在逻辑和计算机科学的前沿，类型理论获得长足进展。在 1982 年，Gérard Huet 联合巴黎高等师范学院的 Guy Cousineau 和 Pierre-Louis Curien 在 INRIA Rocquencourt 实验室启动了 Formel 项目。这个小组在 LCF 系统的启发之下，准备设计和开发一个更强的证明系统，尤其重要的是，他们准备把 ML 语言不仅用于定义 tactics，同时用于实现整个系统。这一项在函数式方面的研究和开发工作在

几年后产生了 CAML 语言族，最终导致了今天的 Objective Caml 语言，它就是现在的 *Coq* 证明器的实现语言。

1984 年，Gilles Kahn 在 Sophia Antipolis 组织了一个类型理论的国际会议，在会上，Thierry Coquand 和 Gérard Huet 展示了一个把依赖类型和多态类型综合在一起的系统，它把 MartinLöf 的构造性理论融入了 Automath 系统的一个扩展，该系统命名为**构造演算**（Calculus of Constructions）。Thierry Coquand 在博士论文中提供了对这一系统的 λ 演算基础的元理论分析。他给出了这一演算终止性的证明，进而给出了关于该演算的逻辑可靠性的证明。这一演算就成了 Formel 项目的证明系统的逻辑基础，Gérard Huet 对这个演算 CoC 做出了第一个验证器，称为**“构造引擎”**（Constructive Engine）。1985 年 4 月，在 Eurocal 会议上演示了在这个验证器上进行的几个形式化数学的开发工作。

这就是 *Coq* 系统的第一阶段：一个 λ 表达式的类型验证器，在这个系统中，λ 表达式表示逻辑系统的证明项，或者是数学对象的定义。这个证明助手的核心是与证明综合工具完全独立的，后者的用途是构造需要验证的项，构造引擎的解释器是一个确定性的程序。Thierry Coquand 实现了序列（Sequent）风格的证明综合算法，它提供了一组类似 LCF 系统的证明策略，支持逐步求精方式构造证明项。*Coq* 第二阶段的开发由 Christine Mohring 完成，在 *Coq* 系统中首次实现 *Prolog* 风格的证明搜索，即著名的 `Auto` tactic。这可以看成是今天的 *Coq* 系统的正式诞生。在现在的版本中，*Coq* 核心依然重新检查用户调用 tactic 所构造的证明项。这一架构有一个附加的优点，即简化了证明搜索的机制，它实际上忽略了类型系统分层所要求的某些约束。

Formel 组很快意识到构造演算可用于综合出带有证书的程序（certified program），这一机制与 Nuprl 系统的做法相似。一个关键点是利用了多态类型的优势，把一个代数结构用 *F* 系统的类型表示出来，比如整数，这里系统性地利用了 Böhm 和 Berarducci 所提出的方法。Christine Mohring 专注于这一问题，她实现了一个复杂的 tactic，在构造演算中综合出归纳原理。以此为基础，她在 1986 年 6 月举行的“计算机科学中的逻辑”（LICS）会议上展示了一种开发带证明算法的形式化方法。然而，当她完成了博士论文之后，她意识到她所使用的 impredicative 编码并不遵从常规模式，即把归纳类型的项限制在类型构造子的复合。多态 λ 演算的编码引入了寄生项，因而不能表示合适的归纳原理。这个部分的失败实际上给了 Christine Mohring 和 Thierry Coquand 一个动力，促使他们在 1988 年设计了“归纳构造演算”，这是原来系统的一个扩展，增加了归纳数据类型上的算法的公理化的一些良好的性质。

Formel 小组在理论研究和系统实验两方面始终保持着小心的平衡。他们用模型来评价各种建议的可行性，用原型来验证系统的能力是否可以扩展到处理实际的证明，他们提供免费的配备了良好的库和手册的越来越完整的系统，并努力在各版本之间维持兼容性。这个小组开发的 CoC 原型系统转变成 *Coq* 系统，通过网上论坛发布的方式把该软件提供给感兴趣的用户。同时，基础问题的研究也从不忽略。比如 Gilles Dowek 系统性地研究了合一理论和类型论中的证明搜索，这为以后的 *Coq* 发展提供了坚实的基础。

1989年，*Coq* 4.1版本发布，该版首次加入了由Benjamin Werner所设计的从证明中抽取函数式程序（Caml语法）的机制。此外，还有一组新的进行自动证明的tactics，以及一个小的数学和计算机科学方面的知识库。这一进展标志着一个新阶段（new era）的开始。Thierry Coquand在Gothenburg获得了一个教学的位置，Christine Paulin-Mohring加入了位于里昂的高等师范学院。此后，*Coq* 组的工作就在里昂和Rocquencourt两地继续进行。同时，一个称为Cristal的新项目开始了，它的主要课题是围绕函数式语言ML展开研究工作。在Rocquencourt，刚在NuPRL组完成了经典逻辑的构造性解释的博士论文的Chet Murthy为 *Coq* 带来了他的新贡献，在 *Coq* 5.8版本中引入了更复杂的体系结构。在欧洲支持的一项基础研究国际项目“逻辑框架”（Logical Framework）下开始了国际合作研究，三年之后，又在“类型”项目之下继续进行。几个小组同时在称为“证明助手”（Proof assistant）的证明工具方面进行开发，他们之间相互激励，分享经验。*Coq* 是这些证明工具中的一个。其他的“证明助手”有：爱丁堡大学的Randy Pollack开发的LEGO系统，剑桥大学的Larry Paulson开发的Isabelle系统，该系统后来由慕尼黑大学的Tobias Nipkow继续，此外，还有Gothenburg小组开发的Alf系统等等。

1991年推出的 *Coq* 5.6版提供了进行数学描述的统一语言（Gallina“vernacular”），原始归纳类型，从证明中抽取程序的机制，和一个图形化用户界面。这使 *Coq* 成为一个可以有效使用的系统，并由此开始了同工业界的有成效的合作，尤其是同CNET和Dassault-Aviation的合作。由于出现了第一批学术界之外的用户，促使 *Coq* 组写出了一个教学讲义和使用手册，此时 *Coq* 的使用艺术对于新手依然神秘。*Coq* 依旧是一个研究性探索的工具和展开实验的场所。在Sophia Antipolis，Yves Bertot在原来的Centaur项目基础上开发了CTCoq界面中的结构操作，使该系统能够利用鼠标进行交互式证明构造，这一技术称为“Proof-by-pointing”，即用户通过鼠标点击来调用tactics。在里昂，Catherine Parent的博士论文研究了证明中抽取程序的问题，并把该问题转换成另一个问题，把加入不变式注解的程序作为它们自身正确性证明的框架。在波尔多，Pierre Castéran的工作显示这一技术可被用于构造具有continuation语义风格的带证明（certified）算法库。在里昂，Eduardo Giménez在他的博士论文中表明，能够以继承性（hereditarily）方式定义有限结构的归纳类型的框架可以被扩展到一个co-inductive类型的框架，后者可用于对无限结构公理化。作为一个推论，他开发了涉及数据流操作的通讯协议的证明，这样就为通讯方面的应用打开了一条路。

在Rocquencourt，Samuel Boutin在他的博士论文中研究了 *Coq* 中自反推理的实现，这一技术在基于代数重写的自动证明中有重要的应用。他的 *Ring* tactic可用于简化多项式表达式，从而隐式地实现常用的算术表达式代数操作。另外一些判定过程也显著地改进了 *Coq* 系统的自动证明能力：比如在Presburger算术中使用的 *Omega*（由CNET-Lannion的Pierre Crégut开发），在命题演算中使用的 *Tauto* 和 *Intuition*（由Rocquencourt的César Muñoz开发）。没有收缩规则（contraction）的谓词演算中使用的Linear（由里昂的Jean-Christophe Filliâtre开发）。Amokrane Saïbi引入了表达继承和隐式强制（coercion）的子类型的概念，这些概念可用于在广义代数中进行模块化证明开发，尤其是可以简练地表达范畴论的主要概念。

1996 年 11 月发布的 *Coq* 6.1 版引入了所有上述理论成就，也包括了几项对提高效率有关键作用的技术，特别是 Bruno Barras 的规约机制，Christina Cornes 的处理归纳定义的高级证明子。还有 Yann Coscoy 的把证明翻译成自然语言（英语和法语）的翻译器，它把证明子构造的证明项用可读的方式表达出来。这是同类证明系统还没有具备的一项重要功能，它使我们能够对形式证明进行检查。

在程序验证的领域中，J.-C. Filliâtre 在他 1999 年的论文中展示了怎样在 *Coq* 中进行命令式程序的证明。他重新采用了 Floyd-Hoare-Dijkstra 倡导的在命令式语言中使用断言的方法，命令式程序被看作是从它们的指称语义所获得的函数表达式所对应的记号。*Coq* 系统是一个两级架构，核心是 CoC 验证器。通过在 *Coq* 中对构造演算的元理论进行形式化，可以从验证算法的正确性证明中抽取出验证器，这一点也显示了进行两级架构划分的意义。这是一项出色的技术成果，它在形式化方法的安全性方面迈出了一大步。为了在数学方面做开发工作，Judicaël Courant 在 Objective Caml 的模块机制的启发之下，勾画了一个模块语言的基础，这也为库的可重用性和大规模软件验证开辟了道路。

新成立的 Trusted Logic 公司使用 Caml 组和 *Coq* 组的技术进行智能卡系统的正确性验证。这也表明了 *Coq* 研究的价值。其他应用项目也纷纷开始。

以后的 *Coq* 系统经历了完全的重新设计，版本7拥有一个函数式核心，主要架构师是 Jean-Christophe Filliätre，Hugo Herbelin 和 Bruno Barras。一个用于 tactics 设计的新语言由 David Delahaye 开发出来，此后人们可以用高级语言为复杂的证明策略编程。为了对数值软件进行正确性验证，Micaela Mayero 研究了实数的公理化问题。同时，Yves Bertot 重新利用 CtCoq 的思想，用 Java 语言开发了一个复杂的图形界面 PCoq。

2002 年，也就是 Judicaël Courant 完成论文之后的第四年，Jacek Chrząszcz 采用了类似 Caml 的方法把模块和函子系统整合在一起。作为理论开发环境中的一个平滑的结合，这一扩展显著地改进了库的通用性。Pieere Letouzey 提供了从证明中提取程序的一个新算法，它把整个 *Coq* 语言都考虑了进去，包括模块系统。

在应用方面，*Coq* 已经足够强壮，并可用作实现特定证明工具的低层语言，比如用于时间自动机模拟和验证的 CALIFE 平台，用于命令式程序证明的 *Why* 工具，在欧洲项目 VERIFICARD 中开发的用于 Java 小应用程序（Java applet）验证的 *Krakatoa* 工具。这些工具使用 *Coq* 语言描述和证明模型的特性，尤其是在自动工具不能处理的复杂情形，这些工具显得很有用。

在经过三年的努力之后，Trusted Logic 成功地完成 JavaCard 语言的整个执行环境的形式化模拟。这项安全方面的工作获得 EAL7 证书奖（公共标准下最高级别的奖励）。这一形式开发工作使用了 121000 行 *Coq* 代码，总共 278 个模块。

Coq 也被用于开发先进的数学定理库，包括构造性数学和经典数学。在构造性数学领域中工作需要对 *Coq* 的逻辑语言进行限制，以便同数学家常用的公理保持逻辑上的和谐一致。

在 2003 年底，在经过对主要的输入语言进行重新设计之后发布了 8.0 版本，这就是本书中采用的版本。

浏览一下 *Coq* 用户群所作的贡献的目录（地址：http://coq.inria.fr/contribs/summary.html），读者将清楚地看到在 *Coq* 中已经完成的数学开发工作的丰富性。开发组遵循 Boyer 和 Moore 提出的要求，在相继发布的 *Coq* 版本之间保持库的兼容性。在必要的时候，提供一些工具把旧的证明脚本自动转换成新的证明脚本，以此确保用户的开发工作不会因新版本的出现而过时。许多这样的库是由 *Coq* 组外的研究人员所开发的，有的在国外，有的在工业界。我们羡慕这个用户群体对 *Coq* 的坚持，他们完成了非常复杂的证明开发。直到今天，使用 *Coq* 总是带有一定的实验性质，尤其是缺乏一个足够全面细致，逐步深入的用户指导书。

有了本书，这一需求现在得到了满足。Yves Bertot 和 Pierre Castéran 多年以来就是 *Coq* 各个版本的专家级用户。他们是 *Coq* 开发组之外的“客户”，正因为如此，他们并不回避 *Coq* 中的潜在问题。他们也不会宣传一个尚不成熟的的解决方案。他们的所有例子都可以在当前的版本中进行验证。他们所写的这本书以渐进的方式介绍了 *Coq* 系统的所有功能。这一陈述详尽的著作所付出的代价是它的厚度。希望初学者不会因此而后退，书中关于内容难度的指示可以帮助他们在学习时作出适合自己的选择，他们不需要从第一页读到最后一页。这本书可以作为一本参考书来使用。用户可以在长期使用 *Coq* 的过程中，在遇到新困难时来查找解决方案。本书包含了大量精心选择循序渐进的例子，这也是此书比较厚的原因。读者常常愿意自己一步步重做一遍这些例子。事实上，我们也建议读者在读这本书的时候要同时使用一台装有 *Coq* 证明器的计算机，一边看书一边操作书中的例子。这本书所展示的是经过 30 年形式化方法研究所积累的成果，该领域的内在难度是不能忽略的，要成为使用 *Coq* 的专家就不得不付出一定的代价。这本书经过了三年的编写和修改，这一过程促使了 *Coq* 中的记号统一化，对证明工具的作用做出更简明的解释，并整理出让非专家也能够理解的出错信息。自然，我们承认以后还会有需要改进之处。愿读者在这个即困难又令人兴奋的世界里的探索获得成功。愿他们的努力能够在完成最后一个 QED 时得到满足，这可能需要数周甚至数月的艰苦工作，能够到达终点的人会得到应有的收获。

Gérard Huet, Christine Paulin-Mohring

目录