软件学报 ISSN 1000-9825, CODEN RUXUEW
2026,37(9):3457−3490 [doi: 10.13328/j.cnki.jos.007604] [CSTR: 32375.14.jos.007604]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

数学分析机械化工程 I: 一元微积分形式化系统*

窦国威[1,2], 郁文生[1,2]

[1](北京邮电大学 电子工程学院, 北京 100876)

[2](天地互联与融合北京市重点实验室 (北京邮电大学), 北京 100876)

通信作者: 郁文生, E-mail: wsyu@bupt.edu.cn

摘　要: 形式化数学是一次数学革命, 结合定理证明器的数学定理机器证明, 不仅是对数学严谨性的一种新标准, 更是发展数学的一种新方式. 随着世界范围不断有数学难题在计算机辅助下的成功解决, 以及专家学者对各种数学形式化项目或工程的发起, 形式化数学的影响力与日俱增, 在数学界与计算机界引起广泛影响. 介绍一项基于定理证明工具 Coq 的数学分析形式化系统, 该系统以华东师范大学数学系编著的《数学分析》为蓝本, 在朴素集合论和初等数论及代数知识体系下进行开发. 当前, 已经实现其上册中一元微积分相关内容的形式化, 包括实数与函数、数列极限、函数极限、函数的连续性、导数和微分、不定积分、定积分等内容. 该系统严格对应教材内容, 全部定理无例外地给出 Coq 的机器证明代码, 所有形式化过程已被 Coq 验证, 并在计算机上运行通过. 读者可以跟随代码学习数学, 也能够对照数学理解代码, 充分体现了基于 Coq 的数学定理机器证明具有可读性、交互性和智能性的特点, 实现让读者跟随计算机学习、理解、构建、教育乃至发展现代数学的尝试, 提高认识数学、感受数学和欣赏数学的素养.

关键词: 形式化数学; 定理机器证明; Coq; 数学分析; 一元微积分

中图法分类号: TP311

中文引用格式: 窦国威, 郁文生. 数学分析机械化工程I: 一元微积分形式化系统. 软件学报, 2026, 37(9): 3457–3490. http://www.jos.org.cn/1000-9825/7604.htm

英文引用格式: Dou GW, Yu WS. Mechanizing Mathematical Analysis I: Formal System of Single-variable Calculus. Ruan Jian Xue Bao/Journal of Software, 2026, 37(9): 3457–3490 (in Chinese). http://www.jos.org.cn/1000-9825/7604.htm

Mechanizing Mathematical Analysis I: Formal System of Single-variable Calculus

DOU Guo-Wei[1,2], YU Wen-Sheng[1,2]

[1](School of Electronic Engineering, Beijing University of Posts and Telecommunications, Beijing 100876, China)

[2](Beijing Key Laboratory of Space-ground Interconnection and Convergence, Beijing University of Posts and Telecommunications, Beijing 100876, China)

Abstract: Formalized mathematics represents a revolution in mathematics. The combination of theorem provers with machine verification of mathematical theorems establishes not only a new standard for mathematical rigor but also a novel approach to developing mathematics. As mathematical challenges worldwide are increasingly solved with computer assistance and various formalization projects are launched by experts, the influence of formalized mathematics continues to grow. It generated significant impact across both the mathematical and computer science communities. This study introduces a formal system for mathematical analysis based on the Coq theorem prover. The formalization is guided by the textbook Mathematical Analysis compiled by the School of Mathematical Sciences at East China Normal University. Developed within the framework of naive set theory and elementary number theory and algebra, the system has formalized the

* 基金项目: 国家自然科学基金 (62476028)
本文由“形式化方法与应用”专题特约编辑安杰副研究员、顾斌研究员、李建文教授推荐.
收稿时间: 2025-09-07; 修改时间: 2025-10-28; 采用时间: 2025-12-08; jos 在线出版时间: 2025-12-24
CNKI 网络首发时间: 2026-06-04

content related to single-variable calculus from the first volume of the textbook. It includes topics such as real numbers and functions, sequence limits, function limits, continuity of functions, derivatives and differentials, indefinite integrals, and definite integrals. The proposed system strictly corresponds to the textbook content, where all theorems are provided with machine-verifiable Coq proofs. The entire formalization is verified by Coq and executed successfully on a computer. Readers can learn mathematics by following the code and can also understand the code by comparing it with the mathematics, which demonstrates the readability, interactivity, and intelligence of Coq-based machine theorem proving. It represents an attempt to enable readers to follow the computer in learning, understanding, constructing, educating, and even developing modern mathematics, thereby enhancing their ability to understand, experience, and appreciate mathematics.

Key words: formalized mathematics; machine-assisted theorem proving; Coq; mathematical analysis; single-variable calculus

2025 年 5 月, 国际著名华裔数学家陶哲轩 (Terence Tao) 发起了一项名为“A Lean companion to ‘Analysis I’”的形式化数学项目[1], 该项目旨在利用定理证明器 Lean4[2], 形式化地重塑其本人于 20 年前撰写的经典数学分析教材《Analysis I》[3]中的内容, 在数学界与计算机界引起了广泛影响. 数学理论的形式化, 特别是基于教材的数学形式化, 不仅是对数学严谨性的极高保障与精益求精, 更是辅助理解学习数学乃至发展数学的强大支撑与有效途径.

关于形式化数学, 在 20 世纪初对于数学基础的深入讨论中受到重视, 对整个 20 世纪数学的发展产生了深远的影响[4–12]. 进入 21 世纪, 计算机形式化辅助证明工具 (定理证明器) 的出现, 例如 Coq[13,14]、Isabelle[15,16]、Mizar[17]及 Lean[2]等, 使得结合定理证明器的形式化证明技术在学术界得到极大重视. 需要说明的是, 自 2025 年 3 月, Coq 已正式更名为 Rocq, 详情可见 https://rocq-prover.org/changelog, 但根据习惯, 本文仍沿用“Coq”的称呼. 近年来, 随着机器学习理论以及各种人工智能大模型 (如 ChatGPT、DeepSeek 等) 的发展与涌现, 将定理证明器与机器学习理论以及大模型相结合的数学形式化辅助证明技术也成为学术界讨论的热点.

形式化数学的发展如火如荼, 这也正印证了著名数学家和计算机专家 Wiedijk 认为“形式化数学是一次数学革命”的观点, 他在文献 [12] 中指出: “数学历史上发生过 3 次革命. 第 1 次是公元前 3 世纪, 古希腊数学家 Euclid《几何原本》引入数学证明方法; 第 2 次是 19 世纪 Cauchy 等人引入‘严格’数学方法, 以及后来的数理逻辑和集合论; 第 3 次就是当前正在进行的形式化数学.” 此外, 1998 年菲尔兹奖获得者 Gowers[18]、2002 年菲尔兹奖获得者 Voevodsky[19]、2006 年菲尔兹奖获得者 Tao[10]、2010 年菲尔兹奖获得者 Villani[20]、2018 年菲尔兹奖获得者 Scholze 以及 1987 年沃尔夫奖和 2005 年阿贝尔奖获得者 Lax 都大力倡导发展形式化数学[21,22]. Gowers[18]指出: “21 世纪计算机在证明定理的过程中会起到巨大作用, 理论数学研究的模式将会彻底改观, 计算机的作用有可能超出我们现在的想象”, 甚至预测“2099 年之前, 电脑或可完成所有重要的数学. 电脑会提出猜想、找到证明. 而数学家的工作, 是试着去理解和运用其中的一些结果”. 文献 [22] 中介绍, Lax 认为“计算机对于数学的影响可以与望远镜对天文学和显微镜对生物学的影响相比拟”. Tao[10]则致力于将形式化数学与人工智能大模型相结合, 让模型以经过证明器检验的形式化语言作为输入和输出, 以解决大模型在数学内容生成时出现的“幻觉”问题, 甚至借助大模型提出和解决数学猜想与难题. 在文献 [5] 中, 计算机与形式化理论专家 Avigad 和 Harrison 指出“在计算机证明辅助工具的帮助下, 形式化数学或将成为数学严谨性的新标准.”

当前, 已有许多重要数学结论借助定理证明器得到严格形式化证明, 包括素数定理[23]、四色定理[24]、约当曲线定理[25]、哥德尔不完备性定理[26]、不动点定理[27]、中心极限定理[28]、奇阶定理[29]、开普勒猜想[30]等. 这些著名成果使得结合形式化证明辅助工具的数学理论研究方法备受瞩目.

除了以理论研究为导向的应用, 数学形式化还可以成为辅助知识理解、学习乃至教育的有效途径, 无论是在计算机科学还是数学领域. 例如, Hendriks 等人[31]基于 Coq 开发了 ProofWeb 系统, 用于本科计算机专业逻辑学的教学. Avigad[32]在卡内基梅隆大学开设了一门课程, 使用定理证明器 Lean 帮助学生阅读、编写和理解数学证明. 根据文献 [21,33] 介绍, 约翰霍普金斯大学也曾开设类似课程, 让学生使用定理证明器辅助学习数学. 文献 [33] 提及有学者设想未来定理证明器可以帮助数学期刊审稿人理解数学证明, 甚至代替审稿人的工作, “如果你想让你的论文被接受, 你必须让它通过定理证明器的检查.” 国内也有研究团队基于 Coq 开发了针对教学场景的 ZFC 集合

论形式化系统, 通过教学实践, 明显提升了学生理解相关定理并自主编写证明过程的能力[34].

最近 Tao[1]发起的“A Lean companion to ‘Analysis I’”, 为数学教材开发配套的形式化系统, 则又是一个在数学分析课程中具有潜在教学应用价值的启发性项目, 人们可以跟随代码理解和学习课本中的数学证明, 也可以参考人工证明自主编写形式化代码, 以检验是否真正掌握相关内容.

本文介绍一个配套国内经典本科数学教材——华东师范大学《数学分析》[35]的形式化系统, 该系统使用定理证明器 Coq, 并以集合论为数学基础进行开发, 现阶段已基本实现教材上册中一元微积分相关内容的形式化, 包括极限理论、函数连续性、导数和微分、不定积分和黎曼积分等重要概念, 所有相关定义和定理均与教材内容严格对应, 并给出严格的 Coq 形式化描述和证明. 同时, 利用 Coq 中的 Notation 机制, 我们在系统中尽可能引入了常用数学符号, 使得代码简洁易读. 图 1 给出一个定理描述及部分证明的示例: 若函数 f 在 x_0 处连续, g 在 u_0 处连续, 且 $u_0 = f(x_0)$, 则复合函数 $g \circ f$ 在 x_0 处连续. 相信即便是没有系统性学习过 Coq 开发的读者也能够读懂图中绿色部分形式化定理所描述的内容.

```
(* 复合函数的连续性 *)
Theorem Theorem4_5 : ∀ f g x0 u0, Continuous f x0 -> Continuous g u0
  -> u0 = f[x0] -> Continuous (g ∘ f) x0.
Proof.
  intros. red in H. destruct H as [H H']. red in H0. destruct H0 as [H0 H0'].
  destruct H0' as [H0' [δ' [I1[I2 I3]]]], H' as [H' [δ'' [J1[J2 J3]]]].
  repeat split.
  - rewrite (Comp_Fun_P2 g f); auto. apply Classifier1. split; auto.
    apply Classifier1. rewrite <- H1; auto.
  - apply Comp_Fun_P1; auto.
  - assert (x0 ∈ dom[g ∘ f]).
    { rewrite (Comp_Fun_P2 g f); auto. apply Classifier1; split; auto.
```

图 1　复合函数连续性的描述及部分证明

我们主要介绍该形式化系统的组织结构、数学内容及其具体实现, 期望能为读者提供一个与教材配套的学习或教学工具. 利用本系统, 学习者可以对照课本, 交互地查看 Coq 形式化中的所有证明细节, 从而更好地理解和学习相关数学内容; 教学人员也可以尝试将本系统用于教学实践, 以检验学生是否真正理解和掌握数学理论. 该系统的完整形式化代码可从以下链接获取: https://github.com/1DGW/a-formal-system-of-mathematical-analysis.

本文第 1 节介绍背景知识. 第 2 节介绍本文形式化系统组织结构. 第 3 节介绍系统的逻辑与集合论基础. 第 4 节介绍系统所依赖的实数理论基础. 第 5 节和第 6 节分别介绍微分和积分部分相关概念的形式化描述. 第 7 节介绍几个重要定理的具体形式化证明. 最后总结全文.

1　研究背景

1.1　数学分析形式化现状

数学分析是使用场景最为广泛的数学基础理论, 因此许多定理证明器的官方库业已涵盖其大部分内容的形式化[36], 例如 Coq 标准库[37]的 Reals.Ranalysis 模块, Isabelle/HOL 库[38]中的 Analysis 部分, 以及基于 Lean 开发的大型数学库 Mathlib[39]中 Analysis 目录下的相关内容. 一些独立开发的著名形式化项目, 例如基于 Coq 开发的 Coqlicot[40]和 C-CorN[41], 也包含了传统数学分析的内容[36].

以上相关工作尽管早已展开, 但相关应用大多集中于形式化项目本身的开发, 或结合人工智能大模型的研发等工作, 少有紧密贴合教材的开发与应用. 例如, Coqlicot 项目是对 Coq 标准库的兼容性优化与补充, 以简化公式与定理的形式化书写[40]; C-CorN 是对一个名为“FTA-project”前期项目的扩展, 以检验该前期项目的可用性[41]; Mathlib 则与 DeepSeek 相结合, 形成专为形式化数学定理证明设计的大语言模型 DeepSeek-Prover-V2[42]. 此外, 大多形式化项目完全基于类型论开发, 更具有“计算机风格”, 对于纯数学的学习和研究者不够友好.

本文所开发的形式化系统以教材对应为导向, 使用者可以一边与代码交互, 一边对照课本内容理解证明细节, 并且以集合论作为数学基础, 注重推理过程而非类型计算, 便于数学使用者的理解, 同时, 我们在系统中尽可能地

引入了与数学符号一致的形式化记号, 增强代码的可读性.

1.2 定理证明器 Coq

Coq 是一个交互式的定理证明器与程序开发系统平台, 其基本原理是归纳构造演算, 是一个用于描述定理内容、验证定理证明的计算机工具, 这些定理可能涉及普通数学、证明理论或程序验证等多个方面[13,14,43]. Coq 采用高阶逻辑, 允许对性质和命题进行量化, 在推理和编程方面都拥有足够强大的表达能力, 从构造简单的项, 执行简单的证明, 直至建立完整的理论.

在定理证明方面, 用户可通过命令式的证明策略 (tactic) 进行逻辑推导, 能够实时观察当前的证明状态, 包括已知条件和证明目标, 以人机对话的方式一问一答, 边设计、边修改, 使证明中的每一处细节得到验证, 并且可以自行编写证明策略, 利用已证命题进行自动推理.

图 2 展示了 Windows 系统中 CoqIDE (Coq 集成开发环境) 的使用界面, 整个窗口分为 3 部分, 其中左半部分用于代码的编写, 包括定义的描述和定理证明, 绿色表明代码运行通过, 红色则表示代码运行出错; 右上部分显示当前的证明环境, 可以实时查看已有条件和待证目标, 当所有目标被解决时, 即意味着当前定理证明完成; 右下部分为信息栏, 可以显示运行错误原因、定理查询结果等关键信息. 如此, 使用者便可交互地查看和编写证明细节.

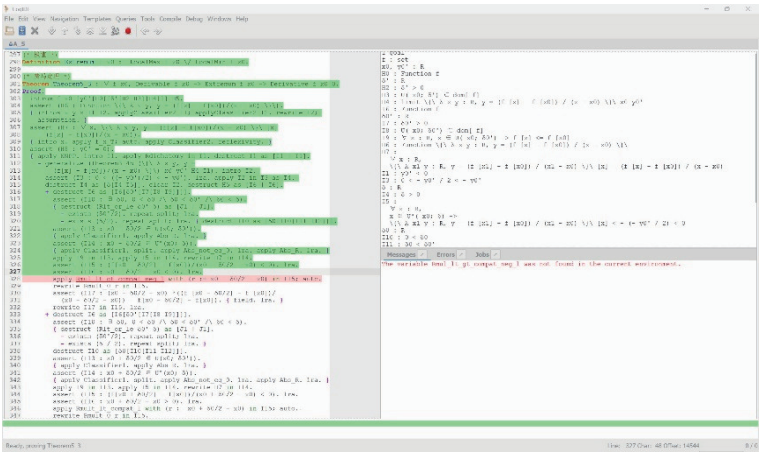

图 2 CoqIDE 交互式界面

此外, Coq 由法国计算机与自动化研究院 (INRIA) 的团队开发和管理, 技术成熟且完全开源, 是一个开放且使用友好的交互式定理证明器.

2 系统组织结构

《数学分析》(后文称为教材)[35]主要由 11 章组成, 其中第 7 章实数的完备性实际上是对实数理论的深入探讨, 其内容相对独立, 本文暂不做详细介绍, 相关内容可参考我们先前关于实数理论的形式化系统开发[44–46]; 第 10 章主要为一些例题; 第 11 章反常积分是对定积分与极限的综合扩展, 尚处于开发阶段. 除去以上内容, 其余 8 章可分为 4 个部分: (1) 第 1 章实数与函数基础; (2) 第 2–4 章极限与函数连续性; (3) 第 5、6 章导数与微分; (4) 第 8、9 章不定积分与定积分.

我们开发的形式化系统主要包含 15 个 Coq 的 v 文件, 图 3 给出这些文件与教材及本文章节的对应关系. 图中箭头表示文件之间的依赖关系; 文件 A_0 为整个系统的逻辑和集合论基础, 尽管教材中无实际章节对应, 但对于逻辑自洽性和数学结构的严密性来说至关重要; 教材第 9 章因内容丰富, 按其子章节对应, 共由 6 个文件组成, 其中教材第 9.3 章可积准则的证明实际上使用了第 9.6 章关于可积性理论的补充内容, 因此文件 A_9_3 需依赖于第 9.6 章对应文件 A_9_6, 而第 9.2 章内容相对独立, 直到第 9.5 章中的相关证明才有所依赖; 其余每章均对应一

个同名 v 文件.

此外, 对系统中定义和定理的命名规则做简要约定. 定义主要按照相关英文名称进行命名, 例如, 极限的定义以“limit”命名, 导数以“Derivative”命名; 引理、定理、推论等以数字或字母编号进行命名, 例如教材中定理 4.9 以“Theorem4_9”命名, 定理 9.3 与 9.3′分别以“Theorem9_3a”与“Theorem9_3b”命名, 定理 6.5 的第 3 个引理以“Lemma6_5c”命名等, 以此可与教材中的定理在形式与内容上均形成良好对应. 在大部分定义与定理形式化描述前都写有注释, 可以帮助使用者理解相关内容.

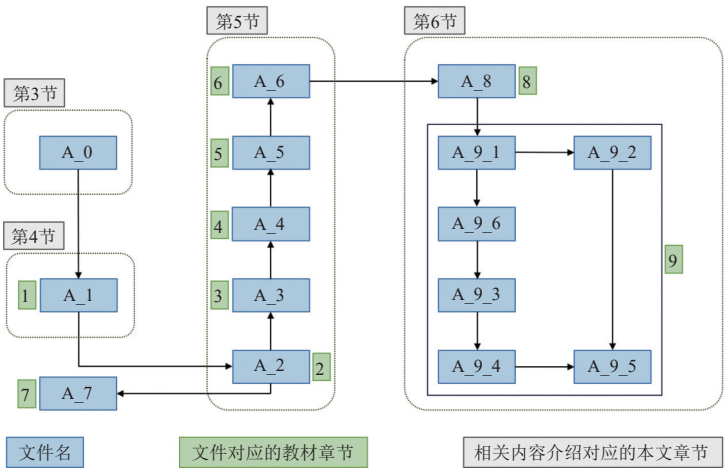

图 3 一元微积分形式化系统结构组织图

3 逻辑与集合论基础的形式化

3.1 Coq 形式化语言和逻辑基础

数学定理的形式化大致可以分为两部分, 其一是对公理、定义、定理及命题等的形式化描述, 其二是利用相应的证明策略构造证明过程、消解证明目标, 从而实现对所描述内容的证明. 其中第 1 步, 即形式化描述, 本质上是一种从人工语言到形式化语言的“翻译”, 而确保这种翻译的正确性则是整个形式化过程中的关键部分. 因此, 首先介绍本文系统中用于描述公理、定义和定理的 Coq 命令及其使用规则.

Axiom 用于公理的形式化描述, 其声明的内容无需通过证明策略进行验证. 下面以排中律为例说明.

公理 1. 排中律. 对于任意命题 P, 只存在 P 是真或 P 是假两种情况.

排中律在分析学中默认成立, 因此在一般数学分析相关书籍中不会特别说明. 然而, Coq 本身建立在直觉逻辑上, 并未承认排中律, 这意味着所有证明必须找到具体构造, 反证法在该逻辑下无法适用. 反证法对于数学证明的重要性不言而喻, 正如 Hilbert[47]所说: “禁止数学家使用排中律就像禁止天文学家使用望远镜及拳击手使用自己的拳头一样.” 因此必须加入排中律, 将 Coq 的内部逻辑扩展至可以使用反证法进行推理的古典逻辑. 排中律的形式化描述为:

```
Axiom classic : ∀ (P : Prop), P \/ ~P.
```

这里使用 Axiom 命令写入一个名为 classic 的公理, Prop 是 Coq 中表示命题的类型, 通常可由 Coq 自身的类型推断功能自动判断, 可以省略. 符号“∀”表示任意量词, “\ /”表示逻辑连词“或”, “~”表示逻辑符号“非”. 可见, Coq 中的一些逻辑符号与数学常用符号一致, 能够使得形式化代码简洁可读.

与 Axiom 作用类似的还有 Parameter 命令, 二者在代码功能上相同, 习惯上后者主要用于不加证明地承认和声明全局常量.

关于逻辑符号, 除了以上排中律使用到的, 表 1 列出了所有本文中常用的逻辑符号.

表 1 Coq 常用逻辑符号及其含义

Coq符号	数学符号	逻辑含义	Coq符号	数学符号	逻辑含义
->	⇒	蕴含	/\	∧	且
<->	⇔	等价	\/	∨	或
∀	∀	任意量词	~	¬	非
∃	∃	存在量词	—	—	—

Theorem 用于定理的形式化描述, 其声明的内容需要使用证明策略进行严格验证. 基于排中律, 本文系统所依赖的所有逻辑命题均可在这里给出:

```
Theorem NNPP : ∀ P, ~~P <-> P.
Proof. split; intros; destruct (classic P); tauto. Qed.
Theorem not_and_or : ∀ P Q, (~(P /\ Q) <-> (~P) \/ (~Q)) /\ (~(P \/ Q) <-> (~P) /\ (~Q)).
Proof. intros; destruct (classic P); tauto. Qed.
Theorem not_all_ex_not : ∀ U (P : U -> Prop), ~ (∀ n, P n) -> (∃ n, ~ P n).
Proof. intros. apply NNPP; intro. elim H. intros. apply NNPP; intro. elim H0; eauto. Qed.
Theorem not_ex_all_not : ∀ U (P : U -> Prop), ~ (∀ n, P n) -> ( ∀ n, ~ P n).
Proof. intros. intro. elim H; eauto. Qed.
Theorem imply_to_or : ∀ P Q, (P -> Q) -> (~ P) \/ Q.
Proof. intros. destruct (classic P); auto. Qed.
Theorem imply_to_and : ∀ P Q, ~ (P -> Q) -> P /\ ~ Q.
Proof. intros. destruct (classic P). split; auto. elim H. intros. elim H0; auto. Qed.
```

与公理描述类似, 以上定理中的冒号前蓝色字体可视为定理名称, 冒号之后为定理内容. 其中“`P : U -> Prop`”表示 P 为一个带有参数的命题, 其余形式化描述的含义不言自明. 此外, 由 Proof 和 Qed 所包裹的内容即为证明策略, 其具体应用和交互方法将在第 7 节详细说明. 以下至第 6 节内容中, 我们重点介绍形式化系统与教材[35]中内容的对应, 暂不对证明策略进行展开.

其他与 Theorem 作用类似的命令还有: Lemma (引理)、Corollary (推论)、Fact (事实)、Property (性质)、Proposition (命题), 它们的使用规则完全一致, 所陈述的内容均需要严格验证, 区别仅在于语义和可读性上的不同, 可以根据具体数学含义互换使用.

Definition 用于给出定义的形式化描述, 在本文系统中, 大部分定义均以该命令给出形式化描述. 下面以函数连续性定义的形式化为例给予说明.

```
Definition Continuous f x0 := x0 ∈ dom[f] /\ limit f x0 f[x0].
```

该形式化定义中, Continuous 为定义名称, f 和 x_0 为定义所需参数, 与公理 (Axiom) 和定理 (Theorem) 命令不同, 这里使用符号“:=”将右侧定义内容赋值给左侧, 这相当于将右侧定义简记为“`Continuous f x0`”. 该形式化代码描述了函数 f 在 x_0 处连续的定义: x_0 属于 f 的定义域 (domain), 并且 f 在 x_0 处的极限值为 $f(x_0)$, 即 f 在 x_0 处的取值. 定义中相关概念的具体实现将在本文第 3 节展开.

此外, 在本文系统开发过程中, 对于一些需要使用递归或归纳方式的定义, 例如求和运算、阶乘等, 这些定义不易直接使用 Definition 实现, 需要使用 Fixpoint 或 Inductive 命令, 此处暂不展开, 下文中将结合具体使用场景具体说明.

3.2 基于简单类型的集合论基础

集合是现代数学理论的基础, 所有的数学表达都或多或少需要引入一些集合相关概念. 尽管罗素悖论的发现使得集合论的公理化成为必要[48–50], 但为方便理解, 一般数学分析教材中并不会特别强调公理化集合论, 而仍采用更为简洁直观的朴素集合论, 但这在对逻辑严密性具有极高要求的定理证明工具中是行不通的.

一种简单的办法是, 直接使用类型论进行形式化开发, 例如 Coq 官方标准库[37]中的分析学部分, 然而类型论对于大部分数学学习者来说可能较为陌生, 不易理解. 另一种办法是, 在 Coq 中开发一套公理化集合论体系, 进而在此基础上展开数学理论的形式化工作, 例如我们之前的工作[44–46,51,52]. 然而理解公理化集合论本身也可能需要花费大量时间, 难以将系统应用于数学分析的辅助学习或教学.

针对上述问题, 我们采用一个折衷的解决方案, 基于简单类型的朴素集合论. 传统的朴素集合论将集合描述成“由一系列 (满足某特定性质的) 元素所构成的整体”, 这种未对集合规模加以限制的定义自然会导致悖论的出现: $\{u \mid u \notin u\} \in \{u \mid u \notin u\}$ 当且仅当 $\{u \mid u \notin u\} \notin \{u \mid u \notin u\}$. 为此, 可以将集合限制在某些简单的类型中.

一个集合, 本质上也是一个带有参数的命题, 例如集合 $\{u \mid 2 < u\}$, 表示由所有大于 2 的元素构成的集合, 这里的 u 即为参数, “大于 2 ”为参数需满足的命题. 然而, 这里的参数 u 并未加以任何限制, 它可以是自然数、有理数、实数、甚至是序数, 集合的具体含义也可因 u 所属类型的不同而不同. ZFC 公理化集合论[48–50]通过内涵公理增加了这种限制, 使得每个集合都作为某个已有集合的子集而存在. 沿用这种思想, 如果将参数 u 预先限定在某个类型中, 则可以严谨表述集合 $\{u \mid 2 < u\}$ 的含义. 例如 Coq 系统中内建有自然数类型 nat, 集合 $\{u \mid 2 < u\}$ 可以非形式化的表示为 $\{u\colon \mathrm{nat} \mid 2 < u\}$, 表明该集合由所有大于 2 的自然数构成. 由此, 集合可以按如下方式给出形式化定义:

```
Definition set {U : Type} := U -> Prop.
```

这里的 U 表示一个预先指定的类型, “{U : Type}”将 U 隐式表达, Coq 系统可根据 U 的具体值判断 U 是何种类型. 事实上, Coq 官方标准库[37]的 Sets.Ensembles 模块中即按此方法定义集合, 不过其后的数学分析库并未依赖该模块, 而是基于类型论展开. 本文系统基于该集合体系进行开发.

按照如上定义, 集合是一个带有类型为 U 参数的命题表达式. 为便于理解, 我们在此定义的基础上嵌套一层与数学中集合符号类似的形式化记法, 以便于使用者理解.

```
Definition ClassifierI {U : Type} (P : U -> Prop) := P.
Notation "\{ P \}" := (ClassifierI P) (at level 0).
Notation " 'λ' x .. y , t" := (fun x => .. (fun y => t) ..)
    (at level 200, x binder, y binder, right associativity).
```

按照如上记法, 以“空集”和“全集”的概念为例, 可形式化定义为:

```
Definition Empty (A : Type) := \{ λ (x : A), x <> x \}.
Definition Full (A : Type) := \{ λ (x : A), x = x \}.
```

这与数学中的习惯写法 $\{x \mid x \neq x\}$ 和 $\{x \mid x = x\}$ 类似, 从而增强了代码的可读性. 不同之处在于, 形式化中的参数 x 被限制在了类型 A 中, 因而并不等同于朴素集合论中“全集”的概念, 不会产生悖论. 定义中的“λ (x : A), x <> x”和“λ (x : A), x = x”即为形式化符号“\{ P \}”中 P 的实例; 当参数 x 的类型可由 Coq 根据其后的命题自行推断时, 可简写为“\{\ λ x, x <> x \}\”和“\{\ λ x, x = x \}\”.

实际上, 上述记法的实现过程利用了类型论中的 λ 演算, 具体可参考文献 [13].

基于以上内容, 可以进一步对集合基本概念进行形式化描述.

(1) 属于关系

一个元素 x 属于 A, 实际上就是 x 满足与 A 对应的性质, 因此集合的属于关系可形式化定义如下:

```
Definition In {U : Type} (x : U) (A : set) := A x.
Notation "x ∈ A" := (In x A) (at level 80).
Definition NotIn {U : Type} (x : U) (A : set) := ~ (x ∈ A).
Notation "x ∉ A" := (NotIn x A)(at level 80).
```

按照这种定义, 可以迅速得出一个习以为常的基本事实:

```
Fact Classifier1 : ∀ (A : Type) (x : A) (P : A -> Prop), x ∈ \{ P \} <-> P x.
Proof. split; intros; auto. Qed.
```

例如, 非形式化表达 $x \in \{u: \text{nat} \mid 2 < u\}$ 即等价于 $2 < x$, 其形式化描述为:

```
x ∈ \{ λ (u : nat), 2 < u \} <-> 2 < x
```

(2) 序偶对与关系

序偶对是集合中的一个重要概念, 其对于函数的定义起到关键作用. 与之前诸多概念的定义有所不同, 此处利用 Inductive 命令进行定义.

```
Inductive prod {X Y : Type} : Type := | pair (x : X) (y : Y).
Notation "[ x , y ]" := (pair x y)(at level 0).
```

以上代码定义了一个新的类型 prod, 它接受两个既定的类型 X 和 Y, pair 为该类型的唯一构造方式, 称为是 prod 的构造子 (constructor); pair 自身接受两个类型分别为 X 和 Y 的值 x 和 y, 得到的"`pair x y`"则对应于数学中的序偶对 (x,y), 形式化记为"`[x, y]`". 例如, 假定序偶对 $(1,\pi)$ 中的 1 为自然数类型 nat 的值, π 为实数类型 R 的值, 则 $(1,\pi)$ 整体的类型为"`prod nat R`", 其形式化的描述和记号分别为"`pair 1 π`"和"`[1, π]`" (如果圆周率在 Coq 中已被严格定义并记为 π).

这种定义的好处在于, 序偶对具有了类型结构, 这种结构较为简洁, 便于理解, 同时也可以利用类型匹配 (match) 的方式迅速给出坐标的定义.

```
Definition fst {X Y : Type} (p : (@prod X Y)) := match p with [x, y] => x end.
Definition snd {X Y : Type} (p : (@prod X Y)) := match p with [x, y] => y end.
```

以上分别给出了序偶对 (x,y) 第一坐标和第二坐标的定义. 其中符号"@"用于将隐式参数显式化, 以准确表达参数 p 的类型; 由于 p 是一个序偶类型 prod 的值, 按定义其结构必定形如"`[x, y]`", 因此可利用匹配机制 (match) 直接对 $[x, y]$ 进行操作. 直观地, 序偶对 $[x, y]$ 的第一坐标为 x, 第二坐标为 y.

此外, 可以证明如下简单性质:

```
Property ProdEqual : ∀ {X Y : Type} (x u : X) (y v : Y), [x, y] = [u, v] -> x = u / \
y = v.
```

两个序偶对相等, 则其第一和第二坐标分别对应相等.

关于序偶对, 还可引入一种记法以专门描述由序偶对组成的集合:

```
Definition ClassifierII {X Y : Type} (P : X -> Y -> Prop) := \{ λ u , ∃ x y, u = [x,
y] /\ P x y \}.
Notation "\{\ P \}\" := (ClassifierII P) (at level 0).
```

基于以上定义和记号, 集合 $\{(u,v) \mid u < v\}$ (设 u, v 均为自然数) 的形式化写法为"`\{\ λ (u v : nat), u < v \}\`", 其中"`λ (u v : nat), u < v`"即为形式化符号"`\{\ P \}\`"中 P 的一个实例; 当参数 u 和 v 的类型可由 Coq 根据其后的命题自行推断时, 可简写为"`\{\ λ u v, u < v \}\`".

按上述记法, 可以迅速得出序偶对集合的属于关系:

```
Fact Classifier2 : ∀ (X Y : Type) (x : X) (y : Y) (P : X -> Y -> Prop), [x, y] ∈ \{\
P \}\ <-> P x y.
```

例如, 非形式化表达 $(x,y) \in \{(u,v) \mid u < v\}$ 即等价于 $x < y$ (假定针对自然数), 其形式化描述为:

```
[x, y] ∈ \{\ λ (u v : nat), u < v \}\ <-> x < y
```

(3) 其他基本定义

基于已有内容, 可以形式化定义一些常用集合概念, 例如子集的定义、集合的交并补运算等. 现将本文形式化系统中使用到的基本集合概念及符号约定汇总于表 2.

(4) 外延性公理与选择公理

外延性公理是对等号意义在集合中的扩展, 在证明两个集合相等的过程中, 通常会运用"两个集合含有相同元素"的思想, 这里即使用了外延性公理.

表 2 基本集合概念

数学含义	Coq定义	Coq符号	数学符号
集合	`Definition set {U : Type} := U -> Prop.`	`\{ λ _, _ \}` `\{\ λ _ _, _ \}\`	$\{\ldots\mid\ldots\}$ $\{(\_,\_)\mid\ldots\}$
x属于A	`Definition In {U : Type} (x : U) (A : set) := A x.`	`X ∈ A`	$x\in A$
x不属于A	`Definition NotIn {U : Type} (x : U) (A : set) := ~ (x ∈ A).`	`X ∉ A`	$x\notin A$
序偶对	`Inductive prod {X Y : Type} : Type := | pair (x : X) (y : Y).`	`[x, y]`	(x,y)
第一坐标	`Definition fst {X Y : Type} (p : (@prod X Y)) :=` `match p with [x , y] => x end.`	—	—
第二坐标	`Definition snd {X Y : Type} (p : (@prod X Y)) :=` `match p with [x , y] => y end.`	—	—
S为非空集	`Definition NotEmpty {A : Type} (S : set) := ∃ (x : A), x ∈ S.`	—	—
空集	`Definition Empty (A : Type) := \{ λ (x : A), x <> x \}.`	—	$\varnothing$
全集	`Definition Full (A : Type) := \{ λ x : A, x = x \}.`	—	—
单点集	`Definition Singleton {A: Type} (x : A) := \{ λ z, z = x \}.`	`[x]`	$\{x\}$
x包含于y	`Definition Contain {A : Type} (x y : (@set A)) :=` `∀ z, z ∈ x -> z ∈ y.`	`x ⊂ y`	$x\subset y$; $x\subseteq y$
x并y	`Definition Union {A : Type} (x y : (@set A)) :=` `\{ λ z : A, z ∈ x \/ z ∈ y \} .`	`x ∪ y`	$x\cup y$
x交y	`Definition Intersection {A : Type} (x y : (@set A)) :=` `\{ λ z : A, z ∈ x /\ z ∈ y \}.`	`x ∩ y`	$x\cap y$
x的余集	`Definition Complement {A : Type} (x : (@set A)) :=` `\{ λ y : A, y ∉ x \}.`	`¬x`	$\sim x$
x差y	`Definition Difference {A : Type} (x y : (@set A)) :=` `x ∩ (¬ y).`	`x-y`	$x-y$
笛卡尔乘积	`Definition Cartesian {X Y : Type} (x : (@set X))` `(y : (@set Y)) := \{\ λ u v, u ∈ x /\ v ∈ y \}\.`	`x × y`	$x\times y$

公理 2. 外延性公理. 对于集合 x 与 y, $x=y$ 成立的充要条件为对于任意元素 z, $z\in x$ 当且仅当 $z\in y$.

```
Axiom AxiomI : ∀ x y, x = y <-> (∀ z, z ∈ x <-> z ∈ y).
```

选择公理是处理从无限集合中选择元素, 以及构造非构造性对象的重要工具, 它在古典逻辑的证明中往往会不知不觉地被使用[50]. 选择公理常用的数学表述如下.

公理 3. 选择公理. 设 a 是由非空集合组成的集族, 则存在以 a 为定义域的函数 f, 对任意 $x\in a$, 有 $f(x)\in x$.

可以看到, 选择公理实际上承认了每一个集合都存在一种可以从中选取元素的方式, 即公理中的函数 f (称为选择函数). 由于此处的形式化中尚未引入数学里的函数概念, 因此可通过类型的方式实现这种“取元操作”.

```
Parameter c : ∀ {U : Type}, @set U -> U.
Axiom Axiomc : ∀ {U : Type} (x : @set U), NotEmpty x -> (c x) ∈ x.
```

以上代码首先声明了一个全局常量 c, 该常量可接受一个 U 类型的集合作为参数输入, 输出一个 U 类型的值; 第 2 行的公理则规定了该输出值一定属于输入集合, 由此便实现了集合的“取元操作”.

上述“取元操作”不仅可以实现非构造性对象和无限集合元素的选取, 也可以方便地处理一些简单结构的元素选取问题. 例如, 可以利用常量 c 将单点集中的元素取出.

```
Fact ChooseSingleton : ∀ {U : Type} (x : U), c [x] = x.
```

在本文后续一些定义的形式化描述和定理的证明过程中, 可见更多选择公理和“取元操作”的运用.

值得指出, 上述逻辑与集合论体系中涉及的 3 条公理: 排中律、外延性公理和选择公理 (取元操作), 均为在一般数学中得到承认且不会产生矛盾的公理[50], 并且 3 条公理也分别可通过读入 Coq 官方标准库中的 Logic.Classical、Sets.Ensembles 和 Logic.Epsilon 模块得到实例化或证明 (但这势必造成一些代码冗余, 针对性地提取其中的公理则更精准简洁), 可以放心使用. 感兴趣的读者可进一步参考这些模块中的内容.

3.3 函数概念的形式化

函数是两个集合 X, Y 之间满足单值性原则的一种对应法则, 因此在集合论中, 函数实际上应当是一系列形如 (x,y) 的序偶对组成的集合, 其中 $x\in X$, $y\in Y$, 并且 y 对 x 具有单值性对应, X 与 Y 分别称为函数的定义域和值域. 从而函数可定义如下:

```
Definition Function {A B : Type} (f : (@set (@prod A B))) :=
    ∀ x y z, [x, y] ∈ f -> [x, z] ∈ f -> y = z.
```

首先, f 被限定为一个序偶集合, 参数 A 与 B 指定了序偶的类型, f 是一个函数就是说, 不允许其中的两个不同元素含有相同的的第 1 坐标 (一但有, 那么两个元素必相同), 以此实现 f 的单值性. 函数的定义域和值域则分别为 f 中所有元素的第 1 坐标和第 2 坐标组成的集合.

```
Definition Domain {A B : Type} (f : (@set (@prod A B))) := \{ λ x, ∃ y, [x, y] ∈ f \}.
Notation "dom[f]" := (Domain f) (at level 5).
Definition Range {A B : Type} (f : (@set (@prod A B))) := \{ λ y, ∃ x, [x, y] ∈ f \}.
Notation "ran[f]" := (Range f) (at level 5).
```

形式化中以 `dom[f]` 表示 f 的定义域, `ran[f]` 表示值域.

函数在 x 处的取值, 即为 x 所在序偶所对应的第 2 坐标, 形式化中可通过取元方式得到 x 对应的函数值.

```
Definition Value {X Y : Type} (f : (@set (@prod X Y))) (x : X) := c \{ λ y, [x, y] ∈ f \}.
Notation "f[x]" := (Value f x) (at level 5).
```

下面 3 条性质可验证以上函数相关定义的合理性, 其所陈述的内容是简洁明了的.

```
Property f_x_T : ∀ {X Y : Type} (f : (@set (@prod X Y))) (x : X)(y : Y), Function f ->
[x, y] ∈ f -> f[x] = y.
Property x_fx_T : ∀ {X Y : Type} (f : (@set (@prod X Y))) (x : X), Function f -> x ∈
dom[f] -> [x, f[x]] ∈ f.
Property fx_ran_T : ∀ {X Y : Type} (f : (@set (@prod X Y))) (x : X), Function f -> x
∈ dom[f] -> [x, f[x]] ∈ f.
```

数学分析中经常有只讨论函数在某段区间或某个集合上的特性, 例如讨论函数在某段区间的连续性和单调性, 计算函数在区间上的定积分等. 为此引入“函数限制”的形式化定义.

```
Definition Restriction {X Y : Type} (f : (@set (@prod X Y))) (D : (@set X)) := f ∩
(D × (Full Y)).
Notation "f|[D]" := (Restriction f D)(at level 30).
```

该定义的作用是将函数 f 的定义域限定在集合 D 上, 使得限制后函数 (数学符号记为 $f|_D$) 的定义域为 f 定义域与 D 的交集, 同时不会改变原有函数值.

```
Property RestrictFun : ∀ {X Y : Type} (f : (@set (@prod X Y))) (D : (@set X)),
Function f -> (Function (f|[D]) /\ dom[f|[D]] = (D ∩ (dom[f])) /\ (∀ x, x ∈ dom[f|[D]] ->
(f|[D])[x] = f[x])).
```

最后, 定义复合函数和反函数.

```
Definition Composition {X Y Z : Type} (f : (@set (@prod Y Z))) (g : (@set (@prod X
Y))) := \{\ λ x y, (∃ z, [x, z] ∈ g /\ [z, y] ∈ f) \}\.
Notation "f∘g" := (Composition f g) (at level 50, no associativity).
Definition Inverse {X Y : Type} (f : (@set (@prod X Y))) := \{\ λ x y, [y, x] ∈ f \}\.
Notation "f^-1" := (Inverse f) (at level 5).
```

需要注意, 以上形式化定义中的相关参数 f 和 g 仅是序偶对集合, 并不一定是函数. 实际上, `f∘g` 是两个序偶

集合的复合; f^{-1} 是将集合 f 中所有序偶对的第一和第二坐标互换得到的结果, 是集合 f 的逆. 这种定义方式具有更广泛的适用范围, 在需要的情形仅需增加"f 和 g 是函数 (Function f 和 Function g)"的条件即可.

由于一个序偶集合的逆未必为一个函数, 1-1 函数 (即双射函数) 则定义为其逆也为函数的函数, 此时函数的逆称为原函数的反函数.

```
Definition Function1_1 {X Y : Type} (f : (@set (@prod X Y))) := Function f /\ Function (f^-1).
Definition Inverse_Function {X Y : Type} (f : (@set (@prod X Y))) g := Function1_1 f /\ g = f^-1.
```

1-1 函数的定义域与值域中的元素具有一一对应的关系, 对于接下来有限集的定义至关重要.

3.4 自然数与有限集

数学分析中不乏涉及有限与无限概念的表述, 例如, 函数具有有限个间断点、集合含有无限个元素、黎曼积分也是先对积分区间进行有限步划分后再向无限逼近. 对于这些陈述, 学习者往往习惯于直观层面的理解, 忽略其中有限与无限概念的严谨性定义, 但在形式化系统中, 必须给出其严格定义, 并且, 借助形式化系统, 也有助于理清有限与无限这一对重要概念在相关定理中的作用.

一个集合 A 是有限的, 就是要求 A 中的元素能够被某个自然数 n 计数, 其数学表达为: 存在一个 1-1 函数, 其定义域为集合 $\{u \mid u \leqslant n\}$, 值域为 A.

对于有限概念的定义, 自然数是必不可少的, 这可以借助 Coq 内建的自然数类型 nat 实现, 该类型在打开 Coq 编译环境时会自动加载, 可用"`Print nat.`"命令查看其具体定义.

```
Inductive nat :=
  | O : nat
  | S : nat -> nat.
```

该定义使用 Inductive 命令实现, 是一种归纳定义. 具体来说, nat 类型含有两个构造子: O 和 S, 其中 O 是归纳的起点, 表示自然数 0, 代码中也可直接输入"0"; S 用于表示自然数的后继, 它接受一个自然数作为输入, 输出的自然数则表示输入自然数的后继. 例如, `S 0` 表示 1, `S (S 0)` 表示 2, 以此类推.

实际上, 数学中的自然数通常是采用 Peano 5 条公理[50]展开的, Coq 具有强大的归纳构造演算功能, 5 条公理可以直接作为 Coq 系统中的推论被证明, 其描述如下.

```
Corollary AxiomP1 : 0 ∈ Full nat.
Corollary AxiomP2 : ∀ m n p, S m = n -> S m = p -> n = p.
Corollary AxiomP3 : ∀ m , ~ (S m = 0).
Corollary AxiomP4 : ∀ m n, ~ (m = n) -> ~ (S m = S n).
Corollary AxiomP5 : ∀ M, 0 ∈ M -> (∀ n, n ∈ M -> (S n) ∈ M) -> ∀ m, m ∈ M.
```

此外, 得益于 Coq 的符号表示机制, 代码中可以直接使用阿拉伯数字记法表示相应自然数, 例如, 数字"10"不必采用以下输入:

```
S (S (S (S (S (S (S (S (S O)))))))))
```

只需在代码中输入"10"即可. 同时, 自然数的相关运算 (加法、减法、乘法和大小关系) 也均已形式化定义, 且引入了与数学符号一致的记法, 稍有不同的仅有: 小于等于号 $\leqslant$ (<=) 和大于等于号 $\geqslant$ (>=).

基于 Coq 内建的自然数体系, 现可以形式化定义有限集:

```
Definition NotEmptyFinite {X} (A : set) := ∃ n (f : @set (@prod nat X)), Function1_1 f /\ dom[f] = \{ λ x, x <= n \} /\ ran[f] = A.
```

需要注意的是, 上面的定义实际描述了"非空有限集", 此因集合"`\{ λ x, x <= n \}`"始终非空. 考虑将空集也纳入有限集, 则有限集应当定义为:

```
Definition Finite {X} A := NotEmptyFinite A \/ A = (Empty X).
```

目前为止, 自然数类型 nat 是本文系统中引入的第 1 个具体的类型, 加上第 4 节即将介绍的实数类型 R, 整个集合论实际仅建立在这两个类型之上, 再无引入其他类型, 是一种基于简单类型的朴素集合论.

4 基于 Coq 标准库的实数理论基础

实数理论是开展数学分析的基础, 严格来说, 应首先对其进行形式化实现. 实际上, 实数理论在数学中可作为一个相对独立的理论体系, 我们已有的工作对于实数理论的形式化[44–46]已有较为深入的探讨和研究.

在本文系统的开发中, 为避免使用者陷入实数理论的形式化细节, 我们直接采用 Coq 标准库中的实数进行开发, 可通过以下代码导入相关内容:

```
Require Export Coq.Reals.RIneq.
```

该模块提供了一个基于类型论的实数公理化[53]定义, 包括实数类型 (命名为 R)、运算 (加、减、乘、除)、序关系 (即实数的大小关系)、完备性公理等, 相关运算符号与自然数 nat 类型完全一致. 一些熟知的实数性质, 如稠密性、阿基米德性等, 均已基于类型论得到形式化证明.

为兼容本文系统, 本节与教材[35]第 1 章对应, 用集合论补充和扩展 Coq 标准库中的实数相关内容.

4.1 数集与确界原理

熟知绝对值的定义为:

$$|a| = \begin{cases} a, & a \geqslant 0 \\ -a, & a < 0 \end{cases}$$

这实际上是一个函数, 因此在形式化中需要定义一个绝对值函数, a 的绝对值记为函数在 a 处的取值.

```
Definition Abs := \{\ λ x y , (x < 0 /\ y = -x) \/ (x >= 0 /\ y = x) \}\.
Notation "|a|" := (Abs[a])(at level 5, a at level 0).
```

区间与邻域是实数中的两类重要数集, 可基于集合概念给出它们的形式化描述. 以开区间为例, 其形式化定义为:

```
Definition OpenInterval1 a b := \{ λ x, x > a /\ x < b \}.
Notation "\( a , b \)" := (OpenInterval1 a b) (at level 5).
```

而左去心邻域又可基于开区间给出形式化定义:

```
Definition leftNeighbor0 a δ := \(a-δ, a\).
Notation "U-º( a ; δ )" := (leftNeighbor0 a δ)(a, δ at level 0, at level 4).
```

表 3 给出常用区间及邻域的形式化及其对应数学表示.

表 3 常用区间及邻域概念

数学含义	Coq定义	Coq符号	数学符号
开区间	`Definition OpenInterval1 a b := \{ λ x, x > a /\ x < b \}.`	`\(a, b\)`	(a,b)
闭区间	`Definition CloseInterval a b := \{ λ x, x >= a /\ x <= b \}.`	`\[a, b\]`	$[a,b]$
左闭右开区间	`Definition CloseOpen a b := \{ λ x, x >= a /\ x < b \}.`	`\[a, b\)`	$[a,b)$
左开右闭区间	`Definition OpenClose a b := \{ λ x, x > a /\ x <= b \}.`	`\(a, b\]`	$(a,b]$
正无穷区间	`Definition OpenPositiveInf a := \{ λ x, x > a \}.`	`\(a, +∞\)`	$(a,+\infty)$
负无穷区间	`Definition OpenNegativeInf a := \{ λ x, x < a \}.`	`\(-∞, a\)`	$(-\infty,a)$
邻域	`Definition Neighbor a δ := \{ λ x, |(x-a)| < δ \}.`	`U°(a; δ)`	$U(a;\delta)$
左邻域	`Definition leftNeighbor a δ := \(a-δ, a\].`	`U-(a; δ)`	$U_{-}(a;\delta)$
右邻域	`Definition rightNeighbor a δ := \[a, a+δ\).`	`U+(a; δ)`	$U_{+}(a;\delta)$
去心邻域	`Definition Neighbor0 a δ := \{ λ x, 0 < |(x-a)| < δ \}.`	`Uº(a; δ)`	$U^{\circ}(a;\delta)$
左去心邻域	`Definition leftNeighbor0 a δ := \(a-δ, a\).`	`U-º(a; δ)`	$U^{\circ}_{-}(a;\delta)$
右去心邻域	`Definition rightNeighbor0 a δ := \(a, a+δ\).`	`U+º(a; δ)`	$U^{\circ}_{+}(a;\delta)$

确界原理是体现实数完备性的重要定理, 而完备性是有理数所不具备的性质, 是区分实数与有理数本质的重要依据. Coq 标准库中已有对实数确界原理的描述和证明, 这里给出基于本文系统的形式化版本.

定义 1. 集合上界与下界. 设 S 为一个实数集合, 若存在数 $M(L)$, 使得对一切 $x\in S$ 都有 $x\leqslant M(x\geqslant L)$, 则称数 $M(L)$ 为 S 的上界 (下界), 称 S 为一个有上界 (下界) 的数集.

```
Definition Upper S M := ∀ x, x ∈ S -> x <= M.
Definition Lower S L := ∀ x, x ∈ S -> x >= L.
```

若数集 S 既有上界也有下界, 则称该数集是一个有界集; 若 S 不是有界的, 则称其为无界集.

```
Definition Bounded S := ∃ M L : R, (Upper S M) /\ (Lower S L).
Definition unBounded S := ~ (Bounded S).
```

定义 2. 集合上确界与下确界. 设 S 为一个实数集合, 若数 $\eta(\xi)$ 满足:

(i) 对一切 $x\in S$ 都有 $x\leqslant\eta(x\geqslant\xi)$, 即 $\eta(\xi)$ 是 S 的上界 (下界);

(ii) 对任何 $\alpha<\eta$ (对任何 $\beta>\xi$), 存在 $x_0\in S$, 使得 $x_0>\alpha$ ($x_0<\beta$), 即 $\eta(\xi)$ 又是 S 的最小上界 (最大下界);

则称数 $\eta(\xi)$ 为数集 S 的上确界 (下确界).

```
Definition sup S η := (Upper S η) /\ (∀ a, a < η -> (∃ x0, x0 ∈ S /\ x0 > a)).
Definition inf S ξ := (Lower S ξ) /\ (∀ b, b > ξ -> (∃ x0, x0 ∈ S /\ x0 < b)).
```

定理 1. 确界原理. 若非空数集 S 有上界, 则 S 有上确界; 若 S 下界, 则必有下确界.

```
Theorem Sup_inf_principle : ∀ A, NotEmpty A
  -> ((∃ M, Upper A M) -> (∃ η, sup A η)) /\ ((∃ L, Lower A L) -> (∃ ξ, inf A ξ)).
```

4.2 具有某些特性的函数

对于定义在实数上的函数, 本小节主要给出有界函数、单调函数以及函数四则运算的形式化.

定义 3. 有上界函数与有下界函数. 设 f 为定义在数集 D 上的函数, 若存在数 $M(L)$, 使得对每个 $x\in D$ 有:

$$f(x)\leqslant M(f(x)\geqslant L),$$

则称 f 为 D 上的有上 (下) 界函数, $M(L)$ 称为 f 在 D 上的一个上 (下) 界.

```
Definition UpBoundedFun f D := Function f /\ D ⊂ dom[f] /\ ∃ M, (∀ x : R, x ∈ D ->
f[x] <= M).
Definition LowBoundedFun f D := Function f /\ D ⊂ dom[f] /\ ∃ L, (∀ x : R, x ∈ D ->
f[x] >= L).
```

定义 4. 有界函数. 设 f 为定义在数集 D 上的函数, 若存在正数 M, 使得对每个 $x\in D$ 有:

$$|f(x)|\leqslant M,$$

则称 f 为 D 上的有界函数.

```
Definition DBoundedFun f D := Function f /\ D ⊂ dom[f] /\ ∃ M, (∀ x : R, x ∈ D ->
|(f[x])| <= M).
```

定义 5. 增函数与减函数. 设 f 为定义在数集 D 上的函数, 若对任意 $x_1,x_2\in D$, 当 $x_1<x_2$ 时, 总有:

(i) $f(x_1)\leqslant f(x_2)$, 则称 f 为 D 上的 (递) 增函数, 若严格成立 $f(x_1)<f(x_2)$, 称 f 为 D 上的严格 (递) 增函数;

(ii) $f(x_1)\geqslant f(x_2)$, 则称 f 为 D 上的 (递) 减函数, 若严格成立 $f(x_1)>f(x_2)$, 称 f 为 D 上的严格 (递) 减函数.

```
Definition DIncreaseFun f D := Function f /\ D ⊂ dom[f]
  /\ (∀ x1 x2, x1 ∈ D -> x2 ∈ D -> x1 < x2 -> f[x1] <= f[x2]).
Definition DStrictlyIncreaseFun f D := Function f /\ D ⊂ dom[f]
  /\ (∀ x1 x2, x1 ∈ D -> x2 ∈ D -> x1 < x2 -> f[x1] < f[x2]).
Definition DDecreaseFun f D := Function f /\ D ⊂ dom[f]
  /\ (∀ (x1 x2 : R), x1 ∈ D -> x2 ∈ D -> x1 < x2 -> f[x1] >= f[x2]).
```

```
Definition DStrictlyDecreaseFun f D := Function f /\ D ⊂ dom[f]
  /\ (∀ (x1 x2 : R), x1 ∈ D -> x2 ∈ D -> x1 < x2 -> f[x1] > f[x2]).
```

增函数与减函数统称为单调函数, 严格增函数与严格减函数统称为严格单调函数.

```
Definition DMonotonicFun f D := DIncreaseFun f D \/ DDecreaseFun f D.
Definition DStrictlyMonotonicFun f D := DStrictlyIncreaseFun f D \/
DStrictlyDecreaseFun f D.
```

以上定义的形式化描述中, 数集 D 均被限定为函数 f 定义域的子集, 并不一定严格等于 f 的定义域, 这样可使定义具有更广泛的适用范围.

下面给出函数四则运算的严格形式化定义. 以函数加法为例, 设 F 为给定函数 f 和 g 的求和结果, 则 F 应当为一个函数, 并且满足 $F(x)=f(x)+g(x)$, 其中 x 同属于 f 和 g 的定义域. 因此, 形式化定义函数加法, 也就是要将这里的函数 F 描述出来.

```
Definition plus_fun (f g : @set (@prod R R)) :=
  (\{\ λ x y, x ∈ dom[f] /\ x ∈ dom[g] /\ y = f[x] + g[x] \}\).
Notation "f \+ g" := (plus_fun f g) (at level 45, left associativity).
```

这里的形式化定义 plus_fun 实际上就是对函数 F 的描述, 可以证明它是一个函数:

```
Corollary Corollary_plus_fun_a : ∀ f g, Function (f \+ g).
```

类似地可以定义函数的减法、乘法和除法.

```
Definition sub_fun (f g : @set (@prod R R)) :=
  (\{\ λ x y, x ∈ dom[f] /\ x ∈ dom[g] /\ y = f[x] - g[x] \}\).
Notation "f \- g" := (sub_fun f g)(at level 45, left associativity).
Definition mult_fun (f g : @set (@prod R R)) :=
  (\{\ λ x y, x ∈ dom[f] /\ x ∈ dom[g] /\ y = f[x] * g[x] \}\).
Notation "f \* g" := (mult_fun f g) (at level 40, left associativity).
Definition div_fun (f g : @set (@prod R R)) :=
  (\{\ λ x y, x ∈ dom[f] /\ x ∈ dom[g] /\ g[x] <> 0 /\ y = f[x] / g[x] \}\).
Notation "f // g" := (div_fun f g)(at level 40, left associativity).
```

为方便区分, 函数运算符号与实数 (R 类型) 以及自然数 (nat 类型) 的运算符号略有区别.

4.3 自然数嵌入实数

由于自然数与实数具有各自独立的类型结构, 同时又使用了相同的运算符号, 这会引发两个问题: (1) 运算符号如何解释, 例如“u+v”应该被解释为两个自然数相加还是两个实数相加; (2) 如何让自然数作为实数参与运算, 例如在表示数列 $\left\{\frac{1}{n}\right\}$ 时, n 在形式化中具有自然数类型, 然而进行除法运算时却要求 n 为实数类型.

对于第 1 个问题, Coq 系统分别通过界定关键字 (delimiting key) “%nat”和“%r”区分自然数和实数符号的作用域, 例如“(u+v)%nat”仅表示自然数相加, “(u<v)%r”则仅表示实数的大小关系. 不过, 在本文系统读入实数库之后, 我们通过下面一行指令使得系统中所有运算符号默认作用于实数.

```
Open Scope R_scope.
```

因此, 在无特别说明的情况下, “u+v”表示实数相加, “(u+v)%nat”表示自然数相加, 其他运算符同理.

上述过程的实现实际上依赖于 Coq 中的解释辖域 (interpretation scope) 机制, 详情可参见文献 [13].

对于第 2 个问题, Coq 实数库中通过以下形式化定义将自然数嵌入实数:

```
Fixpoint INR (n : nat) : R :=
  match n with
  | O => 0%r
```

```
  | S O => 1%r
  | S n => ((INR n) + 1)%r
  end.
```

该定义采用命令 Fixpoint 实现, 是一种递归方式的定义, 它接受一个自然数类型的值为输入, 输出是一个实数. 其递归思路为: 将自然数 0 应到实数 0; 将自然数 1 对应到实数 1; 将自然数 n 的后继对应到实数的 n_r+1 (这里的 n_r 表示 n 对应的实数, 即代码中的"`INR n`") . 按这种定义, 可将每个自然数类型的值 n 转换为其对应实数类型的值 INR n, 从而直接参与运算, 实现自然数在实数中的嵌入, 其本质也是一个类型转换过程.

以下以实数阿基米德性的一种形式化描述为例体现这种嵌入的作用.

```
Property Archimedes : ∀ a b, 0 < a -> 0 < b -> ∃ (n : nat), (INR n) * a > b.
```

该性质表明, 对任意正数 a 和 b, 总存在一个自然数 n 使得 $na>b$, 表明了实数是可有限度量的. 代码中由于 n 具有自然数类型, 而 a 具有实数类型, 因此需利用 INR 将 n 转换为实数类型后再与 a 相乘.

至此, 本文系统所依赖的逻辑、集合与实数基础介绍完毕, 相关概念相互联系, 构成本文数学分析形式化系统的基础, 图 4 表达了这一基础的整体架构.

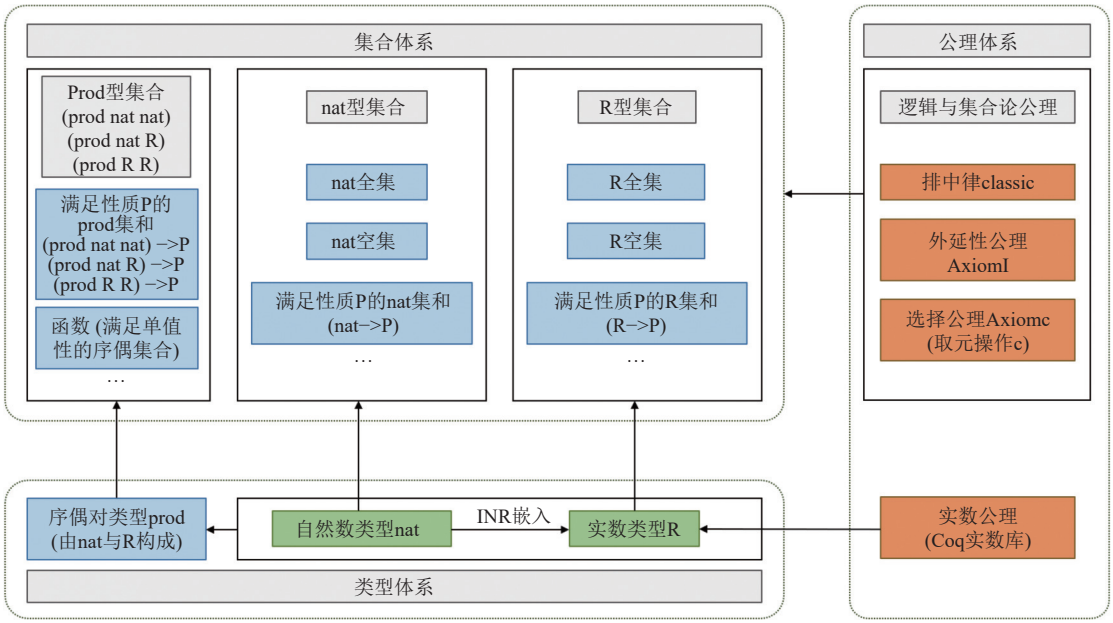

图 4　逻辑、集合与实数基础架构与组织关系

5　极限、导数与微分的形式化

从本节开始, 我们严格对照教材[35]上册的内容介绍一元微积分形式化系统.

5.1　数列极限

数列本质上是定义域为全体自然数集的函数, 其形式化定义为:

```
Definition Seq := @set (@prod nat R).
Definition IsSeq (f : Seq) := Function f /\ dom[f] = Full nat.
```

其中, Seq 定义为数列引入了一种特定类型, 确保数列是由第 1 坐标为自然数、第 2 坐标为实数的序偶对所组成的集合, 进而利用 `IsSeq f` 定义 f 是定义域为全体自然数集 (Full nat) 的函数.

数列极限有两种定义方式.

定义 6. 数列极限 I. 设 f 为数列, a 为常数. 若对任意正数 ε, 总存在正整数 N, 使得当 $n>N$ 时有:

$$|f(n)-a|<\varepsilon,$$

则称数列 f 收敛于 a, 常数 a 称为数列 f 的极限, 记为 $\lim\limits_{n\to\infty}f_n=a$.

定义 7. 数列极限 II. 对任意正数 ε, 若在邻域 $U(a;\varepsilon)$ 之外数列 f 的项至多有有限个, 则称数列 f 收敛于 a.

首先分别对两种定义进行形式化描述:

```
Definition limit_seq f a := IsSeq f /\ ∀ ε, ε > 0 -> (∃ N, ∀ n, (n > N)%nat ->
|(f[n] - a)| < ε).
Definition limit_seq' f a := IsSeq f /\ ∀ ε, ε > 0 -> Finite \{ λ u, f[u] ∉ U(a; ε) \}.
```

进而验证两个定义的等价, 以便于替换调用:

```
Fact EqualLimit : ∀ f a, limit_seq f a <-> limit_seq' f a.
```

以上形式化描述与数学记法 $\lim\limits_{n\to\infty}f_n$ 仍略有区别, 为尽量一致, 可以通过之前第 3.2 节中引入的全局常量 c 进行取元, 将数列的极限取出.

```
Definition lim f := c (\{ λ a, limit_seq f a \}).
Corollary lim_a : ∀ x a, limit_seq x a -> lim x = a.
```

极限的唯一性可确保以上取元过程的合理性.

```
Theorem Theorem2_2 : ∀ f a b, limit_seq f a -> limit_seq f b -> a = b.
```

5.2 函数极限

函数的极限分为两种, 一种是自变量 x 趋于无穷时的极限, 另一种是 x 在某一点 x_0 处的极限. 其中使用较多的是后者, 这里主要介绍函数在某一点处极限的形式化.

定义 8. 函数极限. 设函数 f 在点 x_0 的某个空心邻域 $U^{\circ}(x_0;\delta')$ 内有定义, A 为一定数. 若对任意 $\varepsilon>0$, 总存在正数 $\delta(<\delta')$, 使得当 $0<|x-x_0|<\delta$ 时有:

$$|f(x)-A|<\varepsilon,$$

则称函数 f 在 x_0 处以 A 为极限, 记作 $\lim\limits_{x\to x_0}f(x)=A$.

```
Definition limit f x0 A := Function f /\ (∃ δ', δ' > 0 /\ Uº(x0; δ') ⊂ dom[f]
  /\ (∀ ε, ε > 0 -> ∃ δ, 0 < δ < δ' /\ (∀ x, 0 < |(x - x0)| < δ -> |(f[x] - A)| < ε))).
```

形式化描述中“limit f x0 A”表达的含义即为: 函数 f 在 x_0 处的极限值为 A.

有时需要讨论函数在某一点处的单侧极限, 即左极限与右极限的概念, 其形式化相比上述函数极限的定义, 只需要做些许细节上的改动.

```
Definition limit_neg f x0 A := Function f /\ (∃ δ', δ' > 0 /\ U-º(x0; δ') ⊂ dom[f]
  /\ (∀ ε, ε > 0 -> ∃ δ, 0 < δ < δ' /\ (∀ x, x0 - δ < x < x0 -> |(f[x] - A)| < ε))).
Definition limit_pos f x0 A := Function f /\ (∃ δ', δ' > 0 /\ U+º(x0; δ') ⊂ dom[f]
  /\ (∀ ε, ε > 0 -> ∃ δ, 0 < δ < δ' /\ (∀ x, x0 < x < x0 + δ -> |(f[x] - A)| < ε))).
```

以上形式化定义的含义分别为: “limit_neg f x0 A”函数 f 在 x_0 处的左极限值为 A; “limit_pos f x0 A”函数 f 在 x_0 处的右极限值为 A, 数学上记作 $\lim\limits_{x\to x_0^-}f(x)=A$ 和 $\lim\limits_{x\to x_0^+}f(x)=A$.

函数在点 x_0 处极限为 A 当且仅当函数在点 x_0 处的左、右极限均为 A.

```
Theorem Theorem3_1 : ∀ f x0 A, limit f x0 A <-> (limit_pos f x0 A /\ limit_neg f x0 A).
```

利用极限概念可给出无穷小量与无穷大量的形式化定义. 若函数 f 在 x_0 处的极限值为 0, 则称 f 为自变量趋于 x_0 时的无穷小量.

```
Definition Infinitesimal f x0 := limit f x0 0.
```

对于两个同为趋于 x_0 时的无穷小量 f 和 g, 可进行比值分析, 以比较两个无穷小量收敛于 0 的速度快慢.

(1) 若 $\lim\limits_{x\to x_0}\dfrac{f(x)}{g(x)}=0$, 则称 f 为 g 在 $x\to x_0$ 时的高阶无穷小量, 或称 g 为 f 的低阶无穷小量.

```
Definition InfinitesimalHigherOrder f g x0 := Infinitesimal f x0 /\ Infinitesimal g x0
  /\ limit (f // g) x0 0.
```

形式化中需将 $\dfrac{f(x)}{g(x)}$ 整体作为一个函数, 因而需要使用函数的除法运算 (`f // g`).

(2) 若存在正数 K 和 L, 使得在某邻域 $U^\circ(x_0;\delta)$ 上有:

$$K\leqslant\left|\frac{f(x)}{g(x)}\right|\leqslant L,$$

则称 f 与 g 是在 $x\to x_0$ 时的同阶无穷小量.

同阶无穷小量的定义并未使用极限概念, 因此其中实际隐藏了一些条件: 首先函数 f 和 g 在邻域 $U^\circ(x_0;\delta)$ 上需均有定义, 即 $U^\circ(x_0;\delta)$ 必须包含于 f 和 g 的定义域; 其次函数 g 在 $U^\circ(x_0;\delta)$ 中不能等于 0, 否则分式无意义. 因此, 同阶无穷小量的完整形式化定义为:

```
Definition InfinitesimalSameOrder f g x0 := Infinitesimal f x0 /\ Infinitesimal g x0
  /\ (∃ δ, 0 < δ /\ Uº(x0;δ) ⊂ dom[f] /\ Uº(x0;δ) ⊂ dom[g] /\ (∀ x, x ∈ Uº(x0;δ) ->
g[x] <> 0)
    /\ (∃ K L, 0 < K <= L /\ (∀ x, x ∈ Uº(x0;δ) -> K <= |(f[x]/g[x])| <= L))).
```

可以证明, 当 $\lim\limits_{x\to x_0}\dfrac{f(x)}{g(x)}=c\neq 0$ 时, f 和 g 必为同阶无穷小量.

```
Corollary Corollary_InfinitesimalSameOrder : ∀ f g x0, Infinitesimal f x0 ->
Infinitesimal g x0
  -> (∃ c, c <> 0 /\ limit (f // g) x0 c) -> InfinitesimalSameOrder f g x0.
```

如此便可通过类似高阶无穷小量定义的方式判别同阶无穷小量.

(3) 若 $\lim\limits_{x\to x_0}\dfrac{f(x)}{g(x)}=1$, 则称 f 与 g 是在 $x\to x_0$ 时的等价无穷小量.

```
Definition InfinitesimalEquivalent f g x0 := Infinitesimal f x0 /\ Infinitesimal g x0
  /\ limit (f // g) x0 1.
```

显然等价无穷小量也是同阶无穷小量.

等价无穷小量对于一些极限问题的求解具有重要作用. 设函数 f 和 g 是在 $x\to x_0$ 时的等价无穷小量, 函数 h 在 x_0 的某去心邻域上有定义, 则有:

(i) 若 $\lim\limits_{x\to x_0}f(x)h(x)=A$, 则 $\lim\limits_{x\to x_0}g(x)h(x)=A$;

(ii) 若 $\lim\limits_{x\to x_0}\dfrac{h(x)}{f(x)}=B$, 则 $\lim\limits_{x\to x_0}\dfrac{h(x)}{g(x)}=B$.

```
Theorem Theorem3_12a : ∀ f g h x0 A, InfinitesimalEquivalent f g x0 -> limit (f \* h) x0 A
  -> limit (g \* h) x0 A.
Theorem Theorem3_12b : ∀ f g h x0 B, InfinitesimalEquivalent f g x0 -> limit (h // f) x0 B
  -> limit (h // g) x0 B.
```

这便是重要的"等价无穷小替换"求解极限法则.

类似地, 也可以定义无穷大量, 并且可对不同无穷大量进行比较, 思路与无穷小量一致, 这里不再详述.

此外, 关于极限理论, 借助一些级数相关概念, 可以形式化描述和验证两个特殊极限:

$$\lim_{x\to 0}\frac{\sin(x)}{x}=1,\ \lim_{n\to\infty}\left(1+\frac{1}{n}\right)^n=\mathrm{e},$$

具体可参见文献 [54], 本文不再展开.

5.3 函数连续性

定义 9. 函数连续. 设函数 f 在点 x_0 的某邻域 $U(x_0;\delta)$ 有定义.若 $\lim\limits_{x\to x_0}f(x)=f(x_0)$, 则称函数 f 在 x_0 处连续.

注意在函数极限的形式化定义中, 已经含有“f 在点 x_0 的某空心邻域 $U^{\circ}(x_0;\delta)$ 有定义”的条件, 因此在函数连续的定义中, 只需再加入 x_0 属于 f 定义域的条件.

```
Definition Continuous f x0 := x0 ∈ dom[f] /\ limit f x0 f[x0].
```

关于函数在某一点处的连续性, 还有另一种表述方式. 记 $\Delta x = x - x_0$, 称为自变量 x 在点 x_0 处的增量, 相应函数 $f(x)$ 在 x_0 处增量记为:

$$\Delta y = f(x) - f(x_0) = f(x_0 + \Delta x) - f(x_0),$$

则函数 f 在 x_0 处连续等价于:

$$\lim_{\Delta x \to 0} \Delta y = 0.$$

以下给出这种增量法定义的形式化描述:

```
Definition Continuous' f x0 := Function f /\ (∃ δ, 0 < δ /\ U(x0; δ) ⊂ dom[f])
  /\ limit \{\ λ δx δy, δy = f[x0 + δx] - f[x0]\}\ 0 0.
```

需注意该形式化定义与函数连续的原始定义之间条件的细微区别, 但可证明二者是完全等价的.

```
Corollary EquaContinuous : ∀ f x0, Continuous f x0 <-> Continuous' f x0.
```

可以看到, 函数在一点处连续, 就是函数在该点处的极限值等于函数在该点处的具体取值, 本质上是对极限概念的一种延伸. 类似可通过函数左、右极限的概念定义函数的左、右连续性.

```
Definition ContinuousLeft f x0 := x0 ∈ dom[f] /\ limit_neg f x0 f[x0].
Definition ContinuousRight f x0 := x0 ∈ dom[f] /\ limit_pos f x0 f[x0].
```

函数在点 x_0 处连续的一个充要条件是: 函数在 x_0 处既是左连续的也是右连续的.

```
Theorem Theorem4_1 : ∀ f x0, Continuous f x0 <-> (ContinuousLeft f x0 /\ ContinuousRight f x0).
```

若函数 f 在 x_0 的某邻域有定义, 但在 x_0 处无定义或不连续, 则称 x_0 为函数 f 的间断点. 间断点可根据函数在该点处的极限进行分类.

(1) 若 $\lim\limits_{x \to x_0} f(x) = A$, 而 f 在点 x_0 处无定义或有定义但 $f(x_0) \neq A$, 则称 x_0 为 f 的可去间断点;

```
Definition RemoveDiscontinuous f x0 := ∃ A, limit f x0 A
  /\ (x0 ∉ dom[f] \/ (x0 ∈ dom[f] /\ f[x0] <> A)).
```

(2) 若函数 f 在 x_0 处的左右极限均存在但不相等, 则称 x_0 为 f 的跳跃间断点.

```
Definition secondDiscontinuous f x0 := (∀ A, ~ limit_pos f x0 A) \/ (∀ B, ~ limit_neg
f x0 B).
```

以上两种间断点因函数在 x_0 处的左、右极限均存在, 统称为第 1 类间断点.

```
Definition firstDiscontinuous f x0 := RemoveDiscontinuous f x0 \/ LeapDiscontinuous f x0.
```

若函数 f 在 x_0 处的左极限或右极限至少有一个不存在, 则称 x_0 为 f 的第 2 类间断点.

```
Definition secondDiscontinuous f x0 := (∀ A, ~ limit_pos f x0 A) \/ (∀ B, ~ limit_neg
f x0 B).
```

基于函数在某一点处的连续性, 可以定义函数在区间上的连续性. 例如, 函数在开区间 (a,b) 上连续, 就是函数在区间内的每一点处都连续.

```
Definition ContinuousOpen f a b := ∀ x, a < x < b -> Continuous f x.
```

函数在闭区间 $[a,b]$ 上连续, 就是函数在开区间 (a,b) 上连续, 并且在 a 处右连续, 在 b 处左连续.

```
Definition ContinuousCO f a b:= ContinuousOpen f a b /\ ContinuousRight f a /\ ContinuousLeft f b.
```

类似可定义函数在区间 $[a,b)$ 和 $(a,b]$ 的连续性.

关于函数在区间上的连续性, 有一个重要概念, 一致连续性.

定义 10. 一致连续. 设函数 f 在区间 I 上有定义.若对任给的 $\varepsilon > 0$, 存在正数 δ, 使得对任何 $x_1, x_2 \in I$, 只要 $|x_1 - x_2| < \delta$, 就有:

$$|f(x_1)-f(x_2)|<\varepsilon,$$

则称函数 f 在区间 I 上一致连续.

```
Definition UniCon f I := Function f /\ I ⊂ dom[f]
  /\ (∀ ε, ε > 0
    -> (∃ δ, δ > 0 /\ (∀ x1 x2, x1 ∈ I -> x2 ∈ I -> |(x1 - x2)| < δ -> |(f[x1] - f[x2])| < ε))).
```

该定义的形式化描述中, I 仅设定为函数定义域的子集, 在具体应用该定义时只需将 I 具体化为区间即可. 例如, 关于函数的一致连续性有一个重要定理 (一致连续性定理): 若函数 f 在闭区间 $[a,b]$ 上连续, 则 f 在 $[a,b]$ 上一致连续.

```
Theorem Theorem4_9 : ∀ f a b, ContinuousClose f a b -> UniCon f (\[a, b\]).
```

形式化描述中使用闭区间\[a, b\] 对参数 I 进行具体化表达.

5.4 导数与微分

导数与微分概念均是基于极限理论而给出.

定义 11. 导数. 设函数 f 在点 x_0 的某邻域上有定义, 若极限

$$\lim_{x\to x_0}\frac{f(x)-f(x_0)}{x-x_0}$$

存在, 则称函数 f 在点 x_0 处可导, 并称该极限为函数 f 在 x_0 处的导数, 记为 $f'(x_0)$.

```
Definition Derivative f x0 y0' := Function f /\ (∃ δ', δ' > 0 /\ U(x0; δ') ⊂ dom[f])
  /\ (limit \{\ λ x y, y = (f[x] - f[x0])/(x - x0) \}\ x0 y0').
```

若将 $x-x_0$ 记为 Δx, 则导数定义中的极限式可改写为:

$$\lim_{\Delta x\to 0}\frac{\Delta y}{\Delta x}=\lim_{\Delta x\to 0}\frac{f(x_0+\Delta x)-f(x_0)}{\Delta x},$$

相应的形式化定义与原定义完全等价.

```
Definition Derivative' f x0 y0' := Function f /\ (∃ δ', δ' > 0 /\ U(x0; δ') ⊂ dom[f])
  /\ (limit \{\ λ δx y, y = (f[x0 + δx] - f[x0])/δx \}\ 0 y0').
Corollary EquaDerivative : ∀ f x0 y0', Derivative f x0 y0' <-> Derivative' f x0 y0'.
```

导数形式化定义"Derivative f x0 y0'"的含义为: 函数 f 在 x0 处的导数值为 y0'. 为与导数值的数学符号一致, 可以通过取元操作将导数值 y0'取出, 并引入相应记号.

```
Definition DerivativeValue f x := c \{ λ y', Derivative f x y' \}.
Notation "f '[ x ]" := (DerivativeValue f x)(at level 20).
Corollary Corollary_DerivativeValue_a : ∀ f x y', Derivative f x y' -> f'[x] = y'.
```

类似函数的左、右连续性, 可根据导数定义中极限式值的左、右存在性, 区分函数的左导数和右导数, 并且函数在 x_0 处可导的一个充要条件为: 函数在 x_0 处既是左可导, 也是右可导的. 具体实现不再赘述.

函数 f 的导函数是将 f 的每一个可导点 x 映射为 f 在 x 处导数值的函数.

```
Definition DerivativeFun f := \{\ λ x y, Derivative f x y \}\.
Corollary Derivative_is_Fun : ∀ f, Function (DerivativeFun f).
```

定义 12. 二阶导数. 若函数 f 的导函数 f' 在 x_0 处可导, 则称 f' 在 x_0 的导数为 f 在 x_0 的二阶导数, 记为 $f''(x_0)$.

$$f''(x_0)=\lim_{x\to x_0}\frac{f'(x)-f'(x_0)}{x-x_0},$$

也称 f 在 x_0 处是二阶可导的.

可以看出, 函数的二阶导数是根据其一阶导函数进行定义的, 依次可递归地定义函数的 n 阶导数. 函数 f 在 x_0 的 n 阶导数就是函数的 $n-1$ 阶导函数在 x_0 处的导数, 即

$$\lim_{x\to x_0}\frac{f^{(n-1)}(x)-f^{(n-1)}(x_0)}{x-x_0}=f^n(x_0),$$

其中, $f^n(x)$ 表示函数的 n 阶导数.

对于二阶及以上的高阶导数, 形式化中可直接递归地给出高阶导函数的定义.

```
Fixpoint dN f (n : nat) :=
  match n with
  | O => f
  | S m => \{\ λ x y, Derivative (dN f m) x y \}\
  end.
```

该定义以 dN 命名, 参数 f 用以表示被导函数, n 为导函数阶数. 若 n 为 0, 定义输出原函数 f, 表示不进行求导(0 阶导函数); 当 n 为某个自然数的后继, 即具有 S m 形式时, 定义输出 m 阶导函数"dN f m"的导函数. 如此便可递归地定义任意阶数的导函数.

可通过几个简单事实验证上述高阶导数形式化定义的合理性: (1) 按以上定义得到的 dN f n 是一个函数; (2) n 阶导函数的定义域包含于 f 的定义域; (3) $n+1$ 阶导函数的取值即为 n 阶导数值.

```
Fact dN_Fact1 : ∀ f n, Function f -> Function (dN f n).
Fact dN_Fact2 : ∀ f n, dom[dN f n] ⊂ dom[f].
Fact dN_Fact3 : ∀ x n f, (dN f (S n))[x] = (dN f n)'[x].
```

定义 13. 微分. 设函数 f 在点 x_0 的某邻域 $U(x_0;\delta)$ 上有定义.当给 x_0 一个增量 Δx 并且 $x_0+\Delta x\in U(x_0;\delta)$ 时, 相应可得函数增量为 $\Delta y=f(x_0+\Delta x)-f(x_0)$. 若存在常数 A, 使得 Δy 能表示成:

$$\Delta y=A\Delta x+o(\Delta x),$$

其中, $o(\Delta x)$ 表示增量 Δx 的高阶无穷小, 则称函数 f 在点 x_0 处可微, 并称上式中的 $A\Delta x$ 为 f 在 x_0 处的微分.

仔细分析可知, 以上定义的陈述中包含了一个邻域 $U(x_0;\delta)$、一个常数 A 以及一个高阶无穷小 o 的存在性, 因此在形式化定义中需要以存在量词给出这 3 个参数, 并且其中 o 实际应该是线性函数 $f(x)=x$ 的高阶无穷小. 形式化定义如下:

```
Definition Differentiable f x0 := Function f
  /\ (∃ δ o A, δ > 0 /\ U(x0; δ) ⊂ dom[f] /\ InfinitesimalHigherOrder o \{\ λ x y, y = x \}\ 0
    /\ (∀ δx, (δx + x0) ∈ U(x0; δ) -> f[x0 + δx] - f[x0] = A * δx + o[δx])).
```

可以证明, 一元函数 f 在点 x_0 处的可导性与可微性等价, 并且微分定义中的常数 A 等于导数值 $f'(x_0)$.

```
Theorem Theorem5_9a : ∀ f x0, Differentiable f x0 -> Derivable f x0.
Theorem Theorem5_9b : ∀ f x0, Derivable f x0 -> Differentiable f x0
  /\ (∃ δ o, δ > 0 /\ U(x0; δ) ⊂ dom[f] /\ InfinitesimalHigherOrder o \{\ λ x y, y = x \}\ 0
    /\ (∀ δx, (δx + x0) ∈ U(x0; δ) -> f[x0 + δx] - f[x0] = f'[x0] * δx + o[δx])).
```

因此, 可将函数 f 在 x 处的微分记为 $\mathrm{d}y=f'(x)\Delta x$. 并且, 由于对于恒等函数 $f(x)=x$ 来说, 其微分为 $\mathrm{d}y=\mathrm{d}x=\Delta x$, 因此函数自变量的微分 $\mathrm{d}x$ 等于其增量 Δx, 这使得函数微分可记为 $\mathrm{d}y=f'(x)\mathrm{d}x$.

```
Definition d f x dx := (f'[x] * dx).
```

实际上人工定义中的 y 并不作为表达式中的参数出现, 因而形式化记法中无需引入 y.

根据以上微分表达式, 可以递归地推导出函数高阶微分的表达式. 函数的 n 阶微分是 $n-1$ 阶微分的微分, 记为 $\mathrm{d}^n y$, 于是有:

$$\mathrm{d}^n y=\mathrm{d}(\mathrm{d}^{n-1}y)=\mathrm{d}(f^{(n-1)}(x)(\mathrm{d}x)^{(n-1)})=f^{(n)}(x)(\mathrm{d}x)^n,$$

一般地, 将 $(\mathrm{d}x)^n$ 写为 $\mathrm{d}x^n$, 于是有:

$$\mathrm{d}^n y=\mathrm{d}(f^{(n-1)}(x)\mathrm{d}x^{(n-1)})=f^{(n)}(x)\mathrm{d}x^n.$$

上式中的 $\mathrm{d}(f^{(n-1)}(x)\mathrm{d}x^{(n-1)})$ 实际为函数 $h(x)=f^{(n-1)}(x)\mathrm{d}x^{(n-1)}$ 整体的微分 (注意 $\mathrm{d}x$ 与 x 无关), 因此高阶微分可递归地形式化定义为:

```
Fixpoint dn f n x dx :=
  match n with
  |0 => f[x]
  |S k => d \{\ λ u v, v = dn f k u dx \}\ x dx
  end.
```

本节内容主要介绍极限、导数与微分中核心重点概念的形式化实现, 实际上, 还有更多相关概念也已在本文形式化系统中实现, 并严格与教材[35]对应, 包括函数的极值与最值、函数的凹凸性与拐点、极限与导数的四则运算、微分的运算法则等, 相关内容不便继续展开, 详情可参考文末提供的形式化系统代码.

6 不定积分与黎曼积分的形式化

6.1 不定积分

定义 14. 原函数. 设函数 f 与 F 在区间 I 上都有定义. 若

$$F'(x)=f(x), x\in I,$$

则称 F 为 f 在区间 I 上的一个原函数.

严格来说, 以上原函数的定义根据具体区间 I 的不同应当略有区别. 例如, 若 I 为一个闭区间 $[a,b]$, 则应当只要求 F 在 a 处的右导数为 $f(a)$, 而在 b 处的左导数为 $f(b)$ 即可. 因此, 形式化描述中对于开区间和闭区间上的原函数定义有所区分.

```
Definition Primitive_Open f F a b := Function f /\ Function F /\ a < b
  /\ \(a, b\) ⊂ dom[f] /\ \(a, b\) ⊂ dom[F] /\ (∀ x, x ∈ \(a, b\) -> Derivative F x f[x]).
Definition Primitive_Close f F a b := Function f /\ Function F /\ a < b
  /\ \[a, b\] ⊂ dom[f] /\ \[a, b\] ⊂ dom[F] /\ (∀ x, x ∈ \(a, b\) -> Derivative F x f[x])
  /\ Derivative_pos F a f[a] /\ Derivative_neg F b f[b].
```

类似也可给出左闭右开区间 $[a,b)$ 和左开右闭区间 $(a,b]$ 上原函数的精准形式化描述.

定义 15. 不定积分. 函数 f 在区间 I 上的全体原函数则称为 f 在 I 上的不定积分, 记为:

$$\int f(x)\mathrm{d}x,$$

不定积分与原函数是总体与个体的关系, 即函数 f 的不定积分实际为 f 的所有原函数构成的集合. 此外, 类似原函数的形式化, 定义中需要区分区间 I 的具体形式. 以下为开区间和闭区间上不定积分的形式化定义.

```
Definition Indef_Integral_Open f a b := \{ λ F, Primitive_Open f F a b \}.
Notation "∫( f ; a ; b )" := (Indef_Integral_Open f a b).
Definition Indef_Integral_Close f a b := \{ λ F, Primitive_Close f F a b \}.
Notation "∫[ f ; a ; b ]" := (Indef_Integral_Open f a b).
```

由于不定积分实际上被定义为原函数组成的集合, 具体定理证明的应用中, 形式化描述与人工数学符号化的描述略有区别, 以不定积分求解法则中的分部积分为例给与说明.

定理 2. 分部积分法. 若函数 u, v (在区间 (a,b) 上) 可导, 不定积分 $\int u'(x)v(x)\mathrm{d}x$ 存在, 则 $\int u(x)v'(x)\mathrm{d}x$ 也存在, 且

$$\int u(x)v'(x)\mathrm{d}x=u(x)v(x)-\int u'(x)v(x)\mathrm{d}x.$$

经分析, 以上定理含有 3 个要点.

(1) 函数 u, v 在区间 (a,b) 上均有定义, 并且可导;

(2) 不定积分 $\int u'(x)v(x)\mathrm{d}x$ 存在实际上是指函数 $u'(x)v(x)$ 在区间 (a,b) 上存在原函数;

(3) 等式表明: 对于任意一个 $u'(x)v(x)$ 的原函数 (设为 $G(x)$), 函数 $u(x)v(x)-G(x)$ 都是 $u(x)v'(x)$ 的原函数.

从而该定理的形式化描述应按如下方式给出:

```
Theorem Theorem8_6 : ∀ u v G a b, a < b -> \(a, b\) ⊂ dom[u] -> \(a, b\) ⊂ dom[v]
  -> (∀ t, t ∈ \(a, b\) -> Derivable u t /\ Derivable v t)
  -> Primitive_Open \{\ λ x y, y = u'[x] * v[x] \}\ G a b
  -> Primitive_Open \{\ λ t y, y = u[t] * v'[t] \}\ ((u \* v) \- G) a b.
```

该定理在教材[35]中被编号为“定理 8.6”, 故形式化中命名为“Theorem8_6”.

6.2 黎曼积分

定义 16. 分割. 设闭区间 $[a,b]$ 上有 $n-1$ 个点, 依次为:

$$a=x_0<x_1<\ldots<x_{n-1}<x_n=b,$$

它们把 $[a,b]$ 分成 n 个小区间 $\Delta_i=[x_{i-1},x_i]$, $i=1,2,\ldots,n$.这些分点或这些子区间构成对 $[a,b]$ 的一个分割, 记为:

$$T=\{x_0,x_1,\ldots,x_{n-1},x_n\}\text{ 或 }\{\Delta_0,\Delta_1,\ldots,\Delta_n\}.$$

小区间 Δx_i 的长度为 $\Delta x_i=x_i-x_{i-1}$, 并记:

$$\|T\|=\max_{1\leqslant i\leqslant n}\{\Delta x_i\},$$

即 n 个子区间的最大长度, 称为分割 T 的模.

根据定义, 分割 T 可利用一个有限且数值严格递增的点列 $\{\xi_i\}(i=0,1,2,\ldots,n)$ 来表示, 并且 $x_0=a$ 为被分割区间的左端, $x_n=b$ 为被分割区间的右端, 因而可通过函数表达 T 的这些性质, 即 T 为一个严格单增的函数, $T(i)$ 则表示点列中的点 x_i.

```
Definition Seg T a b n := Function T /\ (0 < n)%nat /\ dom[T] = \{ λ u, (u <= n)%nat \}
  /\ (∀ k, (k < n)%nat -> T[k] < T[S k]) /\ T[0%nat] = a /\ T[n] = b.
```

分割 T 的模也就是 $T(i)-T(i-1)$ 的最大值.

```
Definition SegMold T n x := (∃ k, (k < n)%nat /\ T[S k] - T[k] = x)
  /\ (∀ m, (m < n)%nat -> T[S m] - T[m] <= x).
Definition mold T n := c \{ λ x, SegMold T n x \}.
```

形式化描述中, 首先以参数 x 表示分割 T 的模, 随后通过 c 将该值取出, 以便在相关运算表达式中直接调用.

定义 17. 黎曼和. 设函数 f 在闭区间 $[a,b]$ 上有定义. 对于 $[a,b]$ 的一个分割 $T=\{\Delta_0,\Delta_1,\ldots,\Delta_n\}$, 任取点 $\xi_i\in\Delta_i$, $i=1,2,\ldots,n$, 并作和式:

$$\sum_{i=1}^{n}f(\xi_i)\Delta x_i,$$

称此和式为函数 f 在 $[a,b]$ 上的一个积分和, 也称黎曼和.

形式化定义积分和涉及两个要点, 一是对点列 $\{\xi_i\}$ 的描述, 二是对 Σ 求和的描述.

对于分割 T, 点列 $\{\xi_i\}$ 需满足 $T(i-1)\leqslant\xi_i\leqslant T(i)$, 即 $\xi_i\in\Delta_i$. 为充分利用形式化系统中的自然数结构, 也可写为 $T(i)\leqslant\xi_i\leqslant T(i+1)$, 此时 $i+1$ 也是 i 的后继.

```
Definition SegPoints T ξ n := (∀ k, (k < n)%nat -> T[k] <= ξ[k] <= T[S k]).
```

求和运算可利用 Coq 自身的递归机制进行定义.

```
Fixpoint Σ (f : Seq) (n : nat) :=
  match n with
  | 0%nat => f[0%nat]
  | S m => Σ f m + f[S m]
```

```
  end.
```

这里的参数 f 是一个序列, 用于表示求和的通项公式, n 为一个自然数, 表示求和的项数. 例如, 求和式 $\sum_{i=1}^{n}\frac{1}{i}$ 可写为"`Σ (\{\ λ i s, s = 1/i \}\) n`".

由此, 黎曼和的形式化直接以一个求和表达式给出:

```
Definition IntegralSum (f : (@set (@prod R R))) T n ξ :=
  Σ \{\ λ i s, s = f[ξ[i]] * (T[S i] - T[i]) \}\ n.
```

定义中的 f 表示被求和函数, T 为分割, n 为分割点的数量, 也是积分和的求和项数, ξ 表示积分和中的点列.

定义 18. 黎曼积分. 设函数 f 在闭区间 $[a,b]$ 上有定义, J 为一实数. 若对任给正数 ε, 总存在某一正数 δ, 使得对 $[a,b]$ 任何分割 T, 以及其上任意的点序列 $\{\xi_i\}$, 只要 $\|T\|<\delta$, 就有:

$$\left|\sum_{i=1}^{n} f(\xi_i)\Delta x_i - J\right| < \varepsilon,$$

则称函数 f 在区间 $[a,b]$ 上 (黎曼) 可积; 数 J 称为函数 f 在 $[a,b]$ 上的定积分或黎曼积分, 记作:

$$J = \int_a^b f(x)\mathrm{d}x.$$

以上定义中的 $\sum_{i=1}^{n} f(\xi_i)\Delta x_i$ 实际上就是黎曼和, 可以直接通过"`IntegralSum f T n ξ`"形式化表达. 由于黎曼和的形式化定义中, 求和的最后一项中含有"`T[S n]`", 这相当于将区间分割为了 $n+1$ 段, 因此在形式化描述中需格外注意相关参数的具体表达. 例如这里的 $\|T\|$ 表示分割 T 的模, 形式化中需用"`mold T (S n)`"表达, 而不能使用"`mold T n`". 黎曼积分的完整形式化定义如下:

```
Definition Def_Integral f a b J := Function f /\ a < b /\ \[a, b\] ⊂ dom[f]
  /\ ∀ ε, 0 < ε -> (∃ δ, 0 < δ
    /\ (∀ T ξ n, Seg T a b (S n) -> SegPoints T ξ (S n) -> (mold T (S n) < δ)
      -> |((IntegralSum f T n ξ) - J)| < ε)).
```

由于定积分数学符号中的 a 和 b 均以上下标的非标准字体出现, 无法引入与之保持一致的形式化记号. 实际上, 形式化表达式"`Def_Integral f a b J`"的含义已经清楚: f 在 $[a,b]$ 上的定积分为 J, 无需专门符号化.

6.3 达布和与可积性

在教材[35]中, 达布和与可积性理论作为可选学章节给出; 在 Coq 官方标准库中的数学分析部分, 也未实现达布和与可积性相关内容的形式化. 实际上, 达布和与可积性是对黎曼积分可积条件的重要补充, 这里给出教材中这部分相关概念的形式化.

定义 19. 达布和. 设 T 为区间 $[a,b]$ 上的分割, 函数 f 在 $[a,b]$ 上有界, 则 f 在分割的各小区间 Δ_i 存在上、下确界:

$$M_i = \sup_{x\in\Delta_i} f(x),\ m_i = \inf_{x\in\Delta_i} f(x),\ i = 1,2,\ldots,n,$$

作和

$$S(T) = \sum_{i=1}^{n} M_i\Delta x_i,\ s(T) = \sum_{i=1}^{n} m_i\Delta x_i,$$

分别称为 f 关于分割 T 的 (达布) 上和与 (达布) 下和, 统称为达布和.

根据定义, 需要首先给出函数上确界与下确界的形式化描述. 本文第 4.1 节中已经给出了集合上、下确界的形式化描述, 而函数的确界实际就是函数值域的确界, 函数在某个集合 (区间也是集合) D 上的确界, 则是函数限制在 D 之后值域的确界.

```
Definition sup_fD f (D : @set R) := c (\{ λ u, sup (ran[f|[D]]) u \}).
Definition inf_fD f (D : @set R) := c (\{ λ u, inf (ran[f|[D]]) u \}).
```

为方便求和运算的调用, 形式化代码中直接通过“取元操作”将函数的确界取出.

进而, 上和与下和的表达式形式化描述如下:

```
Definition Up_sum f (T : Seq)(n : nat) :=
  (Σ \{\ λ i s, s = (sup_fD f (\[T[i], T[S i]\])) * (T[S i]- T[i]) \}\ n).
Definition Low_sum f (T : Seq)(n : nat) :=
  (Σ \{\ λ i s, s = (inf_fD f (\[T[i], T[S i]\])) * (T[S i]- T[i]) \}\ n).
```

关于达布和有如下函数可积性的判别准则.

定理 3. 可积准则. 函数 f 在区间 $[a,b]$ 上可积的充要条件是: 任给 $\varepsilon>0$, 总存在相应分割 T, 使得:

$$S(T)-s(T)<\varepsilon.$$

可积准则中的一个隐含细节为: f 在 $[a,b]$ 上有界, 否则无法讨论其上确界与下确界, 更无上和与下和可言. 实际上, 函数 f 在 $[a,b]$ 上有界是函数 f 在 $[a,b]$ 上可积的必要条件. 因此, 可积准则的严格形式化描述为:

```
Theorem Theorem9_3a : ∀ f a b, a < b -> (∃ J, Def_Integral f a b J)
  <-> ((DBoundedFun f (\[a, b\]))
    /\ (∀ δ, δ > 0 -> ∃ T n, Seg T a b (S n) /\ (Up_sum f T n) - (Low_sum f T n) < δ)).
```

可积准则的证明依赖于对上和与下和性质的详尽讨论, 相关形式化细节, 包括 6 条达布和与积分相关性质、以及 3 个可积充要条件的形式化描述及证明, 可进一步参见本文提供的形式化代码中的 A_9_6 文件, 对应于教材[35]第 9.6 章中的内容.

7 定理证明实例

本节选取教材[35]中微积分学的几个重要定理, 包括泰勒展开公式、牛顿-莱布尼茨公式以及微积分学基本定理 (亦即原函数存在定理), 介绍其人工证明的形式化实现.

7.1 证明策略与交互式证明

在定理证明器中, 一个定理的证明过程通常是由一系列被称为“证明策略 (tactic)”的命令构造而成, Coq 也不例外. 这些命令可以实时编写实时运行, 用户根据交互式窗口的证明状态可以选择下一步应该选用何种证明策略. 下面以一个简单的逻辑命题为例, 简要介绍这种交互式证明过程.

```
Theorem prop_trans : forall (P Q R : Prop), (P -> Q) /\ (Q -> R) -> (P -> R).
Proof. intros. destruct H. apply H in H0; apply H1 in H0. apply H0. Qed.
```

该命题表述了蕴含式的传递性, 其证明主要由 3 个证明策略构成: intros、destruct 以及 apply, 每条指令以句点结束, 分号表示策略的连用. 将代码运行至 intros 可在交互式窗口看见如下信息:

```
P, Q, R : Prop
H : (P -> Q) /\ (Q -> R)
H0 : P
_____________________________________(1/1)
R
```

此时 intros 策略将命题的所有变量和前提假设引入证明环境的条件中 (横线上方), 剩余的 R 为证明目标. 条件中的 H 与 H0 是为相应条件的临时命名, 通常可由 Coq 系统自动分配. 显然, 根据现有的条件, 可以考虑将 H 应用至 H0, 即可推出 R. 为此, 先使用 destruct 策略将合取式 H 分解为两个条件:

```
P, Q, R : Prop
H : P -> Q
H1 : Q -> R
H0 : P
```

```
_____________________________________(1/1)
R
```

进而连用两个 apply 策略, 依次将条件 H 与 H1 应用于 H0, 从而使得 H0 的类型变为 R:

```
P, Q, R : Prop
H : P -> Q
H1 : Q -> R
H0 : R
_____________________________________(1/1)
R
```

最后直接对目标使用 apply 策略, 匹配 H0, 即可消去目标, 完成证明.

为简化证明, Coq 内部引入了一些集成策略, 可以批处理一些简单的证明. 例如 auto 策略, 是对 intros 和 apply 等策略的集成和连续使用, 对于上例的证明, 也可以通过 auto 策略进行简化:

```
Theorem prop_trans : forall (P Q R : Prop), (P -> Q) /\ (Q -> R) -> (P -> R).
Proof. intros. destruct H. auto. Qed.
```

针对自然数与实数的线性运算, Coq 也分别设计了 lia 和 lra 策略, 可自动证明简单的线性运算式, 两个策略可分别通过读入 Coq 标准库中的 micromega.Lia 和 micromega.Lra 模块得到.

此外, Coq 允许开发者自行编写集成策略. 我们在本文的微积分系统中即引入了 applyClassifier1 和 applyClassfier2 两个简单的集成策略, 用于将形如“`_ ∈ \{ _ \}`”的条件转换为具体命题, 前者用于单一元素, 后者用于序偶对. 例如, 对“`H : a ∈ \{ λ u, 1 <= u \}`”使用“`applyClassifier1 H`”可得到“`H : 1 <= a`”; 对“`H : [a, b] ∈ \{ λ u v, u <= v \}`”使用“`applyClassifier2 H`”可得到“`H : a <= b`”.

证明策略是用户与 Coq 交互的直接媒介, 策略的集成则体现了 Coq 的自动化与智能性. 表 4 列出本文系统中的常用证明策略及其含义, 更多策略及细节用法可参见文献 [13,14,43].

表 4　常用证明策略及用法

策略名	含义及用法
intro / intros	引入目标中的单个/多个前提至证明条件中
apply H / apply H in H1	对目标/条件H1应用H (H可以是已证定理)
destruct H	拆分条件H中的析取式或合取式; 将存在量词实例化(H可以是已证定理)
split	拆分目标的合取式使之成为两个子目标
rewrite H / rewrite H in H1	重写策略, 利用等式H对目标/条件H1中相应变量进行等量替换
exists a	为目标中的存在量词指定具体值a
left / right	目标为析取式时得到析取式的左边/右边作为子目标
assumption	遍历当前证明上下文寻找与目标相同的条件并解决目标
reflexivity	解决形如“a=a”的等式目标
elim H	对含有归纳定义的条件H进行归纳或分解
induction	对含有归纳定义的条件H进行归纳或分解, 并且引入归纳假设
assert (…)	断言括号中的内容并证明, 使其成为一个条件
pose proof H	生成一个已有条件H或已证命题
unfold A / unfold A in H	展开目标/条件H中的定义A
red / red in H	展开目标/条件H中的最外层定义
repeat T	重复执行策略T直至成功或失败
auto	自动重复执行assumption、intros、apply等策略
lia / lra	自动证明自然数/实数的线性算式
applyClassfier1 H	用于形式为“`a ∈ \{ λ u, P u \}`”的条件H, 使其成为P a的具体形式
applyClassfier2 H	用于形式为“`[a, b] ∈ \{ λ u v, P u v \}`”的条件H, 使其成为P a b的具体形式

7.2 泰勒公式

泰勒公式是微积分学中的重要公式之一, 数学上可用于极限式的求解, 工程上也是数值近似的理论基础. 首先引用教材[35]中这一公式的人工描述.

定理 4. 佩亚诺余项的泰勒展开公式. 若函数 f 在点 x_0 存在直至 n 阶导数, 则有 $f(x)=T_n(x)+o((x-x_0)^n)$, 即:

$$f(x)=f(x_0)+f'(x_0)(x-x_0)+\frac{f''(x_0)}{2!}(x-x_0)^2+\ldots+\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n+o((x-x_0)^n),$$

其中, $o((x-x_0)^n)$ 表示函数 $y=(x-x_0)^n$ 在 x_0 处的高阶无穷小量, 称为佩亚诺余项, $T_n(x)$ 称为泰勒多项式.

该定理实际上只需要证明 $f(x)-T_n(x)$ 为函数 $y=(x-x_0)^n$ 在 x_0 处的高阶无穷小, 而 $f(x)-T_n(x)$ 即为佩亚诺余项, 为此证明该定理的核心在于对泰勒多项式进行严格形式化描述并验证其相关性质.

泰勒多项式实际上是对通项 $\frac{f^{(n)}(x_0)}{n!}(x-x_0)^n$ 的有限次求和, 求和公式的形式化已在本文第 6.2 节中介绍, 下面分别给出阶乘与自然数指数运算的形式化.

```
Fixpoint PowR r n :=
  match n with
  | O => 1
  | S n => r * (PowR r n)
  end.
Notation "r ^ n" := (PowR r n).
Fixpoint Factorial (n : nat) :=
  match n with
  | 0%nat => 1%nat
  | S n => ((S n) * Factorial n)%nat
  end.
Notation "n !" := (Factorial n).
```

两种运算均可由递归方式给出定义, 并引入与习惯数学符号一致的形式化记号. 需要注意的是, 阶乘运算是对自然数 nat 类型的运算, 当运算结果与实数进行再运算时需使用 INR 转换 (见本文第 4.3 节所述).

定理中涉及 f、x_0 和 n 这 3 个参数, 其在泰勒多项式相关性质的验证中均会出现, 因而可将整个定理在一个 Coq 的局部块 (section) 中进行. 我们在代码中以“Section taylor_equation_P”开启这个局部块, 其中“taylor_equation_P”为该局部块的命名, 随后使用 Variable 和 Hypothesis 命令引入 f、x_0 和 n 这 3 个局部变量, 分别代表定理中的 f、x_0 和 n, 并假定其满足的条件:

```
Variable f : @set (@prod R R).
Variable n : nat.
Variable x0 : R.
Hypothesis Function_f : Function f.
Hypothesis f_n_Derivable : (n = 0%nat /\ Derivable f x0)
  \/ ((0 < n)%nat /\ (∀ k, (k < n)%nat -> Derivable (dN f k) x0)).
```

其中, 前 3 条指令引入了局部变量 f、x_0 和 n, 后两条指令假定了 f 为一个在 x_0 处 n 阶可导的函数.

局部块中的局部定义与局部定理均可使用 Let 命令进行描述. 例如, 可先使用 Let 定义泰勒多项式的通项 (设为第 k 项).

```
Let kst k := \{\ λ u v, v = (dN f k)[x0]/(INR (k!)) * (u - x0)^k \}\.
```

随后可基于此定义 k 次泰勒多项式 $T_k(x)$, 该定义为一个 Σ 求和式:

```
Let T_poly x k := Σ \{\ λ u v, v = (kst u)[x] \}\ k.
```

函数 f 的 n 次佩亚诺余项为函数 f 减去 n 次泰勒多项式:

```
Let P_remainder := f \- \{\ λ u v, v = T_poly u n \}\.
```

可以验证, 佩亚诺余项的任意 m 次 $(m<n)$ 导函数均为 x_0 处的无穷小:

```
Let P_remainder_Infsimal : Infinitesimal P_remainder x0
  /\ (∀ m, (m < n)%nat -> Infinitesimal (dN P_remainder m) x0).
```

使用 Variable、Hypothesis 和 Let 命令引入的局部变量、假设以及描述的定义与定理仅在局部块中有效, 不会对块之外的证明环境产生影响.

在结束该局部块之前, 我们使用 Theorem 命令验证泰勒公式的核心要点: 佩亚诺余项为 $y=(x-x_0)^n$ 在 x_0 处的高阶无穷小, 同时 $f(x)$ 等于泰勒多项式加上佩亚诺余项.

```
Theorem Theorem6_9a : (0 < n)%nat
  -> InfinitesimalHigherOrder P_remainder \{\ λ u v, v = (u - x0)^n \}\ x0.
Theorem Theorem6_9b : ∀ x, x ∈ dom[f] -> f[x] = T_poly x n + P_remainder[x].
```

之后以"End taylor_equation_P"结束该局部块.

使用局部块的优势在于, 对于反复出现的变量, 例如本例中的 f, n, x_0, 可以统一引入, 省去了在每个命题中写入的麻烦; 同时, 使用 Let 命令描述的定义和定理不会对外部证明环境产生影响, 结构上更为清晰.

最后, 为与教材中的人工描述保持一致, 我们以如下形式化完成泰勒公式的描述及验证.

```
1 Theorem Theorem6_9 : ∀ f n x0, Function f -> (0 < n)%nat
2 -> (∀ k, (k < n)%nat -> Derivable (dN f k) x0)
3 -> (∃ o, dom[o] = dom[f] /\ InfinitesimalHigherOrder o \{\ λ x y, y = (x - x0)^n \}\ x0
4 /\ (∀ x, x ∈ dom[f]
5 -> f[x] = (Σ \{\ λ k v, v = (dN f k)[x0]/(INR (k!)) * (x - x0)^k \}\ n) + o[x])).
6 Proof.
7 intros. pose proof H0. apply (Theorem6_9a f H n x0) in H2; auto.
8 exists (f \- \{\ λ u v, v = Σ \{\ λ u0 v0, v0 = \{\ λ u1 v1, v1
9 = (dN f u0)[x0]/(INR (u0!)) * (u1 - x0)^u0 \}\[u] \}\ n \}\).
10 split. rewrite Corollary_sub_fun_c. apply AxiomI; split; intros.
11 applyClassifier1 H3. destruct H3. auto. apply Classifier1; split; auto. apply Classifier1.
12 exists (Σ \{\ λ u0 v0, v0 = \{\ λ u1 v1, v1 = (dN f u0)[x0]/(INR (u0!))
14 * (u1 - x0)^u0 \}\[z] \}\ n). apply Classifier2; auto.
15 split; auto. intros. pose proof H3. apply (Theorem6_9b f n x0) in H3.
16 rewrite Corollary_sub_fun_b, FunValue; auto.
17 assert (\{\ λ k v, v = (dN f k)[x0]/(INR (k!)) * (x - x0)^k \}\
18 = \{\ λ u0 v0, v0 = \{\ λ u1 v1, v1 = (dN f u0)[x0]/(INR (u0!)) * (u1 - x0)^u0 \}\[x] \}\).
19 { apply AxiomI; split; intros; destruct z; applyClassifier2 H5; apply Classifier2.
20 rewrite FunValue; auto. rewrite FunValue in H5; auto. }
21 rewrite <-H5; lra. apply Classifier1.
22 exists (Σ \{\ λ u0 v0, v0 = \{\ λ u1 v1, v1 = (dN f u0)[x0]/(INR (u0!))*(u1-x0)^u0 \}\[x] \}\ n).
23 apply Classifier2; auto.
24 Qed.
```

对应证明过程如下.

第 7、8 行: 通过已证明的 Theorem6_9a 引入佩亚诺余项 $f(x)-T_n(x)$ 是函数 $y=(x-x_0)^n$ 在 x_0 处高阶无穷小的条件, 并为目标中的存在量词指定具体值 (即指定佩亚诺余项可满足定理中的无穷小量 o).

第 9–14 行: 证明函数 f 的定义域与佩亚诺余项的定义域相等, 过程中主要利用集合的外延性公理 (AxiomI), 验证两个集合含有相同的元素.

第 15–23 行: 验证佩亚诺余项为 $y=(x-x_0)^n$ 在 x_0 处的高阶无穷小 (该结论已通过 Theorem6_9a 引入条件, 第 15 行处的“split; auto.”指令可直接消解该子目标), 并用 Theorem6_9b 验证函数 f 等于泰勒多项式加佩亚诺余项.

对于陌生定理或命题, 可通过 Check 命令查看其内容, 通过 Locate 命令确定其所在位置. 例如上述证明细节中涉及的命题 FunValue 未在本文详细说明, 读者可输入并运行“Check FunValue.”和“Locate FunValue.”分别查看其内容和位置.

7.3 牛顿-莱布尼茨公式

牛顿-莱布尼茨公式是连接导数、不定积分与定积分的重要纽带, 是微积分学中的核心定理之一, 这里同样先给出其人工描述.

定理 5. 牛顿-莱布尼茨公式. 若函数 f 在 $[a,b]$ 上连续, 且存在原函数 F, 即对 $x\in[a,b]$ 有 $F'(x)=f(x)$, 则有:

$$\int_a^b f(x)\mathrm{d}x=F(b)-F(a).$$

实际上, 该定理的条件可以减弱: f 不必是 $[a,b]$ 上的连续函数, 只需是可积函数; F 不必在整个闭区间 $[a,b]$ 上可导, 只需在开区间 (a,b) 可导而在 $[a,b]$ 上连续. 我们在形式化中直接给出这种弱版本的描述与证明.

```
1 Theorem Theorem9_1 : ∀ f F a b, (∃ J, Def_Integral f a b J) -> Primitive_Open f F a b
2 -> ContinuousClose F a b -> Def_Integral f a b (F[b] - F[a]).
3 Proof.
4 intros. destruct H as [J].
5 assert (∀ ε, 0 < ε -> |((F[b] - F[a]) - J)| < ε).
6 { intros. destruct H as [H[H3[]]]. pose proof H2. apply H5 in H6 as [δ[]].
7 destruct (Examp1 δ (b-a)) as [δ0[H8[]]]; auto. lra.
8 assert (0 < δ0 <= b - a). lra. apply Seg_Exists in H11 as [T[n[]]]; auto.
9 set (ξ := \{\ λ u v, v = c \{ λ i, T[u] < i < T[S u] /\ Derivative F i f[i]
10 /\ f[i] = (F[T[S u]] - F[T[u]])/(T[S u] - T[u]) \} \}\).
11 assert (∀ k, (k <= n)%nat -> T[k] < ξ[k] < T[S k] /\ Derivative F ξ[k] f[ξ[k]]
12 /\ f[ξ[k]] = (F[T[S k]] - F[T[k]])/(T[S k] - T[k])). { ... }
13 assert (SegPoints T ξ (S n)). { ... }
14 pose proof H11. apply (H7 T ξ) in H15; auto; [ |rewrite <-H12; auto].
15 assert (Σ \{\ λ u v, v = F[T[S u]] - F[T[u]] \}\ n = F[b] - F[a]). { ... }
16 assert (F[b] - F[a] = IntegralSum f T n ξ). { ... }
17 rewrite H17; auto. }
18 pose proof (Abs_Rge0 (F[b] - F[a] - J)) as []. apply Rgt_lt, H2 in H3.
19 exfalso. lra. apply Abs_eq_0 in H3. replace (F[b] - F[a]) with J; auto. lra.
20 Qed.
```

由于代码篇幅较长, “{ ... }”中省略了相应 assert 命令所断言内容的证明细节. 定理的总体证明思路是先将可积函数 f 在 $[a,b]$ 上的积分实例化为 J (由第 4 行代码实现), 再验证 $F(b)-F(a)=J$ (即对第 5 行所断言内容的验证), 具体过程对应如下.

第 6 行: 将目标中的前提 $0<\varepsilon$ 引入证明条件中, 并根据定积分的定义, 存在正数 δ, 使得对任意 $[a,b]$ 上的分割 $T=\{\Delta_0,\Delta_1,\ldots,\Delta_n\}$ 和任意点列 $\{\xi_i\}$, $\xi_i\in\Delta_i$, $i=1,2,\ldots,n$, 并作和式, 只要分割的模 $\|T\|<\delta$, 就有:

$$\left|\sum_{i=1}^{n} f(\xi_i)\Delta x_i - J\right| < \varepsilon.$$

第 7 行: 通过已证命题 Examp1 实例化一个正数 δ_0 使得 $\delta_0 < \delta$ 且 $\delta_0 < b-a$.

第 8 行: 通过已证命题 Seg_Exists 实例化一个以 δ_0 为模的 $[a,b]$ 上的分割 T, 使得 $\|T\| = \delta_0 < \delta$.

第 9–13 行: 选取一个点列 $\{\xi_i\}$, 使得对任意 $k \leqslant n$ 都有:

$$T(k) < \xi_k < T(k+1), F'(\xi_k) = f(\xi_k) = \frac{F(T(k+1)) - F(T(k))}{T(k+1) - T(k)},$$

即 $\xi_k \in \Delta_k$ 并且 $f(\xi_k) = \dfrac{F(T(k+1)) - F(T(k))}{\Delta_{x_k}}$. 该过程是对 f 的原函数 F 在分割 T 的每个小区间上使用拉格朗日中值定理而证明, 拉格朗日中值定理的形式化描述如下.

```
Theorem Theorem6_2 : ∀ f (a b : R), a < b -> ContinuousClose f a b -> (∀ x, a < x < b ->
Derivable f x)
  -> ∃ ξ, a < ξ < b /\ Derivative f ξ ((f[b] - f[a])/(b - a)).
```

可以看到, 使用拉格朗日中值定理所需的前提在当前证明条件中均已存在.

第 14 行: 将已实例化的分割 T 与点列 $\{\xi_i\}$ 代入第 6 行代码所得条件中, 即可得到经过实例化后的不等式:

$$\left|\sum_{i=1}^{n} f(\xi_i)\Delta x_i - J\right| < \varepsilon,$$

此处的点列 $\{\xi_i\}$ 以及 Δx_i 所联系的分割 T 不再由任意量词约束, 而是经过实例化的具体值.

第 15–17 行: 根据第 9–13 行代码的结果验证:

$$\sum_{i=1}^{n} f(\xi_i)\Delta x_i = \sum_{i=1}^{n} F(T(i)) - F(T(i-1)) = F(T(n)) - F(T(0)) = F(b) - F(a),$$

于是第 5 行代码的断言成立.

第 18、19 行: 由于 $|(F(b)-F(a))-J|$ 小于任意正数, 从而 $F(b)-F(a)=J$, 定理成立.

7.4 微积分学基本定理

牛顿-莱布尼茨公式给出了一种可积函数定积分的计算方式, 其结论是基于被积函数存在原函数的假设下. 而微积分学基本定理则保证了连续函数的原函数存在性, 因此也称为原函数存在定理.

定理 6. 原函数存在性. 若函数 f 在 $[a,b]$ 上连续, 则函数 $\phi(x) = \int_a^x f(t)\mathrm{d}t$, $x \in [a,b]$, 在 $[a,b]$ 上可导, 且

$$\phi'(x) = \frac{\mathrm{d}}{\mathrm{d}x}\int_a^x f(t)\mathrm{d}t = f(x),\ x \in [a,b].$$

其中, $\phi(x)$ 是一个以积分上限 x 为自变量的函数, 称为变上限的定积分.

证明该定理首先需要形式化实现变上限定积分的相关概念及性质, 这里涉及两个要点: (1) 变上限积分实际是一种函数, 并且需要两个参数分别作为被积函数 f 和确定的积分下限 a; (2) 当 $a < x$ 时, 函数值为 f 在 $[a,x]$ 上的积分值; 当 $a = x$ 时函数值为 0; 当 $x < a$ 时, 函数值应当为 f 在 $[x,a]$ 上积分值的相反数. 由此, 变上限积分的形式化定义如下.

```
Definition VarUp_Integral_Fun f c := \{\ λ x y, (x < c -> Def_Integral f x c (-y))
  /\ (x = c -> y = 0) /\ (c < x -> Def_Integral f c x y) \}\.
```

参数 f 代表被积函数, 参数 c 表示积分下限.

为检验定义正确性, 可分别验证积分上限 (即自变量 x) 大于下限、等于下限和小于下限时的积分结果.

```
Corollary VarUp_Value_gt : ∀ f c x, x ∈ dom[VarUp_Integral_Fun f c] -> c < x
  -> Def_Integral f c x ((VarUp_Integral_Fun f c)[x]).
Corollary VarUp_Value_eq : ∀ f c, (VarUp_Integral_Fun f c)[c] = 0.
```

```
Corollary VarUp_Value_lt : ∀ f c x, x ∈ dom[VarUp_Integral_Fun f c] -> x < c
  -> Def_Integral f x c (-(VarUp_Integral_Fun f c)[x]).
```

在描述原函数存在定理时应当注意, $\phi'(x)=f(x)$ 实际应该分为 $x\in[a,b]$、$x=a$ 和 $x=b$ 这 3 种情况, 因此严格的形式化描述如下.

```
Theorem Theorem9_10 : ∀ f a b, a < b -> ContinuousClose f a b
  -> (∀ x, x ∈ \(a, b\) -> Derivative (VarUp_Integral_Fun f a) x f[x])
    /\ Derivative_pos (VarUp_Integral_Fun f a) a f[a] /\ Derivative_neg
(VarUp_Integral_Fun f a) b f[b].
```

该定理证明过程较长, 此处不便展开, 细节可参见代码.

原函数存在定理的一个应用是可以给出牛顿-莱布尼茨公式的另一种证明思路, 这与第 7.3 节中通过定积分定义本身给出的证明不同.

具体来说, 由原函数存在定理可先导出两个推论: 变上限积分函数 $\phi(x)$ 是被积函数 $f(x)$ 在积分区间 $[a,b]$ 上的原函数 (这与原函数存在定理所述内容本质上是一致的); 变上限积分函数 $\phi(x)$ 与被积函数 $f(x)$ 的任意原函数仅相差一个常数, 此由上一条推论结合原函数的相关性质而证明. 两条推论的形式化描述如下.

```
Corollary Corollary9_10a : ∀ f a b, a < b -> ContinuousClose f a b
  -> Primitive_Close f (VarUp_Integral_Fun f a) a b.
Corollary Corollary9_10b : ∀ f F a b, a < b -> ContinuousClose f a b -> Primitive_Close
f F a b
  -> ∃ C, (∀ x, x ∈ \[a, b\] -> F[x] = (VarUp_Integral_Fun f a)[x] + C).
```

最后, 给出基于以上原函数存在定理相关推论的牛顿-莱布尼茨公式的完整形式化证明.

```
1 Corollary Corollary9_10c : ∀ f F a b, a < b -> ContinuousClose f a b -> Primitive_Close f F a b
2 -> Def_Integral f a b (F[b] - F[a]).
3 Proof.
4 intros. pose proof H1. apply Corollary9_10b in H2 as [C]; auto.
5 assert (F[b] = (VarUp_Integral_Fun f a)[b] + C /\ F[a] = (VarUp_Integral_Fun f a)[a] + C) as [].
6 { split; apply H2, Classifier1; lra. }
7 rewrite H3, H4, VarUp_Value_eq.
8 replace ((VarUp_Integral_Fun f a)[b] + C - (0 + C)) with (VarUp_Integral_Fun f a)[b]; [ |lra].
9 apply VarUp_Value_gt; auto.
10 assert (\[a, b\] ⊂ dom[VarUp_Integral_Fun f a]).
11 { apply VarUp_dom_b. apply Theorem9_4; auto. lra. }
12 apply H5, Classifier1; lra.
13 Qed.
```

第 4 行: 引入前提至证明条件中, 同时利用第 2 条推论, 得到条件: 任意 $x\in[a,b]$ 都有 $F(x)=\int_a^x f(t)\mathrm{d}t+C$, 其中 C 为一个实例化的常数.

第 5、6 行: 根据第 4 行代码引入的条件验证 $F(b)=\phi(b)=\int_a^b f(t)\mathrm{d}t+C$ 和 $F(a)=\phi(a)=\int_a^a f(t)\mathrm{d}t+C$, 并且将其引入条件.

第 7、8 行: 将目标中的 $F(b)-F(a)$ 替换为 $\int_a^b f(t)\mathrm{d}t$, 即 $\phi(b)$, 此时需证明: $f(x)$ 在 $[a,b]$ 上的定积分为 $\phi(b)$.

第 9 行: 由于证明条件中含有“$a<b$”, 因此 $\phi(b)$ 的取值为函数 $\phi(x)$ 的积分上限大于下限的情况, 可使用刚才在定义变上限积分函数时引入的推论 VarUp_Value_gt 解决当前的目标“$f(x)$ 在 $[a,b]$ 上的定积分为 $\phi(b)$”, 同时生

成一个新的目标: b 应当属于变上限积分函数 $\phi(x)$ 的定义域, 也即 VarUp_Value_gt 的前提. 仅有当一个命题的前提全部证明成立, 才能运用该命题.

第 10–12 行: 通过证明“区间 $[a,b]$ 包含于 $\phi(x)$ 的定义域”即可验证 b 属于 $\phi(x)$ 的定义域.

8 总结与展望

8.1 本文工作总结

本文工作完成之际, 数学家 Tao 也宣布完成了他所发起的“A Lean companion to‘Analysis I’”形式化项目[1], 实现了本科教材 Analysis I 中微积分内容的形式化, 形式化伴侣 (companion) 的用词也正体现了数学形式化对于辅助和指导教材学习的意义. 作为形式化数学的倡导者以及项目的引领者, Tao 更注重定义与定理的描述, 而将一些具体的验证过程留给可能的参与者共同探讨和完成.

本文介绍的数学分析形式化系统针对国内经典本科教材[35], 严格对应前 9 章一元微积分内容, 对其中所有定义均给出严格形式化描述, 对定理及其证明过程也均给出对应的严格形式化实现与证明. 整个形式化系统主要建立在集合论基础上, 这在基于类型论进行数学形式化的主流做法中相对少见, 其优势在于只需引入少而简单的类型结构, 可降低学习成本, 并且集合论广为熟知, 更便于数学使用者的理解和掌握. 读者在计算机运行我们代码的过程中完全可以中断命令、逐条验证定理证明过程的每一细节, 也可以根据自身学习和掌握情况, 对相关例题与习题进行形式化描述与证明, 从而检验学习效果.

形式化数学发展至今, 随着四色定理、开普勒猜想等这类知名数学问题的解决, 越来越多的学者开始认识到计算机辅助证明的重要价值, 各种基于定理证明器的形式化数学项目也在被不断开发和完善. 2021 年 6 月, 德国著名数学家 Scholze 借助计算机成功检验了其“凝聚态数学 (condensed mathematics)”理论证明的核心部分[21]; 2023 年 12 月, 由 Tao[55]发起的对多项式 Freiman-Ruzsa 猜想的形式化证明项目也取得了成功; 2024 年, 英国帝国理工学院的数学教授 Buzzard 则发起了一个用 Lean 形式化证明费马大定理的项目[56,57].

形式化数学在辅助学习与教学方面也应当发挥出其独特优势. 曾有学者与我们讨论, 希望制定一套可以检验学生自主学习效果的方案, 我们认为, 用 Coq 形式化完成数学定理的证明可以验证学生对数学理论的理解和掌握! 诚然, 对于一般数学学习者而言, 掌握一门类似 Coq 这样的计算机语言需要花费额外时间精力, 同时数学的形式化描述本身也会因开发者的不同而版本各异, 难以统一. 我们给出的一种方案是, 对应权威教材进行形式化开发, 这在一定程度上可使形式化描述风格保持一致, 并且有对应的数学参考, 学习者可以跟随代码学习数学, 也可以跟随数学理解代码, 从而既学习理解相关数学理论, 又实践提高计算机 Coq 技术, 将人的智慧与机器的智能结合起来, 充分体会基于 Coq 定理机器证明的代码可读性与交互性, 以及证明严谨性与可靠性等特点, 在一定程度上实现人们跟随计算机学习、理解、构建、教育乃至发展数学的尝试.

8.2 未来工作展望

数学分析的形式化是一项复杂工程, 一元微积分学仅是一个起点. 事实上, 我们也正在计划展开教材[35]中其余内容的形式化, 包括级数理论、多元函数微分、线积分、面积分等内容, 以构建一个较为完整的数学分析形式化系统, 并且相应工作不仅局限于证明器 Coq, 还可以基于其他证明器, 例如当下备受关注的 Lean; 其理论基础也不仅限于朴素集合论, 而可以基于严格的公理化集合论; 甚至还可以开发非标准分析[58]的形式化系统, 将实无限的概念引入计算机中, 丰富 Coq 社区的数学库.

数学形式化与人工智能大模型的结合也是当前学术界讨论的热点. 尽管本文的工作仍属于传统的形式化开发, 并未借助大模型工具, 但如今的大模型在数学形式化领域已经展现出强大潜力[10], 可为将来的形式化开发带来许多便利, 这主要表现在:

(1) 基于大模型的定理自动证明. 数学形式化的严谨性要求开发者必须验证定理中的每一处细节, 这有时反成形式化的一项缺点, 迫使开发者不得不花费大量时间处理各种显而易见的细节. 这些索然无味的重复性工作则可以交给大模型完成, 例如, 有团队针对 Coq 开发插件 CoqPilot[59], 可以让大模型自动生成证明策略, 填补这些显然

而繁琐的证明细节, 降低开发工作的时间成本.

(2) 形式化语言之间的相互转译. 各种定理证明器的形式语言在语法上互有差异, 而大模型则可以作为连接不同定理证明器的桥梁, 将一种证明器中的形式化代码转译为另一种形式化代码. 我们曾尝试在大模型 Claude 中将 Coq 代码转为 Lean 代码, 只需人工稍作调试即可在 Lean 中完全运行验证, 这大大简化了形式化项目在不同定理证明器中整体迁移的工作量, 可提高不同形式化语言开发者之间的沟通效率.

(3) 基于人工语言生成形式化代码. 让大模型从人工自然语言的数学文本直接生成形式化代码, 关键要确保人工数学文本的准确性、形式化代码的规范性以及人工描述与形式化描述之间对应的严格性, 而基于权威数学资料进行形式化开发, 则可以有效保障这些关键信息, 以作为大模型训练的基础数据.

值得指出, 尽管大模型在上述方面已经展现出不俗能力, 然而大模型普遍出现的“幻觉”问题也对辅助数学研究产生很大障碍[10], 在利用大模型带来便利的同时, 我们也应该清醒地认识到其在数学严谨上的局限性, 充分发挥人的主观能动性. 正如 Hamilton 在其著作[60]中所说: “哪怕最终计算机遍布世界, 它也绝不能代替数学家.” 尽管这句评论出自 20 世纪 70 年代, 它在今天仍具有参考意义.

References

[1] Tao T. A Lean companion to “Analysis I”. 2025. https://terrytao.wordpress.com/2025/05/31/a-lean-companion-to-analysis-i/

[2] De Moura L, Ullrich S. The Lean 4 theorem prover and programming language. In: Platzer A, Sutcliffe G, eds. Proc. of the 28th Int’l Conf. on Automated Deduction. Springer, 2021. 625−635. [doi: 10.1007/978-3-030-79876-5_37]

[3] Tao T. Analysis I. 4th ed., Singapore: Springer, 2022. [doi: 10.1007/978-981-19-7261-4]

[4] Avigad J. The mechanization of mathematics. Notices of the American Mathematical Society, 2018, 65(6): 681–690. [doi: 10.1090/noti1688]

[5] Avigad J, Harrison J. Formally verified mathematics. Communications of the ACM, 2014, 57(4): 66–75. [doi: 10.1145/2591012]

[6] Beeson MJ. The mechanization of mathematics. In: Teuscher C, ed. Alan Turing: Life and Legacy of a Great Thinker. Berlin: Springer, 2004. 77−134. [doi: 10.1007/978-3-662-05642-4_5]

[7] Hales TC. Formal proof. Notices of the American Mathematical Society, 2008, 55(11): 1370–1380.

[8] Harrison J, Urban J, Wiedijk F. History of interactive theorem proving. In: Handbook of the History of Logic. Amsterdam: Elsevier, 2014, 9: 135–214. [doi: 10.1016/B978-0-444-51624-4.50004-6]

[9] Jiang N, Li QA, Wang LM, Zhang XT, He YX. Overview on mechanized theorem proving. Ruan Jian Xue Bao/Journal of Software, 2020, 31(1): 82–112 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/5870.htm [doi: 10.13328/j.cnki.jos.005870]

[10] Tao T. Machine-assisted proof. Notices of the American Mathematical Society, 2025, 72(1): 6–13. [doi: 10.1090/noti3041]

[11] Wang H. Toward mechanical mathematics. IBM Journal of Research and Development, 1960, 4(1): 2−22. [doi: 10.1147/rd.41.0002]

[12] Wiedijk F. Formal proof—Getting started. Notices of the American Mathematical Society, 2008, 55(11): 1408−1414.

[13] Bertot Y, Castéran P. Interactive Theorem Proving and Program Development—Coq’Art: The Calculus of Inductive Constructions. Berlin: Springer, 2004. [doi: 10.1007/978-3-662-07964-5]

[14] Rocq Reference Manual (v 9.0.0). 2025. https://rocq-prover.org/doc/V9.0.0/refman/index.html

[15] Nipow T, Paulson LC, Wenzel M. Isabelle/HOL: A Proof Assistant for Higher-order Logic. Berlin: Springer, 2025.

[16] Paulson LC. Isabelle: A Generic Theorem Prover. Berlin: Springer, 1994. [doi: 10.1007/BFb0030541]

[17] Bancerek G, Byliński C, Grabowski A, Korniłowicz A, Matuszewski R, Naumowicz A, Pąk K, Urban J. Mizar: State-of-the-art and beyond. In: Proc. of the 2015 Int’l Conf. on Intelligent Computer Mathematics. Washington: Springer, 2015. 261−279. [doi: 10.1007/978-3-319-20615-8_17]

[18] Gowers WT. Rough structure and classification. In: Alon N, Bourgain J, Connes A, Gromov M, Milman V, eds. Visions in Mathematics, GAFA 2000 Special Volume, Part I. Modern Birkhäuser Classics. Basel: Birkhäuser, 2010. 79−117. [doi: 10.1007/978-3-0346-0422-2_4]

[19] Voevodsky V. Univalent foundations of mathematics. In: Beklemishev LD, de Queiroz R, eds. Proc. of the 18th Int’l Workshop on Logic, Language, Information and Computation (WoLLIC 2011). Philadelphia: Springer, 2011. 4. [doi: 10.1007/978-3-642-20920-8_4]

[20] Villani C. Théorème Vivant. Beijing: Posts & Telecom Press, 2016 (in Chinese).

[21] Castelvecchi D. Mathematicians welcome computer-assisted proof in ‘Grand Unification’ theory. Nature, 2021, 595(7865): 18–19. [doi: 10.1038/d41586-021-01627-2]

[22] Holden H, Piene R. The Abel Prize 2003–2007: The First Five Years. Berlin: Springer-Verlag, 2010. [doi: 10.1007/978-3-642-01373-7]

[23] Avigad J, Donnelly K, Gray D, Raff P. A formally verified proof of the prime number theorem. ACM Trans. on Computational Logic, 2007, 9(1): 2–es. [doi: 10.1145/1297658.1297660]

[24] Gonthier G. Formal proof—The four-color theorem. Notices of the American Mathematical Society, 2008, 55(11): 1382–1393.

[25] Hales TC. The Jordan curve theorem, formally and informally. The American Mathematical Monthly, 2007, 114(10): 882–894. [doi: 10.1080/00029890.2007.11920481]

[26] Paulson LC. A machine-assisted proof of Gödel's incompleteness theorems for the theory of hereditarily finite sets. The Review of Symbolic Logic, 2014, 7(3): 484–498. [doi: 10.1017/S1755020314000112]

[27] Ciolli G, Gentili G, Maggesi M. A certified proof of the Cartan Fixed Point Theorems. Journal of Automated Reasoning, 2011, 47(3): 319–336. [doi: 10.1007/s10817-010-9198-6]

[28] Avigad J, Hölzl J, Serafin L. A formally verified proof of the Central Limit Theorem. arXiv:1405.7012, 2017.

[29] Gonthier G, Asperti A, Avigad J, Bertot Y, Cohen C, Garillot F, Le Roux S, Mahboubi A, O'Connor R, Biha SO, Pasca I, Rideau L, Solovyev A, Tassi E, Théry L. A machine-checked proof of the Odd Order Theorem. In: Blazy S, Paulin-Mohring C, Pichardie D, eds. Proc. of the 4th Int'l Conf. on Interactive Theorem Proving. Rennes: Springer, 2013. 163−179. [doi: 10.1007/978-3-642-39634-2_14]

[30] Hales T, Adams M, Bauer G, Dang DT, Harrison J, Le Hoang T, Kaliszyk C, Magron V, McLaughlin S, Tat Nguyen T, Nguyen TQ, Nipkow T, Obua S, Pleso J, Rute J, Solovyev A, Thi Ta AH, Tran TN, Trieu DT, Urban J, Vu KK, Zumkeller R. A formal proof of the Kepler conjecture. arXiv:1501.02155, 2015.

[31] Hendriks M, Kaliszyk C, van Raamsdonk F, Wiedijk F. Teaching logic using a state-of-the-art proof assistant. Acta Didactica Napocensia, 2010, 3(2): 35–48.

[32] Avigad J. Learning logic and proof with an interactive theorem prover. In: Hanna G, Reid DA, de Villiers M, eds. Proof Technology in Mathematics Research and Teaching. Cham: Springer, 2019. 277−290. [doi: 10.1007/978-3-030-28483-1_13]

[33] Ornes S. How close are computers to automating mathematical reasoning? Quantamagazine. 2020. https://www.quantamagazine.org/how-close-are-computers-to-automating-mathematical-reasoning-20200827/

[34] Wan XY, Xu K, Cao QX. Coq Formalization of ZFC set theory for teaching scenarios. Ruan Jian Xue Bao/Journal of Software, 2023, 34(8): 3549–3573 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/6868.htm [doi: 10.13328/j.cnki.jos.006868]

[35] School of Mathematical Sciences, East China Normal University. Mathematical Analysis. 5th ed., Beijing: Higher Education Press, 2019 (in Chinese).

[36] Boldo S, Lelay C, Melquiond G. Formalization of real analysis: A survey of proof assistants and libraries. Mathematical Structures in Computer Science, 2016, 26(7): 1196–1233. [doi: 10.1017/S0960129514000437]

[37] The Rocq Standard Library. 2025. https://rocq-prover.org/doc/V9.0.0/stdlib/index.html

[38] Isabelle/HOL sessions. 2025. https://isabelle.in.tum.de/dist/library/HOL/index.html

[39] Leanprover-community: Mathlib4. 2025. https://github.com/leanprover-community/mathlib4

[40] Boldo S, Lelay C, Melquiond G. Coquelicot: A user-friendly library of real analysis for Coq. Mathematics in Computer Science, 2015, 9(1): 41–62. [doi: 10.1007/s11786-014-0181-1]

[41] Cruz-Filipe L, Geuvers H, Wiedijk F. C-CoRN, the constructive Coq repository at Nijmegen. In: Asperti A, Bancerek G, Trybulec A, eds. Proc. of the 3rd Int'l Conf. on Mathematical Knowledge Management (MKM 2004). Bialowieza: Springer, 2004. 88−103. [doi: 10.1007/978-3-540-27818-4_7]

[42] Ren ZZ, Shao ZH, Song JX, Xin HJ, Wang HC, Zhao WJ, Zhang LY, Fu Z, Zhu QH, Yang DJ, Wu ZF, Gou ZB, Ma SR, Tang HX, Liu YX, Gao WJ, Guo DY, Ruan C. DeepSeek-Prover-V2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. arXiv:2504.21801, 2025.

[43] Pierce BC, de Amorim AA, Casinghino C, *et al*. Software foundations. 2025. http://softwarefoundations.cis.upenn.edu/

[44] Guo DK, Leng SK, Dou GW, Chen S, Yu WS. Formalization in Coq of the consistency and categoricity of the real number axiomatic system based on Morse-Kelley set theory. Control Theory & Applications, 2024, 41(7): 1274–1285 (in Chinese with English abstract). [doi: 10.7641/CTA.2024.30698]

[45] Yu WS, Dou GW. A Machine Proof System for End-extensions of N in βN. Beijing: Science Press, 2024 (in Chinese).

[46] Yu WS, Fu YS, Guo LQ. Machine Proof System of Foundations of Analysis. Beijing: Science Press, 2022 (in Chinese).

[47] Hilbert D. Die grundlagen der mathematik. Abhandlungen Aus Dem Mathematischen Seminar Der Universitä t Hamburg, 1928, 6(1): 65–85. [doi: 10.1007/BF02940602]

[48] Bernays P, Fraenkel AA. Axiomatic Set Theory. Amsterdam: North Holland Publishing Company, 1958.

[49] Enderton HB. Elements of Set Theory. New York: Academic Press, 1977.

[50] Wang FT. Mathematical Foundations (Revised Ed). 2nd ed., Beijing: Higher Education Press, 2018 (in Chinese).

[51] Sun TY, Yu WS. A formal system of axiomatic set theory in Coq. IEEE Access, 2020, 8: 21510–21523. [doi: 10.1109/ACCESS.2020.2969486]
[52] Yu WS, Sun TY, Fu YS. A Machine Proof System for Axiomatic Set Theory. Beijing: Science Press, 2020 (in Chinese).
[53] Zorich VA. Mathematical Analysis I. New York: Springer, 2004.
[54] Zhao BQ, Yu C, Yu WS. A formal proof of two important limits in Coq. Highlights of Sciencepaper Online, 2021, 14(2): 177–186 (in Chinese with English abstract).
[55] Tao T. Formalizing the proof of PFR in Lean4 using Blueprint: A short tour. 2023. https://terrytao.wordpress.com/2023/11/18/formalizing-the-proof-of-pfr-in-lean4-using-blueprint-a-short-tour/
[56] Buzzard K. Fermat's last theorem—How it's going. 2024. https://xenaproject.wordpress.com/2024/12/11/fermats-last-theorem-how-its-going/
[57] Buzzard K, Taylor R. A blueprint for fermat's last theorem. https://imperialcollegelondon.github.io/FLT/blueprint/index.html
[58] Robinson A. Non-standard Analysis, Revised ed., Amsterdam: North Holland Publishing Company, 1974.
[59] Kozyrev A, Solovev G, Khramov N, Podkopaev A. CoqPilot, a plugin for LLM-based generation of proofs. In: Proc. of the 39th IEEE/ACM Int'l Conf. on Automated Software Engineering. Sacramento: ACM, 2024. 2382−2385. [doi: 10.1145/3691620.3695357]
[60] Hamilton AG. Logic for Mathematicians. Cambridge: Cambridge University Press, 1978.

附中文参考文献

[9] 江南, 李清安, 汪吕蒙, 张晓瞳, 何炎祥. 机械化定理证明研究综述. 软件学报, 2020, 31(1): 82–112. http://www.jos.org.cn/1000-9825/5870.htm [doi: 10.13328/j.cnki.jos.005870]
[20] 塞德里克·维拉尼. 一个定理的诞生. 马跃, 杨苑艺, 译. 北京: 人民邮电出版社, 2016.
[34] 万新熠, 徐轲, 曹钦翔. 针对教学场景的 ZFC 集合论 Coq 形式化. 软件学报, 2023, 34(8): 3549–3573. http://www.jos.org.cn/1000-9825/6868.htm [doi: 10.13328/j.cnki.jos.006868]
[35] 华东师范大学数学科学学院. 数学分析. 第 5 版, 北京: 高等教育出版社, 2019.
[44] 郭达凯, 冷姝锟, 窦国威, 陈思, 郁文生. 基于 MK 的实数公理系统相容性和范畴性的 Coq 形式化. 控制理论与应用, 2024, 41(7): 1274–1285. [doi: 10.7641/CTA.2024.30698]
[45] 郁文生, 窦国威. 自然数的紧化延伸机器证明系统. 北京: 科学出版社, 2024.
[46] 郁文生, 付尧顺, 郭礼权. 分析基础机器证明系统. 北京: 科学出版社, 2022.
[50] 汪芳庭. 数学基础 (修订本). 第 2 版, 北京: 高等教育出版社, 2018.
[52] 郁文生, 孙天宇, 付尧顺. 公理化集合论机器证明系统. 北京: 科学出版社, 2020.
[54] 赵保强, 于畅, 郁文生. 基于 Coq 的两个重要极限的形式化证明. 中国科技论文在线精品论文, 2021, 14(2): 177–186.

作者简介

窦国威, 博士生, CCF 学生会员, 主要研究领域为人工智能与自动推理.
郁文生, 博士, 教授, 博士生导师, 主要研究领域为控制理论与控制工程, 泛函微分方程理论, 符号计算及实代数理论及应用, 人工智能与自动推理.