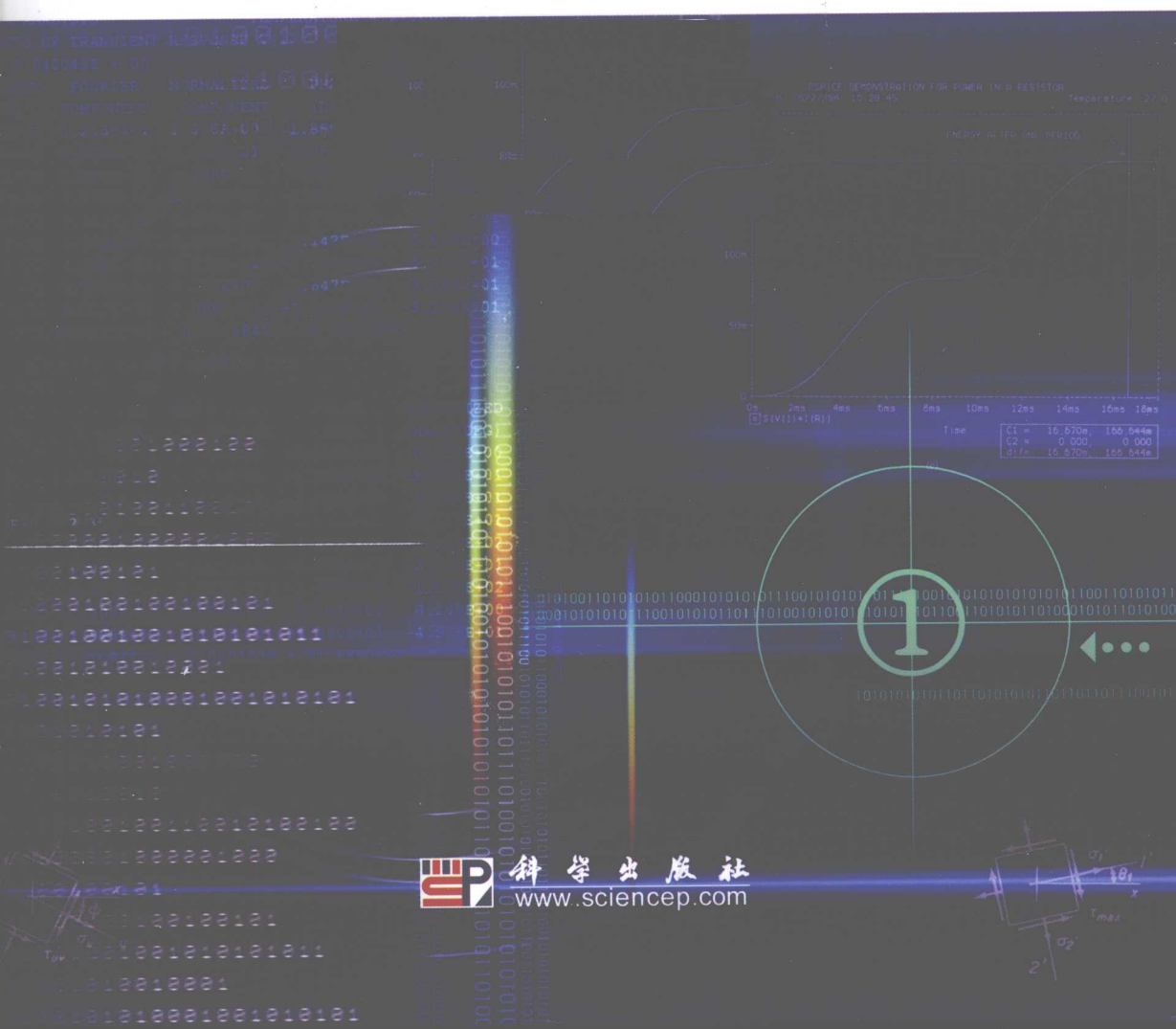
实用程序设计理论

A PRACTICAL THEORY OF PROGRAMMING

〔加拿大〕Eric C.R.Hehner 著
万剑怡 郑宇华 译



科学出版社
www.sciencep.com

实用程序设计理论

〔加拿大〕Eric C. R. Hehner 著

万剑怡 郑宇华 译

科学出版社

北京

图字：01-2010-1592

内 容 简 介

全书正文部分共分为 12 章，系统地介绍了一种程序设计理论，包括基本理论、基本数据结构、函数理论、程序理论、程序设计语言、递归定义、理论设计与实现、并发和交互等内容. 本书的内容既系统、丰富、连贯，又非常精练，浅显易懂. 另外，本书还附有 400 多道有趣的练习题(全部集中在第 10 章).

本书可作为高等院校计算机科学专业的高年级本科生和研究生程序理论课程的教材，也可作为对软件形式化方法有兴趣的研究人员和技术人员的参考书.

本书是 *A Practical Theory of Programming* 更新的网络版的中文翻译版，由原书作者授权出版.

This is a Chinese translation of the updated on-line edition of *A Practical Theory of Programming*, the publication of which has been authorized by the authors.

图书在版编目(CIP)数据

实用程序设计理论/〔加拿大〕Eric C. R. Hehner 著，万剑怡，郑宇华译. —北京：科学出版社，2010

ISBN 978-7-03-027425-0

Ⅰ. 实… Ⅱ. ①E… ②万… ③郑… Ⅲ. 程序设计 Ⅳ. TP311.1

中国版本图书馆 CIP 数据核字 (2010) 第 080550 号

责任编辑：赵彦超 于宏丽 / 责任校对：陈玉凤
责任印制：钱玉芬 / 封面设计：王 浩

科 学 出 版 社出版
北京东黄城根北街 16 号
邮政编码：100717
http://www.sciencep.com

骏 走 印 刷 厂 印刷
科学出版社发行 各地新华书店经销
*
2010 年 6 月第 一 版 开本：B5(720×1000)
2010 年 6 月第一次印刷 印张：18 3/4
印数：1—3 000 字数：362 000

定价：56.00 元

(如有印装质量问题，我社负责调换)

中 译 本 序

《实用程序设计理论》的原著者为加拿大多伦多大学计算机科学系 Eric C. R. Hehner 教授. Hehner 教授多年来一直从事软件形式化方法、程序设计方法学和软件工程的教学和科研, 是一位在程序逻辑和软件方法领域思想十分活跃的著名学者. 针对现有的形式化方法逻辑结构复杂、表示方法繁琐、难学难用的缺陷, Hehner 教授提出了一种实用的程序设计方法. 和同类著作相比, 该书具有以下特色:

(1) 原理简单, 篇幅短小, 却涵盖了顺序和并行计算、终结和非终结计算、独立和交互计算、概率计算、程序运行时间和空间界定计算以及实时计算, 知识点全面、系统, 通用性好;

(2) 容易理解, 读者只需具有初等布尔代数和程序设计语言概要方面的预备知识即可;

(3) 程序开发和程序修改与程序验证同步进行.

自从图灵奖获得者 C. A. R. Hoare 提出研发可验证软件正确的编译程序这一重大挑战性问题以来, 世界各国均高度关注程序正确性和软件可信度方面的研究. 我国也把可信软件的研究列入国家自然科学基金的重大研究方向、"863" 计划和 "973" 计划. 我国计算机领域的专业人员对软件的验证与构造的理论和技术正在进行系统深入的研究, 并探索实用的软件设计方法. 万剑怡和郑宇华两位博士正是为了满足我国学者这方面的需要而翻译了该书. 她们曾先后在多伦多大学进修访问, 在 Hehner 指导下从事科学研究, 对他的科学思想有深刻的认识和理解. 因此, 由她们合作将该书译成中文, 介绍给中国读者, 是十分合适的.

该书可供从事软件形式化方法研究的专业人员阅读参考. 希望 Hehner 追求简洁实用的思想能与我国学者的智慧发生碰撞, 产生更加绚丽的火花, 结出硕果. 该书也可作为大学高年级本科生和研究生程序设计方法学课程的入门教材, 使学生能尽快掌握软件构造和验证的基本方法. 软件开发者也可以通过阅读该书提高开发可信软件的能力.

薛锦云

2009 年 9 月

目　录

第0章 绪 言

0.0 引 言

程序设计理论有什么用? 谁需要它呢? 没有任何理论, 成千上万的程序员还是每天都在编程, 那么又何必费心去学一门理论呢? 该问题的答案其实和任何其他理论一样. 譬如, 为什么人人都要学习运动学理论? 人们可以在不懂理论的情况下行动自如, 也可以在不懂理论的情况下投球. 但是, 在中学讲授运动学理论却是相当重要的.

对上述问题的回答之一是: 通过提供一种演算的方法, 可以使数学理论达到更加精确的程度. 没有关于运动学理论, 根本不可能发射火箭到木星, 甚至投球手们也发现理论专家可以帮助他们改进投球技术. 同样, 没有理论的帮助, 许多大众化的程序设计也可以完成, 但对更复杂的程序设计, 没有好的理论就难以准确无误完成. 这一点可以由软件产业中无数因程序错误而造成的惨痛教训来证明. 况且, 即使是通常的编程也会因适当运用理论而得到改进.

回答之二是: 理论可以提高理解能力. 当人们学会了运动学理论时, 对运动的控制和预见能力就从艺术走向了科学. 同样, 当人们学会以理解数学定理的方式来理解程序时, 程序设计也就从艺术走向了科学. 有了科学的观点, 就可以改变对世界的看法, 更少地依赖灵感或运气, 更明白什么是可能的, 什么是不可能的. 对任何人而言, 这都是教育带来的宝贵财富.

专业工程学在社会上享有很高的声誉, 就是因为它坚持这样一点: 要成为一名专业工程师, 就必须知道应用相关的理论. 一个土木工程师必须知道运用几何和材料力学, 一个电气工程师必须懂得并善于运用电磁理论, 软件工程师要想名副其实, 也必须懂得并善于运用程序设计理论.

本书的主题有时称为 "程序设计方法学", 有时又称为 "程序设计科学"、"程序设计逻辑"、"程序设计理论"、"程序开发形式化方法" 或 "程序证明", 它涉及程序设计中服从数学证明的那些方面. 好的理论能帮助人们写出精确的规范, 并设计出按该规范执行的程序. 本书将考虑计算的状态、时间、空间以及交互. 但本书没有论及一些有关软件设计和制作的重要方面, 如人员管理、用户界面、文档化和测试等.

第一个有用的程序设计理论通常称为 "Hoare 逻辑", 该理论可能仍是最广为人知的. 在 Hoare 逻辑中, 一条规范是一对谓词: 前置条件和后置条件 (这两个术

语及其后提及的所有术语都将在以后的相应章节中给出定义). 另一个紧密相关的理论是 Dijkstra 的最弱前置谓词转换器, 它将程序和后置条件通过转换得到前置条件, 后来发展成了 Back 的精化演算. Jones 的维也纳开发方法已被某些产业采用并从中获益, 其中, 一条规范是一对谓词 (正如 Hoare 逻辑), 但其中第二个谓词是一个关系. 另外, 还有一些专门用于实时程序设计、概率程序设计、交互式程序设计的理论.

本书的理论比上面提到的都要简单. 在该理论中, 一条规范就是一个布尔表达式, 精化就是通常的蕴含式. 该理论也比上述理论更为通用, 同时适用于终止和非终止计算、顺序和并行计算, 以及独立和交互式计算. 同时本书可以有只需关注初值和终值的变量、需始终对其值进行观测的变量、只知道其可能值的变量和用于计算时间和空间的变量. 它们都适合于同一理论, 其基础在于将规范表示成布尔表达式, 而布尔表达式的变量可以是任何感兴趣的变量.

有一种通过穷举测试所有输入来证明程序的方法, 称为模型检测 (model checking), 它优于本书理论的一点在于可以完全自动化. 目前一条设计精巧的布尔表达式 (见练习 6) 可以使模型检测的状态数量约达到 10^{60} 个, 这比宇宙中所估计的原子个数还多! 这个数字令人印象深刻, 10^{60} 约为 2^{200}, 这意味着是 200 位二进制数, 也就是 6 个 32 位二进制数变量的状态空间. 对任何一个超过 6 个变量的程序进行模型检测就需要抽象, 每个抽象要求证明其保持所感兴趣的性质, 而这些证明不是自动的. 为了保证实用性, 模型检测必须与其他的证明方法相结合, 如本书中的方法.

本书还要强调的一点是, 程序开发的每一步都是伴随证明的, 而不是在开发以后再进行证明.

0.1　当前版本

在本书的当前版本中增加了空间约束和概率程序设计的新资料, 归纳了 for 循环规则, 简化了对并发的处理, 且对并行进程之间的合作提供了选择: 通信 (如第一版) 和交互变量. 本书加强了解释和说明, 并增加了更多整理过的例子.

在增加的同时也有删减. 为了保证本书的精练, 书中删除了所有在课程中将略过的资料. 本书主要内容共有 184 页, 后面的只是练习和参考材料.

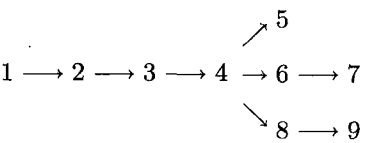
授课教师可从本书作者处获得有关该课程的课件和练习解答.

0.2　快速浏览

本书用到的所有术语都会在书中得到解释. 每一条新术语都用楷体表示. 在解释这些术语时, 本书尽可能采用释义的方法而少用纪念性的人名 (值得一提的例外

是 "Boole"), 并且不使用缩略词、首字母缩略语或其他一些难懂的语言, 以便读者理解. 本书假定读者不具备专门的数学知识或程序设计经验, 但在本书第 1 ~ 3 章中, 有关布尔 (Boole)、数字 (number)、表 (list) 和函数 (function) 的预备知识只作了简单介绍, 具有预备的知识或经验可能会有帮助.

下图表示了各章与先前章节的依赖关系:



第 4 章 "程序理论" 是本书的核心章节, 其后的章节可以根据上图和兴趣进行取舍. 上图仅有的例外是第 9 章使用了 5.0.0 小节介绍的变量说明, 可选读的 9.1.3 小节要用到第 6 章的知识. 每章中有可选读标志 (*) 的节或小节可以忽略, 且不会对后面章节的理解产生影响.

第 10 章全部由练习组成, 这些练习所用到的理论都已在前面各章中作过介绍. "程序理论" 一节的全部练习可利用第 4 章所介绍的方法完成, 不过, 其后几章中又介绍了一些新的记号和方法, 用这些方法重做那些练习会变得更加容易.

本书的最后一章, 即第 11 章包含了参考材料. 11.0 节 "释疑" (justification) 将回答以前章节的一些问题, 诸如: "为什么要用这种方法表示"、"为什么要表示这个"、"为什么不是表示别的" 等. 也许只有那些已经具有丰富程序设计理论知识的教师和研究工作者才对此章感兴趣, 而那些第一次接触到形式化方法的学生可能不感兴趣. 不过, 如果发现有上述问题时, 不妨参阅 "释疑" 这一节.

第 11 章还包括术语的索引和本书所用到的所有规则的清单. 对于那些热衷于程序设计的学生, 这些规则将会成为他们的朋友. 本书最后列出了书中使用的全部记号, 读者并不需要预先知道这些记号, 因为它们在首次使用时都会被解释. 读者可以发明一些新的记号, 当然, 必须说明它们的用途. 有时记号的选择差异会影响到解决一个问题的能力.

0.3 致　谢

感谢国际信息处理联合会 (IFIP) 的 2.3 工作小组 (程序设计方法学), 特别是 Edsger Dijkstra, David Gries, Tony Hoare, Jim Horning, Cliff Jones, Bill McKeeman, Carroll Morgan, Greg Nelson, John Reynolds 和 Wlad Turski, 他们给予我很多启发和指导; 特别感谢 Doug McIlroy 的鼓励. 感谢我的研究生和助教, 他们给了我不少有益的提示, 特别是 Ray Blaak, Benet Devereux, Lorene Gupta, Peter Kanareitsev, Yannis Kassios, Victor Kwan, Albert Lai, Chris Lengauer, Andrew Malton, Theo

Norvell, Rich Paige, Dimi Paun, Mark Pichora, Hugh Redelmeier 和 Alan Rosenthal. 感谢 Wim Hesselink, Jim Horning 和 Jan van de Snepscheut 对初稿进行了严格有益的审阅. 感谢 Ralph Back, Eike Best, Wim Feijen, Netty van Gasteren, Nicolas Halbwachs, Gilles Kahn, Leslie Lamport, Alain Martin, Joe Morris, Martin Rem, Pierre-Yves Schobbens, Mary Shaw, Bob Tennent 和 Jan Tijmen Udding, 他们向我提出了好的建议. 感谢 Jules Desharnais, Andy Gravell, Peter Lauer, Ali Mili, Bernhard Moeller, Helmut Partsch, Jorgen Steensgaard-Madsen 和 Norbert Voelker, 他们阅读了文稿并提出了改进意见. 感谢我班上的学生, 他们找出了本书中的一些错误.

第1章 基本理论

1.0 布尔理论

布尔理论 (Boolean theory), 又称为布尔逻辑, 原本是作为推理的辅助工具而设计的, 这里用它对计算进行推理. 布尔理论的表达式称为*布尔表达式*, 分为两类: 一类叫*定理* (theorem), 另一类叫*反定理* (antitheorem).

布尔表达式可用来表示客观世界命题, 定理表示真命题, 反定理表示假命题. 这是布尔理论的基本应用, 是该理论的设计原理, 且由此给出了大量的术语. 布尔理论的另一个完美的应用是在数字电路设计中, 此时布尔表达式代表了电路, 定理代表输出为高电压的电路, 而反定理代表输出为低电压的电路.

两个最简单的布尔表达式是 $\top$ 和 $\bot$. 前者是定理, 后者是反定理. 当布尔理论用于其基本应用时, 将 $\top$ 读为 "真", $\bot$ 读成 "假", 因为前者表示任意一个为真的命题, 后者表示任意一个为假的命题. 当布尔理论用于数字电路设计时, 将 $\top$ 读成 "高电压" (high voltage), $\bot$ 读成 "低电压" (low voltage), 或者分别读成 "火线" (power) 和 "地线" (ground). 它们有时也被称作 "布尔值", 或者被称作 "空布尔操作符" (nullary Boolean operators), 意指它们无需操作数.

单目布尔运算符共有四个, 这里只讨论其中之一, 其符号为 $\neg$, 读作 "非" (not). 它是一个前缀运算符 (置于运算对象之前), 对于形如 $\neg x$ 的表达式, 称作*否定式* (negation). 如果将定理求否就得到反定理, 而将反定理求否就得到定理. 可以用下面的*真值表* (truth table) 描述这一求否过程:

$\neg$	$\top$	$\bot$
	$\bot$	$\top$

在横线之上, $\top$ 表示操作数为定理, $\bot$ 表示操作数为反定理; 在横线之下, $\top$ 表示结果为定理, $\bot$ 表示结果为反定理.

双目布尔运算符共有 16 个, 出于习惯, 只使用其中的 6 个. 它们都是中缀运算符 (置于两个运算对象之间). 以下是它们的符号和一些读法:

$\wedge$ 与

$\vee$ 或

$\Rightarrow$ 蕴含, 等于或强于

$\Leftarrow$ 由 $\cdots$ 导出, 被 $\cdots$ 蕴含, 弱于或等于

$=$　等于, 当且仅当

$\neq$　不同于, 不等于, 异或, 布尔加

形如 $x \wedge y$ 的表达式称作**合取式** (conjunction), 运算对象 x 和 y 称作**合取因子** (conjunct). 形如 $x \vee y$ 的表达式称为**析取式** (disjunction), 其中的运算对象称作**析取因子** (disjunct). 形如 $x \Rightarrow y$ 的表达式称为**蕴含式** (implication), 其中 x 称作**前件** (antecedent), y 称作**后件** (consequent). 形如表达式 $x \Leftarrow y$ 也称为蕴含式, 只是 x 为后件, y 为前件. 形如表达式 $x = y$ 称为**等式** (equation), 其中的运算对象分别称作**左式** (left side) 和**右式** (right side). 形如表达式 $x \neq y$ 称为**不等式** (unequation), 其中的运算对象也分别称作左式和右式.

下面的真值表显示了不同种类的运算对象在双目运算符操作下的不同结果. 横线之上, $\top\top$ 表示两个运算对象都是定理; $\top\bot$ 表示左运算对象是定理, 右运算对象是反定理; 依此类推. 横线之下, $\top$ 表示结果是定理, $\bot$ 表示结果是反定理.

	$\top\top$	$\top\bot$	$\bot\top$	$\bot\bot$
$\wedge$	$\top$	$\bot$	$\bot$	$\bot$
$\vee$	$\top$	$\top$	$\top$	$\bot$
$\Rightarrow$	$\top$	$\bot$	$\top$	$\top$
$\Leftarrow$	$\top$	$\top$	$\bot$	$\top$
$=$	$\top$	$\bot$	$\bot$	$\top$
$\neq$	$\bot$	$\top$	$\top$	$\bot$

中缀运算符可能造成一些表达式具有歧义. 例如, $\bot \wedge \top \vee \top$, 可被理解为 $\bot \wedge \top$ 先做合取, 结果为反定理, 然后再与 $\top$ 做析取, 最后结果为定理; 还可以被理解为 $\bot$ 与析取式 $\top \vee \top$ 做合取, 最后结果为反定理. 利用圆括号可以区别表示上述的不同理解: $(\bot \wedge \top) \vee \top$ 或 $\bot \wedge (\top \vee \top)$. 为了避免过多使用圆括号, 可引入优先级表, 该表列在 11.7 节. 在这张优先级表中, $\wedge$ 优先级为 9, $\vee$ 的优先级为 10, 这意味着在缺省括号的情形下, $\wedge$ 的运算优先于 $\vee$. 因此, 上例 $\bot \wedge \top \vee \top$ 的运算结果应是定理.

运算符 $=\!\!\Rightarrow\Leftarrow$ 在优先级表中分别出现了两次, 大的 $=\!\!\Rightarrow\Leftarrow$ 优先级为 16, 是所有运算符中优先级最低的, 小的运算符除了优先级不同外, 其他操作性质都与相应的大运算符一样. 在一定的使用限制下, 这些重复的运算符有时可以进一步减少圆括号的使用量, 从而提高表达式的可读性. 提醒一句: 适当使用一些圆括号, 尽管根据优先级这些括号可能是不必要的, 但可以帮助理解表达式结构. 可根据具体情况决定是否使用它们.

三目运算符共有 256 个, 这里只介绍 1 个, 称为**条件组合** (conditional compo-

sition), 写成 **if** x **then** y **else** z. 下面是它的真值表:

	$\top\top\top$	$\top\top\bot$	$\top\bot\top$	$\top\bot\bot$	$\bot\top\top$	$\bot\top\bot$	$\bot\bot\top$	$\bot\bot\bot$
if then else	$\top$	$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\top$	$\bot$

依此类推, n 目运算符共有 2^{2^n} 个, 但现在已足够了.

在前面介绍的合取表达式 $x \wedge y$ 中, x 和 y 为**变量** (variables), 它们分别可以被任意的布尔表达式替换, 这样, $x \wedge y$ 就可以代表所有的合取表达式. 例如, 用 $(\bot \Rightarrow \neg(\bot \vee \top))$ 代替 x, 用 $(\bot \vee \top)$ 代替 y, 就得到下面的合取式:

$$(\bot \Rightarrow \neg(\bot \vee \top)) \wedge (\bot \vee \top)$$

用表达式替换变量叫做**置换** (substitution), 或者叫做**实例化** (instantiation). 由于变量可以被置换, 所以表达式中允许使用变量, 但应注意以下两点:

- 在用表达式置换变量时, 为了不改变运算符执行的优先次序, 有时需将表达式用括号括起来. 例如, 在前述的置换例子中, 用蕴含式 $\bot \Rightarrow \neg(\bot \vee \top)$ 代替 x, 但由于合取运算优先于蕴含, 所以必须在该蕴含式外加上括号. 类似地, 在用析取式 $\bot \vee \top$ 代替 y 时也应加上括号.
- 当同一个变量在一个表达式中出现两次或两次以上时, 每次都必须使用相同的表达式去置换该变量. 例如, 表达式 $x \wedge x$ 可以被置换成 $\top \wedge \top$, 但不能置换成 $\top \wedge \bot$. 对于不同的变量可以用相同的或不相同的表达式来置换, 例如, $x \wedge y$ 可以被置换成 $\top \wedge \top$ 或 $\top \wedge \bot$.

在介绍其他理论时, 书中将引入一些利用该理论中的表达式而构成的新布尔表达式, 并将它们进行归类. 例如, 当介绍数论 (number theory) 时, 需要引入自然数表达式 $1+1$ 和 2, 从而引入布尔表达式 $1+1=2$, 这个布尔表达式归类为定理. 不能将一个布尔表达式既归类成定理又归类成反定理, 因为客观世界中的命题在同一含义下不能既是真又是假; 一个电路的输出不能既是高电压又是低电压. 如果不小心将一个布尔表达式同时归类成这两个结果, 就犯了一个严重的错误. 但是允许一个布尔表达式无类别 (unclassified). 例如, $1/0=5$, 它既不是定理又不是反定理. 一个无类别的表达式可以对应于这样一个命题, 即不知道或者不关心它是真或是假, 或对应于这样一个电路, 它的输出无法预测. 如果一个理论中不存在既是定理又是反定理的布尔表达式, 则称该理论是**一致的** (consistent); 反之, 若存在既是定理又是反定理的布尔表达式, 则称为**不一致的** (inconsistent). 如果一个理论中每个完全实例化的布尔表达式或者是定理或者是反定理, 那么称该理论是**完备的** (complete); 反之, 若存在一些完全实例化的布尔表达式既不是定理又不是反定理, 则称该理论是**不完备的** (incomplete).