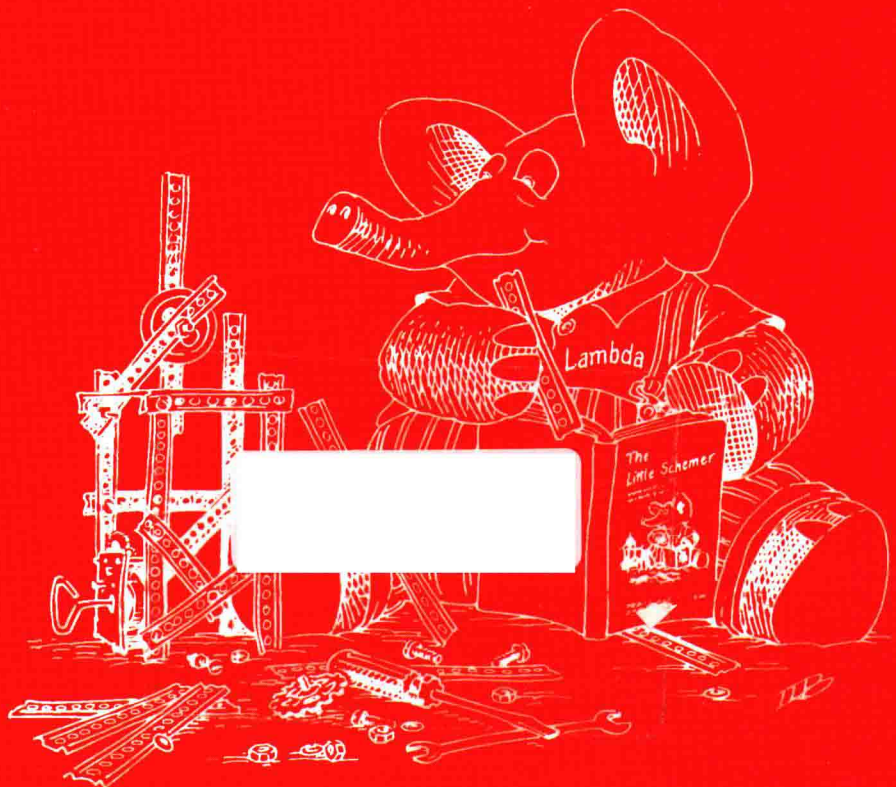
Broadview®
www.broadview.com.cn

The Little Schemer
Fourth Edition

The Little Schemer

递归与函数式的奥妙



[美] Daniel P. Friedman
Matthias Felleisen 著

Gerald J. Sussman 作序

卢俊祥 译

中国工信出版集团
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

The Little Schemer
Fourth Edition

The Little Schemer
递归与函数式的奥妙

[美] Daniel P. Friedman
Matthias Felleisen 著

Gerald J. Sussman 作序

卢俊祥 译

電子工業出版社
Publishing House of Electronics Industry
北京•BEIJING

内 容 简 介

本书是一本久负盛名的经典之作，两位作者 Daniel P. Friedman、Matthias Felleisen 在程序语言界名声显赫。全书介绍了 Scheme 的基本结构及其应用、Scheme 的五法十诫、Continuation-Passing-Style、Partial Function、Y-Combinator、Interpreter 等内容，并通过这些内容阐述了计算的一般本质。本书没有什么理论性描述，所有概念都蕴含在独特的引导式一问一答过程中，这种方式让读者对程序大师运用熟稔的程序方法来驾驭概念的能力叹为观止。

通过阅读本书，可以让读者领略递归的奥妙、函数式编程风格的魅力。阅读完毕会有一种意犹未尽的感觉。

本书适合所有程序员阅读，特别是函数式编程爱好者。好好享用!

版权贸易合同登记号　图字：01-2016-1163

图书在版编目（CIP）数据

The Little Schemer: 递归与函数式的奥妙/（美）丹尼尔 · P. 弗里德曼（Daniel P. Friedman），（美）马提亚 · 费雷森（Matthias Felleisen）著；卢俊祥译. —北京：电子工业出版社，2017.7
书名原文：The Little Schemer - 4th Edition
ISBN 978-7-121-31725-5

Ⅰ. ①T… Ⅱ. ①丹… ②马… ③卢… Ⅲ. ①程序语言－程序设计 Ⅳ. ①TP312

中国版本图书馆 CIP 数据核字（2017）第 121125 号

策划编辑：张春雨
责任编辑：刘　舫
印　　刷：北京中新伟业印刷有限公司
装　　订：北京中新伟业印刷有限公司
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×980　1/16　　印张：13.25　　字数：291 千字
版　　次：2017 年 7 月第 1 版
印　　次：2017 年 7 月第 1 次印刷
定　　价：65.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：010-51260888-819　faq@phei.com.cn。

译 者 序

进入互联网、移动互联网时代，软件开发方面的好书层出不穷，绝大部分是技术新、方法新。然而，本书很独特，其出版于 1995 年，至今已有二十余年，而其前身 *The Little LISPer* 则出版于 1987 年，堪称“古董”！

为什么一本老书还有出版的必要？因为“经典”！因为其内容揭示了计算的一般本质，其价值历经时光的检验而含金量不减！其实本书已不用过多着墨加以介绍，其在广大程序员心中早已竖起了一座丰碑。

我酷爱编程，也接触过许多函数式编程语言，但没有任何一种编程语言能够像 LISP 那样擅于通过直接和简单的方式表达编程思维，不熟悉者迷惑于它的括号，而登堂入室者则能领略其精髓，最终游刃有余。本书只借助了 Scheme 编程语言的若干基础元件，就演绎出了各种问题的解决方式——这就是最佳诠释！

也许你在工作中不会用到 Scheme，但是本书贵在作者深厚的编程积累，并能将丰富的经验充分发挥到本书内容中。全书的每一步都不显山露水，但最终蓦然回首时，轻舟已过万重山。探索计算本质的过程竟然如此巧妙，不禁让人拍案叫绝——手中用的是 Scheme 的招式，而心中洞察到的却是计算的内涵！

当其他编程书籍在讨论大量 Hack 技巧、各种设计模式的运用、形形色色的语法糖变化的时候，本书无疑就像一部另辟蹊径的武林秘籍，能大大增强习练者的内功。

作为一名有追求的程序员，这本书就是为你准备的。同时非常期待本书的姊妹篇 *The Seasoned Schemer*。

参与本书翻译工作的还有林长瑞、吴桐、朱建宝、周荣华、吴胜华、叶铭辉、李禧强、姚建峰、郑秀玲。

感谢我的妻子和孩子，他们给了我很大的支持，小宝贝还给我带来了许许多多的乐趣。同时还要感谢本书编辑张春雨，在他的鼓励下，我的翻译过程充满愉悦。

卢俊祥

2017 年 6 月

译者简介

卢俊祥

程序员；译者，爱读书；武当二十八式太极拳。

微博：@2gua。

个人网站：http://www.2gua.info/。

知乎专栏：https://zhuanlan.zhihu.com/guagua

序

本序最初出现在 The Little LISPer[①]一书的第二、三版中。经作者许可，特在此重现。

时光回到 1967 年，那时我报了一门摄影入门课程。包括我在内，大多数参加该课程的同学都憧憬着早日掌握创造性的摄影知识，希望自己有朝一日能成为又一个爱德华·韦斯顿[②]。第一天，老师详细地列出了一长串这学期要掌握的技能点。其中一个关键技能是安塞尔·亚当斯（Ansel Adams）的“区域曝光法”——用于预先视觉化冲印数据（最终冲印的灰度），及从景物光线强度中获取灰度。为了使用区域曝光法，还得学习曝光表用法以度量光线强度，以及通过曝光时间及显影时间来控制图像的灰度和对比度。反过来，这些技能又需要诸如胶片安装、显影、冲印和药水调制等更加底层的技能来支持。你必须学会将感光材料的显影过程程序化，以便在日后处理中获得一致的效果。第一次实验课是设法识别滑滑的显影剂和刺鼻的定影液。

而要让构图更具创造性，则必须首先具备驾驭工具的能力。甚至在能力具备之前都不要去构思如何组织一张好照片。在工程领域，如同其他创造性艺术，必须学会分析以支持我们在各方面的努力。那些有关钢材、扬尘以及大量数学方法等方面的知识，是计算构筑物属性时需要的，缺失了这些知识就无法构建美观而实用的桥梁。同样，未深入理解如何“预先视觉化”编程生成的工序，则无法构造出卓越的计算机系统。

一些摄影师选择 8×10 的黑白底片[③]，而其他一些则选择 35mm 的底片[④]。不同片幅类型的底片各有其优缺点。跟摄影一样，编程也需要选择称心的语言。魔法编程语言 Lisp 属于崇尚自由和灵活风格的程序员！Lisp 最初的设想是作为理论辅助工具，用于递归理论及

---

① 译者注：*The Little LISPer* 是 *The Little Schemer* 的前身。

② 译者注：Edward Weston，1886—1958，被称为“摄影界的毕加索”。

③ 译者注：大幅底片，单位为英寸，一般用于拍摄广告或大型海报。

④ 译者注：底片宽 35mm，135 型，最常用。

符号代数。时至今日，Lisp 已发展成为一个软件开发工具的大家族，魅力独特、功能强大且异常灵活，为软件系统的快速原型设计提供了全方位支持。与其他编程语言一样，技术社区开发出庞大的抽象功能库，Lisp 则将这些功能连接起来。在 Lisp 的世界里，程序是一等数据，以参数方式传递，以值的方式返回，并存储在数据结构中。这种灵活性极具价值，而最重要的是，其为形式化、命名以及精简惯用法——工程设计中必不可少的常用使用模式，提供了机制保障。此外，Lisp 程序能够轻松操纵 Lisp 程序的表述——一个开发庞大结构的程序综合[①]及分析工具（如交叉引用）的支持特性。

The Little LISPer 一书以独特方式阐述了 Lisp 创造性编程哲学里的精髓技法。全书借助大量实际训练——掌握构建递归过程及操纵递归数据结构等技能所必要的实践，相当巧妙地将知识组织起来，让人丝毫感受不到学习的压力。*The Little LISPer* 一书对 Lisp 学习者的意义，不亚于哈农[②]手指练习或车尔尼[③]钢琴研究对于钢琴学生的意义。

杰拉德·杰伊·萨斯曼[④]

剑桥（美国，马萨诸塞州）

① 译者注：关于程序综合（program synthesis）的更多信息详见 https://en.wikipedia.org/wiki/Program_synthesis。

② 译者注：查尔斯·路易斯·哈农（Charles-Louis Hanon）是 19 世纪法国著名的钢琴演奏家、钢琴教育家、管风琴师，哈农钢琴指法创始人。

③ 译者注：卡尔·车尔尼（Carl Czerny）是 19 世纪奥地利著名的钢琴演奏家、教育家、作曲家。

④ 译者注：作序者杰拉德·杰伊·萨斯曼（Gerald Jay Sussman），著名计算机科学家，《计算机程序的构造和解释》（SICP，书中采用 Scheme）作者之一，杰拉德·杰伊·萨斯曼与盖伊·史提尔二世（Guy Lewis Steele Jr.，也是 Emacs 作者之一）是 Scheme 的作者。

前　言

为了给 Scheme 二十周年庆祝生日，我们第三次修订了 The Little LISPer，这次我们把书名改为更贴切的 The Little Schemer，并增写了姊妹篇：The Seasoned Schemer。

程序接受数据并产生数据。程序设计需要彻底理解数据；好的程序会反映出所处理数据的结构。大多数的数据集合，并由此延伸到大多数程序，都是可递归表示的。递归是依据自身定义对象或解决问题的方法。

*本书的目标是引导读者学习递归思维模式。*我们首先需要确定与递归概念搭档的语言。这里有三种相对明确的选择：自然语言，如英语；形式化的数学语言；或者是编程语言。自然语言易产生歧义、不严谨且有时候拖沓冗长。这可能在人们日常交流时没什么问题，但对于简明阐述递归这样的严谨概念，这些特征就容易出问题。数学语言则与自然语言相反：其仅通过一些符号就能表述强大的形式化概念。但很不幸，除非接受过数学专业训练，否则一般人理解不了数学语言。技术与数学的结合带给了我们第三种选择——几乎是最理想的选择：编程语言。我们相信编程语言是表达递归概念的最佳方式。编程语言像数学那样，具备将形式化含义赋予一系列符号的能力。但又不同于数学，可以直接体验编程语言——可以运行本书中的程序，观察其行为，然后修改它，再看看修改效果。

Scheme 大概是用来讲解递归概念的最佳编程语言。符号化是 Scheme 的天然特质——程序员不必过多考虑所用语言符号与计算机表述形式之间的关联。递归是 Scheme 的天然计算机制；主要的 Scheme 编程任务是创建（可能的）递归定义。Scheme 程序主要用于交互——程序员可以立即运行代码并观察结果。此外，至本书结束时，我们收获的最大感悟应该是：Scheme 程序结构与程序所操纵数据之间是直接对应的[①]。

虽然 Scheme 程序可以以一种非常形式化的方式来描述，但理解 Scheme 并不需要特别

---

① 译者注：即“代码即数据”。

的数学知识。实际上，本书基于一个 Scheme 两周“速成”介绍课程的讲义整理而成，该课程针对那些没有编程经验且不喜欢数学的学生。这些学生中有许多人正准备从事公共事务方面的工作。我们的信条是：用 Scheme 递归地编写程序本质上是简单的模式识别（Pattern Recognition）。由于我们唯一关心的是递归编程，因此我们仅在 Scheme 的几个招式上下功夫：car、cdr、cons、eq?、null?、zero?、add1、sub1、number?、and、or、quote、lambda、define 以及 cond。事实上，我们选择了完美的 Scheme 编程语言——我们的程序才能如此简洁。

The Little Schemer 和 *The Seasoned Schemer* 并未涉足应用编程领域，但掌握书中的概念则为你打开了理解计算本质的大门。

阅读须知

你应该具备文字阅读能力，识得数字，还要会数数。

致谢

我们要感谢众多贡献者及他们为本书第二、三版提供的帮助。感谢 Bruce Duba、Kent Dybvig、Chris Haynes、Eugene Kohibecker、Richard Salter、George Springer、Mitch Wand 和 David S. Wise 的无数次讨论，为全书内容构思提供了思路。感谢 Ghassan Abbas、Charles Baker、David Boyer、Mike Dunn、Terry Falkenberg、Rob Friedman、John Gateley、Mayer Goldberg、Iqbal Khan、Julia Lawall、Jon Mendelsohn、John Nienart、Jeffrey D. Perotti、Ed Robertson、Anne Shpuntoff、Erich Smythe、Guy Steele、Todd Stein 和 Larry Weisselberg 在草稿阶段提供了许多重要意见。尤其感谢 Bob Filman 反复审核并提出深刻而尖锐的批评。最后，感谢 Nancy Garrett、Peg Fletcher 和 Bob Filman 为设计与 TeX 排版做出的贡献。

最新的第 4 版受惠于 Dorai Sitaram 的 Scheme 排版程序——无比智能的 SLATEX。Kent Dybvig 的 Chez Scheme 让 Scheme 编程变得非常愉快。真诚感谢 Shelaswau Bushnell、Richard Cobbe、David Combs、Peter Drake、Kent Dybvig、Rob Friedman、Steve Ganz、Chris Haynes、Erik Hilsdale、Eugene Kohlbecker、Shriram Krishnamurthi、Julia Lawall、Suzanne Menzel Collin McCurdy、John Nienart、Jon Rossie、Jonathan Sobel、George Springer、Guy Steele、John David Stone、Vikram Subramaniam、Mitch Wand 以及 Melissa Wingard-Phillips 的批评与建议。

读者指南

阅读本书切勿走马观花、一味图快。请仔细阅读，金玉珠玑分散在书中各个角落。

本书很重要，重要的书至少读三遍。阅读时做到一步一个脚印。在未完全理解一章之前，不要尝试跳到下一章。问题按难度递增排序；解决不了早先的问题，后面的问题则将更难回答。

本书以对话方式组织内容，涉及 Scheme 程序的样例趣谈时，对话将在你（读者）和我们（作者）之间进行。请尽可能动手试验阅读到的样例代码。获取 Scheme 是一件很容易的事。尽管在不同 Scheme 实现之间存在微小语法差异（主要是特殊名称和特定函数方面的拼写），但 Scheme 语法基本上是一致的。接下来要玩转 Scheme，还得定义本书引入的 `atom?`、`sub1` 和 `add1`：

```
(define atom?
  (lambda (x)
    (and (not (pair? x)) (not (null? x)))))
```

试一试`(atom? (quote ()))`，看看其是否返回`#f`，以验证 Scheme 正确定义了 `atom?`。实际上，该概念同样适用于诸如 Common Lisp 这样的现代 Lisp 方言。在 Lisp 中要以函数方式定义 `atom?`：

```
(defun atom? (x)
  (not (listp x)))
```

此外，你可能还需要对书中的程序进行稍加修改。典型的，只是做个别变化。框格注释[①]会给出程序试验的建议。“S:”打头的注释代表 Scheme 相关内容，“L:”打头的注释代表 Common Lisp 相关内容。

第 4 章我们会通过 3 个运算函数：*add1*、*sub1* 和 *zero?*来开发一个基本的算术程序。由于 Scheme 并未提供 *add1* 和 *sub1*，因此需要借助内建的加减基本元件来定义它们。进一步的，为了避免冲突，程序的加减法必须以不同的符号：⊕和⊖来分别实现[②]。

本书不涉及任何形式化定义。我们相信你可以构建自己的定义，并因而记住及理解这些定义，这样的效果比我们一口口喂给你吃要好。但在你出山之前，需确保自己彻底理解了 Scheme 的*五法十诫*[③]。学习 Scheme 的钥匙是“模式识别”。Scheme *十诫心法*指出了具体模式。在本书的早期，一些概念出于简单起见讲解得比较浅显；但随着内容的深入，将适时展开描述。你应该也知道，虽然全书讲的是 Scheme，但 Scheme 自身应用的广泛性，可不是我们用介绍性文字就能够清晰阐述的。在掌握了本书内容之后，就可以着手阅读与理解更加全面且高级的其他 Scheme 书籍了。

---

① 译者注：本书大量采用框格（表格）描述的风格，后面就会见到。

② 译者注：这句话的意思是，为了跟 Scheme 自身的+、-号区别开，本书用空心加、减号来实现这段程序中的加减法。

③ 译者注：五法十诫中的五法将在第 1 章提到，十诫在本书开头处提到。

我们的大量示例都跟食物有关，这里有两个原因。首先，食物比抽象符号更形象（你如果正在节食，显然不适合读这本书，开玩笑）。我们希望各种食物能够帮助你理解示例及相关概念。其次，我们打算乱一下你的心智。我们知道学习之路总是充满各种沮丧，一点点障碍将有助你保持清醒。

你可以整装待发了。祝你好运！希望你好好享受荆棘旅程中的激情挑战。

祝你胃口大开！

丹尼尔·弗里德曼[①]
马提亚·费雷森[②]

读者服务

轻松注册成为博文视点社区用户（www.broadview.com.cn），扫码直达本书页面。

- **下载资源**：本书如提供示例代码及资源文件，均可在下载资源处下载。
- **提交勘误**：您对书中内容的修改意见可在提交勘误处提交，若被采纳，将获赠博文视点社区积分（在您购买电子书时，积分可用来抵扣相应金额）。
- **交流互动**：在页面下方读者评论处留下您的疑问或观点，与我们和其他读者一同学习交流。

页面入口：http://www.broadview.com.cn/31725

① 译者注：本书作者之一（Daniel P. Friedman），程序语言研究领域权威。
② 译者注：本书作者之一（Matthias Felleisen），PLT Scheme（Racket）创始人之一。

Scheme 十诫

第一诫

当对一个原子列表 *lat* 进行递归调用时，询问两个有关 *lat* 的问题：(*null? lat*)和 **else**。

当对一个数字 *n* 进行递归调用时，询问两个有关 *n* 的问题：(*zero? n*)和 **else**。

当对一个 S-表达式列表 *l* 进行递归调用时，询问三个有关 *l* 的问题：(*null? lat*)、(*atom? (car l)*)和 **else**。

第二诫

使用 *cons* 来构建列表。

第三诫

构建一个列表的时候，描述第一个典型元素，之后 *cons* 该元素到一般性递归（natural recursion）[①]上。

第四诫

在递归时总是改变至少一个参数。当对一个原子列表 *lat* 进行递归调用时，使用*(cdr lat)*。当对数字 *n* 进行递归调用时，使用时*(sub1 n)*。当对一个 S-表达式 *l* 进行递归调用时，只要是(*null? l*)和(*atom? (car l)*)都不为 true，那么就同时使用(*car l*)和(*cdr l*)。

在递归时改变的参数，必须向着不断接近结束条件而改变。改变的参数必须在结束条件中得以测试：

当使用 *cdr* 时，用 *null?*测试是否结束；
当使用 *sub1* 时，用 *zero?*测试是否结束。

第五诫

当用✣构建一个值时，总是使用 0 作为结束代码行的值，因为加上 0 不会改变加法的值。

当用×构建一个值时，总是使用 1 作为结束代码行的值，因为乘以 1 不会改变乘法的值。

[①] 译者注：可参考 http://stackoverflow.com/questions/32260444/what-is-the-definition-of-natural-recursion。

当用 *cons* 构建一个值时，总是考虑把()作为结束代码行的值。

第六诫

简化工作只在功能正确之后开展。

第七诫

对具有相同性质的 `subparts`（子部件）进行递归调用：

- 列表的子列表。
- 算术表达式的子表达式。

第八诫

使用辅助函数来抽象表示方式。

第九诫

用函数来抽象通用模式。

第十诫

构建函数，一次收集多个值。

Scheme 五法

Scheme 五法之第一法——car 之法则

基本元件 *car* 仅定义为针对非空列表。

Scheme 五法之第二法——cdr 之法则

基本元件 *cdr* 仅定义为针对非空列表。任意非空列表的 *cdr* 总是另一个列表。

Scheme 五法之第三法——cons 之法则

基本元件 *cons* 需要两个参数。第二个参数必须是一个列表。结果是一个列表。

Scheme 五法之第四法——Null?之法则

基本元件 *null?*仅定义为针对列表。

Scheme 五法之第五法——eq?之法则

基本元件 *eq?* 需要两个参数。每个参数都必须是一个非数字的原子。

目　录

目　录

第 1 章

玩具总动员