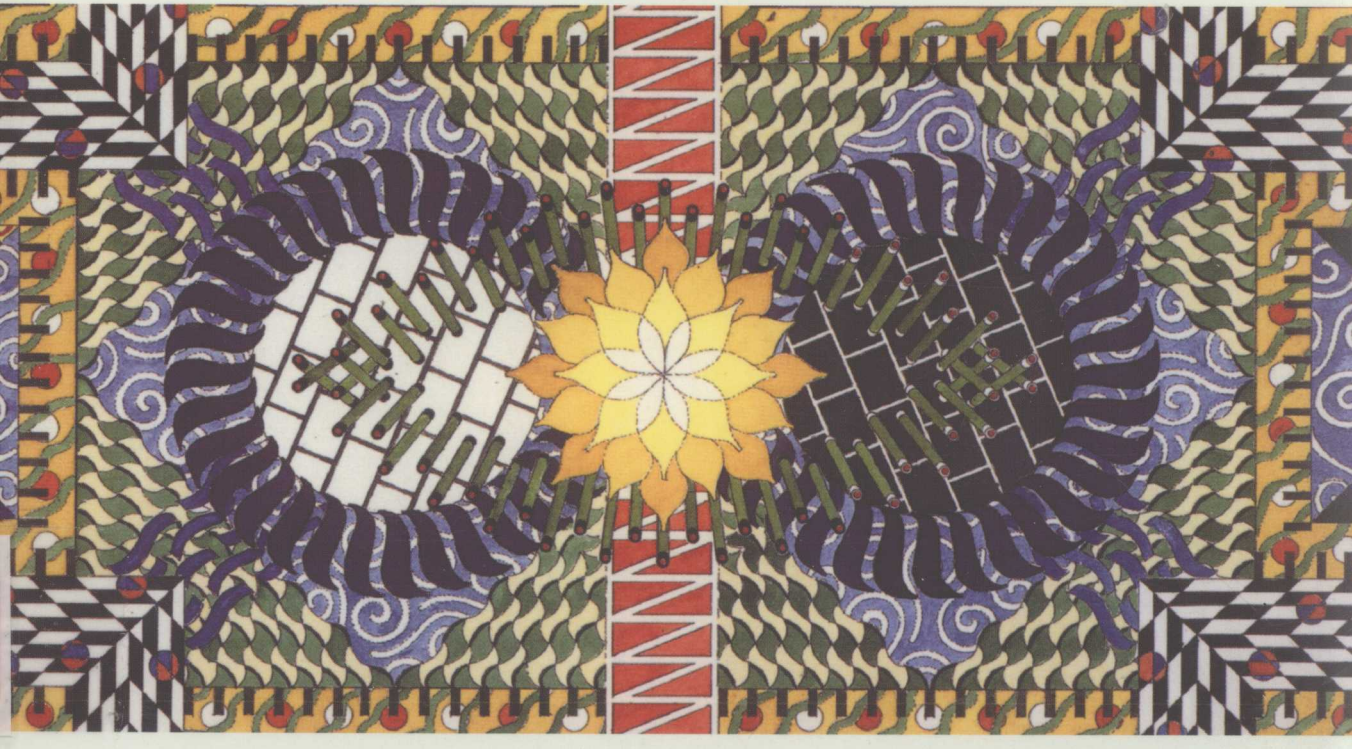
TURING 图灵程序设计丛书

ELSEVIER
爱思唯尔

编译器设计
（第2版）

Engineering a Compiler
Second Edition

[美] Keith D. Cooper　Linda Torczon 著　　郭旭 译



人民邮电出版社
POSTS & TELECOM PRESS

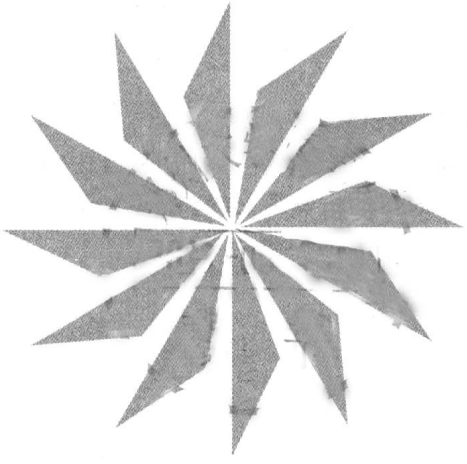
TURING 图灵程序设计丛书

编译器设计（第2版）

Engineering a Compiler

Second Edition

[美] Keith D. Cooper　Linda Torczon 著　　郭旭 译



人民邮电出版社

北　京

图书在版编目（CIP）数据

编译器设计 : 第2版 / (美) 库珀 (Cooper,K.D.), (美) 托克森 (Torczon,L.) 著 ; 郭旭译. -- 北京 : 人民邮电出版社, 2013.1
(图灵程序设计丛书)
书名原文: Engineering a Compiler,Second Edition
ISBN 978-7-115-30194-9

Ⅰ. ①编… Ⅱ. ①库… ②托… ③郭… Ⅲ. ①编译程序—程序设计 Ⅳ. ①TP314

中国版本图书馆CIP数据核字(2012)第288453号

内 容 提 要

本书是编译器设计领域的经典著作，主要从以下四部分详解了编译器的设计过程。

第一部分涵盖编译器前端设计和建立前端所用工具的设计和构建；第二部分探讨从源代码到编译器中间形式的映射，考察前端为优化器和后端所生成代码的种类；第三部分介绍代码优化，同时包含对分析和转换的进一步处理；第四部分专门讲解编译器后端使用的算法。

本书适合作为高等院校计算机专业本科生和研究生编译课程的教材和参考书，也可供相关技术人员参考。

图灵程序设计丛书

编译器设计（第2版）

◆ 著 [美] Keith D. Cooper Linda Torczon
译 郭 旭
责任编辑 朱 巍
执行编辑 刘美英 罗词亮

◆ 人民邮电出版社出版发行 北京市崇文区夕照寺街14号
邮编 100061 电子邮件 315@ptpress.com.cn
网址 http://www.ptpress.com.cn
三河市海波印务有限公司印刷

◆ 开本：800×1000 1/16
印张：37
字数：990千字 2013年1月第1版
印数：1-4 000册 2013年1月河北第1次印刷

著作权合同登记号 图字：01-2012-4880号

ISBN 978-7-115-30194-9

定价：99.00元

读者服务热线：(010)51095186转604 印装质量热线：(010)67129223
反盗版热线：(010)67171154

站在巨人的肩上

Standing on Shoulders of Giants

www.ituring.com.cn

谨将本书献给

- 我们的父母，他们教导我们求知若渴，并支持我们努力锤炼学习技能；
- 我们的子女，他们向我们再次展示了学习与成长的奇妙；
- 当然还有我们的爱人，没有他们的支持本书就不会出版。

关于封面

本书的封面选自画作“The Landing of the Ark”的局部，该画是莱斯大学邓肯楼（Duncan Hall）的天顶画。邓肯楼及其天顶画都是由英国建筑师John Outram设计的，是其在建筑学、装饰艺术及哲学方面所取得的精深造诣的一次完美展示。在邓肯楼的装饰方案中，典礼大厅的天顶画装饰起着重要作用。这种别具一格的天顶画造型彰显了Outram对创世神话的一系列精巧构思。借由巨幅浓色的寓言画表达的思想，Outram成功创造了一个建筑艺术杰作，致使所有步入大厅的参观者都能真切地感受到这幢建筑物的独一无二之处。

第2版与第1版使用了同样的封面，作者意在表明，这个作品包含了一些在建筑学中处于核心地位的重要思想。与Outram的建筑和装饰方案类似，本书既是两位作者职业生涯的智慧结晶，也是传递思想的手段。本书也同Outram的天顶画一样，用新的方法表达了重要的思想。

我们将编译器的设计构建与建筑物的设计构建相联系，旨在揭示这两种截然不同的活动之间的许多相似性。与Outram的多次长时间讨论，使我们了解了维特鲁威[①]的建筑理念：有用、坚固、使人愉悦。这些理念适用于许多结构。同样，这些理念在编译器构建领域也演化成了本书始终如一的主旋律：功能、结构和优雅。功能很重要，生成错误代码的编译器是无用的。结构也很关键，工程细节决定了编译器的效率和健壮性。优雅是灵魂所在，设计完善的编译器是美妙之物，其中的算法和数据结构应流畅有序。

我们很荣幸能用John Outram的作品来装饰本书的封面。

邓肯楼的天顶画是一件有趣的工艺品。Outram在一张纸上画出了图案原稿，然后以1200 dpi的分辨率拍照并扫描，得到大约750 MB数据。该图像被放大，形成234个不同的2英尺×8英尺栅格，它们共同构成了一幅52英尺×72英尺的巨幅图像。这些图像栅格通过12 dpi的丙烯酸喷墨打印机输出到特大号的穿孔乙烯树脂片上。然后这些树脂片被精确地装到2英尺×8英尺的吸音面砖上，最终再把这些砖悬挂到大厅拱顶的铝制框架上。

① 全名为Marcus Vitruvius Pollio，古罗马建筑师，以建筑学巨著《建筑十章》传世。——译者注

前　言

构建编译器的实践方法一直在不断变化，部分是因为处理器和系统的设计会发生变化。例如，当我们在1998年开始写作本书初版时，一些同事对书中指令调度方面的内容颇感疑惑，因为乱序执行威胁到了指令调度，很有可能会使其变得不再重要。现在第2版已经付印，随着多核处理器的崛起和争取更多核心的推动，顺序执行流水线再次展现吸引力，因为这种流水线占地较少，设计者能够将更多核心放置在一块芯片上。短期内，指令调度仍然很重要。

同时，编译器构建社区还将继续产生新的思路和算法，并重新发现原本有效但在很大程度上却被遗忘的旧技术。围绕着寄存器分配中弦图（chordal graph）使用（参见13.5.2节）的最新研究颇为令人振奋。该项工作承诺可以简化图着色分配器（graph-coloring allocator）的某些方面。Brzozowski的算法是一种DFA最小化技术，可以追溯到20世纪60年代早期，但却已有数十年未在编译器课程中讲授了（参见2.6.2节）。该算法提供了一种容易的路径，可以从子集构造（subset construction）的实现得到一个最小化DFA的实现。编译器构建方面的现代课程本该同时包括这两种思想。

那么，为了让学习者准备好进入这个不断变化的领域，我们该如何设计编译器构建课程的结构呢？我们相信，这门课应该使每个学生学会建立新编译器组件和修改现存编译器所需的各项基本技能。学生既需要理解笼统的概念，如链接约定中隐含的编译器、链接器、装载器和操作系统之间的协作，也需要理解微小的细节，如编译器编写者如何减少每个过程调用时保存寄存器的代码总共所占的空间。

第 2 版中的改变

本书提供了两种视角：编译器构建领域中各问题的整体图景，以及各种可选算法方案的详细讨论。在构思本书的过程中，我们专注于该书的可用性，使其既可作为教科书，又可用做专业人士的参考书。为此，我们特别进行了下述改动。

- 改进了阐述思想的流程，以帮助按顺序阅读本书的学生。每章章首简介会解释该章的目的，列出主要的概念，并概述主题相关内容。书中的示例已经重写过，使得章与章之间的内容具有连续性。此外，每章都从摘要和一组关键词开始，以帮助那些会将本书用做参考书的读者。
- 在每节末尾都增加了本节回顾和复习题。复习题用于快速检查读者是否理解了该节的要点。
- 关键术语的定义放在了它们被首次定义和讨论的段落之后。
- 大量修订了有关优化的内容，使其能够更广泛地涵盖优化编译器的各种可能性。

现在的编译器开发专注于优化和代码生成。对于新雇用的编译器编写者来说，他们往往会被指派去将代码生成器移植到新处理器，或去修改优化趟，而不会去编写词法分析器或语法分析器。成功的

编译器编写者必须熟悉优化（如静态单赋值形式的构建）和代码生成领域当前最好的实践技术（如软件流水线）。他们还必须拥有相关的背景和洞察力，能理解未来可能出现的新技术。最后，他们必须深刻理解词法分析、语法分析和语义推敲（semantic elaboration）技术，能构建或修改编译器前端。

本书是一本教科书、一门教程，帮助学生接触到现代编译器领域中的各种关键问题，并向学生提供解决这些问题所需的背景知识。从第1版开始，我们就维持了各主题之间的基本均衡。前端是实用组件，可以从可靠的厂商购买或由某个开源系统改编而得。但是，优化器和代码生成器通常是对特定处理器定制的，有时甚至针对单个处理器型号定制，因为性能严重依赖于所生成代码的底层细节。这些事实影响到了当今构建编译器的方法，它们也应该影响我们讲授编译器构建课程的方法。

本书结构

本书内容划分为篇幅大致相等的四个部分。

- 第一部分（第2章～第4章）涵盖编译器前端及建立前端所用工具的设计和构建。
- 第二部分（第5章～第7章）探讨从源代码到编译器的中间形式的映射，这些章考查前端为优化器和后端所生成代码的种类。
- 第三部分（第8章～第10章）介绍代码优化。第8章提供对优化的概述。第9章和第10章包含了对分析和转换的更深入的处理，本科课程通常略去这两章。
- 第四部分（第11章～第13章）专注于编译器的后端所使用的算法。

编译的艺术性与科学性

编译器构建的内容有两部分，一是将理论应用到实践方面所取得的惊人成就，一是对我们能力受限之处的探讨。这些成就包括：现代词法分析器是通过应用正则语言的理论自动构建识别器而建立的；LR语法分析器使用同样的技术执行句柄识别，进而驱动了一个移进归约语法分析器；数据流分析巧妙有效地将格理论应用到程序分析中；代码生成中使用的近似算法为许多真正困难的问题提供了较好的解。

另一方面，编译器构建也揭示了一些难以解决的复杂问题。用于现代处理器的编译器后端对两个以上的NP完全问题（指令调度、寄存器分配，也许还包括指令和数据安排）采用了近似算法来获取答案。这些NP完全问题，虽然看起来与诸如表达式的代数重新关联这种问题相近（示例见图7-1）。但后者有着大量的解决方案，更糟的是，对于这些NP完全问题来说，所要的解往往取决于编译器和应用程序代码中的上下文信息。在编译器对此类问题近似求解时，会面临编译时间和可用内存上的限制。好的编译器会巧妙地混合理论、实践知识、技术和经验。

打开一个现代优化编译器，你会发现各式各样的技术。编译器使用贪婪启发式搜索来探索很大的解空间，使用确定性有限自动机来识别输入中的单词。不动点算法被用于推断程序行为，通过定理证明程序和代数化简器来预测表达式的值。编译器利用快速模式匹配算法将抽象计算映射到机器层次上的操作。它们使用线性丢番图方程和普瑞斯伯格算术（Pressburger arithmetic）来分析数组下标。最后，编译器使用了大量经典的算法和数据结构，如散列表、图算法和稀疏集实现方法等。

本书尝试同时阐释编译器构建的艺术和科学这两方面内容。通过选取足够广泛的题材，向读者表

明，确实存在一些折中的解决方案，而设计决策的影响可能是微妙而深远的。另一方面，本书也省去了某些长期以来都列入本科编译器构建课程的技术，随着市场、语言和编译器技术或工具可用性方面的改变，这些技术已变得不那么重要了。

讲述方法

编译器构建是一种工程设计实践。每个方案的成本、优点和复杂程度各异，编译器编写者必须在多种备选方案中做出抉择。每个决策都会影响到最终的编译器。最终产品的质量，取决于抉择过程中所做的每一个理性决断。

因而，对于编译器中的许多设计决策来说，并不存在唯一的正确答案。即使在“理解透彻”和“已解决”的问题中，设计和实现中的细微差别都会影响到编译器的行为及其产生的代码的质量。每个决策都涉及许多方面。举例来说，中间表示的选择对于编译器中其余部分有着深刻的影响，无论是时间和空间需求，还是应用不同算法的难易程度。但实际上确定该决策时，通常可供设计者考虑的时间并不多。第5章考察了中间表示的空间需求，以及其他一些应该在选择中间表示时考虑的问题。在本书中其他地方，我们会再次提出该问题，既会在正文中直接提出，也会在习题里间接提出。

本书探索了编译器的设计空间，既从深度上阐释问题，也从广度上探讨可能的答案。它给出了这些问题的某些解决方法，并说明了使用这些方案的约束条件。编译器编写者需要理解这些问题及其答案，以及所作决策对编译器设计的其他方面的影响。只有这样，编写者才能作出理性和明智的选择。

思想观念

本书阐释了我们在构建编译器方面的思想观念，这是在各自超过25年的研究、授课和实践过程中发展起来的。例如，中间表示应该展示最终代码所关注的那些细节，这种理念导致了对底层表示的偏爱。又比如，值应该驻留在寄存器中，直至分配器发现无法继续保留它为止，这种做法产生了使用虚拟寄存器的例子，以及仅在无可避免时才将值存储到内存的例子。每个编译器都应该包括优化，它简化了编译器其余的部分。多年以来的经验，使得我们能够理性地选择书的主题和展现方式。

关于编程习题的选择

编译器构建方面的课程，提供了在一个具体应用程序（编译器）环境中探索软件设计问题的机会，任何具备编译器构建课程背景的学生，都已经透彻理解了该应用程序的基本功能。在多数编译器设计课程中，编程习题发挥了很大的作用。

我们以这样的方式讲授过这门课：学生从头到尾构建一个简单的编译器，从生成的词法分析器和语法分析器开始，结束于针对某个简化的RISC指令集的代码生成器。我们也以另一种方式讲授过这门课程，学生编写程序来解决各个良好自包含的问题，诸如寄存器分配或指令调度。编程习题的选择实际上非常依赖于本课程在相关课程中所扮演的角色。

在某些学校，编译器课程充当高年级的顶级课程，将来自许多其他课程的概念汇集到一个大型的实际设计和实现项目中。在这样的课程中，学生应该为一门简单的语言编写一个完整的编译器，或者修改一个开源的编译器，以支持新的语言或体系结构特性。这门课程可以按本书的内容组织，从头到

尾讲授本书的内容。

在另外一些学校，编译器设计出现在其他课程中，或以其他方式呈现在教学中。此时，编译器设计教师应该专注于算法及其实现，比如局部寄存器分配器或树高重新平衡趟这样的编程习题。在这种情况下，可以选择性讲解本书中的内容，也可以调整讲述的顺序，以满足编程习题的需求。例如，在莱斯大学，我们通常使用简单的局部寄存器分配器作为第一个编程习题，任何具有汇编语言编程经验的学生，都可以理解该问题的基本要素。但这种策略，需要让学生在学习第2章之前，首先接触第13章的内容。

不管采用哪种方案，本课程都应该从其他课程取材。在计算机组织结构、汇编语言编程、操作系统、计算机体系结构、算法和形式语言之间，存在着明显的关联。尽管编译器构建与其他课程的关联不那么明显，但这种关联同等重要。第7章中讨论的字符复制，对于网络协议、文件服务器和Web服务器等应用程序的性能而言，都发挥着关键的作用。第2章中用于词法分析的技术可以应用到文本编辑和URL过滤等领域。第13章中自底向上的局部寄存器分配器是最优离线页面替换算法MIN的“近亲”。

补充材料

还有一些补充的资源可用，可帮助读者改编本书的内容，使之适用于自己的课程。这包括作者在莱斯大学讲授这门课程的一套完整的讲义以及习题答案。读者可以联系本地的Elsevier业务代表，询问如何获取这些补充材料[①]。

致谢

许多人参与了第1版的出版工作，他们的贡献也体现在第2版中。许多人指出了第1版中的问题，包括Amit Saha、Andrew Waters、Anna Youssefi、Ayal Zachs、Daniel Salce、David Peixotto、Fengmei Zhao、Greg Malecha、Hwansoo Han、Jason Eckhardt、Jeffrey Sandoval、John Elliot、Kamal Sharma、Kim Hazelwood、Max Hailperin、Peter Froehlich、Ryan Stinnett、Sachin Rehki、Sağnak Taşırlar、Timothy Harvey和Xipeng Shen，在此向他们致谢。我们还要感谢第2版的审阅者，包括Jeffery von Ronne、Carl Offner、David Orleans、K. Stuart Smith、John Mallozzi、Elizabeth White和Paul C. Anagnostopoulos。Elsevier的产品团队，特别是Alisa Andreola、Andre Cuello和Megan Guiney，在将草稿转换成书的过程中发挥了关键作用。所有这些人都以其深刻的洞察力和无私的帮助，从各个重要方面提升了本书的质量。

最后，在过去5年中，无论是从精神方面，还是从知识方面，许多人都为我们提供了莫大的支持。首先，我们的家庭和莱斯大学的同事都在不断地鼓励我们。特别感谢小女Christine和Carolyn，她们耐心容忍了无数次关于编译器构建方面各种主题的长时间讨论。Nate McFadden以其耐心和出色的幽默感，从开始到出版，一直指导着本书的工作。Penny Anderson对于日常行政事务管理方面的帮助对于本书的完成至关重要。对所有这些人，我们表示衷心的感谢。

① 讲义和习题答案只提供给使用本教材的教师用户。希望使用此材料的教师需登录网站www.textbooks.elsevier.com进行注册，对方审批之后方可下载。申请人可登录图灵社区http://www.ituring.com.cn/book/851的随书下载部分查看补充材料使用指导附件。——编者注

目 录

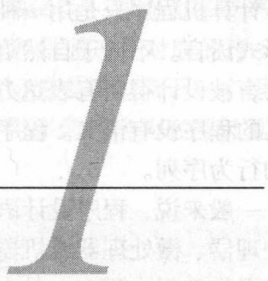
第 1 章 编译概观 ……1
1.1 简介 ……1
1.2 编译器结构 ……4
1.3 转换概述 ……7
1.3.1 前端 ……8
1.3.2 优化器 ……10
1.3.3 后端 ……11
1.4 小结和展望 ……15
第 2 章 词法分析器 ……17
2.1 简介 ……17
2.2 识别单词 ……18
2.2.1 识别器的形式化 ……20
2.2.2 识别更复杂的单词 ……21
2.3 正则表达式 ……24
2.3.1 符号表示法的形式化 ……25
2.3.2 示例 ……26
2.3.3 RE 的闭包性质 ……28
2.4 从正则表达式到词法分析器 ……30
2.4.1 非确定性有限自动机 ……30
2.4.2 从正则表达式到 NFA：Thompson 构造法 ……33
2.4.3 从 NFA 到 DFA：子集构造法 ……34
2.4.4 从 DFA 到最小 DFA：Hopcroft 算法 ……39
2.4.5 将 DFA 用做识别器 ……42
2.5 实现词法分析器 ……43
2.5.1 表驱动词法分析器 ……44
2.5.2 直接编码的词法分析器 ……48
2.5.3 手工编码的词法分析器 ……50
2.5.4 处理关键字 ……53
2.6 高级主题 ……54
2.6.1 从 DFA 到正则表达式 ……54
2.6.2 DFA 最小化的另一种方法：Brzozowski 算法 ……55
2.6.3 无闭包的正则表达式 ……56
2.7 小结和展望 ……57
第 3 章 语法分析器 ……61
3.1 简介 ……61
3.2 语法的表示 ……62
3.2.1 为什么不使用正则表达式 ……62
3.2.2 上下文无关语法 ……63
3.2.3 更复杂的例子 ……66
3.2.4 将语义编码到结构中 ……69
3.2.5 为输入符号串找到推导 ……71
3.3 自顶向下语法分析 ……71
3.3.1 为进行自顶向下语法分析而转换语法 ……73
3.3.2 自顶向下的递归下降语法分析器 ……81
3.3.3 表驱动的 LL(1)语法分析器 ……83
3.4 自底向上语法分析 ……87
3.4.1 LR(1)语法分析算法 ……89
3.4.2 构建 LR(1)表 ……94
3.4.3 表构造过程中的错误 ……103
3.5 实际问题 ……106
3.5.1 出错恢复 ……106
3.5.2 一元运算符 ……107
3.5.3 处理上下文相关的二义性 ……108
3.5.4 左递归与右递归 ……109
3.6 高级主题 ……111
3.6.1 优化语法 ……111
3.6.2 减小 LR(1)表的规模 ……113

3.7 小结和展望……116
第4章 上下文相关分析……120
4.1 简介……120
4.2 类型系统简介……122
4.2.1 类型系统的目标……123
4.2.2 类型系统的组件……126
4.3 属性语法框架……134
4.3.1 求值的方法……137
4.3.2 环……138
4.3.3 扩展实例……138
4.3.4 属性语法方法的问题……143
4.4 特设语法制导转换……146
4.4.1 特设语法制导转换的实现……147
4.4.2 例子……148
4.5 高级主题……155
4.5.1 类型推断中更困难的问题……155
4.5.2 改变结合性……157
4.6 小结和展望……158
第5章 中间表示……162
5.1 简介……162
5.2 图IR……165
5.2.1 与语法相关的树……165
5.2.2 图……168
5.3 线性IR……173
5.3.1 堆栈机代码……173
5.3.2 三地址代码……174
5.3.3 线性代码的表示……175
5.3.4 根据线性代码建立控制流图……176
5.4 将值映射到名字……179
5.4.1 临时值的命名……179
5.4.2 静态单赋值形式……180
5.4.3 内存模型……183
5.5 符号表……186
5.5.1 散列表……187
5.5.2 建立符号表……187
5.5.3 处理嵌套的作用域……188
5.5.4 符号表的许多用途……191
5.5.5 符号表技术的其他用途……193
5.6 小结和展望……193
第6章 过程抽象……198
6.1 简介……198
6.2 过程调用……200
6.3 命名空间……203
6.3.1 类Algol语言的命名空间……203
6.3.2 用于支持类Algol语言的运行时结构……206
6.3.3 面向对象语言的命名空间……210
6.3.4 支持面向对象语言的运行时结构……214
6.4 过程之间值的传递……219
6.4.1 传递参数……219
6.4.2 返回值……222
6.4.3 确定可寻址性……223
6.5 标准化链接……227
6.6 高级主题……231
6.6.1 堆的显式管理……231
6.6.2 隐式释放……234
6.7 小结和展望……237
第7章 代码形式……245
7.1 简介……245
7.2 分配存储位置……247
7.2.1 设定运行时数据结构的位置……248
7.2.2 数据区的布局……249
7.2.3 将值保持在寄存器中……252
7.3 算术运算符……253
7.3.1 减少对寄存器的需求……254
7.3.2 访问参数值……255
7.3.3 表达式中的函数调用……257
7.3.4 其他算术运算符……257
7.3.5 混合类型表达式……258
7.3.6 作为运算符的赋值操作……258
7.4 布尔运算符和关系运算符……259
7.4.1 表示……260
7.4.2 对关系操作的硬件支持……262
7.5 数组的存储和访问……265
7.5.1 引用向量元素……266
7.5.2 数组存储布局……267
7.5.3 引用数组元素……268
7.5.4 范围检查……272

7.6 字符串……273
7.6.1 字符串表示……273
7.6.2 字符串赋值……274
7.6.3 字符串连接……275
7.6.4 字符串长度……276
7.7 结构引用……277
7.7.1 理解结构布局……277
7.7.2 结构数组……278
7.7.3 联合和运行时标记……278
7.7.4 指针和匿名值……279
7.8 控制流结构……281
7.8.1 条件执行……281
7.8.2 循环和迭代……283
7.8.3 case 语句……286
7.9 过程调用……289
7.9.1 实参求值……290
7.9.2 保存和恢复寄存器……291
7.10 小结和展望……292
第 8 章 优化简介……298
8.1 简介……298
8.2 背景……299
8.2.1 例子……300
8.2.2 对优化的考虑……303
8.2.3 优化的时机……305
8.3 优化的范围……306
8.4 局部优化……308
8.4.1 局部值编号……309
8.4.2 树高平衡……314
8.5 区域优化……321
8.5.1 超局部值编号……321
8.5.2 循环展开……324
8.6 全局优化……327
8.6.1 利用活动信息查找未初始化变量……328
8.6.2 全局代码置放……331
8.7 过程间优化……336
8.7.1 内联替换……337
8.7.2 过程置放……340
8.7.3 针对过程间优化的编译器组织结构……344
8.8 小结和展望……345
第 9 章 数据流分析……350
9.1 简介……350
9.2 迭代数据流分析……351
9.2.1 支配性……352
9.2.2 活动变量分析……355
9.2.3 数据流分析的局限性……359
9.2.4 其他数据流问题……361
9.3 静态单赋值形式……365
9.3.1 构造静态单赋值形式的简单方法……366
9.3.2 支配边界……366
9.3.3 放置ϕ函数……369
9.3.4 重命名……372
9.3.5 从静态单赋值形式到其他形式的转换……376
9.3.6 使用静态单赋值形式……379
9.4 过程间分析……383
9.4.1 构建调用图……383
9.4.2 过程间常量传播……385
9.5 高级主题……388
9.5.1 结构性数据流算法和可归约性……388
9.5.2 加速计算支配性的迭代框架算法的执行……391
9.6 小结和展望……393
第 10 章 标量优化……398
10.1 简介……398
10.2 消除无用和不可达代码……401
10.2.1 消除无用代码……402
10.2.2 消除无用控制流……404
10.2.3 消除不可达代码……406
10.3 代码移动……407
10.3.1 缓式代码移动……407
10.3.2 代码提升……413
10.4 特化……414
10.4.1 尾调用优化……415
10.4.2 叶调用优化……416
10.4.3 参数提升……416
10.5 冗余消除……417
10.5.1 值相同与名字相同……417
10.5.2 基于支配者的值编号算法……418

10.6 为其他变换制造时机 ……421
10.6.1 超级块复制 ……421
10.6.2 过程复制 ……422
10.6.3 循环外提 ……423
10.6.4 重命名 ……423
10.7 高级主题 ……425
10.7.1 合并优化 ……425
10.7.2 强度削减 ……429
10.7.3 选择一种优化序列 ……437
10.8 小结和展望 ……438
第 11 章 指令选择 ……441
11.1 简介 ……441
11.2 代码生成 ……443
11.3 扩展简单的树遍历方案 ……445
11.4 通过树模式匹配进行指令选择 ……450
11.4.1 重写规则 ……451
11.4.2 找到平铺方案 ……454
11.4.3 工具 ……457
11.5 通过窥孔优化进行指令选择 ……458
11.5.1 窥孔优化 ……458
11.5.2 窥孔变换程序 ……463
11.6 高级主题 ……465
11.6.1 学习窥孔模式 ……465
11.6.2 生成指令序列 ……466
11.7 小结和展望 ……467
第 12 章 指令调度 ……470
12.1 简介 ……470
12.2 指令调度问题 ……473
12.2.1 度量调度质量的其他方式 ……477
12.2.2 是什么使调度这样难 ……478
12.3 局部表调度 ……478
12.3.1 算法 ……478
12.3.2 调度具有可变延迟的操作 ……481
12.3.3 扩展算法 ……481
12.3.4 在表调度算法中打破平局 ……481
12.3.5 前向表调度与后向表调度 ……482
12.3.6 提高表调度的效率 ……484
12.4 区域性调度 ……485
12.4.1 调度扩展基本程序块 ……486
12.4.2 跟踪调度 ……487
12.4.3 通过复制构建适当的上下文环境 ……488
12.5 高级主题 ……489
12.5.1 软件流水线的策略 ……490
12.5.2 用于实现软件流水线的算法 ……492
12.6 小结和展望 ……495
第 13 章 寄存器分配 ……499
13.1 简介 ……499
13.2 背景问题 ……500
13.2.1 内存与寄存器 ……500
13.2.2 分配与指派 ……501
13.2.3 寄存器类别 ……502
13.3 局部寄存器分配和指派 ……502
13.3.1 自顶向下的局部寄存器分配 ……503
13.3.2 自底向上的局部寄存器分配 ……504
13.3.3 超越单个程序块 ……506
13.4 全局寄存器分配和指派 ……509
13.4.1 找到全局活动范围 ……511
13.4.2 估算全局逐出代价 ……512
13.4.3 冲突和冲突图 ……513
13.4.4 自顶向下着色 ……515
13.4.5 自底向上着色 ……517
13.4.6 合并副本以减小度数 ……518
13.4.7 比较自顶向下和自底向上全局分配器 ……520
13.4.8 将机器的约束条件编码到冲突图中 ……521
13.5 高级主题 ……523
13.5.1 图着色寄存器分配方法的变体 ……523
13.5.2 静态单赋值形式上的全局寄存器分配 ……525
13.6 小结和展望 ……526
附录 A ILOC ……531
附录 B 数据结构 ……540
参考文献 ……559
索引 ……574

第1章 编译概观



本章概述

编译器是一种计算机程序，负责将一种语言编写的程序转换为另一种语言编写的程序。同时，编译器也是一种大型软件系统，包括许多内部组件和算法及其之间复杂的交互。因而，学习编译器构建也就是学习用于转换和改进程序的技术，同时也是一项软件工程实践。本章从概念上概述现代编译器的所有主要组件。

关键词：编译器；解释器；自动转换

1.1 简介

计算机在日常生活中的作用逐年俱增。随着互联网的崛起，计算机及运行于其上的软件提供了通信、新闻、娱乐和安全。嵌入式计算机改变了我们制造汽车、飞机、电话、电视和无线电的方法。从视频游戏到社交网络，计算已经建立了全新的活动范畴。超级计算机预测每日的天气和暴风雨的发展过程。嵌入式计算机可以指挥红绿灯的同步，可以向你的掌上电脑发送电子邮件。

所有这些计算机的应用都依赖软件计算机程序，软件程序基于硬件提供的底层抽象建立了虚拟的工具。几乎所有的软件都是通过称为编译器的工具转换而来。编译器也只是一个计算机程序，它转换其他的计算机程序，并使之准备好执行。本书讲述了用于建立编译器的自动转换的基本技术。本书还描述了编译器构建中出现的许多挑战，以及编译器编写者用于解决这些问题的算法。

编译器

用于转换其他计算机程序的计算机程序。

1. 概念路线图

编译器是一种工具，将一种语言编写的软件转换为另一种语言。为将文本从一种语言转换为另一种语言，该工具必须理解输入语言的形式和内容，或者说语法和语义。它还需要理解输出语言中支配语法和语义的规则。最后，它需要一种方案，以便将内容从源语言映射到目标语言。

典型编译器的结构，通常即衍生于这些简单的观察。编译器有一个前端，用于处理源语言。它还有一个后端，用于处理目标语言。为将前端和后端连接起来，编译器有一种形式化的结构，它用一种中间形式来表示程序，中间形式的语言很大程度上独立于源语言和目标语言。为改进转换，编译器通常包括一个优化器，来分析并重写中间形式。

2. 概述

计算机程序只是用一种程序设计语言编写的抽象操作的序列，程序设计语言是设计用来表示计算的形式语言。不同于自然语言（如中文或葡萄牙文），程序设计语言有着精确的性质和语义。程序设计语言被设计得富有表达力、简洁且清晰，而自然语言允许二义性。程序设计语言应该避免二义性，二义的程序没有语义。程序设计语言应该能指定计算，即可以记录下执行某些任务或产生某些结果所需的行为序列。

一般来说，程序设计语言设计成能允许人类将计算表达为操作的序列。而计算机处理器，下文称为处理器、微处理器或机器，则设计成能执行操作序列。与程序设计语言规定的操作相比，处理器实现的操作在很大程度上是在一个低得多的抽象层次上。例如，程序设计语言通常包括一种将数字输出到文件的简洁方法。在这个单一的程序设计语言语句能够执行前，它必须被逐字地转换为数百个机器操作。

执行这种转换的工具被称为编译器。编译器将以某种语言编写的程序作为输入，产生一个等价的程序作为输出。对于经典意义上的编译器来说，输出程序用某个特定处理器上可用的操作表示，输出程序所针对的处理器通常称为目标机。如果当做一个黑盒子，编译器看起来可以是这样：

通常的“源”语言可能是C、C++、FORTRAN、Java或ML。“目标”语言通常是某种处理器的指令集。

指令集

处理器支持的操作的集合，指令集的总体设计通常称为指令集系统结构（Instruction Set Architecture，ISA）。

一些编译器产生的目标程序是某种面向人类的程序设计语言，而非某种计算机的汇编语言。这些编译器产生的程序需要更进一步的转换，才能在计算机上直接执行。许多研究性编译器产生C程序作为其输出。因为在大多数计算机上都有C编译器可用，这使得目标程序可以在所有这些系统上执行，代价是需要一次额外的编译才能得到最终的目标程序。目标为程序设计语言而非计算机指令集的编译器，通常称为源到源的转换器。

许多其他系统也可视为编译器。例如，可以认为产生PostScript的排版程序是一个编译器。它将文档在印刷纸面上如何布局的规格作为输入，产生PostScript文件作为输出。PostScript只是一种描述图像的语言。因为排版程序的输入是一种可执行的规格，并生成另一种可执行的规格，因此它是一个编译器。

将PostScript转换为像素的代码通常是一个解释器，而不是编译器。解释器将一种可执行规格作为输入，产生的输出是执行该规格的结果。

一些语言，如Perl、Scheme和APL，更多是用解释器实现，而不是编译器。

一些语言采用的转换方案，既包括编译，也包括解释。Java从源代码编译为一种称为字节码的形式，这是一种紧凑的表示，意在减少Java应用程序的下载时间。Java应用程序是通过在对应的Java虚拟机（JVM）上运行字节码来执行的，JVM是一种字节码的解释器。许多JVM的实现包括了一个运行时执行的编译器，有时称为JIT（just-in-time）编译器，它将频繁使用的字节码序列转换为底层计算机的本机码，这使得上文描述的图景进一步复杂化。

虚拟机

虚拟机是针对某种处理器的模拟器，它是针对该机器指令集的解释器。

解释器和编译器有许多共同之处，它们执行许多同样的任务。二者都要分析输入程序，并判定它是否是有效的程序。二者都会建立一个内部模型，表示输入程序的结构和语义。二者都要确定执行期间在何处存储值。然而，解释代码来产生结果，与输出转换后可以执行的目标程序来产生结果，二者有很大不同。本书专注于构建编译器过程中可能出现的问题。但这里讲述的大部分题材，解释器的实现者可能也会发现有用。

3. 为何研究编译器的构建

编译器是一个庞大、复杂的程序。编译器通常包括数十万行代码（即使没有上百万行），组织为多个子系统和组件。编译器的各个部分以复杂的方式进行交互。对编译器一部分作出的设计决策，对其他部分有着重要的影响。因而，编译器的设计和实现，是软件工程中一项很有分量的实践活动。

一个好的编译器是自成天地的，包含了整个计算机科学的一个映像。编译器实际运用了贪心算法（寄存器分配）、启发式搜索技术（表调度）、图算法（死代码消除）、动态规划（指令选择）、有限自动机和下推自动机（词法分析和语法分析）以及不动点算法（数据流分析）。它处理诸如动态分配、同步、命名、局部性、分级存储结构管理和流水线调度等问题。很少有软件系统能汇集同样多且复杂的组件。处理编译器的内部设计和实现在软件工程方面所获取的难得的实践经验，是那些规模较小、复杂度较低的系统所无法提供的。

在计算机科学的主要活动中，编译器发挥着根本的作用：为通过计算机解决问题而做好准备。大多数软件都需要通过编译，该过程的正确性和结果代码的效率，对我们构建大型系统的能力有着直接影响。大多数学生无法满足于仅仅通过阅读而了解这些思想，其中的许多东西都必须得亲自实现才能肯定其价值。因而，学习编译器构建是计算机科学教育的一个重要部分。

编译器是成功将理论应用到实际问题的范例。自动产生词法分析器和语法分析器的工具应用了形式语言理论的结果。这些工具同样可用于文本搜索、网站过滤、文字处理和命令行语言解释器。类型检查和静态分析应用了格理论、数论和其他数学分支的结果，以理解并改进程序。代码生成器使用了树模式匹配、语法分析、动态规划和文本匹配的算法，来自动化指令选择的过程。

但编译器构建领域出现的一些问题仍然未解决，即当前的最佳解决方案仍有改进的余地。尝试设计高级的通用中间表示的努力因复杂性而宣告失败。用于指令调度的主导算法是一个贪心算法，其中包含了几层不相上下的启发式逻辑。编译器显然应该利用交换性和结合性来改进代码，但大多数试图这样做的编译器都只是将表达式重排为某种规范次序。

构建成功的编译器需要精通算法、工程和计划。好的编译器会对困难问题近似求解，它们强调效率，无论是自身的实现还是生成的代码。它们的内部数据结构和知识表示暴露的细节不多不少，既足