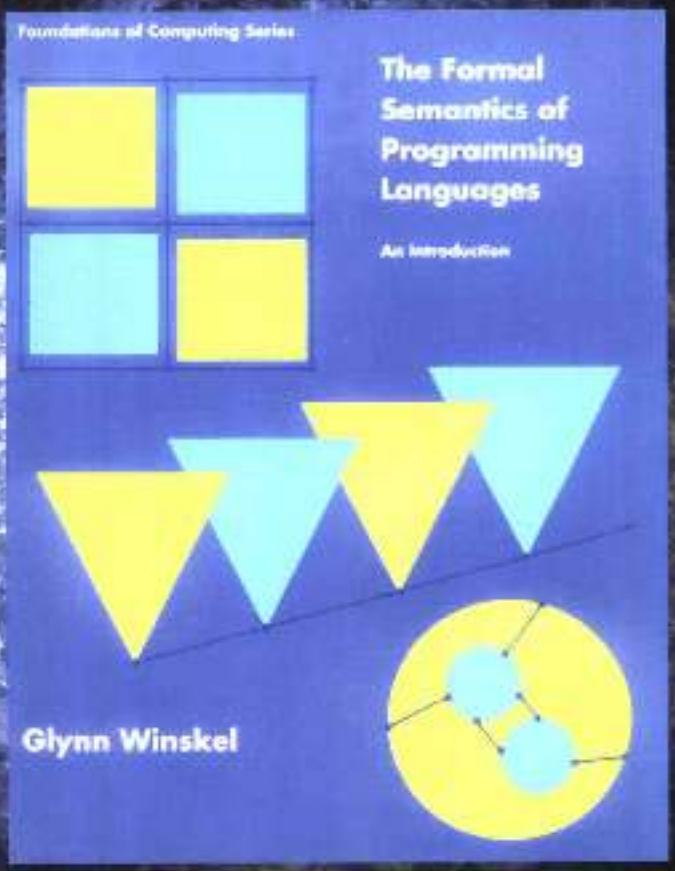
计 算 机 科 学 丛 书

程序设计语言的形式语义

Glynn Winskel 著　宋国新　邵志清　等译



The Formal Semantics of Programming Languages

An Introduction

机械工业出版社 China Machine Press　中信出版社 CITIC PUBLISHING HOUSE

本书是以作者在剑桥大学和 Aarhus 大学的讲义为基础编写的，是一本难得的形式语义学方面的经典著作。书中为初学程序设计语言的语义与逻辑的读者提供了必需的数学知识，介绍了支撑程序设计语言形式语义的数学理论、方法和概念，这些知识可以用于创造、形式化和证明规则，从而可以描述和推导各类程序设计语言的各种成分和性质。

本书内容十分丰富，涉及了集合论、指称语义、操作语义、公理语义、归纳原理、完备性、域论、信息系统、不确定性和并行性、不完备性和不可判定性等内容。同时，每章都包含了丰富的难度不等的练习。

本书适合作为高等院校计算机专业高年级本科生和研究生形式语义课程的教材，也可作为软件开发人员的参考书。

作者简介

Glynn Winskel

曾任丹麦Aarhus大学计算机科学系教授、计算机科学基础研究中心（BRICS）主任，现任剑桥大学计算机实验室教授。

封面设计：张子平

ISBN 7-111-13153-3

9 787111 131533

HZ BOOKS

华章图书

网上购书：www.china-pub.com

北京市西城区百万庄南街1号 100037

读者服务热线：(010)68995259，68995264

读者服务信箱：hzedu@hzbook.com

http://www.hzbook.com

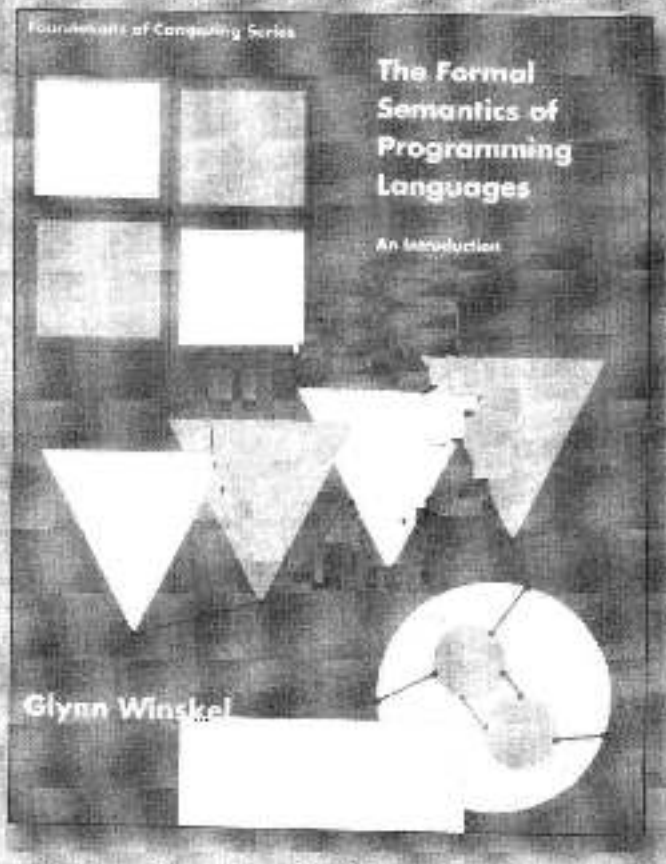
ISBN 7-111-13153-3/TP · 2945

定价：32.00 元

计 算 机 科 学 丛 书

程序设计语言的形式语义

Glynn Winskel 著　宋国新　邵志清　潘俊　孙霖　等译



The Formal Semantics of Programming Languages

An Introduction

机械工业出版社 China Machine Press
中信出版社 CITIC PUBLISHING HOUSE

本书是形式语义方面的一本经典之作，被国内外很多大学选作教材。书中包括集合论基础、指称语义、操作语义、公理语义、归纳原理、归纳定义、完备性、域论、递归方程、递归技术、高阶类型语言、信息系统、递归类型、不确定性和并行性、不完备性和不可判定性等内容。同时，本书始终强调指称语义和操作语义的联系，并给出它们的一致性证明。书中包含了丰富的练习。

本书以作者在剑桥大学和 Aarhus 大学的讲义为基础编写而成，可以作为计算机专业和数学专业的本科生和研究生形式语义课程的教材，也适用于软件开发人员参考。

Glynn Winskel: The Formal Semantics of Programming Languages: An Introduction (ISBN: 0-262-23169-7).

Original English language edition copyright © 1993 by Massachusetts Institute of Technology. All rights reserved.

Simplified Chinese language edition copyright © 2003 by China Machine Press.

All rights reserved. No part of this publication may be reproduced or distributed in any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

本书中文简体版由麻省理工学院出版社授权机械工业出版社和中信出版社共同出版。未经出版者书面许可，不得以任何方式复制或抄袭本书内容。

版权所有，侵权必究。

本书版权登记号：图字：01-2003-8856

图书在版编目(CIP)数据

程序设计语言的形式语义/温斯克尔(Winskel, G.)著；宋国新等译. -北京：机械工业出版社，2004.1

(计算机科学丛书)

书名原文：The Formal Semantics of Programming Languages: An Introduction

ISBN 7-111-13153-3

Ⅰ.程… Ⅱ.①温…②宋… Ⅲ.程序语言-形式语义学-高等学校-教材 Ⅳ.TP301.2

中国版本图书馆 CIP 数据核字(2003)第 092736 号

机械工业出版社(北京市西城区百万庄大街 22 号 邮政编码 100037)

责任编辑：杨海玲

北京昌平奔腾印刷厂印刷 · 新华书店北京发行所发行

2004 年 1 月第 1 版第 1 次印刷

787mm × 1092mm 1/16 · 18.75 印张

印数：0 001 - 4 000 册

定价：32.00 元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

本社购书热线：(010) 68326294

出版者的话

文艺复兴以降，源远流长的科学精神和逐步形成的学术规范，使西方国家在自然科学的各个领域取得了垄断性的优势；也正是这样的传统，使美国在信息技术发展的六十多年间名家辈出、独领风骚。在商业化的进程中，美国的产业界与教育界越来越紧密地结合，计算机学科中的许多泰山北斗同时身处科研和教学的最前线，由此而产生的经典科学著作，不仅擘划了研究的范畴，还揭橥了学术的源变，既遵循学术规范，又自有学者个性，其价值并不会因年月的流逝而减退。

近年，在全球信息化大潮的推动下，我国的计算机产业发展迅猛，对专业人才的需求日益迫切。这对计算机教育界和出版界都既是机遇，也是挑战；而专业教材的建设在教育战略上显得举足轻重。在我国信息技术发展时间较短、从业人员较少的现状下，美国等发达国家在其计算机科学发展的几十年间积淀的经典教材仍有许多值得借鉴之处。因此，引进一批国外优秀计算机教材将对我国计算机教育事业的发展起积极的推动作用，也是与世界接轨、建设真正的世界一流大学的必由之路。

机械工业出版社华章图文信息有限公司较早意识到“出版要为教育服务”。自1998年开始，华章公司就将工作重点放在了遴选、移译国外优秀教材上。经过几年的不懈努力，我们与Prentice Hall，Addison-Wesley，McGraw-Hill，Morgan Kaufmann等世界著名出版公司建立了良好的合作关系，从它们现有的数百种教材中甄选出Tanenbaum，Stroustrup，Kernighan，Jim Gray等大师名家的一批经典作品，以“计算机科学丛书”为总称出版，供读者学习、研究及庋藏。大理石纹理的封面，也正体现了这套丛书的品位和格调。

“计算机科学丛书”的出版工作得到了国内外学者的鼎力襄助，国内的专家不仅提供了中肯的选题指导，还不辞劳苦地担任了翻译和审校的工作；而原书的作者也相当关注其作品在中国的传播，有的还专诚为其书的中译本作序。迄今，“计算机科学丛书”已经出版了近百个品种，这些书籍在读者中树立了良好的口碑，并被许多高校采用为正式教材和参考书籍，为进一步推广与发展打下了坚实的基础。

随着学科建设的初步完善和教材改革的逐渐深化，教育界对国外计算机教材的需求和应用都步入一个新的阶段。为此，华章公司将加大引进教材的力度，在“华章教育”的总规划之下出版三个系列的计算机教材：除“计算机科学丛书”之外，对影印版的教材，则单独开辟出“经典原版书库”；同时，引进全美通行的教学辅导书“Schaum's Outlines”系列组成“全美经典学习指导系列”。为了保证这三套丛书的权威性，同时也为了更好地为学校和老师们服务，华章公司聘请了中国科学院、北京大学、清华大学、国防科技大学、复旦大学、上海交通大学、南京大学、浙江大学、中国科技大学、哈尔滨工业大学、西安交通大学、中国人民大学、北京航空航天大学、北京邮电大学、中山大学、解放军理工大学、郑州大学、湖北工学院、中国国家信息安全测评认证中心等国内重点大学和科研机构在计算机的各个领域的著名学者组成“专家指导委员会”，为我们提供选题意见和出版监督。

这三套丛书是响应教育部提出的使用外版教材的号召，为国内高校的计算机及相关专业

的教学度身订造的。其中许多教材均已为M. I. T.，Stanford，U.C. Berkeley，C. M. U. 等世界名牌大学所采用。不仅涵盖了程序设计、数据结构、操作系统、计算机体系结构、数据库、编译原理、软件工程、图形学、通信与网络、离散数学等国内大学计算机专业普遍开设的核心课程，而且各具特色——有的出自语言设计者之手、有的历经三十年而不衰、有的已被全世界的几百所高校采用。在这些圆熟通博的名师大作的指引之下，读者必将在计算机科学的宫殿中由登堂而入室。

权威的作者、经典的教材、一流的译者、严格的审校、精细的编辑，这些因素使我们的图书有了质量的保证，但我们的目标是尽善尽美，而反馈的意见正是我们达到这一终极目标的重要帮助。教材的出版只是我们的后续服务的起点。华章公司欢迎老师和读者对我们的工作提出建议或给予指正，我们的联系方法如下：

电子邮件：hzedu@hzbook.com
联系电话：（010）68995264
联系地址：北京市西城区百万庄南街1号
邮政编码：100037

专家指导委员会

（按姓氏笔画顺序）

尤晋元	王　珊	冯博琴	史忠植	史美林
石教英	吕　建	孙玉芳	吴世忠	吴时霖
张立昂	李伟琴	李师贤	李建中	杨冬青
邵维忠	陆丽娜	陆鑫达	陈向群	周伯生
周立柱	周克定	周傲英	孟小峰	岳丽华
范　明	郑国梁	施伯乐	钟玉琢	唐世渭
袁崇义	高传善	梅　宏	程　旭	程时端
谢希仁	裘宗燕	戴　葵		

秘　书　组

武卫东　　温莉芳　　刘　江　　杨海玲

译 者 序

“计算机科学与技术”这门学科，顾名思义，要求计算机工作者既具有坚实的计算机科学基础理论方面的涵养，又具备熟练解决实际问题的综合能力。几十年来，计算机科学理论方面的研究结果，为计算机技术与应用的发展提供了有力的支撑；同样，不断迅速发展的应用领域又为理论工作者提出了更多更新的课题。实践表明，正是理论与应用的紧密联系和相互渗透，才使计算机科学不断向前蓬勃发展。

程序设计语言是计算机科学的一个重要组成部分，是理论和应用紧密联系的典型。程序设计语言的每一次技术飞跃都极大地推动了整个计算机科学的发展。尽管程序设计语言品种繁多、层次不一，但总体上讲，与之关系密切的人员主要有三类：一类是程序设计语言的设计者，他们针对具体的应用定义语言的要素；另一类是程序设计语言的实现者，他们针对具体的机器提出语言的实现方案；再一类是程序设计语言的使用者，他们针对具体的问题进行程序设计以给出相应的解决方案。这样，人们可以从各种不同的角度来观察和刻画程序设计语言，从而导致三种主要的语义：指称语义、操作语义和公理语义。这三种语义的精确描述需要严密的数学工具。在长期寻找数学工具的过程中，逐渐形成了有关域论、推导规则、正确性断言等方面的理论和方法，奠定了形义语义学的基础。

本书涉及的程序设计语言既包括命令式语言又包括函数式语言，既包括顺序语言又包括并行语言，既包括确定性语言又包括不确定性语言。书中介绍了支撑这些语言的形式语义的数学理论、方法和概念。读者掌握了这些必要的数学知识，就可以运用它们去创造规则、刻画规则和证明规则，从而可以描述和推导各类程序设计语言的各种成分和性质。

本书第 1 章到第 10 章由潘俊和宋国新翻译，第 11 章到第 14 章由孙霖和宋国新翻译，附录由丁志义翻译，全书由宋国新和邵志清负责整理和统稿。

本书结构合理、内容丰富、蔚为大观，是一本难得的形式语义学的经典著作和教材。译者在翻译过程中获益良多，但限于译者水平，译本中错误和疏漏之处在所难免，读者如果发现译本中的错误，请与译者联系：gxsong@ ecust. edu. cn 或 zshao@ ecust. edu. cn。

译者于华东理工大学

2003 年 11 月 28 日

译者简介

宋国新，教授，博士生导师。1945 年生，1968 年本科毕业于华东水利学院水港系，1983 年毕业于上海交通大学计算机软件专业，获工学硕士学位。1983～1987 年在上海交通大学计算机系任教。1987 年至今在华东理工大学信息学院计算机系任教，1991－1993 年在美国休斯顿大学计算机系作访问学者。曾任华东理工大学计算机系主任兼计算中心主任，现任信息学院副院长，计算机研究所所长。主要研究程序验证、软件测试、软件开发方法、嵌入式系统软硬件协同设计等方向。主持"并行程序的验证"等多项国家自然基金和 863 项目的研究工作。已在国内外主要学术刊物上发表论文 30 多篇。获教育部科技进步二等奖两项，国家教学改革成果二等奖一项，上海市教学改革成果一等奖一项。

邵志清，教授、博士生导师，现任华东理工大学计算机科学与工程系系主任。1966 年生，1986 年毕业于南京大学数学系，获理学学士学位；1989 年毕业于中国科学院软件研究所，获理学硕士学位；1998 年毕业于上海交通大学计算机科学与工程系，获工学博士学位；2001～2002 年以国家杰出访问学者身份赴美国 Rensselaer Polytechnic Institute 计算机系进行合作研究。主持国家自然科学基金项目等多项课题，发表学术论文 60 多篇，获教育部科技进步二等奖；出版教材《离散数学》，受聘为教育部高等学校本科教学工作水平评估中青年专家，获国务院政府特殊津贴、霍英东教育基金会高等院校青年教师奖、上海高校优秀青年教师等奖励。

前　言

在给出程序设计语言的形式语义时，我们着重于建立一个数学模型。目的是使其作为理解程序以及对程序的行为进行论证的基础。数学模型不仅对于各种各样的分析和验证是有用的，同时在更为基础的层次上，应认识到数学模型的重要性在于通过它来精确地刻画程序构造的含义时更能显示出各种精妙细微之处。本书主要介绍支撑形式语义的数学理论、技术和概念。

由于历史的原因，人们通常认为程序设计语言的语义由三条主线构成：

- 操作语义通过规定程序设计语言在抽象机器上的执行过程来描述程序设计语言的含义。我们侧重于“结构化操作语义”，这是 Gordon Plotkin 在 Aarhus 大学的讲座中提倡的方法，由规则规定求值和执行关系，它是语法制导的。
- 指称语义是由 Christopher Strachey 首先提出的一种定义程序设计语言含义的技术，Dana Scott 奠定了它的数学基础。由于使用了完全偏序、连续函数和最小不动点等更为抽象的数学概念，曾一度被称为“数学语义”。
- 公理语义试图通过在程序逻辑的范围内给出证明规则来确定程序设计构造的含义，该方法的代表人物是 R. W. Floyd 和 C. A. R. Hoare。因此，从一开始，公理语义就强调正确性的证明。

然而，将这三种不同类型的语义彼此对立起来是错误的。它们各有各的用处。一个清晰的操作语义有助于语言的实现。公理语义对于特定类型的语言可以给出十分优美的证明系统，对开发和验证程序是有用的。在强大的数学理论支持下，指称语义提供了最有深度和最为广泛的可应用技术。事实上，这些不同的语义类型是相互高度依赖的。例如，要证明公理语义的证明规则是正确的，必须依赖于基本的指称语义或操作语义。要证明语言的实现是正确的，正是要依据指称语义去判断，要求证明操作语义和指称语义是一致的。同时，在操作语义的论证中使用指称语义有巨大的帮助，指称语义往往有远离不重要的实现细节的优势，并给出用来理解计算行为的高层概念。近几年的研究使这几个不同的方法有望得到统一，我们希望能看到指称语义、操作语义和程序逻辑携手共同发展。本书的一个目标是指出操作语义和指称语义是如何取得一致的。

语义学中使用的技术非常是依赖于数理逻辑。如果没有一个好的逻辑知识背景，这些技术并不总能很容易地被计算机科学专业或数学专业的学生所接受。在这里，我们试图以一种完整但尽量基本的方式提出语义。例如，在对操作语义进行介绍之后，引出了归纳定义的论述和推导操作语义的技术，从而使我们处于一个很好的位置，进一步学习抽象的指称语义的基础——完全偏序和连续函数。从操作语义的有限规则到集合上的连续算子，再到连续函数，这种过渡安排有助于理解为什么可计算函数要求有连续性。关于各种归纳原理，包括更一般的良基递归原理的论述，对于在良基关系的集合上定义函数是很重要的。在对递归类型语言的更为深入的研究中，利用信息系统不仅可以给出求解递归域方程的一个基本方法，同时还产生

了能结合操作语义和指称语义的技术。

本书简介

本书以作者在剑桥大学和 Aarhus 大学的讲座内容为基础，主要针对计算机科学专业和数学专业的本科生和研究生而编写，可作为开始学习形式化和推导程序设计语言的方法的教材。本书介绍了必要的数学背景知识，读者可以运用它们去创造、形式化和证明一些规则，使用这些规则可推导各种各样的程序设计语言。本书的内容是基础的，但其中有一些主题来自于最近的研究。书中包含了丰富的从简单到复杂的练习。

本书首先介绍集合论基础，接着是结构化操作语义，并将其作为定义程序设计语言含义的一种方式，同时也介绍了一些基本的证明技术。对指称语义和公理语义是以一个简单的 while 程序语言为例进行说明的，并给出了操作语义和指称语义之间等价的完整证明，以及公理语义的可靠性和相对完备性，也包括哥德尔不完备性定理的一个证明。该定理强调公理语义不可能达到绝对的完备性，这一结论可以从附录中得到支持，附录基于 while 程序介绍了可计算性理论。在域论之后，介绍了指称语义的基础，论述了几种函数式语言的语义和证明方法。最简单的函数式语言是既可以传值调用也可以传名调用求值的递归方程。这些研究工作可以进一步扩展到含有高阶类型和递归类型的语言，其中包括对活性和惰性 λ 演算的论述。本书始终强调指称语义和操作语义的联系，并给出它们的一致性证明。本书较高深的部分之一是递归类型的论述，它要利用信息系统来表示域。在最后一章里介绍了并行程序设计语言，并讨论了不确定性和并行程序的验证方法。

本书的使用方法

下面标明了各章之间的依赖关系，希望有助于阅读、参考和设计讲课内容。例如，“逻辑和计算”课程的内容可以只用第 1 ~ 7 章以及附录。附录包含了第 7 章要用到的概念“可计算性”，如果读者已学习过这方面的知识，则可以跳过附录。此外，像“语义学导论”这样的课程只用第 1 ~ 5 章的内容就可以了，也可以加上第 14 章。第 8、10、12 章的内容可作为“域论”课程的基础内容，有时可能要简单地引用一下第 5 章的内容。第 8 ~ 13 章可以作为“函数式程序设计的数学基础”课程的内容。第 14 章大体上自成体系，只加上第 2 章就可以作为“不确定性和并行性”课程的概论，只是关于模型检查的讨论要用到克纳斯特-塔尔斯基定理，它的证明在第 5 章。

有一些练习包含了一些小的实现任务。在 Aarhus 的课上，我们发现使用诸如 Prolog 之类的语言很有帮助，可以使前面的操作语义的讲述更为生动。在讲述后面几章的语言时，用标准 ML 语言或 Miranda 语言可能更合适一些。

致谢

首先，我要感谢 Dana Scott 和 Gordon Plotkin 所做的基础性工作，他们从根本上影响了本书。在阅读过程中，读者会发现本书受到 Gordon Plotkin 的著作，尤其是他在爱丁堡大学关于完全偏序和指称语义的讲义的极大影响。

在剑桥大学，Tom Melham、Ken Moody、Larry Paulson 和 Andy Pitts 的评论意见对我很有帮助（特别是 Andy 的讲义和对 Eugenio Moggi 工作的评论意见都已写进了域论一章中）。在 Aar-

hus 大学，Mogens Nielsen 提供了有价值的反馈和鼓励，他用最早的草稿开设了课程。Erik Meineche Schmidt 的建议改进了相对完备性定理和哥德尔定理的证明。在 Aarhus 大学，众多的学生提供了改正和建议，我要特别感谢 Henrik Reif Andersen、Torben Brauner、Christian Clausen、Allan Cheng、Urban Engberg、Torben Amtoft Hansen、Ole Hougaard 和 Jakob Seligman。附带要感谢 Bettina Blaaberg Sφrensen，在准备本书的各个阶段，她都快速地阅读并提出建议。感谢 Douglas Gurr 对有关域论的几章提出了诚恳的批评意见。Kim Guldstrand Larsen 对不确定性和并行性一章提出了改进意见。

1991 年秋，Albert Meyer 以本书为基础开设了课程，他和讲师 A. Lent、M. Sheldon 和 C. Yoder 非常友善地对打印错误和证明结构的调整给出了很多宝贵的意见。另外，Albert 慷慨地提供了可计算理论的讲义，作为附录的基础。我感谢他们，但愿他们对最后的结果不感到失望。

感谢 Karen Mφller 在文字录入方面提供的帮助。最后，对 MIT 出版社，特别是 Terry Ehling 的耐心表示感谢。

各章之间的关系

目　录

出版者的话
专家指导委员会
译者序
译者简介
前言
第 1 章　集合论基础 …… 1
1.1　逻辑记号 …… 1
1.2　集合 …… 2
1.2.1　集合与性质 …… 2
1.2.2　一些重要集合 …… 3
1.2.3　集合的构造 …… 3
1.2.4　基本公理 …… 5
1.3　关系与函数 …… 5
1.3.1　λ 记号 …… 5
1.3.2　复合关系与复合函数 …… 6
1.3.3　关系的正象与逆象 …… 7
1.3.4　等价关系 …… 7
1.4　进一步阅读资料 …… 8
第 2 章　操作语义 …… 9
2.1　IMP——一种简单的命令式语言 …… 9
2.2　算术表达式的求值 …… 10
2.3　布尔表达式的求值 …… 13
2.4　命令的执行 …… 14
2.5　一个简单的证明 …… 16
2.6　另一种语义 …… 18
2.7　进一步阅读资料 …… 20
第 3 章　归纳原理 …… 21
3.1　数学归纳法 …… 21
3.2　结构归纳法 …… 22
3.3　良基归纳法 …… 24
3.4　对推导的归纳 …… 27
3.5　归纳定义 …… 30
3.6　进一步阅读资料 …… 31
第 4 章　归纳定义 …… 33
4.1　规则归纳法 …… 33
4.2　特殊的规则归纳法 …… 35
4.3　操作语义的证明规则 …… 36
4.3.1　算术表达式的规则归纳法 …… 36
4.3.2　布尔表达式的规则归纳法 …… 37
4.3.3　命令的规则归纳法 …… 38
4.4　算子及其最小不动点 …… 41
4.5　进一步阅读资料 …… 43
第 5 章　IMP 的指称语义 …… 45
5.1　目的 …… 45
5.2　指称语义 …… 46
5.3　语义的等价性 …… 49
5.4　完全偏序与连续函数 …… 55
5.5　克纳斯特-塔尔斯基定理 …… 59
5.6　进一步阅读资料 …… 60
第 6 章　IMP 的公理语义 …… 61
6.1　基本思想 …… 61
6.2　断言语言 Assn …… 63
6.2.1　自由变量与约束变量 …… 64
6.2.2　代入 …… 65
6.3　断言的语义 …… 66
6.4　部分正确性的证明规则 …… 70
6.5　可靠性 …… 71
6.6　应用霍尔规则的一个示例 …… 73
6.7　进一步阅读资料 …… 75
第 7 章　霍尔规则的完备性 …… 77
7.1　哥德尔不完备性定理 …… 77
7.2　最弱前置条件与可表达性 …… 78
7.3　哥德尔定理的证明 …… 85
7.4　验证条件 …… 86
7.5　谓词转换器 …… 88
7.6　进一步阅读资料 …… 90
第 8 章　域论 …… 91
8.1　基本定义 …… 91

8.2 一个例子——流 …… 92
8.3 完全偏序上的构造 …… 94
8.3.1 离散完全偏序 …… 95
8.3.2 有限积 …… 95
8.3.3 函数空间 …… 98
8.3.4 提升 …… 100
8.3.5 和 …… 102
8.4 元语言 …… 103
8.5 进一步阅读资料 …… 106
第 9 章 递归方程 …… 109
9.1 REC 语言 …… 109
9.2 传值调用的操作语义 …… 110
9.3 传值调用的指称语义 …… 111
9.4 传值调用的语义等价 …… 115
9.5 传名调用的操作语义 …… 118
9.6 传名调用的指称语义 …… 119
9.7 传名调用的语义等价 …… 121
9.8 局部声明 …… 124
9.9 进一步阅读资料 …… 125
第 10 章 递归技术 …… 127
10.1 贝伊克定理 …… 127
10.2 不动点归纳法 …… 129
10.3 良基归纳 …… 136
10.4 良基递归 …… 137
10.5 一个练习 …… 139
10.6 进一步阅读资料 …… 141
第 11 章 高阶类型语言 …… 143
11.1 活性语言 …… 143
11.2 活性操作语义 …… 145
11.3 活性指称语义 …… 146
11.4 活性语义的一致性 …… 148
11.5 惰性语言 …… 156
11.6 惰性操作语义 …… 156
11.7 惰性指称语义 …… 157
11.8 惰性语义的一致性 …… 158
11.9 不动点算子 …… 162
11.10 观察与完全抽象 …… 167
11.11 和 …… 169
11.12 进一步阅读资料 …… 171
第 12 章 信息系统 …… 173
12.1 递归类型 …… 173
12.2 信息系统定义 …… 175
12.3 闭族与斯科特前域 …… 177
12.4 信息系统的完全偏序 …… 180
12.5 构造 …… 182
12.5.1 提升 …… 183
12.5.2 和 …… 185
12.5.3 积 …… 186
12.5.4 提升函数空间 …… 188
12.6 进一步阅读资料 …… 192
第 13 章 递归类型 …… 195
13.1 活性语言 …… 195
13.2 活性操作语义 …… 198
13.3 活性指称语义 …… 200
13.4 活性语义的适用性 …… 204
13.5 活性 λ 演算 …… 208
13.5.1 等式理论 …… 209
13.5.2 不动点算子 …… 211
13.6 惰性语言 …… 215
13.7 惰性操作语义 …… 216
13.8 惰性指称语义 …… 218
13.9 惰性语言的适用性 …… 224
13.10 惰性 λ 演算 …… 225
13.10.1 等式理论 …… 226
13.10.2 不动点算子 …… 227
13.11 进一步阅读资料 …… 230
第 14 章 不确定性和并行性 …… 231
14.1 引言 …… 231
14.2 卫式命令 …… 232
14.3 通信进程 …… 235
14.4 米尔纳的 CCS …… 238
14.5 纯 CCS …… 241
14.6 规范语言 …… 244
14.7 模态 ν 演算 …… 248
14.8 局部模型检查 …… 252
14.9 进一步阅读资料 …… 258
附录 A 不完备性和不可判定性 …… 261
参考文献 …… 273
索引 …… 277

第1章　集合论基础

本章介绍非形式的逻辑记号、集合记号及其基本概念，后面的论证在此基础上展开。它来源于人们的日常语言，用于讨论类似于集合这样的数学对象；但它与我们稍后讨论的程序设计语言的形式语言或形式断言又有不同，二者不应该混淆。

本章是对基础知识的总结，并作为后续章节的参考。建议读者先快速通读一遍，不要求全部理解吸收。

1.1　逻辑记号

我们将使用非形式的逻辑记号来简洁地表示数学命题。例如，对于命题（断言）A 和 B，我们通常使用以下形式的缩写：

- $A \& B$（A 与 B），即 A 和 B 的合取。
- $A \Rightarrow B$（A 蕴涵 B），即如果 A 则 B。
- $A \Longleftrightarrow B$（A 当且仅当 B，简写为 A iff B），即 A 等价于 B。

用析取运算表示命题“A 或 B”，记作 $A \vee B$；用否定运算表示“非 A”，记作 $\neg A$，其值为真当且仅当 A 为假。命题“7 不小于 5”通常不记作 $\neg(7<5)$，而是记作 $7 \not< 5$，以符合我们的表达习惯。

命题中常常会包含变量（未知的或占位用的），例如

$$(x \leqslant 3) \ \& \ (y \leqslant 7)$$

这个命题在变量 x 和 y 分别取小于等于 3 和 7 时为真，否则为假。像 $P(x,y)$ 这样含有变量 x 和 y 的命题称为谓词（或性质、关系、条件），它在 x 和 y 取某一特定值的时候为真或假。

逻辑量词 $\exists$ 读作“存在”，量词 $\forall$ 读作“对于所有的”。断言

$$\exists x. P(x)$$

读作“对于某些 x，$P(x)$”或者“存在 x，使得 $P(x)$”，断言

$$\forall x. P(x)$$

读作“对于所有的 x，$P(x)$”或者“对于任何 x，$P(x)$”。断言

$$\exists x \exists y \cdots \exists z. P(x,y,\cdots,z)$$

简记为

$$\exists x,y,\cdots,z. P(x,y,\cdots,z)$$

并且，断言

$$\forall x \forall y \cdots \forall z. P(x,y,\cdots,z)$$

简记为

$$\forall x,y,\cdots,z. P(x,y,\cdots,z)$$

我们经常希望确定量词对集合 X 的作用范围。我们把 $\forall x. x \in X \Rightarrow P(x)$ 记作 $\forall x \in X. P(x)$，把 $\exists x. x \in X \ \& \ P(x)$ 记作 $\exists x \in X. P(x)$。

还有一个与量词有关的符号，有时我们可能想表达：存在满足 $P(x)$ 性质的惟一 x。约定把

$$(\exists x. P(x)) \ \& \ (\forall y,z. P(y) \ \& \ P(z) \Rightarrow y = z)$$

简记为

$$\exists ! x. P(x)$$

其含义是存在满足性质 P 的 x，且如果任意的 y,z 都满足性质 P，则必有 $y=z$，即存在满足 $P(x)$ 的惟一 x。

1.2 集合

直观上讲，集合就是一组无序的对象，称这些对象为集合的*元素*或者*成员*。我们用 $a \in X$ 表示 a 是集合 X 的一个元素，用 $\{a,b,c,\cdots\}$ 表示由元素 $a,b,c,\cdots$ 组成的集合。

称集合 X 是集合 Y 的*子集*，当且仅当 X 中的所有元素也都是 Y 中的元素，记作 $X \subseteq Y$，即

$$X \subseteq Y \Longleftrightarrow \forall z \in X. z \in Y$$

集合由它的元素惟一确定，称两个集合相等当且仅当它们含有相同的元素。因此，集合 X 和 Y 相等，记作 $X=Y$，当且仅当集合 X 中的每一个元素都是集合 Y 中的一个元素，反之亦然。这里提供了一种方法来证明两个集合 X 和 Y 相等，等价于证明 $X \subseteq Y$ 且 $Y \subseteq X$。

2

1.2.1 集合与性质

集合还可以由性质确定，这时集合的元素恰好满足这个性质，记作：

$$X = \{x \mid P(x)\}$$

表示集合 X 由所有满足性质 $P(x)$ 的 x 组成。

在集合论建立时，人们曾认为任何性质 $P(x)$ 都确定了一个集合

$$\{x \mid P(x)\}$$

但是罗素（Bertrand Russell）发现，这种集合的表示方法会推出矛盾，该结果令人震惊。

罗素悖论表明，按照上述不加限制的方法构造集合，会推出矛盾。由以下步骤开始：考察性质

$$x \notin x$$

它表示“x 不是 x 的元素”。如果性质可以确定集合，则根据该性质的描述，我们就建立了集合

$$R = \{x \mid x \notin x\}$$

这时或者 $R \in R$;或者 $R \notin R$。如果 $R \in R$,那么 R 就是 R 的一个元素,由 R 的定义可以推出 $R \notin R$; 如果 $R \notin R$,由 R 的定义又可以推出 $R \in R$,可见 $R \in R$ 或者 $R \notin R$ 都会推出矛盾,这就是罗素悖论。

人们需要用其他方法来表示集合以避免上面的困境。一个解决办法是规定集合的构造方法,从给定的初始集合开始,规定新的集合只能通过特定的、安全的方法从已有的集合中构造出来。我们假定总存在这样的初始集合和构造方法。避开罗素悖论后,集合论强大的表述能力可以为我们提供数学上的理论支撑。

1.2.2 一些重要集合

本书用特定的符号字母表示常用的基本集合。约定:

- $\emptyset$ 表示空集。
- ω 表示由自然数 0,1,2,…组成的集合。

3

我们一般用{"a","b","c","d","e",…,"z"}这样的集合表示符号,尽管它也可以表示为特定的数字。符号集合上的相等关系是指语法恒等式给出的相等性;两个符号相等当且仅当它们是相同的符号。

1.2.3 集合的构造

我们先给出集合上的几个运算,以便从给定的集合构造新的集合。

概括 如果 X 是一个集合,$P(x)$ 是一个性质,则我们能构造一个集合

$$\{x \in X \mid P(x)\}$$

它也可以记作

$$\{x \mid x \in X \,\&\, P(x)\}$$

它是集合 X 的子集,由 X 中所有满足 $P(x)$ 的元素 x 组成。

我们还可以使用进一步简化的记号,设 $e(x_1,\cdots,x_n)$ 是某一表达式,它对特定的元素 $x_1 \in X_1,\cdots,x_n \in X_n$ 产生一个特定的元素,并且 $P(x_1,\cdots,x_n)$ 是这些 $x_1,\cdots,x_n$ 的一个性质,则我们把

$$\{y \mid \exists x_1 \in X_1,\cdots,x_n \in X_n.\, y = e(x_1,\cdots,x_n) \,\&\, P(x_1,\cdots,x_n)\}$$

简记为

$$\{e(x_1,\cdots,x_n) \mid x_1 \in X_1 \,\&\, \cdots \,\&\, x_n \in X_n \,\&\, P(x_1,\cdots,x_n)\}$$

例如,

$$\{2m+1 \mid m \in \omega \,\&\, m > 1\}$$

表示大于 3 的奇数集合。

幂集 集合的幂集是由该集合所有子集组成的集合,记作

$$\mathscr{P}ow(X) = \{Y \mid Y \subseteq X\}$$

加标集合 假设 I 是集合,对任意的 $i \in I$,存在一个惟一的对象 x_i,(x_i 本身可能是集合),于是

$$\{x_i \mid i \in I\}$$

[4] 是一个集合。称元素 x_i 由元素 $i \in I$ 加标。

并集 两个集合 X 和 Y 的并集定义为

$$X \cup Y = \{a \mid a \in X \text{或} a \in Y\}$$

广义并集 设 X 是集合的集合,则集合

$$\bigcup X = \{a \mid \exists x \in X. a \in x\}$$

称为广义并集。如果 $X = \{x_i \mid i \in I\}$,I 为某个加标集合,则 $\cup X$ 通常记作 $\cup_{i \in I} x_i$。

交集 两个集合 X 和 Y 的交集 $X \cap Y$ 的元素是同时在这两个集合中的元素,即

$$X \cap Y = \{a \mid a \in X \,\&\, a \in Y\}$$

广义交集 设 X 是集合的非空集合,则集合

$$\bigcap X = \{a \mid \forall x \in X. a \in x\}$$

称为广义交集。如果 $X = \{x_i \mid i \in I\}$,I 为非空加标集合,则 $\cap X$ 通常记作 $\cap_{i \in I} x_i$。

积 两个元素 a,b 的序偶记作 (a,b)。如果用集合来表示序偶,则序偶 (a,b) 可以表示成 $\{\{a\},\{a,b\}\}$——这是将序偶编码为集合的一种特殊方式。显然,用这种方式表示的两个序偶相等当且仅当它们的第一个分量和第二个分量都分别相等,即

$$(a,b) = (a',b') \Longleftrightarrow a = a' \,\&\, b = b'$$

不管用什么方法将序偶表示为集合,在证明序偶的性质时,这个性质应该总是成立的。

练习 1.1 对于推荐的序偶的表示方法,试证明上面的性质成立。(不要以为这很容易!在遇到困难的情况下,读者可以参考文献[39]的第 36 页或文献[47]的第 23 页。) □

集合 X 和 Y 的积定义为

$$X \times Y = \{(a,b) \mid a \in X \,\&\, b \in Y\}$$

它是由 $a \in X$ 和 $b \in Y$ 构成的序偶 (a,b) 组成的集合。

推广这个概念,三元组 (a,b,c) 是集合 $(a,(b,c))$,并且积 $X \times Y \times Z$ 是三元组的集合 $\{(x,y,z) \mid x \in X \,\&\, y \in Y \,\&\, z \in Z\}$。更一般地,积 $X_1 \times X_2 \times \cdots \times X_n$ 是 n 元组 $(x_1,x_2,\cdots,x_n) =$
[5] $(x_1,(x_2,(x_3,\cdots)))$ 组成的集合。

不相交并集 在由并运算得到的新集合中,如果每一个元素都能够指明自己来源于哪个集合,就称这样的并运算为不相交的并。可以将集合中的元素拷贝过来加以标志,以便在它们来自不同的集合时使其可以被强制区分。不相交并集定义为

$$X_0 \uplus X_1 \uplus \cdots \uplus X_n = (\{0\} \times X_0) \cup (\{1\} \times X_1) \cup \cdots \cup (\{n\} \times X_n)$$

特别地,对于 $X \uplus Y$,$(\{0\} \times X)$ 和 $(\{1\} \times Y)$ 是不相交的,即

$$(\{0\} \times X) \cap (\{1\} \times Y) = \emptyset$$

这是因为如果交集不为空,那么序偶的第一个分量既为0又为1,这显然是不可能的。

差集 差集定义为

$$X \backslash Y = \{x \mid x \in X \,\&\, x \notin Y\}$$

这个运算从集合 X 中"减去"集合 Y,即去掉 X 中含有的与 Y 相同的元素。

1.2.4 基本公理

由基本集合使用上述构造方法可以得到新的集合。在对集合的认识过程中,人们发现了集合的一个重要特征(称作基本公理),它符合我们对集合的非形式的理解以及构造集合的方式。考察集合 b_0 的一个元素 b_1。b_1 可以是一个基本元素,如整数或者符号,也可以是集合。如果 b_1 是集合,则 b_1 本身必须是由更早的集合构造出来的,依次类推,可以得到一条隶属关系链

$$\cdots b_n \in \cdots \in b_1 \in b_0$$

我们希望这条链能以某个 b_n(它是基本元素或空集)结束。所谓的基本公理就是这样一个命题:任何这样的下降隶属关系链必须是有限的。基本公理表明集合 X 不可以作为自身的元素,否则就会得到一条无限的下降链 $\cdots X \in \cdots \in X \in X$,也就违背了公理。

1.3 关系与函数

集合 X 和 Y 之间的二元关系是幂集 $\mathscr{P}ow(X \times Y)$ 的一个元素,因此,也是关系中序偶的子集。如果 R 是关系 $R \subseteq X \times Y$,则我们通常把 $(x,y) \in R$ 记作 xRy。

从集合 X 到集合 Y 的部分函数是关系 $f \subseteq X \times Y$,它满足 6

$$\forall x,y,y'.\ (x,y) \in f \,\&\, (x,y') \in f \Rightarrow y = y'$$

我们用记号 $f(x) = y$ 表示存在一个 y 使得 $(x,y) \in f$,这时称 $f(x)$ 有定义,否则称 $f(x)$ 无定义。对 $y = f(x)$,可以记作 $f: x \mapsto y$,在 f 被省略时记作 $x \mapsto y$。有时还可以省略 $f(x)$ 的括号,记作 fx。

从集合 X 到集合 Y 的完全函数是这样的部分函数,使得对于所有的 $x \in X$,存在某一 $y \in Y$ 使得 $f(x) = y$。虽然完全函数是部分函数的特例,但通常我们用完全函数来描述事物,除非明确指出某函数是部分函数,否则本书所指的函数都是完全函数。

需要注意的是,关系和函数也是集合。

为了区分,用 $f: X \rightharpoonup Y$ 表示 f 是从 X 到 Y 的部分函数,用 $f: X \to Y$ 表示 f 是从 X 到 Y 的完全函数。同样,用 $(X \rightharpoonup Y)$ 表示从 X 到 Y 的所有部分函数的集合,用 $(X \to Y)$ 表示从 X 到 Y 的所有完全函数的集合。

练习 1.2 在 X,Y 是集合时,为什么可以将 $(X \rightharpoonup Y)$ 和 $(X \to Y)$ 也称为集合? □

1.3.1 λ 记号

λ 记号是描述函数的有用工具,它可以引用一个函数而不必给出函数名。考察函数 $f: X \to Y$,定义域 X 中的任一元素 x 都确定了一个函数值 $f(x)$,它由可能包含变量 x 的表达式 e 来

确切地表示。因此,函数 f 有时也可以记作

$$\lambda x \in X.\, e$$

于是有

$$\lambda x \in X.\, e = \{(x,e) \mid x \in X\}$$

可见,$\lambda x \in X.\, e$ 表示由表达式 e 所确定的输入-输出值的集合。例如,$\lambda x \in \omega.\, (x+1)$ 表示一个后继函数。

1.3.2 复合关系与复合函数

关系和函数可以复合。对于集合 X 与集合 Y 之间的关系 R 和集合 Y 与集合 Z 之间的关系 S,以如下方式定义 R 和 S 的复合:

7

$$S \circ R =_{def} \{(x,z) \in X \times Z \mid \exists y \in Y.\, (x,y) \in R \,\&\, (y,z) \in S\}$$

它是集合 X 与集合 Z 之间的一个关系。同样,函数 $g \circ f: X \to Z$ 表示函数 $f: X \to Y$ 和 $g: Y \to Z$ 的复合。每一个集合 X 都有一个恒等函数 Id_X,其中 $Id_X = \{(x,x) \mid x \in X\}$。

练习 1.3 令 $R \subseteq X \times Y, S \subseteq Y \times Z, T \subseteq Z \times W$。试证明:

1) $T \circ (S \circ R) = (T \circ S) \circ R$ (即复合是可结合的)。

2) $R \circ Id_X = Id_Y \circ R = R$ (即对应于复合,恒等函数是单位元)。 □

函数 $f: X \to Y$ 存在*逆函数* $g: Y \to X$,当且仅当对于所有的 $x \in X$ 有 $g(f(x)) = x$,并且对于所有的 $y \in Y$ 有 $f(g(y)) = y$。此时,我们称集合 X 和集合 Y 是一一对应的。(注意:只有完全函数才有逆函数。)

任何与自然数集 ω 的一个子集一一对应的集合称为是*可数的*。

练习 1.4 设 X 和 Y 是集合,试证明在函数集合 $(X \to \mathscr{P}ow(Y))$ 与关系集合 $\mathscr{P}ow(X \times Y)$ 之间存在一一对应关系。 □

康托尔对角线方法

乔治·康托尔 (Georg Cantor) 是集合论的创始者之一,他在 19 世纪末提出了计算理论中频繁出现的一个重要方法——*对角线方法*。康托尔用这个方法证明了任何集合 X 与它的幂集 $\mathscr{P}ow(X)$ 之间不是一一对应的。这个结论对有限集合是显而易见的,但它对无限集合也同样成立。他用*归谬法*证明,即通过证明假设成立否则导致矛盾。

假设集合 X 与其幂集 $\mathscr{P}ow(X)$ 是一一对应的,设 $\theta: X \to \mathscr{P}ow(X)$ 是一一对应的。构造一个集合

$$Y = \{x \in X \mid x \notin \theta(x)\}$$

显然,Y 是 X 的子集,因此与某个元素 $y \in X$ 相对应,即 $\theta(y) = Y$。我们要问,y 属于 Y 吗?要么 $y \in Y$,要么 $y \notin Y$,但这两种可能性都是荒谬的。如果 $y \in Y$,即 $y \in \theta(y)$,从而 $y \notin Y$。如果 $y \notin Y$,即 $y \notin \theta(y)$,从而 $y \in Y$。因此假设错误,也就是说,不存在集合与它的幂集之间的一一对应关系。

康托尔对角线方法与罗素悖论有异曲同工之处。但是,罗素悖论中的矛盾起源于关于如

何构造集合的基本错误假设，而康托尔论证方法中的矛盾则来源于对想要证明的事实的否定。

为什么称之为对角线方法呢？假设集合 X 与其幂集 $\mathscr{P}ow(X)$ 之间一一对应，则 X 的元素可枚举为 $x_0, x_1, x_2, \cdots, x_n, \cdots$。画一张表来表示一一对应关系 θ，如果 $x_i \in \theta(x_j)$，则第 i 行第 j 列置 1，否则置 0。例如在表 [8]

	$\theta(x_0)$	$\theta(x_1)$	$\theta(x_2)$	$\cdots$	$\theta(x_j)$	$\cdots$
x_0	0	1	1	$\cdots$	1	$\cdots$
x_1	1	1	1	$\cdots$	0	$\cdots$
x_2	0	0	1	$\cdots$	0	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	
x_i	0	1	0	$\cdots$	1	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$		$\vdots$	

中，$x_0 \notin \theta(x_0)$，$x_1 \in \theta(x_0)$，$x_i \in \theta(x_j)$。康托尔的证明过程中起关键作用的是集合 Y。Y 由这张表中 0 和 1 交替的对角线来定义：x_n 是集合 Y 的元素，当且仅当对角线的第 n 个元素的值为 0。

练习 1.5 试证明对于任何集合 X 和 Y，如果 Y 至少包含两个元素，则在 X 与函数集合 $(X \to Y)$ 之间不可能有一一对应关系。 □

1.3.3 关系的正象与逆象

可以将关系（部分函数和完全函数）扩展到子集上的函数。设关系 $R: X \times Y$，对集合 $A \subseteq X$，定义

$$RA = \{y \in Y \mid \exists x \in A.\ (x,y) \in R\}$$

称 RA 为 A 在关系 R 下的*正象*。对集合 $B \subseteq Y$，定义

$$R^{-1}B = \{x \in X \mid \exists y \in B.\ (x,y) \in R\}$$

称 $R^{-1}B$ 为 B 在关系 R 下的*逆象*。当然，对于关系是一个函数的特殊情况，正象和逆象的概念也同样适用。

1.3.4 等价关系

对集合 X 上的关系 R，$R \subseteq X \times X$，如果 R 满足

- 自反性：$\forall x \in X.\ xRx$。
- 对称性：$\forall x, y \in X.\ xRy \Rightarrow yRx$。
- 传递性：$\forall x, y, z \in X.\ xRy\ \&\ yRz \Rightarrow xRz$。

则称 R 为 X 上的*等价关系*。如果 R 是集合 X 上的等价关系，则元素 $x \in X$ 的 R *等价类*是子集 $\{x\}_R =_{def} \{y \in X \mid yRx\}$。 [9]

练习 1.6 设 R 是集合 X 上的等价关系，试证明：对于任何元素 $x, y \in X$，如果 $\{x\}_R \cap \{y\}_R \neq \emptyset$，则 $\{x\}_R = \{y\}_R$。 □

练习 1.7 设 xRy 为集合的集合 X 上的一个关系，关系 xRy 成立当且仅当 X 中集合 x 和 y 是一一对应的，试证明 R 是等价关系。 □

设 R 是集合 X 上的关系，定义 $R^0 = Id_X$（Id_X 是集合 X 上的恒等关系），$R^1 = R$，并假设 R^n 被定义，则定义 $R^{n+1} = R \circ R^n$，因此，R^n 是关系 R 的 n 次复合。定义 R 的传递闭包 R^+ 是关系

$$R^+ = \bigcup_{n \in \omega} R^{n+1}$$

定义 R 的传递自反闭包 R^* 是关系

$$R^* = \bigcup_{n \in \omega} R^n$$

所以，$R^* = Id_X \cup R^+$。

练习 1.8 设 R 是集合 X 上的关系，记它的逆关系为 R^{op}，$R^{op} = \{(y,x) \mid (x,y) \in R\}$。试证明 $(R \cup R^{op})^*$ 是一个等价关系，但 $R^* \cup (R^{op})^*$ 未必是等价关系。 □

1.4 进一步阅读资料

本章非形式地介绍 Zermelo-Fraenkel 的公理集合论，这里介绍的集合论中包含了原子，而没有把符号表示为集合。如果读者需要更多的阅读材料，则推荐两本书：Halmos 的《朴素集合论》[47]是一本很有可读性的入门读物，还有一本较为深入的是 Enderton 的《集合论原理》[39]。

10

第2章 操作语义

本章介绍程序设计语言 **IMP** 的语法，**IMP** 是一种带 **while** 程序的小型语言，其程序在运行时要依靠一连串的显式命令进行状态转换，因此 **IMP** 被称为是一种命令式语言。从形式上看，**IMP** 的行为由一组规则来描述，规则规定了表达式如何计算和命令如何执行。**IMP** 的操作语义也由规则来确定，因此和语言实现的关系比较紧密，例如在 Prolog 语言中就是这样。同时，规则也是证明命令之间等价的基本工具。

2.1 IMP——一种简单的命令式语言

IMP 语言的语法集合有：

- 数集 **N**，它是正整数、负整数和零的集合。
- 真值集 **T** = {**true**, **false**}。
- 存储单元集 **Loc**。
- 算术表达式集 **Aexp**。
- 布尔表达式集 **Bexp**。
- 命令集 **Com**。

假设已知数集 **N** 和存储单元集 **Loc** 的语法结构。例如，**Loc** 为非空字母或后跟数字的非空字符串的集合，而 **N** 为带符号位的正负十进制数的集合。实际上这些正是考虑特定例子时采用的表示。（存储单元常被称作程序变量，但程序变量在本书中表示另外一个概念）。

对于其余语法集合，我们必须说明其元素的构造方式。我们使用 BNF（巴科斯-诺尔范式）的一种变形形式。作为书写这些语法集合的元素的构造规则的方式，构造规则形如：

如果 a_0 和 a_1 是算术表达式，则 $a_0 + a_1$ 也是算术表达式。

显然符号 a_0 和 a_1 被用于代表任意算术表达式，称 a_0 和 a_1 为*元变量*。我们用元变量来表示语法集合的元素，例如，此处的 a_0, a_1 就表示算术表达式集合 **Aexp** 的元素。在描述 **IMP** 的语法中还约定几个特定的字母： [11]

- n, m 表示数集 **N** 中的元素。
- X, Y 表示存储单元集 **Loc** 中的元素。
- a 表示算术表达式集 **Aexp** 中的元素。
- b 表示布尔表达式集 **Bexp** 中的元素。
- c 表示命令集 **Com** 中的元素。

用来表示语法范畴的元变量可以加撇或下标，例如 X, X', X_0, X_1, Y'' 都表示存储单元。

算术表达式集 **Aexp** 的构造规则可描述如下：

$$a ::= n \mid X \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$$

符号"::="读作"可以为",符号"|"读作"或";于是,算术表达式 a 可以是一个数 n,或一个存储单元 X,或由一个 a_0 和 a_1 构造的表达式 a_0+a_1、a_0-a_1、$a_0\times a_1$。

如何用这个规则来分析算术表达式

$$2+3\times4-5$$

呢? 到底是 $2+((3\times4)-5)$,还是 $(2+3)\times(4-5)$,或者其他? 不清楚。这说明上面的规则仅给出了构造新算术表达式的算术表达式的*抽象语法*。而要保证算术表达式是无歧义的,就必须要加上括号。*抽象语法*给出的是语言的分析树,而*具体语法*正是通过括号和运算符的优先级来保证进行惟一正确的语法分析。对我们来说,抽象语法已经足以满足要求了,因为我们关心的是程序设计语言的含义而不是如何描述它们的理论。

IMP 语言完整的构造规则如下:

Aexp:

$$a ::= n \mid X \mid a_0+a_1 \mid a_0-a_1 \mid a_0\times a_1$$

Bexp:

$$b ::= \mathbf{true} \mid \mathbf{false} \mid a_0=a_1 \mid a_0\leqslant a_1 \mid \neg b \mid b_0\wedge b_1 \mid b_0\vee b_1$$

Com:

12

$$c ::= \mathbf{skip} \mid X:=a \mid c_0;c_1 \mid \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ c$$

从集合论的观点来看,上述规则是对 **IMP** 语法集合的*归纳定义*,由此得到的集合是对构造规则封闭的最小集。本书的下两章将会对此作进一步讨论。现在读者只需要知道如何用上述规则构造语法集合的元素就可以了。

当相同语法集合的两个元素 e_0,e_1 是由抽象语法用完全相同的方法构造出来时,它们具有相同的分析树结构。这时,如何表示它们之间的恒等关系呢? 我们用 $e_0\equiv e_1$ 表示 e_0 和 e_1 是恒等的。但是,由数 3 和 5 构造的算术表达式 $3+5$ 在语法上不等于算术表达式 $5+3$ 或 8,尽管我们期望它们都计算出相同的数。所以 $3+5\equiv5+3$ 是错误的,但是 $(3+5)\equiv3+5$ 是正确的!

练习 2.1 如果读者熟悉 ML 语言(参阅[101])或者 Miranda 语言(参阅[22]),试用 **IMP** 的语法集定义数据类型。如果熟悉 Prolog 语言(参阅[31]),试用它给出 **IMP** 构造规则。试设计一个程序,用于检查文法元素 e_0 和 e_1 的 $e_0\equiv e_1$ 是否成立。 □

下面讨论 **IMP** 语法的语义,即程序运行时的行为。

2.2 算术表达式的求值

一般来说,人们在理解用 **IMP** 编写的程序的行为时脑海里都会有一个直观的模型。这些模型的思想主要是:存储单元的内容决定了状态。对于每个状态,算术表达式的求值结果都是一个整数,布尔表达式的求值结果都是一个真值。求值的结果影响命令的执行,从而引起状态的变化。本书对 **IMP** 的行为的形式化描述也体现了这一思想。我们首先定义程序运行的*状态*,然后定义算术表达式和布尔表达式的*求值*,最后定义命令的*执行*。

状态集合 Σ 由存储单元集到数集的函数 σ:**Loc**→**N** 组成。于是,$\sigma(X)$ 是状态 σ 下存储单元 X 的值或内容。

下面讨论在状态 σ 下算术表达式 a 的求值过程。用序偶 $\langle a,\sigma\rangle$ 来表示在状态 σ 下表达式 a 等待求值,并把这个序偶与数集间的求值关系定义为

$$\langle a,\sigma\rangle \to n$$

13

它表示在状态 σ 下表达式 a 的求值结果是 n。称序偶 $\langle a,\sigma\rangle$ 为算术表达式的*格局*。

考察算术表达式 a_0+a_1 的求值过程,其顺序是:

1. 计算 a_0 得到结果 n_0。
2. 计算 a_1 得到结果 n_1。
3. 将 n_0 与 n_1 相加得到 n,n 就是 a_0+a_1 的求值结果。

这个过程通过分别计算加数和被加数的值得到求值结果,这种规范称为*语法制导下的规范*。求值关系的形式化规范由一组规则给出,这组规则和上面对求值过程直观的非形式化的描述非常接近。

语法制导下算术表达式的求值关系按以下规则指定。

整数求值:

$$\langle n,\sigma\rangle \to n$$

显然,任一整数的求值结果就是本身。

存储单元求值:

$$\langle X,\sigma\rangle \to \sigma(X)$$

显然,在一个状态下存储单元的求值结果就是它的内容。

加法求值:

$$\frac{\langle a_0,\sigma\rangle \to n_0 \quad \langle a_1,\sigma\rangle \to n_1}{\langle a_0+a_1,\sigma\rangle \to n}\text{,其中 } n \text{ 是 } n_0 \text{ 与 } n_1 \text{ 的和}$$

减法求值:

$$\frac{\langle a_0,\sigma\rangle \to n_0 \quad \langle a_1,\sigma\rangle \to n_1}{\langle a_0-a_1,\sigma\rangle \to n}\text{,其中 } n \text{ 是 } n_0 \text{ 与 } n_1 \text{ 的差}$$

乘积求值:

$$\frac{\langle a_0,\sigma\rangle \to n_0 \quad \langle a_1,\sigma\rangle \to n_1}{\langle a_0\times a_1,\sigma\rangle \to n}\text{,其中 } n \text{ 是 } n_0 \text{ 与 } n_1 \text{ 的积}$$

我们以加法求值规则为例来说明规则的读法。加法规则可以读作:如果 $\langle a_0,\sigma\rangle\to n_0$ 并且 $\langle a_1,\sigma\rangle\to n_1$,那么 $\langle a_0+a_1,\sigma\rangle\to n$,其中 n 是 n_0 与 n_1 之和。每条规则一般都包含一个前提和一个结论,通常将前提写在上方,结论写在下方,中间用一条实线将它们分开。由前提推出结论称为规则的一个应用。有些规则没有前提部分,比如整数求值规则和存储单元求值规则,但有时还是在上面加一条实线,如下所示:

14

$$\frac{}{\langle n,\sigma\rangle \to n}$$

称前提为空的规则为公理。给定任何一个算术表达式 a、状态 σ 和整数 n，如果从公理开始，根据一组规则可以推出$\langle a,\sigma\rangle \to n$，则称 a 在状态 σ 下求值得到 n。稍后我们还会具体说明。

加法求值规则表明，两个表达式的和等于表达式各自的求值结果的和，至于每一个表达式的求值结果是如何得出的，规则并没有说明。本书不打算对各种数制的求值作详细的分析，因此上述规则仅仅表明了如何从表达式中消去存储单元和运算符号 +、-、× 而得到求值结果。如果要分析十进制或者罗马数字的求值，那么对于乘法这样的运算还需要增加相应的规则。这样的描述在考虑硬件设备时非常有用，但本书仅讨论我们所熟悉的简单算术运算而避免涉及这样的细节。

求值规则由属于相关语法集合的元变量 n, X, a_0, a_1 以及属于状态集合的 σ 组成。用特定的数、存储单元、表达式以及状态来替代规则的元变量，就得到了一个规则实例。例如，设 σ_0 是某个特定状态，存储单元的内容皆为 0，下面是一个规则实例：

$$\frac{\langle 2,\sigma_0\rangle \to 2 \quad \langle 3,\sigma_0\rangle \to 3}{\langle 2\times 3,\sigma_0\rangle \to 6}$$

下面也是一个规则实例：

$$\frac{\langle 2,\sigma_0\rangle \to 3 \quad \langle 3,\sigma_0\rangle \to 4}{\langle 2\times 3,\sigma_0\rangle \to 12}$$

尽管其中的前提和结论是不可能推导出来的。

下面我们介绍推导的结构。考察状态 σ_0 下的表达式 $a \equiv (\mathrm{Init}+5)+(7+9)$ 的求值，其中 Init 是状态 σ_0 下 $\sigma_0(\mathrm{Init})=0$ 的存储单元。根据规则可知，首先要计算出$(\mathrm{Init}+5)$和$(7+9)$两个子表达式的值，而每个子表达式的求值又依赖于其他的求值。实际上，$\langle a,\sigma_0\rangle$ 的求值过程可以看作是依赖于一棵求值树：

$$\frac{\dfrac{\dfrac{}{\langle \mathrm{Init},\sigma_0\rangle \to 0} \quad \dfrac{}{\langle 5,\sigma_0\rangle \to 5}}{\langle (\mathrm{Init}+5),\sigma_0\rangle \to 5} \qquad \dfrac{\dfrac{}{\langle 7,\sigma_0\rangle \to 7} \quad \dfrac{}{\langle 9,\sigma_0\rangle \to 9}}{\langle 7+9,\sigma_0\rangle \to 16}}{\langle (\mathrm{Init}+5)+(7+9),\sigma_0\rangle \to 21}$$

15

我们称这样的结构为推导树，简称推导。推导树由规则的实例构成，每个实例的前提正好是上一层实例的结论；公理位于最顶层，并且公理的上方没有前提部分。最底层的实例的结论称为整个推导的结论。如果某个推导存在结论，就说该结论是从规则可精确推导的。

一般来说，a 在状态 σ 下求值可得 n，记作$\langle a,\sigma\rangle \to n$，当且仅当 n 可以由算术表达式的求值规则推导得到。因此上面的推导可以得到：

$$\langle (\mathrm{Init}+5)+(7+9),\sigma_0\rangle \to 21$$

表示在状态 σ_0 下计算$(\mathrm{Init}+5)+(7+9)$得到我们所期望的结果 21。

我们来考察在某个状态 σ 下计算算术表达式 a 的问题。这就等于寻找一个其结论的左部与$\langle a,\sigma\rangle$相匹配的推导。这样的推导可以用自底向上的方式来查找：从找到结论与$\langle a,\sigma\rangle$匹配的规则开始；若这条规则是公理，则推导完成；否则从前提出发向上构造推导。如果成

功,则把形如$\langle a,\sigma\rangle \to n$的结论填到第一条规则的结论中。

虽然表达式求值的结果是惟一的,但一般来说符合左部与给定的格局匹配的规则会有多条。为了确保找到与结论匹配的一棵推导树(如果存在的话),必须要考虑所有左部与格局匹配的规则,以便看它们是否符合推导的结论。对于符合条件的所有推导,必须并行地构造。

在这种方式下,规则提供了基于查找推导树计算算术表达式的一个算法,因为它易于直接实现,所以这些规则以操作的方式规定了算术表达式的含义或语义,此时我们就说这组规则给出了表达式的*操作语义*。当然,描述易于实现的表达式语义的方法还有很多,任何一种详细描述实现过程的语义都是操作语义,我们给出的方法仅是其中之一。但是,我们选择的语义风格正在流行的一种,因为规则是以语法制导的方式给出的,所以称为*结构化操作语义*;同时,因为它的推导方式类似于自然推理——一种构造形式化证明的方法,所以又称为*自然语义*。后面我们会讨论一些复杂的例子,以帮助读者更深入地理解操作语义。

求值关系确定了算术表达式之间的一个自然的等价关系。定义 [16]

$$a_0 \sim a_1 \text{ 当且仅当 } (\forall n \in \mathbf{N}\ \forall \sigma \in \Sigma.\ \langle a_0,\sigma\rangle \to n \Longleftrightarrow \langle a_1,\sigma\rangle \to n)$$

它表明如果在任何状态下两个算术表达式的求值结果相等,则它们是等价的。

练习 2.2 试用 Prolog 语言、ML 语言或者其他你所熟悉的语言,编写算术表达式求值规则的程序。当然,这需要用 Prolog 或 ML 来描述这种表达式的抽象语法。 □

2.3 布尔表达式的求值

布尔表达式的求值为真值(**true**, **false**)的规则如下:

$$\langle \mathbf{true},\sigma\rangle \to \mathbf{true}$$

$$\langle \mathbf{false},\sigma\rangle \to \mathbf{false}$$

$$\frac{\langle a_0,\sigma\rangle \to n \quad \langle a_1,\sigma\rangle \to m}{\langle a_0 = a_1,\sigma\rangle \to \mathbf{true}}, \text{ 如果 } n \text{ 等于 } m$$

$$\frac{\langle a_0,\sigma\rangle \to n \quad \langle a_1,\sigma\rangle \to m}{\langle a_0 = a_1,\sigma\rangle \to \mathbf{false}}, \text{ 如果 } n \text{ 不等于 } m$$

$$\frac{\langle a_0,\sigma\rangle \to n \quad \langle a_1,\sigma\rangle \to m}{\langle a_0 \leqslant a_1,\sigma\rangle \to \mathbf{true}}, \text{ 如果 } n \text{ 小于等于 } m$$

$$\frac{\langle a_0,\sigma\rangle \to n \quad \langle a_1,\sigma\rangle \to m}{\langle a_0 \leqslant a_1,\sigma\rangle \to \mathbf{false}}, \text{ 如果 } n \text{ 大于 } m$$

$$\frac{\langle b,\sigma\rangle \to \mathbf{true}}{\langle \neg\, b,\sigma\rangle \to \mathbf{false}} \qquad \frac{\langle b,\sigma\rangle \to \mathbf{false}}{\langle \neg\, b,\sigma\rangle \to \mathbf{true}}$$ [17]

$$\frac{\langle b_0,\sigma\rangle \to t_0 \quad \langle b_1,\sigma\rangle \to t_1}{\langle b_0 \wedge b_1,\sigma\rangle \to t}, \quad \text{当 } t_0 \equiv \mathbf{true} \text{ 且 } t_1 \equiv \mathbf{true} \text{ 时 } t \text{ 为 } \mathbf{true}\text{,否则为 } \mathbf{false}$$

$$\frac{\langle b_0,\sigma\rangle \to t_0 \quad \langle b_1,\sigma\rangle \to t_1}{\langle b_0 \vee b_1,\sigma\rangle \to t}, \quad \text{当 } t_0 \equiv \mathbf{true} \text{ 或 } t_1 \equiv \mathbf{true} \text{ 时 } t \text{ 为 } \mathbf{true}\text{,否则为 } \mathbf{false}$$

这组规则的目的是告诉我们如何消去布尔表达式的布尔运算符和连接词符号，归约到一个真值。两个布尔表达式如果在所有状态下的求值结果都相等，则它们是等价的。定义如下：

$$b_0 \sim b_1 \text{ 当且仅当 } \forall t \forall \sigma \in \Sigma. \langle b_0, \sigma \rangle \to t \Longleftrightarrow \langle b_1, \sigma \rangle \to t$$

但是上述规则确定的表达式求值方法效率并不高。例如：用这些规则计算 $b_0 \wedge b_1$，b_0 和 b_1 的值都要求出来。但是如果 b_0 的值为 **false**，则并不需要计算 b_1。更好的策略是先计算 b_0，只有当 b_0 为 **true** 时才继续计算 b_1，这种方法称为*最左顺序*计算。它的求值规则为：

$$\frac{\langle b_0, \sigma \rangle \to \mathbf{false}}{\langle b_0 \wedge b_1, \sigma \rangle \to \mathbf{false}}$$

$$\frac{\langle b_0, \sigma \rangle \to \mathbf{true} \quad \langle b_1, \sigma \rangle \to \mathbf{false}}{\langle b_0 \wedge b_1, \sigma \rangle \to \mathbf{false}}$$

$$\frac{\langle b_0, \sigma \rangle \to \mathbf{true} \quad \langle b_1, \sigma \rangle \to \mathbf{true}}{\langle b_0 \wedge b_1, \sigma \rangle \to \mathbf{true}}$$

练习 2.3 试给出布尔表达式 $b_0 \vee b_1$ 的求值规则，应考虑当 b_0 为真时就不必计算 b_1（因为此时不管 b_1 取何值表达式恒为真）；该规则还要能够描述从左到右求值的方法（当然，根据对称性，存在从右到左的求值方法）。 □

练习 2.4 试给出布尔表达式 $b_0 \vee b_1$ 中 b_0 和 b_1 的并行求值规则。根据该规则，只要 b_0 和 b_1 有一个值为真，就终止计算，此时表达式为真。 □

18

上面我们给出了真值的加法运算或合取运算的机制，读者可能认为对此作了太多的讨论，那么请看下面的练习。

练习 2.5 我们已经讨论了算术表达式和布尔表达式的语义，用同样的方法，试给出串（或表）表达式语义的求值结果。要求给出串类型的基本运算，并给出串表达式和串表达式求值的定义。通过将整数表示为串，整数运算表示串的运算，用你熟悉的语言来实现 **IMP** 的表达式。（证明正确实现了对整数的运算是有一些难度的。） □

2.4 命令的执行

表达式的作用是求得某一状态下的值，而程序和命令的作用则是通过执行来改变状态。当我们运行一个 **IMP** 程序时，假定初始状态下所有的存储单元都已置零，即对于所有的存储单元 X，在*初始状态* σ_0 下都有 $\sigma_0(X) = 0$。我们知道，程序在运行过程中可能*终止*于某一终态，也可能会*发散*而不能到达任何终态。用序偶 $\langle c, \sigma \rangle$ 表示*命令的格局*，它表示在状态 σ 下将要执行 c。定义关系

$$\langle c, \sigma \rangle \to \sigma'$$

它表示在状态 σ 下执行完命令 c 终止于终态 σ'。例如，

$$\langle X := 5, \sigma \rangle \to \sigma'$$

表示在状态 σ 下执行完命令 $X := 5$ 终止于新终态 σ'，其中 σ' 为状态 σ 下将存储单元 X 修改成 5 的状态。σ' 可以用下面的记号来表示。

记号 令 σ 是某个状态，$m \in \mathbf{N}, X \in \mathbf{Loc}$；符号 $\sigma[m/X]$ 表示在状态 σ 下用 m 替换 X 中的内容后得到的状态，即定义

$$\sigma[m/X](Y) = \begin{cases} m & Y = X \\ \sigma(Y) & Y \neq X \end{cases}$$

于是上面的关系可以记作：

$$\langle X := 5, \sigma \rangle \to \sigma[5/X]$$

命令和状态的执行关系由以下规则给出。 [19]

命令的规则

原子命令：

$$\langle \mathbf{skip}, \sigma \rangle \to \sigma$$

$$\frac{\langle a, \sigma \rangle \to m}{\langle X := a, \sigma \rangle \to \sigma[m/X]}$$

顺序命令：

$$\frac{\langle c_0, \sigma \rangle \to \sigma'' \quad \langle c_1, \sigma'' \rangle \to \sigma'}{\langle c_0; c_1, \sigma \rangle \to \sigma'}$$

条件命令：

$$\frac{\langle b, \sigma \rangle \to \mathbf{true} \quad \langle c_0, \sigma \rangle \to \sigma'}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma \rangle \to \sigma'}$$

$$\frac{\langle b, \sigma \rangle \to \mathbf{false} \quad \langle c_1, \sigma \rangle \to \sigma'}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1, \sigma \rangle \to \sigma'}$$

while 循环命令：

$$\frac{\langle b, \sigma \rangle \to \mathbf{false}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma \rangle \to \sigma}$$

$$\frac{\langle b, \sigma \rangle \to \mathbf{true} \quad \langle c, \sigma \rangle \to \sigma'' \quad \langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma'' \rangle \to \sigma'}{\langle \mathbf{while}\ b\ \mathbf{do}\ c, \sigma \rangle \to \sigma'}$$

定义命令之间的等价关系：

$$c_0 \sim c_1 \text{ 当且仅当 } \forall \sigma, \sigma' \in \Sigma. \langle c_0, \sigma \rangle \to \sigma' \Longleftrightarrow \langle c_1, \sigma \rangle \to \sigma'$$

练习 2.6 完成 2.2 节的练习 2.2，试用 Prolog 或者 ML 语言给出布尔表达式的求值规则和命令执行的规则。 □

练习 2.7 令 $w \equiv$ **while true do skip**，试根据推导的形式解释为什么对于任何状态 σ，不

存在状态 σ',使得$\langle w,\sigma\rangle \to \sigma'$。 □

2.5 一个简单的证明

[20] 前面我们用规则给出了语法集合 **Aexp**、**Bexp** 和 **Com** 的操作语义。通过规则,我们定义了两种表达式的求值关系和命令的执行关系。这三种关系都属于一个更一般的概念——*转换关系*。转换关系又称为*转换系统*,在转换系统中将格局看成是某种状态,而将关系看成是表示状态之间的可能的转换或变化。举例来说,以下都可以看作是转换关系:

$$\langle 3,\sigma\rangle \to 3,\quad \langle \mathbf{true},\sigma\rangle \to \mathbf{true},\quad \langle X := 2,\sigma\rangle \to \sigma[2/X]$$

由于用规则给出了 **IMP** 的转换关系,因此用转换关系来证明 **IMP** 操作语义的性质是一个有效而且简便的方法。

下面举例说明。我们考察循环命令 $w \equiv \mathbf{while}\ b\ \mathbf{do}\ c$ 的执行过程,其中 $b \in \mathbf{Bexp}$,$c \in \mathbf{Com}$,当前状态为 σ。我们希望,如果在状态 σ 下 b 为真,则执行 c 后再转移到 w,当 b 为假时,w 执行终止,状态保持不变。即对于所有的状态 σ,σ',有

$$\langle w,\sigma\rangle \to \sigma' \text{ 当且仅当 } \langle \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip},\sigma\rangle \to \sigma'$$

即下面的命题成立。

命题 2.8 令 $w \equiv \mathbf{while}\ b\ \mathbf{do}\ c$,其中 $b \in \mathbf{Bexp}$,$c \in \mathbf{Com}$,则

$$w \sim \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip}$$

证明 对于所有的状态 σ,σ',我们要证明

$$\langle w,\sigma\rangle \to \sigma' \text{ 当且仅当 } \langle \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip},\sigma\rangle \to \sigma'$$

"$\Rightarrow$":设对于所有的状态 σ,σ',有$\langle w,\sigma\rangle \to \sigma'$,则必存在$\langle w,\sigma\rangle \to \sigma'$的推导。考虑这样的推导可能采用的形式。根据命令的规则可知该推导的最终规则或者为

$$\frac{\langle b,\sigma\rangle \to \mathbf{false}}{\langle w,\sigma\rangle \to \sigma} \qquad (1\Rightarrow)$$

或者为

$$\frac{\langle b,\sigma\rangle \to \mathbf{true} \quad \langle c,\sigma\rangle \to \sigma'' \quad \langle w,\sigma''\rangle \to \sigma'}{\langle w,\sigma\rangle \to \sigma'} \qquad (2\Rightarrow)$$

如果是$(1\Rightarrow)$的情况,$\langle w,\sigma\rangle \to \sigma'$的推导必须有以下形式:

$$\frac{\dfrac{\vdots}{\langle b,\sigma\rangle \to \mathbf{false}}}{\langle w,\sigma\rangle \to \sigma}$$

[21]

它包括$\langle b,\sigma\rangle \to \mathbf{false}$ 的推导。根据这个推导,我们可以构造$\langle \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip},\sigma\rangle \to \sigma$的推导:

$$\frac{\dfrac{\vdots}{\langle b,\sigma\rangle \to \mathbf{false}} \quad \dfrac{}{\langle \mathbf{skip},\sigma\rangle \to \sigma}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip},\sigma\rangle \to \sigma}$$

如果是(2⇒)的情况,$\langle w,\sigma\rangle \to \sigma'$的推导必须采用以下形式:

$$\frac{\dfrac{\vdots}{\langle b,\sigma\rangle \to \mathbf{true}} \quad \dfrac{\vdots}{\langle c,\sigma\rangle \to \sigma''} \quad \dfrac{\vdots}{\langle w,\sigma''\rangle \to \sigma'}}{\langle w,\sigma\rangle \to \sigma'}$$

它包括$\langle b,\sigma\rangle \to \mathbf{true}$,$\langle c,\sigma\rangle \to \sigma''$和$\langle w,\sigma''\rangle \to \sigma'$的推导。根据这些推导,我们可以得到$\langle c;w,\sigma\rangle \to \sigma'$的推导:

$$\frac{\dfrac{\vdots}{\langle c,\sigma\rangle \to \sigma''} \quad \dfrac{\vdots}{\langle w,\sigma''\rangle \to \sigma'}}{\langle c;w,\sigma\rangle \to \sigma'}$$

整理后得

$$\frac{\dfrac{\vdots}{\langle b,\sigma\rangle \to \mathbf{true}} \quad \dfrac{\dfrac{\vdots}{\langle c,\sigma\rangle \to \sigma''} \quad \dfrac{\vdots}{\langle w,\sigma''\rangle \to \sigma'}}{\langle c;w,\sigma\rangle \to \sigma'}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip},\sigma\rangle \to \sigma'}$$

在(1⇒)和(2⇒)两种情况下,我们都从

$$\langle w,\sigma\rangle \to \sigma'$$

的一个推导得出了

$$\langle \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip},\sigma\rangle \to \sigma'$$

的一个推导。所以,对于所有的状态σ,σ',

$$\langle w,\sigma\rangle \to \sigma' \Rightarrow \langle \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip},\sigma\rangle \to \sigma'$$

"⇐":设对于所有的状态σ,σ',如果$\langle \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip},\sigma\rangle \to \sigma'$,则有$\langle w,\sigma\rangle \to \sigma'$。 22

假设对于所有的状态σ,σ',$\langle \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip},\sigma\rangle \to \sigma'$成立,则存在一个或者是(1⇐)或者是(2⇐)的两种形式的推导,

$$\frac{\dfrac{\vdots}{\langle b,\sigma\rangle \to \mathbf{false}} \quad \dfrac{}{\langle \mathbf{skip},\sigma\rangle \to \sigma}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip},\sigma\rangle \to \sigma} \qquad (1\Leftarrow)$$

$$\frac{\dfrac{\vdots}{\langle b,\sigma\rangle \to \mathbf{true}} \quad \dfrac{\vdots}{\langle c;w,\sigma\rangle \to \sigma'}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip},\sigma\rangle \to \sigma'} \qquad (2\Leftarrow)$$

其中第一种情况(1⇐)中,因为

$$\frac{}{\langle \mathbf{skip},\sigma\rangle \to \sigma}$$

是 **skip** 的惟一可能的推导，所以我们有 $\sigma' = \sigma$。

不管是从哪个推导，(1⇐)还是(2⇐)，我们都可以构造出$\langle w,\sigma\rangle \to \sigma'$的推导。先讨论较复杂的第二种情况。推导(2⇐)包含了$\langle c;w,\sigma\rangle \to \sigma'$的一个推导，对某个状态 σ''，其形式如下：

$$\frac{\dfrac{\vdots}{\langle c,\sigma\rangle \to \sigma''} \quad \dfrac{\vdots}{\langle w,\sigma''\rangle \to \sigma'}}{\langle c;w,\sigma\rangle \to \sigma'}$$

根据$\langle c,\sigma\rangle \to \sigma''$，$\langle w,\sigma''\rangle \to \sigma'$和$\langle b,\sigma\rangle \to \mathbf{true}$ 的推导，可以构造一个推导

$$\frac{\dfrac{\vdots}{\langle b,\sigma\rangle \to \mathbf{true}} \quad \dfrac{\vdots}{\langle c,\sigma\rangle \to \sigma''} \quad \dfrac{\vdots}{\langle w,\sigma''\rangle \to \sigma'}}{\langle w,\sigma\rangle \to \sigma'}$$

第一种情况比较简单，我们直接由(1⇐)的推导去构造$\langle w,\sigma\rangle \to \sigma'$的推导。(请读者自己证明。)

于是，对于所有的状态 σ,σ'，如果$\langle \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip},\sigma\rangle \to \sigma'$，则有$\langle w,\sigma\rangle \to \sigma'$。从而对于所有的状态 σ,σ'，有

$$\langle w,\sigma\rangle \to \sigma' \text{ 当且仅当} \langle \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip},\sigma\rangle \to \sigma'$$

因此，我们有

23

$$w \sim \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip} \qquad \square$$

这个简单的证明揭示了数学上的一个重要方法：为了证明操作语义的性质，应分别考察所有不同形式的推导。虽然本书不会再有这么详尽的证明例子，但是它的思想在本书的其他章节会反复地体现出来。稍后会介绍另一种证明方法——规则归纳法，它的原理也体现了这一思想。其他的一些证明方法更为抽象，有时更容易混淆。总之，读者在论证操作语义性质的时候，一定要考虑所有的推导形式，这一点十分重要。

2.6 另一种语义

表达式求值关系

$$\langle a,\sigma\rangle \to n \text{ 和} \langle b,\sigma\rangle \to t$$

的求值过程是一步到位的，给定一个表达式和当前状态就能直接得到求值结果。也可以给出表达式求值中每一步的计算规则。我们先定义格局之间的求值关系，例如：

$$\langle a,\sigma\rangle \to_1 \langle a',\sigma'\rangle$$

它表示在状态 σ 下，a 经过一个步骤得到了状态 σ'下的 a'。可以用规则给出它的定义，下列规则给出了从左到右求和的每一步骤的含义：

$$\frac{\langle a_0,\sigma\rangle \to_1 \langle a_0',\sigma\rangle}{\langle a_0 + a_1,\sigma\rangle \to_1 \langle a_0' + a_1,\sigma\rangle}$$

$$\frac{\langle a_1,\sigma\rangle \rightarrow_1 \langle a_1',\sigma\rangle}{\langle n+a_1,\sigma\rangle \rightarrow_1 \langle n+a_1',\sigma\rangle}$$

$$\langle n+m,\sigma\rangle \rightarrow_1 \langle p,\sigma\rangle$$

其中 p 是 n 与 m 之和。

下面我们来分析上述规则如何定义了从左到右依次求和的每一个步骤。求和规则的含义是:如果在状态 σ 下,经过一个步骤 a_0 得到 a_0' 而状态保持不变,那么在状态 σ 下,a_0+a_1 经过一个步骤得到 $a_0'+a_1$,状态也保持不变。因而,求和过程首先求其中一个子表达式的值,然后求另一个子表达式的值,最后将两个值相加。 24

练习 2.9 上面给出了从左到右依次求和每一步骤$\rightarrow_1$ 的规则,请读者给出整数求值和布尔表达式求值每一步骤$\rightarrow_1$ 的规则。你采用的是哪一种策略呢?(是从左到右依次求和还是其他) □

我们用命令格局间的关系

$$\langle c,\sigma\rangle \rightarrow \sigma'$$

定义了在特定状态下命令的完全执行。同样,我们也可以用关系给出执行的每一个步骤。两个命令格局之间的单步关系

$$\langle c,\sigma\rangle \rightarrow_1 \langle c',\sigma'\rangle$$

表示在状态 σ 下,执行命令 c 的一条指令导致了另一个格局,即在状态 σ'下执行命令 c'。举个例子:

$$\langle X:=5;Y:=1,\sigma\rangle \rightarrow_1 \langle Y:=1,\sigma[5/X]\rangle$$

在本例中,当执行完 $X:=5$ 这一步骤后,如果继续执行,需要某种方式来表明 $Y:=1$ 执行后命令为空。我们用一个单独的状态来表示命令为空的格局。于是继续执行上面的指令有:

$$\langle X:=5;Y:=1,\sigma\rangle \rightarrow_1 \langle Y:=1,\sigma[5/X]\rangle \rightarrow_1 \sigma[5/X][1/Y]$$

我们把定义每个步骤执行关系的规则的详细表示留给读者作为练习。读者还要注意命令执行时,单个步骤可能有多种选择。如果

$$\langle b,\sigma\rangle \rightarrow_1 \langle \mathbf{true},\sigma\rangle$$

我们是希望选择

$$\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1,\sigma\rangle \rightarrow_1 \langle c_0,\sigma\rangle$$

还是选择

$$\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1,\sigma\rangle \rightarrow_1 \langle \mathbf{if\ true\ then}\ c_0\ \mathbf{else}\ c_1,\sigma\rangle$$

作为单个步骤呢?对于 **IMP** 语言来说,这个问题不是至关重要的,但是对于并行语言,选择不同的步骤可能会影响执行的终态。 25

练习 2.10 试写出命令格局上的$\rightarrow_1$的一组完整的规则，其中$\rightarrow_1$表示从特定状态下命令的执行的单个步骤。用$\langle c,\sigma\rangle$表示命令的格局，用σ表示命令为空的格局。如果在命令的单步执行中有多种选择，规则还要指明你选择的是哪一种。（证明：$\langle c,\sigma\rangle \rightarrow_1^* \sigma'$，当且仅当$\langle c,\sigma\rangle \rightarrow \sigma'$是有些困难的，需要下面两章介绍的归纳原理。） □

练习 2.11 **IMP** 语言的表达式求值是无副作用的，即求值不会改变状态。如果要考虑副作用，考虑以下形式的求值关系是很自然的：

$$\langle a,\sigma\rangle \rightarrow \langle n,\sigma'\rangle$$

其中，σ'是在原来的状态σ下对a求值后得到的状态。如果我们要在 **IMP** 的算术表达式求值中引入副作用，可以通过加入构造

$$c \textbf{ resultis } a$$

对算术表达式进行扩充，其中c是命令，a是算术表达式。为了计算这样的表达式，首先执行命令c，然后在改变后的状态下对a求值。试用形式化方法实现上述思想，要求先给出语言的完整语法，然后给出它的操作语义。 □

2.7 进一步阅读资料

结构化操作语义应用很广泛，读者可以参考 Gordon Plotkin 1981 年在丹麦 Aarhus 大学的讲义[81]，里面有很多令人信服的例子。在位于法国 Sophia Antipolis 的 INRIA，Gilles Kahn 领导下的一个小组目前正致力于支持结构化操作语义的自动工具的研究，他们的研究重点是能获得最终结果或状态的求值计算或执行过程，这种特殊的操作语义有时称作“自然语义”[26,28,29]。本书后面将要讨论函数式程序设计语言的操作语义、不确定性和并行性，并列出相应的参考资料。关于抽象语法的更多介绍读者可以参考 Wikström[101]、Mosses[68]、Tennent[97]的相关著作或相关章节。

26

第3章 归纳原理

要证明程序的性质,通常采用一种（实际上是一类）证明方法,即归纳法。数学归纳法和结构归纳法是最常用的两种归纳法,它们都属于良基归纳法。良基归纳法是一种功能很强的证明方法。

3.1 数学归纳法

自然数集合的所有元素都是从0开始,通过后继运算依次得到的。*数学归纳法*的证明原理与此类似。

令 $P(n)$ 为自然数 $n=0,1,\cdots$ 的一个性质。数学归纳法的原理为:欲证明所有的自然数 n 都满足性质 $P(n)$,只要证明:

- $P(0)$ 为真。
- 对所有的自然数 m,如果 $P(m)$ 为真,则 $P(m+1)$ 为真。

用逻辑符号来表达,就是:

$$(P(0)\ \&\ \forall m \in \omega.\ P(m) \Rightarrow P(m+1)) \Rightarrow \forall n \in \omega.\ P(n)$$

数学归纳法的原理直观上容易理解:如果已知 $P(0)$ 为真,并且有一个从假设 $P(m)$ 为真而证明 $P(m+1)$ 为真的方法,那么由 $P(0)$ 为真可得 $P(1)$ 为真,再应用该方法可得 $P(2)$、$P(3)$、…都为真。称断言 $P(0)$ 为*归纳奠基*,$P(m)$ 为*归纳假设*,$(\forall m \in \omega.\ P(m) \Rightarrow P(m+1))$ 为*归纳步骤*。

数学归纳法和其他归纳法有个共同的特征,即证明过程中归纳假设是容易得到的,但是由归纳假设到归纳步骤就较难证明。例如要证明所有的自然数都满足性质 P,容易得到归纳假设 $P(m)$,但是归纳步骤 $\forall m \in \omega.\ (P(m) \Rightarrow P(m+1))$ 的证明往往很难甚至无法证明。这是因为归纳假设 $P(m)$ 还不够强,所以要采用更强的蕴涵 $P(m)$ 的归纳假设 $P'(m)$ 。当然找到合适的 $P'(m)$ 也需要技巧,如果归纳假设 $P'(m)$ 太强,归纳步骤 $P'(m+1)$ 证明的难度会增加,有时甚至是无法证明的。

在归纳证明对所有的 m 性质 $Q(m)$ 都满足时,归纳步骤 $Q(m+1)$ 为真可能不仅只依赖于它的前趋 $Q(m)$,还可能与 $Q(m)$ 的前趋有关,这时应该采用比 $Q(m)$ 更强的归纳假设 $P(m)$,$P(m)$ 表示为 $\forall k < m.\ Q(k)$。由此我们得到了归纳法证明的另一种形式: 27

奠基

$$\forall k < 0.\ Q(k)$$

归纳步骤

$$\forall m \in \omega.\ ((\forall k < m.\ Q(k)) \Rightarrow (\forall k < m+1.\ Q(k)))$$

因为自然数都不小于0,所以归纳奠基为真。该归纳步骤等价于

$$\forall m \in \omega.(\forall k < m.Q(k)) \Rightarrow Q(m)$$

这种特殊的数学归纳法称为*串值归纳法*：

$$(\forall m \in \omega.(\forall k < m.Q(k)) \Rightarrow Q(m)) \Rightarrow \forall n \in \omega.Q(n)$$

练习 3.1 试用数学归纳法证明，对所有的自然数，下面的性质 P 都成立：

$$P(n) \Longleftrightarrow_{def} \Sigma_{i=1}^{n}(2i-1) = n^2$$

（记号 $\Sigma_{i=k}^{l} s_i$ 表示 $s_k + s_{k+1} + \cdots + s_l$，其中 l,k 为整数且 $k < l$。） □

练习 3.2 串定义为符号的序列。串的*长度* n 定义为串 $a_1a_2\cdots a_n$ 中符号的个数 n，长度为 0 的串称为空串；串 s 和串 t 可以连结成串 st。试用数学归纳法证明：对两个不同的符号 a 和 b，不存在满足 $au = ub$ 的串 u。 □

3.2 结构归纳法

对所有的算术表达式 a、状态 σ 以及数 m,m'，要证明

$$\langle a,\sigma\rangle \to m \ \&\ \langle a,\sigma\rangle \to m' \Rightarrow m = m'$$

这个"明显的"事实，一种标准证明方法是*结构归纳法*。**IMP** 算术表达式的求值是*确定的*。结构归纳法不仅适用于算术表达式，而且适用于 **IMP** 语言的所有语法集合。

令 $P(a)$ 是算术表达式 a 的一个性质，要证明对所有的算术表达式 a 性质 $P(a)$ 都成立，只要证明：

28

- 对所有的整数 m，$P(m)$ 为真。
- 对所有的存储单元 X，$P(X)$ 为真。
- 对所有的算术表达式 a_0 和 a_1，如果 $P(a_0)$ 和 $P(a_1)$ 为真，则 $P(a_0 + a_1)$ 也为真。
- 对所有的算术表达式 a_0 和 a_1，如果 $P(a_0)$ 和 $P(a_1)$ 为真，则 $P(a_0 - a_1)$ 也为真。
- 对所有的算术表达式 a_0 和 a_1，如果 $P(a_0)$ 和 $P(a_1)$ 为真，则 $P(a_0 \times a_1)$ 也为真。

断言 $P(a)$ 称为*性质*[⊖]。结构归纳法的原理是：要证明对所有的算术表达式性质 $P(a)$ 为真，只要证明对所有的原子表达式性质 $P(a)$ 为真，并且对所有的算术表达式的各种构成方法该性质也保持为真。这个原理很直观，算术表达式的定义正是按上述规则构造出来的。用逻辑符号可以更简洁地表示：

$$\begin{aligned}
&(\forall m \in \mathbf{N}.P(m)) \ \&\ (\forall X \in \mathbf{Loc}.P(X)) \ \& \\
&(\forall a_0,a_1 \in \mathbf{Aexp}.P(a_0) \ \&\ P(a_1) \Rightarrow P(a_0 + a_1)) \ \& \\
&(\forall a_0,a_1 \in \mathbf{Aexp}.P(a_0) \ \&\ P(a_1) \Rightarrow P(a_0 - a_1)) \ \& \\
&(\forall a_0,a_1 \in \mathbf{Aexp}.P(a_0) \ \&\ P(a_1) \Rightarrow P(a_0 \times a_1)) \\
&\Rightarrow \\
&\forall a \in \mathbf{Aexp}.P(a)
\end{aligned}$$

⊖ 原文为 induction hypothesis，疑误，本书译为"性质"。因为按惯例，归纳假设只在归纳步骤中出现。——译者注

事实上很明显，上述条件不仅蕴涵着 $\forall a \in \mathbf{Aexp}.\ P(a)$，而且它与 $\forall a \in \mathbf{Aexp}.\ P(a)$ 还是等价的。

有时候用结构归纳法的退化形式也就足够了。要证明对所有的表达式性质 P 都满足，不必证明对所有的子表达式性质 P 都满足，只要用分情形论证的方法证明对表达式的不同结构形式性质 P 满足即可。例如，对算术表达式使用分情形论证的方法使用了这样的事实：如果

$$
\begin{aligned}
&(\forall m \in \mathbf{N}.\ P(m))\ \& \\
&(\forall X \in \mathbf{Loc}.\ P(X))\ \& \\
&(\forall a_0, a_1 \in \mathbf{Aexp}.\ P(a_0 + a_1))\ \& \\
&(\forall a_0, a_1 \in \mathbf{Aexp}.\ P(a_0 - a_1))\ \& \\
&(\forall a_0, a_1 \in \mathbf{Aexp}.\ P(a_0 \times a_1))
\end{aligned}
$$

29

则有 $\forall a \in \mathbf{Aexp}.\ P(a)$。

作为结构归纳法证明的一个例子，下面我们来证明算术表达式求值的确定性。

命题 3.3 对于所有的算术表达式 a、状态 σ 和整数 m, m'，有

$$\langle a, \sigma \rangle \to m\ \&\ \langle a, \sigma \rangle \to m' \Rightarrow m = m'$$

证明 我们用结构归纳法来证明，使用性质 $P(a)$，其中

$$P(a) \text{ 当且仅当 } \forall \sigma, m, m'.\ (\langle a, \sigma \rangle \to m\ \&\ \langle a, \sigma \rangle \to m' \Rightarrow m = m')$$

为了简便，把 $\langle a, \sigma \rangle \to m\ \&\ \langle a, \sigma \rangle \to m'$ 记作 $\langle a, \sigma \rangle \to m, m'$。根据算术表达式的结构，使用结构归纳法证明分为下面几种情况：

$a \equiv n$：如果 $\langle a, \sigma \rangle \to m, m'$，则 $m = m' = n$，因为对整数求值只有一条规则可以应用。

$a \equiv a_0 + a_1$：如果 $\langle a, \sigma \rangle \to m, m'$，则根据加法规则必存在 m_0、m_1 和 m_0'、m_1' 满足：

$$\langle a_0, \sigma \rangle \to m_0 \text{ 且 } \langle a_1, \sigma \rangle \to m_1 \quad (m = m_0 + m_1)$$

以及

$$\langle a_0, \sigma \rangle \to m_0' \text{ 且 } \langle a_1, \sigma \rangle \to m_1' \quad (m' = m_0' + m_1')$$

由应用于 a_0 和 a_1 的归纳假设，得 $m_0 = m_0'$，$m_1 = m_1'$。于是有 $m = m_0 + m_1 = m_0' + m_1' = m'$。

同理可证明其他几种情况。这样，根据结构归纳法我们就证明了对所有的 $a \in \mathbf{Aexp}$ 性质 $P(a)$ 都满足。 □

结构归纳法还可以证明算术表达式的终止性，同时布尔表达式也有相应的性质。

练习 3.4 试用结构归纳法证明算术表达式求值具有终止性，即对于所有的算术表达式 a 和状态 σ，存在整数 m 使得 $\langle a, \sigma \rangle \to m$。 □

练习 3.5 利用算术表达式的上述性质，试用结构归纳法证明布尔表达式的求值是确定的，而且还是完全的（即具有终止性）。 □

练习 3.6 如果用结构归纳法来证明命令的执行是确定的，会出现什么问题？（在 3.4 节我们将对推导使用结构归纳法来证明命令执行的确定性。）□

30

3.3 良基归纳法

数学归纳法和结构归纳法都是良基归纳法的特例，良基归纳法是一种功能强大的证明方法。结构归纳法的实质是把表达式分解为子表达式，不断分解一直到原子表达式（因为原子表达式不可再分解）。如果一个性质对某个表达式不成立，那么必然存在不满足该性质的最小表达式，但这个最小表达式的所有子表达式都满足该性质。这证实了结构归纳法的原理：要证明某个性质对所有表达式都成立，只要证明如果任一表达式的所有子表达式都满足该性质，则这个表达式也满足该性质。类似地，对于自然数，如果某一性质不是对所有的自然数都成立，那么必存在一个不满足该性质的最小的自然数。结构归纳法的子表达式关系与数学归纳法的自然数前趋关系有着共同的本质特征，即避免无穷下降链。如果一个关系支持良基归纳法，则避免无穷下降链也是关系的重要特征。

定义 设$\prec$是集合 A 上的二元关系，如果不存在由 A 中元素构成的无穷下降链$\cdots\prec a_i\prec\cdots\prec a_1\prec a_0$，则称$\prec$为*良基关系*。若 $a\prec b$ 则称 a 是 b 的*前趋*。

良基关系必是*非自反的*，也就是说，$a\prec a$ 不成立，否则就会有一条无穷下降链$\cdots\prec a\prec\cdots\prec a\prec a$。一般我们用$\preceq$来表示关系$\prec$的自反闭包，即

$$a \preceq b \Longleftrightarrow a = b \text{ 或 } a \prec b$$

有时还可以用极小元的概念来定义良基关系。

命题 3.7 *设$\prec$是集合 A 上的二元关系，称关系$\prec$是良基的当且仅当 A 的所有非空子集 Q 都含有一个极小元 m。即*

$$m \in Q \ \&\ \forall b \prec m.\ b \notin Q$$

证明

充分性：假设 A 的所有非空子集都含有一个极小元，如果$\cdots\prec a_i\prec\cdots\prec a_1\prec a_0$ 是一条无穷下降链，则 $Q=\{a_i \mid i\in\omega\}$ 是不含极小元的非空集合，这与假设矛盾。因此，关系$\prec$是良基的。

必要性：假设 Q 为 A 的非空子集，用如下方法构造一条链，取 a_0 为 Q 中的任一元素，假设 Q 中已经构造一条链 $a_n\prec\cdots\prec a_0$，此时，或者 Q 中存在 b，使得 $b\prec a_n$，或者 Q 中不存在这样的 b。若 Q 中不存在这样的 b，则停止链的构造；否则，取 $a_{n+1}=b$。由于$\prec$是良基的，因此链$\cdots\prec a_i\prec\cdots\prec a_1\prec a_0$ 不可能是无限的，所以它是有限的，链形如 $a_n\prec\cdots\prec a_0$，满足 $\forall b\prec a_n.\ b\notin Q$。于是取极小元 m 为 a_n。□

31

练习 3.8 设$\prec$为集合 B 上的一个良基关系，试证明

1. 它的传递闭包$\prec^+$也是良基的。
2. 它的自反传递闭包$\prec^*$是偏序的。□

良基归纳法的原理

设$<$是集合A上的良基关系，P是某一性质，则$\forall a \in A. P(a)$当且仅当

$$\forall a \in A. ([\forall b < a. P(b)] \Rightarrow P(a))$$

良基归纳法的原理是：要证明某个对良基集的所有元素性质成立，只要证明如果对任一元素a的所有前趋该性质成立，则对a该性质也成立。

现在我们来证明这个原理，证明的依据是"如果$<$是集合A上的一个良基关系，则A的任何非空子集Q必然含有一个极小元"。显然，如果对A中的所有元素性质$P(a)$成立，则$\forall a \in A. ([\forall b < a. P(b)] \Rightarrow P(a))$。反过来，假设$\forall a \in A. ([\forall b < a. P(b)] \Rightarrow P(a))$成立，但存在某个$a \in A$使得$\neg P(a)$。这样，我们可以构造一个集合$\{a \in A \mid \neg P(a)\}$，它必然含有极小元$m$。于是有$\neg P(m)$为真，但从$\forall b < m. P(b)$可以得到$P(m)$为真，这与假设矛盾。

数学上良基归纳法又称作*诺特归纳法*，以纪念数学家诺特（Emmy Noether）。但是一些计算机科学文献（如[59]）却错误地称之为"结构归纳法"。

例 如果将关系$<$视为非负整数集上的后继关系

$$n < m \text{ 当且仅当 } m = n + 1$$

则此时的良基归纳法就是数学归纳法。 □

例 如果将关系$<$视为非负整数集上的"严格小于"关系$<$，则此时的良基归纳法就是串值归纳法。 □ 32

例 如果将关系$<$视为表达式之间的关系，使得$a < b$当且仅当a是表达式b的直接子表达式，则此时的良基归纳法就是结构归纳法。 □

命题3.7提供了良基归纳法的另一种证明思路。设A是良基集，我们不是直接用良基归纳法去证明A中的所有元素都满足性质P，而是考虑由A中不满足性质P的反例构成的子集F。根据命题3.7，要证明F是$\emptyset$，只要证明F不可能含有极小元。可以用反证法证明，先假设F含有极小元，然后推出矛盾（反证法的例子请参考命题3.12的证明）。到底采用哪种证明方法只是个人的喜好问题，当然，有时候某种方法可能对解决某类问题来说更为简单。

练习3.9 定义串上的一个合适的良基关系，试用上面描述的"找不到反例"的方法证明：对两个不同的符号a和b，不存在满足$au = ub$的串u。请读者将自己的证明方法和良基归纳法以及数学归纳法（见练习3.2）进行比较。 □

良基关系的选择对证明过程十分重要，第10章我们将会介绍构造良基关系的几种常用方法。

下面说明操作语义在证明中的作用，我们以求两个非负整数的最大公约数的欧几里得算法为例证明该算法的终止性。不过这样的问题用指称语义证明可能会更简单一些（练习6.16将证明该算法的正确性）。求两个正整数的最大公约数的欧几里得算法，用**IMP**语言表示为：

$$\text{Euclid} \equiv \textbf{while } \neg(M = N) \textbf{ do}$$
$$\textbf{if } M \leqslant N$$

$$\textbf{then } N := N - M$$
$$\textbf{else } M := M - N$$

定理 3.10 对于所有的状态 σ,有

$$\sigma(M) \geqslant 1 \ \& \ \sigma(N) \geqslant 1 \Rightarrow \exists \sigma'. \langle \text{Euclid}, \sigma \rangle \rightarrow \sigma'$$

证明 设性质 P 为

33

$$P(\sigma) \Longleftrightarrow \exists \sigma'. \langle \text{Euclid}, \sigma \rangle \rightarrow \sigma'$$

我们想证明对 S 中的所有状态 σ 性质 $P(\sigma)$ 都满足,其中

$$S = \{\sigma \in \Sigma \mid \sigma(M) \geqslant 1 \ \& \ \sigma(N) \geqslant 1\}$$

我们首先定义 S 上的关系 $<$:对于 S 中的状态 σ',σ 有

$$\sigma' < \sigma \text{ 当且仅当} (\sigma'(M) \leqslant \sigma(M) \ \& \ \sigma'(N) \leqslant \sigma(N)) \ \& \ (\sigma'(M) \neq \sigma(M) \ \vee \ \sigma'(N) \neq \sigma(N))$$

显然 $<$ 是良基的,因为 M 和 N 的值不会无限减少且始终为正。

令 $\sigma \in S$,设 $\forall \sigma' < \sigma. P(\sigma')$ 为真,记 $\sigma(M) = m, \sigma(N) = n$。

如果 $m = n$,则有 $\langle \neg(M = N), \sigma \rangle \rightarrow \textbf{false}$。根据 **while** 循环命令的规则,当布尔条件为假的时候,我们可以构造一个推导:

$$\dfrac{\dfrac{\vdots}{\langle \neg(M = N), \sigma \rangle \rightarrow \textbf{false}}}{\langle \text{Euclid}, \sigma \rangle \rightarrow \sigma}$$

所以当 $m = n$ 的时候,有 $\langle \text{Euclid}, \sigma \rangle \rightarrow \sigma$,即 $P(\sigma)$ 为真。

如果 $m \neq n$,则有 $\langle \neg(M = N), \sigma \rangle \rightarrow \textbf{true}$。根据命令执行的规则,我们得到

$$\langle \textbf{if } M \leqslant N \textbf{ then } N := N - M \textbf{ else } M := M - N, \sigma \rangle \rightarrow \sigma''$$

其中

$$\sigma'' = \begin{cases} \sigma[n - m/N], & m < n \\ \sigma[m - n/M], & n < m \end{cases}$$

不管哪一种情况都有 $\sigma'' < \sigma$,因此 $P(\sigma'')$ 为真,所以存在某个状态 σ',使得 $\langle \text{Euclid}, \sigma'' \rangle \rightarrow \sigma'$。根据 **while** 循环的其他规则,可以得到 $\langle \text{Euclid}, \sigma \rangle \rightarrow \sigma'$ 的一个推导:

$$\dfrac{\dfrac{\vdots}{\langle \neg(M = N), \sigma \rangle \rightarrow \textbf{true}} \qquad \dfrac{\vdots}{\langle \textbf{if } M \leqslant N \textbf{ then } N := N - M \textbf{ else } M := M - N, \sigma \rangle \rightarrow \sigma''} \qquad \dfrac{\vdots}{\langle \text{Euclid}, \sigma'' \rangle \rightarrow \sigma'}}{\langle \text{Euclid}, \sigma \rangle \rightarrow \sigma'}$$

因此 $P(\sigma)$ 为真。

根据良基归纳法,我们得 $\forall \sigma \in S. P(\sigma)$。 □

良基归纳法是证明程序终止性的最重要的方法。由于程序中存在循环和递归,程序可能不会正常终止,但是如果能够证明执行程序的循环或递归会使良基集的值变小,则程序必会终止。 34

3.4 对推导的归纳

要证明操作语义的性质,单靠结构归纳法通常是不够的。一个常用的方法是通过对推导的结构进行归纳。先给出几个概念的形式化描述。

人们借助于规则来确定可能的推导,规则的实例形如

$$\frac{}{x} \quad \text{或者} \quad \frac{x_1,\cdots,x_n}{x}$$

其中,第一条规则是公理,其前提是空集,结论是 x;第二条规则的一组前提是 $\{x_1,\cdots,x_n\}$,结论是 x。规则指明了如何去构造推导,规则通过这些推导定义集合,规则定义的集合恰好由存在一个推导的那些元素所组成。元素 x 的推导采用树的形式,它或者是一个公理实例

$$\frac{}{x}$$

或者形如

$$\frac{\frac{\vdots}{x_1},\cdots,\frac{\vdots}{x_n}}{x}$$

后者包含了以 x 为结论的规则实例的前提 $x_1,\cdots,x_n$ 的推导,在这样的推导中,称

$$\frac{\vdots}{x_1},\cdots,\frac{\vdots}{x_n}$$

为推导 x 的子推导。

通过用实际的项或值对规则中的元变量进行代入,就得到了规则实例。我们感兴趣的所有规则都是有限的,因为它们的前提都是有限的。因此,所有规则实例都有一个有限的(可能是空集的)前提以及一个结论。我们从一组规则实例的概念开始讨论推导的形式化。

一个规则实例的集合 R 由一组序偶 (X/y) 所组成,其中 X 是一个有限集合,y 是一个元素。称序偶 (X/y) 是以 X 为前提,y 为结论的一个规则实例。

习惯上,把规则实例 (X/y) 看作

$$\frac{}{y} \text{ 如果 } X=\emptyset \text{ ,或者} \frac{x_1,\cdots,x_n}{y} \text{ 如果 } X=\{x_1,\cdots,x_n\}$$

假设 R 是一组规则实例,y 的 R 推导或者是规则实例 $(\emptyset/y)$,或者是序偶 $(\{d_1,\cdots,d_n\}/y)$, 35
其中 $(\{x_1,\cdots,x_n\}/y)$ 是一个规则实例,且 d_1 是 x_1 的 R 推导,$\cdots$,d_n 是 x_n 的 R 推导。我们用 $d \vdash_R y$ 表示 d 是 y 的一个 R 推导。于是

$$(\emptyset/y) \vdash_R y \text{ 如果} (\emptyset/y) \in R\text{,且}$$

$$(\{d_1,\cdots,d_n\}/y) \vdash_R y \text{ 如果} (\{x_1,\cdots,x_n\}/y) \in R \ \&\ d_1 \vdash_R x_1 \ \&\ \cdots \ \&\ d_n \vdash_R x_n$$

如果存在 y 的 R 推导，即对于某个推导 d，有 $d \vdash_R y$，则称 y 是由 R 推导出来的。记号 $\vdash_R y$ 表示 y 是由 R 推导出来的。当我们理解了规则时，可以简单记为 $d \vdash y$ 或者 $\vdash y$。

在操作语义中，前提和结论都是元组。其中，

$$\vdash \langle c,\sigma\rangle \to \sigma'$$

表示 $\langle c,\sigma\rangle \to \sigma'$ 是由命令的操作语义推导出的，通常记为 $\langle c,\sigma\rangle \to \sigma'$。一般把 $\langle c,\sigma\rangle \to \sigma'$ 理解为已经包含了推导的含义，只有在强调推导存在时才把它记作 $\vdash \langle c,\sigma\rangle \to \sigma'$。

令 d,d' 都是推导，称 d' 是 d 的直接子推导（记作 $d' <_1 d$）当且仅当 d 形为 (D/y) 且 $d' \in D$。用 $<$ 表示 $<_1$ 的传递闭包，即 $< = <_1^+$。称 d' 是 d 的真子推导当且仅当 $d' < d$。因为推导是有限的，所以直接子推导 $<_1$ 和真子推导关系都是良基的。这个事实能用来证明命令的执行是确定的。

定理 3.11 令 c 是一条命令，σ_0 是一个状态，对于所有的状态 σ,σ_1，如果 $\langle c,\sigma_0\rangle \to \sigma_1$ 且 $\langle c,\sigma_0\rangle \to \sigma$，则 $\sigma = \sigma_1$。

证明 施良基归纳法于执行命令的推导之间的真子推导关系 $<$ 进行证明。我们要证明的对所有这样的推导 d 成立的性质 P 为：

$$P(d) \Longleftrightarrow \forall c \in \mathbf{Com}, \sigma_0,\sigma,\sigma_1 \in \Sigma.\ d \vdash \langle c,\sigma_0\rangle \to \sigma \ \&\ \langle c,\sigma_0\rangle \to \sigma_1 \Rightarrow \sigma = \sigma_1$$

根据良基归纳原理，只需证明 $\forall d' < d.\ P(d')$ 蕴涵 $P(d)$。

令 d 是命令的操作语义的一个推导，设 $\forall d' < d.\ P(d')$ 成立。假设

$$d \vdash \langle c,\sigma_0\rangle \to \sigma \text{ 且 } \vdash \langle c,\sigma_0\rangle \to \sigma_1$$

36 则对某个 d_1，有 $d_1 \vdash \langle c,\sigma_0\rangle \to \sigma_1$。

现在，我们根据 c 的结构分几种情况来证明 $\sigma = \sigma_1$。

$c \equiv \mathbf{skip}$：此时有

$$d = d_1 = \frac{}{\langle \mathbf{skip},\sigma_0\rangle \to \sigma_0}$$

$c \equiv X := a$：此时两个推导的形式类似：

$$d = \frac{\dfrac{\vdots}{\langle a,\sigma_0\rangle \to m}}{\langle X := a,\sigma_0\rangle \to \sigma_0[m/X]} \qquad d_1 = \frac{\dfrac{\vdots}{\langle a,\sigma_0\rangle \to m_1}}{\langle X := a,\sigma_0\rangle \to \sigma_0[m_1/X]}$$

其中 $\sigma = \sigma_0[m/X]$，$\sigma_1 = \sigma_0[m_1/X]$。因为算术表达式的计算是确定的，即 $m = m_1$，所以 $\sigma = \sigma_1$。

$c \equiv c_0;c_1$：此时，

$$d = \frac{\dfrac{\vdots}{\langle c_0,\sigma_0\rangle \to \sigma'} \quad \dfrac{\vdots}{\langle c_1,\sigma'\rangle \to \sigma}}{\langle c_0;c_1,\sigma_0\rangle \to \sigma} \qquad d_1 = \frac{\dfrac{\vdots}{\langle c_0,\sigma_0\rangle \to \sigma_1'} \quad \dfrac{\vdots}{\langle c_1,\sigma_1'\rangle \to \sigma_1}}{\langle c_0;c_1,\sigma_0\rangle \to \sigma_1}$$

令 d 中 d^0 为子推导

$$\frac{\vdots}{\langle c_0,\sigma_0\rangle \to \sigma'}$$

且 d^1 为子推导

$$\frac{\vdots}{\langle c_1,\sigma'\rangle \to \sigma}$$

则有 $d^0 < d$ 且 $d^1 < d$，于是 $P(d^0)$ 和 $P(d^1)$ 成立。从而，$\sigma' = \sigma_1'$ 且 $\sigma = \sigma_1$（请读者想一想，为什么?）。

$c \equiv$ **if** b **then** c_0 **else** c_1：应用哪一条条件规则取决于布尔表达式 b 的计算结果。由 3.2 节的练习可知，布尔表达式的求值是确定的，所以要么 $\langle b,\sigma_0\rangle \to$ **true**，要么 $\langle b,\sigma_0\rangle \to$ **false**，但两者不能同时成立。

当 $\langle b,\sigma_0\rangle \to$ **true** 时，我们有：

$$d = \frac{\dfrac{\vdots}{\langle b,\sigma_0\rangle \to \mathbf{true}} \quad \dfrac{\vdots}{\langle c_0,\sigma_0\rangle \to \sigma}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1,\sigma_0\rangle \to \sigma} \qquad d_1 = \frac{\dfrac{\vdots}{\langle b,\sigma_0\rangle \to \mathbf{true}} \quad \dfrac{\vdots}{\langle c_0,\sigma_0\rangle \to \sigma_1}}{\langle \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1,\sigma_0\rangle \to \sigma_1}$$

37

令 d' 为 d 中 $\langle c_0,\sigma_0\rangle \to \sigma$ 的子推导，则 $d' < d$，于是 $P(d')$ 成立，从而 $\sigma = \sigma_1$。当 $\langle b,\sigma_0\rangle \to$ **false** 时，证明与此类似。

$c \equiv$ **while** b **do** c：同样，应用 **while** 循环命令的规则也取决于 b 是如何计算的。要么 $\langle b,\sigma_0\rangle \to$ **true**，要么 $\langle b,\sigma_0\rangle \to$ **false**，但两者不能同时成立。

当 $\langle b,\sigma_0\rangle \to$ **false** 时，我们有：

$$d = \frac{\dfrac{\vdots}{\langle b,\sigma_0\rangle \to \mathbf{false}}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c,\sigma_0\rangle \to \sigma_0} \qquad d_1 = \frac{\dfrac{\vdots}{\langle b,\sigma_0\rangle \to \mathbf{false}}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c,\sigma_0\rangle \to \sigma_0}$$

显然 $\sigma = \sigma_0 = \sigma_1$。

当 $\langle b,\sigma_0\rangle \to$ **true** 时，我们有：

$$d = \frac{\dfrac{\vdots}{\langle b,\sigma_0\rangle \to \mathbf{true}} \quad \dfrac{\vdots}{\langle c,\sigma_0\rangle \to \sigma'} \quad \dfrac{\vdots}{\langle \mathbf{while}\ b\ \mathbf{do}\ c,\sigma'\rangle \to \sigma}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c,\sigma_0\rangle \to \sigma}$$

$$d_1 = \frac{\dfrac{\vdots}{\langle b,\sigma_0\rangle \to \mathbf{true}} \quad \dfrac{\vdots}{\langle c,\sigma_0\rangle \to \sigma_1'} \quad \dfrac{\vdots}{\langle \mathbf{while}\ b\ \mathbf{do}\ c,\sigma_1'\rangle \to \sigma_1}}{\langle \mathbf{while}\ b\ \mathbf{do}\ c,\sigma_0\rangle \to \sigma_1}$$

令 d' 为 d 中 $\langle c,\sigma_0\rangle \to \sigma'$ 的子推导，d'' 为 d 中 $\langle$ **while** b **do** $c,\sigma'\rangle \to \sigma$ 的子推导，于是 $d' < d$ 且 $d'' < d$，故 $P(d')$，$P(d'')$ 成立。从而 $\sigma' = \sigma_1'$，继而得到 $\sigma = \sigma_1$。

对 c 的所有结构进行分析后，我们证明了 $d \vdash \langle c,\sigma_0\rangle \to \sigma$ 且 $\langle c,\sigma_0\rangle \to \sigma_1$ 蕴涵 $\sigma = \sigma_1$。

根据良基归纳法原理，我们得到了对执行命令的所有推导 d 性质 $P(d)$ 都满足。这等价于

$$\forall c \in \mathbf{Com}, \sigma_0, \sigma, \sigma_1 \in \Sigma. \langle c, \sigma_0 \rangle \to \sigma \ \& \ \langle c, \sigma_0 \rangle \to \sigma_1 \Rightarrow \sigma = \sigma_1$$

于是定理得证。 □

我们注意到，命题 3.7 提供了用良基归纳法进行证明的另一种方法。对推导的归纳，是用于证明对所有的推导某个性质都满足的良基归纳法的特殊情况。按照命题 3.7 提供的方法，证明中先假设存在一个不满足该性质的最小推导，然后再推得矛盾。这种证明方法说明如下。

[38]

命题 3.12 对于所有的状态 σ, σ'，有

$$\langle \textbf{while true do skip}, \sigma \rangle \not\to \sigma'$$

证明 记 $w \equiv$ **while true do skip**。对状态 σ, σ' 假设 $\langle w, \sigma \rangle \to \sigma'$，则存在一个极小推导 d，使得 $\exists \sigma, \sigma' \in \Sigma. d \vdash \langle w, \sigma \rangle \to \sigma'$。$d$ 的最终规则只能有一条，使 d 形如：

$$d = \frac{\dfrac{\vdots}{\langle \textbf{true}, \sigma \rangle \to \textbf{true}} \quad \dfrac{\vdots}{\langle c, \sigma \rangle \to \sigma''} \quad \dfrac{\vdots}{\langle \textbf{while true do } c, \sigma'' \rangle \to \sigma'}}{\langle \textbf{while true do } c, \sigma \rangle \to \sigma'}$$

但这时 d 就含有一个真子推导 $d' \vdash \langle w, \sigma'' \rangle \to \sigma'$[⊖]，与 d 的极小性矛盾。 □

3.5 归纳定义

人们经常使用像结构归纳法这样的技术去定义给定集合上的运算。整数和算术表达式有着共同之处，它们都是按照相似的方法构造出来的。整数或者是 0，或者是某一整数的后继[⊖]，而算术表达式或者是原子表达式，或者是对两个算术表达式进行和或积等运算而得到的表达式。正是基于它们这种构造上的特点，我们才能对整数和表达式给出归纳定义。例如，要定义表达式的长度，通常可以用子表达式的长度来定义它。对算术表达式的长度，可以定义

$$\begin{aligned} &\text{length}(n) = \text{length}(X) = 1 \\ &\text{length}(a_0 + a_1) = 1 + \text{length}(a_0) + \text{length}(a_1) \\ &\cdots \end{aligned}$$

我们再定义集合 $\text{loc}_L(c)$，它的元素为出现在赋值命令左边的存储单元。对命令 c，用结构归纳法来定义 $\text{loc}_L(c)$：

$$\begin{aligned} \text{loc}_L(\textbf{skip}) &= \emptyset \qquad \text{loc}_L(X := a) = \{X\}, \\ \text{loc}_L(c_0; c_1) &= \text{loc}_L(c_0) \cup \text{loc}_L(c_1) \qquad \text{loc}_L(\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1) = \text{loc}_L(c_0) \cup \text{loc}_L(c_1), \\ \text{loc}_L(\textbf{while } b \textbf{ do } c) &= \text{loc}_L(c) \end{aligned}$$

类似地，可以用数学归纳法来定义自然数集合上的运算，以及定义由规则确定的集合上的运算。在命题 3.7（良基集的任一非空子集都有一个极小元）的证明中隐含使用了对自然数的归纳定义，用于构造一条含极小元的非空集合链。

[39]

⊖ 原文为 $d' \vdash \langle w, \sigma \rangle \to \sigma'$，有误。——译者注

⊖ 要完整地定义整数，还需要其他的构造算子（如前趋函数）。——译者注

不管是结构归纳定义还是数学归纳定义，都是良基归纳定义的特殊情况。良基归纳又称为*良基递归*，因为它采取的是递归定义的方式。例如，算术表达式的长度可以用以下方式来定义：

$$\text{length}(a) = \begin{cases} 1 & a \equiv n(\text{一个数}) \\ \text{length}(a_0) + \text{length}(a_1) & a \equiv (a_0 + a_1) \\ \vdots & \end{cases}$$

通过计算 a_0 和 a_1 的长度来求 $(a_0 + a_1)$ 的长度，因此这个函数是由自身递归定义的。待求项的值通过求更小项的值递归计算。同样，基于任意良基集的函数都可以用递归方法来定义。但是递归的基本原理建立在相对复杂的集合构造方法上就比较难理解。因此，关于递归的详细讨论在 10.4 节进行。（虽然那里的内容并不会影响现在的讨论，但如果读者感兴趣的话可以翻到后面先行阅读，一般不会有理解上的困难。）

练习 3.13 请给出 $\text{loc}(a)$，$\text{loc}(b)$ 和 $\text{loc}_R(c)$ 的结构归纳定义，其中 $\text{loc}(a)$ 是算术表达式 a 的存储单元集合，$\text{loc}(b)$ 是布尔表达式 b 的存储单元集合，$\text{loc}_R(c)$ 是赋值命令 c 右边的存储单元集合。 □

3.6 进一步阅读资料

本章讨论了数理逻辑中的一些常见的基本方法和思想。操作语义与自然演绎紧密相关，因而与证明论也就有了方法上的共同之处，正如在文献[84]中所指出的——推导实际上就是一种简单的证明。如果读者想更深入地理解计算机科学的证明理论，可以参考文献[51,40]，这些文献中介绍了证明和推导的更多记号。关于良基归纳法的详细讨论和应用，可以参考文献[59]和[21]（那里“良基归纳法”被称为“结构归纳法”）以及[58]和[73]，或者本书的第 10 章。 40

第4章 归纳定义

本章以语法表示和操作语义为例，介绍归纳定义集合的理论。由规则归纳定义的集合是对该规则封闭的最小集。集合的构造始终体现着规则归纳法的原理。我们着重讨论论证 **IMP** 操作语义的证明规则。

4.1 规则归纳法

我们把算术表达式的语法集合 **Aexp** 定义为由算术表达式的构造规则所确定的集合。上一章已经介绍了与之相应的归纳原理——算术表达式的结构归纳法原理。通过用一组规则定义表达式的求值关系和命令的执行关系，我们已经定义了 **while** 程序的操作语义。根据规则，表达式的求值关系或者命令的执行关系都要分解为更小的语法单元来处理。例如，根据 2.2 节中的规则，算术表达式的求值关系定义为三元关系，即 $\mathbf{Aexp} \times \Sigma \times \mathbf{N}$ 上的三元组 (a,σ,n) 构成的集合，使得 $\langle a,\sigma\rangle \to n$。作为结构归纳法的特例，我们把这种归纳原理称为规则归纳法。

我们关心的是怎样由规则来定义集合。抽象地看，规则实例形如 $(\emptyset/x)$ 或 $(\{x_1,\cdots,x_n\}/x)$。给定一组规则实例 R，我们把 R 定义的集合记为 I_R，I_R 恰好由存在推导的那些元素 x 组成，即

$$I_R = \{x \mid \Vdash_R x\}$$

要证明对规则所定义的集合的所有元素某个性质为真，常常用规则归纳原理证明。它的基本思想是：在一个推导中，如果一个性质在从所有规则实例的前提移动到结论的过程中保持为真，那么该推导的结论也具有该性质。因此，对由这些规则定义的集合中的所有元素该性质也为真。

规则归纳法的一般原理

设 I_R 是规则实例 R 定义的集合，P 是某个性质，则 $\forall x \in I_R.\ P(x)$ 当且仅当对于 R 中所有的规则实例 (X/y)，其中 $X \subseteq I_R$，有

$$(\forall x \in X.\ P(x)) \Rightarrow P(y)$$

注意到对 $X = \emptyset$ 的形如 (X/y) 的规则实例，最后的条件等价于 $P(y)$。显然，因为空集 $\emptyset$ 不含任何元素，所以 $\forall x \in X.\ x \in I_R \ \&\ P(x)$ 自动为真。规则归纳法描述如下：对规则实例 R，我们有 $\forall y \in I_R.\ P(y)$ 当且仅当对所有的公理实例 [41]

$$\frac{}{x}$$

$P(x)$ 为真，并且对所有的规则实例

$$\frac{x_1,\cdots,x_n}{x}$$

如果对所有的前提 $x_k(1\leqslant k\leqslant n)$，$x_k\in I_R$ 且 $P(x_k)$ 为真，则结论 x 的 $P(x)$ 也为真。

规则归纳法的原理相当直观，它和数学中常用的方法虽然形式不同，但实质上是一样的（该原理也导致规则归纳有效性的证明）。我们称集合 Q 对规则实例 R 是封闭的（简称为 R 封闭）当且仅当对于所有的规则实例 (X/y)，有

$$X\subseteq Q\Rightarrow y\in Q$$

换句话说，如果每当任一规则实例的前提属于该集合时，它的结论也属于该集合，则一个集合对规则实例是封闭的。特别地，一个 R 封闭集合必须包含所有的公理实例。在下述意义下，I_R 是对 R 封闭的最小集。

命题 4.1 对应于规则实例 R，

(i) I_R 是 R 封闭的，且

(ii) 如果 Q 是对 R 封闭的集合，则 $I_R\subseteq Q$。

证明

(i) 容易证得 I_R 是 R 封闭的。设 (X/y) 是 R 中的一个规则实例，且 $X\subseteq I_R$。根据 I_R 的定义，X 的每一元素都存在推导。如果集合 X 非空，那么这些推导结合规则实例 (X/y)，来提供 y 的一个推导；否则，如果集合 X 是空集，那么 $(\emptyset/y)$ 直接就提供 y 的一个推导。不管是哪种情况，我们都能得到 y 的一个推导，因此 y 也必定在 I_R 中。因此 I_R 对 R 封闭。

(ii) 设 Q 是对 R 封闭的集合，我们要证明 $I_R\subseteq Q$。I_R 中任一元素都是某个推导的结论，而任一推导都是由规则实例 (X/y) 构造出来的。如果前提 X 属于集合 Q，那么结论 y 也属于集合 Q（特别地，任一公理的结论属于 Q）。因此，对任一推导，我们可以从公理开始，逐步向下证明其结论属于 Q。更形式化地，施归纳于真子推导关系 $<$ 来证明对所有的 R 推导 d，有

42

$$\forall y\in I_R.\ d\vdash_R y\Rightarrow y\in Q$$

所以 $I_R\subseteq Q$。 □

练习 4.2 试给出上面证明中提到的对推导的归纳证明。 □

假设我们要证明对规则 R 定义的集合 I_R 的所有元素性质 P 为真，那么命题 4.1 的两个条件提供了一个证明方法。定义集合

$$Q=\{x\in I_R\mid P(x)\}$$

对 I_R 的所有元素性质 P 为真当且仅当 $I_R\subseteq Q$。根据条件(ii)，要证明 $I_R\subseteq Q$，只要证明 Q 是 R 封闭的。于是需要证明，对于所有的规则实例 (X/y)，有

$$(\forall x\in X.\ x\in I_R\ \&\ P(x))\Rightarrow P(y)$$

而这正是用规则归纳法证明对 I_R 中的所有元素性质 P 都成立时所需要的。对于证明对 I_R 的所有元素性质 P 成立，上述命题不仅是充分的而且还是必要的。假设对所有的 $x\in I_R$，$P(x)$ 成立，令 (X/y) 是满足

$$\forall x\in X.\ x\in I_R\ \&\ P(x)$$

的规则实例，由条件(i)，I_R 是 R 封闭的，我们有 $y\in I_R$，因此 $P(y)$ 为真。用这种方法，我们从

条件(i)和(ii)推得了规则归纳法的原理,其中 I_R 是最小 R 封闭集。

练习 4.3 对于规则实例 R,试证明集合 $\cap\{Q \mid Q$ 是 R 封闭的$\}$ 是 R 封闭的。该集合是什么? □

练习 4.4 设规则由($\emptyset/0$)和($\{n\}/(n+1)$)组成,其中 n 是自然数。试给出该组规则所定义的集合和相应的规则归纳法。 □

我们用与定义操作语义时使用的相同风格来表示规则。当用规则定义语法集合时,BNF 是一种传统方法,它可以有不同的表示方式。例如,要表述"如果 a_0 和 a_1 都是合式算术表达式,则 a_0+a_1 也是算术表达式",用 BNF 可以表示为

$$a ::= \cdots \mid a_0 + a_1 \mid \cdots$$

43

也可以表示为

$$\frac{a_0 : \mathbf{Aexp} \quad a_1 : \mathbf{Aexp}}{a_0 + a_1 : \mathbf{Aexp}}$$

后一种语法表示方式正为越来越多的人采用。

练习 4.5 试给出 **Aexp** 的构造规则的规则归纳法和 **Bexp** 的构造规则的规则归纳法。(注意! 参考下一节。) □

4.2 特殊的规则归纳法

观察布尔表达式和命令的语法集合,你会很明显地发现有时候定义语法集合的规则中包含了其他语法集合的元素。例如,命令的构造规则说明了如何由算术表达式、布尔表达式和其他的命令来构成命令。为了一致起见,我们把构造规则

$$c ::= \cdots \mid X := a \mid \cdots \mid \textbf{ if } b \textbf{ then } c_0 \textbf{ else } c_1 \mid \cdots$$

改写成

$$\frac{X : \mathbf{Loc} \quad a : \mathbf{Aexp}}{X := a : \mathbf{Com}} \quad 和 \quad \frac{b : \mathbf{Bexp} \quad c_0 : \mathbf{Com} \quad c_1 : \mathbf{Com}}{\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1 : \mathbf{Com}}$$

规则归纳法的工作就是证明规则保持了某个性质。这意味着如果我们要使用规则归纳法证明所有的命令具有某个性质,则我们必须确保这个性质的证明考虑了所有的算术表达式和布尔表达式。因而,对规则归纳法的原理进行实例化不会得到命令上的结构归纳法,而是一个更难处理的、同时结合算术表达式、布尔表达式和命令三者之上的结构归纳法的证明原理。如果要建立规则定义的集合的子集的某些性质,我们需要对规则归纳法原理做一些修改。

规则归纳法的特殊原理

设 I_R 是规则实例 R 定义的集合,$A \subseteq I_R$,Q 是某个性质,则 $\forall a \in A.\ Q(a)$ 当且仅当对于 R 中所有的规则实例(X/y),其中 $X \subseteq I_R$,$y \in A$,有

$$(\forall x \in X \cap A.\ Q(x)) \Rightarrow Q(y)$$

44

规则归纳法的特殊原理和规则归纳法的一般原理没有什么本质不同。令 R 是一组规则实例，A 是 R 定义的集合 I_R 的子集，假设 $Q(x)$ 是我们要证明的对 A 中所有元素为真的性质。定义相应的一个性质 $P(x)$

$$P(x) \Longleftrightarrow (x \in A \Rightarrow Q(x))$$

要证明对所有的 $a \in A$ 性质 $Q(a)$ 为真，等价于证明对所有的 $x \in I_R$ 性质 $P(x)$ 为真。按照规则归纳法的一般原理，后者等价于

$$\forall (X/y) \in R.\, X \subseteq I_R \,\&\, (\forall x \in X.\, (x \in A \Rightarrow Q(x))) \Rightarrow (y \in A \Rightarrow Q(y))$$

而这在逻辑上等价于

$$\forall (X/y) \in R.\, (X \subseteq I_R \,\&\, y \in A \,\&\, (\forall x \in X.\, (x \in A \Rightarrow Q(x)))) \Rightarrow Q(y)$$

这等价于规则归纳法的特殊原理所需要的条件。

练习 4.6 请说明命令和布尔表达式的结构归纳法如何体现了规则归纳法的特殊原理。

□

特殊来自一般，使用特殊规则归纳法能够证明的命题，一般规则归纳法也一定能证明。但是，在实践中往往采取特殊的归纳原理，以大量减少证明中使用的规则的数量。这一点对操作语义的规则归纳法十分重要。

4.3 操作语义的证明规则

规则归纳法无疑是证明由规则表示的操作语义性质的有用工具，因为规则所定义的集合是元组的集合，所以规则归纳和其他证明方法在形式上有所不同。本节介绍规则归纳法的特殊情况，稍后我们在论证 **IMP** 程序的操作行为时要用到这些知识。

4.3.1 算术表达式的规则归纳法

算术表达式求值的规则归纳法原理来自算术表达式操作语义的规则。它是规则归纳法的例子，对所有的求值关系 $\langle a,\sigma\rangle \to n$，性质 $P(a,\sigma,n)$ 为真当且仅当构造该求值关系的规则保持性质 P。

45

$$
\begin{aligned}
&\forall a \in \mathbf{Aexp}, \sigma \in \Sigma, n \in \mathbf{N}.\, \langle a,\sigma\rangle \to n \Rightarrow P(a,\sigma,n)\\
&\text{当且仅当}\\
&[\ \forall n \in \mathbf{N}, \sigma \in \Sigma.\, P(n,\sigma,n)\\
&\&\\
&\ \forall X \in \mathbf{Loc}, \sigma \in \Sigma.\, P(X,\sigma,\sigma(X))\\
&\&\\
&\ \forall a_0, a_1 \in \mathbf{Aexp}, \sigma \in \Sigma, n_0, n_1 \in \mathbf{N}.\\
&\langle a_0,\sigma\rangle \to n_0 \,\&\, P(a_0,\sigma,n_0) \,\&\, \langle a_1,\sigma\rangle \to n_1 \,\&\, P(a_1,\sigma,n_1)\\
&\Rightarrow P(a_0 + a_1, \sigma, n_0 + n_1)\\
&\&\\
&\ \forall a_0, a_1 \in \mathbf{Aexp}, \sigma \in \Sigma, n_0, n_1 \in \mathbf{N}.
\end{aligned}
$$

$$\langle a_0,\sigma\rangle \to n_0 \ \&\ P(a_0,\sigma,n_0)\ \&\ \langle a_1,\sigma\rangle \to n_1 \ \&\ P(a_1,\sigma,n_1)$$
$$\Rightarrow P(a_0 - a_1,\sigma,n_0 - n_1)$$
$$\&$$
$$\forall a_0,a_1 \in \mathbf{Aexp},\sigma \in \Sigma,n_0,n_1 \in \mathbf{N}.$$
$$\langle a_0,\sigma\rangle \to n_0 \ \&\ P(a_0,\sigma,n_0)\ \&\ \langle a_1,\sigma\rangle \to n_1 \ \&\ P(a_1,\sigma,n_1)$$
$$\Rightarrow P(a_0 \times a_1,\sigma,n_0 \times n_1)]$$

请读者将这个规则归纳法的特殊原理和规则归纳法的一般原理进行一下比较，并思考求值规则是如何将所有可能的规则实例都包括在内的。

4.3.2 布尔表达式的规则归纳法

布尔表达式的求值规则与算术表达式的求值规则有关。这些求值规则一起定义了集合

$$(\mathbf{Aexp} \times \Sigma \times \mathbf{N}) \cup (\mathbf{Bexp} \times \Sigma \times \mathbf{T})$$

的一个子集。论证布尔表达式的操作语义来自对子集 $\mathbf{Bexp} \times \Sigma \times \mathbf{T}$ 上性质 $P(b,\sigma,t)$ 的规则归纳法的特殊原理。 46

$$\forall b \in \mathbf{Bexp},\sigma \in \Sigma,t \in \mathbf{T}.\ \langle b,\sigma\rangle \to t \Rightarrow P(b,\sigma,t)$$

当且仅当

$$[\ \forall \sigma \in \Sigma.\ P(\mathbf{false},\sigma,\mathbf{false})\ \&\ \forall \sigma \in \Sigma.\ P(\mathbf{true},\sigma,\mathbf{true})$$
$$\&$$
$$\forall a_0,a_1 \in \mathbf{Aexp},\sigma \in \Sigma,m,n \in \mathbf{N}.$$
$$\langle a_0,\sigma\rangle \to m \ \&\ \langle a_1,\sigma\rangle \to n \ \&\ m = n \Rightarrow P(a_0 = a_1,\sigma,\mathbf{true})$$
$$\&$$
$$\forall a_0,a_1 \in \mathbf{Aexp},\sigma \in \Sigma,m,n \in \mathbf{N}.$$
$$\langle a_0,\sigma\rangle \to m \ \&\ \langle a_1,\sigma\rangle \to n \ \&\ m \neq n \Rightarrow P(a_0 = a_1,\sigma,\mathbf{false})$$
$$\&$$
$$\forall a_0,a_1 \in \mathbf{Aexp},\sigma \in \Sigma,m,n \in \mathbf{N}.$$
$$\langle a_0,\sigma\rangle \to m \ \&\ \langle a_1,\sigma\rangle \to n \ \&\ m \leqslant n \Rightarrow P(a_0 \leqslant a_1,\sigma,\mathbf{true})$$
$$\&$$
$$\forall a_0,a_1 \in \mathbf{Aexp},\sigma \in \Sigma,m,n \in \mathbf{N}.$$
$$\langle a_0,\sigma\rangle \to m \ \&\ \langle a_1,\sigma\rangle \to n \ \&\ m \not\leqslant n \Rightarrow P(a_0 \leqslant a_1,\sigma,\mathbf{false})$$
$$\&$$
$$\forall b \in \mathbf{Bexp},\sigma \in \Sigma,t \in \mathbf{T}.$$
$$\langle b,\sigma\rangle \to t \ \&\ P(b,\sigma,t) \Rightarrow P(\neg b,\sigma,\neg t)$$
$$\&$$
$$\forall b_0,b_1 \in \mathbf{Bexp},\sigma \in \Sigma,t_0,t_1 \in \mathbf{T}.$$
$$\langle b_0,\sigma\rangle \to t_0 \ \&\ P(b_0,\sigma,t_0)\ \&\ \langle b_1,\sigma\rangle \to t_1 \ \&\ P(b_1,\sigma,t_1) \Rightarrow P(b_0 \wedge b_1,\sigma,t_0 \wedge t_1)$$
$$\&$$
$$\forall b_0,b_1 \in \mathbf{Bexp},\sigma \in \Sigma,t_0,t_1 \in \mathbf{T}.$$

$$\langle b_0,\sigma\rangle \to t_0 \ \&\ P(b_0,\sigma,t_0)\ \&\ \langle b_1,\sigma\rangle \to t_1\ \&\ P(b_1,\sigma,t_1) \Rightarrow P(b_0 \vee b_1,\sigma,t_0 \vee t_1)]$$

4.3.3 命令的规则归纳法

用于论证命令的操作语义的规则归纳法原理是规则归纳法特殊原理的一个例子。命令的执行的规则与算术表达式以及布尔表达式的求值有关。这三种不同的语法集合的操作语义的规则一起定义了集合

47

$$(\mathbf{Aexp} \times \Sigma \times \mathbf{N}) \cup (\mathbf{Bexp} \times \Sigma \times \mathbf{T}) \cup (\mathbf{Com} \times \Sigma \times \Sigma)$$

的一个子集。我们使用了子集 $\mathbf{Com} \times \Sigma \times \Sigma$ 上性质 $P(c,\sigma,\sigma')$ 的规则归纳法的特殊原理。(请读者自己写出来并和下面的结果比较。)

$$\forall c \in \mathbf{Com},\sigma,\sigma' \in \Sigma.\ \langle c,\sigma\rangle \to \sigma' \Rightarrow P(c,\sigma,\sigma')$$

当且仅当

$$[\ \forall \sigma \in \Sigma.\ P(\mathbf{skip},\sigma,\sigma)$$

$$\&$$

$$\forall X \in \mathbf{Loc},a \in \mathbf{Aexp},\sigma \in \Sigma,m \in \mathbf{N}.\ \langle a,\sigma\rangle \to m \Rightarrow P(X := a,\sigma,\sigma[m/X])$$

$$\&$$

$$\forall c_0,c_1 \in \mathbf{Com},\sigma,\sigma',\sigma'' \in \Sigma.$$

$$\langle c_0,\sigma\rangle \to \sigma''\ \&\ P(c_0,\sigma,\sigma'')\ \&\ \langle c_1,\sigma''\rangle \to \sigma'\ \&\ P(c_1,\sigma'',\sigma') \Rightarrow P(c_0;c_1,\sigma,\sigma')$$

$$\&$$

$$\forall c_0,c_1 \in \mathbf{Com},b \in \mathbf{Bexp},\sigma,\sigma' \in \Sigma.$$

$$\langle b,\sigma\rangle \to \mathbf{true}\ \&\ \langle c_0,\sigma\rangle \to \sigma'\ \&\ P(c_0,\sigma,\sigma') \Rightarrow P(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1,\sigma,\sigma')$$

$$\&$$

$$\forall c_0,c_1 \in \mathbf{Com},b \in \mathbf{Bexp},\sigma,\sigma' \in \Sigma.$$

$$\langle b,\sigma\rangle \to \mathbf{false}\ \&\ \langle c_1,\sigma\rangle \to \sigma'\ \&\ P(c_1,\sigma,\sigma') \Rightarrow P(\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1,\sigma,\sigma')$$

$$\&$$

$$\forall c \in \mathbf{Com},b \in \mathbf{Bexp},\sigma \in \Sigma.$$

$$\langle b,\sigma\rangle \to \mathbf{false} \Rightarrow P(\mathbf{while}\ b\ \mathbf{do}\ c,\sigma,\sigma)$$

$$\&$$

$$\forall c \in \mathbf{Com},b \in \mathbf{Bexp},\sigma,\sigma',\sigma'' \in \Sigma.$$

$$\langle b,\sigma\rangle \to \mathbf{true}\ \&\ \langle c,\sigma\rangle \to \sigma''\ \&\ P(c,\sigma,\sigma'')\ \&$$

$$\langle \mathbf{while}\ b\ \mathbf{do}\ c,\sigma''\rangle \to \sigma'\ \&\ P(\mathbf{while}\ b\ \mathbf{do}\ c,\sigma'',\sigma')$$

$$\Rightarrow P(\mathbf{while}\ b\ \mathbf{do}\ c,\sigma,\sigma')]$$

作为一个例子,我们用规则归纳法去证明一个显而易见的事实:如果存储单元 Y 不在命令 c 的赋值左部出现,则执行 c 不会影响 Y 的值。请读者回忆命令 c 的存储单元 $\mathrm{loc}_L(c)$ 的定义(见3.5节)。

48

命题 4.7 设 $Y \in \mathbf{Loc}$,对于所有的命令 c 和状态 σ,σ',有

$$Y \notin \mathrm{loc}_L(c)\ \&\ \langle c,\sigma\rangle \to \sigma' \Rightarrow \sigma(Y) = \sigma'(Y)$$

证明 设 P 为性质

$$P(c,\sigma,\sigma') \Longleftrightarrow (Y \notin \mathrm{loc}_L(c) \Rightarrow \sigma(Y) = \sigma'(Y))$$

施规则归纳于命令去证明

$$\forall c \in \mathbf{Com}, \sigma, \sigma' \in \Sigma. \langle c,\sigma\rangle \to \sigma' \Rightarrow P(c,\sigma,\sigma')$$

对于任一 $\sigma \in \Sigma$,显然 $P(\mathbf{skip},\sigma,\sigma)$ 成立。

令 $X \in \mathbf{Loc}, a \in \mathbf{Aexp}, \sigma \in \Sigma, m \in \mathbf{N}$,假定 $\langle a,\sigma\rangle \to m$,如果 $Y \notin \mathrm{loc}_L(X := a)$ 则 $Y \neq X$,所以 $\sigma(Y) = \sigma[m/X](Y)$,于是 $P(X := a,\sigma,\sigma[m/X])$ 成立。

令 $c_0, c_1 \in \mathbf{Com}, \sigma, \sigma' \in \Sigma$,我们假设

$$\langle c_0,\sigma\rangle \to \sigma'' \ \&\ P(c_0,\sigma,\sigma'') \ \&\ \langle c_1,\sigma''\rangle \to \sigma' \ \&\ P(c_1,\sigma'',\sigma')$$

即假设

$$\begin{aligned}&\langle c_0,\sigma\rangle \to \sigma'' \ \&\ (Y \notin \mathrm{loc}_L(c_0) \Rightarrow \sigma(Y) = \sigma''(Y)) \ \&\\ &\langle c_1,\sigma''\rangle \to \sigma' \ \&\ (Y \notin \mathrm{loc}_L(c_1) \Rightarrow \sigma''(Y) = \sigma'(Y))\end{aligned}$$

设 $Y \notin \mathrm{loc}_L(c_0;c_1)$,则由 $\mathrm{loc}_L(c_0;c_1) = \mathrm{loc}_L(c_0) \cup \mathrm{loc}_L(c_1)$,得 $Y \notin \mathrm{loc}_L(c_0)$ 且 $Y \notin \mathrm{loc}_L(c_1)$。于是由假设得 $\sigma(Y) = \sigma''(Y) = \sigma'(Y)$,从而 $P(c_0;c_1,\sigma,\sigma')$ 成立。

下面我们只讨论另外一种规则实例。

令 $c \in \mathbf{Com}, b \in \mathbf{Bexp}, \sigma, \sigma', \sigma'' \in \Sigma$,设 $w \equiv$ **while** b **do** c,假设

$$\begin{aligned}\langle b,\sigma\rangle \to \mathbf{true} \ \&\ \langle c,\sigma\rangle \to \sigma'' \ \&\ P(c,\sigma,\sigma'') \ \&\\ \langle w,\sigma''\rangle \to \sigma' \ \&\ P(w,\sigma'',\sigma')\end{aligned}$$

即假设

$$\begin{aligned}\langle b,\sigma\rangle \to \mathbf{true} \ \&\ \langle c,\sigma\rangle \to \sigma'' \ \&\ (Y \notin \mathrm{loc}_L(c) \Rightarrow \sigma(Y) = \sigma''(Y)) \ \&\\ \langle w,\sigma''\rangle \to \sigma' \ \&\ (Y \notin \mathrm{loc}_L(w) \Rightarrow \sigma''(Y) = \sigma'(Y))\end{aligned}$$

设 $Y \notin \mathrm{loc}_L(w)$,由假设得 $\sigma''(Y) = \sigma'(Y)$。又因为 $\mathrm{loc}_L(w) = \mathrm{loc}_L(c)$,于是 $Y \notin \mathrm{loc}_L(c)$,所以由假设得 $\sigma(Y) = \sigma''(Y)$。这样就有 $\sigma(Y) = \sigma'(Y)$,于是 $P(w,\sigma,\sigma')$ 成立。

其余的情况同理可证,留作练习。 □ 49

本书后面几章会有更多的规则归纳法证明的例子,大多比较简单。下面的几道练习则有一定的难度。前两道练习说明规则归纳法的应用有时还是比较棘手的。

练习 4.8 令 $w \equiv$ **while true do skip**,试用特殊规则归纳法证明:

$$\forall \sigma, \sigma'. \langle w,\sigma\rangle \not\to \sigma'$$

(提示:对集合

$$\{(w,\sigma,\sigma') \mid \sigma,\sigma' \in \Sigma\}$$

应用特殊的规则归纳法,并令性质 $P(w,\sigma,\sigma')$ 恒为假。

将本题的证明与 3.4 节中命题 3.12 的证明相比较，读者会发现有时候规则归纳法并不比用推导进行证明直观。） □

规则归纳法可以代替对推导的归纳，但不是说它完全取代了后者的地位。一味地使用规则归纳法有时反而使证明变得更冗长更复杂，比如练习 4.9。

练习 4.9 简化的算术表达式语法为：

$$a ::= n \mid X \mid a_0 + a_1$$

简化的算术表达式的求值规则如下：

$$\langle n,\sigma\rangle \to n$$

$$\langle X,\sigma\rangle \to \sigma(X)$$

$$\frac{\langle a_0,\sigma\rangle \to n_0 \quad \langle a_1,\sigma\rangle \to n_1}{\langle a_0 + a_1,\sigma\rangle \to n}$$

其中 n 是 n_0 与 n_1 之和

根据推导的惟一形式我们很容易得出 $\langle n,\sigma\rangle \to m$ 蕴涵 $m \equiv n$，试用特殊的规则归纳法去证明它。对操作语义用规则归纳法（而不是用对推导的归纳）证明表达式的求值是确定的。

50

（提示：对于后一个问题，考虑把

$$P(a,\sigma,m) \Longleftrightarrow_{def} \forall m' \in \mathbf{N}.\ \langle a,\sigma\rangle \to m' \Rightarrow m = m'$$

作为归纳假设，然后再用特殊的规则归纳法证明。）

3.2 节中对命题 3.3 的证明采用了结构归纳法，并且考虑了所有可能的推导形式。请问规则归纳法和命题 3.3 的证明有何异同？ □

下面这个练习要求证明两个操作语义是等价的。

练习 4.10 这两个操作语义分别是第 2 章中关系 $\langle c,\sigma\rangle \to \sigma'$ 的操作语义以及 2.6 节中单步执行关系 $\langle c,\sigma\rangle \to_1 \langle c',\sigma'\rangle$ 的语义。为了简单起见，表达式的求值规则和第 2 章中所述的规则相同。例如，对于两个命令的顺序执行有如下规则：

$$\frac{\langle c_0,\sigma\rangle \to_1 \langle c_0',\sigma'\rangle}{\langle c_0;c_1,\sigma\rangle \to_1 \langle c_0';c_1,\sigma'\rangle} \qquad \frac{\langle c_0,\sigma\rangle \to_1 \sigma'}{\langle c_0;c_1,\sigma\rangle \to_1 \langle c_1,\sigma'\rangle}$$

先证明引理：对于所有的命令 c_0, c_1 和所有的状态 σ,σ'，有

$$\langle c_0;c_1,\sigma\rangle \to_1^* \sigma' \text{ 当且仅当 } \exists \sigma''.\ \langle c_0,\sigma\rangle \to_1^* \sigma'' \ \&\ \langle c_1,\sigma''\rangle \to_1^* \sigma'$$

引理的证明可以分为两步，第一步用数学归纳法施归纳于计算的长度 n：

$$\forall \sigma,\sigma'.\ [\langle c_0;c_1,\sigma\rangle \to_1^n \sigma' \Rightarrow \exists \sigma''.\ \langle c_0,\sigma\rangle \to_1^* \sigma'' \ \&\ \langle c_1,\sigma''\rangle \to_1^* \sigma']$$

第二步用数学归纳法施归纳于状态 σ 下执行 c_0 的长度 n：

$$\forall \sigma,\sigma',\sigma''.\ [\langle c_0,\sigma\rangle \to_1^n \sigma'' \ \&\ \langle c_1,\sigma''\rangle \to_1^* \sigma' \Rightarrow \langle c_0;c_1,\sigma\rangle \to_1^* \sigma']$$

于是引理成立。然后证明下面的定理：

$$\forall \sigma,\sigma'.[\langle c,\sigma\rangle \rightarrow_1^* \sigma' \text{ 当且仅当} \langle c,\sigma\rangle \rightarrow \sigma']$$

定理的充分性通过施结构归纳于 c 来证明，而当 c 为 while 循环时对计算的长度进行归纳。定理的必要性用规则归纳法（或者用对推导的归纳）去证明。 □ 51

4.4 算子及其最小不动点

从另一个角度看规则定义的集合，一组规则实例 R 确定了集合上算子 $\hat{R}$，$\hat{R}$ 作用于给定集合 B 得到了另一个集合 $\hat{R}(B)$：

$$\hat{R}(B) = \{y \mid \exists X \subseteq B.(X/y) \in R\}$$

也可以用算子 $\hat{R}$ 表示一个集合是 R 封闭的。

命题 4.11 一个集合 B 是 R 封闭的，当且仅当 $\hat{R}(B) \subseteq B$。

证明 由定义来直接给出证明。 □

算子 $\hat{R}$ 提供了构造集合 I_R 的方法。如果

$$A \subseteq B \Rightarrow \hat{R}(A) \subseteq \hat{R}(B)$$

则算子 $\hat{R}$ 是单调的。将算子 $\hat{R}$ 反复作用于空集 $\emptyset$，得到以下集合序列：

$$\begin{aligned}
A_0 &= \hat{R}^0(\emptyset) = \emptyset \\
A_1 &= \hat{R}^1(\emptyset) = \hat{R}(\emptyset) \\
A_2 &= \hat{R}(\hat{R}(\emptyset)) = \hat{R}^2(\emptyset) \\
&\vdots \\
A_n &= \hat{R}^n(\emptyset) \\
&\vdots
\end{aligned}$$

集合 A_1 由所有的公理实例的结论组成；一般地，集合 A_{n+1} 中的元素从前提在 A_n 中的规则实例直接推导得到。显然 $\emptyset \subseteq \hat{R}(\emptyset)$，即 $A_0 \subseteq A_1$。因算子 $\hat{R}$ 是单调的，所以 $\hat{R}(A_0) \subseteq \hat{R}(A_1)$，也就是 $A_1 \subseteq A_2$。用同样的方法可以得到 $A_2 \subseteq A_3, \cdots$，于是这个序列构成一条链

$$A_0 \subseteq A_1 \subseteq \cdots \subseteq A_n \subseteq \cdots$$

取 $A = \cup_{n\in\omega} A_n$，我们有下面的命题。

命题 4.12

(i) A 是 R 封闭的。

(ii) $\hat{R}(A) = A$。

(iii) A 是最小 R 封闭集。 52

证明

(i) 设 $(X/y) \in R$ 且 $X \subseteq A$。$A = \cup_n A_n$ 是递增集合链的并集。因为 X 是有穷的集合，所以存在某一 n，使得 $X \subseteq A_n$。（集合 X 或者为空，因此 $X \subseteq A_0$；或者形如 $\{x_1,\cdots,x_k\}$，在这种情况下对某些 $n_1,\cdots,n_k$，我们有 $x_1 \in A_{n_1},\cdots,x_k \in A_{n_k}$，取 n 为大于所有 $n_1,\cdots,n_k$ 的整数，因为

序列 $A_0, A_1, \cdots, A_n, \cdots$ 是递增的，所以必有 $X \subseteq A_n$。）由于 $X \subseteq A_n$，可得 $y \in \hat{R}(A_n) = A_{n+1}$，于是 $y \in \cup_n A_n = A$。从而证明了 A 是 R 封闭的。

(ii) 由命题 4.11 可知 A 是 R 封闭的，所以我们已经知道 $\hat{R}(A) \subseteq A$。下面只需证明 $A \subseteq \hat{R}(A)$。设 $y \in A$，则对于某个 $n > 0$ 有 $y \in A_n$，于是 $y \in \hat{R}(A_{n-1})$。这说明存在某个 $(X/y) \in R$，且 $X \subseteq A_{n-1}$，再由 $A_{n-1} \subseteq A$ 得 $X \subseteq A$，且 $(X/y) \in R$，$y \in \hat{R}(A)$，于是 $A \subseteq \hat{R}(A)$。这样就证明了 $\hat{R}(A) = A$。

(iii) 要证明 A 是最小 R 封闭集，只要证明如果 B 是另一个 R 封闭集，则 $A \subseteq B$。设 B 是 R 封闭的，则 $\hat{R}(B) \subseteq B$。下面我们用数学归纳法证明：对于所有的自然数 $n \in \omega$，

$$A_n \subseteq B$$

因为 A_0 是空集，奠基 $A_0 \subseteq B$ 显然为真。为了证明归纳步骤，假设 $A_n \subseteq B$。因为 $\hat{R}$ 是单调的且 B 是 R 封闭的，所以

$$A_{n+1} = \hat{R}(A_n) \subseteq \hat{R}(B) \subseteq B$$

命题得证。 □

请注意，证明(i)的关键在于规则实例是有穷的，即在规则实例 (X/y) 中，前提的集合 X 是有穷的。(i)和(iii)说明 $A = I_R$，且其元素都存在 R 推导。而(ii)恰好说明了 I_R 是 $\hat{R}$ 的一个不动点。而且，(iii)蕴涵 I_R 是 $\hat{R}$ 的最小不动点，即

$$\hat{R}(B) = B \Rightarrow I_R \subseteq B$$

因为如果其他任一集合 B 是不动点，则 B 是对 R 封闭的，因此根据命题 4.1 就可以得到 $I_R \subseteq B$。由规则实例 R 定义的集合 I_R 是通过构造

$$fix(\hat{R}) =_{def} \bigcup_{n \in \omega} \hat{R}^n(\emptyset)$$

53 获得的最小不动点 $fix(\hat{R})$，下一章中，最小不动点将起关键作用。

练习 4.13 给定一组规则 R，定义另一个算子 $\bar{R}$：

$$\bar{R}(A) = A \cup \{y \mid \exists X \subseteq A.\ (X/y) \in R\}$$

显然 $\bar{R}$ 是单调的，且满足性质

$$A \subseteq \bar{R}(A)$$

称满足这个性质的算子是递增的。试给出一个非递增的单调算子。证明，给定任一集合 A，$\bar{R}$ 存在一个包含 A 的最小不动点，且该性质对单调运算可能不成立。 □

练习 4.14 设 R 是一组规则实例，试证明 $\hat{R}$ 是连续的，即对于任意递增集合链 $B_0 \subseteq \cdots \subseteq B_n \subseteq \cdots$ 有

$$\bigcup_{n \in \omega} \hat{R}(B_n) = \hat{R}(\bigcup_{n \in \omega} B_n)$$

（我们将在下一章给出这个练习的答案。） □

4.5 进一步阅读资料

本章简要介绍了归纳定义的数学理论,更详细的讨论请参考 Peter Aczel 的手册[4]的有关章节。本书没有引入序数,所有的规则都是有限的。作者 1984 年在剑桥大学讲座的时候提出了“规则归纳法”的概念,它看上去比较容易理解(规则归纳法的原理广为人知,例如,对于规则集合 R,文献[4]把它称为 R 归纳)。在论证操作语义时可以用规则归纳法,也可以用概念清晰但过程冗长的对推导的归纳法,本书并不强求用哪一种方法,这完全取决于个人的喜好。 54

第 5 章　IMP 的指称语义

本章介绍 **IMP** 语言的指称语义，并证明它与第 2 章给出的操作语义是等价的。本章还介绍了指称语义的数学基础（完全偏序、连续函数和最小不动点）以及克纳斯特-塔尔斯基（Knaster-Tarski）定理。

5.1　目的

上一章我们从操作语义的角度讨论了用 **IMP** 语言编写的程序的行为特征，通过对转换关系的定义，我们描述了表达式的求值和命令的执行。其中对规则的选择是任意的，例如在选择转换步骤的大小时，可以选择单步执行的规则，也可以选择完全执行的规则。还要注意的是在描述程序行为时用到了混合型语法。这种用语法建立转换关系的语义风格使得两种不同的程序设计语言编写的程序很难比较。尽管如此，操作语义与语言的实现相当紧密，例如，**IMP** 的操作语义描述很容易转换成用 Prolog 编写的解释器。根据操作语义还得到算术表达式、布尔表达式和命令之间的等价定义。例如，我们有如下定义：

$$c_0 \sim c_1 \text{ 当且仅当 } (\forall \sigma,\sigma'.\langle c_0,\sigma\rangle \to \sigma' \Leftrightarrow \langle c_1,\sigma\rangle \to \sigma')$$

读者可能已经意识到，捕获 **IMP** 语义有更直接的方法，如果我们只对命令之间的等价 ~ 感兴趣，注意到 $c_0 \sim c_1$ 当且仅当

$$\{(\sigma,\sigma') \mid \langle c_0,\sigma\rangle \to \sigma'\} = \{(\sigma,\sigma') \mid \langle c_1,\sigma\rangle \to \sigma'\}$$

换句话说，$c_0 \sim c_1$ 当且仅当 c_0 和 c_1 确定了状态集上的同一个部分函数。这就意味着可以在更抽象的层次上来定义 **IMP** 的语义，此时我们用状态集上的部分函数来表示命令的指称，并称这种新的语义描述风格为*指称语义*。指称语义的应用非常广泛，尽管它的基本框架不适于描述并行性和“公平性”（见第 14 章），但实际上它可以描述包括 **IMP** 这种简单程序设计语言在内的几乎所有的程序设计语言。指称语义是由斯特雷奇（Christopher Strachey）提出的，斯科特（Dana Scott）奠定了它的数学理论基础。我们仅以 **IMP** 语言为例介绍指称语义。接下来几章我们将着重讨论指称语义的应用及其理论基础。

算术表达式 $a \in \mathbf{Aexp}$ 指称函数 $\mathscr{A}[a]: \Sigma \to \mathbf{N}$。

布尔表达式 $b \in \mathbf{Bexp}$ 指称函数 $\mathscr{B}[b]: \Sigma \to \mathbf{T}$，该函数是从状态集到真值集的映射。 55

命令 $c \in \mathbf{Com}$ 指称部分函数 $\mathscr{C}[c]: \Sigma \to \Sigma$。

括号 $[\,]$ 是指称语义的一个常用符号。我们看到，$\mathscr{A}$ 实际上是类型为 $\mathbf{Aexp} \to (\Sigma \to \mathbf{N})$ 的算术表达式的一个函数。在普通代数中，表达式是要计算的，而括号 $[a]$ 却表示引用算术表达式 a 而不是计算 a。我们当然也可以用 $\mathscr{A}(\text{“}3+5\text{”})\sigma = 8$ 来代替 $\mathscr{A}[3+5]\sigma = 8$。引号说明语法对象“3 +5”受 $\mathscr{A}$ 作用。但是对于 $\mathscr{A}[a_0 + a_1]$（其中 a_0 和 a_1 是表示算术表达式的元变量）来说，情况稍微有些不同。括号 $[\,]$ 内的语法对象是由两个语法对象 a_0 和 a_1 加上中间的一个

符号“ + ”得到的，因此括号[ ]并不是完整意义上的引用。我们使用括号[ ]围起来的是语义函数的一个变量，以此来表示该变量是一个语法对象。

5.2 指称语义

我们用结构归纳法定义语义函数：

$$\mathscr{A}:\mathbf{Aexp} \to (\Sigma \to \mathbf{N})$$
$$\mathscr{B}:\mathbf{Bexp} \to (\Sigma \to \mathbf{T})$$
$$\mathscr{C}:\mathbf{Com} \to (\Sigma \to \Sigma)$$

例如，对于命令，当我们给每条命令 c 定义一个部分函数 $\mathscr{C}[c]$ 时，假定 c 的子命令 c' 已有定义。称命令 c 指称 $\mathscr{C}[c]$，或称 $\mathscr{C}[c]$ 是 c 的指称。

算术表达式的指称

首先，用结构归纳法把算术表达式的指称定义为状态和数之间的关系：

$$\mathscr{A}[n] = \{(\sigma,n) \mid \sigma \in \Sigma\}$$
$$\mathscr{A}[X] = \{(\sigma,\sigma(X)) \mid \sigma \in \Sigma\}$$
$$\mathscr{A}[a_0 + a_1] = \{(\sigma,n_0 + n_1) \mid (\sigma,n_0) \in \mathscr{A}[a_0] \ \&\ (\sigma,n_1) \in \mathscr{A}[a_1]\}$$
$$\mathscr{A}[a_0 - a_1] = \{(\sigma,n_0 - n_1) \mid (\sigma,n_0) \in \mathscr{A}[a_0] \ \&\ (\sigma,n_1) \in \mathscr{A}[a_1]\}$$
$$\mathscr{A}[a_0 \times a_1] = \{(\sigma,n_0 \times n_1) \mid (\sigma,n_0) \in \mathscr{A}[a_0] \ \&\ (\sigma,n_1) \in \mathscr{A}[a_1]\}$$

对算术表达式 a 进行结构归纳，可以看到每个指称 $\mathscr{A}[a]$ 其实是一个函数。注意等号左边的符号“ + ”，“ - ”，“ × ”表示 **IMP** 的语法符号，而右边的符号则表示对数字的运算。因此，对于任一状态 σ，有

56

$$\mathscr{A}[3 + 5]\sigma = \mathscr{A}[3]\sigma + \mathscr{A}[5]\sigma = 3 + 5 = 8$$

也可以用 λ 记号来给出语义的定义，例如：

$$\mathscr{A}[n] = \lambda\sigma \in \Sigma.\, n$$
$$\mathscr{A}[X] = \lambda\sigma \in \Sigma.\, \sigma(X)$$
$$\mathscr{A}[a_0 + a_1] = \lambda\sigma \in \Sigma.\, (\mathscr{A}[a_0]\sigma + \mathscr{A}[a_1]\sigma)$$
$$\mathscr{A}[a_0 - a_1] = \lambda\sigma \in \Sigma.\, (\mathscr{A}[a_0]\sigma - \mathscr{A}[a_1]\sigma)$$
$$\mathscr{A}[a_0 \times a_1] = \lambda\sigma \in \Sigma.\, (\mathscr{A}[a_0]\sigma \times \mathscr{A}[a_1]\sigma)$$

布尔表达式的指称

布尔表达式的语义函数由作用于真值集的逻辑运算符：合取 $\wedge_T$、析取 $\vee_T$ 以及否定 $\neg_T$ 给出。由结构归纳法，布尔表达式的指称被定义为状态和真值之间的关系：

$$\mathscr{B}[\mathbf{true}] = \{(\sigma,\mathbf{true}) \mid \sigma \in \Sigma\}$$

$$\mathscr{B}[\mathbf{false}] = \{(\sigma,\mathbf{false}) \mid \sigma \in \Sigma\}$$

$$\mathscr{B}[a_0 = a_1] = \{(\sigma,\mathbf{true}) \mid \sigma \in \Sigma \ \&\ \mathscr{A}[a_0]\sigma = \mathscr{A}[a_1]\sigma\} \cup$$

$$\{(\sigma,\mathbf{false}) \mid \sigma \in \Sigma \ \&\ \mathscr{A}[a_0]\sigma \neq \mathscr{A}[a_1]\sigma\}$$

$$\mathscr{B}[a_0 \leqslant a_1] = \{(\sigma,\mathbf{true}) \mid \sigma \in \Sigma \ \&\ \mathscr{A}[a_0]\sigma \leqslant \mathscr{A}[a_1]\sigma\} \cup$$

$$\{(\sigma,\mathbf{false}) \mid \sigma \in \Sigma \ \&\ \mathscr{A}[a_0]\sigma \not\leqslant \mathscr{A}[a_1]\sigma\}$$

$$\mathscr{B}[\neg\, b] = \{(\sigma, \neg_T t) \mid \sigma \in \Sigma \ \&\ (\sigma,t) \in \mathscr{B}[b]\}$$

$$\mathscr{B}[b_0 \wedge b_1] = \{(\sigma,t_0 \wedge_T t_1 \mid \sigma \in \Sigma \ \&\ (\sigma,t_0) \in \mathscr{B}[b_0] \ \&\ (\sigma,t_1) \in \mathscr{B}[b_1]\}$$

$$\mathscr{B}[b_0 \vee b_1] = \{(\sigma,t_0 \vee_T t_1 \mid \sigma \in \Sigma \ \&\ (\sigma,t_0) \in \mathscr{B}[b_0] \ \&\ (\sigma,t_1) \in \mathscr{B}[b_1]\}$$ 57

对布尔表达式进行简单结构归纳，可以看出每个指称其实是一个函数。例如，对于所有的 $\sigma \in \Sigma$，有

$$\mathscr{B}[a_0 \leqslant a_1]\sigma = \begin{cases} \mathbf{true} & \mathscr{A}[a_0]\sigma \leqslant \mathscr{A}[a_1]\sigma \\ \mathbf{false} & \mathscr{A}[a_0]\sigma \not\leqslant \mathscr{A}[a_1]\sigma \end{cases}$$

命令的指称

对于命令 c 来说，$\mathscr{C}[c]$的定义比较复杂。我们首先给定指称为状态之间的关系，接着，用结构归纳法可以看到，每个指称其实是一个部分函数。显然，我们有

$$\mathscr{C}[\mathbf{skip}] = \{(\sigma,\sigma) \mid \sigma \in \Sigma\}$$

$$\mathscr{C}[X := a] = \{(\sigma,\sigma[n/X]) \mid \sigma \in \Sigma \ \&\ n = \mathscr{A}[a]\sigma\}$$

$$\mathscr{C}[c_0;c_1] = \mathscr{C}[c_1] \circ \mathscr{C}[c_0], \text{关系的复合}$$

其定义说明了 c_0 和 c_1 的相反次序，

$$\mathscr{C}[\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1] =$$
$$\{(\sigma,\sigma') \mid \mathscr{B}[b]\sigma = \mathbf{true} \ \&\ (\sigma,\sigma') \in \mathscr{C}[c_0]\} \cup$$
$$\{(\sigma,\sigma') \mid \mathscr{B}[b]\sigma = \mathbf{false} \ \&\ (\sigma,\sigma') \in \mathscr{C}[c_1]\}$$

但是考虑 while 循环命令的指称时有些复杂，令

$$w \equiv \mathbf{while}\ b\ \mathbf{do}\ c$$

我们已经指出过下述二个语句是等价的

$$w \sim \mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip}$$

因此，部分函数 $\mathscr{C}[w]$与 $\mathscr{C}[\mathbf{if}\ b\ \mathbf{then}\ c;w\ \mathbf{else\ skip}]$是相等的。于是，我们有

$$\begin{aligned}\mathscr{C}[w] &= \{(\sigma,\sigma') \mid \mathscr{B}[b]\sigma = \mathbf{true} \ \&\ (\sigma,\sigma') \in \mathscr{C}[c;w]\} \cup \\ &\quad \{(\sigma,\sigma) \mid \mathscr{B}[b]\sigma = \mathbf{false}\} \\ &= \{(\sigma,\sigma') \mid \mathscr{B}[b]\sigma = \mathbf{true} \ \&\ (\sigma,\sigma') \in \mathscr{C}[w] \circ \mathscr{C}[c]\} \cup \\ &\quad \{(\sigma,\sigma) \mid \mathscr{B}[b]\sigma = \mathbf{false}\}\end{aligned}$$

用 φ 表示 $\mathscr{C}[w]$、β 表示 $\mathscr{B}[b]$、γ 表示 $\mathscr{C}[c]$，我们要求部分函数 φ 满足

$$\varphi = \{(\sigma,\sigma') \mid \beta(\sigma) = \textbf{true} \ \& \ (\sigma,\sigma') \in \varphi \circ \gamma\} \cup$$
58
$$\{(\sigma,\sigma) \mid \beta(\sigma) = \textbf{false}\}$$

但是等式两边都含有 φ,这时如何来求得 φ 呢? 显然,我们需要某一技术解这类形式的递归方程。(我们将这种左边待求的值出现在右边的方程称为递归方程。) 用另一种方式来考察,我们把 Γ 作为输入 φ 则返回 $\Gamma(\varphi)$ 的一个函数,其中

$$\begin{aligned}\Gamma(\varphi) &= \{(\sigma,\sigma') \mid \beta(\sigma) = \textbf{true} \ \& \ (\sigma,\sigma') \in \varphi \circ \gamma\} \cup \\ &\quad \{(\sigma,\sigma) \mid \beta(\sigma) = \textbf{false}\} \\ &= \{(\sigma,\sigma') \mid \exists \sigma''. \beta(\sigma) = \textbf{true} \ \& \ (\sigma,\sigma'') \in \gamma \ \& \ (\sigma'',\sigma') \in \varphi\} \cup \\ &\quad \{(\sigma,\sigma) \mid \beta(\sigma) = \textbf{false}\}\end{aligned}$$

我们需要的是 Γ 的一个不动点 φ,使得

$$\varphi = \Gamma(\varphi)$$

4.4 节提供了求这一个解的思路。不难看出,函数 Γ 等于 $\hat{R}$,其中 $\hat{R}$ 是由规则实例 R 确定的集合上的算子,规则实例 R 为

$$\begin{aligned}R = &\{(\{(\sigma'',\sigma')\}/(\sigma,\sigma')) \mid \beta(\sigma) = \textbf{true} \ \& \ (\sigma,\sigma'') \in \gamma\} \cup \\ &\{(\emptyset/(\sigma,\sigma)) \mid \beta(\sigma) = \textbf{false}\}\end{aligned}$$

在 4.4 节中曾经指出 $\hat{R}$ 有一个最小不动点

$$\varphi = fix(\hat{R})$$

其中 φ 是具有如下性质的一个集合 (此处是一个序偶集合)

$$\hat{R}(\theta) = \theta \Rightarrow \varphi \subseteq \theta$$

我们将这个最小不动点作为 while 循环 w 的指称。显然,w 的指称应该是不动点,但为什么是最小不动点呢? 下一节我们会加以说明。在下节读者会发现,对指称语义选择最小不动点还能够与操作语义保持一致。

59 下面,我们用结构归纳法定义命令的指称语义:

$$\mathscr{C}[\textbf{skip}] = \{(\sigma,\sigma) \mid \sigma \in \Sigma\}$$

$$\mathscr{C}[X := a] = \{(\sigma,\sigma[n/X]) \mid \sigma \in \Sigma \ \& \ n = \mathscr{A}[a]\sigma\}$$

$$\mathscr{C}[c_0;c_1] = \mathscr{C}[c_1] \circ \mathscr{C}[c_0]$$

$$\begin{aligned}&\mathscr{C}[\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1] = \\ &\{(\sigma,\sigma') \mid \mathscr{B}[b]\sigma = \textbf{true} \ \& \ (\sigma,\sigma') \in \mathscr{C}[c_0]\} \cup \\ &\{(\sigma,\sigma') \mid \mathscr{B}[b]\sigma = \textbf{false} \ \& \ (\sigma,\sigma') \in \mathscr{C}[c_1]\}\end{aligned}$$

$$\mathscr{C}[\textbf{while } b \textbf{ do } c] = fix(\Gamma)$$

其中,

$$\Gamma(\varphi) = \{(\sigma,\sigma') \mid \mathscr{B}[b]\sigma = \textbf{true} \ \& \ (\sigma,\sigma') \in \varphi \circ \mathscr{C}[c]\} \cup$$

$$\{(\sigma,\sigma) \mid \mathscr{B}[\![b]\!]\sigma = \textbf{false}\}$$

在这种方法中,我们将每条命令的指称定义为状态之间的关系。关于语义的定义,我们采取了复合的方式,一条命令的指称是由其直接子命令的指称构成的,这正是结构归纳法的特点,也是指称语义的特点。但是,**IMP** 的操作语义并不具备这个特点,因为在 **while** 循环的规则中,**while** 循环重新出现在规则的前提部分。

我们曾把 **while** 程序展开成条件命令,并讨论了展开式与 **while** 循环在操作语义上的等价性,本章在此基础上给出了 **while** 程序语义函数的定义。下面,我们根据指称语义考察这种等价性。

命题 5.1 设

$$w \equiv \textbf{while } b \textbf{ do } c$$

其中 c 是命令,b 是布尔表达式,则有

$$\mathscr{C}[\![w]\!] = \mathscr{C}[\![\textbf{if } b \textbf{ then } c;w \textbf{ else skip}]\!]$$ 60

证明 由上述定义,w 的指称是 Γ 的不动点,于是

$$\begin{aligned}
\mathscr{C}[\![w]\!] &= \Gamma(\mathscr{C}[\![w]\!]) \\
&= \{(\sigma,\sigma') \mid \mathscr{B}[\![b]\!]\sigma = \textbf{true} \ \&\ (\sigma,\sigma') \in \mathscr{C}[\![w]\!] \circ \mathscr{C}[\![c]\!]\} \cup \\
&\quad \{(\sigma,\sigma) \mid \mathscr{B}[\![b]\!]\sigma = \textbf{false}\} \\
&= \{(\sigma,\sigma') \mid \mathscr{B}[\![b]\!]\sigma = \textbf{true} \ \&\ (\sigma,\sigma') \in \mathscr{C}[\![c;w]\!]\} \cup \\
&\quad \{(\sigma,\sigma') \mid \mathscr{B}[\![b]\!]\sigma = \textbf{false} \ \&\ (\sigma,\sigma') \in \mathscr{C}[\![\textbf{skip}]\!]\} \\
&= \mathscr{C}[\![\textbf{if } b \textbf{ then } c;w \textbf{ else skip}]\!]
\end{aligned}$$ □

练习 5.2 对所有的命令 c,用结构归纳法证明指称 $\mathscr{C}[\![c]\!]$是一个部分函数。(在证明 while 循环的时候,用数学归纳法证明:对于所有的自然数 n,$\Gamma^n(\emptyset)$是状态之间的部分函数,它们组成了一条递增链,并且这样一条部分函数链的并集本身也是一个部分函数。) □

在 5.4 节我们将介绍不动点的一般理论,它适用于递归定义的对象不是按包含关系排序的集合。

5.3 语义的等价性

尽管我们在定义指称语义时受到了 **IMP** 操作行为的启发,但我们还没有证明指称语义和操作语义的一致性。我们首先考察两者在表达式求值上的一致性。

引理 5.3 对所有的 $a \in \textbf{Aexp}$,有

$$\mathscr{A}[\![a]\!] = \{(\sigma,n) \mid \langle a,\sigma\rangle \to n\}$$

证明 用结构归纳法证明该引理。取性质为:

$$P(a) \Longleftrightarrow_{def} \mathscr{A}[\![a]\!] = \{(\sigma,n) \mid \langle a,\sigma\rangle \to n\}$$

采用结构归纳法的证明模式,根据算术表达式 a 的结构,证明分为如下几种情况:

$a \equiv m$:由语义函数的定义,当 a 是整数 m 的时候,有

61

$$(\sigma,n) \in \mathscr{A}[m] \Longleftrightarrow \sigma \in \Sigma \ \& \ n \equiv m$$

显然,如果$(\sigma,n) \in \mathscr{A}[m]$,那么$n \equiv m$且$\langle m,\sigma\rangle \to n$。反过来,如果$\langle m,\sigma\rangle \to n$,那么惟一可能的推导是$n \equiv m$的推导,于是$(\sigma,n) \in \mathscr{A}[m]$。

$a \equiv X$:同理,如果a是存储单元X,则

$$\begin{aligned}(\sigma,n) \in \mathscr{A}[X] &\Longleftrightarrow (\sigma \in \Sigma \ \& \ n \equiv \sigma(X)) \\ &\Longleftrightarrow \langle X,\sigma\rangle \to n\end{aligned}$$

$a \equiv a_0 + a_1$:假设对两个算术表达式a_0和a_1,性质$P(a_0)$和$P(a_1)$为真,则有

$$(\sigma,n) \in \mathscr{A}[a_0 + a_1] \Longleftrightarrow \exists n_0,n_1.\ n = n_0 + n_1 \ \& \ (\sigma,n_0) \in \mathscr{A}[a_0] \ \& \ (\sigma,n_1) \in \mathscr{A}[a_1]$$

设$(\sigma,n) \in \mathscr{A}[a_0 + a_1]$,则存在$n_0,n_1$,使得$n = n_0 + n_1$且$(\sigma,n_0) \in \mathscr{A}[a_0]$以及$(\sigma,n_1) \in \mathscr{A}[a_1]$。由假设$P(a_0)$和$P(a_1)$为真,得

$$\langle a_0,\sigma\rangle \to n_0 \quad 且 \quad \langle a_1,\sigma\rangle \to n_1$$

于是,我们能推出$\langle a_0 + a_1,\sigma\rangle \to n$。反过来,$\langle a_0 + a_1,\sigma\rangle \to n$的任一推导必有以下形式:

$$\frac{\dfrac{\vdots}{\langle a_0,\sigma\rangle \to n_0} \quad \dfrac{\vdots}{\langle a_1,\sigma\rangle \to n_1}}{\langle a_0 + a_1,\sigma\rangle \to n}$$

对某个n_0和n_1,使得$n = n_0 + n_1$。这时,由假设$P(a_0)$和$P(a_1)$为真,得$(\sigma,n_0) \in \mathscr{A}[a_0]$且$(\sigma,n_1) \in \mathscr{A}[a_1]$,于是$(\sigma,n) \in \mathscr{A}[a]$。

同理可证,当算术表达式形如$a_0 - a_1$和$a_0 \times a_1$的时候,性质$P(a_0 - a_1)$和$P(a_0 \times a_1)$也为真。根据对算术表达式的结构归纳法,这样就证明了对于所有的算术表达式a,有

$$\mathscr{A}[a] = \{(\sigma,n) \mid \langle a,\sigma\rangle \to n\}$$

□

引理 5.4 对所有的$b \in \mathbf{Bexp}$,有

$$\mathscr{B}[b] = \{(\sigma,t) \mid \langle b,\sigma\rangle \to t\}$$

证明 布尔表达式的证明和算术表达式的证明相似,施归纳于布尔表达式。对布尔表达式b,取性质$P(b)$为:

62

$$P(b) \Longleftrightarrow_{def} \mathscr{B}[b] = \{(\sigma,t) \mid \langle b,\sigma\rangle \to t\}$$

我们只给出两种典型归纳情况的证明,其他情况的证明留给读者。

$b \equiv (a_0 = a_1)$:令a_0和a_1为算术表达式,由定义得

$$\begin{aligned}\mathscr{B}[a_0 = a_1] = &\{(\sigma,\mathbf{true}) \mid \sigma \in \Sigma \ \& \ \mathscr{A}[a_0]\sigma = \mathscr{A}[a_1]\sigma\} \cup \\ &\{(\sigma,\mathbf{false}) \mid \sigma \in \Sigma \ \& \ \mathscr{A}[a_0]\sigma \neq \mathscr{A}[a_1]\sigma\}\end{aligned}$$

于是

$$(\sigma,\mathbf{true}) \in \mathscr{B}[a_0 = a_1] \Longleftrightarrow \sigma \in \Sigma \ \& \ \mathscr{A}[a_0]\sigma = \mathscr{A}[a_1]\sigma$$

如果$(\sigma,\mathbf{true})\in\mathscr{B}[\![a_0=a_1]\!]$,则$\mathscr{A}[\![a_0]\!]\sigma=\mathscr{A}[\![a_1]\!]\sigma$,由引理5.3知,对某个数$n$有

$$\langle a_0,\sigma\rangle\to n \quad 且 \quad \langle a_1,\sigma\rangle\to n$$

因此,由布尔表达式的操作语义,我们得到

$$\langle a_0 = a_1,\sigma\rangle\to\mathbf{true}$$

反过来,假设$\langle a_0=a_1,\sigma\rangle\to\mathbf{true}$,则必存在如下形式的推导

$$\frac{\dfrac{\vdots}{\langle a_0,\sigma\rangle\to n} \quad \dfrac{\vdots}{\langle a_1,\sigma\rangle\to n}}{\langle a_0 = a_1,\sigma\rangle\to\mathbf{true}}$$

再由引理5.3,得$\mathscr{A}[\![a_0]\!]\sigma=n=\mathscr{A}[\![a_1]\!]\sigma$。于是,$(\sigma,\mathbf{true})\in\mathscr{B}[\![a_0=a_1]\!]$,因此

$$(\sigma,\mathbf{true}) \in \mathscr{B}[\![a_0 = a_1]\!]\Longleftrightarrow\langle a_0 = a_1,\sigma\rangle\to\mathbf{true}$$

同理,

$$(\sigma,\mathbf{false}) \in \mathscr{B}[\![a_0 = a_1]\!]\Longleftrightarrow\langle a_0 = a_1,\sigma\rangle\to\mathbf{false}$$

这样就得到

$$\mathscr{B}[\![a_0 = a_1]\!] = \{(\sigma,t)\mid\langle a_0 = a_1,\sigma\rangle\to t\}$$

$b\equiv(b_0\wedge b_1)$:令b_0和b_1为布尔表达式,假设$P(b_0)$和$P(b_1)$为真,由定义得

$$\begin{aligned}&(\sigma,t) \in \mathscr{B}[\![b_0 \wedge b_1]\!]\Longleftrightarrow\\&\sigma \in \Sigma \ \&\ \exists t_0,t_1.\ t = t_0 \wedge_T t_1 \ \&\ (\sigma,t_0) \in \mathscr{B}[\![b_0]\!]\&(\sigma,t_1) \in \mathscr{B}[\![b_1]\!]\end{aligned}$$

因此,我们设$(\sigma,t)\in\mathscr{B}[\![b_0\wedge b_1]\!]$,则存在$t_0,t_1$,使得$(\sigma,t_0)\in\mathscr{B}[\![b_0]\!]$且$(\sigma,t_1)\in\mathscr{B}[\![b_1]\!]$。由假设$P(b_0)$和$P(b_1)$为真,得

$$\langle b_0,\sigma\rangle\to t_0 \quad 且 \quad \langle b_1,\sigma\rangle\to t_1$$ 63

于是可以推导$\langle b_0\wedge b_1,\sigma\rangle\to t$,其中$t=t_0\wedge_T t_1$。反过来,$\langle b_0\wedge b_1,\sigma\rangle\to t$的任一推导必有以下形式:

$$\frac{\dfrac{\vdots}{\langle b_0,\sigma\rangle\to t_0} \quad \dfrac{\vdots}{\langle b_1,\sigma\rangle\to t_1}}{\langle b_0 \wedge b_1,\sigma\rangle\to t}$$

对某个t_0和t_1,使得$t=t_0\wedge_T t_1$。由$P(b_0)$和$P(b_1)$为真,得$(\sigma,t_0)\in\mathscr{B}[\![b_0]\!]$且$(\sigma,t_1)\in\mathscr{B}[\![b_1]\!]$。于是,$(\sigma,t)\in\mathscr{B}[\![b]\!]$。

其他情况的证明完全类似。 □

练习5.5 上面两个引理是根据推导的形式来证明的,也可以把结构归纳法和规则归纳法结合起来进行证明。例如,试证明:对所有的算术表达式a,

1. $\{(\sigma,n)\mid\langle a,\sigma\rangle\to n\}\subseteq\mathscr{A}[\![a]\!]$
2. $\mathscr{A}[\![a]\!]\subseteq\{(\sigma,n)\mid\langle a,\sigma\rangle\to n\}$

使用规则归纳法对算术表达式的操作语义进行归纳证明 1，使用结构归纳法对算术表达式进行归纳证明 2。 □

下面我们证明命令的指称语义与其操作语义的一致性。

引理 5.6 对所有的命令 c 和状态 σ,σ'，有

$$\langle c,\sigma\rangle \to \sigma' \Rightarrow (\sigma,\sigma') \in \mathscr{C}[\![c]\!]$$

证明 我们用 4.3.3 节的方法，对命令的操作语义使用规则归纳法进行证明。对 $c \in \mathbf{Com}, \sigma,\sigma' \in \Sigma$，定义性质 P 为

$$P(c,\sigma,\sigma') \Longleftrightarrow_{def} (\sigma,\sigma') \in \mathscr{C}[\![c]\!]$$

根据 4.3.3 节的方法，只要能够证明 P 对命令执行的规则是封闭的，那么对任意的命令 c 和状态 σ,σ'，就有

$$\langle c,\sigma\rangle \to \sigma' \Rightarrow P(c,\sigma,\sigma')$$

这里，我们只考察 4.3.3 节 **while** 循环中条件为真的情况。其规则是：

$$\frac{\langle b,\sigma\rangle \to \mathbf{true} \quad \langle c,\sigma\rangle \to \sigma'' \quad \langle w,\sigma''\rangle \to \sigma'}{\langle w,\sigma\rangle \to \sigma'}$$

64

其中 $w \equiv$ **while** b **do** c。按照 4.3.3 节的证明模式，假设

$$\langle b,\sigma\rangle \to \mathbf{true} \ \& \ \langle c,\sigma\rangle \to \sigma'' \ \& \ P(c,\sigma,\sigma'') \ \& \ \langle w,\sigma''\rangle \to \sigma' \ \& \ P(w,\sigma'',\sigma')$$

由引理 5.4，

$$\mathscr{B}[\![b]\!]\sigma = \mathbf{true}$$

根据 P 的定义直接可得

$$\mathscr{C}[\![c]\!]\sigma = \sigma'' \quad 且 \quad \mathscr{C}[\![w]\!]\sigma'' = \sigma'$$

由指称语义的定义，得

$$\mathscr{C}[\![w]\!]\sigma = \mathscr{C}[\![c;w]\!]\sigma = \mathscr{C}[\![w]\!](\mathscr{C}[\![c]\!]\sigma) = \mathscr{C}[\![w]\!]\sigma'' = \sigma'$$

但是，$\mathscr{C}[\![w]\!]\sigma = \sigma'$ 意味着 $P(w,\sigma,\sigma')$ 为真，即对规则的结论性质 P 成立，因此 P 对该规则封闭。类似地，我们可以证明性质 P 对其他的命令执行规则也是封闭的（留作练习）。于是，根据规则归纳法，引理得证。 □

下面的定理说明了命令的操作语义和指称语义是等价的，我们用结构归纳法来证明，其中对 while 循环用数学归纳法证明。

定理 5.7 对所有的命令 c，有

$$\mathscr{C}[\![c]\!] = \{(\sigma,\sigma') \mid \langle c,\sigma\rangle \to \sigma'\}$$

证明 定理重新表述为：对所有的命令 c，所有的状态 σ,σ'，有

$$(\sigma,\sigma') \in \mathscr{C}[c] \Longleftrightarrow \langle c,\sigma\rangle \to \sigma'$$

引理 5.6 已经证明了"⇐",下面证明"⇒"。

施结构归纳于命令 c。取性质 P 为

$$\forall \sigma,\sigma' \in \Sigma.(\sigma,\sigma') \in \mathscr{C}[c] \Longleftrightarrow \langle c,\sigma\rangle \to \sigma'$$

$c \equiv$ **skip**:由定义,$\mathscr{C}[\mathbf{skip}] = \{(\sigma,\sigma) \mid \sigma \in \Sigma\}$。如果$(\sigma,\sigma') \in \mathscr{C}[c]$,则 $\sigma' = \sigma$,根据 **skip** 规则,$\langle \mathbf{skip},\sigma\rangle \to \sigma'$,在这种情况下性质 P 成立。

$c \equiv X := a$:设$(\sigma,\sigma') \in \mathscr{C}[X := a]$,则 $\sigma' = \sigma[n/X]$,其中 $n = \mathscr{A}[a]\sigma$。由引理 5.3,$\langle a,\sigma\rangle \to n$。于是$\langle c,\sigma\rangle \to \sigma'$,在这种情况下性质 P 成立。 65

$c \equiv c_0;c_1$:假设对命令 c_0 和 c_1,性质 P 成立。设$(\sigma,\sigma') \in \mathscr{C}[c]$,则存在某个状态 σ''使得$(\sigma,\sigma'') \in \mathscr{C}[c_0]$且$(\sigma'',\sigma') \in \mathscr{C}[c_1]$。根据命令 c_0 和 c_1 的归纳假设,我们知道,

$$\langle c_0,\sigma\rangle \to \sigma'' \quad 且 \quad \langle c_1,\sigma''\rangle \to \sigma'$$

于是,根据命令的操作语义的复合规则,有$\langle c_0;c_1,\sigma\rangle \to \sigma'$。于是,对命令 c 性质 P 成立。

$c \equiv$ **if** b **then** c_0 **else** c_1:假设对命令 c_0 和 c_1 性质 P 成立。因为

$$\begin{aligned}\mathscr{C}[c] = {} & \{(\sigma,\sigma') \mid \mathscr{B}[b]\sigma = \mathbf{true} \ \&\ (\sigma,\sigma') \in \mathscr{C}[c_0]\} \cup \\ & \{(\sigma,\sigma') \mid \mathscr{B}[b]\sigma = \mathbf{false} \ \&\ (\sigma,\sigma') \in \mathscr{C}[c_1]\}\end{aligned}$$

所以,如果$(\sigma,\sigma') \in \mathscr{C}[c]$,则

(i) $\mathscr{B}[b]\sigma = \mathbf{true} \ \&\ (\sigma,\sigma') \in \mathscr{C}[c_0]$,或者

(ii) $\mathscr{B}[b]\sigma = \mathbf{false} \ \&\ (\sigma,\sigma') \in \mathscr{C}[c_1]$。

如果是(i)的情况,由引理 5.4 得,$\langle b,\sigma\rangle \to \mathbf{true}$,又因为对 c_0 性质 P 成立,得$\langle c_0,\sigma\rangle \to \sigma'$。于是,根据命令的操作语义的条件规则,可得$\langle c,\sigma\rangle \to \sigma'$。如果是(ii)的情况,同理可得$\langle c,\sigma\rangle \to \sigma'$。于是,对命令 c 性质 P 成立。

$c \equiv$ **while** b **do** c_0:假设对命令 c_0,性质 P 成立。因为

$$\mathscr{C}[\mathbf{while}\ b\ \mathbf{do}\ c_0] = fix(\Gamma)$$

其中

$$\begin{aligned}\Gamma(\varphi) = {} & \{(\sigma,\sigma') \mid \mathscr{B}[b]\sigma = \mathbf{true} \ \&\ (\sigma,\sigma') \in \varphi \circ \mathscr{C}[c_0]\} \cup \\ & \{(\sigma,\sigma) \mid \mathscr{B}[b]\sigma = \mathbf{false}\}\end{aligned}$$

因此,用 θ_n 表示 $\Gamma^n(\emptyset)$,我们有

$$\mathscr{C}[c] = \bigcup_{n\in\omega} \theta_n$$

其中

$$\theta_0 = \emptyset$$

$$\theta_{n+1} = \{(\sigma,\sigma') \mid \mathscr{B}[b]\sigma = \mathbf{true} \ \&\ (\sigma,\sigma') \in \theta_n \circ \mathscr{C}[c_0]\} \cup$$

$$\{(\sigma,\sigma) \mid \mathscr{B}[b]\sigma = \textbf{false}\}$$

下面我们用数学归纳法证明:对所有的 $n \in \omega$,有

66

$$\forall \sigma,\sigma' \in \Sigma.\ (\sigma,\sigma') \in \theta_n \Rightarrow \langle c,\sigma\rangle \to \sigma' \tag{1}$$

从而推出,对所有的状态 σ,σ',有 $(\sigma,\sigma') \in \mathscr{C}[c] \Longleftrightarrow \langle c,\sigma\rangle \to \sigma'$。

施归纳于性质(1)。

奠基 $n=0$:当 $n=0$ 时,$\theta_0 = \emptyset$,性质(1)自动为真。

归纳步骤:假设对任意的 $n \in \omega$,性质(1)为真,我们要证明对于任何状态 σ,σ',有

$$(\sigma,\sigma') \in \theta_{n+1} \Rightarrow \langle c,\sigma\rangle \to \sigma'$$

设 $(\sigma,\sigma') \in \theta_{n+1}$,则

(i) $\mathscr{B}[b]\sigma = \textbf{true}\ \&\ (\sigma,\sigma') \in \theta_n \circ \mathscr{C}[c_0]$,或者

(ii) $\mathscr{B}[b]\sigma = \textbf{false}\ \&\ \sigma' = \sigma$。

如果是(i)的情况,由引理 5.4 得,$\langle b,\sigma\rangle \to \textbf{true}$。对某一状态 σ'',有 $(\sigma,\sigma'') \in \mathscr{C}[c_0]$ 且 $(\sigma'',\sigma') \in \theta_n$。根据归纳假设,有 $\langle c,\sigma''\rangle \to \sigma'$。又根据对命令 c_0 的结构归纳法的归纳假设有 $\langle c_0,\sigma\rangle \to \sigma''$,由 while 循环的规则得 $\langle c,\sigma\rangle \to \sigma'$。

如果是(ii)的情况,由于 $\mathscr{B}[b]\sigma = \textbf{false}$,由引理 5.4 得 $\langle b,\sigma\rangle \to \textbf{false}$,又 $\sigma' = \sigma$,所以 $\langle c,\sigma\rangle \to \sigma$,在这种情况下性质 P 成立。

这样,就证明了当 $n+1$ 时,性质(1)也为真。

从而,我们用数学归纳法证明了,当 $c \equiv \textbf{while}\ b\ \textbf{do}\ c_0$ 时,对于所有的状态 σ,σ',有

$$(\sigma,\sigma') \in \mathscr{C}[c] \Rightarrow \langle c,\sigma\rangle \to \sigma'$$

最后,根据结构归纳法,定理得证。 □

练习 5.8 令 $w \equiv \textbf{while}\ b\ \textbf{do}\ c_0$,试证明:

$$\begin{aligned}
\mathscr{C}[w]\sigma = \sigma' \text{ 当且仅当 } & \mathscr{B}[b]\sigma = \textbf{false}\ \&\ \sigma = \sigma' \\
& \text{或} \\
& \exists \sigma_0,\cdots,\sigma_n \in \Sigma \\
& \sigma = \sigma_0\ \&\ \sigma' = \sigma_n\ \&\ \mathscr{B}[b]\sigma_n = \textbf{false}\ \& \\
& \forall i(0 \leqslant i < n).\ \mathscr{B}[b]\sigma_i = \textbf{true}\ \&\ \mathscr{C}[c]\sigma_i = \sigma_{i+1}
\end{aligned}$$

(从左到右的证明可以对构造 w 指称时使用的 $\Gamma^n(\emptyset)$ 进行归纳;从右到左的证明可以对状态
67 链的长度进行归纳。) □

练习 5.9 设含 repeat 结构的简单命令式语言的命令的语法为:

$$c ::= X := e \mid c_0;c_1 \mid \textbf{if}\ b\ \textbf{then}\ c_0\ \textbf{else}\ c_1 \mid \textbf{repeat}\ c\ \textbf{until}\ b$$

其中 X 是存储单元,e 是算术表达式,b 是布尔表达式,c、c_0、c_1 是命令。根据对命令执行的理解,试说明怎样才能将 while 循环的语义转变为 repeat 循环的语义,并且给出:

(i) 以规则的形式给出 repeat 循环的操作语义，并产生如下形式的转换$\langle c,\sigma\rangle \to \sigma'$，它表示在状态 σ 下执行 c 后终止于状态 σ'。

(ii) 给出命令的指称语义，其中每条命令 c 都由一个从状态到状态的部分函数 $\mathscr{C}[\![c]\!]$ 指称。

(iii) 简要证明操作语义和指称语义的一致性，即$\langle c,\sigma\rangle \to \sigma'$ 当且仅当 $\mathscr{C}[\![c]\!]\sigma = \sigma'$，主要证明 c 是 repeat 循环时的情况。 □

5.4 完全偏序与连续函数

上一章介绍了归纳定义的基本理论。我们已经用归纳定义去表示 **IMP** 的指称语义。但是，实践中往往不能直接用集合上算子的最小不动点去给出递归定义，为此，我们引入更抽象的概念——完全偏序和连续函数来定义递归，两者都是指称语义的标准工具。下面，我们用归纳定义的方法，给出完全偏序和连续函数的定义，通过这种方法读者可以很容易理解完全偏序的更抽象的概念，以及它们和操作语义中的具体概念之间的紧密联系。

设 R 为形为(X/y)的一组规则实例。给定集合 B，考察 R 如何确定集合 B 上的算子 $\hat{R}$ 以产生新的集合 $\hat{R}(B)$：

$$\hat{R}(B) = \{y \mid \exists (X/y) \in R.\, X \subseteq B\}$$

以及考察如何由集合链

$$\emptyset \subseteq \hat{R}(\emptyset) \subseteq \cdots \subseteq \hat{R}^n(\emptyset) \subseteq \cdots$$

的并集得到算子 $\hat{R}$ 的最小不动点

$$fix(\hat{R}) =_{def} \bigcup_{n\in\omega} \hat{R}^n(\emptyset)$$

68

由

$$\hat{R}(fix(\hat{R})) = fix(\hat{R})$$

可知它是一个不动点。又因为任意的不动点 B 都包含 $fix(\hat{R})$，即

$$\hat{R}(B) = B \Rightarrow fix(\hat{R}) \subseteq B$$

因此$fix(\hat{R})$是最小不动点。实际上，4.4 节中的命题 4.12 证明了 $fix(\hat{R})$是最小 R 封闭集，我们可以用 B 去刻画 R 封闭集的特征，有

$$\hat{R}(B) \subseteq B$$

用这种方法，通过选择合适的规则实例 R，我们就能解 while 循环的指称所需的递归方程了。但是，我们还需要更一般的方法，从上面的例子中抽取能获得最小不动点的最本质的数学性质。这就引出了完全偏序和连续函数的概念。

所谓的“最小”，只有在包含关系或子集关系下才有意义，这里我们使用更一般的偏序概念。

定义 如果集合 P 上的二元关系 $\sqsubseteq$ 满足：

(i) 自反性：$\forall p \in P. p \sqsubseteq p$。

(ii) 传递性：$\forall p,q,r \in P. p \sqsubseteq q \ \& \ q \sqsubseteq r \Rightarrow p \sqsubseteq r$。

(iii) 反对称性：$\forall p,q \in P. p \sqsubseteq q \ \& \ q \sqsubseteq p \Rightarrow p = q$。

则称集合$(P, \sqsubseteq)$是一个偏序(p. o.)。

不是所有的偏序关系都支持集合的构造方法，在构造最小不动点中我们有$\cup_{n\in\omega} A_n$，它是从最小集 $\emptyset$ 开始的 ω 链 $A_0 \subseteq A_1 \subseteq \cdots A_n \subseteq \cdots$的并集。对并集按照包含关系排序，我们引入偏序关系的最小上界的概念。把这些性质与偏序概念结合，就得到了完全偏序的定义。

定义 对于偏序$(P, \sqsubseteq)$和子集$X \subseteq P$，称 p 是 X 的上界当且仅当

$$\forall q \in X. q \sqsubseteq p$$

称 p 是 X 的最小上界当且仅当

(i) p 是 X 的上界，且

(ii) 对所有上界 $q \in X$，有 $p \sqsubseteq q$。

当偏序的子集 X 有最小上界时，我们将其记作$\bigsqcup X$。可以将$\bigsqcup\{d_1, \cdots, d_m\}$记作 $d_1 \sqcup \cdots \sqcup d_m$。

69

定义 设$(D, \sqsubseteq_D)$是一个偏序。

该偏序的 ω 链是其元素的递增链 $d_0 \sqsubseteq_D d_1 \sqsubseteq_D \cdots \sqsubseteq_D d_n \sqsubseteq_D \cdots$。

如果对所有 ω 链 $d_0 \sqsubseteq_D d_1 \sqsubseteq_D \cdots \sqsubseteq_D d_n \sqsubseteq_D \cdots$都有最小上界，即 D 中元素的任一递增链$\{d_n \mid n \in \omega\}$都有一个在 D 中的最小上界$\bigsqcup\{d_n \mid n \in \omega\}$，记为$\bigsqcup_{n\in\omega} d_n$，则称$(D, \sqsubseteq_D)$是完全偏序（简记为 cpo）。

如果它有一个最小元$\perp_D$(称为底)，则称$(D, \sqsubseteq_D)$是含底的 cpo[⊖]。

记号 后面我们把完全偏序$(D, \sqsubseteq_D)$的次序简记为$\sqsubseteq$，当它含底元素时把它的底元素记作$\perp$。根据上下文可以明确知道所指的是哪一个完全偏序。

恒等关系排序的集合是完全偏序的，且不含底元素。这样的完全偏序称为离散的，或平坦的。

练习 5.10 证明对于任何集合 X，$(\mathscr{P}ow(X), \subseteq)$是含底的完全偏序。证明按$\subseteq$排序的部分函数 $\Sigma \rightharpoonup \Sigma$ 的集合组成含底的完全偏序。 □

与定义集合上运算相类似，我们在这里也定义从完全偏序 D 到 D 的函数$f: D \to D$。我们要求这种函数以某种方式保持 D 上的次序。为此，我们考察由规则实例 R 定义的算子。设

$$B_0 \subseteq B_1 \subseteq \cdots B_n \subseteq \cdots$$

则

$$\hat{R}(B_0) \subseteq \hat{R}(B_1) \subseteq \cdots \hat{R}(B_n) \subseteq \cdots$$

⊖ 本书的 cpo 称为(无底的)ω-cpo 或前域。

也是一条递增的集合链，这是因为 $\hat{R}$ 是单调的，即

$$B \subseteq C \Rightarrow \hat{R}(B) \subseteq \hat{R}(C)$$

根据单调性，因为每一 $B_n \subseteq \cup_{n\in\omega} B_n$，所以有

$$\bigcup_{n\in\omega} \hat{R}(B_n) \subseteq \hat{R}(\bigcup_{n\in\omega} B_n)$$

事实上，由于规则实例的有穷性，上面的式子对逆包含关系和相等关系也成立。设 $y \in \hat{R}(\cup_{n\in\omega} B_n)$，则对某个有穷集合 $X \subseteq \cup_{n\in\omega} B_n$，有 $(X/y) \in R$。因为 X 是有穷的，故对某个 n 有 $X \subseteq B_n$，所以有 $y \in \hat{R}(B_n)$，于是 $y \in \cup_{n\in\omega} \hat{R}(B_n)$。这样，对于任何递增链 $B_0 \subseteq \cdots \subseteq B_n \subseteq \cdots$，有 70

$$\bigcup_{n\in\omega} \hat{R}(B_n) = \hat{R}(\bigcup_{n\in\omega} B_n)$$

从而证明了 $\hat{R}$ 是连续的。上式之所以成立是因为规则的有限性，即每一规则 (X/y) 中的 X 只包含有限个前提。

下面我们用这些性质定义完全偏序之间的连续函数。

定义 从完全偏序 D 到 E 上的函数 $f: D \to E$ 是单调的当且仅当

$$\forall d, d' \in D.\ d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d')$$

这样一个函数 f 是连续的当且仅当它是单调的并且对 D 中的所有链 $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$ 有

$$\bigsqcup_{n\in\omega} f(d_n) = f(\bigsqcup_{n\in\omega} d_n)$$

这个定义的重要结论是：任何含底的完全偏序到自身的任一连续函数都有一个最小不动点。读者可以参考4.4 节中关于集合算子的讨论。事实上，我们引入前缀不动点以帮助读者理解对规则封闭的集合的概念。（请读者回忆集合 B 对规则实例 R 是封闭的当且仅当 $\hat{R}(B) \subseteq B$。）

定义 设函数 $f: D \to D$ 是完全偏序 D 上的连续函数。f 的不动点是 D 中使得 $f(d) = d$ 的元素 d。f 的前缀不动点是 D 中使得 $f(d) \sqsubseteq d$ 的元素 d。

下面，我们介绍一个简单而重要的不动点定理，该定理给出了完全偏序 D 上的连续函数 f 的最小不动点 $fix(f)$ 的显式构造法。

定理 5.11（不动点定理） 设函数 $f: D \to D$ 是含底的完全偏序 D 上的连续函数。定义

$$fix(f) = \bigsqcup_{n\in\omega} f^n(\perp)$$

则 $fix(f)$ 是 f 的一个不动点和最小前缀不动点，即 (i) $f(fix(f)) = fix(f)$，且 (ii) 如果 $f(d) \sqsubseteq d$，则 $fix(f) \sqsubseteq d$。因此 $fix(f)$ 是 f 的最小不动点。 71

证明

(i) 由连续性,

$$
\begin{aligned}
f(fix(f)) &= f(\bigsqcup_{n\in\omega} f^n(\perp)) \\
&= \bigsqcup_{n\in\omega} f^{n+1}(\perp) \\
&= (\bigsqcup_{n\in\omega} f^{n+1}(\perp)) \sqcup \{\perp\} \\
&= \bigsqcup_{n\in\omega} f^n(\perp) \\
&= fix(f)
\end{aligned}
$$

所以 $fix(f)$ 是不动点。

(ii) 设 d 是前缀不动点,显然,$\perp \sqsubseteq d$。由单调性,得 $f(\perp) \sqsubseteq f(d)$,由于 d 是前缀不动点,即 $f(d) \sqsubseteq d$,因此,$f(\perp) \sqsubseteq d$,通过归纳法可以推得 $f^n(\perp) \sqsubseteq d$。于是,$fix(f) = \bigsqcup_{n\in\omega} f^n(\perp) \sqsubseteq d$。

由于不动点一定也是前缀不动点,所以 $fix(f)$ 是 f 的最小不动点。 □

我们先简单讨论一下完全偏序和连续函数的直观意义,后面的章节以更精确的方式对此作进一步的讨论。完全偏序对应于数据的类型,这些数据用作计算的输入或输出。完全偏序之间的连续函数是可计算函数的模型。完全偏序的每一个元素都被看作是信息点,$x \sqsubseteq y$ 表示 x 逼近 y (即, x 的信息少于 y 的信息或 x 的信息等于 y 的信息)。显然,底 $\perp$ 包含的信息量最小。

接下来我们在上面的一般框架内讨论 **IMP** 的指称语义。我们用状态集 Σ 上的一个部分函数来指称一条命令。从表面上看这似乎不符合由命令计算的函数必须连续的要求,但是状态之间的部分函数可以看作是连续的完全函数。在状态集 Σ 中添加新元素 $\perp$,对于所有的状态 σ,有

$$\perp \sqsubseteq \sigma$$

这样把 Σ 扩充成完全偏序 $\Sigma_\perp$,它包含了一个额外的元素 $\perp$,$\perp$ 表示无定义的状态,或者更准确地说表示信息为空的状态。随着计算的进行,它包含的信息量逐渐增加,最终确定了特定的程序终态。不难看出,部分函数集合 $\Sigma \rightharpoonup \Sigma$ 和完全函数集合 $\Sigma \to \Sigma_\perp$ 是一一对应的,这种情况下任一完全函数都是连续的。部分函数之间的包含关系对应于 $\Sigma \to \Sigma_\perp$ 中函数之间的"逐点排序"次序关系,即

72

$$f \sqsubseteq g \text{ 当且仅当 } \forall \sigma \in \Sigma. f(\sigma) \sqsubseteq g(\sigma)$$

因为部分函数构成一个完全偏序,因此逐点排序的完全函数的集合 $[\Sigma \to \Sigma_\perp]$ 也构成一个完全偏序。于是,指称语义可以等价地看作将命令指称为完全偏序 $[\Sigma \to \Sigma_\perp]$ 中的连续函数。在讨论 while 程序的指称语义的时候,我们曾经用连续函数的最小不动点求解递归方程,当时我们用部分函数的完全偏序上的连续函数,而现在我们用完全偏序 $[\Sigma \to \Sigma_\perp]$ 上的连续函数。

完全偏序 $[\Sigma \to \Sigma_\perp]$ 和部分函数的完全偏序同构。若一个函数有较多的输入/输出行为,它对应于较多的信息量,而完全偏序的 $\perp$ 包含的信息为空,它对应于没有输入和输出的空函数。函数本身可以看作是用于计算的数据或计算产生的数据。有关这样的函数的信息产生了

离散单元,即输入/输出对。许多从计算建模中产生的完全偏序都具有这种离散性质。我们将看到,可计算函数必定连续,因为可计算函数的输出所包含的每一个信息单元只依赖于输入所包含的有穷个信息单元。否则,如果输入包含的信息单元是无穷的,那么就无法产生输出。这和我们前面的讨论是一致的,如果规则实例是有限的,即它们具有的前提是有限的,那么这些规则实例就确定了一个连续算子。

练习 5.12

(i) 试证明从 Σ 到 $\Sigma_\perp$ 的单调映射是连续的,并且和部分函数 $\Sigma \rightharpoonup \Sigma$ 是一一对应的。证实上述命题:一个部分函数包含在另一个部分函数中当且仅当对应的函数 $\Sigma \to \Sigma_\perp$ 是逐点排序的。

(ii) 令 D 和 E 是完全偏序的,设 D 的每条 ω 链 $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$ 是稳态的,即存在 n 使得对于所有的 $m \geqslant n$ 有 $d_m = d_n$。试证明从 D 到 E 的所有单调函数都是连续的。 □

练习 5.13 试证明:如果放宽条件,允许规则是无限的,因此规则实例有无穷个前提,则不能保证一组规则实例所确定的算子是连续的。 □ 73

5.5 克纳斯特-塔尔斯基定理

本节讨论最小不动点的另一个抽象特征。读者可以先略过本节的内容,讨论的结论只是在后面才用到。回顾上一章,我们介绍了集合算子的最小不动点的一个特征。在 4.1 节的练习 4.3 中,曾要求证明:对一组规则实例 R,

$$I_R = \bigcap \{Q \mid Q \text{ 是 } R \text{ 封闭的}\}$$

根据 4.4 节的内容,这个问题也可以表述为

$$fix(\hat{R}) = \bigcap \{Q \mid \hat{R}(Q) \subseteq Q\}$$

即算子 $\hat{R}$ 的最小不动点可以用它的前缀不动点的交集来刻画。这就是*克纳斯特-塔尔斯基定理*的特殊情况,该定理给出了最小不动点存在的一般结果。正如所期望的,定理涉及了将集合上的交运算推广到一个对应于偏序的最小上界的概念。

定义 对偏序 $(P, \sqsubseteq)$ 和子集 $X \subseteq P$ 称 p 是 X 的下界当且仅当

$$\forall q \in X.\, p \sqsubseteq q$$

称 p 是 X 的*最大下界* (glb) 当且仅当

(i) p 是 X 的下界,

(ii) 对 X 的所有下界 q,有 $q \sqsubseteq p$。

如果一个偏序的子集 X 有最大下界,我们把它记为 $\bigsqcap X$,把 $\bigsqcap \{d_0, d_1\}$ 记为 $d_0 \sqcap d_1$。

最小上界有时又称为*上确界* (或 sups),同样地最大下界又称为*下确界* (或 infs)。

定义 如果一个偏序的任意子集都有最大下界,则称这个偏序为*完全格*。

虽然这里完全格用含最大下界的偏序来定义,但它也可以用含最小上界的偏序来定义,见

下面的练习。

练习 5.14 试证明完全格的任意子集也必有最小上界。推导:如果$(L,\sqsubseteq)$是完全格,则
[74] 对$\sqsubseteq$的逆关系,$(L,\sqsupseteq)$也是完全格。 □

定理 5.15 (克纳斯特-塔尔斯基最小不动点定理) 令$(L,\sqsubseteq)$是完全格,$f:L\to L$是单调函数,即若$x\sqsubseteq y$则$f(x)\sqsubseteq f(y)$(但f不一定连续)。定义

$$m=\bigsqcap\{x\in L\mid f(x)\sqsubseteq x\}$$

则m是f的不动点和f的最小前缀不动点。

证明 记$X=\{x\in L\mid f(x)\sqsubseteq x\}$。由上可知,定义$m=\bigsqcap X$。设$x\in X$,显然$m\sqsubseteq x$,由$f$的单调性得$f(m)\sqsubseteq f(x)$,又由$x\in X$得$f(x)\sqsubseteq x$,这样,对于任何$x\in X$,都有$f(m)\sqsubseteq x$。于是$f(m)\sqsubseteq\bigsqcap X=m$。这里的$m$就是前缀不动点,并且由定义知,它还是最小不动点。由f的单调性,从$f(m)\sqsubseteq m$得$f(f(m))\sqsubseteq f(m)$,于是$f(m)\in X$,得$m\sqsubseteq f(m)$,从而$f(m)=m$。这样我们就证明了m确实是f的不动点和f的最小前缀不动点。 □

作为该定理的推论,下面我们证明完全格上的单调函数有一个最大不动点。

定理 5.16 (克纳斯特-塔尔斯基最大不动点定理) 令$(L,\sqsubseteq)$是完全格,$f:L\to L$是单调函数。定义

$$M=\bigsqcup\{x\in L\mid x\sqsubseteq f(x)\}$$

则M是f的不动点和f的最大后缀不动点。(后缀不动点是满足$x\sqsubseteq f(x)$的元素x。)

证明 证明过程与定理 5.15 完全一样,只要注意到完全格$(L,\sqsubseteq)$上的单调函数在完全格$(L,\sqsupseteq)$上仍然是单调的。 □

克纳斯特-塔尔斯基定理的重要性在于,它能应用到完全格上的任一单调函数。然而,大多数情况下,我们更关心连续函数的最小不动点,根据上一节的方法,通过完全偏序上的ω链的最小上界构造出来。

5.6 进一步阅读资料

本章给出了指称语义的一个例子。后面的几章将继续讨论指称方法的研究范围和作用。
[75] 如果读者要进一步了解指称语义,可以参考下列著作:Bird 的[21],Loeckx 和 Sieber 的[58],Schmidt 的[88]和 Stoy 的[95](虽然[95]基于完全格而不是完全偏序)。更深入详尽的讨论请参考 de Bakker 的[13]。**IMP** 的指称语义用最小不动点来代替规则,因而描述较为抽象,但是由于引入完全偏序和连续函数的概念,**IMP** 程序就有了坚实的数学基础。文献[69]给出了
[76] while 程序语言的几个应用例子。

第6章 IMP的公理语义

本章讨论 **IMP** 程序的系统验证工作。我们将介绍证明程序的部分正确性的霍尔（Hoare）规则，并证明霍尔规则是可靠的。还介绍将布尔表达式扩充成描述程序状态的断言的语言。本章最后给出使用霍尔规则框架的一个验证示例。

6.1 基本思想

如何证明用 **IMP** 编写的程序实现了所需要的功能呢？

我们举一个简单的程序例子，它计算前 100 个自然数之和，即计算 $\Sigma_{1\leqslant m\leqslant 100} m$（记号 $\Sigma_{1\leqslant m\leqslant 100} m$ 表示 $1+2+\cdots+100$），**IMP** 程序为：

$$
\begin{aligned}
&S := 0;\\
&N := 1;\\
&(\textbf{while } \neg(N = 101) \textbf{ do } S := S + N; N := N + 1)
\end{aligned}
$$

如何证明当该程序运行结束的时候使得 S 的值为 $\Sigma_{1\leqslant m\leqslant 100} m$ 呢？

当然，一个方法是按照我们的操作语义去运行这个程序，且观察运行的结果。还有一个方法，我们稍微改动一下程序，把“**while** $\neg(N=101)$ **do**…”改为“**while** $\neg(N=P+1)$ **do**…”，并设想在程序开始之前给 P 任意赋一个值。这种情况下，无论 P 取何值，程序执行完之后 S 的值都应该是 $\Sigma_{1\leqslant m\leqslant p} m$。由于 P 能取无穷多个值，因此，不可能简单地通过对 P 的所有初始值运行该程序而验证这一事实。我们需要清晰的、抽象的、逻辑的方法来论证该程序。

最后，我们得到一个关于 **IMP** 程序性质的形式化证明系统，它建立在 **IMP** 的每个程序结构的证明规则上，这些规则称为霍尔规则或者费洛伊德-霍尔规则。历史上，费洛伊德（R. W. Floyd）提出了论证程序流图的规则，而霍尔（C. A. R. Hoare）修改扩充了这些规则，用以处理像 **IMP** 这样的但包含过程的语言。起初，人们十分提倡他们的方法，因为它不仅能够证明程序的性质，而且还给出了解释程序结构含义的一种方法。我们可以用“公理”（更精确地说是规则）来说明结构的含义，并说明如何证明它的性质，因此习惯上把这种方法称为公理语义。

现在，我们看一下上面这个程序的非形式化推导，即从直观上去理解程序的运行过程。命令 $S := 0; N := 1$ 初始化存储单元中的值，所以可以给程序加一个注释： 77

$$
\begin{aligned}
&S := 0; N := 1\\
&\{S = 0 \wedge N = 1\}\\
&(\textbf{while } \neg(N = 101) \textbf{ do } S := S + N; N := N + 1)
\end{aligned}
$$

注释中 $\{S=0\wedge N=1\}$ 就像处理布尔表达式那样，其中 $S=0$ 理解为存储单元 S 的值是 0。最终把程序注释为：

$$S := 0; N := 1$$
$$\{S = 0 \wedge N = 1\}$$
$$(\textbf{while } \neg(N = 101) \textbf{ do } S := S + N; N := N + 1)$$
$$\{S = \sum_{1 \leq m \leq 100} m\}$$

它的含义是，如果在执行 while 循环前 $S=0 \wedge N=1$，则执行程序后 $S=\Sigma_{1 \leq m \leq 100} m$。

现在看布尔表达式，在执行完 while 循环之后，N 必须为 101，因为如果 N 不为 101，则 while 循环还要继续运行。所以，在循环结束之后，我们得到 $N=101$。但是，我们想知道的是 S 的值而不是 N 的值！

当然，通过这个简单的程序，我们可以看到，第一次循环执行后 S 和 N 的值为 $S=1, N=2$。第二次循环执行后，S 和 N 的值为 $S=1+2, N=3, \cdots$，以此类推，我们得到一个模式：第 i 次循环执行后，$S=1+2+\cdots+i$ 且 $N=i+1$。于是，当退出循环时，因为 $N=101$，所以 $S=1+2+\cdots+100$。

在循环的每次迭代的开始和结束时都有

$$S = 1 + 2 + 3 + \cdots + (N - 1) \tag{I}$$

它描述了存储单元 S 的值和存储单元 N 的值之间的最主要的关系。断言 I 称为 while 循环的不变式，因为在循环的每次迭代中它都保持为真，一直到最后循环终止。我们以后会进一步讨论不变式。

现在，我们用基于如下形式的断言建立证明系统：

78

$$\{A\}c\{B\}$$

其中 A 和 B 是像布尔表达式 **Bexp** 中那样的断言，c 是一条命令。下面给出这样一个复合断言的精确解释：

> 对于所有满足断言 A 的状态 σ，如果从状态 σ 下执行 c 后程序终止于状态 σ'，那么 σ' 满足断言 B。

换句话说，$\{A\}c\{B\}$ 意味着从满足 A 的一个状态开始，c 的成功执行结束于满足 B 的一个状态。断言 A 称为部分正确性断言 $\{A\}c\{B\}$ 的前置条件，断言 B 称为后置条件。

形如 $\{A\}c\{B\}$ 的断言称为部分正确性断言，因为该断言不涉及命令 c 执行不终止的情形。举一个例子：

$$c \equiv \textbf{while true do skip}$$

从任一状态开始，c 的执行都不会终止。按照我们上面给出的解释，下面的部分正确性断言是有效的：

$$\{\textbf{true}\}c\{\textbf{false}\}$$

原因很简单，因为 c 的执行不会终止。更一般地，由于 c 是一个循环，任意部分正确性断言 $\{A\}c\{B\}$ 都是有效的。与之相对，把完全正确性断言写成

$$[A]c[B]$$

它表示从满足 A 的任一状态开始，c 的执行终止于满足 B 的一个状态。本书对完全正确性断言不作过多讨论。

注意 程序的部分正确性和完全正确性可以由多种符号系统来表示，读者在阅读这方面的著作时，一定要明确书中采用了哪一种符号系统。

上面的讨论尚不够严密，我们没有说明在部分正确性断言 $\{A\}c\{B\}$ 中，A 与 B 应满足何种条件？稍后，我们将从更一般的角度来详细讨论这个问题。

下面讨论一个比较实用的记号，从概念上讲也可以把这个记号视为部分正确性断言的语义（参照 7.5 节关于采用谓词转换器的指称语义的介绍）。首先引入一个缩写来表示状态 σ 满足断言 A，或者等价地表示断言 A 在状态 σ 下为真，记作

$$\sigma \models A$$

79

虽然这个记号的含义很直观，但仍然需要定义。请回忆一下我们是如何解释部分正确性断言 $\{A\}c\{B\}$ 的。当命令 c 指称一个从初态到终态的部分函数时，部分正确性断言意味着：

$$\forall \sigma.\ (\sigma \models A\ \&\ \mathscr{C}[\![c]\!]\sigma \text{ 有定义}) \Rightarrow \mathscr{C}[\![c]\!]\sigma \models B$$

在这个定义中，我们以 $\mathscr{C}[\![c]\!]\sigma$ 有定义作为前提，这是麻烦的事情。第 5 章关于 **IMP** 的指称语义的讨论中，我们用符号 $\perp$ 表示一个无定义的状态（更严格地说，是状态空信息）。这样，每当 $\mathscr{C}[\![c]\!]\sigma$ 无定义就记作 $\mathscr{C}[\![c]\!]\sigma = \perp$。为了和部分函数保持一致，规定 $\mathscr{C}[\![c]\!]\perp = \perp$。如果我们约定 $\perp$ 满足所有的断言，则部分正确性断言可用更简单的记号表示。考虑到对于所有的断言 A，有

$$\perp \models A$$

我们可以以下式来描述 $\{A\}c\{B\}$ 的含义：

$$\forall \sigma \in \Sigma.\ \sigma \models A \Rightarrow \mathscr{C}[\![c]\!]\sigma \models B$$

因为上面的约定和我们对部分正确性含义的解释是一致的。很明显，不终止的计算指称到 $\perp$ 并且满足任一后置条件。

6.2 断言语言 Assn

什么样的断言比较适合描述 **IMP** 程序呢？因为我们要论证布尔表达式，所以要包含 **Bexp** 的所有断言。我们希望用量词“$\forall i \cdots$”和“$\exists i \cdots$”表述断言，所以要扩充 **Bexp** 和 **Aexp**，使之包含量化的整型变量 i。例如，如果整数 k 是另一个整数 l 的倍数，则记为

$$\exists i.\ k = i \times l$$

我们将会阐述如何在特定的断言语言 **Assn** 中引入整型变量和量词。**Assn** 具有很强的表达能力，原则上可以用 **Assn** 描述任何断言。不过在示例和习题中，我们还是用不同的方式对 **Assn** 进行了扩充。（例如，在一个例子中我们用 $n! = n \times (n-1) \times \cdots \times 2 \times 1$ 表示阶乘函数。）

80

首先，扩充 **Aexp**，使之包括整型变量 $i, j, k, \cdots$，这只要添加规则以扩充 **Aexp** 的 BNF 描述即可，添加的规则将整型变量 $i, j, k, \cdots$ 视为整数表达式，因此，扩充的算术表达式的语法范畴

Aexpv 由下式给出：

$$a ::= n \mid X \mid i \mid a_0 + a_1 \mid a_0 - a_1 \mid a_0 \times a_1$$

其中，n 属于整数集 **N**，X 属于存储单元集 **Loc**，i 属于整型变量集 **Intvar**。

然后，扩充布尔表达式，使之包括更一般的算术表达式、量词和蕴涵关系。这些规则为：

$$A ::= \mathbf{true} \mid \mathbf{false} \mid a_0 = a_1 \mid a_0 \leqslant a_1 \mid A_0 \wedge A_1 \mid A_0 \vee A_1 \mid \neg A \mid A_0 \Rightarrow A_1 \mid \forall i.A \mid \exists i.A$$

我们称扩充的布尔断言的集合为断言语言 **Assn**。

以前读者已经接触过表达式的计算，不过有些符号（例如量词）那时没有介绍，所以不能写出更简洁的表达式。以前，整型变量 i 作为一个任意的整数来理解并像未知数 $x,y,\cdots$ 那样进行计算。像 $A_0 \Rightarrow A_1$ 那样的蕴涵关系被理解为：如果 A_0 则 A_1，且当 A_0 为假或者 A_1 为真时取真。以前我们在数学中使用过蕴涵的概念，现在我们在形式断言语言 **Assn** 中引入蕴涵，并约定以后的表达式与断言都是指扩充后的 **Assn**，在练习中也是如此。然而，因为我们想论证证明系统是基于断言的（不仅仅用例子来描述），所以我们应该用更形式化的方法，给出具有整型变量的表达式和断言的含义的理论，它也是谓词演算的一部分。

6.2.1 自由变量与约束变量

如果断言中的 i 在前束谓词 $\forall i$ 或 $\exists i$ 的作用域内，则称整型变量 i 是*约束变量*，否则称 i 为*自由变量*。例如，在断言

$$\exists i.k = i \times l$$

中，整型变量 i 是约束变量，而 k 和 l 是自由变量，这里 k 和 l 表示已知的或未知的特定数。在同一个断言中相同的整型变量可能有不同的出现，一个出现是自由的，而另一个出现却是约束的。例如，在断言

81

$$(i + 100 \leqslant 77) \wedge (\forall i.j + 1 = i + 3)$$

中，i 的第一个出现是自由的，而第二个出现是约束的，j 只出现一次，是自由的。

虽然这种非形式化的解释可能够了，但下面还是用结构归纳法给出它们的形式化定义。首先，用结构归纳法定义扩充了整型变量的算术表达式 $a \in$ **Aexpv** 的自由变量集合 $\mathrm{FV}(a)$：对所有的 $n \in \mathbf{N}, X \in \mathbf{Loc}, i \in \mathbf{Intvar}$ 以及 $a_0, a_1 \in \mathbf{Aexpv}$，有

$$\begin{aligned}
&\mathrm{FV}(n) = \mathrm{FV}(X) = \emptyset \\
&\mathrm{FV}(i) = \{i\} \\
&\mathrm{FV}(a_0 + a_1) = \mathrm{FV}(a_0 - a_1) = \mathrm{FV}(a_0 \times a_1) = \mathrm{FV}(a_0) \cup \mathrm{FV}(a_1)
\end{aligned}$$

然后，对所有的 $a_0, a_1 \in$ **Aexpv**，整型变量 i 和断言 A_0, A_1, A，用结构归纳法定义断言 A 的自由变量集合 $\mathrm{FV}(A)$ 为

$$\begin{aligned}
&\mathrm{FV}(\mathbf{true}) = \mathrm{FV}(\mathbf{false}) = \emptyset \\
&\mathrm{FV}(a_0 = a_1) = \mathrm{FV}(a_0 \leqslant a_1) = \mathrm{FV}(a_0) \cup \mathrm{FV}(a_1) \\
&\mathrm{FV}(A_0 \wedge A_1) = \mathrm{FV}(A_0 \vee A_1) = \mathrm{FV}(A_0 \Rightarrow A_1) = \mathrm{FV}(A_0) \cup \mathrm{FV}(A_1)
\end{aligned}$$

$$\mathrm{FV}(\neg A) = \mathrm{FV}(A)$$
$$\mathrm{FV}(\forall i.A) = \mathrm{FV}(\exists i.A) = \mathrm{FV}(A)\backslash\{i\}$$

于是我们有了自由变量的精确定义。如果出现在断言 A 中的任一变量不是自由变量,那么它就是约束变量。我们称不含自由变量的断言是封闭的。

6.2.2 代入

断言 A 可以刻画为:

$$---i---i--$$

其中 i 为自由出现的整型变量。设 a 是一个算术表达式,为了简便不妨设 a 不含整型变量。于是

$$A[a/i] \equiv ---a---a--$$

是 A 中用 a 代入 i 的结果。如果 a 包含整型变量,那么为了避免 a 的变量变成被 A 中量词约束的约束变量,需要对 A 的某些约束变量换名。这就是为什么要有一般的代入方法的原因。 [82]

我们举几个简单的例子较精确地说明代入。设 i 是一个整型变量,a 是一个不含整型变量的算术表达式。首先用以下结构归纳法定义算术表达式中的代入:

$$n[a/i] \equiv n \qquad X[a/i] \equiv X$$
$$j[a/i] \equiv j \qquad i[a/i] \equiv a$$
$$(a_0 + a_1)[a/i] \equiv (a_0[a/i] + a_1[a/i])$$
$$(a_0 - a_1)[a/i] \equiv (a_0[a/i] - a_1[a/i])$$
$$(a_0 \times a_1)[a/i] \equiv (a_0[a/i] \times a_1[a/i])$$

其中 n 是整数,X 是存储单元,j 是不等于 i 的整型变量,$a_0, a_1 \in$ **Aexpv**。现在用结构归纳法定义 a 对断言中 i 的代入(此处 a 不含自由变量,所以不必担心 a 的变量会变成约束变量):

$$\mathbf{true}[a/i] \equiv \mathbf{true} \qquad \mathbf{false}[a/i] \equiv \mathbf{false}$$
$$(a_0 = a_1)[a/i] \equiv (a_0[a/i] = a_1[a/i])$$
$$(a_0 \leqslant a_1)[a/i] \equiv (a_0[a/i] \leqslant a_1[a/i])$$
$$(A_0 \wedge A_1)[a/i] \equiv (A_0[a/i] \wedge A_1[a/i])$$
$$(A_0 \vee A_1)[a/i] \equiv (A_0[a/i] \vee A_1[a/i])$$
$$(\neg A)[a/i] \equiv \neg(A[a/i])$$
$$(A_0 \Rightarrow A_1)[a/i] \equiv (A_0[a/i] \Rightarrow A_1[a/i])$$
$$(\forall j.A)[a/i] \equiv \forall j.(A[a/i])$$
$$(\forall i.A)[a/i] \equiv \forall i.A$$
$$(\exists j.A)[a/i] \equiv \exists j.(A[a/i])$$
$$(\exists i.A)[a/i] \equiv \exists i.A$$

其中,$a_0, a_1 \in$ **Aexpv**,A_0, A_1, A 是断言,j 是不等于 i 的整型变量。

正如前面所提到的,如果 a 含有自由变量,则对代入 $A[a/i]$ 的定义有些复杂,因为还要对约束变量换名。幸好,目前我们不需要这种更复杂的代入定义。

用同样的方法,我们可以表示存储单元 X 的代入,如果断言 $A \equiv ---X--$,用 a 代入 X,则 $A[a/X] \equiv ---a--$。请读者给出这个代入的形式化定义。

练习 6.1 试写一个含有自由整型变量 i 的表示素数的断言 $A \in \mathbf{Assn}$,即要求 $\sigma \models^I A$ 当且仅当 $I(i)$ 是一个素数。 □

83 **练习 6.2** 试定义一个含有自由变量 i,j 和 k 的公式 $LCM \in \mathbf{Assn}$(它表示"i 是 j 和 k 的最小公倍数"),即需要 $\sigma \models^I LCM$ 当且仅当 $I(i)$ 是 $I(j)$ 和 $I(k)$ 的最小公倍数。

(提示:两个数的最小公倍数是能被这两个数都整除的最小非负整数。) □

6.3 断言的语义

由于算术表达式扩充后包含整型变量,所以我们不能用前面的语义函数 $\mathscr{A}$ 精确描述这些新表达式的值。这里我们首先必须把整型变量解释为特定的整数,为此我们引入解释的概念。

所谓解释是指一个函数,它将一个整数值赋给整型变量,记作 I: **Intvar**→**N**。

表达式 **Aexpv** 的含义

现在,我们定义一个语义函数 $\mathscr{A}v$,在特定的状态和特定的解释下,它将值和有整型变量的算术表达式相关联。在解释 I 和状态 σ 下,算术表达式 $a \in \mathbf{Aexpv}$ 的值记为 $\mathscr{A}v[\![a]\!]I\sigma$ 或者等价地记为 $(\mathscr{A}v[\![a]\!](I))(\sigma)$。用结构归纳法定义如下:

$$
\begin{aligned}
\mathscr{A}v[\![n]\!]I\sigma &= n \\
\mathscr{A}v[\![X]\!]I\sigma &= \sigma(X) \\
\mathscr{A}v[\![i]\!]I\sigma &= I(i) \\
\mathscr{A}v[\![a_0 + a_1]\!]I\sigma &= \mathscr{A}v[\![a_0]\!]I\sigma + \mathscr{A}v[\![a_1]\!]I\sigma \\
\mathscr{A}v[\![a_0 - a_1]\!]I\sigma &= \mathscr{A}v[\![a_0]\!]I\sigma - \mathscr{A}v[\![a_1]\!]I\sigma \\
\mathscr{A}v[\![a_0 \times a_1]\!]I\sigma &= \mathscr{A}v[\![a_0]\!]I\sigma \times \mathscr{A}v[\![a_1]\!]I\sigma
\end{aligned}
$$

有整型变量的算术表达式的语义定义扩充了第 5 章无整型变量的算术表达式的指称语义。

命题 6.3 *对所有无整型变量的 $a \in \mathbf{Aexp}$,所有的状态 σ 和所有的解释 I,有*

$$\mathscr{A}[\![a]\!]\sigma = \mathscr{A}v[\![a]\!]I\sigma$$

84 **证明** 施结构归纳于算术表达式。这个证明是一个简单的练习,由读者完成。 □

断言 **Assn** 的含义

由于引入了整型变量,所以语义函数需要一个解释函数作为它的参数,解释函数的作用只是提供 **N** 中的一个值给整型变量。

记号 我们用记号 $I[n/i]$ 表示从解释 I 中将整型变量 i 的值改变成 n 而得到的解释,即

$$I[n/i](j) = \begin{cases} n & j \equiv i \\ I(j) & \text{其他} \end{cases}$$

我们已经给出了具有整型变量的算术表达式的含义，用同样的方法，可以确定断言 **Assn** 的含义。但现在我们将解释和状态作为语义函数的参数，把语义函数作为一个从断言到返回真值的函数。我们选择另一种等价的表示方法，给定解释 I，我们直接定义满足断言的那些状态。

事实上，可以很方便地把状态集合 Σ 扩充为 $\Sigma_\perp$，它包含了与计算不终止相关的值 $\perp$。因此，$\Sigma_\perp =_{def} \Sigma \cup \{\perp\}$。新加进去的元素 $\perp$ 表示一个不终止的计算。对断言 $A \in \mathbf{Assn}$，状态 $\sigma \in \Sigma$ 和解释 I，用结构归纳法定义

$$\sigma \models^I A$$

我们将它扩充，定义 $\perp \models^I A$。关系 $\sigma \models^I A$ 表示在解释 I 下状态 σ 满足 A，或者说，在解释 I 下，当状态为 σ 时断言 A 为真。对于解释 I 和所有的状态 $\sigma \in \Sigma$，施结构归纳于断言，我们定义：

$$\begin{aligned}
&\sigma \models^I \mathbf{true}\\
&\sigma \models^I (a_0 = a_1) \text{ 如果 } \mathscr{A}v[\![a_0]\!]I\sigma = \mathscr{A}v[\![a_1]\!]I\sigma\\
&\sigma \models^I (a_0 \leqslant a_1) \text{ 如果 } \mathscr{A}v[\![a_0]\!]I\sigma \leqslant \mathscr{A}v[\![a_1]\!]I\sigma\\
&\sigma \models^I A \wedge B \text{ 如果 } \sigma \models^I A \text{ 且 } \sigma \models^I B\\
&\sigma \models^I A \vee B \text{ 如果 } \sigma \models^I A \text{ 或 } \sigma \models^I B\\
&\sigma \models^I \neg A \text{ 如果非 } \sigma \models^I A\\
&\sigma \models^I A \Rightarrow B \text{ 如果（非 } \sigma \models^I A\text{）或 } \sigma \models^I B\\
&\sigma \models^I \forall i.A \text{ 如果对于所有的 } n \in \mathbf{N}, \sigma \models^{I[n/i]} A\\
&\sigma \models^I \exists i.A \text{ 如果对于某个 } n \in \mathbf{N}, \sigma \models^{I[n/i]} A\\
&\perp \models^I A
\end{aligned}$$

85

注意，非 $\sigma \models^I A$ 通常记作 $\sigma \not\models^I A$。

上述讨论形式化地告诉我们，一旦用某个解释给整型变量赋值时，断言在某一状态下为真意味着什么。布尔表达式的语义提供了另一种方法，它说明了这类断言在一个状态下为真或者为假的含义。这两种情况其实是一致的。

命题 6.4 对 $b \in \mathbf{Bexp}, \sigma \in \Sigma$ 和任一解释 I，有

$$\mathscr{B}[\![b]\!]\sigma = \mathbf{true} \text{ 当且仅当 } \sigma \models^I b, \text{且}$$
$$\mathscr{B}[\![b]\!]\sigma = \mathbf{false} \text{ 当且仅当 } \sigma \not\models^I b$$

证明 施结构归纳于布尔表达式，并利用命题 6.3 来证明。 □

练习 6.5 给出命题 6.4 的证明。 □

练习 6.6 施结构归纳于算术表达式 $a \in \mathbf{Aexpv}$，试证明：

$$\mathscr{A}v[\![a]\!]I[n/i]\sigma = \mathscr{A}v[\![a[n/i]]\!]I\sigma$$

（注意，左边的 n 作为 $\mathbf{N}$ 的元素出现，而右边的 n 作为 $\mathbf{N}$ 中对应的数出现。）

根据这个结论，证明：

$$\sigma \models^I \forall i.A \text{ 当且仅当 } \sigma \models^I A[n/i] \text{ 对于所有的 } n \in \mathbf{N} \text{ 成立，且}$$

$$\sigma \models^I \exists i.A \text{ 当且仅当 } \sigma \models^I A[n/i] \text{ 对于某个 } n \in \mathbf{N} \text{ 成立}$$ □

断言的扩充

设 I 是一个解释，当讨论断言和部分正确性断言的性质时，对应于解释 I，考虑对断言进行的扩充（即断言为真的状态的集合）是有用的。

对应于解释 I，断言 A 的扩充定义为

86

$$A^I = \{\sigma \in \Sigma_\perp \mid \sigma \models^I A\}$$

部分正确性断言

部分正确性断言形如

$$\{A\}c\{B\}$$

其中 $A, B \in \mathbf{Assn}, c \in \mathbf{Com}$。注意部分正确性断言并不属于 **Assn**。

设 I 是一个解释，$\sigma \in \Sigma_\perp$，对应于解释 I，我们定义状态和部分正确性断言之间的满足关系如下：

$$\sigma \models^I \{A\}c\{B\} \text{ 当且仅当 } (\sigma \models^I A \Rightarrow \mathscr{C}[c]\sigma \models^I B)$$

换言之，状态 σ 满足对应于解释 I 的部分正确性断言，当且仅当从状态 σ 下 c 的任何成功的计算都终止于满足 B 的状态。

有效性

设 I 是一个解释，考察 $\{A\}c\{B\}$。我们不太关心某一特定状态下这个部分正确性断言为真，我们更关心在所有状态下它是否为真，即

$$\forall \sigma \in \Sigma_\perp . \sigma \models^I \{A\}c\{B\}$$

也可以记作

$$\models^I \{A\}c\{B\}$$

它表示对应于解释 I，部分正确性断言是有效的，因为不管在什么状态下，$\{A\}c\{B\}$ 都为真。进一步，考虑在例子

$$\{i < X\}X := X + 1\{i < X\}$$

中，我们不太关心解释 I 对 i 赋于特定值，我们更关心在所有状态下，对所有的解释 I，它是否为真。为此引入有效性的概念，定义

$$\models \{A\}c\{B\}$$

表示对于所有的解释 I 和所有的状态 σ，有

$$\sigma \models^I \{A\}c\{B\}$$

87 如果 $\models \{A\}c\{B\}$ 成立，则称部分正确性断言 $\{A\}c\{B\}$ 是有效的。

类似地，对于任一断言 A，记作 $\models A$ 当且仅当对于所有的解释 I 和所有的状态 σ 有 σ

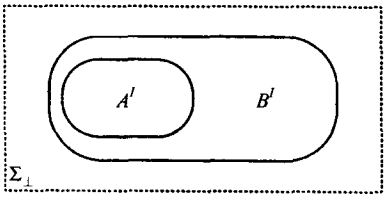
$\models^I A$。于是,称断言 A 是有效的。

注意 此处的有效性和谓词演算或逻辑程序设计标准课程中的有效性在概念上有所不同。后者所指的有效性更为广泛,称断言是有效的当且仅当对于运算符 +, ×…, 数字 0,1,…以及自由变量的所有解释,断言始终为真。我们关心的不是一般意义上任意的解释,因为 IMP 程序在基于存储单元的状态下进行操作,而状态依赖的存储单元的内容仅是整数和整数的运算。为了与一般意义上的有效性相区别,这里我们称之为算术有效性,省去"算术"二字,简称有效性。

例 设$\models(A \Rightarrow B)$,则对于任意的解释 I,有

$$\forall \sigma \in \Sigma_{\perp}((\sigma \models^I A) \Rightarrow (\sigma \models^I B))$$

即 $A^I \subseteq B^I$,如图:



所以,$\models(A \Rightarrow B)$当且仅当对于所有的解释 I,所有满足 A 的状态同时也满足 B。 □

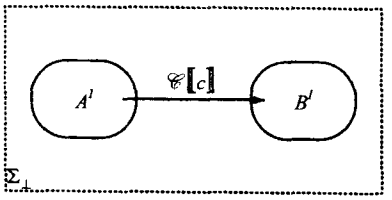
例 设$\models\{A\}c\{B\}$,则对于任意的解释 I,有

$$\forall \sigma \in \Sigma_{\perp}((\sigma \models^I A) \Rightarrow (\mathscr{C}[\![c]\!]\sigma \models^I B))$$

即 A 在 $\mathscr{C}[\![c]\!]$下的象包含于 B 中,也即

$$\mathscr{C}[\![c]\!]A^I \subseteq B^I$$

如图:



88

因此,$\models\{A\}c\{B\}$当且仅当对于所有的解释 I,如果在满足 A 的状态下执行 c,并且它的执行终止于一个状态,则那个终止状态将满足 B。[⊖] □

练习 6.7 在前面的练习中,要求写一个含有一个自由整型变量 i 的表示素数的断言 $A \in$ Assn。根据状态和断言之间的满足关系$\models^I$ 的定义,试推导$\models^I A$ 当且仅当 $I(i)$ 的确是一个素数。 □

⊖ 该图错误地暗示,断言 A^I 和 B^I 的扩充是不相交的;事实上,它们两者总是包含 $\perp$,还可能包含其他相同的状态。

6.4 部分正确性的证明规则

下面我们介绍产生有效的部分正确性断言的证明规则，这些证明规则是语法制导的，它们把证明一条复合命令的部分正确性断言简化成证明它的直接子命令的部分正确性断言。人们通常把这组证明规则称为霍尔规则，又把这组规则组成的证明系统称为霍尔逻辑。

skip 规则：

$$\{A\}\ \mathbf{skip}\ \{A\}$$

赋值规则：

$$\{B[a/X]\}\ X := a\ \{B\}$$

顺序复合规则：

$$\frac{\{A\}\ c_0\ \{C\} \quad \{C\}\ c_1\ \{B\}}{\{A\}\ c_0;c_1\ \{B\}}$$

条件规则：

$$\frac{\{A \wedge b\}\ c_0\ \{B\} \quad \{A \wedge \neg b\}\ c_1\ \{B\}}{\{A\}\ \mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1\ \{B\}}$$

while 循环规则：

$$\frac{\{A \wedge b\}\ c\ \{A\}}{\{A\}\ \mathbf{while}\ b\ \mathbf{do}\ c\ \{A \wedge \neg b\}}$$

推论规则：

$$\frac{\models (A \Rightarrow A') \quad \{A'\}\ c\ \{B'\} \quad \models (B' \Rightarrow B)}{\{A\}\ c\ \{B\}}$$

89

有了霍尔规则就存在霍尔规则的推导，从这个意义上讲霍尔规则就是一个证明系统，其中的推导称为证明，而推导的结论称为定理。如果$\{A\}\ c\ \{B\}$是一条定理，则记为$\vdash\{A\}\ c\ \{B\}$。

除了赋值规则和 while 循环规则之外，其余的规则都很容易理解。如果在 **skip** 执行之前的状态下断言为真，那么在 **skip** 执行之后的断言仍为真，因为该状态没有发生改变。这就是 **skip** 规则的内容。

赋值语句有些复杂，不过可以通过一个例子来说明。例如，对于简单赋值语句 $X := X + 3$，可以通过考察断言 $X = 3$ 来理解。

顺序复合规则表示：如果$\{A\}\ c_0\ \{C\}$和$\{C\}\ c_1\ \{B\}$是有效的，则$\{A\}\ c_0;c_1\ \{B\}$也是有效的，即如果在满足 A 的状态下，c_0 的成功执行终止于满足 C 的状态，且在满足 C 的状态下，c_1 的成功执行终止于满足 B 的状态，则在满足状态 A 的状态下，任何 c_0 和 c_1 的相继成功执行，也终止于满足 B 的状态。

条件规则的两个前提分别与条件命令的两个分支对应。

while 循环 **while** b **do** c 规则中断言 A 称为不变式，因为它的前提“$\{A \wedge b\}\ c\ \{A\}$是有效的”，表示循环体一次完全的执行后断言 A 保持为真，且 while 循环中这样的执行只发生在满足 b 的状态下。从满足 A 的状态开始，while 循环的执行要么不终止，要么循环体有限次被执行，且循环的每一次执行都从满足 b 的状态开始。如果循环执行的次数是有限的，由于 A 是不变式，所以终态要满足 A 且退出循环时也满足 $\neg b$。

推论规则有些特殊，因为它的前提包括有效的蕴涵。推论规则的每一个实例都有形如 $\models(A \Rightarrow A')$ 和 $\models(B' \Rightarrow B)$ 的前提，因此在证明中应用推论规则以得到它的一个实例首先要依赖于证明断言 $(A \Rightarrow A')$ 和 $(B' \Rightarrow B)$ 是有效的。而一般来说这是非常困难的，因为这种蕴涵关系能表示复杂的算术运算。幸好，因为程序通常不会涉及高深的数学运算，所以上述有效性的证明通常只需要用到初等数学的知识。 90

6.5 可靠性

对于霍尔规则，我们考虑逻辑系统的两个普遍性质——可靠性和完备性。

可靠性 如果假设某一条规则的前提是有效的，那么它的结论也是有效的，即规则的有效性能够保持，这时我们称这条规则是可靠的。如果一个证明系统的每条规则都是可靠的，则称这个证明系统本身也是可靠的。根据规则归纳法原理可知，从霍尔规则证明系统中得到的每条定理都是有效的部分正确性断言。（规则后的注释是其可靠性的非形式的叙述。）

完备性 当然，我们希望证明系统足够强大，使得所有有效的部分正确性断言都能作为定理。从这个意义上讲，我们希望这个证明系统是完备的。（关于完备性的细节放在下一章讨论。）

规则的可靠性证明要用到代入的有关结论。

引理 6.8 设 I 是一个解释，$a, a_0 \in \mathbf{Aexpv}, X \in \mathbf{Loc}$，则对于所有的解释 I 和状态 σ，有

$$\mathscr{A}v[a_0[a/X]]I\sigma = \mathscr{A}v[a_0]I\sigma[\mathscr{A}v[a]I\sigma/X]$$

证明 通过施结构归纳于 a_0 来证明，留作练习。 □

引理 6.9 设 I 是一个解释，$B \in \mathbf{Assn}, X \in \mathbf{Loc}, a \in \mathbf{Aexp}$，则对于所有的状态 $\sigma \in \Sigma$，有

$$\sigma \models^I B[a/X] \text{ 当且仅当 } \sigma[\mathscr{A}[a]\sigma/X] \models^I B$$

证明 通过施结构归纳于 B 来证明，留作练习。 □

练习 6.10 试证明上面两个引理。 □

定理 6.11 设 $\{A\}c\{B\}$ 是部分正确性断言，如果 $\vdash\{A\}c\{B\}$，则 $\models\{A\}c\{B\}$。

证明 显然，如果我们能够证明每一条规则都是可靠的（即如果规则的前提由有效的断言和部分正确性断言所组成，则它的结论也是有效的，从这个意义上讲，每条规则保持有效性），那么由规则归纳法可知，每一条定理都是有效的。

skip 规则：显然 $\models\{A\}\mathbf{skip}\{A\}$ 成立，因此 **skip** 规则是可靠的。 91

赋值规则：设 $c \equiv (X := a)$，设 I 是一个解释，根据引理 6.9，有 $\sigma \models^I B[a/X]$ 当且仅当 $\sigma[\mathscr{A}[a]\sigma/X] \models^I B$。于是

$$\sigma \models^I B[a/X] \Rightarrow \mathscr{C}[X := a]\sigma \models^I B$$

从而有 $\models\{B[a/X]\}X := a\{B\}$，即证明了赋值规则是可靠的。

顺序复合规则：设 $\models\{A\}c_0\{C\}$ 且 $\models\{C\}c_1\{B\}$，设 I 是一个解释，设 $\sigma \models^I A$，于是因为

$\models^I\{A\}c_0\{C\}$，得 $\mathscr{C}[c_0]\sigma\models^I C$。同样，由于 $\models^I\{C\}c_1\{B\}$，得 $\mathscr{C}[c_1](\mathscr{C}[c_0]\sigma)\models^I B$。所以有 $\models\{A\}c_0;c_1\{B\}$，即证明了顺序复合规则是可靠的。

条件规则：设 $\models\{A\wedge b\}c_0\{B\}$，且 $\models\{A\wedge\neg b\}c_1\{B\}$，设 I 是一个解释，假设 $\sigma\models^I A$，则或者 $\sigma\models^I b$，或者 $\sigma\models^I\neg b$。如果是前一种情况，有 $\sigma\models^I A\wedge b$，因为 $\models^I\{A\wedge b\}c_0\{B\}$，所以 $\mathscr{C}[c_0]\sigma\models^I B$。如果是后一种情况，有 $\sigma\models^I A\wedge\neg b$，因为 $\models^I\{A\wedge\neg b\}c_1\{B\}$，所以 $\mathscr{C}[c_1]\sigma\models^I B$。这就确保了 $\models\{A\}$ **if** b **then** c_0 **else** $c_1\{B\}$。

while 循环规则：假设 $\models\{A\wedge b\}c\{A\}$，即 A 是循环语句

$$w\equiv\textbf{while } b \textbf{ do } c$$

的一个不变式。设 I 为一个解释。读者回忆 $\mathscr{C}[w]=\cup_{n\in\omega}\theta_n$，其中

$$\theta_0=\emptyset,$$

$$\theta_{n+1}=\{(\sigma,\sigma')\mid\mathscr{B}[b]\sigma=\textbf{true}\ \&\ (\sigma,\sigma')\in\theta_n\circ\mathscr{C}[c]\}\cup\{(\sigma,\sigma)\mid\mathscr{B}[b]\sigma=\textbf{false}\}$$

下面我们用数学归纳法证明下述 $P(n)$ 成立，$P(n)$ 定义为：对所有的 $n\in\omega$，

$$P(n)\Longleftrightarrow_{def}\forall\sigma,\sigma'\in\Sigma.\ (\sigma,\sigma')\in\theta_n\ \&\ \sigma\models^I A\Rightarrow\sigma'\models^I A\wedge\neg b$$

于是，对于所有的状态 σ，有

$$\sigma\models^I A\Rightarrow\mathscr{C}[w]\sigma\models^I A\wedge\neg b$$

因此 $\models\{A\}w\{A\wedge\neg b\}$。

奠基：令 $n=0$，则 $\theta_0=\emptyset$，所以性质 $P(0)$ 自动为真。

归纳步骤：我们假设对 $n\geqslant 0$，性质 $P(n)$ 为真，现要证明性质 $P(n+1)$ 为真。假设 $(\sigma,\sigma')\in\theta_{n+1}$，且 $\sigma\models^I A$，则

(i) $\mathscr{B}[b]\sigma=\textbf{true}\ \&\ (\sigma,\sigma')\in\theta_n\circ\mathscr{C}[c]$，或者

92 (ii) $\mathscr{B}[b]\sigma=\textbf{false}\ \&\ \sigma'=\sigma$。

下面我们证明，不管哪一种情况都有 $\sigma'\models^I A\wedge\neg b$。

假设是(i)的情况，由于 $\mathscr{B}[b]\sigma=\textbf{true}$，于是 $\sigma\models^I b$，因此 $\sigma\models^I A\wedge b$。同理，对某个状态 σ'' 使得 $(\sigma,\sigma'')\in\mathscr{C}[c]$，且 $(\sigma'',\sigma')\in\theta_n$，由于 $\models\{A\wedge b\}c\{A\}$，可得 $\sigma''\models^I A$，由性质 $P(n)$ 为真，得 $\sigma'\models^I A\wedge\neg b$。

假设是(ii)的情况，由于 $\mathscr{B}[b]\sigma=\textbf{false}$，于是 $\sigma\models^I\neg b$，因此，$\sigma\models^I A\wedge\neg b$，而 $\sigma'=\sigma$，所以 $\sigma'\models^I A\wedge\neg b$。

这样我们就证明了性质 $P(n+1)$ 为真。根据数学归纳法，我们得到对于所有的 n 性质 $P(n)$ 都为真，所以 while 循环规则是可靠的。

推论规则：设 $\models(A\Rightarrow A')$，$\models\{A'\}c\{B'\}$ 且 $\models(B'\Rightarrow B)$；令 I 是一个解释；设 $\sigma\models^I A$。于是 $\sigma\models^I A'$，因此 $\mathscr{C}[c]\sigma\models^I B'$，于是 $\mathscr{C}[c]\sigma\models^I B$，从而得 $\models\{A\}c\{B\}$，即推论规则是可靠的。

根据规则归纳原理，每一条定理都是有效的。 □

练习 6.12 在不用指称语义，只用操作语义的情况下，试证明上面的定理。你用什么方

法证明 while 循环的可靠性? □

6.6 应用霍尔规则的一个示例

用霍尔规则进行推导能得到部分正确性断言的形式化证明,所以霍尔规则能用于机器证明。但是,在实践中人们的任务只是验证程序,而并不要求严格的形式化证明,当我们使用霍尔规则时,能在相对非形式化的层面上验证程序。(过于形式化的推导会分散人们在证明过程上的精力,可以把这方面的工作交给一些辅助证明工具如 LCF 或 HOL[74][43]去完成。)

下面举一个应用霍尔规则的例子,详细说明如何使用霍尔规则。验证命令

$$w \equiv (\textbf{while } X > 0 \textbf{ do } Y := X \times Y; X := X - 1)$$

确实计算阶乘函数 $n! = n \times (n-1) \times (n-2) \times \cdots \times 2 \times 1$。我们约定 $0! = 1$, X 的初始值为非负整数 n, Y 为 1。[⊖]

更准确地说,我们要证明:

$$\{X = n \wedge n \geqslant 0 \wedge Y = 1\} w \{Y = n!\}$$

显然,证明要用到 while 循环的证明规则及其不变式。令不变式为

$$I \equiv (Y \times X! = n! \wedge X \geqslant 0)$$

93

我们先证明 I 确实是一个不变式,即

$$\{I \wedge X > 0\} Y := X \times Y; X := X - 1 \{I\}$$

由赋值规则,我们得到

$$\{I[(X-1)/X]\} X := X - 1 \{I\}$$

其中 $I[(X-1)/X] \equiv (Y \times (X-1)! = n! \wedge (X-1) \geqslant 0)$。再由赋值规则得

$$\{X \times Y \times (X-1)! = n! \wedge (X-1) \geqslant 0\} Y := X \times Y \{I[(X-1)/X]\}$$

这样,由顺序复合规则,

$$\{X \times Y \times (X-1)! = n! \wedge (X-1) \geqslant 0\} Y := X \times Y; X := X - 1 \{I\}$$

显然,

$$\begin{aligned} I \wedge X > 0 &\Rightarrow Y \times X! = n! \wedge X \geqslant 0 \wedge X > 0 \\ &\Rightarrow Y \times X! = n! \wedge X \geqslant 1 \\ &\Rightarrow X \times Y \times (X-1)! = n! \wedge (X-1) \geqslant 0 \end{aligned}$$

于是,由推论规则得

$$\{I \wedge X > 0\} Y := X \times Y; X := X - 1 \{I\}$$

⊖ 本例中,我们设想扩充程序和断言的语法,使之包含 > 和阶乘函数,而严格来讲,我们早先定义的布尔表达式和算术表达式并不包含它们。

这说明 I 确实是一个不变式。

现在,我们应用 while 循环规则,得

$$\{I\}w\{I \wedge X \not> 0\}$$

显然 $(X=n) \wedge (n \geqslant 0) \wedge (Y=1) \Rightarrow I$,且

$$\begin{aligned} I \wedge X \not> 0 &\Rightarrow Y \times X! = n! \wedge X \geqslant 0 \wedge X \not> 0 \\ &\Rightarrow Y \times X! = n! \wedge X = 0 \\ &\Rightarrow Y \times 0! = Y = n! \end{aligned} \qquad (*)$$

这样,由推论规则,我们得出结论

$$\{(X = n) \wedge (Y = 1)\}w\{Y = n!\}$$

本例的证明中有几点要注意。首先,分析顺序复合命令的时候一般采取从右往左的方法比较容易,因为赋值规则本身就有这个特点。其次,要尽可能选择比较强的不变式 I。本例证明中在不变式里又加上了断言 $X \geqslant 0$,这样才能够在式(*)中推出 while 循环的出口处 $X=0$。在证明过程中经常要加强不变式,这和归纳证明中的归纳假设相似。加强不变式的一种常见方法是指明变量的值域且尽可能贴近存储单元的值。毫无疑问,常见的难点在于证明"循环出口条件"。在本例中,一个好的想法是利用布尔表达式中变量和存储单元的信息来加强循环不变式。

94

可见,即使证明很小的程序的正确性也不是那么容易。当然,如果用形式证明系统去证明数学中所有细节问题的话,将会更困难。形式证明系统的一个优点是它能够自动证明程序的某些性质,见文献[74][41]以及 7.4 节关于验证条件的讨论。形式证明系统的另外一个应用方法就是迪杰斯特拉(Dijkstra)和格里斯(Gries)等提出的观点:程序设计过程中应该贯穿程序正确性的理论。格里斯在《程序设计科学》(*The Science of Programming*)[44]一书中提出:

> "程序正确性证明的研究促进了程序开发方法的发展。事实上,程序及其正确性证明应该同时进行,而程序正确性的证明思想引导了程序设计的方法!"

读者可参考格里斯的书中有关该方法的许多例子。

练习 6.13 试用霍尔规则证明下述部分正确性断言的正确性:

$$\begin{aligned} &\{1 \leqslant N\} \\ &P := 0; \\ &C := 1; \\ &(\textbf{while } C \leqslant N \textbf{ do } P := P + M; C := C + 1) \\ &\{P = M \times N\} \end{aligned}$$

□

练习 6.14 找出一个合适的不变式,以便用于 **while** 循环规则中去证明下述部分正确性断言:

95

$$\{i = Y\}X := 1; \textbf{while } \neg(Y = 0) \textbf{ do } Y := Y - 1; X := 2 \times X\{X = 2^i\}$$ □

练习 6.15 试用霍尔规则证明:对于整数 n, m,有

$$\{X = m \wedge Y = n \wedge Z = 1\}c\{Z = m^n\}$$

其中 c 是 while 程序，即

$$\begin{aligned}&\textbf{while } \neg(Y = 0) \textbf{ do}\\&((\textbf{while } even(Y) \textbf{ do } X := X \times X; Y := Y/2);\\&Z := Z \times X; Y := Y - 1)\end{aligned}$$

循环命令中 $Y/2$ 表示用 2 除 Y 的内容所得的整数，$even(Y)$ 表示 Y 的内容是一个偶数。

（提示：用 $m^n = Z \times X^Y$ 作为不变式。） □

练习 6.16 （i）试证明两个正数 n,m 的最大公约数 $\gcd(n,m)$ 满足：

$$\begin{aligned}&(a)\ n > m \Rightarrow \gcd(n,m) = \gcd(n-m,m)\\&(b)\ \gcd(n,m) = \gcd(m,n)\\&(c)\ \gcd(n,n) = n\end{aligned}$$

（ii）试用霍尔规则证明

$$\{N = n \wedge M = m \wedge 1 \leqslant n \wedge 1 \leqslant m\}\text{Euclid}\{X = \gcd(n,m)\}$$

其中

$$\begin{aligned}\text{Euclid} \equiv \textbf{while } \neg(M = N) \textbf{ do}\\ \textbf{if } M \leqslant N\\ \textbf{then } N := N - M\\ \textbf{else } M := M - N\end{aligned}$$ □

练习 6.17 请给出 repeat 结构的霍尔规则，并证明该规则是可靠的（见练习 5.9）。 □

6.7 进一步阅读资料

已经提到过 Gries 的书[44]。Dijkstra 的《程序设计的训练》(*A discipline of programming*)[36]一书影响非常广泛。Backhouse 的《程序构造与验证》(*Program construction and verification*)[12]一书的内容较为基础。Cohen 的《九十年代的程序设计》(*Programming in the* 1990's)[32]是新近出版的一本书。Alagić 和 Arbib 的《良结构正确程序的设计》(*The design of well-structured and correct programs*)[5]是一本好书，还附有许多练习。Gordon 的新作[42]对霍尔逻辑作了基本而又有启发性的讨论。Bakker 的《程序正确性的数学理论》(*Mathematical theory of program correctness*)[13]和 Loeckx 与 Sieber 合著的《程序验证基础》(*The foundations of program verification*)[58]则对语义问题作了较多的讨论。 96 97

第7章　霍尔规则的完备性

本章讨论霍尔规则的完备性问题。根据哥德尔不完备性定理，能精确建立有效断言的完备的证明系统是不存在的，霍尔系统也是不完备的。然而，我们把断言语言的不完备性与由于程序语言结构的公理和规则的不当选择而引起的不完备性分开，从而得到库克（Cook）的相对完备性。证明霍尔规则是相对完备的要依赖于最弱宽松前置条件。本章最后讨论验证条件生成器。

7.1　哥德尔不完备性定理

再看一下部分正确性断言的证明规则，特别是其中的推论规则。如果要得到推论规则的一个规则实例，需要我们确定 **Assn** 中相应的断言是有效的。当然，在理想状态下我们希望对断言有一个公理和规则的证明系统，它能证明 **Assn** 中所有有效的断言，而不证明任何无效的断言。很自然，我们希望证明系统是能行的，即能判别输入是否确实为一个规则实例。证明系统需要有一个程序形式的计算方法：如果输入一个真正的规则实例，则程序返回一个确认值；如果输入不是真正的规则实例，则程序将不返回确认值，甚至可能不会终止。但是这样的计算方法往往难以找到，因此用这种方法检验规则实例是不现实的。我们不能宣称霍尔规则的证明系统是能行的，因为不存在一个计算方法以检验推论规则的实例。显然，对规则实例的检验取决于对 **Assn** 断言的有效性检验的计算方法。但在这一点上我们受到了绝对的限制。著名的奥地利逻辑学家哥德尔（Kurt Gödel）指出：在逻辑上不存在一个能行的证明系统，人们用它能够恰好证明 **Assn** 的所有有效断言。哥德尔的这一著名结果称作哥德尔不完备性定理[⊖]，用可计算性理论的有关结论可证明这个定理，证明过程在 7.3 节给出。如果读者理解这个证明有困难，可以参阅附录中基于 **IMP** 语言的关于可计算性和不可判定性的介绍。

定理 7.1（哥德尔不完备性定理（1931））　*不存在 **Assn** 的一个能行的证明系统，使得 **Assn** 的有效断言恰好是该系统的定理。*

该定理意味着，不存在部分正确性断言的能行的证明系统。由于$\models B$ 当且仅当$\models\{\mathbf{true}\}\ \mathbf{skip}\{B\}$，如果存在一个部分正确性断言的能行的证明系统，那么这个证明系统可简化为证明 **Assn** 断言的能行的证明系统，根据哥德尔不完备性定理，这是不可能的。事实上，可以更直接地证明对部分正确性断言不存在能行的证明系统。

命题 7.2　*对部分正确性断言，不存在一个能行的证明系统，使得其定理恰好是有效的部分正确性断言。*

证明　注意到$\models\{\mathbf{true}\}\ c\{\mathbf{false}\}$当且且仅当命令 c 在所有的状态下都发散。如果我们有

⊖ 不要混淆不完备性定理和哥德尔完备性定理。后者指谓词演算的证明系统恰好生成那些对所有的解释均有效的断言。

一个部分正确性断言的能行的证明系统，那么该系统可生成一个计算方法，以确认命令 c 在所有的状态下都发散。而这是不可能的——请参见附录中练习 A.13。 □

以上事实说明，尽管由于推论规则的实例要求相应的断言必须有效，因而霍尔证明系统不是能行的，但我们满足于霍尔规则的证明系统。我们仍然可以探讨该系统的完备性。库克在文献[33]中指出，霍尔系统是相对完备的。如果一个部分正确性断言是有效的，则使用霍尔规则存在该断言的一个证明，即对于任何部分正确性断言 $\{A\}c\{B\}$，有

$$\models\{A\}c\{B\} \text{ 蕴涵 } \vdash\{A\}c\{B\}$$

当然这个证明依赖于 **Assn** 断言的有效性。在构造证明的每一步中，人们都要知道 **Assn** 断言是否有效。人们称库克的结论建立了部分正确性的霍尔规则的相对完备性，其相对完备性和算术运算的有效断言有关。这样，人们就把程序及其推理从算术运算及其证明系统的不完备性中分离出来。

7.2 最弱前置条件与可表达性

相对完备性的证明依赖于最弱前置条件这一概念。考察要证明

100

$$\{A\}c_0;c_1\{B\}$$

为了使用复合规则，我们需要一个中间断言 C，使得

$$\{A\}c_0\{C\} \text{ 和 } \{C\}c_1\{B\}$$

都是可证明的。我们怎么知道这样一个中间断言 C 能找到呢？一个充分条件是，对每条命令 c 和后置条件 B，我们能用 **Assn** 表示它们的最弱前置条件[⊖]。

设 $c\in\mathbf{Com}$，$B\in\mathbf{Assn}$，I 是解释，则 B 关于解释 I 对应于 c 的最弱前置条件 $wp^I[c,B]$ 定义为：

$$wp^I[c,B] = \{\sigma\in\Sigma_\perp \mid \mathscr{C}[c]\sigma\models^I B\}$$

$wp^I[c,B]$ 是所有那些状态的集合，在这些状态下执行 c 要么发散，要么终止于满足 B 的终态。于是，如果 $\models^I\{A\}c\{B\}$，则我们知道有 $A^I\subseteq wp^I[c,B]$，反之亦然。这样有 $\models^I\{A\}c\{B\}$ 当且仅当 $A^I\subseteq wp^I[c,B]$。

假设存在断言 A_0 使得对于所有的解释 I，有

$$A_0^I = wp^I[c,B]$$

于是对于所有的解释 I，有

$$\models^I\{A\}c\{B\} \text{ 当且仅当 } \models^I(A\Rightarrow A_0)$$

即

$$\models\{A\}c\{B\} \text{ 当且仅当 } \models(A\Rightarrow A_0)$$

⊖ 我们所谓的最弱前置条件一般称作最弱宽松前置条件，最弱前置条件这一术语涉及到完全正确性的相关记号。

现在我们知道为什么称它为最弱前置条件了，因为使部分正确性断言有效的任一前置条件，都蕴涵着最弱前置条件。但是，特定的断言语言并不一定都包含断言 A_0，使得 $A_0^I = wp^I[c,B]$。

定义 称 **Assn** 是可表达的当且仅当对于每一个命令 c 和断言 B 存在一个断言 A_0，使得对任意的解释 $I, A_0^I = wp^I[c,B]$。

在证明可表达性中，我们用哥德尔的 β 谓词对 **Assn** 断言的状态序列进行编码。β 谓词包含了求 a 除以 b 的余数的模运算 a **mod** b。我们将这种概念表示为 **Assn** 断言，对 $x = a$ **mod** b，记作 101

$$a \geqslant 0 \wedge b \geqslant 0 \wedge$$
$$\exists k.\,[0 \leqslant x < b \wedge x = a - (k \times b)]$$

引理 7.3 设 $\beta(a,b,i,x)$ 是定义在自然数上的谓词，

$$\beta(a,b,i,x) \Longleftrightarrow_{def} x = a \ \mathbf{mod}(1 + (1 + i) \times b)$$

对于自然数的任一序列 $n_0,\cdots,n_k$，存在自然数 n,m，使得对所有的 j $(0 \leqslant j \leqslant k)$ 及所有的 x，有

$$\beta(n,m,j,x) \Longleftrightarrow x = n_j$$

证明 证明这个算术运算事实作为本节最后的习题，留给读者完成。 □

β 谓词非常重要，因为它可以把由 $k+1$ 个自然数构成的序列 $n_0,\cdots,n_k$ 编码成一对数 n，m。给定 n 和 m，对任何长度 k，我们能得到一个序列，即自然数序列 $n_0,\cdots,n_k$，使得

$$\beta(n,m,j,n_j) \ (0 \leqslant j \leqslant k)$$

β 的定义表明序列 $n_0,\cdots,n_k$ 由所选择的 n,m 惟一确定。引理 7.3 表明任何一个自然数序列 $n_0,\cdots,n_k$ 都可以用 β 谓词来编码。

这里还有一个问题，我们讨论的断言语言和状态包含正整数和负整数。因此为了对正整数和负整数序列进行编码还必须扩充 β 谓词。幸好，一个简便的编码方法是：正整数用偶数编码，负整数用奇数编码。

引理 7.4 设 $F(x,y)$ 是定义在自然数 x 和整数 y 上的谓词，由

$$F(x,y) \equiv x \geqslant 0 \ \&$$
$$\exists z \geqslant 0. \quad [(x = 2 \times z \Rightarrow y = z) \ \&$$
$$(x = 2 \times z - 1 \Rightarrow y = -z)]$$

给出。定义

$$\beta^{\pm}(n,m,j,y) \Longleftrightarrow_{def} \exists x.\,(\beta(n,m,j,x) \wedge F(x,y))$$ 102

则对于任何正整数或负整数序列 $n_0,\cdots,n_k$，存在自然数 n,m，使得对所有的 $j(0 \leqslant j \leqslant k)$ 及所有的 x，有

$$\beta^{\pm}(n,m,j,x) \Longleftrightarrow x = n_j$$

证明 显然，$F(m,n)$ 表示自然数 $m \in \omega$ 和整数 $n \in \mathbf{N}$ 之间的一一对应关系，其中偶数 m 代表非负整数，而奇数 m 代表负整数。本引理可以由引理 7.3 推出。 □

谓词 $\beta^{\pm}$ 在 **Assn** 中是可表达的，因为 β 和 F 都是可表达的。为了避免引入新的符号，仍采用 $\beta^{\pm}$ 来表示 **Assn** 中代表该谓词的断言。**Assn** 中该断言含有自由整型变量 n,m,j,x，它们与上述引理中的含义相同，即用 n,m 对第 j 个元素 x 的序列进行编码。除 n,m,j,x 之外，我们想使用其他的整型变量，因此把 $\beta^{\pm}[n'/n,m'/m,j'/j,x'/x]$ 简记为 $\beta^{\pm}(n',m',j',x')$（用 n' 替代 n，用 m' 替代 m，依此类推）。现在我们还没有给出断言中整型变量代入的形式化定义，在 6.2.2 节中我们曾定义断言 A 的代入 $A[a/i]$，只不过当时是用不含整型变量的算术表达式 a 对 A 的整型变量 i 进行代入。然而，如果变量 n',m',j',x' 是“新”的，即它们各不相同且在 $\beta^{\pm}$ 中不含（自由或约束）出现，则上述定义对于含整型变量的代入也同样适用；于是根据 6.2.2[⊖] 节的定义，断言 $\beta^{\pm}[n'/n,m'/m,j'/j,x'/x]$ 可以由 $\beta^{\pm}[n'/n][m'/m][j'/j][x'/x]$ 给出。

下面我们证明一个定理。

定理 7.5 **Assn** 是可表达的。

证明 施结构归纳于命令 c，对所有的断言 B，存在一个断言 $w[c,B]$ 使得对于所有的解释 I 和所有的命令 c，有

$$wp^I[c,B] = w[c,B]^I$$

根据最弱前置条件的定义，对于解释 I，等式 $wp^I[c,B] = w[c,B]^I$ 可表示为：对所有的状态 σ，有

103

$$\sigma \models^I w[c,B] \text{ 当且仅当 } \mathscr{C}[c]\sigma \models^I B$$

证明中会用到后一种表示。

$c \equiv \mathbf{skip}$：此时，取 $w[\mathbf{skip},B] \equiv B$。显然，对于所有的状态 σ 和解释 I，有

$$\begin{aligned}\sigma \in wp^I[\mathbf{skip},B] &\text{ 当且仅当 } \mathscr{C}[\mathbf{skip}]\sigma \models^I B\\ &\text{ 当且仅当 } \sigma \models^I B\\ &\text{ 当且仅当 } \sigma \models^I w[\mathbf{skip},B]\end{aligned}$$

$c \equiv (X := a)$：此时，定义 $w[X := a,B] \equiv B[a/X]$。于是

$$\begin{aligned}\sigma \in wp^I[X := a,B] &\text{ 当且仅当 } \sigma[\mathscr{A}[a]\sigma/X] \models^I B\\ &\text{ 当且仅当 } \sigma \models^I B[a/X] \text{（由引理 6.9）}\\ &\text{ 当且仅当 } \sigma \models^I w[X := a,B]\end{aligned}$$

$c \equiv c_0;c_1$：此时，归纳定义 $w[c_0;c_1,B] \equiv w[c_0,w[c_1,B]]$。于是，对 $\sigma \in \Sigma$ 和解释 I，有

$$\sigma \in wp^I[c_0;c_1,B] \text{ 当且仅当 } \mathscr{C}[c_0;c_1]\sigma \models^I B$$

⊖ 为了说明用不是新的变元进行代入会引起的技术问题，考察断言 $A \equiv (\exists i'.2 \times i' = i)$，它意味着“$i$ 是偶数”，一个简单的定义 $A[i'/i]$ 产生了断言 $(\exists i'.2 \times i' = i')$，该断言正好也是有效的，但并不意味着“$i'$ 是偶数”。

当且仅当 $\mathscr{C}[\![c_1]\!](\mathscr{C}[\![c_0]\!]\sigma)\models^I B$

当且仅当 $\mathscr{C}[\![c_0]\!]\sigma\models^I w[\![c_1,B]\!]$（由归纳假设）

当且仅当 $\sigma\models^I w[\![c_0,w[\![c_1,B]\!]]\!]$（由归纳假设）

当且仅当 $\sigma\models^I w[\![c_0;c_1,B]\!]$

$c\equiv$ **if** b **then** c_0 **else** c_1：定义

$$w[\![\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1,B]\!] \equiv [(b \wedge w[\![c_0,B]\!]) \vee (\neg b \wedge w[\![c_1,B]\!])]$$

于是，对 $\sigma\in\Sigma$ 和解释 I，有

$\sigma \in wp^I[\![c,B]\!]$ 当且仅当 $\mathscr{C}[\![c]\!]\sigma\models^I B$

当且仅当（$[\mathscr{B}[\![b]\!]\sigma = \textbf{true} \ \&\ \mathscr{C}[\![c_0]\!]\sigma\models^I B]$ 或

$[\mathscr{B}[\![b]\!]\sigma = \textbf{false} \ \&\ \mathscr{C}[\![c_1]\!]\sigma\models^I B]$）

当且仅当（$[\sigma\models^I b \ \&\ \sigma\models^I w[\![c_0,B]\!]]$ 或

$[\sigma\models^I \neg b \ \&\ \sigma\models^I w[\![c_1,B]\!]]$）（由归纳假设）

当且仅当 $\sigma\models^I[(b \wedge w[\![c_0,B]\!]) \vee (\neg b \wedge w[\![c_1,B]\!])]$

当且仅当 $\sigma\models^I w[\![c,B]\!]$ 104

$c\equiv$ **while** b **do** c_0：此时证明有些困难。由练习 5.8 的结论，对于状态 σ 和解释 I，有 $\sigma\in wp^I[\![c,B]\!]$ 当且仅当

$$\begin{aligned}
&\forall k \forall \sigma_0,\cdots,\sigma_k \in \Sigma \\
&[\sigma = \sigma_0 \ \& \\
&\forall i(0 \leqslant i < k).(\sigma_i\models^I b \ \& \\
&\qquad\qquad \mathscr{C}[\![c_0]\!]\sigma_i = \sigma_{i+1})] \\
&\Rightarrow(\sigma_k\models^I b \vee B)
\end{aligned} \tag{1}$$

可见，$wp^I[\![c,B]\!]$ 中状态 σ 的数学刻画不是 **Assn** 断言，特别地，它直接引用 $\sigma_0,\cdots,\sigma_k$。然而，我们可以给出一个等价的描述来替换它。首先用命令 c 和断言 B 中存储单元的内容去替换所有的状态 $\sigma_0,\cdots,\sigma_k$。设 $\bar{X}=X_1,\cdots,X_l$ 是与 B 和 c 相关的存储单元，其余存储单元的内容是与计算无关的，因而不予考虑。先看一看下面的事实。

设 A 是一个 **Assn** 断言，且 A 只涉及 $\bar{X}=X_1,\cdots,X_l$ 中的存储单元。对于状态 σ，令 $s_i=\sigma(X_i)$，$1\leqslant i\leqslant l$，并令 $\bar{s}=s_1,\cdots,s_l$，则对于任何解释 I，有

$$\sigma\models^I A \text{ 当且仅当 } \models^I A[\bar{s}/\bar{X}] \tag{*}$$

断言 $A[\bar{s}/\bar{X}]$ 是同时用 $\bar{s}$ 来替换 A 中的 $\bar{X}$ 获得的。可以用结构归纳法证明这个事实（留作练习）。

由结论（*），断言（1）可以转换为另一个等价的断言序列。将 **N** 中序列 $s_{i1},\cdots,s_{il}(i\geqslant 0)$ 简记为 $\bar{s}_i$，我们宣称：$\sigma\in wp^I[\![c,B]\!]$ 当且仅当

$$\forall k \forall \bar{s}_0,\cdots,\bar{s}_k \in \mathbf{N}$$

$$
\begin{aligned}
&[\sigma \models^I \bar{X} = \bar{s}_0 \ \& \\
&\forall i(0 \leqslant i < k).(\models^I b[\bar{s}_i/\bar{X}] \ \& \\
&\qquad\qquad \models^I (w[c_0, \bar{X} = \bar{s}_{i+1}] \wedge \neg\, w[c_0, \mathbf{false}])[\bar{s}_i/\bar{X}]) \\
&\Rightarrow \models^I (b \vee B)[\bar{s}_k/\bar{X}]
\end{aligned} \tag{2}
$$

105 这里，我们把 $X_1 = s_{01} \wedge \cdots \wedge X_l = s_{0l}$ 简记为 $\bar{X} = \bar{s}_0$。

为了证明(2)，我们要证明(1)和(2)是等价的。其中部分证明很直观。例如，由(*)直接可得，如果在状态 σ_i 下，$\bar{X}$ 中的内容为 $\bar{s}_i$，则对于任意的解释 I，有

$$\sigma_i \models^I b \text{ 当且仅当 } \models^I b[\bar{s}_i/\bar{X}]$$

证明的难点在于证明：如果在状态 σ_i 和 σ_{i+1} 下，$\bar{X}$ 的值分别为 $\bar{s}_i$ 和 $\bar{s}_{i+1}$ 而在其他的地方取值相同，则对于解释 I，我们有

$$\mathscr{E}[c_0]\sigma_i = \sigma_{i+1} \text{ 当且仅当 } \models^I (w[c_0, \bar{X} = \bar{s}_{i+1}] \wedge \neg\, w[c_0, \mathbf{false}])\ [\bar{s}_i/\bar{X}]$$

首先，注意到

$$\mathscr{E}[c_0]\sigma_i = \sigma_{i+1} \text{ 当且仅当 } \sigma_i \in wp^I[c_0, \bar{X} = \bar{s}_{i+1}] \ \& \ \mathscr{E}[c_0]\sigma_i \text{ 有定义}$$

(请读者思考为什么?)由归纳假设得

$$
\begin{aligned}
&\sigma_i \in wp^I[c_0, \bar{X} = \bar{s}_{i+1}] \text{ 当且仅当 } \sigma_i \models^I w[c_0, \bar{X} = \bar{s}_{i+1}] \\
&\text{且 } \mathscr{E}[c_0]\sigma_i \text{ 有定义} \quad \text{当且仅当 } \sigma_i \models^I \neg\, w[c_0, \mathbf{false}]
\end{aligned}
$$

读者回忆一下我们曾经证明过 $\sigma_i \in wp^I[c_0, \mathbf{false}]$ 当且仅当 c_0 在状态 σ_i 下发散。于是，

$$
\begin{aligned}
\mathscr{E}[c_0]\sigma_i = \sigma_{i+1} &\text{ 当且仅当 } \sigma_i \models^I (w[c_0, \bar{X} = \bar{s}_{i+1}] \wedge \neg\, w[c_0, \mathbf{false}]) \\
&\text{当且仅当 } \models^I (w[c_0, \bar{X} = \bar{s}_{i+1}] \wedge \neg\, w[c_0, \mathbf{false}])[\bar{s}_i/\bar{X}] \quad (\text{由}(*)\text{式})
\end{aligned}
$$

从而证明了(1)和(2)是等价的。

最后，我们说明如何用哥德尔谓词 β^* 把(2)表示成 **Assn** 断言。为简单起见，设 $l = 1$，$\bar{X} = X$，于是(2)重新描述为：$\sigma \in wp^I[c, B]$ 当且仅当

$$
\begin{aligned}
\sigma \models^I\ & \forall k \forall m, n \geqslant 0 \\
& [\beta^*(n, m, 0, X) \wedge \\
& \forall i(0 \leqslant i < k).(\forall x.\beta^*(n, m, i, x) \Rightarrow b[x/X]) \wedge \\
& \qquad \forall x, y.(\beta^*(n, m, i, x) \wedge \beta^*(n, m, i+1, y) \Rightarrow \\
& \qquad (w[c_0, X = y] \wedge \neg\, w[c_0, \mathbf{false}])[x/X])] \\
& \Rightarrow \forall x.(\beta^*(n, m, k, x) \Rightarrow (b \vee B)[x/X])
\end{aligned}
$$

这正好是表示 $w[c, B]$ 的断言(读者可以把这个断言逐行和(2)比较，注意 $\beta^*(n, m, i, x)$ 表示
106 x 是用 n, m 编码的序列中的第 i 个元素)。更一般地，对于任意的 l 也可以给出表示 $w[c, B]$ 的断言，虽然稍微复杂但断言的形式类似，留给读者完成。

这就用结构归纳法完成了证明。 □

对于所有的命令 c 和断言 B,如果 **Assn** 是可表达的,则存在断言 $w[c,B]$,对任一解释 I,它具有性质

$$w[c,B]^I = wp^I[c,B]$$

当然,上述证明中构造的断言 $w[c,B]$ 并不是具有这个性质的惟一断言(请思考一下为什么?)。然而,假设 A_0 是另外的断言,对于所有的解释 I,$A_0^I = wp^I[c,B]$。于是

$$\models (w[c,B] \Longleftrightarrow A_0)$$

因此,在逻辑等价意义下,表示最弱前置条件的断言是惟一的,而断言 $w[c,B]$ 最关键的作用是:根据最弱前置条件的定义,对所有的状态 σ 和解释 I,可以由下式来刻画这个断言:

$$\sigma \models^I w[c,B] \text{ 当且仅当 } \mathscr{C}[c]\sigma \models^I B$$

从 **Assn** 的可表达性,我们可以证明相对完备性。首先证明一个重要的引理。

引理 7.6 对于 $c \in \mathbf{Com}, B \in \mathbf{Assn}$,令 $w[c,B]$ 是表示最弱前置条件的断言,即 $w[c,B]^I = wp^I[c,B]$(断言 $w[c,B]$ 不必按照定理 7.5 的方法构造),则

$$\vdash \{w[c,B]\}\, c\, \{B\}$$

证明 令 $w[c,B]$ 是表示命令 c 和后置条件 B 的最弱前置条件的断言。对所有的 $B \in \mathbf{Assn}$,施结构归纳于命令 c,证明 $\vdash \{w[c,B]\}\, c\, \{B\}$。

(在下面的证明中,除最后一种情况外,与定理 7.5 的证法相同。)

$c \equiv \mathbf{skip}$:此时,$\models w[\mathbf{skip},B] \Longleftrightarrow B$,于是,根据推论规则,有 $\vdash \{w[\mathbf{skip},B]\}\, \mathbf{skip}\, \{B\}$。 107

$c \equiv (X := a)$:此时,因为

$$\begin{aligned} \sigma \in wp^I[c,B] &\text{ 当且仅当 } \sigma[\mathscr{A}[a]\sigma/X] \models^I B \\ &\text{ 当且仅当 } \sigma \models^I B[a/X] \end{aligned}$$

所以 $\models (w[c,B] \Longleftrightarrow B[a/X])$。因此,由赋值规则和推论规则可知 $\vdash \{w[c,B]\}\, c\, \{B\}$。

$c \equiv c_0; c_1$:此时,对于状态 $\sigma \in \Sigma$ 和解释 I,有

$$\begin{aligned} \sigma \models^I w[c_0;c_1,B] &\text{ 当且仅当 } \mathscr{C}[c_0;c_1]\sigma \models^I B \\ &\text{ 当且仅当 } \mathscr{C}[c_1](\mathscr{C}[c_0]\sigma) \models^I B \\ &\text{ 当且仅当 } \mathscr{C}[c_0]\sigma \models^I w[c_1,B] \\ &\text{ 当且仅当 } \sigma \models^I w[c_0,w[c_1,B]] \end{aligned}$$

于是,$\models w[c_0;c_1,B] \Longleftrightarrow w[c_0,w[c_1,B]]$。由归纳假设,得

$$\begin{aligned} &\vdash \{w[c_0,w[c_1,B]]\}\, c_0\, \{w[c_1,B]\} \text{ 且} \\ &\vdash \{w[c_1,B]\}\, c_1\, \{B\} \end{aligned}$$

由顺序复合规则,可化简为

$$\vdash \{w[c_0,w[c_1,B]]\}\, c_0;c_1\, \{B\}$$

由推论规则,我们得

$$\vdash\{w[c_0;c_1,B]\}c_0;c_1\{B\}$$

$c\equiv$ **if** b **then** c_0 **else** c_1：此时，对于状态 $\sigma\in\Sigma$ 和解释 I，有

$$\begin{aligned}\sigma\models^I w[c,B] \text{ 当且仅当 } & \mathscr{C}[c]\sigma\models^I B\\ \text{当且仅当 } & ([\mathscr{B}[b]\sigma = \textbf{true} \ \&\ \mathscr{C}[c_0]\sigma\models^I B] \text{ 或}\\ & \quad [\mathscr{B}[b]\sigma = \textbf{false} \ \&\ \mathscr{C}[c_1]\sigma\models^I B])\\ \text{当且仅当 } & ([\sigma\models^I b \ \&\ \sigma\models^I w[c_0,B]] \text{ 或}\\ & \quad [\sigma\models^I \neg b \ \&\ \sigma\models^I w[c_1,B]])\\ \text{当且仅当 } & \sigma\models^I[(b\wedge w[c_0,B])\vee(\neg b\wedge w[c_1,B])]\end{aligned}$$

因此，

108

$$\models w[c,B]\Longleftrightarrow[(b\wedge w[c_0,B])\vee(\neg b\wedge w[c_1,B])]$$

现在，由归纳假设，得

$$\vdash\{w[c_0,B]\}c_0\{B\} \text{ 以及 } \vdash\{w[c_1,B]\}c_1\{B\}$$

但是

$$\begin{aligned}&\models(w[c,B]\wedge b)\Longleftrightarrow w[c_0,B] \text{ 且}\\ &\models(w[c,B]\wedge\neg b)\Longleftrightarrow w[c_1,B]\end{aligned}$$

因此，根据推论规则，有

$$\vdash\{w[c,B]\wedge b\}c_0\{B\} \text{ 以及 } \vdash\{w[c,B]\wedge\neg b\}c_1\{B\}$$

由条件规则，此时得 $\vdash\{w[c,B]\}c\{B\}$。

最后一种情况是：

$c\equiv$ **while** b **do** c_0：此时，取 $A\equiv w[c,B]$，我们先证明

(1) $\models\{A\wedge b\}c_0\{A\}$

(2) $\models(A\wedge\neg b)\Rightarrow B$

然后，根据(1)，由归纳假设，我们得 $\vdash\{A\wedge b\}c_0\{A\}$，再由循环命令规则，得 $\vdash\{A\}c\{A\wedge\neg b\}$。然后对(2)应用推论规则，就有 $\vdash\{A\}c\{B\}$。下面我们证明(1)和(2)。

(1) 设对于解释 I，有 $\sigma\models^I A\wedge b$，则 $\sigma\models^I w[c,B]$ 且 $\sigma\models^I b$，即 $\mathscr{C}[c]\sigma\models^I B$ 且 $\sigma\models^I b$。因为 $\mathscr{C}[c]$ 有定义，所以

$$\mathscr{C}[c] = \mathscr{C}[\textbf{if } b \textbf{ then } c_0;c \textbf{ else skip}]$$

这使得 $\mathscr{C}[c_0;c]\sigma\models^I B$，即 $\mathscr{C}[c](\mathscr{C}[c_0]\sigma)\models^I B$。因此，$\mathscr{C}[c_0]\sigma\models^I w[c,B]$，即 $\mathscr{C}[c_0]\sigma\models^I A$，所以 $\models\{A\wedge b\}c_0\{A\}$。

(2) 设对于解释 I，有 $\sigma\models^I A\wedge\neg b$，则 $\mathscr{C}[c]\sigma\models^I B$ 且 $\sigma\models^I\neg b$。同样，因为 $\mathscr{C}[c]=\mathscr{C}[\textbf{if } b \textbf{ then } c_0;c \textbf{ else skip}]$，所以 $\mathscr{C}[c]\sigma=\sigma$，所以 $\sigma\models^I B$。从而有 $\models^I A\wedge\neg b\Rightarrow B$，于是 $\models^I A\wedge\neg b$

$\Rightarrow B$,所以(2)得证。

综上所述,我们用结构归纳法证明了所有的情况,因此引理得证。 □

定理7.7 部分正确性断言的证明系统是相对完备的,即对于任意的部分正确性断言$\{A\}c\{B\}$,有

$$\text{如果 } \vdash\{A\}c\{B\}\text{,则 } \vdash\{A\}c\{B\}$$

109

证明 假设$\models\{A\}c\{B\}$,根据引理7.6有$\vdash\{w[c,B]\}c\{B\}$,其中对任意的解释I,$w[c,B]^I=wp^I[c,B]$,那么,由于$\models(A\Rightarrow w[c,B])$,再根据推论规则,可得$\vdash\{A\}c\{B\}$。 □

练习7.8 (哥德尔β谓词)

(a) 设$n_0,\cdots,n_k$为自然数序列,令

$$m=(\max\{k,n_0,\cdots,n_k\})!$$

试证明

$$p_i=1+(1+i)\times m \qquad (0\leqslant i\leqslant k)$$

是互素的(即对于$i\neq j,\gcd(p_i,p_j)=1$)且$n_i<p_i$。

(b) 我们进一步定义

$$c_i=p_0\times\cdots\times p_k/p_i \qquad (0\leqslant i\leqslant k)$$

试证明对于所有的$i(0\leqslant i\leqslant k)$,存在惟一的$d_i(0\leqslant d_i<p_i)$使得$(c_i\times d_i)\ \mathbf{mod}\ p_i=1$

(c) 我们另外定义

$$n=\sum_{i=0}^{k}c_i\times d_i\times n_i$$

试证明

$$n_i=n\ \mathbf{mod}\ p_i \qquad (0\leqslant i\leqslant k)$$

(d) 最后试证明引理7.3。 □

7.3 哥德尔定理的证明

哥德尔不完备性定理是指**Assn**中有效断言的子集不是递归可枚举的。(也就是说,不存在这样的程序,给定输入断言,对有效的断言返回一个确认值。读者可以参考附录中关于可计算性的精确定义和更详尽的阐述。)

定理7.9 有效断言的子集$\{A\in\mathbf{Assn}\mid\models A\}$不是递归可枚举的。

110

证明 假设不然,即集合$\{A\in\mathbf{Assn}\mid\models A\}$是递归可枚举的,则存在一个确认断言是有效的计算方法。根据这个计算方法,可以确认命令c在状态σ_0下不终止,其中σ_0表示所有存储单元X的内容都为零。如同定理7.5中证明的那样,构造断言$w[c,\mathbf{false}]$,设$\vec{X}$由$w[c,\mathbf{false}]$中所有的存储单元组成,A为由0替换存储单元而得到的断言$w[c,\mathbf{false}][\vec{0}/\vec{X}]$。于是,通过检查$A$的有效性,可以确认命令$c$在状态$\sigma_0$下是否不终止。这样我们就得到了一个计算方法。而根据附录中定理A.12,在状态σ_0下不终止的命令c不会构成递归可枚举集,这就推出

了矛盾,从而证得$\{A \in \mathbf{Assn} \mid \models A\}$不是递归可枚举的。 □

定理 7.9 的推论就是哥德尔不完备定理:

定理 7.10(即定理 7.1 的重新陈述,哥德尔不完备性定理) 不存在 **Assn** 的一个能行的证明系统,使得 **Assn** 的有效断言恰好是该系统的定理。

证明 假设存在一个能行的证明系统,使得断言 A 是可证明的当且仅当 A 是有效的。证明系统是能行的蕴涵着:存在一个计算方法以确认有效的断言是定理。系统地搜索所有的证明,直到找到断言 A 的一个证明为止。这样就提供了一个计算方法,当断言 A 有效时,它就确认。于是,不存在一个能行的证明系统。 □

虽然我们介绍了断言 **Assn** 的哥德尔定理,但没有讨论存储单元的基本作用。事实上,哥德尔定理针对的是不含存储单元的较小的断言语言——算术语言。算术语言的有效断言不能构成递归可枚举集,意味着算术公理化永远不能实现(即总存在一些算术事实不可证明)。我们也不能指望得到一个程序,它能够生成无穷的公理和有效的证明规则使得所有算术的有效断言都是可证明的。如果这样的程序存在,那么就得到算术断言的一个能行的证明系统,这与哥德尔不完备性定理是矛盾的。

哥德尔的结论具有深远的历史意义。当时可计算性概念尚未提出,正是哥德尔的结论激励许多逻辑学家开始研究可计算性理论。哥德尔最初的证明用断言本身来表示断言的形式系统的可证明性概念,他构造了一个断言,该断言是有效的当且仅当它是不可证明的。本章只讨论了哥德尔第一不完备性定理,还有哥德尔第二不完备性定理,即算术的形式系统不可能证明自身无矛盾。在哥德尔看来,不完备性的证明实质在于可用断言去表示自然数上特定的函数——*原始递归函数*。原始递归函数的延伸——可计算函数概念的提出则是在几年之后,这个时期最高的成就是丘奇-图灵论题。不完备性定理打破了希尔伯特(Hilbert)制定的计划。当时为了解决数学基础中出现的一些悖论(如罗素悖论),希尔伯特提出了用有限的方法来推导形式系统的设想,藉此希望证明像算术系统那样的一些重要证明系统的一致性和完备性,而哥德尔定理恰恰说明了这种有穷推理方法的局限性。

111

7.4 验证条件

从原理上讲,正是 **Assn** 是可表达的这一事实,使得我们把部分正确性断言有效性的证明,简化为 **Assn** 断言有效性的证明;部分正确性断言$\{A\}\,c\,\{B\}$的有效性等价于消去命令 c 之后的断言 $A \Rightarrow w[c,B]$的有效性。这样,给定一个谓词演算的定理证明器,我们可期望推得 **IMP** 程序的定理证明器。但是,要想推得 $w[c,B]$是一件复杂而低效的工作,因而上述方法是不切实际的。

然而,如果能够结合人工的指导,我们可以用类似的方法求得部分正确性断言有效性证明的自动工具。我们用断言来注释程序,定义有注释的命令的语法集合为:

$$c ::= \mathbf{skip} \mid X := a \mid c_0;(X := a) \mid c_0;\{D\}c_1 \mid$$
$$\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1 \mid \mathbf{while}\ b\ \mathbf{do}\ \{D\}c$$

其中 X 为存储单元,a 为算术表达式,b 为布尔表达式,c,c_0,c_1 为有注释的命令,在 $c_0;\{D\}c_1$ 中,D 为断言,且有注释的命令 c_1 不是赋值命令。其主要思想是:每当控制流图达到命令的注

释点时，该点的断言为真。这样，当控制由 c_0 移动到 c_1 时，我们才对命令 $c_0;c_1$ 进行注释。但是当 c_1 是赋值命令 $X := a$ 时，没有必要这样注释，因为在这种情况下由后置条件可以直接得出注释。有注释的循环命令

$$\textbf{while } b \textbf{ do } \{D\}c$$

112

包含了断言 D，它就是我们期望的不变式。

有注释的部分正确性断言形如

$$\{A\}c\{B\}$$

其中 c 是有注释的命令。顾名思义，有注释的命令就是在原来命令中加上注释而已，有时候把注释的命令视作普通命令反而更加方便。正因为如此，如果与其相关的未注释的部分正确性断言是有效的，则我们说注释的部分正确性断言也是有效的。

有注释的 **while** 循环命令

$$\{A\}\ \textbf{while } b \textbf{ do } \{D\}c\{B\}$$

包含了断言 D，我们希望所选择的 D 是一个不变式。所谓不变式是指

$$\{D \wedge b\}c\{D\}$$

是有效的。如果已知 D 是一个不变式，要保证

$$\{A\}\ \textbf{while } b \textbf{ do } \{D\}c\{B\}$$

是有效的，只需证明断言

$$A \Rightarrow D, \quad D \wedge \neg b \Rightarrow B$$

都是有效的。此时，可以用已知可靠的霍尔规则，从 $\{D \wedge b\}c\{D\}$ 中推导出 $\{A\}$ **while** b **do** $c\{B\}$。显然，并非所有有注释的部分正确性断言都是有效的，不过我们只需要建立某些断言的有效性就足够了，这些断言中所有的命令已经消去，称之为验证条件。施结构归纳于有注释的命令，定义有注释的部分正确性断言的验证条件（简记为 vc）：

$$\begin{aligned}
vc(\{A\}\textbf{skip}\{B\}) &= \{A \Rightarrow B\} \\
vc(\{A\}X := a\{B\}) &= \{A \Rightarrow B[a/X]\} \\
vc(\{A\}c_0;X := a\{B\}) &= vc(\{A\}c_0\{B[a/X]\}) \\
vc(\{A\}c_0;\{D\}c_1\{B\}) &= vc(\{A\}c_0\{D\}) \cup vc(\{D\}c_1\{B\}) \\
&\qquad (\text{其中 } c_1 \text{ 不是赋值命令}) \\
vc(\{A\}\ \textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1\{B\}) &= vc(\{A \wedge b\}c_0\{B\}) \cup vc(\{A \wedge \neg b\}c_1\{B\}) \\
vc(\{A\}\ \textbf{while } b \textbf{ do } \{D\}c\{B\}) &= vc(\{D \wedge b\}c\{D\}) \cup \{A \Rightarrow D\} \\
&\qquad \cup \{D \wedge \neg b \Rightarrow B\}
\end{aligned}$$

113

练习 7.11 施结构归纳于有注释的命令，试证明对于所有有注释的部分正确性断言 $\{A\}c\{B\}$，如果 $vc(\{A\}c\{B\})$ 中所有断言都是有效的，则 $\{A\}c\{B\}$ 也是有效的。（提示：本题的证明类似于引理 7.6 的证明。文献[42]的 3.5 节也给出了一个证明。） □

于是,要证明有注释的部分正确性断言的有效性,只需证明其验证条件是有效的即可。这样,程序验证的任务交给谓词演算的定理证明器来完成。一些商业化的程序验证系统(如 Gypsy [41])就采用了这种方法。

需要注意的是,尽管验证条件的有效性充分保证有注释的部分正确性断言的有效性,但它不是必要条件。这是因为所选择的不变式可能并不适合前置条件和后置条件。例如,虽然

$$\{\mathbf{true}\}\ \mathbf{while\ false\ do}\ \{\mathbf{false}\}\ \mathbf{skip}\ \{\mathbf{true}\}$$

以 **false** 为不变式显然是有效的,但其验证条件包含

$$\mathbf{true} \Rightarrow \mathbf{false}$$

它却不是一个有效的断言。

本节最后说明有注释的命令的一种特殊情况。例如,有两条命令 $(c;X := a_1);X := a_2$ 和 $c;(X := a_1;X := a_2)$,对这两条命令一般都以同样的方式理解,事实上许多命令式语言也都把它们记为:

$$\begin{aligned} &c;\\ &X := a_1;\\ &X := a_2 \end{aligned}$$

但是,根据有注释的命令的语法,它们的注释却是不同的。第一条命令可能的注释只能出现在 c 中,而第二条命令可能被注释为 $c;\{D\}(X := a_1;X := a_2)$。注释规则不是把注释放在单个赋值语句之前,而把注释放在整个赋值语句序列之前。虽然如此,通过一系列的代入,还是可以很方便地由后置条件得出注释。

练习 7.12 试对有注释的命令的语法及其验证条件的定义进行修改以处理上述这种特殊情况,使得赋值命令(或 **skip** 命令)的任何序列和单个赋值命令同样处理。 □

114

练习 7.13 作为一个大作业,试用一种程序设计语言(如标准 ML 或者 Prolog)编写一个验证条件生成器,它以有注释的部分正确性断言作为输入,其输出为一组验证条件。(参见 Gordon 的书[42]中的 Lisp 程序。) □

7.5 谓词转换器

本节是选学的内容,对断言和最弱前置条件进行更抽象的数学描述。我们把命令抽象地表示成从状态 Σ 到状态 $\Sigma_\perp$ 的一个函数 $f:\Sigma\to\Sigma_\perp$,其中 $\Sigma_\perp = \Sigma\cup\{\perp\}$,而 $\perp$ 表示无定义的状态。有时候称这样的函数为*状态转换器*。状态转换器生成完全偏序,与状态上的部分函数的逐点排序的完全偏序是同构的。我们抽象地把部分正确性断言表示成含有 $\perp$ 的状态的子集,而把*部分正确性谓词*定义为

$$\mathrm{Pred}(\Sigma) = \{Q \mid Q \subseteq \Sigma_\perp \ \&\ \perp \in Q\}$$

可以按照 $\supseteq$ 关系把这些谓词排成一个完全偏序,部分正确性谓词的完全偏序为

$$(\mathrm{Pred}(\Sigma),\ \supseteq)$$

这里,如果命令执行能够终止,由命令的格局给出的关于终态的更多信息相应地包含在一个更小的集合内;特别地,如果终态包含的信息量最小,则对应的元素是 $\perp_{Pred} = \Sigma \cup \{\perp\}$。以后,我们用 Pred(Σ)来表示部分正确性谓词的完全偏序。

最弱前置条件的结构确定了谓词的完全偏序的连续函数——谓词转换器[⊖]。

定义 设 $f: \Sigma \to \Sigma_\perp$ 是状态的部分函数,定义

$$Wf: \mathrm{Pred}(\Sigma) \to \mathrm{Pred}(\Sigma);$$
$$(Wf)(Q) = (f^{-1}Q) \cup \{\perp\}$$
$$\text{即},(Wf)(Q) = \{\sigma \in \Sigma \mid f(\sigma) \in Q\} \cup \{\perp\}$$

命令 c 指称状态转换器 $\mathscr{C}[\![c]\!]: \Sigma \to \Sigma_\perp$,按照约定 $\perp$ 表示无定义状态。设 B 是一个断言,对应于解释 I,有

$$(W(\mathscr{C}[\![c]\!]))(B^I) = wp^I[\![c,B]\!]$$

115

练习 7.14 设 ST 表示状态转换器 $[\Sigma_\perp \to_\perp \Sigma_\perp]$ 的完全偏序,PT 表示谓词转换器 $[\mathrm{Pred}(\Sigma) \to \mathrm{Pred}(\Sigma)]$ 的完全偏序。

试证明:$W: ST \to_\perp PT$ 且 W 是连续的(注意! 这里需要检查许多内容)。

试证明:$W(Id_{\Sigma_\perp}) = Id_{Pred(\Sigma)}$,即 W 将完全偏序状态的恒等函数转换为谓词 Pred(Σ)上的恒等函数。

试证明:$W(f \circ g) = (Wg) \circ (Wf)$。 □

迪杰斯特拉在讨论完全正确性时把命令的含义规定为谓词转换器[36]。他认为理解一条命令等于理解确保后置条件为真的最弱前置条件。下面讨论部分正确性的谓词转换器。我们已经有谓词的完全偏序和谓词转换器的完全偏序

$$[\mathrm{Pred}(\Sigma) \to \mathrm{Pred}(\Sigma)]$$

于是不再用状态转换器而用谓词转换器来表示 **IMP** 命令的指称语义,我们定义从命令到谓词转换器的语义函数:

$$\mathscr{P}t: \mathbf{Com} \to [\mathrm{Pred}(\Sigma) \to \mathrm{Pred}(\Sigma)]$$

虽然用谓词转换器表示的命令的指称语义,和前面用部分函数表示的指称语义显然不同,但是如果处理得当则两者是等价的,因为两条命令指称同一个谓词转换器当且仅当它们指称同一个部分函数。读者可以通过做下面的练习来加深理解。

练习 7.15(用谓词转换器表示命令的指称) 语义函数

$$\mathscr{P}t: \mathbf{Com} \to PT$$

定义为:

$$\mathscr{P}t[\![X := a]\!]Q = \{\sigma \in \Sigma_\perp \mid \sigma[\mathscr{A}[\![a]\!]\sigma/X] \in Q\}$$

⊖ 术语"谓词转换器"通常用于完全正确性的相应概念。

$$\mathscr{P}t[\mathbf{skip}]Q = Q$$
$$\mathscr{P}t[c_0;c_1]Q = \mathscr{P}t[c_0](\mathscr{P}t[c_1]Q)$$
$$\mathscr{P}t[\mathbf{if}\ b\ \mathbf{then}\ c_0\ \mathbf{else}\ c_1]Q = \mathscr{P}t[c_0](\bar{b} \cap Q) \cup \mathscr{P}t[c_1](\overline{\neg b} \cap Q)$$

其中,对任意布尔表达式 b,有 $\bar{b} = \{\sigma \mid \sigma = \perp$ 或者 $\mathscr{B}[b]\sigma = \mathbf{true}\}$

$$\mathscr{P}t[\mathbf{while}\ b\ \mathbf{do}\ c] = fix(G)$$

116 其中,函数 $G: PT \to PT$ 由 $G(p)(Q) = (\bar{b} \cap \mathscr{P}t[c](p(Q)) \cup (\overline{\neg b} \cap Q)$ 确定。

1) 试证明 G 是连续的.

2) 试证明对于任意的命令 c,有 $W(\mathscr{C}[c]_\perp) = \mathscr{P}t[c]$。

3) 试验证对于定义在 $\Sigma_\perp$ 上的两个严格连续函数 f, f',有

$$Wf = Wf' \Rightarrow f = f'$$

4) 试推导

$$\mathscr{C}[c] = \mathscr{C}[c'] \text{ 当且仅当 } \mathscr{P}t[c] = \mathscr{P}t[c']$$

其中 c, c' 为任意命令。

读者回忆一下谓词之间的序关系,因为它是逆包含关系,所以有

$$fix(G) = \bigcap_{n \in \omega} G^n(\perp_{Pred})$$

这说明:如果允许断言语言进行无穷的交运算并且不含量词,那么就能直接表示最弱前置条件。确实如此,我们还可以用无穷的交运算扩充 **Bexp** 而得到另一类断言以代替 **Assn**,通过修改上述语义,可以给出一种新的断言来表示每条命令的最弱前置条件。总之,只要断言是可表达的,我们就可以用 7.2 节的方法来证明新断言的相对完备性。 □

7.6 进一步阅读资料

Crossley 的《什么是数理逻辑》(*What is mathematical logic*?)[34]清晰地阐述了哥德尔不完备性定理。更详细的讨论见 Kleene[54]、Mendelson[61]和 Enderton[38]的逻辑学教材。Kfoury、Moll 和 Arbib 的合著[11]面向计算机科学专业的学生阐述了哥德尔定理。Cook 在[33]中采用"最强后置条件"完成了相对完备性的证明,而 Clarke 在文献[23]和早期的著作中用"最弱前置条件"来证明相对完备性。此外,Clarke 的论文认为,对于比本章所给出的功能更强的程序设计语言,不可能存在可靠的相对完备的证明系统,从而对研究产生了一些负面的影响。Apt 的论文[8]给出了一个很好的研究方向。文献[58]和[13]对本章的内容作
117 了另外的阐述。Gordon 的著作[42]给出了验证条件的基本而详尽的讨论。

第8章 域 论

域论是指称语义的数学基础。本章继续讨论完全偏序(域)、连续函数以及完全偏序上的构造,它们对于程序设计语言的数学描述是很重要的,并为指称语义提供了数学基础。最后介绍支持语义定义的元语言,元语言定义的函数都是连续的。

8.1 基本定义

在指称语义中,通过将给定的语义"域"中的元素赋予程序设计的构造(例如,一条命令或一个表达式),构造就获得了语义。我们称程序设计的构造指称该元素,也称该元素是程序构造的一个指称。例如,**IMP** 的命令由部分函数"域"中的元素来指称,而 **IMP** 的数指称数集 **N** 中的元素。第5章讨论 **IMP** 的指称语义时很清楚,"域"中的结构必须足够多以便能解递归方程。第5章介绍的完全偏序作为支持递归定义的结构,这些完全偏序有理由成为语义"域"的侯选。当然,完全偏序是否合适还要看它对各种程序设计语言的适用范围以及表示的结果与操作语义的关系。经验和研究表明完全偏序是十分重要的,虽然有时候还要在完全偏序中加入新的结构,但是以完全偏序为基础的理论都有能力定义程序设计语言的复合[⊖]语义。首先,我们回忆一下第5章的一些定义。

定义 如果 D 中元素的任一 ω 链 $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$ 都有在 D 中的最小上界 $\bigsqcup_{n\in\omega} d_n$,则偏序 $(D, \sqsubseteq)$ 称为*完全偏序*(简记为 cpo)。

如果它有一个最小元 $\perp$(称为底),则称完全偏序 $(D, \sqsubseteq)$ 是*含底的* cpo[⊖]。

有时候,把完全偏序记为 $(D, \sqsubseteq_D)$,且明确表示关系 $\sqsubseteq_D$ 和底元素 $\perp_D$ 属于哪一个完全偏序。然而,如果根据上下文不会引起歧义,我们一般把它们记作 $\sqsubseteq$ 和 $\perp$。通常,如果上下文中意义明确,我们还把 $\bigsqcup_{n\in\omega} d_n$ 记为 $\bigsqcup_n d_n$。 119

第5章已经介绍了完全偏序的几个例子。

例

(i) 按恒等关系排序的集合是一个*离散的*完全偏序。

(ii) 任何集合 X 的幂集 $\mathscr{P}ow(X)$(按 $\supseteq$ 或 $\subseteq$ 排序)组成的集合是完全偏序,实际上任何完全格都是完全偏序(见5.5节)。

(iii) 两个元素($\perp$ 和 T)的完全偏序 $\perp \sqsubseteq$ T 称为 **O**,这样的次序来自按 $\subseteq$ 排序的单元素集合的幂集。

(iv) 按照包含关系排序的集合 X, Y 间的部分函数集合 $X \rightharpoonup Y$ 是一个完全偏序。

⊖ 在第5章中,如果程序表达式的含义用它的直接子表达式的含义解释,称语义是复合的。

⊖ 这里 cpo 一般称为(无底的)ω-cpo(或前域)。

(v) 用∞扩充非负整数 ω,将它们按 $\sqsubseteq$ 次序形成一条链

$$0 \sqsubseteq 1 \sqsubseteq \cdots \sqsubseteq n \sqsubseteq \cdots \infty$$

得到一个完全偏序(记为 Ω)。 □

要进行递归定义,仅仅靠完全偏序还不够,只有确保完全偏序之间的函数保持 ω 链的最小上界我们才能支持递归定义。

定义 称定义在完全偏序 D 和 E 之间的函数 $f: D \to E$ 是单调的当且仅当

$$\forall d, d' \in D.\ d \sqsubseteq d' \Rightarrow f(d) \sqsubseteq f(d')$$

这样的一个函数是连续的当且仅当它是单调的且对 D 中所有的链 $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$,有

$$\bigsqcup_{n \in \omega} f(d_n) = f(\bigsqcup_{n \in \omega} d_n)$$

例

(i) 从离散完全偏序到完全偏序的所有函数都是连续的。

(ii) 设 Ω 和 $\mathbf{O}$ 是上面例子中定义的完全偏序,对 $n \in \Omega$,定义函数 $f_n: \Omega \to \mathbf{O}$ 为

$$f_n(x) = \begin{cases} \top & n \sqsubseteq x \\ \bot & \text{其他} \end{cases}$$

连续函数 $\Omega \to \mathbf{O}$ 由取值恒为 $\bot$ 的常函数(即 $\lambda x.\ \bot$)和所有的 $f_n (n \in \omega)$ 组成。然而,f_∞ 却不是连续的。(请读者思考为什么?) □

120 **命题 8.1** *完全偏序 D 上的恒等函数 Id_D 是连续的。设函数 $f: D \to E$ 和 $g: E \to F$ 是完全偏序 D、E、F 上的连续函数,则它们的复合函数 $g \circ f: D \to F$ 也是连续的。*

练习 8.2 证明命题 8.1。 □

5.4 节我们证明了含底 $\bot$ 的完全偏序的一个重要性质,即定义在其上的任一连续函数都有一个最小不动点。

定理 8.3(不动点定理) *设 $f: D \to D$ 是含底 $\bot$ 的完全偏序 D 上的连续函数。定义*

$$fix(f) = \bigsqcup_{n \in \omega} f^n(\bot)$$

则 $fix(f)$ 是 f 的一个不动点,且是 f 的最小前缀不动点,即(i) $f(fix(f)) = fix(f)$,(ii) 如果 $f(d) \sqsubseteq d$,则 $fix(f) \sqsubseteq d$。因此 $fix(f)$ 是 f 的最小不动点。 □

8.2 一个例子——流

在第 5 章中,我们从有限规则的归纳定义的角度出发,通过对用于获得集合算子的最小不动点的性质进行抽象,讨论了完全偏序和连续函数。我们知道,假定操作语义通常表示成一组有限的规则,则不难理解连续性和计算之间存在关联。下面看一个对序列进行计算的例子,这个例子更直接地说明了连续性的作用。

该计算的输入是由 0 和 1 组成的有限或者无限序列，有限序列可以（但不是必须）以特殊符号 $ 表示结束。它的思想主要是：序列表示可能的输入，它可能来自另一个计算的结果，也可能直接来自用户；允许用 $ 符号明确表示序列已经结束；除非用 $ 强制终止，否则序列的长度可以无限制地增长；序列在未终止的情况下可以维持有限，此时有可能是输入设备坏了，有可能进入了一个不终止的计算，也有可能用户在输入下一个序列元素或者终止符 $ 之前，因为某种原因而没有输入。

这样的序列有时称为“流”、“惰性表”或者“带终止符的序列”（$ 是终止符），它们之间有一种直观的偏序关系。如果序列 s 是另一个序列 s' 的前缀部分，则称 s 位于 s' 之下。随着序列所包含的信息一点点地增加，对应于这种偏序关系，序列也在不断地递增。对应于这种偏序，存在一个最小的序列，即空序列 ε。最大的序列是含终止符的序列，如

$$0101\ \$$$

[121]

以及无限序列，如

$$000\cdots00\cdots$$

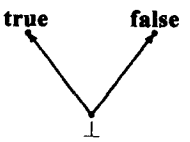
简记为 0^{ω}。实际上，这些序列构成了一个含底元素 ε 的完全偏序，称之为完全偏序 S。

设想我们要检查输入中是否包含有 1。可能会构造一个函数

$$isone: S \rightarrow \{\mathbf{true}, \mathbf{false}\}$$

对于一个输入序列，如果它包含 1，则函数返回 **true**，否则返回 **false**。这个方法其实不可行。如果在某个阶段序列不包含 1，但在后面的阶段包含 1，例如，序列从空序 ε 开始，后来变成 10 了，会发生什么情况呢？在这种情况下，我们必须将原来的输出值由 **false** 改成 **true** 吗？我们知道，当对某一输入函数 *isone* 返回 **false** 时，所表达的含义仅仅是当前阶段不含 1。然而，*isone*(000 $) = **false**，因为序列已经终止，不可能再出现 1，而 *isone*(000) 和 **false** 有区别，当然它不应该为 **true**。可以有两种选择：把 *isone* 定义为部分函数，或者除 **true** 和 **false** 外，引入表示无定义的元素“don't know”。我们采用第二种方案。

随着输入序列的增加，“don't know”可能修改为 **true** 也可能修改为 **false**，我们用 $\perp$ 表示“don't know”，它位于 **true** 和 **false** 之下，如图所示：



用 $\{\mathbf{true}, \mathbf{false}\}_{\perp}$ 表示这个含有最小元 $\perp$ 的简单的完全偏序。这样，输出的信息更能够反映输入的信息。用数学术语表示，*isone* 是一个从 S 到 $\{\mathbf{true}, \mathbf{false}\}_{\perp}$ 的单调函数。

但是，判定

$$isone: S \rightarrow \{\mathbf{true}, \mathbf{false}\}_{\perp}$$

的单调性并不能保证它一定是一个函数，即使我们对任一序列 s 作如下限制：

$$\begin{aligned} isone(1s) &= \mathbf{true} & isone(\$) &= \mathbf{false} \\ isone(0s) &= isone(s) & isone(\varepsilon) &= \perp \end{aligned}$$

因为加以限制后，我们仍然不知道 $isone(0^\omega)$ = **false** 还是 $isone(0^\omega) = \perp$。然而，从计算的角度看，$isone(0^\omega)$ = **false** 是不可行的，因为在输出 **false** 之前要检查无限序列并报告序列始终不含 1。如果取 $isone(0^\omega)$ 为 **false**，则得到一个不连续的函数，这也说明了 $isone(0^\omega)$ 是不可计算的。0^ω 的任一有限子序列为由 n 个 0 组成的 0^n，而 0^ω 的无限序列是最小上界 $\bigsqcup_{n\in\omega}0^n$。我们取 $isone(0^n) = \perp$，于是

122

$$\bigsqcup_{n\in\omega} isone(0^n) = \perp$$

由连续性，只能取 $isone(0^\omega) = \perp$。

练习 8.4 可以把完全偏序看作拓扑空间，而把连续函数看作传统拓扑意义中连续的函数（本题不要求有专门的拓扑学知识）。给定一个完全偏序 $(D, \sqsubseteq)$，定义拓扑（称为斯科特拓扑，根据 Dana Scott 的姓氏命名）如下。称 $U \sqsubseteq D$ 是开的当且仅当

$$\forall d, e \in D.\ d \sqsubseteq e \ \&\ d \in U \Rightarrow e \in U$$

且对于 D 中所有的链 $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$，有

$$\bigsqcup_{n\in\omega} d_n \in U \Rightarrow \exists n \in \omega.\ d_n \in U$$

(i) 试证明上述定义确定了完全偏序 D 的一个拓扑（即，$\emptyset$ 和 D 本身是开的，且任意开集合经过有限次交运算后得到的集合也是开的，且任意开集合的并集也是开的）。

(ii) 试证明对于完全偏序 D 的任意元素 d，集合 $\{x \in D \mid x \not\sqsubseteq d\}$ 是开的。

(iii) 试证明 $f: D \to E$ 是从完全偏序 D 到 E 的连续函数，当且仅当 f 是拓扑连续的。（这样的函数 f 是拓扑连续的，当且仅当对于 E 的任意开集 V，其逆函数 $f^{-1}V$ 是 D 的开集。）

(iv) 试证明完全偏序 D 的开集通常可以表示为连续函数 $f: D \to \mathbf{O}$ 的关于 $\top$ 的逆象集 $f^{-1}\{\top\}$ 的那些集合。试描述本节介绍的特殊的流的完全偏序的开集。 □

8.3 完全偏序上的构造

构造完全偏序的方法有多种，正因如此，完全偏序才能作为各种程序设计构造的语义域。本节介绍完全偏序上的各种构造，以及与之相关的特定的连续函数。后面讨论程序设计语言的指称语义时会用到本节的内容。

123

有时候，在给出构造时，我们不必明确指出构造内建立了什么集合；有很多方法可以得到“实质上相同”的构造。正如第 1 章介绍基本集合论时，我们有很多途径定义集合的积，采用哪一种途径取决于序偶的实现方法；在求不相交的并集时，我们先求不相交的集合再求并，这是几种方法中的一种方法。本节采用更抽象的方法来讨论构造。例如，要计算完全偏序的和 $D_1 + \cdots + D_k$，只需简单地把每个完全偏序 $D_1, \cdots, D_k$ 并放在一起就可以了，这样，我们就只需假设存在一个一一对应的函数 $in_i (1 \leq i \leq k)$，且确保如果 $l \neq m$ 则元素 $in_l(d_l)$ 和 $in_m(d_m)$ 不同。当然，重要的是要知道这样的函数确实存在，这里，它们确实存在的，因为可以用 (i, x) 来实现 $in_i(x)$。这种更抽象的方法是可行的，因为“和构造”是实质上相同的，只要它们满足不同的条件，则不管用什么方法来实现 in_i。

数学上用同构的概念来表示“实质上相同”的结构。称完全偏序 D 和 E 之间的连续函数 $f: D \to E$ 是同构的，如果存在连续函数 $g: E \to D$，使得 $g \circ f = Id_D$ 且 $f \circ g = Id_E$——g 和 f 互为逆函数。这实际上是一般定义的实例，它们应用到一类对象以及对象之间的函数（例如，完全偏序及其连续函数）。由定义可知，同构的完全偏序是“实质上相同”的，只不过对元素的命名不同而已。

命题 8.5　设 $(D, \sqsubseteq_D)$ 和 $(E, \sqsubseteq_E)$ 是两个完全偏序。函数 $f: D \to E$ 是同构，当且仅当对所有的 $x, y \in D$，f 是一一对应的，使得

$$x \sqsubseteq_D y \text{ 当且仅当 } f(x) \sqsubseteq_E f(y)$$

8.3.1　离散完全偏序

最简单的完全偏序是以恒等关系为偏序的集合。此时，ω 链是常量。称以恒等关系为偏序的完全偏序是离散的。语法集合以及像真值、整数等基本值都可以构成离散完全偏序。需要指出的是，从离散完全偏序到完全偏序的任一函数都是连续的（特别地，语法集合的语义函数是连续的）。

练习 8.6　试指出从含底 $\perp$ 的完全偏序到离散完全偏序上的哪些函数是连续的？　□　124

8.3.2　有限积

设 $D_1, \cdots, D_k$ 是完全偏序，它们的积

$$D_1 \times \cdots \times D_k$$

由 k 元组 $(d_1, \cdots, d_k)$ 组成，其中 $d_1 \in D_1, \cdots, d_k \in D_k$。偏序是由它们的“坐标”确定的，即

$$(d_1, \cdots, d_k) \sqsubseteq (d_1', \cdots, d_k') \text{ 当且仅当 } d_1 \sqsubseteq d_1' \ \& \cdots \& \ d_k \sqsubseteq d_k'$$

容易看出，对 $n \in \omega$，积的 ω 链 $(d_{1n}, \cdots, d_{kn})$ 是由“坐标”计算得到的最小上界：

$$\bigsqcup_{n \in \omega} (d_{1n}, \cdots, d_{kn}) = (\bigsqcup_{n \in \omega} d_{1n}, \cdots, \bigsqcup_{n \in \omega} d_{kn})$$

所以，完全偏序的积本身也是一个完全偏序。下面介绍与完全偏序的积 $D_1 \times \cdots \times D_k$ 相关的几个重要函数。

投影函数 $\pi_i: D_1 \times \cdots \times D_k \to D_i$（其中 $i = 1, \cdots, k$）选择元组的第 i 坐标：

$$\pi_i(d_1, \cdots, d_k) = d_i$$

因为链的最小上界是以坐标方式计算得到的，所以容易得到投影函数是连续的。

可以把元组构造扩充到函数上。设 $f_1: E \to D_1, \cdots, f_k: E \to D_k$ 都是连续函数。定义函数

$$\langle f_1, \cdots, f_k \rangle : E \to D_1 \times \cdots \times D_k$$

并取

$$\langle f_1, \cdots, f_k \rangle (e) = (f_1(e), \cdots, f_k(e))^{\ominus}$$

⊖ 原文为 $f_n(e)$；疑误。——译者注

显然，函数$\langle f_1,\cdots,f_k\rangle$满足性质

$$\pi_i\circ\langle f_1,\cdots,f_k\rangle = f_i \quad (\text{其中 } i=1,\cdots,k)$$

实际上，$\langle f_1,\cdots,f_k\rangle$是从$E$到$D_1\times\cdots\times D_k$上具有该性质的惟一函数，而且很容易看出，它是单调的。同时，由于对E的任意ω链$e_0\sqsubseteq e_1\sqsubseteq\cdots\sqsubseteq e_n\sqsubseteq\cdots$，有

$$\begin{aligned}\langle f_1,\cdots,f_k\rangle(\textstyle\bigsqcup_{n\in\omega}e_n) &= (f_1(\textstyle\bigsqcup_{n\in\omega}e_n),\cdots,f_k(\textstyle\bigsqcup_{n\in\omega}e_n)) \quad (\text{由定义})\\ &= (\textstyle\bigsqcup_{n\in\omega}f_1(e_n),\cdots,\textstyle\bigsqcup_{n\in\omega}f_k(e_n))\\ &\quad (\text{因为每个 } f_i \text{ 都是连续的})\\ &= \textstyle\bigsqcup_{n\in\omega}(f_1(e_n),\cdots,f_k(e_n))\\ &\quad (\text{因为积的最小上界是由每一个 cpo 的坐标确定的})\\ &= \textstyle\bigsqcup_{n\in\omega}\langle f_1,\cdots,f_k\rangle(e_n)\end{aligned}$$

125 所以，函数$\langle f_1,\cdots,f_k\rangle$是连续的。

我们也可以把完全偏序的积构造扩充到函数上。对于$f_1:D_1\to E_1,\cdots,f_k:D_k\to E_k$，定义函数

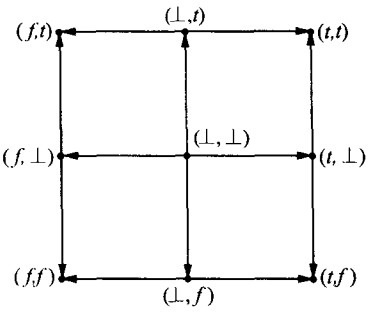
$$f_1\times\cdots\times f_k:D_1\times\cdots\times D_k\to E_1\times\cdots\times E_k$$

并取

$$f_1\times\cdots\times f_k(d_1,\cdots,d_k) = (f_1(d_1),\cdots,f_k(d_k))$$

换言之$f_1\times\cdots\times f_k=\langle f_1\circ\pi_1,\cdots,f_k\circ\pi_k\rangle$。每一个分量$f_i\circ\pi_i$都是连续的，连续函数的复合也是连续的，所以元组$\langle f_1\circ\pi_1,\cdots,f_k\circ\pi_k\rangle$也是连续的，因此，函数$f_1\times\cdots\times f_k$是连续函数。

例 作为完全偏序的积的一个例子，我们考察$\mathbf{T}_\perp\times\mathbf{T}_\perp=\mathbf{T}_\perp^2$，如下图所示：



图中t和f分别表示真值 **true** 和 **false**。 □

练习 8.7 试画出积$\mathbf{O}^0,\mathbf{O}^1,\mathbf{O}^2$和$\mathbf{O}^3$。 □

积有两个容易证明但十分重要的性质，第一个性质是拓扑中普通事实的一个实例，第二个性质对后面的讨论十分重要。

引理 8.8 设$h:E\to D_1\times\cdots\times D_k$是从完全偏序$E$到完全偏序的积的函数，则函数$h$是连续的，当且仅当对所有的$i(1\leqslant i\leqslant k)$，函数$\pi_i\circ h:E\to D_i$是连续的。

证明

必要性:连续函数的复合函数是连续的。

充分性:设对所有的 $i(1\leqslant i\leqslant k)$,函数 $\pi_i\circ h$ 是连续的,则对任一 $x\in E$,有

$$\begin{aligned}h(x)&=(\pi_1(h(x)),\cdots,\pi_k(h(x)))\\&=(\pi_1\circ h(x),\cdots,\pi_k\circ h(x))\\&=\langle\pi_1\circ h,\cdots,\pi_k\circ h\rangle(x)\end{aligned}$$

因为每一个 $\pi_i\circ h$ 都是连续的,所以 $h=\langle\pi_1\circ h,\cdots,\pi_k\circ h\rangle$ 也是连续的。 □ 126

第二个更有用的引理依赖于序关系,其证明用到了完全偏序的元素"阵列"的最小上界的重要结论。

命题 8.9 对 $n,m\in\omega$,设 $e_{n,m}$ 是完全偏序 E 的具有如下性质的元素:如果 $n\leqslant n'$ 且 $m\leqslant m'$,则 $e_{n,m}\sqsubseteq e_{n',m'}$,则集合 $\{e_{n,m}\mid n,m\in\omega\}$ 有最小上界

$$\bigsqcup_{n,m\in\omega}e_{n,m}=\bigsqcup_{n\in\omega}(\bigsqcup_{m\in\omega}e_{n,m})=\bigsqcup_{m\in\omega}(\bigsqcup_{n\in\omega}e_{n,m})=\bigsqcup_{n\in\omega}e_{n,n}$$

证明 要证命题成立,只要证明下列集合

$$\{e_{n,m}\mid n,m\in\omega\},\quad\{\bigsqcup_{m\in\omega}e_{n,m}\mid n\in\omega\},$$
$$\{\bigsqcup_{n\in\omega}e_{n,m}\mid m\in\omega\},\quad\{e_{n,n}\mid n\in\omega\}$$

有相同的上界,因此有相同的最小上界。例如,由于任一元素 $e_{n,m}$ 受 $e_{n,n}$ 支配,所以,$\{e_{n,m}\mid n,m\in\omega\}$ 和 $\{e_{n,n}\mid n\in\omega\}$ 有相同的上界。显然,ω 链 $\bigsqcup_n e_{n,n}$ 的最小上界存在,于是,最小上界 $\bigsqcup_{n,m}e_{n,m}$ 也存在并且两者相等。$\{\bigsqcup_m e_{n,m}\mid n\in\omega\}$ 的任一上界必定是 $\{e_{n,m}\mid n,m\in\omega\}$ 的上界;反之,对任一 $m\in\omega$,$\{e_{n,m}\mid n,m\in\omega\}$ 的任一上界支配任一最小上界 $\bigsqcup_m e_{n,m}$,于是,$\{e_{n,m}\mid n,m\in\omega\}$ 和 $\{\bigsqcup_{m\in\omega}e_{n,m}\mid n\in\omega\}$ 共享相同的上界,因而有相等的最小上界。$\bigsqcup_m(\bigsqcup_n e_{n,m})=\bigsqcup_{n,m}e_{n,m}$ 的证明与此类似。 □

引理 8.10 设 $f:D_1\times\cdots\times D_k\to E$ 是一个函数,则 f 是连续的,当且仅当 f"在每个变量上分别连续",即,对于所有的 $i(1\leqslant i\leqslant k)$,对任一 $d_1,\cdots,d_{i-1},d_{i+1},\cdots,d_k$,由 $d_i\longmapsto f(d_1,\cdots,d_i,\cdots,d_k)$ 确定的函数 $D_i\to E$ 是连续的。

证明

"⇒"显然成立。(请读者思考为什么?)

"⇐"为了符号上方便,设 $k=2$(不过很容易将证明扩展到更多的变量),令 $(x_0,y_0)\sqsubseteq\cdots\sqsubseteq(x_n,y_n)\sqsubseteq\cdots$ 是积 $D_1\times D_2$ 的一条链。于是

$$\begin{aligned}f(\bigsqcup_n(x_n,y_n))&=f(\bigsqcup_p x_p,\bigsqcup_q y_q)&&(\text{因为最小上界是坐标确定的})\\&=\bigsqcup_p f(x_p,\bigsqcup_q y_q)&&(\text{因为 }f\text{ 在第一个变量上连续})\\&=\bigsqcup_p\bigsqcup_q f(x_p,y_q)&&(\text{因为 }f\text{ 在第二个变量上连续})\end{aligned}$$

127

$$= \bigsqcup_n f(x_n, y_n) \qquad (\text{由命题 8.9})$$

从而 f 是连续的。 □

本引理的证明思想非常重要;我们要多次用到这个思想:要证明积的某个函数连续,先证明它在每个变量上分别连续。[⊖]

有限积的一种退化是只由一个空元组( )组成的空积{( )},以后用 **1** 表示空积。

8.3.3 函数空间

设 D 和 E 是完全偏序,则从 D 到 E 的所有连续函数的集合能构成一个完全偏序。这一点非常重要。函数空间 $[D \to E]$ 由元素

$$\{f \mid f: D \to E \text{ 是连续的}\}$$

组成,并且元素之间由

$$f \sqsubseteq g \text{ 当且仅当 } \forall d \in D. f(d) \sqsubseteq g(d)$$

逐点排序。

这样的函数空间是一个完全偏序。需要注意的是,如果 E 含有底元素 $\perp_E$,则完全偏序的函数空间也有一个底元素,即值恒为 $\perp_E$ 的函数 $\perp_{[D \to E]}$,对所有的 $d \in D$,有

$$\perp_{[D \to E]}(d) = \perp_E$$

函数链的最小上界也是逐点给出的,即函数链

$$f_0 \sqsubseteq f_1 \sqsubseteq \cdots \sqsubseteq f_n \sqsubseteq \cdots$$

有最小上界 $\bigsqcup_{n \in \omega} f_n$,对于 $d \in D$,有

128

$$(\bigsqcup_n f_n)(d) = \bigsqcup_n (f_n(d))$$

函数链的最小上界是 $[D \to E]$ 中的一个函数,这个事实需要我们检验它的连续性。

设 $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_m \sqsubseteq \cdots$ 是 D 的一条链,则

$$\begin{aligned}
(\bigsqcup_n f_n)(\bigsqcup_m d_m) &= \bigsqcup_n f_n(\bigsqcup_m d_m) && (\text{由函数的最小上界的定义}) \\
&= \bigsqcup_n (\bigsqcup_m f_n(d_m)) && (\text{因为每个 } f_n \text{ 是连续的}) \\
&= \bigsqcup_m (\bigsqcup_n f_n(d_m)) && (\text{由命题 8.9}) \\
&= \bigsqcup_m ((\bigsqcup_n f_n)(d_m)) && (\text{由函数的最小上界的定义})
\end{aligned}$$

⊖ 与引理 8.10 相对应的性质,对实数分析和复数分析中的函数不成立,它更多地利用函数的几个变量上的连续性验证。例如:

$$p(x) = \begin{cases} \dfrac{xy}{x^2 + y^2} & (x, y) \neq (0,0) \\ 0 & x = y = 0 \end{cases}$$

对集合 I 和完全偏序 D,特殊的函数空间$[I \to D]$称为幂空间,记作 D^I。完全偏序 D^I 的元素可以看成是坐标排序的元组$(d_i)_{i \in I}$(如果 I 是无限集合,则元组也是无限的)。当 I 是有限集合$\{1,2,\cdots,k\}$时,则完全偏序 D^I 和完全偏序的积 $D \times \cdots \times D$ 是同构的,我们把 k 个完全偏序 D 的积记为 D^k。

有两个重要的与函数空间构造有关的操作,应用 (application) 和柯灵(currying)[⊖]。定义

$$apply : [D \to E] \times D \to E$$

为 $apply(f,d) = f(d)$。根据引理 8.10,因为函数 $apply$ 在每个变量上都是连续的,所以它也是连续的:

设$f_0 \sqsubseteq \cdots \sqsubseteq f_n \sqsubseteq \cdots$是一条函数链,则

$$\begin{aligned} apply(\textstyle\bigsqcup_n f_n, d) &= \textstyle\bigsqcup_n f_n(d) && \text{(因为最小上界是逐点给出的)} \\ &= \textstyle\bigsqcup_n apply\ (f_n, d) && \text{(由 } apply \text{ 的定义)} \end{aligned}$$

设 $d_0 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$是 D 的一条链,则

$$apply(f, \bigsqcup_n d_n) = f(\bigsqcup_n d_n) = \bigsqcup_n f(d_n) = \bigsqcup_n apply(f, d_n)$$

设 F 是一个完全偏序,且函数

$$g : F \times D \to E$$

是连续的。定义

$$curry(g) : F \to [D \to E]$$

129

为函数

$$curry(g) = \lambda v \in F \lambda d \in D.\, g(v,d)$$

因此,$(curry(g))(v)$是函数,其作用使 $d \in D$ 变成 $g(v,d)$。把 $curry(g)$记为 h,则对于任意的 $v \in F, d \in D$,我们有

$$(h(v))(d) = g(v,d)$$

显然,我们要检验每个 $h(v)$是否连续,且 $curry(g)$本身是否为 F 到$[D \to E]$的连续函数。

首先,设 $v \in F$,我们先来证明函数 $h(v) = \lambda\, d \in D.\, g(v,d)$是连续的。因为 g 是连续的,所以 g 在每个变量上分别连续,从而 $h(v)$是连续函数。其次,设

$$v_0 \sqsubseteq v_1 \sqsubseteq \cdots \sqsubseteq v_n \sqsubseteq \cdots$$

是 F 的元素构成的一条 ω 链,又设 $d \in D$,则

⊖ 柯灵操作是以美国逻辑学家 Haskell Curry 的名字命名的。

$$
\begin{aligned}
h(\bigsqcup_n v_n)(d) &= g(\bigsqcup_n v_n, d) && (\text{由 } h \text{ 的定义}) \\
&= \bigsqcup_n g(v_n, d) && (\text{由 } g \text{ 的连续性}) \\
&= \bigsqcup_n (h(v_n)(d)) && (\text{由 } h \text{ 的定义}) \\
&= (\bigsqcup_n h(v_n))(d) && (\text{由函数链的最小上界的定义})
\end{aligned}
$$

于是，$h(\bigsqcup_n v_n) = \bigsqcup_n h(v_n)$，所以 h 是连续的。事实上，$curry(g)$ 是从 F 到 $[D \to E]$ 的惟一连续函数，使得对所有的 $v \in F, d \in D$，有

$$apply(h(v), d) = g(v, d)$$

练习 8.11 对完全偏序 D 和集合 I，幂空间 D^I 是一个可能为无限的积，其元素是按坐标排序的元组 $(d_i)_{i \in I}$（如果 I 是无限集合，则元组也是无限的）。根据引理 8.10，如果函数在每个变量上都是连续的，那么这个函数也是连续的。试证明该结论在幂空间中不成立：如果 I 是无限集合，那么即使 D^I 的函数在每个变量上都是分别连续的，这个函数也不一定连续。（提示：考虑函数 $\mathbf{O}^\omega \to \mathbf{O}$。）□

130

8.3.4 提升

我们曾经给集合加上一个元素 $\perp$，从而得到一个含有底元素的完全偏序（例如 5.4 节，在状态集合中加上无定义的状态 $\perp$，得到完全偏序 $\Sigma_\perp$）。这种方法称为提升，是一种很有用的构造方法。简单地说，提升就是给原有的完全偏序加上一个底元素。

设 D 是一个完全偏序。提升构造假设元素 $\perp$ 和函数 $\lfloor - \rfloor$ 具有性质

$$
\begin{aligned}
&\lfloor d_0 \rfloor = \lfloor d_1 \rfloor \Rightarrow d_0 = d_1, \text{且} \\
&\perp \neq \lfloor d \rfloor
\end{aligned}
$$

其中所有的 $d, d_0, d_1 \in D$。提升后的完全偏序 $D_\perp$ 具有集合

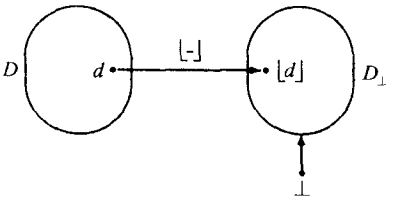
$$D_\perp = \{\lfloor d \rfloor \mid d \in D\} \cup \{\perp\}$$

和偏序

$$
\begin{aligned}
&d_0' \sqsubseteq d_1' \text{ 当且仅当 } (d_0' = \perp) \text{ 或者} \\
&(\exists d_0, d_1 \in D.\ d_0' = \lfloor d_0 \rfloor \ \&\ d_1' = \lfloor d_1 \rfloor \ \&\ d_0 \sqsubseteq_D d_1)
\end{aligned}
$$

从而可以推出，$D_\perp$ 中 $\lfloor d_0 \rfloor \sqsubseteq \lfloor d_1 \rfloor$ 当且仅当 $d_0 \sqsubseteq d_1$，所以在完全偏序 D 中引入底元素 $\perp$ 而得到 $D_\perp$。函数 $\lfloor - \rfloor : D \to D_\perp$ 显然是连续的。虽然实现 $\lfloor - \rfloor$ 和 $\perp$ 有多种方法，但这些方法得到同构的构造。

完全偏序 D 的提升构造图示为：



设 D 是完全偏序，E 是含有底元素的完全偏序，连续函数 $f: D \to E$ 可以扩充为连续函数

$$f^*: D_\perp \to E$$

方法是定义

$$f^*(d') = \begin{cases} f(d) & \text{如果对于某些 } d \in D \text{ 有 } d' = \lfloor d \rfloor \\ \perp & \text{其他} \end{cases}$$

131

设函数 f 由 λ 表达式 $\lambda x.e$ 描述，将函数 f^* 作用于 $d' \in D_\perp$ 得到

$$(\lambda x.e)^*(d')$$

我们把它记为

$$let\ x \Leftarrow d'.\ e$$

这个记号说明，只有当 d' 是非 $\perp$ 时，它才确定 e 的结果，否则结果为 $\perp_E$。

运算 $(-)^*$ 是连续的：设 d' 是 $D_\perp$ 的任意元素，$f_0 \sqsubseteq \cdots \sqsubseteq f_n \sqsubseteq \cdots$ 是 $[D \to E]$ 上函数的 ω 链。在 $d' = \perp$ 的情况下，我们直接可以推出 $(\bigsqcup_n f_n)^*(d')$ 和 $(\bigsqcup_n f_n^*)(d')$ 都为 $\perp_E$。否则 $d' = \lfloor d \rfloor$ 且

$$\begin{aligned}
(\bigsqcup_n f_n)^*(d') &= (\bigsqcup_n f_n)(d) && \text{（由 } (-)^* \text{ 的定义）} \\
&= \bigsqcup_n (f_n(d)) && \text{（因为最小上界是逐点确定的）} \\
&= \bigsqcup_n ((f_n^*)(d')) && \text{（由 } (-)^* \text{ 的定义）} \\
&= (\bigsqcup_n f_n^*)(d') && \text{（因为最小上界是逐点确定的）}
\end{aligned}$$

由于 d' 是任意的，于是，可以得到 $(\bigsqcup_n f_n)^* = \bigsqcup_n (f_n^*)$，即操作 $(-)^*$ 是连续的。

以后我们把

$$let\ x_1 \Leftarrow c_1.\ (let\ x_2 \Leftarrow c_2.\ (\cdots (let\ x_k \Leftarrow c_k.\ e) \cdots))$$

简记为

$$let\ x_1 \Leftarrow c_1, \cdots, x_k \Leftarrow c_k.\ e$$

用 *let* 记号可以把集合的运算扩充为提升 $S_\perp$。例如，在真值集合 $\mathbf{T} = \{\mathbf{true}, \mathbf{false}\}$ 上，“或”函数 $\vee: \mathbf{T} \times \mathbf{T} \to \mathbf{T}$ 可以扩充为：

$$\vee_\perp: \mathbf{T}_\perp \times \mathbf{T}_\perp \to \mathbf{T}_\perp$$

其中，

$$x_1 \vee_\perp x_2 =_{def} (let\ t_1 \Leftarrow x_1, t_2 \Leftarrow x_2. \lfloor t_1 \vee t_2 \rfloor)$$

这种扩充是严格的扩充，因为 x_1 和 x_2 之间只要有一个为 $\perp$，则 $x_1 \vee_\perp x_2$ 也为 $\perp$。扩充 $\vee$ 还有其他的计算方法，如 **true** $\vee \perp =$ **true**（见下面的练习）。同样，整数 **N** 上的算术运算也可以严格扩充 $\mathbf{N}_\perp$ 上的运算。例如，

132

$$x_1 +_\perp x_2 =_{def} let\ n_1 \Leftarrow x_1, n_2 \Leftarrow x_2. \lfloor n_1 + n_2 \rfloor$$

练习 8.12 试描述布尔或运算 $\vee$ 所有可能的连续扩充，并给出真值表。 □

8.3.5 和

我们经常要构造完全偏序的不相交的并，如在正常的计算值中加入出错值。完全偏序的和构造就是对集合进行不相交的并运算。设 $D_1, \cdots, D_k$ 是完全偏序，则它们的和 $D_1 + \cdots + D_k$ 具有集合

$$\{in_1(d_1) \mid d_1 \in D_1\} \cup \cdots \cup \{in_k(d_k) \mid d_k \in D_k\}$$

和偏序

$$\begin{aligned} d \sqsubseteq d' \text{ 当且仅当 } & (\exists d_1, d_1' \in D_1.\ d = in_1(d_1)\ \&\ d' = in_1(d_1')\ \&\ d_1 \sqsubseteq d_1') \text{ 或者} \\ & \vdots \\ & (\exists d_k, d_k' \in D_k.\ d = in_k(d_k)\ \&\ d' = in_k(d_k')\ \&\ d_k \sqsubseteq d_k') \end{aligned}$$

其中，我们假设函数 in_i 是一一对应的，使得对所有的 $d \in D_i, d' \in D_j$（其中 $i \neq j$），有

$$in_i(d) \neq in_j(d')$$

容易看出，$D_1 + \cdots + D_k$ 是一个完全偏序，由不相交的完全偏序 $D_1, \cdots, D_k$ 组成；内射函数 $in_i : D_i \to D_1 + \cdots + D_k (i = 1, \cdots, k)$ 是连续的。虽然有不同的方法实现函数 in_i，但它们得到同构的构造。

设 $f_1 : D_1 \to E, \cdots, f_k : D_k \to E$ 都是连续函数，它们可以构成一个连续函数

$$[f_1, \cdots, f_k] : D_1 + \cdots + D_k \to E$$

其定义为：对于 $i = 1, \cdots, k$，有

$$[f_1, \cdots, f_k](in_i(d_i)) = f_i(d_i), \text{对于所有 } d_i \in D_i$$

换言之，对于 $i = 1, \cdots, k$，有

$$[f_1, \cdots, f_k] \circ in_i = f_i$$

函数集合 $D_1 + \cdots + D_k \to E$ 上的这个性质惟一刻画了 $[f_1, \cdots, f_k]$。

练习 8.13 试证明：由 $f_1 \in [D_1 \to E], \cdots, f_k \in [D_k \to E]$ 得到 $[f_1, \cdots, f_k]$ 的运算是连续的。（使用引理 8.10。） □

133

真值集 $\mathbf{T} = \{$**true**, **false**$\}$ 可以看作是两个单元素完全偏序 $\{$**true**$\}$ 和 $\{$**false**$\}$ 的和，其中内射

函数 $in_1 : \{\mathbf{true}\} \to \mathbf{T}$ 使得 $\mathbf{true} \mapsto \mathbf{true}$，同样地，$in_2 : \{\mathbf{false}\} \to \mathbf{T}$ 使得 $\mathbf{false} \mapsto \mathbf{false}$。设

$$\lambda x_1 . e_1 : \{\mathbf{true}\} \to E$$
$$\lambda x_2 . e_2 : \{\mathbf{false}\} \to E$$

为完全偏序 E 上的两个连续函数，于是不难看出

$$cond(t, e_1, e_2) =_{def} [\lambda x_1 . e_1, \lambda x_2 . e_2](t)$$

是一个条件，即

$$cond(t, e_1, e_2) = \begin{cases} e_1 & t = \mathbf{true} \\ e_2 & t = \mathbf{false} \end{cases}$$

其中 $t \in \mathbf{T}, e_1, e_2 \in E$。因为条件中的真值通常是计算的结果，我们要尽可能地利用 $\mathbf{T}_\perp$ 中的测试条件。设完全偏序 E 有底元素 $\perp_E$，条件定义为

$$(b \to e_1 \mid e_2) =_{def} let\ t \Leftarrow b.\ cond(t, e_1, e_2)$$

则

$$(b \to e_1 \mid e_2) = \begin{cases} e_1 & b = \lfloor \mathbf{true} \rfloor \\ e_2 & b = \lfloor \mathbf{false} \rfloor \\ \perp & b = \perp \end{cases}$$

其中 $b \in \mathbf{T}_\perp$ 且 $e_1, e_2 \in E$。我们将在 8.4 节中证明这两个条件都是连续的。

练习 8.14　试验证 *cond* 运算和上面定义的 $(- \to - \mid -)$ 确为所宣称的条件。 □

和构造及其相关函数能使我们定义一般的 *case* 构造，根据元素属于和的不同组成部分而输出不同的结果。假设 E 是一个完全偏序，$(D_1 + \cdots + D_k)$ 是含元素 d 的完全偏序的和，并设

$$\lambda x_i . e_i : D_i \to E$$

是连续函数 $(1 \leq i \leq k)$。*case* 构造

$$\begin{array}{l} case\ d\ of\ in_1(x_1).\ e_1 \mid \\ \quad \vdots \\ \quad in_k(x_k).\ e_k \end{array}$$

134

在对某个 $d_i \in D_i$ 有 $d = in_i(d_i)$ 的情况下，试图产生输出 e_i。为了做到这一点，*case* 构造定义为

$$[\lambda x_1 . e_1, \cdots, \lambda x_k . e_k](d)$$

练习 8.15　请读者思考为什么上面的定义实现了和构造的目的。 □

最后，我们要指出空完全偏序 $\emptyset$ 是有限和的退化情况，它没有成分。

8.4　元语言

在定义程序设计语言的语义时，我们希望函数是连续的，以求得它们的最小不动点。这就

提出了一个问题,因为我们不希望每一次都要解释定义以检查表达式是否是良定义的,函数是否确实是连续的。如果一旦有一个非形式化的语法,使得符合该语法的数学表达式所表示的函数总是连续的,这样很多繁琐的工作就省掉了。我们把这个语法的所有表达式构成的语言称为元语言,这样程序设计语言的指称语义就可以用元语言来描述。

我们已经见过几个 λ 记号的例子,在域论中也经常要用到 λ 记号。当 x 是完全偏序 D 的元素时,我们设表达式 e 表示完全偏序 E 的一个元素。例如,e 可能是条件"$cond(x,0,1)$",其中 D 为真值集 $\mathbf{T}$,E 为自然数集 ω。我们记

$$\lambda x \in D.\, e$$

表示函数 $h: D \to E$,使得对所有的 $d \in D$,$h(d) = e[d/x]$。如果 x 显然属于集合 D,也可以把它简记为 $\lambda x.\, e$。设 e 是涉及元素 $x \in D_1$ 和 $y \in D_2$ 的表达式,可以记为

$$\lambda(x,y) \in D_1 \times D_2.\, e$$

以代替

$$\lambda z \in D_1 \times D_2.\, e[\pi_1(z)/x, \pi_2(z)/y]$$

更一般地,该函数记为

$$\lambda x \in D_1, y \in D_2.\, e$$

或者仅记为

135

$$\lambda x, y.\, e$$

我们希望尽可能自由地使用 λ 记号,同时确保定义的函数是连续的。一个典型的情况是,表达式 e 表示完全偏序 E 的一个元素,同时它依赖于完全偏序 D 中的变量 x。称这样的表达式 e 对变量 $x \in D$ 连续,当且仅当函数 $\lambda x \in D.\, e: D \to E$ 是连续的。称 e 对它的变量连续,当且仅当 e 对所有的变量都连续。当然,表达式 e 依赖于一些变量而不依赖于其他的变量;如果变量 $x \in D$ 不在 e 中出现,则函数 $\lambda x \in D.\, e$ 是一个常函数,因此它一定是连续的。

我们按照以下方法构造完全偏序上的表达式,其中的运算都是已知的。根据本章的结果,这些表达式对它们的变量都是连续的。

变量 x 是完全偏序 E 的一个元素,只有单个变量 x 组成的表达式对它的变量是连续的,因为对于 $y \in D$,抽象 $\lambda y.\, x$ 要么是恒等函数 $\lambda x.\, x$(此时 y 为变量 x),要么是常函数。

常量 我们已经遇到过完全偏序中许多特殊的元素,比如,完全偏序的底元素 $\perp_D \in D$,真值 **true**,**false** $\in \mathbf{T}$,积的投影函数 $\pi_1 \in [D_1 \times D_2 \to D_1]$,函数空间的施用函数 $apply \in [[D \to E] \times D \to E]$,提升的 $(\text{-})^*$ 函数,和的内射函数及 $[\,,\cdots,]$ 运算,函数 $fix \in [[D \to D] \to D]$($fix$ 是一个连续函数,因而也是完全偏序的元素,它的讨论放在本节的最后)等等。这些常量表达式给出了完全偏序的固定的元素,因而对它们的变量都是连续的。

元组 给定完全偏序 $E_1, \cdots, E_k$ 的表达式 $e_1 \in E_1, \cdots, e_k \in E_k$,我们可以得到完全偏序的积 $E_1 \times \cdots \times E_k$ 中的元组 $(e_1, \cdots, e_k)$。这样的元组对变量 $x \in D$ 是连续的当且仅当

$$\lambda x.\, (e_1, \cdots, e_k) \text{ 是连续的}$$

$$\Longleftrightarrow \pi_i \circ (\lambda x.(e_1,\cdots,e_k)) \text{ 是连续的}(1 \leqslant i \leqslant k) \quad (\text{由引理 8.8})$$
$$\Longleftrightarrow \lambda x.\, e_i \text{ 是连续的}(1 \leqslant i \leqslant k)$$
$$\Longleftrightarrow e_i \text{ 对 } x \text{ 是连续的}(1 \leqslant i \leqslant k)$$

所以,如果元组的每个分量对变量是连续的,则这个元组对该变量也是连续的。

应用　给定 K 是上面"常量"中讨论的常量连续函数,将它应用于合适的参数表达式 e。结果 $K(e)$ 对 x 是连续的当且仅当 136

$$\begin{aligned} & \lambda x.\, K(e) \text{ 是连续的} \\ \Longleftrightarrow\ & K \circ (\lambda x.\, e) \text{ 是连续的} \\ \Longleftarrow\ & \lambda x.\, e \text{ 是连续的} \qquad (\text{由命题 8.1}) \\ \Longleftrightarrow\ & e \text{ 对 } x \text{ 是连续的} \end{aligned}$$

所以,如果应用的变元对变量是连续的,则这个应用对该变量也是连续的。特别地,设表达式 e_1, e_2 对变量是连续的,则形如 $e_1(e_2)$ 的应用对该变量是连续的,这是因为 $e_1(e_2) = \mathbf{apply}(e_1, e_2)$,即将常量 *apply* 应用到元组 (e_1, e_2) 的结果。

λ 抽象　设 $e \in E$ 对它的变量是连续的,选择完全偏序 D 的一个特定变量 y,并构造一个连续函数 $\lambda\, y.\, e: D \to E$。我们希望这个抽象本身对它的变量 x 是连续的。显然,如果 x 正好就是变量 y,结果就为常量函数 $\lambda\, y.\, e$。否则,$\lambda\, y.\, e$ 对 x 是连续的当且仅当

$$\begin{aligned} & \lambda x.\, \lambda y.\, e \text{ 是连续的} \\ \Longleftrightarrow\ & curry(\lambda x, y.\, e) \text{ 是连续的} \\ \Longleftarrow\ & \lambda x, y.\, e \text{ 是连续的} \qquad (\text{因为 } curry \text{ 能保持连续性}) \\ \Longleftrightarrow\ & e \text{ 对于 } x \text{ 和 } y \text{ 是连续的} \end{aligned}$$

所以,如果抽象的体对变量是连续的,则该抽象对其变量是连续的[⊖]。特别地,因为

$$e_1 \circ e_2 = \lambda x.\, e_1(e_2(x))$$

所以,复合函数能保持对变量的连续的性质。要注意的是更一般的抽象如 $\lambda\, x, y \in D_1 \times D_2.\, e$ 也是允许的,因为它等价于 $\lambda\, z \in D_1 \times D_2.\, e[\pi_1(z)/x, \pi_2(z)/y]$。

对固定的连续函数和特殊元素,由上述方法构成的表达式对其变量都是连续的。下面我们再介绍保持这种性质的其他几种重要的构造。

***let* 构造**　设 D 是完全偏序,E 是含底元素的完全偏序。如果 $e_1 \in D_\perp, e_2 \in E$ 对它们的变量是连续的,则表达式

$$let\ x \Leftarrow e_1.\, e_2$$ 137

对它的变量也是连续的。这是因为

$$(let\ x \Leftarrow e_1.\, e_2) = (\lambda x.\, e_2)^*(e_1)$$

⊖ 该条件也是必需的,因为推导中的蕴涵"⇐"可以用等价"⟺"来替代,尽管此处没有证明。本节最后的练习 8.16 说明了这点。

而右边的表达式可以用上面的方法来从 e_1 和 e_2 构造出来。

***case* 构造** 设 E 是一个完全偏序，令 $(D_1+\cdots+D_k)$ 是含有元素 e 的完全偏序的和，假设表达式 e 对它的变量是连续的。设表达式 $e_i \in E(1 \leq i \leq k)$ 对它的变量都是连续的，则 *case* 构造

$$\begin{array}{c} case\ e\ of\ in_1(x_1).\ e_1 \mid \\ \vdots \\ in_k(x_k).\ e_k \end{array}$$

对它的变量也是连续的，这是因为它的定义是

$$[\lambda x_1.\ e_1, \cdots, \lambda x_k.\ e_k](e)$$

它可以用上面的方法构造出来——我们回忆和的运算 $[-,\cdots,-]$ 是连续的，它允许作为我们的常量之一。特别地，8.3.5 节介绍的条件表达式 $cond(t,e_1,e_2)$（其中 t 为真值，e_1,e_2 属于同一个完全偏序）对它的变量是连续的，因为它等价于 $[\lambda\ x_1.\ e_1, \lambda\ x_2.\ e_2](t)$。8.3.5 节介绍的定义在含底元素的完全偏序上的它的变形 $b \rightarrow e_1 \mid e_2$ 对它的变量是连续的，因为它可以定义为 $let\ t \Leftarrow b.\ cond(t,e_1,e_2)$。

不动点算子 每个含底元素的完全偏序都有一个不动点算子 $fix: [D \rightarrow D] \rightarrow D$，事实上函数 fix 本身就是连续的。因为

$$fix = \bigsqcup_{n \in \omega} (\lambda f.\ f^n(\perp))$$

也就是说，fix 是函数

$$\lambda f.\ \perp \sqsubseteq \lambda f.\ f(\perp) \sqsubseteq \lambda f.\ f(f(\perp)) \sqsubseteq \cdots$$

的 ω 链的最小上界，其中链的每一个元素都是连续的，所以按上面的方法构造的完全偏序 $[[D \rightarrow D] \rightarrow D]$ 也是连续的。还可以知道，它们的最小上界 fix 也在 $[[D \rightarrow D] \rightarrow D]$ 中。

138 **记号** 以后我们用记号 $\mu\ x.\ e$ 来简化 $fix(\lambda\ x.\ e)$。

我们用上面的结果来说明表达式是良定义的。虽然我们的讨论是非形式化的，但是完全可以对上述元语言给出形式化定义。形式化描述会精确说明有哪些类型，哪些常量操作构成特定类型的 λ 演算，根据 λ 演算的标准解释，项指称完全偏序的元素，语言的构造规则确保不会出现不连续的函数。这方面的工作有斯科特的可计算函数逻辑（LCF），它包括有谓词以及论证最小不动点的证明规则（见第 10 章的不动点归纳法）和带类型的 λ 演算。

练习 8.16 请读者回忆 8.3.3 节，从 $A=[F \times D \rightarrow E]$ 到 $B=[F \rightarrow [D \rightarrow E]]$ 的函数 $curry = \lambda\ g \lambda\ v \lambda\ d.\ g(v,d)$。本练习表明 $curry$ 是从 A 到 B 的同构。请读者思考为什么 $curry$ 是 $A \rightarrow B$ 的连续函数？试定义 $curry$ 的逆函数 $uncurry: B \rightarrow A$，即 $curry \circ uncurry = Id_B$ 且 $uncurry \circ curry = Id_A$。试证明 $uncurry$ 是 $curry$ 的逆函数，并且是连续的。 □

8.5 进一步阅读资料

本章的内容主要基于 Gordon Plotkin 的讲义（“Pisa notes”[80]和他后来的工作成果

[83])，而很多基础的内容都受 Eugenio Moggi 的书[67]和 Andrew Pitts 的讲义[75]的影响。这些成果都是建立在斯科特在20世纪60年代末的工作基础上的。还要感谢 Christopher Wadworth 在爱丁堡的精彩演讲，从中我学到了很多，遗憾的是当时讲稿没有印刷发行。关于 LCF 逻辑和 ML 语言实现的辅助证明工具请参考 Larry Paulson 的著作[74]。指称语义的介绍还可以参考[88]、[95]和[91]。本章实际上介绍了完全偏序和连续函数的范畴，并说明了由于范畴有积和函数空间，因而它是笛卡儿封闭的，同时根据和构造方法还可以构造出它的对偶积(coproduct)。关于范畴论的基本知识请参考[10]和[15]。 139

第9章 递归方程

本章介绍一种简单的语言 **REC**,它支持整数上函数的递归定义。**IMP** 是命令式语言,而 **REC** 是应用式语言。它可以按照传值或者传名的方式来求值。本章将讨论这两种求值方式的操作语义和指称语义,并证明这两种语义是等价的。

9.1 REC 语言

REC 语言是为了支持函数的递归定义而设计的一种简单的程序设计语言。它包括以下语法集合:

- 整数 $n \in \mathbf{N}$。
- 整数变量 $x \in \mathbf{Var}$。
- 函数变量 $f_1, \cdots, f_k \in \mathbf{Fvar}$。

假设每一个函数变量 $f_i \in \mathbf{Fvar}$ 都拥有元数 $a_i \in \omega$,它是函数所取变量的个数。变量个数可以为0,此时 $f_i()$ 表示作用于空元组的0元函数 f_i,$f_i()$ 通常简记为 f_i。**REC** 语言中的项 $t, t_0, t_1, \cdots$ 由以下语法组成:

$$t ::= n \mid x \mid t_1 + t_2 \mid t_1 - t_2 \mid t_1 \times t_2 \mid \textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2 \mid f_i(t_1, \cdots, t_{a_i})$$

为了简便,我们把布尔表达式也定义为项,并以0表示真,以所有的非零整数表示假。(因此,可以把"析取"记为 $\times$,把"$\neg b$"作为条件表达式 **if** b **then** 1 **else** 0,把两个项 t_0 和 t_1 之间的相等测试($t_0 = t_1$)这样的基本布尔表达式定义为($t_0 - t_1$)——见下面的练习9.1。)如果一个项中不包含 **Var** 中的变量,我们就称该项是封闭的。

函数变量 f 通过声明而获得含义,声明通常由如下形式的方程组成:

$$\begin{aligned} f_1(x_1, \cdots, x_{a_1}) &= t_1 \\ &\vdots \\ f_k(x_1, \cdots, x_{a_k}) &= t_k \end{aligned}$$

其中,$t_i (1 \leq i \leq k)$ 的变量取自 $x_1, \cdots, x_{a_i}$。这些方程可以是递归的,因为 t_i 中可能包含了函数变量 f_i 以及其他的函数变量 $f_1, \cdots, f_k$。我们可以合理地做出下列假设:对同一个函数变量不允许进行两次定义。 141

在定义方程

$$f_i(x_1, \cdots, x_{a_i}) = t_i$$

中,我们称项 t_i 为 f_i 的定义。

REC 语言的操作语义不是那么简单。考察定义方程

$$f_1(x) = f_1(x) + 1$$

由直观计算可知，$f_1(3)$和$f_1(3)+1$ 的求值结果应该相同，同样，$f_1(3)+1$ 应该等于$(f_1(3)+1)+1$，以此类推，$f_1(3)$的求值永远不会终止。实际上，如果$f_1(3)$求值最终得到一个整数 n，则上式就会得到矛盾方程 $n=n+1$。另外，假设我们有一个定义方程

$$f_2(x) = 1$$

在求$f_2(t)$时，对于项 t，我们有两种选择：一种是先求出变量 t 的值，一旦得到其求值结果 n 后，再计算$f_2(n)$。另一种是直接传递给f_2 的定义，用变量 t 去替换所有出现的变量 x。但是当项 t 为$f_1(3)$时，这两种求值计算方式得到的结果是截然不同的：前者发散而后者却终止于 1。在第一种求值计算方式中，我们需要先求出变量的值，然后将它传递给定义方程，这种方式称为*传值调用*；而在第二种求值计算方式中，我们把未求值的项直接传递给定义方程，这种方式称为*传名调用*。显然，如果计算中确实需要变量，则一次性计算该变量的效率比较高；否则相同的项可能需要在这个定义中被多次求值。另一方面，正如我们在求$f_2(f_1(3))$时所看到的那样，如果变量永远不被用到，那么变量的发散不会引起包含它的项的发散。

练习 9.1 根据你对 **REC** 语言中项求值的非形式化的理解，下述声明中定义的函数 s 应如何计算？

$$s(x) = \textbf{if } x \textbf{ then } 0 \textbf{ else } f(x, 0-x)$$
$$f(x,y) = \textbf{if } x \textbf{ then } 1 \textbf{ else } (\textbf{if } y \textbf{ then } -1 \textbf{ else } f(x-1, y-1))$$

试用 **REC** 语言定义一个函数 $lt(x,y)$，使得当 $x<y$ 时该函数返回 0，否则返回一个非零整数。 □

142

9.2 传值调用的操作语义

设声明 d 由下式确定

$$f_1(x_1,\cdots,x_{a_1}) = d_1$$
$$\vdots$$
$$f_k(x_1,\cdots,x_{a_k}) = d_k$$

其中项 d_i 为f_i 的定义$(i=1,\cdots,k)$。对应于这些定义，我们给出 **REC** 中封闭项的求值规则。

我们约定 $t \rightarrow^d_{va} n$ 表示在声明 d 中按照传值调用的方式，对封闭项 t 求值得到整数 n，这种求值关系的规则为：

$$(num) \quad n \rightarrow^d_{va} n$$

$$(op) \quad \frac{t_1 \rightarrow^d_{va} n_1 \quad t_2 \rightarrow^d_{va} n_2}{t_1 \textbf{ op } t_1 \rightarrow^d_{va} n_1 \ op \ n_2}$$

$$(condt) \quad \frac{t_0 \rightarrow^d_{va} 0 \quad t_1 \rightarrow^d_{va} n_1}{\textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2 \rightarrow^d_{va} n_1}$$

$$(condf)\quad \frac{t_0 \rightarrow^d_{va} n_0 \quad t_2 \rightarrow^d_{va} n_2 \quad n_0 \neq 0}{\textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2 \rightarrow^d_{va} n_2}$$

$$(fn)\quad \frac{t_1 \rightarrow^d_{va} n_1 \cdots t_{a_i} \rightarrow^d_{va} n_{a_i} \quad d_i[n_1/x_1, \cdots, n_{a_i}/x_{a_i}] \rightarrow^d_{va} n}{f_i(t_1, \cdots, t_{a_i}) \rightarrow^d_{va} n}$$

这些规则都很容易理解，不过要注意的是这里语法运算符 **op** 和整数上的运算符 *op* 是有区别的；下式为规则(*op*)在加法规则中的一个实例：

$$\frac{3 \rightarrow^d_{va} 3 \quad 4 \rightarrow^d_{va} 4}{3 + 4 \rightarrow^d_{va} 7}$$

在这些规则中，条件规则有些特殊，这是因为我们把 0 定义为真，而把所有非零整数都定义为假的缘故。读者请注意函数的求值规则中，使用函数定义前是怎样先对变量进行求值计算的。 143

传值调用方式下的求值关系是确定的。

命题 9.2 若 $t \rightarrow^d_{va} n_1$ 且 $t \rightarrow^d_{va} n_2$，则 $n_1 \equiv n_2$。

证明 应用规则归纳法进行证明。 □

9.3 传值调用的指称语义

对于项来说，只有当它的变量和函数变量处在一定的环境中，它才会被赋予一定的含义。变量的环境是一个函数

$$\rho : \mathbf{Var} \rightarrow \mathbf{N}$$

对于所有此类环境的完全偏序，我们把它记为 $\mathbf{Env}_{va} = [\mathbf{Var} \rightarrow \mathbf{N}]$。

函数变量 $f_1, \cdots, f_k$ 的环境是一个元组 $\varphi = (\varphi_1, \cdots, \varphi_k)$，其中

$$\varphi_i : \mathbf{N}^{a_i} \rightarrow \mathbf{N}_\perp$$

对于函数变量环境的完全偏序，我们用 $\mathbf{Fenv}_{va}$ 表示 $[\mathbf{N}^{a_1} \rightarrow \mathbf{N}_\perp] \times \cdots \times [\mathbf{N}^{a_k} \rightarrow \mathbf{N}_\perp]$。正如我们所期望的那样，一个声明确定了一个特定的函数环境。

给定函数变量和变量的环境 ρ 和 φ，项 t 指称 $\mathbf{N}_\perp$ 的一个元素。更精确地说，项 t 指称一个函数

$$[\![t]\!]_{va} \in [\mathbf{Fenv}_{va} \rightarrow [\mathbf{Env}_{va} \rightarrow \mathbf{N}_\perp]]$$

它由下述结构归纳法给出：

$$\begin{aligned}
[\![n]\!]_{va} &= \lambda\varphi\lambda\rho.\lfloor n \rfloor \\
[\![x]\!]_{va} &= \lambda\varphi\lambda\rho.\lfloor \rho(x) \rfloor \\
[\![t_1 \textbf{ op } t_2]\!]_{va} &= \lambda\varphi\lambda\rho.[\![t_1]\!]_{va}\varphi\rho \; op_\perp \; [\![t_2]\!]_{va}\varphi\rho \\
&\quad (\textbf{op} \text{ 取 } +, -, \times) \\
[\![\textbf{if } t_0 \textbf{ then } t_1 \textbf{ else } t_2]\!]_{va} &= \lambda\varphi\lambda\rho.Cond([\![t_0]\!]_{va}\varphi\rho, [\![t_1]\!]_{va}\varphi\rho, [\![t_2]\!]_{va}\varphi\rho) \\
[\![f_i(t_1, \cdots, t_{a_i})]\!]_{va} &= \lambda\varphi\lambda\rho.
\end{aligned}$$

$$(let\ v_1 \Leftarrow [\![t_1]\!]_{va}\varphi\rho, \cdots, v_{a_i} \Leftarrow [\![t_{a_i}]\!]_{va}\varphi\rho.\ \varphi_i(v_1, \cdots, v_{a_i}))$$

这个定义中的运算符 $+_\perp$，$-_\perp$，$\times_\perp$ 是整数 **N** 上通常的算术运算的严格扩充；例如，在 8.3.4 节中，有

144

$$z_1 +_\perp z_2 = \begin{cases} \lfloor n_1 + n_2 \rfloor & \text{对某些 } n_1, n_2 \in \mathbf{N}, z_1 = \lfloor n_1 \rfloor \text{ 且 } z_2 = \lfloor n_2 \rfloor \\ \perp & \text{其他} \end{cases}$$

其中 $z_1, z_2 \in \mathbf{N}$。用函数

$$Cond: \mathbf{N}_\perp \times \mathbf{N}_\perp \times \mathbf{N}_\perp \to \mathbf{N}_\perp$$

来定义条件的含义，它满足

$$Cond(z_0, z_1, z_2) = \begin{cases} z_1 & z_0 = \lfloor 0 \rfloor \\ z_2 & z_0 = \lfloor n \rfloor (\text{对于某个 } n \in \mathbf{N}, n \neq 0) \\ \perp & \text{其他} \end{cases}$$

其中 $z_0, z_1, z_2 \in \mathbf{N}_\perp$。也可以从前面 8.3.5 节中介绍的条件中得到上式。当变量取 0 时，我们设函数 *iszero*：$\mathbf{N} \to \mathbf{T}$ 的值为 **true**，否则为 **false**。函数 *iszero* 是定义在离散完全偏序之间的连续函数，因此，它的严格扩充函数

$$iszero_\perp = \lambda z \in \mathbf{N}_\perp.\ let\ n \Leftarrow z.\ \lfloor iszero(n) \rfloor$$

也是连续的，并且

$$iszero_\perp(z) = \begin{cases} \lfloor \mathbf{true} \rfloor & z = \lfloor 0 \rfloor \\ \lfloor \mathbf{false} \rfloor & z = \lfloor n \rfloor \text{ 且 } n \neq 0 \\ \perp & \text{其他} \end{cases}$$

于是

$$Cond(z_0, z_1, z_2) = (iszero_\perp(z_0) \to z_1 \mid z_2)$$

其中 $z_0, z_1, z_2 \in \mathbf{N}_\perp$。根据 8.4 节中的结论，它显然是一个连续函数。事实上，对于 **REC** 语言中的任何一个项 t，语义函数 $[\![t]\!]_{va}$ 都是连续函数。这一点可由下面的引理直接得出。

引理 9.3 对于 **REC** 语言中所有的项 t，指称 $[\![t]\!]_{va}$ 是 $[\mathbf{Fenv}_{va} \to [\mathbf{Env}_{va} \to \mathbf{N}_\perp]]$ 中的一个连续函数。

证明 利用 8.4 节的结论，直接对项 t 进行结构归纳即可证明。 □

请注意一个很直观的事实，一个项在某一环境中的指称的结果独立于项外部的变量所赋的值。

引理 9.4 对于 **REC** 语言中所有的项 t，如果对出现于 t 中所有的变量，环境 $\rho, \rho' \in \mathbf{Env}_{va}$ 产生相同的结果，则对于任何 $\varphi \in \mathbf{Fenv}_{va}$，有

$$[\![t]\!]_{va}\varphi\rho = [\![t]\!]_{va}\varphi\rho'$$

145 特别地，封闭项 t 的指称 $[\![t]\!]_{va}\varphi\rho$ 独立于环境 ρ。

证明 直接对项 t 进行结构归纳。 □

上面的语义表示了对应于函数环境 $\varphi=(\varphi_1,\cdots,\varphi_k)$ 的项的意义。确切的函数环境由下列定义方程组成的声明确定：

$$\begin{aligned} f_1(x_1,\cdots,x_{a_1}) &= d_1 \\ &\vdots \\ f_k(x_1,\cdots,x_{a_k}) &= d_k \end{aligned}$$

上式可理解为 $f_1,\cdots,f_k$ 的递归方程，其中 $f_1,\cdots f_k$ 必须满足函数环境 $\delta=(\delta_1,\cdots,\delta_k)$：

$$\begin{aligned} \delta_1(n_1,\cdots,n_{a_1}) &= [\![d_1]\!]_{va}\delta\rho[n_1/x_1,\cdots,n_{a_1}/x_{a_1}],\ \text{对于所有的}\ n_1,\cdots,n_{a_1}\in\mathbf{N} \\ &\vdots \\ \delta_k(n_1,\cdots,n_{a_k}) &= [\![d_k]\!]_{va}\delta\rho[n_1/x_1,\cdots,n_{a_k}/x_{a_k}],\ \text{对于所有的}\ n_1,\cdots,n_{a_k}\in\mathbf{N} \end{aligned}$$

我们采用新的记号表示环境 ρ 的更新。定义 $\rho[n/x]$ 为环境

$$(\rho[n/x])(y) = \begin{cases} \rho(y) & y \not\equiv x \\ n & y \equiv x \end{cases}$$

其中 $x\in\mathbf{Var}, n\in\mathbf{N}$。我们也可以用 8.4 节介绍的元语言定义环境的更新。注意到离散的完全偏序 **Var** 可以看作是单元素集合 $\{x\}$ 与 $\mathbf{Var}\setminus\{x\}$ 之和，并且内射函数 $in_1:\{x\}\to\mathbf{Var}$ 和 $in_2:(\mathbf{Var}\setminus\{x\})\to\mathbf{Var}$ 都是包含函数。现在我们看到 $\rho[n/x]$ 等于

$$\begin{aligned} \lambda y\in\mathbf{Var}.\ case\ y\ of\ in_1(x).n\ \mid \\ in_2(w).\rho(w) \end{aligned}$$

我们把 $(\rho[n_0/x_0])[n_1/x_1]$ 简记为 $\rho[n_0/x_0,n_1/x_1]$。

（注意，这里我们忽略了整数的完全偏序的特殊性。当变量被限制到其他更复杂的完全偏序的元素时，可以用元语言来定义类似的更新操作。）

这些方程一般不能确定惟一的解，然而存在一个最小解，即连续函数

$$F:\ \mathbf{Fenv}_{va}\to\mathbf{Fenv}_{va}$$ 146

的最小不动点，由下式给出：

$$\begin{aligned} F(\varphi) = (&\lambda n_1,\cdots,n_{a_1}\in\mathbf{N}.\ [\![d_1]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_{a_1}/x_{a_1}],\cdots, \\ &\lambda n_1,\cdots,n_{a_k}\in\mathbf{N}.\ [\![d_k]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_{a_k}/x_{a_k}]) \end{aligned}$$

函数 F 是连续的，因为它是从引理 9.3 中由已知的连续函数 $[\![d_1]\!]_{va},\cdots,[\![d_k]\!]_{va}$ 使用 8.4 节介绍的方法构造出来的。

我们可以把声明 d 确定的函数环境定义为最小不动点

$$\delta = fix(F)$$

对应于这个函数环境，封闭项 t 指称 $\mathbf{N}_\perp$ 中的一个结果 $[\![t]\!]_{va}\delta\rho$，同时又独立于所使用的环境 ρ。当然，我们最好还是检查一下，项指称的结果是否与操作语义给出的值一致。我们将在下一节中进一步讨论。

下面,我们通过几个例子来说明指称语义是如何求值计算的,并结束传值调用方式下 **REC** 语言的指称语义的讨论。

例 考察传值调用方式下指称语义的求值计算,考察声明

$$f_1 = f_1 + 1$$
$$f_2(x) = 1$$

(这里,f_1 是一个递归定义的不带参数的函数,即常函数。)根据指称语义,该声明的作用是用 $\delta = (\delta_1, \delta_2) \in \mathbf{N}_\perp \times [\mathbf{N} \to \mathbf{N}_\perp]$ 指称 f_1 和 f_2,其中

$$\begin{aligned}(\delta_1, \delta_2) &= \mu\varphi.(\llbracket f_1 + 1 \rrbracket_{va}\varphi\rho, \lambda m \in \mathbf{N}.\llbracket 1 \rrbracket_{va}\varphi\rho[m/x]) \\ &= \mu\varphi.(\varphi_1 +_\perp \lfloor 1 \rfloor, \lambda m \in \mathbf{N}.\lfloor 1 \rfloor)\end{aligned}$$

在这种情况下,容易看出

$$(\perp, \lambda m \in \mathbf{N}.\lfloor 1 \rfloor)$$

为所需的最小不动点(可以简单证明这个序偶是 $\lambda\varphi.(\varphi_1 +_\perp \lfloor 1 \rfloor, \lambda m \in \mathbf{N}.\lfloor 1 \rfloor)$ 的一个不动点,而且还是最小不动点)。于是

$$\delta_1 = \perp$$

147

$$\delta_2 = \lambda m \in \mathbf{N}.\lfloor 1 \rfloor$$

从而得到

$$\begin{aligned}\llbracket f_2(f_1) \rrbracket_{va}\delta\rho &= let\ n_1 \Leftarrow \delta_1.\delta_2(n_1) \\ &= \perp\end{aligned}$$

□

例 本例对最小不动点进行了更为详细的分析。考察声明

$$f(x) = \textbf{if } x \textbf{ then } 1 \textbf{ else } x \times f(x-1)$$

这里我们只关心 f,为了简单起见,我们用 $[\mathbf{N} \to \mathbf{N}_\perp]$ 来表示函数环境 $\mathbf{Fenv}_{va}$。根据指称语义,f 声明为下述函数 δ,其中 t 为定义,ρ 为任意的变量环境:

$$\begin{aligned}\delta &= \mu\varphi.(\lambda m.\llbracket t \rrbracket_{va}\varphi\rho[m/x]) \\ &= fix(\lambda\varphi.(\lambda m.\llbracket t \rrbracket_{va}\varphi\rho[m/x])) \\ &= \bigsqcup_{r \in \omega} \delta^{(r)}\end{aligned}$$

上式中我们取

$$F(\varphi) = (\lambda m.\llbracket t \rrbracket_{va}\varphi\rho[m/x])$$

并定义

$$\delta^{(r)} = F^r(\perp)$$

根据指称语义,以及 *Cond* 的定义,我们可以得到

$$F(\varphi)(m) = Cond(\lfloor m \rfloor, \lfloor 1 \rfloor, \lfloor m \rfloor \times_\perp \varphi(m-1))$$

$$= iszero_{\perp}(\lfloor m \rfloor) \rightarrow \lfloor 1 \rfloor \mid \lfloor m \rfloor \times_{\perp} \varphi(m-1)$$

其中 $\varphi \in [\mathbf{N} \rightarrow \mathbf{N}_{\perp}]$ 且 $m \in \mathbf{N}$。注意到

$$F(\varphi)(m) = cond(iszero(m), \lfloor 1 \rfloor, \lfloor m \rfloor \times_{\perp} \varphi(m-1))$$

其中我们使用了 8.3.5 节完全偏序的和上的函数 $cond: \mathbf{T} \times \mathbf{N}_{\perp} \times \mathbf{N}_{\perp} \rightarrow \mathbf{N}_{\perp}$。对任意的 $m \in \mathbf{N}$,我们计算:

$$\delta^{(0)}(m) = \perp$$

$$\delta^{(1)}(m) = F(\delta^{(0)})(m) = cond(iszero(m), \lfloor 1 \rfloor, \lfloor m \rfloor \times_{\perp} \delta^{(0)}(m-1))$$ 148

$$= \begin{cases} \lfloor 1 \rfloor & m = 0 \\ \perp & \text{其他} \end{cases}$$

$$\delta^{(2)}(m) = F(\delta^{(1)})(m) = cond(iszero(m), \lfloor 1 \rfloor, \lfloor m \rfloor \times_{\perp} \delta^{(1)}(m-1))$$

$$= \begin{cases} \lfloor 1 \rfloor & m = 0 \text{ 或 } m = 1 \\ \perp & \text{其他} \end{cases}$$

一般地,我们有

$$\delta^{(r)}(m) = F(\delta^{(r-1)})(m) = cond(iszero(m), \lfloor 1 \rfloor, \lfloor m \rfloor \times_{\perp} \delta^{(r-1)}(m-1))$$

由数学归纳法,我们得到

$$\delta^{(r)}(m) = \begin{cases} \lfloor m! \rfloor & 0 \leqslant m < r \\ \perp & \text{其他} \end{cases}$$

正如我们期望的,当 m 取非负整数时,最小上界 δ 是一个阶乘函数,在其他情况下 δ 是 $\perp$:

$$\delta(m) = \begin{cases} \lfloor m! \rfloor & 0 \leqslant m \\ \perp & \text{其他} \end{cases}$$

(在传名调用方式下,这个例子实质上没有改变。) □

9.4 传值调用的语义等价

操作语义和指称语义这两种语义是一致的。设 δ 为函数环境,它是声明 d 中 F 的最小不动点。本节的讨论结果将表明,对于封闭项 t 和整数 n,有

$$[\![t]\!]_{va}\delta\rho = \lfloor n \rfloor \text{ 当且仅当 } t \rightarrow^{d}_{va} n$$

因为 t 是封闭的,所以环境 ρ 可以是任意的——它不会影响到指称。

语义等价的证明可以归结为两个主要引理,它们分别表示等价的两个方向。下面第一个引理是辅助的代入引理。

引理 9.5(代入引理) 设 t 为项且 n 为整数,$\varphi \in \mathbf{Fenv}_{va}$,$\rho \in \mathbf{Env}_{va}$,则

$$[\![t]\!]_{va}\varphi\rho[n/x] = [\![t[n/x]]\!]_{va}\varphi\rho$$

证明 简单地施结构归纳于 t 进行证明。 □ 149

引理 9.6 设 t 为封闭项，n 为整数，$\rho \in \mathbf{Env}_{va}$，则

$$t \rightarrow_{va}^{d} n \Rightarrow [\![t]\!]_{va}\delta\rho = \lfloor n \rfloor$$

证明 使用规则归纳法来证明下面的性质：

$$P(t,n) \text{ 当且仅当 } [\![t]\!]_{va}\delta\rho = \lfloor n \rfloor$$

其中 t 为项，n 为整数。（由于 t 是封闭的，所以 ρ 为任意环境。）

对于整数 n，考察规则实例 $n \rightarrow_{va}^{d} n$，显然有 $[\![n]\!]_{va}\delta\rho = \lfloor n \rfloor$，所以 $P(n,n)$ 成立。

假设性质 P 对规则 (op) 的前提成立。假设

$$t_1 \rightarrow_{va}^{d} n_1 \text{ 且 } [\![t_1]\!]_{va}\delta\rho = \lfloor n_1 \rfloor, \text{同时}$$
$$t_2 \rightarrow_{va}^{d} n_2 \text{ 且 } [\![t_2]\!]_{va}\delta\rho = \lfloor n_2 \rfloor$$

于是，我们有

$$\begin{aligned}[\![t_1 \ \mathbf{op}\ t_2]\!]_{va}\delta\rho &= [\![t_1]\!]_{va}\delta\rho \ op_{\perp} \ [\![t_2]\!]_{va}\delta\rho \text{（由定义）} \\ &= \lfloor n_1 \rfloor op_{\perp} \lfloor n_2 \rfloor \\ &= \lfloor n_1 \ op \ n_2 \rfloor\end{aligned}$$

因此，$P(t_1 \ \mathbf{op}\ t_2, n_1 \ op \ n_2)$，即性质对规则 (op) 的结论成立。

条件规则的两种情况 $(condt)$、$(condf)$ 的证明方法与此类似，此处省略。

最后，我们考察 (fn) 的规则实例。假设

$$\begin{gathered} t_1 \rightarrow_{va}^{d} n_1 \text{ 且 } [\![t_1]\!]_{va}\delta\rho = \lfloor n_1 \rfloor, \\ \vdots \\ t_{a_i} \rightarrow_{va}^{d} n_{a_i} \text{ 且 } [\![t_{a_i}]\!]_{va}\delta\rho = \lfloor n_{a_i} \rfloor, \text{以及} \\ d_i[n_1/x_1, \cdots, n_{a_i}/x_{a_i}] \rightarrow_{va}^{d} n \text{ 且 } [\![d_i[n_1/x_1, \cdots, n_{a_i}/x_{a_i}]]\!]_{va}\delta\rho = \lfloor n \rfloor \end{gathered}$$

我们知道

$$\begin{aligned}[\![f_i(t_1, \cdots, t_{a_i})]\!]_{va}\delta\rho &= let\ v_1 \Leftarrow [\![t_1]\!]_{va}\delta\rho, \cdots, v_{a_i} \Leftarrow [\![t_{a_i}]\!]_{va}\delta\rho.\ \delta_i(v_1, \cdots, v_{a_i}) \\ &= \delta_i(n_1, \cdots, n_{a_i}) \\ &= [\![d_i]\!]_{va}\delta\rho[n_1/x_1, \cdots, n_{a_i}/x_{a_i}] && \text{（由 } \delta \text{ 的不动点定义）} \\ &= [\![d_i[n_1/x_1, \cdots, n_{a_i}/x_{a_i}]]\!]_{va}\delta\rho && \text{（由代入引理）} \\ &= \lfloor n \rfloor && \text{（由假设）}\end{aligned}$$

于是，性质 P 对规则 (fn) 的结论成立。

150 从而，根据规则归纳法，对任何 $t \rightarrow_{va}^{d} n$，性质 $P(t,n)$ 都成立。 □

引理 9.7 设 t 为封闭项，$\rho \in \mathbf{Env}_{va}$，则对所有的 $n \in \mathbf{N}$，有

$$[\![t]\!]_{va}\delta\rho = \lfloor n \rfloor \Rightarrow t \rightarrow_{va}^{d} n$$

证明 根据操作语义，我们首先把函数 $\varphi_i : \mathbf{N}^{a_i} \rightarrow \mathbf{N}_{\perp}\ (i = 1, \cdots, k)$ 定义为

$$\varphi_i(n_1,\cdots,n_{a_i}) = \begin{cases} \lfloor n \rfloor & d_i[n_1/x_1,\cdots,n_{a_i}/x_{a_i}] \rightarrow_{va}^{d} n \\ \perp & \text{其他} \end{cases}$$

我们宣称 $\varphi = (\varphi_1,\cdots,\varphi_k)$ 是9.3 节定义的函数 F 的前缀不动点，因而 $\delta \sqsubseteq \varphi$。该宣称从更一般的归纳假设中得到。

下面我们施结构归纳于 t 证明：如果 t 中的变量包含在变量表 $x_1,\cdots,x_l$ 中，则对所有的 $n,n_1,\cdots,n_l \in \mathbf{N}$，有

$$[\![t]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_l/x_l] = \lfloor n \rfloor \Rightarrow t[n_1/x_1,\cdots,n_l/x_l] \rightarrow_{va}^{d} n \tag{1}$$

（若 t 中没有变量，则变量表为空。）

$t \equiv m$： 此时指称语义和操作语义产生相同的值。

$t \equiv x$（一个变量）：此时 x 必然是某个变量 $x_j(1 \leqslant j \leqslant l)$，显然蕴涵(1)成立。

$t \equiv t_1$ **op** t_2：假设 $[\![t_1 \textbf{ op } t_2]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_l/x_l] = \lfloor n \rfloor$，且假设 t_1,t_2 中的所有变量都出现在变量表 $x_1,\cdots,x_l$ 中。于是，对某个 m_1 和 m_2，有 $n = m_1\ op\ m_2$，其中 m_1 和 m_2 由下式给出：

$$[\![t_1]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_l/x_l] = \lfloor m_1 \rfloor$$
$$[\![t_2]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_l/x_l] = \lfloor m_2 \rfloor$$

由归纳假设得

$$t_1[n_1/x_1,\cdots,n_l/x_l] \rightarrow_{va}^{d} m_1$$
$$t_2[n_1/x_1,\cdots,n_l/x_l] \rightarrow_{va}^{d} m_2$$

因而得到

$$(t_1 \textbf{ op } t_2)[n_1/x_1,\cdots,n_l/x_l] \rightarrow_{va}^{d} m_1\ op\ m_2 \equiv n$$

$t \equiv$ **if** t_0 **then** t_1 **else** t_2： 此时与上面的讨论类似。 151

$t \equiv f_i(t_1,\cdots,t_{a_i})$： 假设

$$[\![f_i(t_1,\cdots,t_{a_i})]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_l/x_l] = \lfloor n \rfloor$$

且 t 中所有的变量包含在变量表 $x_1,\cdots,x_l$ 中。根据指称语义，我们有

$$\begin{aligned}(let\ v_1 \Leftarrow [\![t_1]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_l/x_l],\\ \vdots\\ v_{a_i} \Leftarrow [\![t_{a_i}]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_l/x_l].\ \varphi_i(v_1,\cdots,v_{a_i})) = \lfloor n \rfloor\end{aligned}$$

但同时，必然存在 $m_1,\cdots,m_{a_i} \in \mathbf{N}$，使得

$$[\![t_1]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_l/x_l] = \lfloor m_1 \rfloor$$
$$\vdots$$
$$[\![t_{a_i}]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_l/x_l] = \lfloor m_{a_i} \rfloor$$

我们进一步有

$$\varphi_i(m_1,\cdots,m_{a_i}) = \lfloor n \rfloor$$

现在,由归纳假设我们得到

$$t_1[n_1/x_1,\cdots,n_l/x_l] \rightarrow_{va}^{d} m_1$$
$$\vdots$$
$$t_{a_i}[n_1/x_1,\cdots,n_l/x_l] \rightarrow_{va}^{d} m_{a_i}$$

注意 $\varphi_i(m_1,\cdots,m_{a_i}) = \lfloor n \rfloor$意味着

$$d_i[m_1/x_1,\cdots,m_{a_i}/x_{a_i}] \rightarrow_{va}^{d} n$$

再结合操作语义,我们推导得到

$$f_i(t_1,\cdots,t_{a_i})\ [n_1/x_1,\cdots,n_l/x_l] \rightarrow_{va}^{d} n$$

这就是我们在 $t \equiv f_i(t_1,\cdots,t_{a_i})$的情况下需要证明的。

我们已经对项 t 的所有情况给出了性质(1)的证明。作为(1)的特例,对 $i=1,\cdots,k$,我们得到

152

$$[\![d_i]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_{a_i}/x_{a_i}] = \lfloor n \rfloor \Rightarrow d_i[n_1/x_1,\cdots,n_{a_i}/x_{a_i}] \rightarrow_{va}^{d} n$$

其中 $n,n_1,\cdots,n_{a_i} \in \mathbf{N}$,于是根据 φ 的定义,有

$$\lambda n_1,\cdots,n_{a_1} \in \mathbf{N}.\ [\![d_1]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_{a_1}/x_{a_1}] \sqsubseteq \varphi_1$$
$$\vdots$$
$$\lambda n_1,\cdots,n_{a_k} \in \mathbf{N}.\ [\![d_k]\!]_{va}\varphi\rho[n_1/x_1,\cdots,n_{a_k}/x_{a_k}] \sqsubseteq \varphi_k$$

这样,如前面所宣称的,φ 就是 F 的前缀不动点,从而保证了 $\delta \sqsubseteq \varphi$。

最后,设 t 是一个封闭项,我们可得

$$[\![t]\!]_{va}\delta\rho = \lfloor n \rfloor \Rightarrow [\![t]\!]_{va}\varphi\rho = \lfloor n \rfloor$$

(根据引理 9.3 给出的$[\![t]\!]_{va}$ 的单调性)

$$\Rightarrow t \rightarrow_{va}^{d} n$$

(由性质(1),其中变量表为空) □

定理 9.8 对于封闭项 t,整数 n,以及任意的环境 ρ,有

$$[\![t]\!]_{va}\delta\rho = \lfloor n \rfloor \text{ 当且仅当 } t \rightarrow_{va}^{d} n$$

证明 综合上述两个引理得证。 □

9.5 传名调用的操作语义

下面,我们讨论在传名调用的方式下,给出对 **REC** 的封闭项进行求值的规则。设声明 d 由下列定义方程组成:

$$f_1(x_1,\cdots,x_{a_1}) = d_1$$
$$\vdots$$
$$f_k(x_1,\cdots,x_{a_k}) = d_k$$

我们用关系 $t \rightarrow_{na}^{d} n$ 形式化地表示传名调用方式下的求值计算，即封闭项 t 在传名调用的方式下求值得到整数 n。这种求值计算关系的规则为 153

$$n \rightarrow_{na}^{d} n$$

$$\frac{t_1 \rightarrow_{na}^{d} n_1 \quad t_2 \rightarrow_{na}^{d} n_2}{t_1 \ \mathbf{op}\ t_2 \rightarrow_{na}^{d} n_1 \ op\ n_2}$$

$$\frac{t_0 \rightarrow_{na}^{d} 0 \quad t_1 \rightarrow_{na}^{d} n_1}{\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 \rightarrow_{na}^{d} n_1}$$

$$\frac{t_0 \rightarrow_{na}^{d} n_0 \quad t_2 \rightarrow_{na}^{d} n_2 \quad n_0 \neq 0}{\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 \rightarrow_{na}^{d} n_2}$$

$$\frac{d_i[t_1/x_1,\cdots,t_{a_i}/x_{a_i}] \rightarrow_{na}^{d} n}{f_i(t_1,\cdots,t_{a_i}) \rightarrow_{na}^{d} n}$$

传名调用和传值调用的求值计算规则中只有最后一条不同，在传名调用中函数应用之前不必先求出函数变量的值。同样，这种求值关系是确定的。

命题 9.9 若 $t \rightarrow_{na}^{d} n_1$ 且 $t \rightarrow_{na}^{d} n_2$，则 $n_1 \equiv n_2$。

证明 应用规则归纳法进行证明。 □

9.6 传名调用的指称语义

就传值调用而论，对应于变量和函数变量，把 $\mathbf{N}_\perp$ 中的值赋予项，这个项就获得了含义。按照传名调用的方式，不需要先对函数的变量求值。变量的环境是一个函数

$$\rho : \mathbf{Var} \rightarrow \mathbf{N}_\perp$$

我们用 $\mathbf{Env}_{na}$ 来表示此类环境的完全偏序

$$[\mathbf{Var} \rightarrow \mathbf{N}_\perp]$$ 154

另一方面，函数变量 $f_1,\cdots,f_k$ 的环境由 $\varphi = (\varphi_1,\cdots,\varphi_k)$ 组成，其中每一个

$$\varphi_i : \mathbf{N}_\perp^{a_i} \rightarrow \mathbf{N}_\perp$$

都是连续的函数（$1 \leq i \leq k$）；我们用 $\mathbf{Fenv}_{na}$ 表示函数变量的环境的完全偏序

$$[\mathbf{N}_\perp^{a_1} \rightarrow \mathbf{N}_\perp] \times \cdots \times [\mathbf{N}_\perp^{a_k} \rightarrow \mathbf{N}_\perp]$$

下面，我们继续定义 $[\![t]\!]_{na}$: $\mathbf{Fenv}_{na} \rightarrow [\mathbf{Env}_{na} \rightarrow \mathbf{N}_\perp]$，用结构归纳法给出项 t 的指称：

$$[n]_{na} = \lambda\varphi\lambda\rho.\lfloor n\rfloor$$
$$[x]_{na} = \lambda\varphi\lambda\rho.\rho(x)$$
$$[t_1 \ \mathbf{op}\ t_2]_{na} = \lambda\varphi\lambda\rho.[t_1]_{na}\varphi\rho\ op_\perp\ [t_2]_{na}\varphi\rho$$
$$(\text{其中 } \mathbf{op} \text{ 是 } +, -, \times)$$
$$[\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2]_{na} = \lambda\varphi\lambda\rho.Cond([t_0]_{na}\varphi\rho,[t_1]_{na}\varphi\rho,[t_2]_{na}\varphi\rho)$$
$$[f_i(t_1,\cdots,t_{a_i})]_{na} = \lambda\varphi\lambda\rho.\varphi_i([t_1]_{na}\varphi\rho,\cdots,[t_{a_i}]_{na}\varphi\rho)$$

同样,语义函数也是连续的,并且它在一个环境中的结果与项之外的变量的值无关。

引理 9.10 设 t 为 **REC** 中的项,则指称 $[t]_{na}$ 是一个连续函数 $\mathbf{Fenv}_{na}\to[\mathbf{Env}_{na}\to\mathbf{N}_\perp]$。

证明 使用 8.4 节的结论,用结构归纳法证明。 □

引理 9.11 对于 **REC** 中所有的项 t,如果环境 $\rho,\rho'\in\mathbf{Env}_{na}$ 对出现于 t 中所有的变量产生相同的结果,则对于任何 $\varphi\in\mathbf{Fenv}_{na}$,有

$$[t]_{na}\varphi\rho = [t]_{na}\varphi\rho'$$

特别地,封闭项 t 的指称 $[t]_{na}\varphi\rho$ 独立于环境 ρ。

155 **证明** 直接对项 t 进行结构归纳。 □

声明 d 确定了一个特定的函数环境。设 d 由以下定义方程组成:

$$f_1(x_1,\cdots,x_{a_1}) = d_1$$
$$\vdots$$
$$f_k(x_1,\cdots,x_{a_k}) = d_k$$

定义 F: $\mathbf{Fenv}_{na}\to\mathbf{Fenv}_{na}$ 如下,

$$F(\varphi) = (\lambda z_1,\cdots,z_{a_1}\in\mathbf{N}_\perp.[d_1]_{na}\varphi\rho[z_1/x_1,\cdots,z_{a_1}/x_{a_1}],$$
$$\vdots$$
$$\lambda z_1,\cdots,z_{a_k}\in\mathbf{N}_\perp.[d_k]_{na}\varphi\rho[z_1/x_1,\cdots,z_{a_k}/x_{a_k}])$$

如同 9.4 节中传值调用的求值方式一样,对环境的更新操作可以用 8.4 节的元语言来定义。按照 8.4 节的方法,我们知道 F 是连续的,于是,F 有最小不动点 $\delta = fix(F)$。

例 考察传名调用方式下指称语义的求值计算,考察声明

$$f_1 = f_1 + 1$$
$$f_2(x) = 1$$

按照传名调用的指称语义,该声明的作用是用 $\delta=(\delta_1,\delta_2)\in\mathbf{N}_\perp\times[\mathbf{N}_\perp\to\mathbf{N}_\perp]$ 来指称 f_1,f_2,其中

$$\begin{aligned}(\delta_1,\delta_2) &= \mu\varphi.([f_1+1]_{na}\varphi\rho,\lambda z\in\mathbf{N}_\perp.[1]_{na}\varphi\rho[z/x])\\ &= \mu\varphi.(\varphi_1 +_\perp \lfloor 1\rfloor,\lambda z\in\mathbf{N}_\perp.\lfloor 1\rfloor)\\ &= (\perp,\lambda z\in\mathbf{N}_\perp.\lfloor 1\rfloor)\end{aligned}$$

容易证明,后者即为所需的最小不动点。于是

$$[f_2(f_1)]_{na}\delta\rho = \delta_2(\delta_1) = \lfloor 1 \rfloor$$ □

我们期望,如果 t 是封闭项,则

$$[t]_{na}\delta\rho = \lfloor n \rfloor \text{ 当且仅当 } t \rightarrow^{d}_{na} n$$

在下一节,我们可以证明指称语义和操作语义确实是等价的。 156

9.7 传名调用的语义等价

在传名调用的方式下指称语义和操作语义也是等价的,其证明方法和传值调用方式下的证明方法大致相同。其中一个引理的证明采用规则归纳法,而另一个引理的证明采用不动点归纳法。首先,我们介绍代入引理。

引理 9.12(代入引理) 设 t, t' 为项,设 $\varphi \in \mathbf{Fenv}_{na}$ 且 $\rho \in \mathbf{Env}_{na}$,则

$$[t]_{na}\varphi\rho[[t']_{na}\varphi\rho/x] = [t[t'/x]]_{na}\varphi\rho$$

证明 对 t 进行简单的归纳去证明,留作练习。 □

引理 9.13 设 t 为封闭项,n 为整数,ρ 为变量的环境,则

$$t \rightarrow^{d}_{na} n \Rightarrow [t]_{na}\delta\rho = \lfloor n \rfloor$$

证明 设 ρ 为变量的环境,用规则归纳法来证明下面的性质

$$P(t,n) \Leftrightarrow_{def} [t]_{na}\delta\rho = \lfloor n \rfloor$$

其中 t 为封闭项,n 为整数。惟一复杂一点的规则是

$$\frac{d_i[t_1/x_1,\cdots,t_{a_i}/x_{a_i}] \rightarrow^{d}_{na} n}{f_i(t_1,\cdots,t_{a_i}) \rightarrow^{d}_{na} n}$$

假设 $d_i[t_1/x_1,\cdots,t_{a_i}/x_{a_i}] \rightarrow^{d}_{na} n$ 且 $P(d_i[t_1/x_1,\cdots,t_{a_i}/x_{a_i}],n)$,即

$$[d_i[t_1/x_1,\cdots,t_{a_i}/x_{a_i}]]_{na}\delta\rho = \lfloor n \rfloor$$

我们可推导得到

$$\begin{aligned}
[f_i(t_1,\cdots,t_{a_i})]_{na}\delta\rho &= \delta_i([t_1]_{na}\delta\rho,\cdots,[t_{a_i}]_{na}\delta\rho) \\
&= [d_i]_{na}\delta\rho[[t_1]_{na}\delta\rho/x_1,\cdots,[t_{a_i}]_{na}\delta\rho/x_{a_i}] \\
&\quad \text{(根据 } \delta \text{ 的不动点定义)} \\
&= [d_i[t_1/x_1,\cdots,t_{a_i}/x_{a_i}]]_{na}\delta\rho \\
&\quad \text{(根据代入引理 9.12 的应用,由于每一个 } t_j \text{ 都是封闭的,} \\
&\quad \text{所以 } [t_j]_{na}\delta\rho \text{ 独立于 } \rho) \\
&= \lfloor n \rfloor
\end{aligned}$$

于是,我们有 $P(f_i(t_1,\cdots,t_{a_i}),n)$。容易证明其余的规则也都保持性质 P。从而由规则归纳

157 法,引理得证。 □

下面这个引理的证明使用了基于对最小不动点 δ 的逐次逼近的数学归纳法。由于 $\delta = fix(F)$,因此

$$\delta = \bigsqcup_{r\in\omega} F^r(\perp)$$

其中

$$\begin{aligned} F(\varphi) = (&\lambda z_1,\cdots,z_{a_1} \in \mathbf{N}_\perp.\ [\![d_1]\!]_{na}\varphi\rho[z_1/x_1,\cdots,z_{a_1}/x_{a_1}],\\ &\vdots\\ &\lambda z_1,\cdots,z_{a_k} \in \mathbf{N}_\perp.\ [\![d_k]\!]_{na}\varphi\rho[z_1/x_1,\cdots,z_{a_k}/x_{a_k}]) \end{aligned}$$

记

$$\delta^{(r)} = F^r(\perp)$$

为第 r 次逼近。于是,对所有的 $z_1,\cdots,z_{a_i}\in\mathbf{N}_\perp$,有 $\delta_i^{(0)}(z_1,\cdots,z_{a_i}) = \perp$,其中 $1\leqslant i\leqslant k$。对 $r>0$,有 $\delta^{(r)} = F(\delta^{(r-1)})$,即对 $i=1,\cdots,k$,有

$$\delta_i^{(r)}(z_1,\cdots,z_{a_i}) = [\![d_i]\!]_{na}\delta^{(r-1)}\rho[z_1/x_1,\cdots,z_{a_i}/x_{a_i}]$$

下面的证明中将用到这个递推关系。

引理 9.14 设 t 为封闭项,n 为整数,ρ 为变量的环境,则

$$[\![t]\!]_{na}\delta\rho = \lfloor n\rfloor \Rightarrow t \rightarrow_{na}^{d} n$$

证明 设 ρ 为变量的环境,对封闭项 t,定义

$$res(t) = \begin{cases} \lfloor n\rfloor & t\rightarrow_{na}^{d} n \\ \perp & \text{其他} \end{cases}$$

(即定义了操作语义下 t 的结果。)

如上所述,设 $\delta^{(r)}$ 是递归定义的函数环境 δ 的第 r 次逼近。下面我们对 $r\in\omega$ 进行归纳证明:对于所有的项 t,整数 n,封闭项 $u_1,\cdots,u_s$ 以及包含了出现在 t 中所有变量的 $y_1,\cdots,y_s$,有

$$[\![t]\!]_{na}\delta^{(r)}\rho[res(u_1)/y_1,\cdots,res(u_s)/y_s] = \lfloor n\rfloor \Rightarrow t[u_1/y_1,\cdots,u_s/y_s] \rightarrow_{na}^{d} n \qquad (1)$$

注意,(1)能等价地写成

158 $$[\![t]\!]_{na}\delta^{(r)}\rho[res(u_1)/y_1,\cdots,res(u_s)/y_s] \sqsubseteq res(t[u_1/y_1,\cdots,u_s/y_s])$$

奠基,$r=0$:我们要证明

$$[\![t]\!]_{na} \perp \rho[res(u_1)/y_1,\cdots,res(u_s)/y_s] = \lfloor n\rfloor \Rightarrow t[u_1/y_1,\cdots,u_s/y_s] \rightarrow_{na}^{d} n$$

其中 n 为整数,$u_1,\cdots,u_s$ 为封闭项,以及具有 $\{y_1,\cdots,y_s\}$ 中变量的项 t。施归纳于 t 的结构进行证明。一种基本情况是 t 为变量,此时必然有某个 $y_j(1\leqslant j\leqslant s)$,于是有

$$[\![y_j]\!]_{na} \perp \rho[res(u_1)/y_1,\cdots,res(u_s)/y_s] = res(u_j)$$

根据定义，$res(u_j)=\lfloor n\rfloor$蕴涵着 $y_j[u_1/y_1,\cdots,u_s/y_s]\equiv u_j\rightarrow^{d}_{na}n$。在 t 为$f_i(t_1,\cdots,t_{a_i})$的情况下，有

$$[\![f_i(t_1,\cdots,t_{a_i})]\!]_{na}\perp\rho[res(u_1)/y_1,\cdots,res(u_s)/y_s]=\perp$$

而不是值$\lfloor n\rfloor$，因此，要证明的蕴涵式自动成立。其他的情况都比较简单，留给读者自己去证明。

归纳步骤：假设 $r>0$ 且对$(r-1)$归纳假设成立，我们要证明对所有的整数 n，封闭项 $u_1,\cdots,u_s$，以及具有$\{y_1,\cdots,y_s\}$中变量的项 t，有

$$[\![t]\!]_{na}\delta^{(r)}\rho[res(u_1)/y_1,\cdots,res(u_s)/y_s]=\lfloor n\rfloor\Rightarrow t[u_1/y_1,\cdots,u_s/y_s]\rightarrow^{d}_{na}n$$

同样，施结构归纳于 t 来证明，证明类似于 $r=0$ 的情况，只有当 t 形如$f_i(t_1,\cdots,t_{a_i})$时证明的方法不同。此时令 $\rho'=\rho[res(u_1)/y_1,\cdots,res(u_s)/y_s]$，根据 $\delta^{(r)}$ 的定义，有

$$\begin{aligned}[\![f_i(t_1,\cdots,t_{a_i})]\!]_{na}\delta^{(r)}\rho' &= \delta_i^{(r)}([\![t_1]\!]_{na}\delta^{(r)}\rho',\cdots,[\![t_{a_i}]\!]_{na}\delta^{(r)}\rho')\\ &= [\![d_i]\!]_{na}\delta^{(r-1)}\rho'[[\![t_1]\!]_{na}\delta^{(r)}\rho'/x_1,\cdots,[\![t_{a_i}]\!]_{na}\delta^{(r)}\rho'/x_{a_i}]\end{aligned}$$

$t_j(1\leqslant j\leqslant a_i)$的变量显然位于$\{y_1,\cdots,y_s\}$中，因此，根据结构归纳法，有

$$\begin{aligned}[\![t_j]\!]_{na}\delta^{(r)}\rho' &= [\![t_j]\!]_{na}\delta^{(r)}\rho[res(u_1)/y_1,\cdots,res(u_s)/y_s]\\ &\sqsubseteq res(t_j[u_1/y_1,\cdots,u_s/y_s])\end{aligned}$$

这样，根据指称$[\![d_i]\!]_{na}$的单调性（即引理 9.10 的结论），我们可以推得

$$[\![f_i(t_1,\cdots,t_{a_i})]\!]_{na}\delta^{(r)}\rho'\sqsubseteq[\![d_i]\!]_{na}\delta^{(r-1)}\rho'[res(t_1')/x_1,\cdots,res\ (t_{a_i}')/x_{a_i}]$$ 159

其中，我们把 $t_j[u_1/y_1,\cdots,u_s/y_s]$简记为 $t_j'(1\leqslant j\leqslant a_i)$。现在由数学归纳法，我们可以得到

$$[\![d_i]\!]_{na}\delta^{(r-1)}\rho'[res(t_1')/x_1,\cdots,res\ (t_{a_i}')/x_{a_i}]\sqsubseteq res(d_i[t_1'/x_1,\cdots,t_{a_i}'/x_{a_i}])$$

根据我们对声明的假设，d_i 的变量位于 $x_1,\cdots,x_{a_i}$中。由操作语义，我们注意到

$$res(f_i(t_1',\cdots,t_{a_i}'))=res(d_i[t_1'/x_1,\cdots,t_{a_i}'/x_{a_i}])$$

从而，有

$$\begin{aligned}&[\![f_i(t_1,\cdots,t_{a_i})]\!]_{na}\delta^{(r)}\rho[res(u_1)/y_1,\cdots,res(u_s)/y_s]\\ &\qquad\sqsubseteq res(f_i(t_1,\cdots,t_{a_i})[u_1/y_1,\cdots,u_s/y_s])\end{aligned}$$

于是，在这种情况下，我们要证明的性质成立。

数学归纳法的结果使我们得到结论：对所有的 $r\in\omega$ 以及对任一封闭项 t，有

$$[\![t]\!]_{na}\delta^{(r)}\rho=\lfloor n\rfloor\Rightarrow t\rightarrow^{d}_{na}n$$

根据语义函数的连续性（引理 9.10），得

$$[\![t]\!]_{na}\delta\rho=[\![t]\!]_{na}\bigsqcup_r\delta^{(r)}\rho$$

$$= \bigsqcup_r [\![t]\!]_{na}\delta^{(r)}\rho$$

于是,对某个 $r \in \omega$,$[\![t]\!]_{na}\delta\rho = \lfloor n \rfloor$ 蕴涵 $[\![t]\!]_{na}\delta^{(r)}\rho = \lfloor n \rfloor$,从而蕴涵 $t \rightarrow^{d}_{na} n$。 □

根据这两个引理,我们可以得到传名调用的操作语义和指称语义是等价的。

定理 9.15 设 t 为封闭项,n 为整数,则

$$[\![t]\!]_{na}\delta\rho = \lfloor n \rfloor \text{ 当且仅当 } t \rightarrow^{d}_{na} n$$

练习 9.16 引理 9.14 的证明中使用的方法也可以用于传值调用方式下的证明。试施归纳于逼近次数去证明引理 9.7。 □

160

9.8 局部声明

从程序设计语言的角度来看,**REC** 是相当严格的。特别地,一个 **REC** 程序本质上可以看作是一个序偶,它由待求值的项和确定其函数变量含义的声明组成。大多数函数式程序设计语言都允许函数变量在需要的时候才定义;换句话说,它们都允许下述形式的局部声明:

$$\textbf{let rec } f(x_1,\cdots,x_{a_1}) = d \textbf{ in } t$$

这个声明提供了与待求值的项 t 相对应的 f 的递归定义。这些程序设计语言通常都支持我们在声明中见到过的联立递归,并且允许下面这种更一般的声明:

$$\begin{aligned} &\textbf{let rec } f_1(x_1,\cdots,x_{a_1}) = d_1 \textbf{ and} \\ &\qquad\vdots \\ &\qquad f_k(x_1,\cdots,x_{a_k}) = d_k \\ &\textbf{in } t \end{aligned}$$

它同时递归定义了函数 $f_1,\cdots,f_k$ 的元组。

为了理解这种语言的指称语义,我们考察

$$S \equiv \textbf{let rec } A \Leftarrow t \textbf{ and } B \Leftarrow u \textbf{ in } v$$

的指称,其中 A 和 B 是不同的 0 元函数变量。假设定义中按照传名调用的方式来求值,则 S 在函数环境 $\varphi \in \mathbf{Fenv}_{na}$ 和变量环境 $\rho \in \mathbf{Env}_{na}$ 中的指称为

$$[\![S]\!]\varphi\rho = [\![v]\!]\varphi[\alpha_0/A,\beta_0/B]\rho$$

其中,(α_0,β_0) 是连续函数

$$(\alpha,\beta) \mapsto ([\![t]\!]\varphi[\alpha/A,\beta/B]\rho, [\![u]\!]\varphi[\alpha/A,\beta/B]\rho)$$

的最小不动点。

练习 9.17 扩充 **REC** 语言的语法,使之支持局部声明。试给出在传名调用的方式下扩充后的语言的指称语义。在传值调用的方式下应该如何修改你的语义? □

161

实际上,对于支持单个函数的局部声明的语言来说,虽然联立递归机制提高了它的效率,但并没有增强它的表达能力。例如,上面的程序 S 可替换为

$$T \equiv \textbf{let rec } B \Leftarrow (\textbf{let rec } A \Leftarrow t \textbf{ in } u)$$
$$\textbf{in}(\textbf{let rec } A \Leftarrow t \textbf{ in } v)$$

其中 A 和 B 是不同的 0 元函数变量。它的证明实际上是贝伊克定理的主要内容，我们将在下一章中讨论。

9.9 进一步阅读资料

有关递归方程的语言和语义的其他介绍见文献[59]、[21]、[13]和[58]（后者主要基于[13]），不过它们主要是针对传名调用的情况。Zohar Manna 的著作[59]吸收了 Jean Vuillemin 关于递归方程的研究成果[99]。本章参考了 Robin Milner 讲义的内容，以及 Gordon Plotkin 早期的工作成果（但本章提供了不同的证明）。关于传值调用方式下的证明，读者可以参考 Andrew Pitts 在剑桥大学的授课笔记[75]。通过局部声明扩充的语言的操作语义稍微复杂一点，至少在静态约束中，声明的时候要考虑有关环境的信息，详细讨论见[101]。 162

第10章 递归技术

本章介绍证明连续函数最小不动点的性质的技术。在将利用最小不动点刻画为最小前缀不动点的过程中给出了一种称为帕克（Park）归纳法的方法，用于建立贝伊克（Bekić）定理。贝伊克定理的结论很重要，利用它可以用各种不同的方法求得完全偏序的积的最小不动点。本章还介绍斯科特的不动点归纳法，它依赖于包含性质，我们将介绍包含性质的构造方法。本章专门有一节举例说明良基归纳法的应用，拓展了我们先前的工作，特别是论述了如何构造良基关系。我们还介绍称为良基递归的一般方法，以定义具有良基关系的集合上的函数。本章最后一节讨论一道综合性练习，它运用了多种技术去证明表上两个递归函数的等价性。

10.1 贝伊克定理

根据第5章介绍的不动点定理，如果 D 是一个含底元 $\perp$ 的完全偏序，并且函数 $F: D \to D$ 是连续的，则 $fix(F)$ 是 F 的最小前缀不动点。换句话说，对任意 $d \in D$，

$$F(d) \sqsubseteq d \Rightarrow fix(F) \sqsubseteq d \qquad \text{(fix1)}$$

显然，$fix(F)$ 为不动点，即

$$F(fix(F)) = fix(F) \qquad \text{(fix2)}$$

我们说事实（fix1）和（fix2）刻画了 $fix(F)$，它们在证明不动点的性质时也很重要[⊖]。事实（fix1）表述的证明原理有时称为帕克归纳法，是以 David Park 的姓氏命名的。应用（fix1）和（fix2），我们得到一个有意义的结论——贝伊克定理。贝伊克定理阐明了如何同时用一个坐标的递归定义来代替联立递归定义。

定理 10.1（贝伊克定理） 设 $F: D \times E \to D$ 和 $G: D \times E \to E$ 是连续函数，其中 D 和 E 都是含底的完全偏序，则 $\langle F, G\rangle: D \times E \to D \times E$ 的最小不动点是由以下坐标组成的序偶：

$$\hat{f} = \mu f.\, F(f, \mu g.\, G(\mu f.\, F(f,g), g))$$
$$\hat{g} = \mu g.\, G(\mu f.\, F(f,g), g)$$

163

证明 首先证明 $\langle \hat{f}, \hat{g}\rangle$ 是 $\langle F, G\rangle$ 的一个不动点。由定义得

$$\hat{f} = \mu f.\, F(f, \hat{g})$$

⊖ 事实上，由于 F 是单调的，（fix2）可表示为 $F(fix(F)) \sqsubseteq fix(F)$。于是，由单调性得 $F(F(fix(F))) \sqsubseteq F(fix(F))$，即 $F(fix(F))$ 是前缀不动点。由（fix1）得 $fix(F) \sqsubseteq F(fix(F))$，从而推得（fix2）。

换句话说，$\hat{f}$ 是 $\lambda f.F(f,\hat{g})$ 的最小不动点。因此有 $\hat{f}=F(\hat{f},\hat{g})$。同时，根据 $\hat{g}$ 的定义，有

$$\hat{g}=G(\mu f.F(f,\hat{g}),\hat{g})=G(\hat{f},\hat{g})$$

于是有 $(\hat{f},\hat{g})=\langle F,G\rangle(\hat{f},\hat{g})$，即 $(\hat{f},\hat{g})$ 是 $\langle F,G\rangle$ 的不动点。

设 (f_0,g_0) 是 $\langle F,G\rangle$ 的最小不动点，则必有

$$f_0\sqsubseteq\hat{f}\text{ 且 }g_0\sqsubseteq\hat{g} \tag{1}$$

下面证明另外一个方向。由 $f_0=F(f_0,g_0)$，得

$$\mu f.F(f,g_0)\sqsubseteq f_0$$

根据 G 的单调性，有

$$G(\mu f.F(f,g_0),g_0)\sqsubseteq G(f_0,g_0)=g_0$$

又因为 $\hat{g}$ 是 $\lambda g.G(\mu f.F(f,g),g)$ 的最小前缀不动点，所以

$$\hat{g}\sqsubseteq g_0 \tag{2}$$

根据 F 的单调性，有

$$F(f_0,\hat{g})\sqsubseteq F(f_0,g_0)=f_0$$

同样因为 $\hat{f}$ 是 $\lambda f.F(f,\hat{g})$ 的最小前缀不动点，所以

$$\hat{f}\sqsubseteq f_0 \tag{3}$$

综合(1)、(2)、(3)，即可得到 $(\hat{f},\hat{g})=(f_0,g_0)$，证毕。 □

我们仅依靠单调性和上述(fix1)、(fix2)所表述的最小不动点的性质就完成了证明。因此，如果单调函数是定义在格上的（见 5.5 节），同样可以使用这种证明方法。

贝伊克定理给出了联立最小不动点的不对称形式，我们还可以作为一个推论推导出对称形式：联立最小不动点是序偶

$$\hat{f}=\mu f.F(f,\mu g.G(f,g))$$
$$\hat{g}=\mu g.G(\mu f.F(f,g),g)$$

我们注意到，第二个方程是贝伊克定理的直接结论，而第一个方程是根据 f 和 g 之间的对称性得出的。

164

例 在 9.8 节中，我们提到了如何扩展 **REC** 语言使之允许局部声明。考察项

$$T\equiv\textbf{let rec }B\Leftarrow(\textbf{let rec }A\Leftarrow t\textbf{ in }u)\ \textbf{in }(\textbf{let rec }A\Leftarrow t\textbf{ in }v)$$

其中 A 和 B 是不同的 0 元函数变量。设 ρ,φ 是任意的变量环境及函数变量环境。记

$$F(f,g) = [\![t]\!]\varphi[f/A,g/B]\rho$$
$$G(f,g) = [\![u]\!]\varphi[f/A,g/B]\rho$$

由语义我们可以看出

$$[\![T]\!]\varphi\rho = [\![v]\!]\varphi[\hat{f}/A,\hat{g}/B]\rho$$

其中

$$\begin{aligned}\hat{g} &= \mu g.\ [\![\textbf{let rec } A \Leftarrow t \textbf{ in } u]\!]\varphi[g/B]\rho \\ &= \mu g.\ [\![u]\!]\varphi[g/B,\mu f.\ [\![t]\!]\varphi[f/A,g/B]\rho/A]\rho \\ &= \mu g.\ G(\mu f.\ F(f,g),g)\end{aligned}$$

而

$$\begin{aligned}\hat{f} &= \mu f.\ [\![t]\!]\varphi[f/A,\hat{g}/B]\rho \\ &= \mu f.\ F(f,\hat{g})\end{aligned}$$

根据贝伊克定理,这意味着$(\hat{f},\hat{g})$是$\langle F,G\rangle$的(联立)最小不动点,因此用联立声明也能达到同样的效果;我们有

$$[\![T]\!] = [\![\textbf{let rec } A \Leftarrow t \textbf{ and } B \Leftarrow u \textbf{ in } v]\!]$$

对于函数变量,不管是用传名调用还是传值调用,其证明过程本质上是相同的。很显然,要证明包括联立声明的项和其他项之间的程序等价性,贝伊克定理是十分重要的。 □

练习 10.2 试推广并表述贝伊克定理,使之适用于三个方程的情形。 □

练习 10.3 设 D 和 E 是含底的完全偏序,试证明:如果 $f: D \to E$ 及 $g: E \to D$ 是完全偏序 D 和 E 上的连续函数,则

$$fix(g \circ f) = g(fix(f \circ g))$$

(提示:使用上述(fix1)和(fix2)。) □ [165]

10.2 不动点归纳法

要证明最小不动点满足某个性质,通常只要用数学归纳法证明每个逼近点都满足该性质即可。如第 5 章中定理 5.7 的证明就属于这种情况。定理 5.7 证明了操作语义与指称语义是等价的,在证明

$$(\sigma,\sigma') \in \mathscr{C}[\![c]\!] \Rightarrow \langle c,\sigma\rangle \to \sigma'$$

对于状态 σ,σ'和 c 为 while 循环命令成立时,就是通过对指称的逼近点进行数学归纳而证明的。在定理 5.7 的证明中,很明显,如果一个最小不动点的所有逼近点都满足某个性质,那么这些逼近点的并,即不动点本身也必定满足该性质。不过要注意的是并不是所有的性质都适用这种方法。

不动点归纳法是斯科特提出的,它是证明连续函数的最小不动点的性质的有效方法。作

为一种证明原理,不动点归纳法实质上替代了数学归纳法,数学归纳法是对连续函数 F 的最小不动点 $\bigsqcup_n F^n(\perp)$ 的逼近点 $F^n(\perp)$ 进行归纳。然而,不动点归纳法避免了对整数进行归纳,只对包含性质进行归纳;包含性质确保对最小不动点的所有逼近点性质成立蕴涵着对该不动点本身性质也成立。

定义 设 D 是完全偏序,称 D 的子集 P 是包含的当且仅当对于 D 中所有的 ω 链 $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$,如果对所有的 $n \in \omega, d_n \in P$,则 $\bigsqcup_{n\in\omega} d_n \in P$。 □

从下面称为不动点归纳法的证明原理可看出包含子集的重要性,它由下面的命题给出。

命题 10.4(不动点归纳法——斯科特) 设 D 是含底元素 $\perp$ 的完全偏序,$F: D \to D$ 是连续函数,P 是 D 的包含子集。如果 $\perp \in P$ 且 $\forall x \in D. x \in P \Rightarrow F(x) \in P$,则 $fix(F) \in P$。

证明 我们有 $fix(F) = \bigsqcup_n F^n(\perp)$,如果 P 是满足以上条件的包含子集,则 $\perp \in P$,所以 $F(\perp) \in P$。由归纳法可得 $F^n(\perp) \in P$。显然,经归纳,这些逼近点形成了一条 ω 链

$$\perp \sqsubseteq F(\perp) \sqsubseteq \cdots \sqsubseteq F^n(\perp) \sqsubseteq \cdots$$

166 因而由 P 的包含性,我们得到 $fix(F) \in P$。 □

练习 10.5 Ω 为完全偏序:

$$0 \sqsubseteq 1 \sqsubseteq \cdots \sqsubseteq n \sqsubseteq \cdots \infty$$

试求 Ω 的包含子集。 □

练习 10.6 完全偏序的斯科特闭子集是斯科特开子集的补集(见练习 8.4 的定义)。试证明斯科特闭子集是包含的,并举例说明存在一个不是斯科特闭子集的包含子集。 □

作为不动点归纳法的第一个简单应用,下面我们讨论如何由不动点归纳法得到上一节介绍的帕克归纳法。

命题 10.7 设 $F: D \to D$ 是含底的完全偏序 D 上的连续函数,$d \in D$,如果 $F(d) \sqsubseteq d$,则 $fix(F) \sqsubseteq d$。

证明 (使用不动点归纳法)

设 $d \in D$ 且 $F(d) \sqsubseteq d$,则子集

$$P = \{x \in D \mid x \sqsubseteq d\}$$

是包含的——如果 ω 链 $d_0 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$ 的每一个元素都小于 d,则其最小上界 $\bigsqcup_n d_n$ 也小于 d。显然 $\perp \sqsubseteq d$,从而 $\perp \in P$。下面我们证明 $x \in P \Rightarrow F(x) \in P$。设 $x \in P$,即 $x \sqsubseteq d$,由于 F 是单调的,从而 $F(x) \sqsubseteq F(d) \sqsubseteq d$,于是 $F(x) \in P$。由不动点归纳法我们得到 $fix(F) \in P$,即 $fix(F) \sqsubseteq d$。 □

当然,这个证明不过是用不动点归纳法去间接证明一个已知事实,但是这确实也说明了不动点归纳法的能力不会比帕克归纳法弱。实际上,一些用帕克归纳法无法证明的最小不动点的性质却可以用不动点归纳法来证明。

谓词 $Q(x_1, \cdots, x_k)$ 的自由变量 $x_1, \cdots, x_k$ 分别取值于完全偏序 $D_1, \cdots, D_k$,它确定了 $D_1 \times$

$\cdots \times D_k$ 的一个子集，即集合

$$P = \{(x_1,\cdots,x_k) \in D_1 \times \cdots \times D_k \mid Q(x_1,\cdots,x_k)\}$$

此时我们说如果完全偏序 $D_1 \times \cdots \times D_k$ 的子集是包含的，那么谓词 $Q(x_1,\cdots,x_k)$ 是包含的。同其他的归纳原理一样，在进行不动点归纳时，我们应该尽量使用谓词，而不是使用它们的看作集合的外延。不动点归纳法可以表述为： [167]

设 $F: D_1 \times \cdots \times D_k \to D_1 \times \cdots \times D_k$ 是含有底元素 $(\perp_1,\cdots,\perp_k)$ 的完全偏序积 $D_1 \times \cdots \times D_k$ 上的连续函数，$Q(x_1,\cdots,x_k)$ 是 $D_1 \times \cdots \times D_k$ 上的包含谓词，

如果 $Q(\perp_1,\cdots,\perp_k)$ 且

$\forall x_1 \in D_1,\cdots,x_k \in D_k.\ Q(x_1,\cdots,x_k) \Rightarrow Q(F(x_1,\cdots,x_k))$

则 $Q(fix(F))$。

幸好，我们能确保一大类集合和谓词都是包含的，因为它们是用下述方式构造出来的。

基本关系 设 D 为完全偏序，二元关系

$$\{(x,y) \in D \times D \mid x \sqsubseteq y\} \text{ 和 } \{(x,y) \in D \times D \mid x = y\}$$

是 $D \times D$ 的包含子集（请读者思考为什么?）。从而谓词

$$x \sqsubseteq y, \quad x = y$$

是包含的。

逆象与代入 设 $f: D \to E$ 为完全偏序 D 与 E 之间的连续函数，P 是 E 的包含子集，则逆象

$$f^{-1}P = \{x \in D \mid f(x) \in P\}$$

是 D 的包含子集。

这样在代入的时候，如果所代入的项对包含谓词的变量是连续的，那么包含谓词对用项代入其变量的运算是封闭的。令 $Q(y_1,\cdots,y_l)$ 为 $E_1 \times \cdots \times E_l$ 的包含谓词，也就是说，

$$P =_{def} \{(y_1,\cdots,y_l) \in E_1 \times \cdots \times E_l \mid Q(y_1,\cdots,y_l)\}$$

是 $E_1 \times \cdots \times E_l$ 的包含子集。设表达式 $e_1,\cdots,e_l$ 分别表示 $E_1,\cdots,E_l$ 中元素的表达式，它们对依次取值于 $D_1,\cdots,D_k$ 的变量 $x_1,\cdots,x_k$ 连续——用 8.4 节中的元语言去描述表达式可以确保这一点。于是，定义 f 为

$$\lambda x_1,\cdots,x_k.\ (e_1,\cdots,e_l)$$

确保 f 一定是连续函数。因而，$f^{-1}P$ 为 $D_1 \times \cdots \times D_k$ 的一个包含子集。但这仅意味着

$$\{(x_1,\cdots,x_k) \in D_1 \times \cdots \times D_k \mid Q(e_1,\cdots,e_l)\}$$ [168]

是一个包含子集，从而，$Q(e_1,\cdots,e_l)$ 是 $D_1 \times \cdots \times D_k$ 的一个包含谓词。

例如，取 $f = \lambda x \in D.\ (x,c)$，我们可以看出，如果 $R(x,y)$ 是 $D \times E$ 的一个包含谓词，则把 y 固定为常量 c 而得到的谓词 $Q(x) \Longleftrightarrow_{def} R(x,c)$ 是 D 的一个包含谓词。固定包含谓词中的一

个或几个变量后得到的谓词仍然是包含谓词。

练习 10.8 试证明:如果 $Q(x)$ 是完全偏序 D 的包含谓词,则

$$R(x,y) \Longleftrightarrow_{def} Q(x)$$

是 $D \times E$ 的包含谓词,其中额外的变量 y 取值于完全偏序 E。(这样,我们就可以用额外的变量来"填充"包含谓词。提示:投影函数。) □

逻辑运算 设 D 是一个完全偏序,子集 D 和 $\emptyset$ 是包含的。谓词"true"与"false"分别为 D 和 $\emptyset$ 的外延,所以也是包含的。设 $P \subseteq D$ 和 $Q \subseteq D$ 都是 D 的包含子集,则

$$P \cup Q \text{ 和 } P \cap Q$$

也是包含子集。从谓词的角度来看,如果谓词 $P(x_1,\cdots,x_k)$ 和谓词 $Q(x_1,\cdots,x_k)$ 是包含谓词,那么谓词

$$P(x_1,\cdots,x_k) \vee Q(x_1,\cdots,x_k), \quad P(x_1,\cdots,x_k) \,\&\, Q(x_1,\cdots,x_k)$$

也是包含谓词。如果 $P_i(i \in I)$ 是 D 上包含子集的加标族,则 $\cap_{i \in I} P_i$ 也是 D 的包含子集。因此,如果 $P(x_1,\cdots,x_k)$ 是 $D_1 \times \cdots \times D_k$ 上的包含谓词,则 $\forall x_i \in D_i.\, P(x_1,\cdots,x_k)\,(1 \leqslant i \leqslant k)$ 是 D 上的包含谓词。这是因为相应的子集

$$\{(x_1,\cdots,x_{i-1},x_{i+1},\cdots,x_k) \in D_1 \times \cdots \times D_{i-1} \times D_{i+1} \times \cdots \times D_k \mid \forall x_i \in D_i.\, P(x_1,\cdots,x_k)\}$$

等于包含子集的广义交集

$$\bigcap_{d \in D_i} \{(x_1,\cdots,x_{i-1},x_{i+1},\cdots,x_k) \in D_1 \times \cdots \times D_{i-1} \times D_{i+1} \times \cdots \times D_k \mid P(x_1,\cdots,x_{i-1},d,x_{i+1},\cdots,x_k)\}$$

对于 $d \in D_i$,每一个谓词 $P(x_1,\cdots,x_{i-1},d,x_{i+1},\cdots,x_k)$ 都是包含的,因为它们是通过固定一个变量而得到的。

然而,要注意的是无穷个包含子集的并不一定是包含的,相应地,包含谓词对存在量词 $\exists$ 一般不封闭。

169

练习 10.9

(i) 试给出一个反例,以说明包含谓词对存在量词 $\exists$ 一般不封闭。

(ii) 试证明:对于完全偏序 D 和 E 之间的连续函数 $f: D \to E$ 下,包含子集 $P \subseteq D$ 的正象 fP 不一定是 E 的包含子集。

(iii) 试举出包含子集 $P \subseteq D \times E$ 和 $Q \subseteq E \times F$ 的例子,使得它们的复合关系

$$Q \circ P =_{def} \{(d,f) \mid \exists e \in E.\, (d,e) \in P \,\&\, (e,f) \in Q\}$$

不是包含的。

((iii) 的提示:取 D 为单元素完全偏序 $\{\top\}$,E 为非负整数集 ω 的离散完全偏序,F 为由 ω 链及其最小上界 ∞ 组成的完全偏序 Ω。) □

虽然对一般的连续函数来说,包含子集的正象不一定是包含的,但是在相对于序关系是内

射的(*order-monic*)函数下的正象却保持包含性。设 D、E 是完全偏序,连续函数 $f: D \to E$ 是相对于序关系内射的当且仅当对所有的 $d, d' \in D$,有

$$f(d) \sqsubseteq f(d') \Rightarrow d \sqsubseteq d'$$

相对于序关系内射的例子包括"提升"函数$\lfloor - \rfloor$以及与和有关的内射 in_i。容易看出,如果 f 是相对于序关系内射的,且 P 是 D 的包含子集,那么它的正象 fP 也是 D 的包含子集。这就意味着,如果 $Q(x)$ 是 D 的包含谓词,则具有自由变量 $y \in E$ 的谓词

$$\exists x \in D. y = f(x) \ \& \ Q(x)$$

也是 E 的包含谓词。

下面我们考察与特定的完全偏序和完全偏序上的构造方法相关的包含子集和包含谓词。

离散完全偏序 离散完全偏序的任何子集与任何谓词都是包含的。

积 设 $P_i \subseteq D_i (1 \leq i \leq k)$ 是包含子集,则

$$P_1 \times \cdots \times P_k = \{(x_1, \cdots, x_k) \mid x_1 \in P_1 \ \& \cdots \& \ x_k \in P_k\}$$

是积 $D_1 \times \cdots \times D_k$ 的包含子集。这可由前面的结论和下面的事实证明。对于 $i = 1, \cdots, k$,我们有

$$P_1 \times \cdots \times P_k = \pi_1^{-1} P_1 \cap \cdots \cap \pi_k^{-1} P_k$$ 170

因为每一个逆象 $\pi_i^{-1} P_i$ 都是包含的,所以它们的交也是包含的。

注意 设 $D_1, \cdots, D_k$ 为完全偏序,人们容易误认为:如果谓词 $P(x_1, \cdots, x_k)$(其中 $x_1 \in D_1, \cdots, x_k \in D_k$)在积 $D_1 \times \cdots \times D_k$ 中的每一个变量上都是包含的,则它是积 $D_1 \times \cdots \times D_k$ 上的包含谓词。但是事实并非如此。更精确地说,如果对每一个 $i = 1, \cdots, k$,通过固定除第 i 项以外的所有变量而得到的谓词 $P(d_1, \cdots, d_{i-1}, x_i, d_{i+1}, \cdots, d_k)$ 是 D_i 的包含谓词,则称谓词 $P(x_1, \cdots, x_k)$ 在每一个变量上是包含的。当然,如果 $P(x_1, \cdots, x_k)$ 是包含的,则它在其所有的变量上都是包含的——我们可以用常元来代入某些变量,由前面的讨论知,这样可以保持包含性。但反之却不成立。$P(x_1, \cdots, x_k)$ 在其每一个变量上是包含的并不蕴涵着它是 $D_1 \times \cdots \times D_k$ 的包含谓词。

练习 10.10 设 Ω 为由 ω 和 ∞ 组成的完全偏序:

$$0 \sqsubseteq 1 \sqsubseteq \cdots \sqsubseteq n \sqsubseteq \cdots \sqsubseteq \infty$$

考察谓词

$$P(x, y) \Longleftrightarrow_{def} (x = y \ \& \ x \neq \infty)$$

试证明:一个谓词在每一个变量上是包含的并不蕴涵着它是包含的。 □

函数空间 设 D 和 E 是完全偏序,$P \subseteq D$ 和 $Q \subseteq E$ 是包含子集,则

$$P \to Q =_{def} \{f \in [D \to E] \mid \forall x \in P. f(x) \in Q\}$$

是函数空间$[D \to E]$的包含子集(请读者思考为什么?)。于是,当 $P(x)$ 是 D 的谓词且 $Q(y)$

是 E 的包含谓词时，具有自由变量 $f \in [D \to E]$ 的谓词 $\forall x \in D.\ P(x) \Rightarrow Q(f(x))$ 是包含谓词。

提升 设 P 是完全偏序 D 的包含子集。因为函数 $\lfloor - \rfloor$ 是有序单调的，所以正象 $\lfloor P \rfloor = \{\lfloor d \rfloor \mid d \in P\}$ 是 $D_\perp$ 的包含子集。如果 $Q(x)$ 是 D 的包含谓词，则具有自由变量 $y \in D_\perp$ 的谓词

$$\exists x \in D.\ y = \lfloor x \rfloor \ \&\ Q(x)$$

是 $D_\perp$ 上的包含谓词。

和 设 P_i 是完全偏序 D_i 的包含子集，其中 $i = 1, \cdots, k$，则

171

$$P_1 + \cdots + P_k = in_1 P_1 \cup \cdots \cup in_k P_k$$

是和 $D_1 + \cdots + D_k$ 的包含子集。这是因为每一个内射都是相对于序关系内射的，因此每一个 $in_i P_i$ 都是包含的，而有限个包含集合的并也是包含的。我们用谓词来表达这一事实，得到：如果每一个 $Q_i(x_i)$ 是成分 D_i 的包含谓词，则具有自由变量 $y \in D_1 + \cdots + D_k$ 的谓词

$$(\exists x_1 \in D_1.\ y = in_1(x_1)\ \&\ Q_1(x_1))\ \text{or} \cdots \text{or}\ (\exists x_k \in D_k.\ y = in_k(x_k)\ \&\ Q_k(x_k))$$

是和的包含谓词。

上面的描述方法构成了关于包含谓词的语言的基础。只要按照上面的方法从基本的包含谓词去构造谓词，则所构造的谓词一定是包含的。例如，任何谓词，只要它是从形如 $e_1 \sqsubseteq e_2$ 的基本谓词通过合取和析取构造，然后对其中的一些变元进行全称约束而得到的（其中 e_1、e_2 是元语言中的项），那么这个谓词一定是包含的。

命题 10.11 任何形为

$$\forall x_1, \cdots, x_n.\ P$$

的谓词都是包含谓词，其中 $x_1, \cdots, x_n$ 是特定的完全偏序上的变量，P 是由形如 $e_0 \sqsubseteq e_1$ 或 $e_0 = e_1$ 的基本谓词按合取和析取的方式构成的，而 e_1、e_2 是用 8.4 节介绍的关于表达式的元语言给出的表达式。

但是，这样的语法不能生成证明中所需要的所有谓词，因而当论证递归定义的域时，生成合适的包含谓词变得非常困难。

例 设 $\mathbf{T}_\perp$ 为真值 $\{\mathbf{true}, \mathbf{false}\}_\perp$ 的完全偏序，我们把 $\lfloor \mathbf{true} \rfloor$ 缩写为 tt，把 $\lfloor \mathbf{false} \rfloor$ 缩写为 ff。设 $p: D \to \mathbf{T}_\perp$ 和 $h: D \to D$ 都是连续的，其中 h 是严格的（即 $h(\perp) = \perp$）。设 $f: D \times D \to D$ 是使得对所有的 $x, y \in D$，有

$$f(x,y) = p(x) \to y \mid h(f(h(x), y))$$

的最小连续函数，我们要证明

(i) 对所有的 $b \in \mathbf{T}_\perp$ 和 $d, e \in D$，有 $h(b \to d \mid e) = b \to h(d) \mid h(e)$。

(ii) 对所有的 $x, y \in D$，有 $h(f(x,y)) = f(x, h(y))$。

(i) 的证明很容易，只要考察 $b \in \mathbf{T}_\perp$ 的三个可能的值 $\perp, tt, ff$ 即可。

$$\text{如果 } b = \bot, \text{则 } h(b \to d \mid e) = h(\bot) = \bot = b \to h(d) \mid h(e)$$
$$\text{如果 } b = tt, \text{则 } h(b \to d \mid e) = h(d) = b \to h(d) \mid h(e)$$
$$\text{如果 } b = ff, \text{则 } h(b \to d \mid e) = h(e) = b \to h(d) \mid h(e)$$

172

因此,对布尔变量 b 的所有可能的值,所需方程都成立。

(ii) 用不动点归纳法证明。一个合适的谓词是

$$P(g) \Longleftrightarrow_{def} \forall x,y \in D.\, h(g(x,y)) = g(x,h(y))$$

谓词 $P(g)$ 可用上面的方法构造得到,因而是包含的。又因为 h 是严格的,所以 $P(\bot)$ 为真。为了应用不动点归纳法,我们进一步要证明

$$P(g) \Rightarrow P(F(g))$$

其中 $F(g) = \lambda x,y.\, p(x) \to y \mid h(g(h(x),y))$。

假设 $P(g)$ 成立,令 $x,y \in D$,则

$$\begin{aligned} h((F(g))(x,y)) &= h(p(x) \to y \mid h(g(h(x),y))) \\ &= p(x) \to h(y) \mid h^2(g(h(x),y)) \quad (\text{由(i)}) \\ &= p(x) \to h(y) \mid h(g(h(x),h(y))) \quad (\text{由假设 } P(g)) \\ &= (F(g))(x,h(y)) \end{aligned}$$

于是推得 $P(F(g))$ 成立,因而 $P(g) \Rightarrow p(F(g))$。

根据不动点归纳法,我们得到 $P(fix(F))$ 成立,即 $P(f)$ 成立,因此,$\forall x,y \in D.\, h(f(x,y)) = f(x,h(y))$ 成立。 □

练习 10.12 递归定义 $h: \mathbf{N} \to \mathbf{N}_\bot$ 为

$$h(x) = h(x) +_\bot \lfloor 1 \rfloor$$

试用不动点归纳法证明:函数 $h = \bot$,即 h 处处取值 $\bot$。 □

练习 10.13 设 D 是含底的完全偏序,$p: D \to \mathbf{T}_\bot$ 是严格的(即 $p(\bot) = \bot$)连续函数,而 $h: D \to D$ 是连续函数。设 $f: D \to D$ 是满足下式的最小连续函数:对所有的 $x \in D$,

$$f(x) = p(x) \to x \mid f(f(h(x)))$$

试证明

$$\forall x \in D.\, f(f(x)) = f(x)$$

提示:取归纳假设为谓词

$$P(g) \Longleftrightarrow_{def} \forall x \in D.\, f(g(x)) = g(x)$$

□ 173

练习 10.14 设 $h,k: D \to D$ 是含底的完全偏序 D 上的连续函数,其中 h 是严格的。设 $p: D \to \mathbf{T}_\bot$ 是连续函数。设对所有的 $x,y \in D$,f,g 是满足下式的最小连续函数 $D \times D \to D$:

$$f(x,y) = p(x) \to y \mid h(f(k(x),y))$$
$$g(x,y) = p(x) \to y \mid g(k(x),h(y))$$

试用不动点归纳法证明 $f=g$。

提示:把所求的解作为联立不动点,并取包含谓词为

$$P(f,g) \Longleftrightarrow_{def} \forall x,y.\,[f(x,y) = g(x,y) \& g(x,h(y)) = h(g(x,y))] \qquad \square$$

最后,我们有必要总结一下不动点归纳法。当需要证明最小不动点满足某一个性质时,找到一个适合不动点归纳法的包含性质并不容易。无论是归纳假设还是程序不变式,要找一个合适的包含性质通常需要深刻的洞察力。寻找合适的包含谓词的过程通常能使实际的证明变得简单。有时候我们可以先找出最小不动点的头几个逼近点,试图找到一个可以转变成归纳假设的模式。然后,我们再对逼近点进行数学归纳(如果所有的逼近点都满足某一性质蕴涵着最小不动点也满足该性质),或者使用更简洁的不动点归纳法(只要该性质是包含的)。

10.3 良基归纳

有些论证不适合用不动点归纳法。例如,我们要证明递归定义的整数函数对于整数输入总是终止的。这就无法直接用不动点归纳法来证明。如果要用不动点归纳法的话,我们就需要一个表示终止的包含谓词 P,且它在完全无定义的函数 $\perp$ 上为真。所以,我们需要寻找一个新的证明原理,它能够充分利用计算的对象是递归定义的这一事实。这个证明原理就是良基归纳法。回忆第 3 章的内容,集合 A 上的良基关系是一个不包含无穷下降链的二元关系 $<$。

174 良基归纳原理为:

设 $<$ 为集合 A 上的一个良基关系,P 为性质,则 $\forall a \in A.\,P(a)$ 成立当且仅当

$$\forall a \in A.\,([\forall b < a.\,P(b)] \Rightarrow P(a))$$

应用良基归纳法的关键在于选择一个合适的良基关系。在前面的证明中,我们已经使用过良基关系,比如语法集合上的真子表达式关系,自然数上的 $<$ 关系等。下面我们介绍一些构造良基关系的著名方法。注意,这里我们用 $x \le y$ 表示($x<y$ 或者 $x=y$)。

积 如果 $<_1$ 在 A_1 上是良基的,$<_2$ 在 A_2 上是良基的,则

$$(a_1,a_2) \le (a_1',a_2') \Longleftrightarrow_{def} a_1 \le_1 a_1' \text{ 且 } a_2 \le_2 a_2'$$

确定了 $A_1 \times A_2$ 的一个良基关系 $< = (\le \backslash 1_{A_1 \times A_2})$。但是,积关系不像按字典序产生的关系那样得到广泛应用。

字典序积 设 $<_1$ 在 A_1 上是良基的,$<_2$ 在 A_2 上是良基的,定义

$$(a_1,a_2) <_{lex} (a_1',a_2') \text{ 当且仅当 } a_1 <_1 a_1' \text{ 或 } (a_1 = a_1' \,\&\, a_2 <_2 a_2')$$

逆象 设 $f: A \to B$ 是一个函数,且 $<_B$ 是 B 上的一个良基关系,则 $<_A$ 是 A 上的一个良基关系,其中对于 $a,a' \in A$,有

$$a <_A a' \Longleftrightarrow_{def} f(a) <_B f(a')$$

练习 10.15 设 $<$ 是集合 X 上的一个良基关系,使得 $\le$ 是全序的。证明:对所有的 $y \in X$,集合

$$\{x \in X \mid x < y\}$$

未必是有穷的。

（全序是这样的一个偏序关系≤，使得对它的所有的元素 x,y，均有 $x\leq y$ 或 $y\leq x$。）

（提示：考察 $\omega\times\omega$ 上 $<$ 与 $<$ 的字典序积。） □

练习 10.16 试证明从已知的良基关系中按照积、字典序积以及逆象构造出的关系确实是良基关系。 □

例 著名的例子是 Ackermann 函数，它可以用 **REC** 语言通过下面的声明来定义： 175

$$
\begin{aligned}
A(x,y) = {} & \textbf{if } x \textbf{ then } y+1 \textbf{ else} \\
& \textbf{if } y \textbf{ then } A(x-1,1) \textbf{ else} \\
& \qquad A(x-1,A(x,y-1))
\end{aligned}
$$

在传值调用的指称语义下，这声明了函数 A 指称 $[\mathbf{N}^2\to\mathbf{N}_\perp]$ 中最小函数 a，使得对所有的 m，$n\in\mathbf{N}$，有

$$
a(m,n) = \begin{cases} \lfloor n+1 \rfloor & m=0 \\ a(m-1,1) & m\neq 0, n=0 \\ let\ l \Leftarrow a(m,n-1).\ a(m-1,l) & \text{其他} \end{cases}
$$

Ackermann 函数 $a(m,n)$ 的计算是终止的（整数 $m,n\geq 0$），它可以用字典序 (m,n) 上的良基归纳法来证明。 □

练习 10.17 试用良基归纳法证明 Ackermann 函数 $a(m,n)$ 的计算是终止的（整数 $m,n\geq 0$），取归纳假设为

$$P(m,n) \Longleftrightarrow_{def} (a(m,n)\neq\perp \text{ 且 } a(m,n)\geq 0)$$

其中 $m,n\geq 0$。 □

练习 10.18 McCarthy 的 91 函数定义为 $[\mathbf{N}\to\mathbf{N}_\perp]$ 中的最小函数，使得

$$f(x) = cond(x>100, \lfloor x-10 \rfloor, let\ y \Leftarrow f(x+11).f(y))$$

（这里使用了 8.3.5 节的条件操作。）

试证明：对所有非负整数 x，有

$$f(x) = cond(x>100, \lfloor x-10 \rfloor, \lfloor 91 \rfloor)$$

提示：用关系

$$n \prec m \Longleftrightarrow m < n \leq 101,$$

在对 ω 上进行良基归纳，其中 $n,m\in\omega$。先证明 $\prec$ 是良基关系。 □

10.4 良基递归

我们在第 3 章中看到，归纳定义和结构归纳定义都是递归定义的形式，例如，算术表达式的长度可以用它的严格子表达式的长度来定义；长度函数对特定参数（例如，(a_1+a_2)）的作用可以分解为对更小的参数（例如，a_1 和 a_2）的作用。我们可以用类似的方法定义任意良基 176

集上的函数。设 B 是具有良基关系 $<$ 的集合，良基归纳（又称作良基递归）定义 B 的函数 f 的时候，B 中任一点 b 的值 $f(b)$ 是由 $f(b')$ 来给出的，其中 $b'<b$。我们需要一些记号以精确地表述和验证一般的良基递归方法：B 中每一个元素 b 都有一个前趋集合

$$<^{-1}\{b\} = \{b' \in B \mid b' < b\}$$

对任一 $B' \subseteq B$，函数 $f: B \to C$ 可被限制到如下的函数 $f \restriction B': B' \to C$：

$$f \restriction B' = \{(b, f(b)) \mid b \in B'\}$$

良基递归定义的正确性由下面的定理证实。

定理 10.19（良基递归） 设 $<$ 为集合 B 上的良基关系。假设对于所有的 $b \in B$ 以及函数 $h: <^{-1}\{b\} \to C$，有 $F(b,h) \in C$，则存在一个惟一的函数 $f: B \to C$ 使得

$$\forall b \in B. f(b) = F(b, f \restriction <^{-1}\{b\}) \qquad (*)$$

证明 这个证明包含两部分。我们先来证明惟一性：对任何 $x \in B$，

$$\forall y <^* x. f(y) = F(y, f \restriction <^{-1}\{y\}) \ \&\ g(y) = F(y, g \restriction <^{-1}\{y\}) \Rightarrow f(x) = g(x)$$

通过施良基归纳于 $<$ 来证明 $P(x)$ 的惟一性：对 $x \in B$，假设对每一个 $z<x$，$P(z)$ 都成立，我们要证明 $P(x)$ 成立。为此，假设对所有的 $y <^* x$，有

$$f(y) = F(y, f \restriction <^{-1}\{y\}) \ \&\ g(y) = F(y, g \restriction <^{-1}\{y\})$$

如果 $z<x$，则由 $P(z)$ 成立可得

$$f(z) = g(z)$$

于是

$$f \restriction <^{-1}\{x\} = g \restriction <^{-1}\{x\}$$

现在，我们可得

177

$$f(x) = F(x, f \restriction <^{-1}\{x\}) = F(x, g \restriction <^{-1}\{x\}) = g(x)$$

因而得到 $P(x)$ 成立。

这就得出，满足（$*$）的函数至多只有一个。现在我们证明确实存在这样一个函数。我们通过把一系列函数 $f_x: <^{*-1}\{x\} \to C (x \in B)$ 并起来，去构造这个函数。为了证明这样的函数存在，通过施良基归纳于 $<$ 去证明下列性质 $Q(x)$ 对所有的 $x \in B$ 成立：

$$\exists f_x: <^{*-1}\{x\} \to C.$$
$$\forall y <^* x. f_x(y) = F(y, f_x \restriction <^{-1}\{y\})$$

设 $x \in B$。假设 $\forall z<x. Q(z)$ 成立，则我们宣称

$$h = \bigcup \{f_z \mid z < x\}$$

是一个函数。显然它是一个关系，对每个变量 $z<x$，至少赋予一个值。惟一的困难在于检验

这些函数f_z对变量y的赋值是否一致。但它们必须要一致——否则就违反了上面已经证明的惟一性。令

$$f_x = h \cup \{(x,F(x,h))\}$$

则给出了一个函数$f_x: <^{*-1}\{x\} \to C$,使得

$$\forall y <^* x. f_x(y) = F(y, f_x \upharpoonright <^{-1}\{y\})$$

这就完成了良基归纳法的证明,从而得到$\forall x \in B. Q(x)$成立。

现在我们取$f = \cup_{x \in B} f_x$。由惟一性,可得$f: B \to C$,而且f是满足(*)的惟一函数。 □

良基递归和归纳给出了适合于定义完全函数的一般方法。例如,从递归定理可以直接得出,存在一个惟一的非负整数的完全函数,使得对于所有的$m,n \geq 0$,

$$ack(m,n) = \begin{cases} n+1 & m = 0 \\ ack(m-1,1) & m \neq 0, n = 0 \\ ack(m-1, ack(m,n-1)) & \text{其他} \end{cases}$$

注意到ack函数在(m,n)处的值是由在字典序意义下更小的$(m-1,1)$和$(m,n-1)$处的值定义的。事实上,人们设计的很多递归程序都是使良基集中的某种测度随着计算的进行而依次减小。对于这样的程序,通常用良基递归及归纳法代替最小不动点方法。 178

10.5 一个练习

本章最后给出一个练习来证明表上的两个递归函数是等价的,这个单一问题的解答综合了论证递归定义的许多技巧。我们前面的讨论倾向于集中在算术运算和布尔运算上,这里我们主要探讨整数有限表上的运算。一个*整数表*的典型形式为:

$$[m_1; m_2; \cdots; m_k]$$

它由$\mathbf{N}$中的k个元素组成。空表也是一个表,记为

$$[\,]$$

有两种基本的构造表的运算。一种是常元运算,即把变量为空的元组()定义为空表[ ],另一种运算通常叫做$cons$运算,即把整数m加在表l的前面作为前缀,其结果记为

$$m :: l$$

例如,

$$1 :: [2;3;4] = [1;2;3;4]$$

整数表集合构成了称为$List$(表)的一个离散完全偏序。它是两个离散完全偏序的和:

$$List = in_1\{()\} \cup in_2(\mathbf{N} \times List) = \{()\} + (\mathbf{N} \times List)$$

其对应的内射函数为:

$$in_1() = [\,]$$

$$in_2(m,l) = m :: l$$

按照这种方法,表可以看成是一个和,这反映了整数表上的离散完全偏序与所有的包括空元组()在内的整数元组是同构的。

这个和同时也伴随着一个 *case* 构造

$$\begin{aligned} &case\ l\ of\ [\,].\ e_1 \mid \\ &\qquad x :: l'.\ e_2 \end{aligned}$$

它的作用可以用一个函数的递归定义来说明

179

$$append : List \times List \to (List)_\perp$$

它定义了两个表上的添加运算

$$\begin{aligned} append = \mu\alpha.\ \lambda l, ls \in List \\ Case\ l\ of\ [\,].\ \lfloor ls \rfloor \mid \\ x :: l'.\ (let\ r \Leftarrow \alpha(l', ls).\ \lfloor x :: r \rfloor) \end{aligned}$$

函数 *append* 是完全偏序 $[List \times List \to (List)_\perp]$ 中最小的 α 函数,它满足

$$\begin{aligned} &\alpha([\,], ls) = \lfloor ls \rfloor \\ &\alpha(x :: l', ls) = (let\ r \Leftarrow \alpha(l', ls).\ \lfloor x :: r \rfloor) \end{aligned}$$

对第一个变量中表的大小进行归纳可以证明 *append* 永不为 $\perp$。若两个表之间的关系 $<$ 定义为 $l' < l$ 当且仅当表 l' 严格小于表 l,则我们可以用良基递归法在表上定义一个稍微不同的合并操作@ : $List \times List \to List$。按照良基递归定理 10.19,@ 是惟一的(完全)函数,使得对所有的 $l, ls \in List$,有

$$\begin{aligned} l @ ls = case\ l\ of\ [\,].\ ls \mid \\ x :: l'.\ x :: (l' @ ls) \end{aligned}$$

我们可以用良基归纳证明,对所有的表 l, ls,这两个函数有

$$append(l, ls) = \lfloor l @ ls \rfloor$$

现在我们将问题表述如下。

练习 10.20 设有整数上的函数 $s : \mathbf{N} \times \mathbf{N} \to \mathbf{N}$ 及函数 $r : \mathbf{N} \times \mathbf{N} \to List$。设 f 为 $[List \times \mathbf{N} \to \mathbf{N}_\perp]$ 中满足

$$\begin{aligned} &f([\,], y) = \lfloor y \rfloor \\ &f(x :: xs, y) = f(r(x, y) @ xs, s(x, y)) \end{aligned}$$

的最小函数,设 g 为 $[List \times \mathbf{N} \to \mathbf{N}_\perp]$ 中满足

$$\begin{aligned} &g([\,], y) = \lfloor y \rfloor \\ &g(x :: xs, y) = let\ v \Leftarrow g(r(x, y), s(x, y)).\ g(xs, v) \end{aligned}$$

的最小函数,试证明 $f = g$。

提示:首先对表 l 的大小进行归纳,证明 g 满足

$$g(l@xs,y) = let\ v \Leftarrow g(l,y).\ g(xs,v)$$ 180

从而推出 $f \sqsubseteq g$。接下来用不动点归纳法,证明 f 满足

$$(let\ u \Leftarrow f(l,y).f(xs,u)) \sqsubseteq f(l@xs,y)$$

可以由包含谓词

$$P(F) \Longleftrightarrow_{def} [\ \forall xs,l,y.\ (let\ u \Leftarrow F(l,y).f(xs,u)) \sqsubseteq f(l@xs,y)]$$

推出 $g \sqsubseteq f$。 □

10.6 进一步阅读资料

本章内容的表述受到了文献[80]、[59]和[89]的影响。特别是 Manna 的书[59]提供了丰富的关于不动点和良基归纳法的练习(遗憾的是该书把良基归纳法称为"结构归纳法")。对表的问题我要感谢 Larry Paulson。读者要注意关于"包含"性质和谓词的概念还没有统一的术语。本书中的术语"包含"是从 Gordon Plotkin 的讲义[80]中得到的。其他的术语还有可允许谓词等等。如课域论建立在有向集合而不是 ω 链的基础上,则术语问题变得更为复杂——尽管概念依赖于所使用的术语,但对于 ω 代数的完全偏序,两种方法得到的概念是等价的。其他的参考文献有[13]、[58]和[21](然而,文献[21]错误地认为:完全偏序的积的谓词是包含的,如果它在每一个变量上是包含的)。Enderton 的著作[39]对良基递归有详细的讨论(在[39]的索引中查找对"递归"的引用,注意他的证明采用了一种传递的良基关系,称为"良序")。 181

第11章 高阶类型语言

本章我们要研究高阶类型语言的操作语义和指称语义，这种语言允许显式使用函数空间构造符来构造类型；函数变成“最有用的”值，它能够作为函数的输入或输出。在求值过程中，我们要选择采用传值调用的方式还是传名调用的方式。在传值调用的方式中语言的行为类似于活性语言标准 ML，而在传名调用的方式下，语言的行为类似于惰性语言 Miranda[⊖]，Orwell 或 Haskell。这样就可以开始研究函数式程序设计语言的语义。作为函数式程序设计语言语义的一个应用，我们要研究如何表示活性语言和惰性语言的不动点操作。对应于操作语义以及完全抽象的概念，我们要讨论指称语义的适用性。类型上的主要构造方法是积和函数空间，除此之外，本章还介绍如何将其扩展到和。

11.1 活性语言

在函数式程序设计的上下文中，通常把传值调用方式的求值称为*活性*。为了提高效率，人们用按需调用方式（或*惰性*方式）去实现传名调用；通过共享实现，一个参数至多计算一次。我们选择传值调用（活性）方式还是选择传名调用（惰性）方式去进行求值计算，对语言语法的递归定义方面有一定影响。下面，我们首先研究传值调用方式的求值计算。

和 **REC** 一样，语言中有表示像数字这样基本值的项。这些项可以由数、变量、条件及算术运算构成，其计算结果或者为数或者发散。项的计算结果还可以是序偶或者甚至是函数。（我们马上会看到如何计算产生一个函数。）

为了计算不同类型的项，我们在程序设计语言中引入*类型*。如果项的计算不发散而结果为数，那么我们称该项的类型为 **int**。如果项的计算结果为序偶，那么其类型为 $\tau_1 * \tau_2$ 形式的积。如果项的计算结果为函数，那么其类型为函数 $\tau_1 \to \tau_2$。总之，类型表达式 τ 形为

$$\tau ::= \mathbf{int} \mid \tau_1 * \tau_2 \mid \tau_1 \to \tau_2$$

为了简化语言，我们假设 **Var** 中的变量 $x, y, \cdots$ 具有与之对应的惟一的类型，例如由 $\mathbf{type}(x)$ 给出。（实践中，在定义变量时加入类型 τ，因此，变量 x 形为 $x : \tau$）。项 $t, t_0, t_1, \cdots$ 的语法为 183

$$\begin{aligned} t ::= & x \mid \\ & n \mid t_1 + t_2 \mid t_1 - t_2 \mid t_1 \times t_2 \mid \mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 \mid \\ & (t_1, t_2) \mid \mathbf{fst}(t) \mid \mathbf{snd}(t) \mid \\ & \lambda x.\ t \mid (t_1\ t_2) \mid \\ & \mathbf{let}\ x \Leftarrow t_1\ \mathbf{in}\ t_2 \mid \\ & \mathbf{rec}\ y.\ (\lambda x.\ t) \end{aligned}$$

⊖ Miranda 是 Research Software Ltd 的商标。

该语法描述了：

- 如何用类似于第 9 章中 **REC** 语言的方式书写算术表达式。和 **REC** 语言一样，条件分支是根据算术项而不是布尔项。但是和 **REC** 语言不同，条件分支的计算结果不一定是数字。
- 如何构造序偶(t_1, t_2)，并且用 **fst**(t)和 **snd**(t)分别投影到它的第一个分量和第二个分量。
- 如何用 λ 抽象去定义函数，并且用$(t_1\ t_2)$表示函数 t_1 应用到 t_2。
- 如何在 **let** $x \Leftarrow t_1$ **in** t_2中，计算 t_2 前，先强制计算项 t_1 的值。
- 如何用 **rec** $y.\ (\lambda x.\ t)$ 递归定义函数 y 为 $\lambda x.\ t$，当然项 t 中能够包含 y。注意，在这种活性语言中，任何递归定义必须有函数类型。例如，如果 **rec** $y.\ (\lambda x.\ t) : \tau$，那么对类型 τ_1, τ_2，有 $\tau \equiv \tau_1 -> \tau_2$。选择了这种语法表示，处理风格就能和标准 ML 保持一致。

我们可以写出类似于 **REC** 语言的算术表达式。然而，如果试图添加两个函数或给出的函数参数过多或过少都可能毫无意义。合式项 t 是得到类型 τ 的那些项，记作 $t : \tau$。

184 对于某个类型 τ，将项 t 按以下规则写成 $t : \tau$，我们说项 t 是有类型 τ 的。

类型规则

变量： $x : \tau$ 如果 **type**$(x) = \tau$

操作： n: **int**

$$\frac{t_1 : \textbf{int}\ \ t_2 : \textbf{int}}{t_1\ \textbf{op}\ t_2 : \textbf{int}}, \text{其中 op 是 +, - 或 } \times$$

$$\frac{t_0 : \textbf{int}\ \ t_1 : \tau\ \ t_2 : \tau}{\textbf{if}\ t_0\ \textbf{then}\ t_1\ \textbf{else}\ t_2 : \tau}$$

积：
$$\frac{t_1 : \tau_1\ \ t_2 : \tau_2}{(t_1, t_2) : \tau_1 * \tau_2} \quad \frac{t : \tau_1 * \tau_2}{\textbf{fst}(t) : \tau_1} \quad \frac{t : \tau_1 * \tau_2}{\textbf{snd}(t) : \tau_2}$$

函数：
$$\frac{x : \tau_1\ \ t : \tau_2}{\lambda x.\ t : \tau_1 -> \tau_2} \quad \frac{t_1 : \tau_1 -> \tau_2\ \ t_2 : \tau_1}{(t_1\ t_2) : \tau_2}$$

let：
$$\frac{x : \tau_1\ \ t_1 : \tau_1\ \ t_2 : \tau_2}{\textbf{let}\ x \Leftarrow t_1\ \textbf{in}\ t_2 : \tau_2}$$

rec：
$$\frac{y : \tau\ \ \lambda x.\ t : \tau}{\textbf{rec}\ y.\ (\lambda x.\ t) : \tau}$$

练习 11.1 如果 $t : \tau$ 和 $t : \tau'$蕴涵着 τ 和 τ'是相同的类型，则称项 t 的类型是惟一的。试证明这个性质对所有有类型的项都成立。 □

施结构归纳于项 t，我们可以直接定义项 t 的自由变量集 $FV(t)$：

$$\begin{aligned}
FV(n) &= \emptyset \\
FV(x) &= \{x\} \\
FV(t_1\ \textbf{op}\ t_2) &= FV(t_1) \cup FV(t_2) \\
FV(\textbf{if}\ t_0\ \textbf{then}\ t_1\ \textbf{else}\ t_2) &= FV(t_0) \cup FV(t_1) \cup FV(t_2) \\
FV((t_1, t_2)) &= FV(t_1) \cup FV(t_2)
\end{aligned}$$

$$FV(\mathbf{fst}(t)) = FV(\mathbf{snd}(t)) = FV(t)$$ 185

$$FV(\lambda x.\, t) = FV(t) \setminus \{x\}$$

$$FV((t_1, t_2)) = FV(t_1) \cup FV(t_2)$$

$$FV(\mathbf{let}\ x \Leftarrow t_1\ \mathbf{in}\ t_2) = FV(t_1) \cup (FV(t_2) \setminus \{x\})$$

$$FV(\mathbf{rec}\ y.\ (\lambda x.\, t)) = FV(\lambda x.\, t) \setminus \{y\}$$

其中 **let** 构造的子句可理解为：项 t_2 中变量 x 约束于 **let** 构造中。项 t 是封闭的当且仅当 $FV(t) = \emptyset$，即当项 t 中没有自由变量时，则称项 t 是封闭的。

有时操作语义要求用封闭项 s 去代入项 t 中的自由变量 x。我们用 $t[s/x]$ 表示这样的代入。读者对代入应没有什么困难。一般地，我们把用 $s_1, \cdots, s_k$ 同时代入 t 中的 $x_1, \cdots, x_k$ 记作 $t[s_1/x_1, \cdots, s_k/x_k]$，假设 $x_1, \cdots, x_k$ 是不同的自由变量。

11.2 活性操作语义

至今我们仅仅非形式地介绍了程序设计语言的有关行为。现在我们考察传值调用（或活性）的求值方法。正如 **REC** 语言中那样，为了对应用到某些变量的函数求值，我们首先要对这些变量求值以得到值，然后函数作用到这些值上。但在更一般的语言中，求值计算的值是什么呢？当然我们可以期望数字是值，但把函数应用到函数变量的情况下，我们什么时候停止计算这些函数变量，并认为求值过程已经得到函数变量的值呢？有一种选择是把 λ 抽象作为函数的值。我们可以用更一般的方法表示每一类型的项的值。习惯上，这些项称为标准型。施结构归纳于 τ，我们可以定义类型为 τ 的项 t 的标准型，记作 $t \in C^e_\tau$：

基类型：数是标准型，即 $n \in C^e_{\mathbf{int}}$。

积类型：标准型的序偶是标准型，即

$$(c_1, c_2) \in C^e_{\tau_1 * \tau_2}, \quad \text{如果 } c_1 \in C^e_{\tau_1}\ \&\ c_2 \in C^e_{\tau_2}$$

函数类型：封闭的抽象是标准型，即

$$\lambda x.\, t \in C^e_{\tau_1 \to \tau_2}, \quad \text{如果 } \lambda x.\, t : \tau_1 \to \tau_2 \text{ 且 } \lambda x.\, t \text{ 是封闭的}$$

注意，标准型是一类特殊的封闭项。 186

现在我们给出形为 $t \to^e c$ 的求值关系的规则，其中 t 是有类型的封闭项，c 是标准型，$t \to^e c$ 表示把 t 求值到 c。

求值规则

标准型：$c \to^e c$，其中 $c \in C^e_\tau$

操作：$\dfrac{t_1 \to^e n_1 \quad t_2 \to^e n_2}{(t_1\ \mathbf{op}\ t_2) \to^e n_1\ op\ n_2}$，其中 **op** 是 +，- 或 ×

$$\frac{t_0 \to^e 0 \quad t_1 \to^e c_1}{\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 \to^e c_1} \qquad \frac{t_0 \to^e n \quad t_2 \to^e c_2}{\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 \to^e c_2}\ n \neq 0$$

积：$\dfrac{t_1 \to^e c_1 \quad t_2 \to^e c_2}{(t_1, t_2) \to^e (c_1, c_2)}$

$$\frac{t \to^e (c_1, c_2)}{\mathbf{fst}(t) \to^e c_1} \quad \frac{t \to^e (c_1, c_2)}{\mathbf{snd}(t) \to^e c_2}$$

函数：
$$\frac{t_1 \to^e \lambda x.\, t_1' \quad t_2 \to^e c_2 \quad t_1'[c_2/x] \to^e c}{(t_1\ t_2) \to^e c}$$

let：
$$\frac{t_1 \to^e c_1 \quad t_2[c_1/x] \to^e c_2}{\mathbf{let}\ x \Leftarrow t_1\ \mathbf{in}\ t_2 \to^e c_2}$$

rec：
$$\mathbf{rec}\ y.\ (\lambda x.\ t) \to^e \lambda x.\ (t[\mathbf{rec}\ y.\ (\lambda x.\ t)/y])$$

正如期望的，标准型规则表示标准型求值计算到自身。算术操作和条件操作的规则实际上和第 9 章的 **REC** 语言一样。在活性求值的机制下，对序偶的求值就是对它的分量的求值。投影函数 **fst** 和 **snd** 只有在它的变量完全计算完后才能计算。关键的规则是函数应用的求值规则：一旦函数部分和变量部分都计算完成后，才能处理应用的求值计算。注意 **let** $x \Leftarrow t_1$ **in** t_2 的计算规则要求先计算 t_1。递归定义规则对递归 **rec** $y.\ (\lambda x.\ t)$ "展开" 一次直接得到抽象 $\lambda x.\ (t[\mathbf{rec}\ y.\ (\lambda x.\ t)/y])$，因此，它是一个标准型。注意：为了类型匹配，对类型 τ_1 和 τ_2，有 $y: \tau_1 -> \tau_2$，而 x 的类型为 τ_1。这确保 y 和 x 是不同的，因此我们可以用 $(\lambda x.\ t)[\mathbf{rec}(\lambda x.\ t)/y]$ 来代替 $\lambda x.\ (t[\mathbf{rec}\ y.\ (\lambda x.\ t)/y])$。

187

可以直接证明求值关系是确定的和类型一致的。

命题 11.2 如果 $t \to^e c$ 且 $t \to^e c'$，则 $c \equiv c'$（即求值计算是确定的）。如果 $t \to^e c$ 且 $t: \tau$，则 $c: \tau$（即求值计算是类型一致的）。

证明 用简单的规则归纳法证明这两个性质。 □

练习 11.3 设 $\mathrm{fact} \equiv \mathbf{rec}\, f.\ (\lambda x.\ \mathbf{if}\ x\ \mathbf{then}\ 1\ \mathbf{else}\ x \times f(x-1))$。试用操作语义推导求值计算 fact 2。 □

11.3 活性指称语义

指称语义描述如何把类型为 $\tau_1 -> \tau_2$ 的项作为函数，因此能加深我们对函数式语言编写的程序的非形式的理解。通过在完全偏序和连续函数的框架中解释这种语言，该语言将适用于第 10 章的证明技术。

我们首先考虑如何解释类型表达式。类型为 τ 的封闭项 t 或者求值到类型为 τ 的标准型或者发散。因此，一个合理的想法是用项 t 指称 $(V_\tau^e)_\perp$ 的一个元素，其中 V_τ^e 表示类型为 τ 的值的完全偏序，它包含了标准型的指称。用这种思路，施结构归纳于类型表达式，我们定义：

$$V_{\mathbf{int}}^e = \mathbf{N}$$

$$V_{\tau_1 * \tau_2}^e = V_{\tau_1}^e \times V_{\tau_2}^e$$

$$V_{\tau_1 -> \tau_2}^e = [V_{\tau_1}^e \to (V_{\tau_2}^e)_\perp]$$

最后的子句表示函数值可以作为输入的值，也可以作为输出的值或者发散。

一般来说，项含有自由变量。指称语义需要环境的概念，以便对自由变量赋值。活性语言的环境是一个典型的函数

$$\rho : \mathbf{Var} \to \bigcup \{V_\tau^e \mid \tau \text{ 是类型}\}$$

其中,对任一 $x \in \mathbf{Var}$ 和类型 τ,有

$$x : \tau \Rightarrow \rho(x) \in V_\tau^e$$

188

把 $\mathbf{Env}^e$ 记作所有这样环境的完全偏序。

现在,我们给出活性语言的指称语义;类型为 τ 的项 $t : \tau$ 在环境 ρ 下指称元素 $[\![t]\!]^e\rho \in (V_\tau^e)_\perp$。

指称语义

用下面的结构归纳法给出有类型的项 t 的指称:

$$\begin{aligned}
[\![x]\!]^e &= \lambda\rho.\lfloor\rho(x)\rfloor \\
[\![n]\!]^e &= \lambda\rho.\lfloor n\rfloor \\
[\![t_1 \ \mathbf{op}\ t_2]\!]^e &= \lambda\rho.([\![t_1]\!]^e\rho \ op_\perp [\![t_2]\!]^e\rho), \text{其中 } \mathbf{op} \text{ 是 } +, -, \times \\
[\![\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2]\!]^e &= \lambda\rho.\ Cond([\![t_0]\!]^e\rho, [\![t_1]\!]^e\rho, [\![t_2]\!]^e\rho) \\
[\![(t_1, t_2)]\!]^e &= \lambda\rho.\ let\ v_1 \Leftarrow [\![t_1]\!]^e\rho, v_2 \Leftarrow [\![t_2]\!]^e\rho.\lfloor(v_1, v_2)\rfloor \\
[\![\mathbf{fst}(t)]\!]^e &= \lambda\rho.\ let\ v \Leftarrow [\![t]\!]^e\rho.\lfloor\pi_1(v)\rfloor \\
[\![\mathbf{snd}(t)]\!]^e &= \lambda\rho.\ let\ v \Leftarrow [\![t]\!]^e\rho.\lfloor\pi_2(v)\rfloor \\
[\![\lambda x.\ t]\!]^e &= \lambda\rho.\lfloor\lambda v \in V_{\tau_1}^e.\ [\![t]\!]^e\rho[v/x]\rfloor \\
&\quad \text{其中 } \lambda x.\ t : \tau_1 \to \tau_2 \\
[\![(t_1\ t_2)]\!]^e &= \lambda\rho.\ let\ \varphi \Leftarrow [\![t_1]\!]^e\rho, v \Leftarrow [\![t_2]\!]^e\rho.\ \varphi(v) \\
[\![\mathbf{let}\ x \Leftarrow t_1\ \mathbf{in}\ t_2]\!]^e &= \lambda\rho.\ let\ v \Leftarrow [\![t_1]\!]^e\rho.\ [\![t_2]\!]^e\rho[v/x] \\
[\![\mathbf{rec}\ y.\ (\lambda x.\ t)]\!]^e &= \lambda\rho.\lfloor\mu\varphi.\ (\lambda v.\ [\![t]\!]^e\rho[v/x, \varphi/y])\rfloor
\end{aligned}$$

在给出条件的指称语义的子句中,我们使用了 9.3 节的条件函数 $Cond$ 的推广形式。对含底的完全偏序 D,函数

$$Cond : \mathbf{N}_\perp \times D \times D \to D$$

满足

$$Cond(z_0, z_1, z_2) = \begin{cases} z_1 & z_0 = \lfloor 0\rfloor \\ z_2 & z_0 = \lfloor n\rfloor (\text{对于某些 } n \in \mathbf{N}, n \neq 0) \\ \perp & \text{其他} \end{cases}$$

其中 $z_0 \in \mathbf{N}_\perp, z_1, z_2 \in D$。像 9.3 节中一样,可以证明它是连续函数。注意,上述语义可以用 8.4 节中确保有不动点的元语言去表示。

练习 11.4 按照指称语义,如何用其他构造(但不能用 **let**)去定义 **let** $x \Leftarrow t_1$ **in** t_2? □ 189

引理 11.5 设 t 是有类型的项,设环境 ρ, ρ' 对 t 的自由变量取值相同,则 $[\![t]\!]^e\rho = [\![t]\!]^e\rho'$。

证明 用简单的结构归纳法,证明留给读者。 □

引理 11.6(代入引理) 设 s 是封闭项 $s:\tau$,使得 $[s]^e\rho = \lfloor v \rfloor$。设 x 是变量 $x:\tau$。假设 $t:\tau'$,则 $t[s/x]:\tau'$ 且 $[t[s/x]]^e\rho = [t]^e\rho[v/x]$。

证明 用结构归纳法证明。 □

练习 11.7 在代入引理的证明中,当 t 是抽象或 **let** 构造时,试写出归纳步骤。 □

正如我们期望的,类型为 τ 的一般项的指称在 $(V_\tau^e)_\perp$ 中,而标准型的指称与值有关:

引理 11.8 (i) 如果 $t:\tau$,则对任何 ρ,有 $[t]^e\rho \in (V_\tau^e)_\perp$。

(ii) 如果 $c \in C_\tau^e$,则对任何 ρ,有 $[c]^e\rho \neq \perp$,其中 $\perp$ 是 $(V_\tau^e)_\perp$ 的底元素。

证明 (i) 简单地施结构归纳于 t 去证明。(ii) 施结构归纳于标准型 c 去证明。 □

练习 11.9 证明引理 11.8 的 (ii)。 □

11.4 活性语义的一致性

操作语义和指称语义一致吗?我们将看到它们确实是一致的,虽然一开始不这么认为。以前介绍过操作语义和指称语义紧密相关,可能会使我们不正确地认为,对封闭项 t 和标准型 c,有

$$t \to^e c \Longleftrightarrow [t]^e\rho = [c]^e\rho$$

"$\Leftarrow$"对包含函数空间的任一类型不成立,主要是因为存在有许多具有相同指称的标准型,而项的求值至多能产生其中之一(见后面的练习)。然而,不论 t 为何种类型,这个等价的另一方向"$\Rightarrow$"却成立:

190

$$t \to^e c \Rightarrow [t]^e\rho = [c]^e\rho \tag{1}$$

此外,操作语义和指称语义这两种语义风格,对封闭项的求值是否收敛是一致的。

考察有类型的封闭项 t。根据操作语义的求值规则,t 或者发散或者产生一个标准型。t 的操作收敛定义为

$$t\downarrow^e \text{ 当且仅当 } \exists c.\, t \to^e c$$

从指称语义的观点看,t 的计算结果是 $(V_\tau^e)_\perp$ 中的一个元素 $[t]^e\rho$,其中 τ 是 t 的类型并且 ρ 是任一环境,因为 t 是封闭的,所以,如果 t 发散,则 t 的指称为 $\perp$,如果 t 收敛,则 t 的指称为某个 $\lfloor v \rfloor$。t 的指称收敛定义为

$$t\Downarrow^e \text{ 当且仅当 } \exists v \in V_\tau^e.\, [t]^e\rho = \lfloor v \rfloor$$

我们希望操作收敛和指称收敛这两个概念是一致的,即

$$t\downarrow^e \Longleftrightarrow t\Downarrow^e \tag{2}$$

的确,使用引理 11.8(ii),从(1)可得"$\Rightarrow$"成立,这就是说如果标准型操作收敛则其也指称收敛。

由(1)和(2)得,如果 t: **int**,则

$$t \to^e n \Longleftrightarrow [t]^e\rho = \lfloor n \rfloor \tag{3}$$

为了说明(1)和(2)推出最后的断言(3),注意到(3)的"$\Rightarrow$"恰好是(1)的特例,而具有相同指

称的类型为 int 的两个标准型必然是同一个数。这个事实可推出(3)的相反方向"⇐"。(1)和(2)表达了指称语义对应于操作语义的适用性(*adequancy*)。这样,使我们能从指称语义中论证操作语义的求值关系。

练习 11.10 试证明:对一般的类型,(1)的相反方向,即

$$[t]^e\rho = [c]^e\rho \Rightarrow t \rightarrow^e c$$

不成立。(提示:取 $t \equiv \lambda x.\, x, c \equiv \lambda x.\, x+0$,其中 x: **int**。) □

现在我们证明上述断言(1),即求值关系对指称语义是适用的。

引理 11.11 如果 $t \rightarrow^e c$,则对任意环境 ρ,有 $[t]^e\rho = [c]^e\rho$。 191

证明 施规则归纳于求值规则进行证明。大多数规则显然保持上述性质。下面,我们仅考察感兴趣的几条规则。

考察规则

$$\frac{t \rightarrow^e (c_1, c_2)}{\mathbf{fst}(t) \rightarrow^e c_1}$$

假设对任意环境 ρ,有 $[t]^e\rho = [(c_1, c_2)]^e\rho$,则根据引理 11.8,有 $(c_1, c_2) \Downarrow^e$,故

$$\begin{aligned}
[t]^e\rho &= [(c_1, c_2)]^e\rho \\
&= let\ v_1 \Leftarrow [c_1]^e\rho, v_2 \Leftarrow [c_2]^e\rho.\ \lfloor (v_1, v_2) \rfloor \\
&= \lfloor (v_1, v_2) \rfloor, \text{其中} [c_1]^e\rho = \lfloor v_1 \rfloor \text{和} [c_2]^e\rho = \lfloor v_2 \rfloor
\end{aligned}$$

所以

$$\begin{aligned}
[\mathbf{fst}(t)]^e\rho &= let\ v \Leftarrow [t]^e\rho \cdot \lfloor \pi_1(v) \rfloor \\
&= \lfloor v_1 \rfloor \\
&= [c_1]^e\rho
\end{aligned}$$

考察规则

$$\frac{t_1 \rightarrow^e \lambda x.\, t_1' \quad t_2 \rightarrow^e c_2 \quad t_1'[c_2/x] \rightarrow^e c}{(t_1\ t_2) \rightarrow^e c}$$

假设 $[t_1]^e\rho = [\lambda x.\, t_1']^e\rho$, $[t_2]^e\rho = [c_2]^e\rho$ 和 $[t_1'[c_2/x]]^e\rho = [c]^e\rho$。所以

$$\begin{aligned}
[t_1\ t_2]^e\rho &= let\ \varphi \Leftarrow [t_1]^e\rho, v \Leftarrow [t_2]^e\rho.\ \varphi(v) \\
&= let\ \varphi \Leftarrow [\lambda x.\, t_1']^e\rho, v \Leftarrow [c_2]^e\rho.\ \varphi(v) \\
&= let\ \varphi \Leftarrow \lfloor \lambda v.\, [t_1']^e\rho[v/x] \rfloor, v \Leftarrow [c_2]^e\rho.\ \varphi(v) \\
&= [t_1']^e\rho[v/x], \text{其中} [c_2]^e\rho = \lfloor v \rfloor \quad (\text{由引理 } 11.8) \\
&= [t_1'[c_2/x]]^e\rho \quad (\text{代入引理 } 11.6) \\
&= [c]^e\rho
\end{aligned}$$

考察规则

$$\frac{}{\mathbf{rec}\ y.\ (\lambda x.\ t) \rightarrow^{e} \lambda x.\ (t[\mathbf{rec}\ y.\ (\lambda x.\ t)/y])}$$

根据定义，$[\![\mathbf{rec}\ y.\ (\lambda x.\ t)]\!]^{e}\rho = \lfloor \varphi \rfloor$，其中 φ 是方程

192

$$\varphi = \lambda v.\ [\![t]\!]^{e}\rho[v/x, \varphi/y]$$

的最小解。由代入引理 11.6，得

$$\begin{aligned}[\![\lambda x.\ (t[\mathbf{rec}\ y.\ (\lambda x.\ t)/y])]\!]^{e}\rho &= [\![\lambda x.\ t]\!]^{e}\rho[\varphi/y]\ (y\text{ 和 }x\text{ 是不同的变量})\\ &= \lfloor \lambda v.\ [\![t]\!]^{e}\rho[v/x, \varphi/y] \rfloor\\ &= \lfloor \varphi \rfloor\\ &= [\![\mathbf{rec}\ y.\ (\lambda x.\ t)]\!]^{e}\rho\end{aligned}$$

□

由引理 11.8 和引理 11.11，我们得到对任何类型的封闭项 t，

$$t\downarrow^{e} \quad \text{蕴涵} \quad t\Downarrow^{e}$$

另一方向（“$\Leftarrow$”）用逻辑关系的技术去证明。我们要证明，对任何类型的封闭项 t，

$$t\Downarrow^{e} \quad \text{蕴涵} \quad t\downarrow^{e} \qquad (*)$$

显然，可以施结构归纳于 t 去证明。因此，用式（*）作为我们证明中的归纳假设。考察 t 是应用 $(t_1\ t_2)$，我们归纳假设

$$(t_1\Downarrow^{e} \quad \text{蕴涵} \quad t_1\downarrow^{e})\ \text{和}\ (t_2\Downarrow^{e} \quad \text{蕴涵} \quad t_2\downarrow^{e})$$

要证明（*）假设 $t\Downarrow^{e}$，因为

$$[\![t]\!]^{e}\rho = let\ \varphi \Leftarrow [\![t_1]\!]^{e}\rho, v \Leftarrow [\![t_2]\!]^{e}\rho.\ \varphi(v)$$

这就确保了 $t_1\Downarrow^{e}$ 和 $t_2\Downarrow^{e}$。根据归纳法，对适当的标准型，我们有

$$t_1 \rightarrow^{e} \lambda x.\ t_1' \ \text{和}\ t_2 \rightarrow^{e} c_2$$

于是，$[\![t]\!]^{e}\rho = \varphi(v)$，其中 $\varphi = \lambda u.\ [\![t_1']\!]^{e}\rho[u/x]$ 和 $\lfloor v \rfloor = [\![c_2]\!]^{e}\rho$。因此，由代入引理得

$$\begin{aligned}[\![t]\!]^{e}\rho &= [\![t_1']\!]^{e}\rho[v/x]\\ &= [\![t_1'[c_2/x]]\!]^{e}\rho\end{aligned}$$

因为 $t\Downarrow^{e}$，所以 $t_1'[c_2/x]\Downarrow^{e}$。我们希望推出 $t_1'[c_2/x]\downarrow^{e}$，因而有 $t_1'[c_2/x]\rightarrow^{e}c$。进一步，从操作语义的角度，得到 $t\rightarrow^{e}c$。但我们还不能证明其成立，因为 $t_1'[c_2/x]$ 和 t 的结构关系不明

193

显，使得我们不能合理地应用结构归纳假设。

通常，解决的办法是加强归纳假设。我们用证明项的指称行为和操作行为之间更强的、更详细的“逼近”关系成立，去代替试图证明项的指称收敛蕴涵它的操作收敛。对类型 τ，这表示为完全偏序 $(V_{\tau}^{e})_{\perp}$ 的元素和类型 τ 的封闭项之间的关系 $\lesssim_{\tau}$。用论证高阶类型时经常使用的方法，施结构归纳于项的类型去定义这种关系；这种技术称为逻辑关系。（当然，与使用结构归纳法验证（*）相比，我们要更仔细地考虑自由变量。）

我们要在类型 τ 上定义关系 $\lesssim^{\circ}_{\tau} \subseteq V^{e}_{\tau} \times C^{e}_{\tau}$。将这些基本关系扩展到 $(V^{e}_{\tau})_{\perp}$ 的元素 d 和封闭项 t 之间的关系，定义

$$d \lesssim_{\tau} t \text{ 当且仅当 } \forall v \in V^{e}_{\tau}.\, d = \lfloor v \rfloor \Rightarrow \exists c.\, t \rightarrow^{e} c \;\&\; v \lesssim^{\circ}_{\tau} c$$

施结构归纳于类型 τ 定义基本关系 $\lesssim^{\circ}_{\tau}$ 如下：

基类型：对所有的数 n，有 $n \lesssim^{\circ}_{\mathbf{int}} n$。

积类型：$(v_1, v_2) \lesssim^{\circ}_{\tau_1 * \tau_2} (c_1, c_2)$ 当且仅当 $v_1 \lesssim^{\circ}_{\tau_1} c_1 \;\&\; v_2 \lesssim^{\circ}_{\tau_2} c_2$。

函数类型：$\varphi \lesssim^{\circ}_{\tau_1 \to \tau_2} \lambda x.\, t$ 当且仅当 $\forall v \in V^{e}_{\tau_1}, c \in C^{e}_{\tau_1}.\, v \lesssim^{\circ}_{\tau_1} c \Rightarrow \varphi(v) \lesssim_{\tau_2} t[c/x]$。

最后的子句表达了关键的性质，即函数的操作语义和指称语义这两种表示之间有关系当且仅当它们有相关的变量和相关的结果。作为逻辑关系的例子，这个性质对于类型 τ，构成了 $\lesssim_{\tau}$ 族。

重要的是以后的证明中用到关系 $\lesssim_{\tau}$ 的一些基本性质；特别地，这些性质是包含的。

引理 11.12 设 $t : \tau$，于是有

(i) $\perp_{(V^{e}_{\tau})_{\perp}} \lesssim_{\tau} t$。

(ii) 如果 $d \sqsubseteq d'$ 且 $d' \lesssim_{\tau} t$，则 $d \lesssim_{\tau} t$。

(iii) 如果 $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$ 是 $(V^{e}_{\tau})_{\perp}$ 中的 ω 链，使得对所有的 $n \in \omega$，有 $d_n \lesssim_{\tau} t$，则 $\bigsqcup_{n \in \omega} d_n \lesssim_{\tau} t$。

证明 根据定义，直接可得性质(i)。施结构归纳于类型去证明性质(ii)和性质(iii)对所有的项成立。显然，对基本类型 **int**，性质(ii)和性质(iii)都成立。在函数类型的情况下，我们证明归纳步骤去说明归纳证明。假设 $d_0 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$ 是 $(V^{e}_{\tau_1 \to \tau_2})_{\perp}$ 中的 ω 链，使得对所有的 $n \in \omega$，有 $d_n \lesssim_{\tau_1 \to \tau_2} t$。或者对所有的 $n \in \omega$，有 $d_n = \perp$，或者有 $t \rightarrow^{e} \lambda x.\, t'$ 且有某个 n，当 $m \geq n$ 时，有 $d_m = \lfloor \varphi_m \rfloor$ 以及 $\varphi_m \lesssim^{\circ}_{\tau_1 \to \tau_2} \lambda x.\, t'$。对前一种情况，$\bigsqcup_n d_n = \perp \lesssim_{\tau_1 \to \tau_2} t$；对后一种情况，假设 $v \lesssim^{\circ}_{\tau_1} c$，我们对 $m \geq n$，得到 $\varphi_m(v) \lesssim_{\tau_2} t'[c/x]$。所以，可归纳得到 $\bigsqcup_m (\varphi_m(v)) \lesssim_{\tau_2} t'[c/x]$。因此，每当 $v \lesssim^{\circ}_{\tau_1} c$ 时，有 $\left(\bigsqcup_m \varphi_m\right)(v) \lesssim \tau_2 t'[c/x]$。换言之，$\left(\bigsqcup_m \varphi_m\right) \lesssim^{\circ}_{\tau_1 \to \tau_2} \lambda x.\, t'$，所以，需要证明的性质 $\bigsqcup_m d_m = \bigsqcup_m \varphi_m \lesssim_{\tau_1 \to \tau_2} t$ 成立。 □ 194

练习 11.13 试证明引理 11.12 中(ii)和(iii)的余下部分的归纳步骤。 □

读者也许会从上面的论述和下面的应用情况的证明比较中得到启发。

引理 11.14 设 t 是有类型的封闭项，则有

$$t \Downarrow^{e} \quad \text{蕴涵} \quad t \downarrow^{e}$$

证明 施结构归纳于项进行证明：对所有的含有自由变量 $x_1 : \tau_1, \cdots, x_k : \tau_k$ 的项 $t : \tau$，如果 $\lfloor v_1 \rfloor \lesssim_{\tau_1} s_1, \cdots, \lfloor v_k \rfloor \lesssim_{\tau_k} s_k$，则

$$[\![t]\!]^{e} \rho[v_1/x_1, \cdots, v_k/x_k] \lesssim_{\tau} t[s_1/x_1, \cdots, s_k/x_k]$$

取 t 是封闭的，根据 $\lesssim \tau$ 的定义，如果 $t \Downarrow^{e}$，则对某个 v，有 $[\![t]\!]^{e} \rho = \lfloor v \rfloor$，所以，对某个标准型 c，

有 $t \to^e c$,即 $t \downarrow^e$。

首先,根据引理 11.5,要建立关于项 t 的归纳假设,仅需考虑 t 中的自由变量。

$t \equiv x$,x 是类型为 τ 的变量:假设 $\lfloor v \rfloor \lesssim_\tau s$,则由语义知,$[x]^e\rho\,[v/x] = \lfloor v \rfloor \lesssim_\tau s \equiv x[s/x]$。

$t \equiv n$,n 为一个数:由定义 $n \lesssim^\circ_{\text{int}} n$ 知,归纳假设成立。

$t \equiv t_1$ **op** t_2:设 $x_1 : \tau_1, \cdots, x_k : \tau_k$ 是 t 的所有自由变量,设 $\lfloor v_1 \rfloor \lesssim_{\tau_1} s_1, \cdots, \lfloor v_k \rfloor \lesssim_{\tau_k} s_k$,又设 $[t_1 \textbf{ op } t_2]^e\rho\,[v_1/x_1, \cdots, v_k/x_k] = \lfloor n \rfloor$。于是,由指称语义得,对整数 n_1, n_2 以及 $n = n_1\ op\ n_2$,

$$[t_1]^e\rho[v_1/x_1, \cdots, v_k/x_k] = \lfloor n_1 \rfloor$$

195

$$[t_2]^e\rho[v_1/x_1, \cdots, v_k/x_k] = \lfloor n_2 \rfloor$$

由归纳得

$$[t_1]^e\rho[v_1/x_1, \cdots, v_k/x_k] \lesssim_{\text{int}} t_1[s_1/x_1, \cdots, s_k/x_k]$$
$$[t_2]^e\rho[v_1/x_1, \cdots, v_k/x_k] \lesssim_{\text{int}} t_2[s_1/x_1, \cdots, s_k/x_k]$$

根据 $\lesssim_{\text{int}}$ 的定义,我们有

$$t_1[s_1/x_1, \cdots, s_k/x_k] \to^e n_1$$
$$t_2[s_1/x_1, \cdots, s_k/x_k] \to^e n_2$$

所以,由操作语义知

$$(t_1 \textbf{ op } t_2)[s_1/x_1, \cdots, s_k/x_k] \to^e n$$

所以

$$[t_1 \textbf{ op } t_2]^e\rho[v_1/x_1, \cdots, v_k/x_k] \lesssim_{\text{int}} (t_1 \textbf{ op } t_2)[s_1/x_1, \cdots, s_k/x_k]$$

$t \equiv$ **if** t_0 **then** t_1 **else** t_2:这种情况与上面的 $t \equiv t_1$ **op** t_2 类似。

$t \equiv (t_1, t_2)$:设 $t_1 : \sigma_1, t_2 : \sigma_2$,又设 $x_1 : \tau_1, \cdots, x_k : \tau_k$ 是 t 的所有自由变量,$\lfloor v_1 \rfloor \lesssim_{\tau_1} s_1, \cdots, \lfloor v_k \rfloor \lesssim_{\tau_k} s_k$,假设 $[(t_1, t_2)]^e\rho\,[v_1/x_1, \cdots, v_k/x_k] = \lfloor u \rfloor$,于是,由指称语义知,存在 u_1 和 u_2,使得 $u = (u_1, u_2)$ 并且

$$[t_1]^e\rho[v_1/x_1, \cdots, v_k/x_k] = \lfloor u_1 \rfloor$$
$$[t_2]^e\rho[v_1/x_1, \cdots, v_k/x_k] = \lfloor u_2 \rfloor$$

由归纳得

$$\lfloor u_1 \rfloor \lesssim_{\sigma_1} t_1[s_1/x_1, \cdots, s_k/x_k]$$
$$\lfloor u_2 \rfloor \lesssim_{\sigma_2} t_2[s_1/x_1, \cdots, s_k/x_k]$$

所以,存在标准型 c_1, c_2,使得

$$u_1 \lesssim^\circ_{\sigma_1} c_1 \ \&\ t_1[s_1/x_1, \cdots, s_k/x_k] \to^e c_1$$
$$u_2 \lesssim^\circ_{\sigma_2} c_2 \ \&\ t_2[s_1/x_1, \cdots, s_k/x_k] \to^e c_2$$

根据操作语义,得 $(u_1, u_2) \lesssim^\circ_{\sigma_1 * \sigma_2} (c_1, c_2)$ 和 $(t_1, t_2)[s_1/x_1, \cdots, s_k/x_k] \to^e (c_1, c_2)$。所以,

$[\![(t_1,t_2)]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k] \lesssim_{\sigma_1*\sigma_2} (t_1,t_2)[s_1/x_1,\cdots,s_k/x_k]$。

$t \equiv \mathbf{fst}(s)$：设 $\mathbf{fst}(s)$ 是有类型的，因此 s 必然有类型 $\sigma_1*\sigma_2$，又设 $x_1:\tau_1,\cdots,x_k:\tau_k$ 是 t 的所有自由变量，$\lfloor v_1\rfloor \lesssim_{\tau_1} s_1,\cdots,\lfloor v_k\rfloor \lesssim_{\tau_k} s_k$。假设 $[\![\mathbf{fst}(s)]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k]=\lfloor u\rfloor$，于是，由指称语义，我们有 $u=u_1$，其中对某个 $u_1\in V^e_{\sigma_1}$，$u_2\in V^e_{\sigma_2}$，有 $[\![s]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k]=\lfloor(u_1,u_2)\rfloor$，由归纳，我们有 196

$$[\![s]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k] \lesssim_{\sigma_1*\sigma_2} s[s_1/x_1,\cdots,s_k/x_k]$$

所以，存在标准型 (c_1,c_2)，使得

$$(u_1,u_2) \lesssim^{\circ}_{\sigma_1*\sigma_2} (c_1,c_2)\ \&\ s[s_1/x_1,\cdots,s_k/x_k] \rightarrow^e (c_1,c_2)$$

这可推出 $u_1 \lesssim^{\circ}_{\sigma_1} c_1$，且 $\mathbf{fst}(s[s_1/x_1,\cdots,s_k/x_k]) \rightarrow^e c_1$，因此，有

$$[\![\mathbf{fst}(s)]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k] \lesssim_{\sigma_1} \mathbf{fst}(s[s_1/x_1,\cdots,s_k/x_k])$$

$t \equiv \mathbf{snd}(s)$：和 $\mathbf{fst}(s)$ 的证明类似。

$t\equiv\lambda x.\,t_2$：设 $x:\sigma_1$，$t_2:\sigma_2$，又设 $x_1:\tau_1,\cdots,x_k:\tau_k$ 是 t 的所有自由变量，$\lfloor v_1\rfloor\lesssim_{\tau_1}s_1,\cdots,\lfloor v_k\rfloor\lesssim_{\tau_k}s_k$，假设 $[\![\lambda x.\,t_2]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k]=\lfloor\varphi\rfloor$，于是 $\lambda v\in V^e_{\sigma_1}.\,[\![t_2]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k,v/x]=\varphi$。我们希望 $\varphi\lesssim^{\circ}_{\sigma_1\to\sigma_2}\lambda x.\,t_2[s_1/x_1,\cdots,s_k/x_k]$。然而，假设 $v\lesssim^{\circ}_{\sigma_1}c$，我们有 $\lfloor v\rfloor\lesssim_{\sigma_1}c$。由归纳，得到

$$\varphi(v) = [\![t_2]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k,v/x] \lesssim_{\sigma_2} t_2[s_1/x_1,\cdots,s_k/x_k,c/x]$$

$t\equiv(t_1\ t_2)$：设 $t_1:\sigma_2\to\sigma$，$t_2:\sigma_2$，又设 t 有自由变量 $x_1:\tau_1,\cdots,x_k:\tau_k$，$\lfloor v_1\rfloor\lesssim_{\tau_1}s_1,\cdots,\lfloor v_k\rfloor\lesssim_{\tau_k}s_k$，根据指称语义，得

$$\begin{aligned}&[\![t_1\ t_2]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k] = \\ &let\ \varphi\Leftarrow[\![t_1]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k],v\Leftarrow[\![t_2]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k].\,\varphi(v)\end{aligned}$$

假设对 $u\in V^e_\sigma$，$[\![t_1\ t_2]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k]=\lfloor u\rfloor$。于是存在 φ，v，使得 $\varphi(v)=\lfloor u\rfloor$ 且

$$\begin{aligned}[\![t_1]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k] &= \lfloor\varphi\rfloor \\ [\![t_2]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k] &= \lfloor v\rfloor\end{aligned}$$

由归纳得

$$\begin{aligned}&[\![t_1]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k] \lesssim_{\sigma_2\to\sigma} t_1[s_1/x_1,\cdots,s_k/x_k] \\ &[\![t_2]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k] \lesssim_{\sigma_2} t_2[s_1/x_1,\cdots,s_k/x_k]\end{aligned}$$ 197

回想起以项 $\lesssim^{\circ}_{\sigma_2\to\sigma}$ 和 $\lesssim^{\circ}_{\sigma_2}$ 去定义 $\lesssim_{\sigma_2\to\sigma_1}$ 和 $\lesssim_{\sigma_2}$，我们知道，存在标准型，使得

$$\begin{aligned}&t_1[s_1/x_1,\cdots,s_k/x_k]\rightarrow^e\lambda x.\,t_1'\ \&\ \varphi\lesssim^{\circ}_{\sigma_2\to\sigma}\lambda x.\,t_1' \\ &t_2[s_1/x_1,\cdots,s_k/x_k]\rightarrow^e c_2\ \&\ v\lesssim^{\circ}_{\sigma_2} c_2\end{aligned}$$

现在由$\lesssim^{\circ}_{\sigma_2 \to \sigma}$的定义,得到

$$\varphi(v) \lesssim_{\sigma} t_1'[c_2/x]$$

因为$\varphi(v) = \lfloor u \rfloor$,所以存在一个$c \in C^e_{\sigma}$使得

$$t_1'[c_2/x] \to^e c \ \& \ u \lesssim^{\circ}_{\sigma} c$$

这样,我们得到了求值规则(函数)的前提,因而可推出$(t_1\ t_2)[s_1/x_1,\cdots,s_k/x_k] \to^e c$。现在,因为$u \lesssim^{\circ}_{\sigma} c$,所以

$$[\![t_1\ t_2]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k] \lesssim_{\sigma} (t_1\ t_2)[s_1/x_1,\cdots,s_k/x_k]$$

$t \equiv$ **let** $x \Leftarrow t_1$ **in** t_2:设$t_1 : \sigma_1, t_2 : \sigma_2$,又设$x_1 : \tau_1,\cdots,x_k : \tau_k$是$t$的所有自由变量,$\lfloor v_1 \rfloor \lesssim_{\tau_1} s_1,\cdots,\lfloor v_k \rfloor \lesssim_{\tau_k} s_k$。根据指称语义,如果$[\![$**let** $x \Leftarrow t_1$ **in** $t_2]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k] = \lfloor u \rfloor$则存在$u_1 \in V^e_{\sigma_1}$,有

$$[\![t_1]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k] = \lfloor u_1 \rfloor$$
$$[\![t_2]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k][u_1/x] = \lfloor u \rfloor$$

(因为x可能出现在$x_1,\cdots,x_k$中,所以我们用$\rho[v_1/x_1,\cdots,v_k/x_k][u_1/x]$代替$\rho[v_1/x_1,\cdots,v_k/x_k,u_1/x]$。)

由归纳可知,存在标准型c_1和c_2,使得

$$u_1 \lesssim^{\circ}_{\sigma_1} c_1 \ \& \ t_1[s_1/x_1,\cdots,s_k/x_k] \to^e c_1$$
$$u \lesssim^{\circ}_{\sigma_2} c_2 \ \& \ t_2[c_1/x][s_1/x_1,\cdots,s_k/x_k] \to^e c_2$$

(又因为x可能出现在$x_1,\cdots,x_k$中,所以我们必须仔细处理代入$t_2[c_1/x][s_1/x_1,\cdots,s_k/x_k]$。)

所以,由操作语义得到

198

$$(\textbf{let } x \Leftarrow t_1 \textbf{ in } t_2)[s_1/x_1,\cdots,s_k/x_k] \to^e c_2$$

从而推得

$$[\![\textbf{let } x \Leftarrow t_1 \textbf{ in } t_2]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k] \lesssim_{\sigma_2} (\textbf{let } x \Leftarrow t_1 \textbf{ in } t_2)[s_1/x_1,\cdots,s_k/x_k]$$

$t \equiv$ **rec** $y.(\lambda x.t_1)$:设$x : \sigma, t_1 : \sigma_1$,又设$x_1 : \tau_1,\cdots,x_k : \tau_k$是$t$的所有自由变量,$\lfloor v_1 \rfloor \lesssim_{\tau_1} s_1,\cdots,\lfloor v_k \rfloor \lesssim_{\tau_k} s_k$,我们假设,对$\varphi \in V^e_{\sigma \to \sigma_1}$,有

$$[\![\textbf{rec } y.(\lambda x.t_1)]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k] = \lfloor \varphi \rfloor$$

根据指称语义,有

$$\varphi = \mu\varphi.\lambda v.[\![t_1]\!]^e\rho[v_1/x_1,\cdots,v_k/x_k,v/x,\varphi/y]$$

所以,$\varphi = \bigsqcup_{n \in \omega}\varphi^{(n)}$,其中每一$\varphi^{(n)} \in V^e_{\sigma \to \sigma_1}$由下式归纳给出:

$$\varphi^{(0)} = \perp_{V^e_{\sigma\to\sigma_1}}$$

$$\varphi^{(n+1)} = \lambda v.\ [t_1]^e\rho[v_1/x_1,\cdots,v_k/x_k,v/x,\varphi^{(n)}/y]$$

根据归纳法,我们可以证明

$$\varphi^{(n)} \lesssim^{\circ}_{\sigma\to\sigma_1} \lambda x.\ t_1[s_1/x_1,\cdots,s_k/x_k,t[s_1/x_1,\cdots,s_k/x_k]/y] \tag{1}$$

由引理 11.12 得

$$\varphi \lesssim^{\circ}_{\sigma\to\sigma_1} \lambda x.\ t_1[s_1/x_1,\cdots,s_k/x_k,t[s_1/x_1,\cdots,s_k/x_k]/y]$$

因为

$$t[s_1/x_1,\cdots,s_k/x_k] \to^e \lambda x.\ t_1[s_1/x_1,\cdots,s_k/x_k,t[s_1/x_1,\cdots,s_k/x_k]/y]$$

所以,我们能得到

$$\lfloor\varphi\rfloor \lesssim_{\sigma\to\sigma_1} t[s_1/x_1,\cdots,s_k/x_k]$$

我们现在用归纳法证明(1)。

奠基 $n=0$:我们希望证明 $\varphi^{(0)} \lesssim^{\circ}_{\sigma\to\sigma_1} \lambda x.\ t_1[s_1/x_1,\cdots,s_k/x_k,t[s_1/x_1,\cdots,s_k/x_k]/y]$,即每当 $v\lesssim^{\circ}_{\sigma} c$,有 $\varphi^{(0)}(v) \lesssim_{\sigma_1} t_1[s_1/x_1,\cdots,s_k/x_k,t[s_1/x_1,\cdots,s_k/x_k]/y,c/x]$。但由引理 11.12(i)知,因为 $\varphi^{(0)}(v) = \perp$,上式(1)成立。

归纳步骤:归纳假设

$$\varphi^{(n)} \lesssim^{\circ}_{\sigma\to\sigma_1} \lambda x.\ t_1[s_1/x_1,\cdots,s_k/x_k,t[s_1/x_1,\cdots,s_k/x_k]/y]$$ 199

则

$$\lfloor\varphi^{(n)}\rfloor \lesssim_{\sigma\to\sigma_1} t[s_1/x_1,\cdots,s_k/x_k] \tag{2}$$

我们需要证明

$$\varphi^{(n+1)} \lesssim^{\circ}_{\sigma\to\sigma_1} \lambda x.\ t_1[s_1/x_1,\cdots,s_k/x_k,t[s_1/x_1,\cdots,s_k/x_k]/y]$$

即,对所有 $v\lesssim^{\circ}_{\sigma} c$,有

$$\varphi^{(n+1)}(v) \lesssim_{\sigma_1} t_1[s_1/x_1,\cdots,s_k/x_k,c/x,t[s_1/x_1,\cdots,s_k/x_k]/y]$$

为此,设 $v\lesssim^{\circ}_{\sigma} c$,则

$$\lfloor v\rfloor \lesssim_{\sigma} c \tag{3}$$

我们回想到 $\varphi^{(n+1)}(v) = [t_1]^e\rho[v_1/x_1,\cdots,v_k/x_k,v/x,\varphi^{(n)}/y]$,所以,根据主要的结构归纳假设,使用(2)和(3),有

$$[t_1]^e\rho[v_1/x_1,\cdots,v_k/x_k,v/x,\varphi^{(n)}/y] \lesssim_{\sigma_1}$$
$$t_1[s_1/x_1,\cdots,s_k/x_k,c/x,t[s_1/x_1,\cdots,s_k/x_k]/y]$$

这就是我们需要证明的。数学归纳法证明完毕,从而完成了整个结构归纳法的最后一种情况。

□

正如本节开始所述,求值关系和指称语义对基类型 **int** 是一致的。

推论 11.15 假设 t 是封闭项 t: **int**,则对任意 $n \in \mathbf{N}$,有

$$t \rightarrow^{e} n \text{ 当且仅当 } [t]^{e}\rho = \lfloor n \rfloor$$

11.5 惰性语言

我们现在考察用传名调用(或惰性)方式求值的高阶类型语言。我们给出它的操作语义和指称语义以及它们之间的一致性。其语法和活性语言语法几乎相同,惟一的差别是递归的语法有所不同。

200 惰性语言的递归定义形为 **rec** $x.\ t$,它和活性语言不同,我们不坚持体 t 是一个抽象。类型规则修改为

$$\frac{x : \tau \quad t : \tau}{\mathbf{rec}\ x.\ t : \tau}$$

除了递归定义更为宽松外,惰性语言的语法和活性语言的语法相同。类似地,当存在一个类型 τ 时,我们说 t 是有类型的,其中 $t : \tau$ 能从类型规则中推导出来。项的自由变量的定义和以前一样,但对递归定义的自由变量的定义略有不同,即

$$FV(\mathbf{rec}\ x.\ t) = FV(t) \setminus \{x\}$$

不含自由变量的项称为封闭的项。

11.6 惰性操作语义

有类型的封闭项的求值结果是标准型。在惰性语言中,基类型和函数类型的标准型分别是数和抽象。然而,和活性语言不同,积类型的标准型是有类型的封闭项的任一序偶,这些封闭项不一定是标准型。惰性语言的标准型 C^{l}_{τ} 由类型 τ 归纳给出:

基类型:$n \in C^{l}_{\mathbf{int}}$ ⊖。

积类型:$(t_1, t_2) \in C^{l}_{\tau_1 * \tau_2}$,如果 $t_1 : \tau_1$ & $t_2 : \tau_2$ 且 t_1 和 t_2 是封闭的。

函数类型:$\lambda x.\ t \in C^{l}_{\tau_1 \to \tau_2}$ 如果 $\lambda x.\ t : \tau_1 \to \tau_2$ 且 $\lambda x.\ t$ 是封闭的。

201 惰性求值用有类型的封闭项 t 和标准型 c 之间的关系 $t \rightarrow^{l} c$ 表示。

求值规则

标准型: $c \rightarrow^{l} c$, 其中 $c \in C^{l}_{\tau}$

操作: $\dfrac{t_1 \rightarrow^{l} n_1 \quad t_2 \rightarrow^{l} n_2}{t_1\ \mathbf{op}\ t_2 \rightarrow^{l} n_1\ op\ n_2}$ 其中 **op** 是 +, −, ×

$\dfrac{t_0 \rightarrow^{l} 0 \quad t_1 \rightarrow^{l} c_1}{\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 \rightarrow^{l} c_1}$ $\dfrac{t_0 \rightarrow^{l} n \quad t_2 \rightarrow^{l} c_2 \quad n \neq 0}{\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2 \rightarrow^{l} c_2}$

⊖ 原文为 int,有误。——译者注

积：
$$\frac{t \to^l (t_1, t_2) \quad t_1 \to^l c_1}{\mathbf{fst}(t) \to^l c_1} \qquad \frac{t \to^l (t_1, t_2) \quad t_2 \to^l c_2}{\mathbf{snd}(t) \to^l c_2}$$

函数：
$$\frac{t_1 \to^l \lambda x.\, t_1' \quad t_1'[t_2/x] \to^l c}{(t_1\ t_2) \to^l c}$$

let：
$$\frac{t_2[t_1/x] \to^l c}{\mathbf{let}\ x \Leftarrow t_1\ \mathbf{in}\ t_2 \to^l c}$$

rec：
$$\frac{t[\mathbf{rec}\ x.\, t/x] \to^l c}{\mathbf{rec}\ x.\, t \to^l c}$$

惰性求值和活性求值在函数应用的情况下有显著区别；在惰性求值中，不必先计算函数的变量——这是惰性求值的本质。同样注意到积规则不再规定如何求值序偶——因为它们已经是标准型，所以没有必要再规定它们如何求值。由于序偶的分量未必是标准型，所以序偶的第一分量和第二分量的获得需要进一步求值。由于不再是递归定义的一次展开就产生一个标准型，所以递归定义的求值规则与活性求值下的情况也不同。在惰性语言中我们使用简化的方式处理 **let** 表达式。

为了进一步的引用，我们注意到惰性计算是确定的和类型一致的。

命题 11.16 如果 $t \to^l c$ 并且 $t \to^l c'$，则 $c \equiv c'$。如果 $t \to^l c$ 并且 $t : \tau$，则 $c : \tau$。

证明 用规则归纳法证明。 □ 202

11.7 惰性指称语义

有类型的封闭项可以惰性求值到标准型或者发散。因此，我们把它的指称取为 $(V_\tau^l)_\perp$ 的一个元素，其中 V_τ^l 是值的完全偏序包括类型为 τ 的标准型的指称。

施结构归纳于类型 τ 来定义 V_τ^l：

$$V_{\mathbf{Int}}^l = \mathbf{N}$$
$$V_{\tau_1 * \tau_2}^l = (V_{\tau_1}^l)_\perp \times (V_{\tau_2}^l)_\perp$$
$$V_{\tau_1 \to \tau_2}^l = [(V_{\tau_1}^l)_\perp \to (V_{\tau_2}^l)_\perp]$$

这些定义反映出积类型的值是序偶，而不论序偶是否含有发散的分量，也反映出函数类型的值可以看成是函数，该函数未必先计算出它的变量值。

惰性语言的环境是函数 ρ：

$$\rho : \mathbf{Var} \to \bigcup \{(V_\tau^l)_\perp \mid \tau \text{ 是类型}\}$$

它保持类型一致性，即，对任一变量 x 和类型 τ，如果 $x : \tau$，则 $\rho(x) \in (V_\tau^l)_\perp$。记 $\mathbf{Env}^l$ 为惰性环境的完全偏序。

现在我们给出惰性语言的指称语义。类型为 τ 的项 t 的指称是从环境 $\mathbf{Env}^l$ 到完全偏序 $(V_\tau^l)_\perp$ 的函数。用结构归纳法定义有类型的项 t 的指称，同样没有超出 8.4 节元语言的范围：

$$
\begin{aligned}
[x]^l &= \lambda\rho.\,\rho(x)\\
[n]^l &= \lambda\rho.\,\lfloor n\rfloor\\
[t_1\ \mathbf{op}\ t_2]^l &= \lambda\rho.\,([t_1]^l\rho\ op_\perp\ [t_2]^l\rho),\text{其中 } \mathbf{op} \text{ 是 } +,\ -,\ \times\\
[\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2]^l &= \lambda\rho.\,Cond([t_0]^l\rho,[t_1]^l\rho,[t_2]^l\rho)\\
[(t_1,t_2)]^l &= \lambda\rho.\,\lfloor([t_1]^l\rho,[t_2]^l\rho)\rfloor\\
[\mathbf{fst}(t)]^l &= \lambda\rho.\,let\ v\Leftarrow[t]^l\rho.\,\pi_1(v)\\
[\mathbf{snd}(t)]^l &= \lambda\rho.\,let\ v\Leftarrow[t]^l\rho.\,\pi_2(v)\\
[\lambda x.\,t]^l &= \lambda\rho.\,\lfloor\lambda d\in(V^l_{\tau_1})_\perp.\,[t]^l\rho[d/x]\rfloor\\
&\qquad \text{其中 } \lambda x.\,t:\tau_1 -> \tau_2\\
[(t_1\ t_2)]^l &= \lambda\rho.\,let\ \varphi\Leftarrow[t_1]^l\rho.\,\varphi([t_2]^l\rho)\\
[\mathbf{let}\ x\Leftarrow t_1\ \mathbf{in}\ t_2]^l &= \lambda\rho.\,[t_2]^l\rho[[t_1]^l\rho/x]\\
[\mathbf{rec}\ x.\,t]^l &= \lambda\rho.\,(\mu d.\,[t]^l\rho[d/x])
\end{aligned}
$$

203

我们注意,一些后面用到的事实。

引理 11.17 设 t 是有类型的项,设环境 ρ,ρ' 对 t 的 $FV(t)$ 取值相同,则 $[t]^l\rho = [t]^l\rho'$。

证明 用简单的结构归纳法证明。 □

引理 11.18(代入引理) 设 s 是封闭项 $s:\tau$,x 是变量 $x:\tau$,假设项 t 是 $t:\tau'$,则 $t[s/x]:\tau'$ 且 $[t[s/x]]^l\rho = [t]^l\rho\,[[s]^l\rho/x]$。

证明 用结构归纳法证明。 □

引理 11.19

(i) 如果 $t:\tau$,则对任一环境 $\rho\in\mathbf{Env}^l$,有 $[t]^l\rho\in(V^l_\tau)_\perp$。

(ii) 如果 $c\in C^l_\tau$,则对任一环境 $\rho\in\mathbf{Env}^l$,有 $[c]^l\rho\neq\perp$,其中 $\perp$ 为 $(V^l_\tau)_\perp$ 的底元素。

证明 简单地施结构归纳于 t 去证明(i),施结构归纳于标准型 c 去证明(ii)。 □

11.8 惰性语义的一致性

我们证明指称语义适用于操作语义,即当项的求值收敛时,指称语义保持求值关系并且和操作语义是一致的。

设 t 是有类型的封闭项。对应于惰性求值,操作收敛定义为

$$t\downarrow^l \text{ 当且仅当 } \exists c.\,t\rightarrow^l c$$

而指称收敛定义为

$$t\Downarrow^l \text{ 当且仅当 } \exists v\in V^l_\tau.\,[t]^l\rho = \lfloor v\rfloor$$

其中 ρ 是 $\mathbf{Env}^l$ 中的任意环境。

假设 $t\downarrow^l$,则对某个标准型 c,有 $t\rightarrow^l c$。我们将会证明,对任意环境 ρ,有 $[t]^l\rho = [c]^l\rho$。因为由引理 11.19 知 $c\Downarrow^l$,所以蕴涵 $t\Downarrow^l$。我们还将证明较难的另外一个方向,如果 $t\Downarrow^l$,则 $t\downarrow^l$;根据指称语义,如果 t 指称 $\lfloor v\rfloor$,则 t 的求值收敛于一个标准型 c,其指称是 $\lfloor v\rfloor$。证明方

204

法和活性语言中的证明方法类似。

首先我们证明,指称语义保持求值关系。

引理 11.20 如果 $t \to^l c$,则对任意环境 ρ,有 $[t]^l\rho = [c]^l\rho$。

证明 施规则归纳于惰性求值规则去证明。可参考引理 11.11 的证明,作为练习留给读者。 □

练习 11.21 证明引理 11.20。 □

现在我们证明较困难的相反方向:对某个标准型 c,如果 t 是封闭项且 $t \Downarrow^l$,则 $t \to^l c$。与在活性语言中的做法相类似,我们使用项及其指称之间较强的逻辑关系去证明。现在我们对类型 τ,定义逻辑关系 $\lesssim^\circ_\tau \subseteq V^l_\tau \times C^l_\tau$。和前面一样,我们把值和标准型之间的这些基本关系扩展到 $(V^l_\tau)_\perp$ 的元素 d 和封闭项 t 之间的关系。定义

$$d \lesssim_\tau t \text{ 当且仅当}$$
$$\forall v \in V^l_\tau.\ d = \lfloor v \rfloor \Rightarrow \exists c.\ t \to^l c \ \&\ v \lesssim^\circ_\tau c$$

施结构归纳于类型 τ 去定义基本关系 $\lesssim^\circ_\tau$:

基类型:对所有的数 n,有 $n \lesssim^\circ_{\mathrm{int}} n$。

积类型:$(d_1, d_2) \lesssim^\circ_{\tau_1 * \tau_2} (t_1, t_2)$ 当且仅当 $d_1 \lesssim_{\tau_1} t_1 \ \&\ d_2 \lesssim_{\tau_2} t_2$。

函数类型:$\varphi \lesssim^\circ_{\tau_1 \to \tau_2} \lambda x.\ t$ 当且仅当 $\forall d \in (V^l_{\tau_1})_\perp$,封闭项 $u: \tau_1.\ d \lesssim_{\tau_1} u \Rightarrow \varphi(d) \lesssim_{\tau_2} t[u/x]$。

我们用类似于引理 11.12 的方法去考察逻辑关系的性质。

引理 11.22 设 $t: \tau$,则

(i) $\perp_{(V^l_\tau)_\perp} \lesssim_\tau t$。

(ii) 如果 $d \sqsubseteq d'$ 且 $d' \lesssim_\tau t$,则 $d \lesssim_\tau t$。

(iii) 如果 $d_0 \sqsubseteq d_1 \sqsubseteq \cdots \sqsubseteq d_n \sqsubseteq \cdots$ 是 $(V^l_\tau)_\perp$ 中的 ω 链,使得对所有的 $n \in \omega$,有 $d_n \lesssim_\tau t$,则 $\bigsqcup_{n \in \omega} d_n \lesssim_\tau t$。 205

证明 证明与引理 11.12 类似。根据定义,直接可得性质(i)。施结构归纳于类型去证明性质(ii)和性质(iii)对所有的项成立。 □

引理 11.23 设 t 是有类型的封闭项,则有

$$t \Downarrow^l \quad \text{蕴涵} \quad t \downarrow^l$$

证明 证明和引理 11.14 的证明类似。施结构归纳于项进行证明:对所有的自由变量来自于 $x_1: \tau_1, \cdots, x_k: \tau_k$ 的项 $t: \tau$,如果 $d_1 \lesssim_{\tau_1} s_1, \cdots, d_k \lesssim_{\tau_k} s_k$,则

$$[t]^l\rho[d_1/x_1, \cdots, d_k/x_k] \lesssim_\tau t[s_1/x_1, \cdots, s_k/x_k]$$

取 t 是封闭的,根据 $\lesssim_\tau$ 的定义,如果 $t \Downarrow^l$,则对某个 v,有 $[t]^l\rho = \lfloor v \rfloor$,因此,对某个标准型 c,有 $t \to^l c$。以下只给出和引理 11.14 证明有明显不同之处的证明部分:

$t \equiv \mathbf{fst}(s)$:我们设 $\mathbf{fst}(s)$ 是有类型的,因此,s 必须有类型 $\sigma_1 * \sigma_2$,又设 $x_1: \tau_1 \cdots, x_k: \tau_k$

是 t 的所有自由变量，且 $d_1 \lesssim_{\tau_1} s_1, \cdots, d_k \lesssim_{\tau_k} s_k$，假设 $[\mathbf{fst}(s)]^l\rho[d_1/x_1, \cdots, d_k/x_k] = \lfloor v_1 \rfloor$。则由指称语义，我们有 $[s]^l\rho[d_1/x_1, \cdots, d_k/x_k] = \lfloor u \rfloor$，其中 $\lfloor v_1 \rfloor = \pi_1(u)$。由归纳，我们得

$$[s]^l\rho[d_1/x_1, \cdots, d_k/x_k] \lesssim_{\sigma_1 * \sigma_2} s[s_1/x_1, \cdots, s_k/x_k]$$

于是

$$s[s_1/x_1, \cdots, s_k/x_k] \rightarrow^l (t_1, t_2), \text{其中 } u \lesssim^{\circ}_{\sigma_1 * \sigma_2} (t_1, t_2)$$

根据 $\lesssim^{\circ}_{\sigma_1 * \sigma_2}$ 的定义，有

$$\lfloor v_1 \rfloor \lesssim_{\sigma_1} t_1$$

再由 $\lesssim_{\sigma_1}$ 的定义，对某个标准型 c_1，我们得

$$v_1 \lesssim^{\circ}_{\sigma_1} c_1 \ \& \ t_1 \rightarrow^l c_1$$

由操作语义，可知

$$\mathbf{fst}(s)[s_1/x_1, \cdots, s_k/x_k] \equiv \mathbf{fst}(s[s_1/x_1, \cdots, s_k/x_k]) \rightarrow^l c_1$$

206 所以，正如我们期望的，有 $[\mathbf{fst}(s)]^l\rho[d_1/x_1, \cdots, d_k/x_k] \lesssim_{\sigma_1} \mathbf{fst}(s)[s_1/x_1, \cdots, s_k/x_k]$。

$t \equiv t_1\ t_2$：设 $t_1 : \sigma_2 \rightarrow \sigma$，$t_2 : \sigma_2$，又设 t 有自由变量 $x_1 : \tau_1, \cdots, x_k : \tau_k$，以及 $d_1 \lesssim_{\tau_1} s_1, \cdots, d_k \lesssim_{\tau_k} s_k$，假设

$$d = [t_2]^l\rho[d_1/x_1, \cdots, d_k/x_k]$$

根据指称语义，我们得

$$[t_1\ t_2]^l\rho[d_1/x_1, \cdots, d_k/x_k] =$$
$$let\ \varphi \Leftarrow [t_1]^l\rho[d_1/x_1, \cdots, d_k/x_k].\ \varphi(d)$$

假设对 $u \in V^l_\tau$，$[t_1\ t_2]^l\rho[d_1/x_1, \cdots, d_k/x_k] = \lfloor u \rfloor$，于是存在一个 φ 使得

$$[t_1]^l\rho[d_1/x_1, \cdots, d_k/x_k] = \lfloor \varphi \rfloor$$

其中

$$\varphi(d) = \lfloor u \rfloor$$

注意到，由归纳，我们有

$$[t_1]^l\rho[d_1/x_1, \cdots, d_k/x_k] \lesssim_{\sigma_2 \rightarrow \sigma} t_1[s_1/x_1, \cdots, s_k/x_k]$$

对标准型 $\lambda x.\ t_1'$，我们得到

$$t_1[s_1/x_1, \cdots, s_k/x_k] \rightarrow^l \lambda x.\ t_1' \ \& \ \varphi \lesssim^{\circ}_{\sigma_2 \rightarrow \sigma} \lambda x.\ t_1'$$

同样，根据归纳法，因为 $d = [t_2]^l\rho[d_1/x_1, \cdots, d_k/x_k]$，所以

$$d \lesssim_{\sigma_2} t_2[s_1/x_1, \cdots, s_k/x_k]$$

由$\lesssim^{\circ}_{\sigma_2 \to \sigma}$的定义,得到

$$\varphi(d) \lesssim_{\sigma} t_1'[t_2[s_1/x_1,\cdots,s_k/x_k]/x]$$

由于$\varphi(d) = \lfloor u \rfloor$,所以存在$c \in C^l_{\sigma}$使得

$$t_1'[t_2[s_1/x_1,\cdots,s_k/x_k]/x] \to^l c \ \&\ u \lesssim^{\circ}_{\sigma} c$$

根据操作语义,我们推出

$$(t_1\ t_2)[s_1/x_1,\cdots,s_k/x_k] \to^l c$$

并得到

$$[t_1\ t_2]^l\rho[d_1/x_1,\cdots,d_k/x_k] \lesssim_{\sigma} (t_1\ t_2)[s_1/x_1,\cdots,s_k/x_k]$$ 207

$t \equiv \mathbf{rec}\ y.\ t_1$:设$y:\sigma$且$t_1:\sigma_1$,又设$x_1:\tau_1,\cdots,x_k:\tau_k$是$t$的所有自由变量,假设$d_1 \lesssim_{\tau_1} s_1,\cdots,d_k \lesssim_{\tau_k} s_k$,根据指称语义,有

$$\theta =_{def} [\mathbf{rec}\ y.\ t_1]^l\rho[d_1/x_1,\cdots,d_k/x_k] = \mu\theta.\ [t_1]^l\rho[d_1/x_1,\cdots,d_k/x_k,\theta/y]$$

于是$\theta = \bigsqcup_{n\in\omega}\theta^{(n)}$,其中每一$\theta^{(n)} \in (V^l_{\sigma})_{\perp}$由下式归纳给出:

$$\theta^{(0)} = \perp_{(V^l_{\sigma})_{\perp}}$$
$$\theta^{(n+1)} = [t_1]^l\rho[d_1/x_1,\cdots,d_k/x_k,\theta^{(n)}/y]$$

根据归纳,可得到

$$\theta^{(n)} \lesssim_{\sigma} \mathbf{rec}\ y.\ t_1[s_1/x_1,\cdots,s_k/x_k] \tag{1}$$

(注意$\mathbf{rec}\ y.\ t_1$的所有自由变量$x_1,\cdots,x_k$必须与y不同,因此,代入

$$(\mathbf{rec}\ y.\ t_1)[s_1/x_1,\cdots,s_k/x_k]$$

和代入

$$\mathbf{rec}\ y.\ (t_1[s_1/x_1,\cdots,s_k/x_k])$$

得到相同的项。)

由引理11.22,得

$$\theta \lesssim_{\sigma} \mathbf{rec}\ y.\ t_1[s_1/x_1,\cdots,s_k/x_k]$$

我们现在用归纳法证明(1):

归纳奠基 $n=0$:我们要证明,$\theta^{(0)} \lesssim_{\sigma} \mathbf{rec}\ y.\ t_1[s_1/x_1,\cdots,s_k/x_k]$,由引理11.22(i),因为$\theta^{(0)} = \perp$,所以上式成立。

归纳步骤:归纳假设$\theta^{(n)} \lesssim_{\sigma} \mathbf{rec}\ y.\ t_1[s_1/x_1,\cdots,s_k/x_k]$,则根据结构归纳法

$$\theta^{(n+1)} = [t_1]^l\rho[d_1/x_1,\cdots,d_k/x_k,\theta^{(n)}/y]$$

$$\leq_\sigma t_1[s_1/x_1,\cdots,s_k/x_k,\mathbf{rec}\ y.\ t_1[s_1/x_1,\cdots,s_k/x_k]/y]$$

208

$$\equiv t_1[\mathbf{rec}\ y.\ t_1/y][s_1/x_1,\cdots,s_k/x_k]$$

根据操作语义,我们可知,对标准型 c,

如果 $t_1[\mathbf{rec}\ y.\ t_1/y][s_1/x_1,\cdots,s_k/x_k] \rightarrow^l c$,则 $\mathbf{rec}\ y.\ t_1[s_1/x_1,\cdots,s_k/x_k] \rightarrow^l c$

现在,根据 $\leq_\sigma$ 的定义,我们得

$$\theta^{(n+1)} \leq_\sigma \mathbf{rec}\ y.\ t_1[s_1/x_1,\cdots,s_k/x_k]$$

对这种情形的数学归纳法证明完毕。 □

下面的推论表明求值关系和指称语义在基类型 **int** 上是一致的。

推论 11.24 假设 t 是封闭项 t: **int**,则对任意 $n \in \mathbf{N}$,有

$$t \rightarrow^l n \text{ 当且仅当 } [\![t]\!]^l\rho = \lfloor n \rfloor$$

11.9 不动点算子

指称语义给出了论证高阶类型语言的项求值的数学模型。下面我们要研究如何用活性语言和惰性语言去表达不动点算子。

首先我们考虑惰性求值。不动点算子是类型为 $(\tau \rightarrow \tau) \rightarrow \tau$ 的封闭项 Y,当 Y 应用到抽象 F 上时,就产生 F 的不动点,即:

$$[\![F(YF)]\!]^l\rho = [\![YF]\!]^l\rho$$

假设 Y 满足上述方程,一个合理猜测的合适定义是

$$\mathbf{rec}\ Y.\ (\lambda f.\ f(Yf))$$

实际上,按照指称语义,这确实定义了一个不动点算子。

我们考察

$$R \equiv \mathbf{rec}\ Y.\ (\lambda f.\ f(Yf))$$

的指称——假设 R 是有类型的,则 R: $(\tau \rightarrow \tau) \rightarrow \tau$。按照指称语义,

$$[\![R]\!]^l\rho = \mu U.\ [\![\lambda f.\ f(Yf)]\!]^l\rho[U/Y]$$

209

$$= \mu U \lfloor \lambda\varphi.\ let\ \varphi' \Leftarrow \varphi.\ \varphi'\ (let\ U' \Leftarrow U.\ U'(\varphi)) \rfloor$$

先借助于从(含底 $\perp_C$ 的)完全偏序提升 $C_\perp$ 到 C 的连续函数

$$down_C:\ C_\perp \rightarrow C$$

简化这个表达式是有帮助的。这样一个函数定义为

$$down_C(\varphi) = let\ \varphi' \Leftarrow \varphi.\ \varphi'$$

或等价地定义为

$$down_C(\varphi) = \begin{cases} \varphi' & \varphi = \lfloor \varphi' \rfloor \\ \perp_C & \text{其他} \end{cases}$$

我们关注特殊情况下这样的函数 C,考察 C 是形如 $D \to E$ 的函数空间,其中 E 是含底的完全偏序。在这种情况下,我们有下述引理。

引理 11.25 设 C 是完全偏序 $[D \to E]$,其中 E 是含底元素 $\perp_E$ 的完全偏序,则对 $\varphi \in C_\perp, d \in D$,有

$$(down_C(\varphi))(d) = let\ \varphi' \Leftarrow \varphi.\ \varphi'(d)$$

证明 当 φ 形为 $\lfloor \varphi' \rfloor$ 时,等式显然成立。在 $\varphi = \perp$ 的情况下,右边是 $\perp_E$,而左边 $(\lambda d \in D.\ \perp_E)(d) = \perp_E$,两边相等。 □

由引理知,$V^l_{\tau \to \tau}$ 和 $V^l_{(\tau \to \tau) \to \tau}$ 都是含底的完全偏序,具有引理所要求的形式。相应地,存在函数

$$down : (V^l_{\tau \to \tau})_\perp \to V^l_{\tau \to \tau} \text{ 和}$$
$$down : (V^l_{(\tau \to \tau) \to \tau})_\perp \to V^l_{(\tau \to \tau) \to \tau}$$

(这里忽略了两个不同的“*down*”函数的下标。)

使用它们可以简化 $[R]^l\rho$:

$$[R]^l\rho = \mu U.\ \lfloor \lambda\varphi.\ (down(\varphi))((down(U))(\varphi)) \rfloor$$

从 R 的指称的这个简化形式,我们可得

$$[R]^l\rho = \bigsqcup_{n \in \omega} U^{(n)}$$

其中 $U^{(0)} = \perp$

$$U^{(1)} = \lfloor \lambda\varphi.\ (down(\varphi))(\perp(\varphi)) \rfloor = \lfloor \lambda\varphi.\ (down(\varphi))(\perp) \rfloor$$

210

且由归纳得

$$U^{(n)} = \lfloor \lambda\varphi.\ (down(\varphi))((down(U^{(n-1)}))(\varphi)) \rfloor = \lfloor \lambda\varphi.\ (down(\varphi))^n(\perp) \rfloor$$

所以

$$\begin{aligned}
[R]^l\rho &= \bigsqcup_{n \in \omega} U^{(n)} \\
&= \bigsqcup_{n \in \omega} \lfloor \lambda\varphi.\ (down(\varphi))^n(\perp) \rfloor \\
&= \lfloor \bigsqcup_{n \in \omega} \lambda\varphi.\ (down(\varphi))^n(\perp) \rfloor && \text{(由}\lfloor\text{-}\rfloor\text{的连续性)} \\
&= \lfloor \lambda\varphi.\ \bigsqcup_{n \in \omega} (down(\varphi))^n(\perp) \rfloor && \text{(因为函数的最小上界是逐点逼近的)} \\
&= \lfloor \lambda\varphi.\ fix(down(\varphi)) \rfloor && \text{(由 } fix \text{ 的定义)}
\end{aligned}$$

从这个刻画,我们得到 R 是一不动点算子。在 F 是类型为 $\tau \to \tau$ 的抽象的情况下,对某个

$\varphi': (V_\tau^l)_\perp \to (V_\tau^l)_\perp$，我们有

$$[F]^l\rho = \lfloor \varphi' \rfloor$$

所以

$$\begin{aligned}[F(RF)]^l\rho &= \varphi'([RF]^l\rho) \\ &= \varphi'(fix(down(\lfloor \varphi' \rfloor))) \\ &= \varphi'(fix(\varphi')) \\ &= fix(\varphi') \\ &= [RF]^l\rho\end{aligned}$$

练习 11.26 试证明：对 $F: \tau \to \tau$，即使 $[F]^l\rho = \perp$，也有

$$[F(RF)]^l\rho = [RF]^l\rho$$ □

$[R]^l\rho$ 的刻画使我们能证明程序

211

$$R(\lambda x.\ t) \quad \mathbf{rec}\ x.\ t$$

是等价的，因为它们有相同的指称。利用指称语义，我们可以简单地证明：

$$\begin{aligned}[R(\lambda x.\ t)]^l\rho &= fix(\lambda d.\ [t]^l\rho[d/x]) \\ &= \mu d.\ [t]^l\rho[d/x] \\ &= [\mathbf{rec}\ x.\ t]^l\rho\end{aligned}$$

因此，对惰性求值，不动点算子的定义是相当直观的，那么活性求值的不动点算子是什么呢？下面我们将证明同样的定义不再成立。由活性语言的指称语义可知

$$\begin{aligned}[R]^e\rho &= \lfloor \mu U.\ (\lambda\varphi.\ [f(Yf)]^e\rho[\varphi/f, U/Y]) \rfloor \\ &= \lfloor \mu U.\ (\lambda\varphi.\ let\ v \Leftarrow U(\varphi).\ \varphi(v)) \rfloor\end{aligned}$$

通过考察它的逼近点，我们能论证

$$\mu U.\ (\lambda\varphi.\ let\ v \Leftarrow U(\varphi).\ \varphi(v)) = \lambda\varphi.\ \perp$$

我们知道，这个不动点是 $\bigsqcup_{n\in\omega} U^{(n)}$，其中

$$\begin{aligned}U^{(0)} &= \lambda\varphi.\ \perp \\ U^{(n)} &= \lambda\varphi.\ (let\ v \Leftarrow U^{(n-1)}(\varphi).\ \varphi(v))，其中\ n > 0\end{aligned}$$

由此我们看到

$$\begin{aligned}U^{(1)} &= \lambda\varphi.\ (let\ v \Leftarrow \perp.\ \varphi(v)) \\ &= \lambda\varphi.\ \perp\end{aligned}$$

通过类似的简单归纳，对所有的 $n > 0$，有

$$U^{(n)} = \lambda\varphi.\ (let\ v \Leftarrow U^{n-1}(\varphi).\ \varphi(v))$$

$$= \lambda\varphi.\perp$$

所以,我们就得到

$$\mu U.(\lambda\varphi.\ let\ v \Leftarrow U(\varphi).\varphi(v)) = \lambda\varphi.\perp$$

因此,有

$$[\![R]\!]^e\rho = \lfloor \lambda\varphi.\perp \rfloor$$

于是

$$[\![R(\lambda x.t)]\!]^e\rho = \perp$$ 212

应用 $R(\lambda x.t)$ 产生了发散的结果 $\perp$,而没有得到不动点。主要原因是:根据指称语义,$U^{(n)}$ 的定义包含了作用于 φ 的 $U^{(n-1)}(\varphi)$ 的预先求值,通过归纳可以发现,这将永远取值 $\perp$。

练习 11.27 试从操作语义的角度证明:$R(\lambda x.t)$ 是发散的,即不存在标准型 c,使得 $R(\lambda x.t) \rightarrow^e c$。 □

因此,我们如何定义活性求值计算的不动点算子呢?关键是使用抽象以延迟求值,用这种方法模仿惰性语言及其不动点算子的简单表达式。

举一个特例,两个项

$$F(YF), \lambda x.((F(YF))x)$$

的活性求值或惰性求值方式是不同的;后者是标准型,所以可直接计算到它自己,而前者须先计算 F,在活性求值的情况下,还要先计算 (YF)。这和数学中的情况不同,在数学中,函数

$$\varphi : X \rightarrow Y$$

和函数

$$\lambda x \in X.\varphi(x) : X \rightarrow Y$$

是一样的(同样的序偶集)。我们来研究这种不同如何反映在指称语义中。

假设 τ 是形如 $\sigma -> \sigma'$ 的函数类型,又假设

$$f : \tau -> \tau,\ Y : (\tau -> \tau) -> \tau \text{ 和 } x : \sigma$$

是变量。我们考察两个具有类型 τ 的项

$$f(Yf),\quad \lambda x.((f(Yf))x)$$

在环境 ρ 下的指称,其中 $\rho(f) = \varphi$, $\rho(Y) = U$。用函数

$$down : (V_\tau^e)_\perp \rightarrow V_\tau^e$$

去简化指称,这个函数将 $\lfloor v \rfloor$ 映为 v,将 $\perp$ 映为 V_τ^e 中取值恒为 $\perp$ 的函数。正如前面那样,由引理 11.25,我们观察到对 $\psi \in (V_\tau^e)_\perp$,有

$$down(\psi) = \lambda w.\ let\ \theta \Leftarrow \psi.\theta(w) \qquad (*)$$ 213

现在由指称语义,我们看到一方面是

$$[\![f(Yf)]\!]^e\rho = let\ v \Leftarrow U(\varphi).\ \varphi(v)$$

它可能为 $\perp \in (V_\tau^e)_\perp$。而另一方面,

$$\begin{aligned}
[\![\lambda x.\ ((f(Yf))x)]\!]^e\rho &= \lfloor \lambda w.\ let\ \theta \Leftarrow [\![f(Yf)]\!]^e\rho.\ \theta(w) \rfloor \\
&= \lfloor \lambda w.\ (down([\![f(Yf)]\!]^e\rho)(w)) \rfloor \quad (\text{由}(*)) \\
&= \lfloor down([\![f(Yf)]\!]^e\rho) \rfloor \quad (\text{数学函数的性质}) \\
&= \lfloor down(let\ v \Leftarrow U(\varphi).\ \varphi(v)) \rfloor
\end{aligned}$$

它永远是 $(V_\tau^e)_\perp$ 中非 $\perp$ 的元素。这种不同是我们在活性求值中得到不动点算子的关键。

我们把 R 重新定义为

$$\mathbf{rec}\ Y.\ (\lambda f.\ \lambda x.\ ((f(Yf))x))$$

于是,由指称语义我们得到

$$[\![R]\!]^e\rho = \lfloor \mu U.\ \lambda\varphi.\ [\![\lambda x.\ (f(Yf))x]\!]^e\rho[\varphi/f, U/Y] \rfloor$$

我们已经简化了 $\lambda x.\ ((f(Yf))x)$ 的指称,使用它我们得到

$$[\![R]\!]^e\rho = \lfloor \mu U.\ \lambda\varphi.\ \lfloor down(let\ v \Leftarrow U(\varphi).\ \varphi(v)) \rfloor \rfloor$$

不动点

$$\mu U.\ \lambda\varphi.\ \lfloor down(let\ v \Leftarrow U(\varphi).\ \varphi(v)) \rfloor$$

为 $\bigsqcup_{n\in\omega} U^{(n)}$,它是由下面归纳给出的"逼近点"的最小上界:

$$\begin{aligned}
U^{(0)} &= \lambda\varphi.\perp \\
U^{(n)} &= \lambda\varphi.\ \lfloor down(let\ v \Leftarrow U^{(n-1)}(\varphi).\ \varphi(v)) \rfloor, \text{其中 } n > 0
\end{aligned}$$

所以,我们得到

$$\begin{aligned}
U^{(1)} &= \lambda\varphi.\ \lfloor down(\perp) \rfloor = \lambda\varphi.\ \lfloor \perp \rfloor \\
U^{(2)} &= \lambda\varphi.\ \lfloor down(\varphi(\perp)) \rfloor = \lambda\varphi.\ \lfloor (down \circ \varphi)(\perp) \rfloor
\end{aligned}$$

并且,由归纳得

214

$$U^{(n)} = \lambda\varphi.\ \lfloor (down \circ \varphi)^{(n-1)}(\perp) \rfloor$$

所以

$$\begin{aligned}
[\![R]\!]^e\rho &= \lfloor \textstyle\bigsqcup_{n\in\omega} U^{(n)} \rfloor \\
&= \lfloor \textstyle\bigsqcup_{n\in\omega} (\lambda\varphi.\ \lfloor (down \circ \varphi)^{(n-1)}(\perp) \rfloor) \rfloor \\
&= \lfloor \lambda\varphi.\ \lfloor \textstyle\bigsqcup_{n\in\omega} (down \circ \varphi)^{(n-1)}(\perp) \rfloor \rfloor \\
&\quad (\text{因为函数的最小上界是逐点逼近的并且} \lfloor - \rfloor \text{是连续的}) \\
&= \lfloor \lambda\varphi.\ \lfloor fix(down \circ \varphi) \rfloor \rfloor
\end{aligned}$$

现在我们能证明

$$[\![R(\lambda y.\ \lambda x.\ t)]\!]^e\rho = [\![\mathbf{rec}\ y.\ (\lambda x.\ t)]\!]^e\rho$$

根据指称语义,我们有

$$\begin{aligned}
&[\![R(\lambda y.\ \lambda x.\ t)]\!]^e\rho \\
&= (\lambda\varphi.\ \lfloor fix(down \circ \varphi)\rfloor)(\lambda\theta.\ \lfloor \lambda v.\ [\![t]\!]^e\rho[v/x,\theta/y]\rfloor) \\
&= \lfloor fix(down \circ (\lambda\theta.\ \lfloor \lambda v.\ [\![t]\!]^e\rho[v/x,\theta/y]\rfloor))\rfloor \\
&= \lfloor fix(\lambda\theta.\ \lambda v.\ [\![t]\!]^e\rho[v/x,\theta/y])\rfloor \quad \text{(读者回想一下 } down \text{ 是如何作用的)} \\
&= \lfloor \mu\theta.\ \lambda v.\ [\![t]\!]^e\rho[v/x,\theta/y]\rfloor \\
&= [\![\mathbf{rec}\ y.\ (\lambda x.\ t)]\!]^e\rho
\end{aligned}$$

11.10 观察与完全抽象

我们已经看到了一些用指称语义的数学模型中的推理去解释程序行为的例子。按照指称语义,有些项当作不动点算子。这些事实很难只用操作语义去证明或者正确表达。也许有人会问,为什么我们可以用指称语义来推导程序在机器上的运行过程?当然,前提是我们如实按照操作语义来实现语言的。因为操作语义和指称语义对感兴趣问题的观察是一致的。如果指称语义表明类型为 **int** 的封闭项指称特定的整数,则它可以精确地求值到该整数;反之亦然。对其他的类型,如果项收敛,即不是指称到 $\perp$,则它的求值也收敛;反之亦然。操作语义和指称语义在项是否收敛上是一致的;在类型为 **int** 的项的求值到整数上是一致的。这种一致是有关表示指称语义对应于操作语义的适用性的结果的内容。实际上,我们把观察限制在收敛的项上。对应于收敛的适用性将确保两种语义在类型为 **int** 的项的求值上是一致的。以下是一个简单的论证,将项代入到下面的上下文中: 215

$$\mathbf{if}\ \text{-}\ \mathbf{then}\ 0\ \mathbf{else}\ \mathrm{Diverge}$$

其中 Diverge:**int** 是发散的封闭项。对封闭项 t: **int** 和数 n,活性语义和惰性语义有

$$\begin{aligned}
t \to n &\Longleftrightarrow \mathbf{if}\ (t - n)\ \mathbf{then}\ 0\ \mathbf{else}\ \mathrm{Diverge}\downarrow \\
&\Longleftrightarrow \mathbf{if}\ (t - n)\ \mathbf{then}\ 0\ \mathbf{else}\ \mathrm{Diverge}\Downarrow \quad \text{(由适用性)} \\
&\Longleftrightarrow [\![t]\!]\rho = n
\end{aligned}$$

对类型为 **int** 的求值和收敛的观察选择是否合理呢?当然,许多实现报告精确地反馈给用户我们讨论过的一类收敛行为,只产生像整数和链表那样的具体数据类型的具体值。从这个角度来说,我们的选择是合理的。另一方面,如果我们对其他性质感兴趣,例如计算一个项需多少时间,我们希望有更精确的观察和更详细的语义。

对观察进行限制也是可能的,从而对固定的操作语义而言可得到更粗略的指称语义。为此,我们给出另一种惰性语言的指称语义。这一次我们忽略了一般高阶类型的收敛情况,但对封闭项 t: **int** 和整数 n 确保基本类型 **int** 有:

$$t \to^l n \text{ 当且仅当 } [\![t]\!]\rho = \lfloor n \rfloor$$

我们观察怎样从类型为 **int** 的项的求值去得到具体的值。

施结构归纳于 τ,去定义类型为 τ 的指称的完全偏序 D_τ:

$$D_{\text{int}} = \mathbf{N}_\perp$$
$$D_{\tau_1 * \tau_2} = D_{\tau_1} \times D_{\tau_2}$$
$$D_{\tau_1 \to \tau_2} = [D_{\tau_1} \to D_{\tau_2}]$$

惰性语言的环境被定义为函数:

216

$$\rho : \mathbf{Var} \to \bigcup \{D_\tau \mid \tau \text{ 为类型}\}$$

使得对任一变量 x 和类型 τ,如果 $x : \tau$,则 $\rho(x) \in D_\tau$。设 **Env** 表示环境的完全偏序,用 8.4 节的元语言,有类型的项 t 的指称是用下面结构归纳法定义的元素 $[\![t]\!]$:

$$\begin{aligned}
[\![x]\!] &= \lambda\rho.\, \rho(x) \\
[\![n]\!] &= \lambda\rho.\, \lfloor n \rfloor \\
[\![t_1 \ \mathbf{op}\ t_2]\!] &= \lambda\rho.\, ([\![t_1]\!]\rho \ op_\perp [\![t_2]\!]\rho), \quad \text{其中 } \mathbf{op} \text{ 是 } +, -, \times \\
[\![\mathbf{if}\ t_0\ \mathbf{then}\ t_1\ \mathbf{else}\ t_2]\!] &= \lambda\rho.\, Cond([\![t_0]\!]\rho, [\![t_1]\!]\rho, [\![t_2]\!]\rho) \\
[\![(t_1, t_2)]\!] &= \lambda\rho.\, ([\![t_1]\!]\rho, [\![t_2]\!]\rho) \\
[\![\mathbf{fst}(t)]\!] &= \lambda\rho.\, \pi_1([\![t]\!]\rho) \\
[\![\mathbf{snd}(t)]\!] &= \lambda\rho.\, \pi_2([\![t]\!]\rho) \\
[\![\lambda x.\, t]\!] &= \lambda\rho.\, \lambda d \in D_{\tau_1}.\, [\![t]\!]\rho[d/x], \quad \text{其中 } \lambda x.\, t : \tau_1 \to \tau_2 \\
[\![(t_1\ t_2)]\!] &= \lambda\rho.\, [\![t_1]\!]\rho([\![t_2]\!]\rho) \\
[\![\mathbf{let}\ x \Leftarrow t_1\ \mathbf{in}\ t_2]\!] &= \lambda\rho.\, [\![t_2]\!]\rho[[\![t_1]\!]\rho/x] \\
[\![\mathbf{rec}\ x.\, t]\!] &= \lambda\rho.\, (\mu d.\, [\![t]\!]\rho[d/x])
\end{aligned}$$

练习 11.28

(1) 假设变量 $x : \mathbf{int} \to \mathbf{int}$, $w : \mathbf{int} \to \mathbf{int}$, $y : \mathbf{int}$,试计算 $((\lambda x.\, x)\Omega)$ 和 $((\lambda x.\, \lambda y.\, (x\ y))\Omega)$ 的指称,其中 $\Omega \equiv \mathbf{rec} w.\, w$。

(2) 试证明:对应于惰性语言的操作语义,对任意环境 ρ,

$$t \to^l c \Rightarrow [\![t]\!]\rho = [\![c]\!]\rho$$

(用规则归纳法证明时,仅需考虑几种情形。你可以假设代入引理的变形,但要叙述清晰。)

(3) 对封闭项 $t : \mathbf{int}$ 证明:对任一 $n \in \mathbf{N}$,有

$$t \to^l n \text{ 当且仅当 } [\![t]\!]\rho = \lfloor n \rfloor$$

建议使用 D_τ 的元素和类型为 τ 的封闭项之间的逻辑关系 $\lesssim_\tau$ 去证明。施结构归纳于类型,定义逻辑关系 $\lesssim_\tau$ 如下:

$$d \lesssim_{\text{int}} t \Longleftrightarrow \forall n \in \mathbf{N}.\, d = \lfloor n \rfloor \Rightarrow t \to^l n,$$
$$d \lesssim_{\tau_1 * \tau_2} t \Longleftrightarrow \pi_1(d) \lesssim_{\tau_1} \mathbf{fst}(t)\ \&\ \pi_2(d) \lesssim_{\tau_2} \mathbf{snd}(t),$$

217

$$d \lesssim_{\tau_1 \to \tau_2} t \Longleftrightarrow \forall e, s.\, e \lesssim_{\tau_1} s \Rightarrow d(e) \lesssim_{\tau_2} (t\ s)$$

首先,施结构归纳于类型 τ 去证明:

$$[d \lesssim_\tau t_1 \ \&\ (t_1 \to^l c \Rightarrow t_2 \to^l c)] \Rightarrow d \lesssim_\tau t_2 \qquad \square$$

对于观察的选择而言,指称语义对于操作语义的适用性结果,对使用指称语义的更能进行数学处理的模型来预测和论证程序行为是至关重要的。这里,有另一个使指称语义成为观察方法的重要准则:语义要完全抽象。完全抽象对指称语义来说,通常是一个比适用性更难满足的性质。幸好它不是很关键。但它在某些方面是一个有用的和重要的性质,因为对获得完全抽象的语义的追求衍生了一个重要的研究方向。要达到类似于本章中的语言的完全抽象,需要在域论的数学框架内,对一些重要的操作思想(如串行性)进行形式化描述。

为了定义某一观察的完全抽象,我们先证明观察如何诱导出一个项的等价关系。这需要一个上下文的概念。直观上,上下文是一个带"空位"[ ]的项 $C[\ ]$,在[ ]中我们可以插入有类型的项 t,来产生一个有类型的项 $C[t]$;形式地讲,上下文可以定义为一个用不同的用于代入的自由变量的项。对应于某个观察的选择,对相同类型的项 t_1 和 t_2,记 $t_1 \sim t_2$ 当且仅当对所有的上下文 $C[\ ]$,$C[t_1]$和 $C[t_2]$是封闭的有类型的项,并且 $C[t_1]$和 $C[t_2]$的观察相同。例如,如果观察只涉及项的收敛性,则对上下文 $C[\ ]$, $C[t_1]$和 $C[t_2]$是封闭的且有类型的,我们有:

$$t_1 \sim t_2 \text{当且仅当} (C[t_1]\downarrow \Longleftrightarrow C[t_2]\downarrow)$$

注意,尽管这里利用了操作语义定义等价关系 ~,但也可利用指称语义定义,只要它是适用的。我们说指称语义是相对于观察完全抽象的,当且仅当

$$[\![t_1]\!] = [\![t_2]\!] \text{ 当且仅当 } t_1 \sim t_2$$

实际上,只要指称语义是适用的,则"⇒"方向成立(请读者思考为什么?),因此困难在于如何证明"⇐"方向。 218

因此,在某种意义下,完全抽象的语义只区别了观察中强制的不同。但完全抽象很难实现,特别是,对应于收敛的观察,我们提出的活性指称语义或惰性指称语义,不是完全抽象的(考察练习 11.28中的对于类型 **int** 的求值观察的指称语义)。

现在我们考虑为什么高阶类型语言的完全抽象促进对高阶类型的串行性研究。获得完全抽象的困难是因为即使项 t_1, t_2 有不同的指称,也不能用程序设计语言定义的上下文区分开来。为什么呢? 因为在指称的完全偏序中,存在类似"并行或"[⊖]这样的元素,它不能用不同的项 t_1, t_2 定义。项有串行的特点不能被这些"并行的"元素共享。因此有一个办法:为了实现完全抽象,重新定义完全偏序上的构造以防止出现无定义的元素,特别是,把函数空间中所有连续函数限制为"串行的"函数。如果不考虑完全偏序构造中操作语义某些编码形式,那么至少用语法无关的方式很难做到这一点。完全抽象引起了对串行性一般定义的研究。但即使获得成功对一些串行性进行操作分析,也不可能自动解决完全抽象问题。

11.11 和

我们考虑如何将语言扩展使其包含类型的和。在类型 τ_1 和 τ_2 之间增加一个构造 $\tau_1 + \tau_2$。

⊖ "并行或"是 $\mathbf{T}_\perp$ 上的连续函数 *por*,扩展了在真值域上析取,具有性质 *por*(**ture**, ⊥) = *por*(⊥, **true**) = **true**; 看上去函数对每个参数的检查是并行的,而不是顺序的。如果有任一参数为 **true**,那么函数返回 **true**。

相应地,扩展项的语言使其包含项的内射,函数 **inl**(t),**inr**(t)分别内射到和的左边部分和右边部分。和函数可以用 *case* 构造描述

$$\textbf{case } t \textbf{ of inl}(x_1).\, t_1,\ \textbf{inr}(x_2).\, t_2$$

t_1 中 x_1 的自由出现和 t_2 中 x_2 的自由出现在这个新构造中是约束出现的,新构造的自由变量为

219 $$\mathrm{FV}(\textbf{case } t \textbf{ of inl}(x_1).\, t_1, \textbf{inr}(x_2).\, t_2) = \mathrm{FV}(t) \cup (\mathrm{FV}(t_1) \backslash \{x_1\}) \cup (\mathrm{FV}(t_2) \backslash \{x_2\})$$

非形式地讲,这样的 *case* 构造检查 t 的形式,根据它出现在和的左部还是和的右部进行求值。下面是附加的类型规则以确保项的合式定义:

$$\frac{t : \tau_1}{\textbf{inl}(t) : \tau_1 + \tau_2} \qquad \frac{t : \tau_2}{\textbf{inr}(t) : \tau_1 + \tau_2}$$

$$\frac{t : \tau_1 + \tau_2 \quad x_1 : \tau_1 \quad x_2 : \tau_2 \quad t_1 : \tau \quad t_2 : \tau}{\textbf{case } t \textbf{ of inl}(x_1).\, t_1, \textbf{inr}(x_2).\, t_2 : \tau}$$

注意,由于内射的类型规则,项能具有多于一种类型,例如,

$$\textbf{inl}(5) : \textbf{int} + \textbf{int} \quad \text{且} \quad \textbf{inl}(5) : \textbf{int} + (\textbf{int} \rightarrow \textbf{int})$$

含有和的项的计算取决于它是活性计算还是惰性计算。在活性计算的操作语义中,我们说像 **inl**(t)这样的内射是一个标准型当且仅当 t 本身是标准型。我们通过以下语句在活性求值中定义和类型的标准型:

$$\textbf{inl}(c) \in C^e_{\tau_1+\tau_2}, \text{如果 } c \in C^e_{\tau_1}, \quad \textbf{inr}(c) \in C^e_{\tau_1+\tau_2}, \text{如果 } c \in C^e_{\tau_2}$$

这样的标准型又求值到本身。操作语义的规则可增加:

$$\frac{t \rightarrow^e \textbf{inl}\ (c_1) \quad t_1[c_1/x_1] \rightarrow^e c}{(\textbf{case } t \textbf{ of inl}(x_1).\, t_1, \textbf{inr}(x_2).\, t_2) \rightarrow^e c} \qquad \frac{t \rightarrow^e \textbf{inr}\ (c_2) \quad t_2[c_2/x_2] \rightarrow^e c}{(\textbf{case } t \textbf{ of inl}(x_1).\, t_1, \textbf{inr}(x_2).\, t_2) \rightarrow^e c}$$

对活性求值的指称语义,和类型的值的完全偏序恰好是每个分量值的完全偏序的和;即

$$V^e_{\tau_1+\tau_2} = V^e_{\tau_1} + V^e_{\tau_2}$$

如前所述,环境 ρ 下的项 t 的指称是$(V^e_\tau)_\perp$的一个元素。但是,和的扩展意味着项 t 不必具有惟一的类型,因为随着和的分量的变化,内射函数可以有不同的表示形式,所以,项 t 的指称应指明类型 $t : \tau$。

$$[t : \tau]^e \rho \in (V^e_\tau)_\perp$$

对惰性情况,标准型可以是本身没有求值过的封闭项的内射。这样,惰性语言的标准型包含了由下述语句给出的和的标准型:

$$\textbf{inl}(t) \in C^l_{\tau_1+\tau_2}, \text{如果 } t : \tau_1 \text{ 和 } t \text{ 是封闭的}$$

220 $$\textbf{inr}(t) \in C^l_{\tau_1+\tau_2}, \text{如果 } t : \tau_2 \text{ 和 } t \text{ 是封闭的}$$

case 结构的惰性求值规则为

$$\frac{t \rightarrow^l \mathbf{inl}(t') \quad t_1[t'/x_1] \rightarrow^l c}{(\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).\ t_1, \mathbf{inr}(x_2).\ t_2) \rightarrow^l c} \qquad \frac{t \rightarrow^l \mathbf{inr}(t') \quad t_2[t'/x_2] \rightarrow^l c}{(\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).\ t_1, \mathbf{inr}(x_2).\ t_2) \rightarrow^l c}$$

因为和类型的值不需要其分量的预先求值,所以扩展的指称语义基于值的选择,因此

$$V^l_{\tau_1+\tau_2} = (V^l_{\tau_1})_\perp + (V^l_{\tau_2})_\perp$$

在环境 ρ 下,有类型项 $t: \tau$ 的指称语义表示为:

$$[\![t: \tau]\!]^l\rho \in (V^l_\tau)_\perp$$

不难将本章的结论扩展到带有和类型的语言中。

练习 11.29 对活性和惰性两种求值,对应于类型

$$\mathbf{inl}(t): \tau_1 + \tau_2$$
$$(\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).\ t_1, \mathbf{inr}(x_2).\ t_2): \tau$$

试写出内射和 *case* 构造的指称语义。作为检验你的指称语义是否正确,对封闭项 t,试用规则归纳法证明(你只需考虑新的情况):对 τ_1 的标准型 c 以及任一环境 ρ,

$$t \rightarrow^e c \Rightarrow [\![t: \tau]\!]^e\rho = [\![c: \tau]\!]^e\rho \text{ 且}$$
$$t \rightarrow^l c \Rightarrow [\![t: \tau]\!]^l\rho = [\![c: \tau]\!]^l\rho$$

□

11.12 进一步阅读资料

推荐三本函数式程序设计语言的优秀著作:(活性)标准 ML[101]和[73];(惰性)[22]。有关逻辑关系及其历史和使用的总结可参考[65]。完全抽象的两篇经典论文是 Plotkin 的[78]和 Minler 的[62]。这两篇论文都考虑把完全抽象限制在基类型整型和布尔类型的项的观察上。Plotkin 证明了不仅通过去掉无定义的元素,而且还通过扩展包含并行条件的语言以获得完全抽象。[94]和[16]中介绍了语言的完全抽象问题最新进展情况,[16]写于十年前,但仍是一本好的总结性书籍。[27]是最近的论文。[3]中讨论了语言以及直觉逻辑和线性逻辑之间的关系。 221

第12章 信息系统

信息系统提供了重要的一类被称为斯科特域的完全偏序的表示方法。本章介绍信息系统，并且阐述如何使用信息系统去找出递归域方程的最小解，这有助于理解递归类型。该方法基于信息系统之间的子构造关系。实质上，可以把信息系统作为含底的完全偏序。像积、和以及（提升）函数空间等有用的构造能构造这个完全偏序上的连续函数，所以，递归域方程的解归结为形成连续函数最小不动点的熟悉的构造。使用信息系统比直接用域论更有技术优势。完全偏序的性质能够推导而不是假设，而且更有助于数学表示。特别是下一章中我们能得到证明类似于递归类型语言的操作语义和指称语义之间对应关系的性质的基本方法。

12.1 递归类型

首先，我们看一下满足递归域方程的一个熟悉的完全偏序。方程

$$X = \mathbf{1} + X$$

理解为完全偏序（或域）X 等于它们本身加上单位元的完全偏序 $\mathbf{1}$。这是关于 X 的递归方程。尽管它不止有一个解，但其中一个解是自然数 ω 的离散完全偏序。许多程序设计语言允许递归类型的定义（下一章介绍这类语言）。即使这些语言不允许递归类型的定义，通常也使用递归定义的完全偏序来直接描述其语义。程序设计的动态约束特点用递归定义类型也可以方便建模。事实上，斯科特的重要突破是发现了求递归类型定义

$$D \cong [D \to D]$$

的一般解（即不是单元素）的 λ 演算模型。严格地讲，这不是方程，而是完全偏序 D 和 $[D \to D]$ 之间的同构。我们可以从中得到启发，不必解等式而应考虑它们之间的同构关系。

我们如何递归地定义类型呢？我们通过一个简单的例子去说明归纳定义的某些机制。有限整数表（10.5 节讨论过）用一个集合 L 来表示，它满足

$$L = \{()\} + (\mathbf{N} \times L) \qquad (*)$$

223

空元组表示空表，而序偶 (n,l) 表示把整数 n 插到表 l 头部的添加操作。有限整数表不是 $(*)$ 的惟一解。如果表 L 由有限整数表和无限整数表组成，那么这个表也是 $(*)$ 的解。但是，把有限整数表 L 作为元素也产生满足 $(*)$ 的最小解。有限表的定义适合第 4 章中讨论的归纳定义模式。L 是 $(*)$ 的一个解明确要求 L 包含 $()$ 并且 L 对 *cons* 操作是封闭的，即 L 对下列规则是封闭的：

$$\emptyset/(), \quad \{l\}/(n,l)$$

其中 $n \in \mathbf{N}$。有限表的集合是对这些规则封闭的最小集。另外，我们也可以把有限表的集合

定义为$fix(\psi)$,其中ψ是作用于集合上的单调且连续的算子,定义为

$$\psi(X) = \{()\} + (\mathbf{N} \times X)$$

有相当多的递归类型可以以使用归纳定义的类似的方法构造出来。

练习 12.1 试描述如何将具有整数叶子的二叉树类型定义为一个归纳定义。 □

练习 12.2 试描述一个集合,它是域方程$X = \mathbf{1} + X$的一个解,但它不与自然数集合ω同构。 □

然而,存在其他的一些不能直接用同样的方法来定义的递归类型。例如,我们如何定义8.2节中无限的流类型或"带终止符的序列"呢?我们猜想流是完全偏序之间的方程

$$L = (\{\$\} + \mathbf{N} \times L)_\perp$$

的最小解。尽管可行,但任何满足这个方程的完全偏序L必须首先包含$\perp$,即终止符$\lfloor \$ \rfloor$,然后包含像$\lfloor (n, \perp) \rfloor$和$\lfloor (n, \lfloor \$ \rfloor) \rfloor$这样的序列,其中$n \in \mathbf{N}$。继续构造下去,我们可以论证$L$含有形如

$$\lfloor (n_1, \lfloor (n_2 \cdots, \lfloor \$ \rfloor) \cdots) \rfloor) \rfloor \quad 和 \quad \lfloor (n_1, \lfloor (n_2 \cdots, \perp) \cdots) \rfloor$$

的序列,其中$n_1, n_2, \cdots$是整数。换句话说,L包含所有的有限"带终止符"的序列和"不带终止符"的序列。但无论是这种风格的论证还是归纳定义,都不能产生诸如

224

$$(n_1, \lfloor (n_2, \lfloor (n_3 \cdots) \cdots)$$

的无限序列。这个限制给如何定义这样的递归类型提供了一些线索:首先,用归纳定义的方法去构造有限元素,然后用某种完备化过程去推导出无限元素,即通过有限逼近的方式来构造出无限元素。

*信息系统*表示如何从有限元素符号中构造出的完全偏序。因为它们仅显式处理有限元素,所以信息系统遵从归纳定义技术,因此,人们能递归地定义信息系统。信息系统可以看成是如何构造完全偏序的规定。更详细地说,一个信息系统可以认为由与继承关系以及一致性关系有关的描述计算的断言或命题所组成的。信息系统确定了一个完全偏序,其元素是那些一致的并且对推导关系是封闭的符号(token)的集合;次序关系就是包含关系。完全偏序中的元素可以认为是关于可能计算的一组真值,其本身是逻辑上封闭的和一致的一组断言。虽然不是所有的完全偏序都可以用信息系统表示,但信息系统确实可以表示一大类完全偏序,即斯科特域。

现在我们应该注意到,我们不能期望解所有的域方程,因为我们的完全偏序未必含底元素。尤其是根据康托尔方法,我们不能期望得到域方程

$$X \cong [X \to 2]$$

的解,其中2是两个元素的离散完全偏序。我们在域方程中只允许"提升函数空间"构造;对两个完全偏序D, E,其提升函数空间是$[D \to E_\perp]$。本章的方法将给出任一域方程

$$X \cong F(X)$$

的最小解,其中 F 是由单位域 **1**（恰好只有一个元素）和空域 $\emptyset$ 上使用积、提升函数空间、提升以及和而构成的。

12.2 信息系统定义

信息系统由一组符号组成,这些符号是与一致性关系和推导关系有关的对之计算的断言或命题。一致性关系识别出那些计算都为真的有限符号子集。例如,一个整数不可能同时计算为 3 和 5,因此,断言这两个输出的符号是不一致的。有限的一组符号的真假推导出另外一组符号的真假。例如,两个符号推导出第三个符号,如果第三个符号表示这两个符号的合取。 225

记号 $X \subseteq^{fin} A$ 表示 X 是集合 A 的有限子集。$Fin(A)$ 表示由 A 的所有有限子集组成的集合,即 $Fin(A) = \{X \mid X \subseteq^{fin} A\}$。

定义 信息系统定义为一个结构 $\mathscr{A} = (A, \mathrm{Con}, \vdash)$,其中 A 是可数集（符号),Con（一致性集合）是 $Fin(A)$ 的非空子集,$\vdash$（推导关系）是 $(\mathrm{Con} \setminus \{\emptyset\}) \times A$ 的满足如下 5 个公理的子集:

1. $X \subseteq Y \in \mathrm{Con} \Rightarrow X \in \mathrm{Con}$
2. $a \in A \Rightarrow \{a\} \in \mathrm{Con}$
3. $X \vdash a \Rightarrow X \cup \{a\} \in \mathrm{Con}$
4. $X \in \mathrm{Con}\ \&\ a \in X \Rightarrow X \vdash a$
5. $(X, Y \in \mathrm{Con}\ \&\ \forall b \in Y. X \vdash b\ \&\ Y \vdash c) \Rightarrow X \vdash c$

条件 $\vdash \subseteq (\mathrm{Con} \setminus \{\emptyset\}) \times A$ 等价于 $\emptyset \vdash a$ 永不成立,即空集什么都不推导。一般的逻辑系统与公理 1 ~ 5 相比,这些公理假设有更明确的定义。它的假设确实简化了诸如和这样的构造,这有助于我们以后的工作。习惯上,一般文献中信息系统导致含底元素的完全偏序。这里为了表示不含底的完全偏序,稍微修改一下一般的定义。

信息系统确定了称为*元素*的一族符号的子集。把符号看作关于计算的断言——假设一旦计算为真,符号保持为真。直观上说,信息系统的元素是如实论述计算的符号的集合。这组符号可以作为计算的信息内容,符号本身不应相互矛盾（它们应保持一致）并且在推导关系下是封闭的。为了表示未必含有底元素的完全偏序,我们规定元素是非空的——这样把空集排除在外。

定义 信息系统 $\mathscr{A} = (A, \mathrm{Con}, \vdash)$ 的*元素* $|\mathscr{A}|$ 是 A 的满足下列性质的子集 x,它们是

1. 非空的:$x \neq \emptyset$。
2. 一致的:$X \subseteq^{fin} x \Rightarrow X \in \mathrm{Con}$。
3. $\vdash$封闭的:$X \subseteq x\ \&\ X \vdash a \Rightarrow a \in x$。 226

于是,信息系统确定了一族集合。正如我们在下一节看到的,这族集合有一个简单的性质。这族集合按包含关系排序形成了完全偏序。注意,空集 $\emptyset$ 是一致的和 $\vdash$封闭的,所以空集是最小元素但它不能是非空集合。因为空集排除在外,所以,完全偏序中不必含有底元素。

命题 12.3 按包含关系排序的信息系统的元素构成了完全偏序。

证明 设 $\mathscr{A}=(A,\mathrm{Con},\vdash)$ 是信息系统。我们要证明 $|\mathscr{A}|$ 是一个完全偏序。假设 $x_0\subseteq\cdots\subseteq x_n\subseteq\cdots$ 是 $|\mathscr{A}|$ 中的 ω 链，我们来证明 $\bigcup_n x_n\in|\mathscr{A}|$。首先因为 $\bigcup_n x_n$ 的任一元素非空，所以 $\bigcup_n x_n$ 非空。其次，$\bigcup_n x_n$ 是一致的。假设 $X\subseteq^{fin}\bigcup_n x_n$，于是，因为 X 是有限的，对某一 $n\in\omega$，有 $X\subseteq x_n$，所以 $X\in\mathrm{Con}$。第三，$\bigcup_n x_n$ 是 $\vdash$ 封闭的。假设 $X\in\mathrm{Con}, X\vdash a$, $X\subseteq\bigcup_n x_n$。于是因为 X 是有限的，所以对某个 n，有 $X\subseteq x_n$。然而，$x_n\in|\mathscr{A}|$，所以 $a\in x_n$。于是 $a\in\bigcup_n x_n$。因此，$|\mathscr{A}|$ 是 ω 链的并，且 $|\mathscr{A}|$ 是一个完全偏序。 □

因此，信息系统确定一个完全偏序。现在对斯科特在域论中介绍的信息系统的精妙思想作直观的解释。通过把信息系统表示为完全偏序，我们把和计算相关的信息看成一组符号，把增加信息看成在集合中增加为真的符号。

尽管从信息系统获得的那些完全偏序形成了重要的一大类完全偏序，但不是所有的完全偏序都能作为信息系统的元素。下一节我们讨论信息系统的结构，从有限非空的一致的集合中推导出作为闭包的元素将起着特殊的作用。

引理 12.4 设 $\mathscr{A}=(A,\mathrm{Con},\vdash)$ 是信息系统。假设 $\emptyset\neq X\in\mathrm{Con}$ 且 Y 是 A 的有限子集。

1. 对任意 $b\in Y$，如果 $X\vdash b$，则 $X\cup Y\in\mathrm{Con}$ 并且 $Y\in\mathrm{Con}$。
2. 集合 $\overline{X}=\{a\in A\mid X\vdash a\}$ 是 $\mathscr{A}$ 的一个元素。

证明

(1) 对任意 $b\in Y$，设 $X\vdash b$，施简单归纳于 Y 的大小去证明：$X\cup Y\in\mathrm{Con}$ 且 $Y\in\mathrm{Con}$。当 Y 为空时，显然成立。假设 Y 非空，Y 含有符号 b'，并且对所有 $b\in Y$，$X\vdash b$。于是对所有 $b\in Y\setminus\{b'\}$，有 $X\vdash b$，根据归纳，$X\cup(Y\setminus\{b'\})\in\mathrm{Con}$。由信息系统公理 4 和 5 得 $X\cup(Y\setminus\{b'\})\vdash b'$。由公理 3 得 $X\cup Y\in\mathrm{Con}$。由公理 1 得 $Y\in\mathrm{Con}$。

227

(2) 由(1)得，$\overline{X}=\{a\mid X\vdash a\}$ 是一致的。同时 $\overline{X}=\{a\mid X\vdash a\}$ 是 $\vdash$ 封闭的，因为如果 $Y\subseteq\{a\mid X\vdash a\}$ 且 $Y\vdash a'$，则由定义信息系统的公理 5，得 $X\vdash a'$。 □

记号 一致性集合和符号之间的推导关系以明显的方式可扩展到一致性集合之间的推导关系。设 $\mathscr{A}=(A,\mathrm{Con},\vdash)$ 是信息系统。X,Y 在 Con_A 中。用 $X\vdash^* Y$ 表示 $\forall a\in Y.\ X\vdash a$。使用这个记号我们看到

$$\emptyset\vdash^* Y\Longleftrightarrow Y=\emptyset$$

最早推导 $\vdash$ 的推论是 $(\mathrm{Con}\setminus\{\emptyset\})\times A$ 的子集。从 $\vdash^*$ 的定义，我们直接得到

$$X\vdash^* Y\ \&\ X\vdash^* Y'\Rightarrow X\vdash^*(Y\cup Y')$$

我们可以用

$$X\vdash^* Y\ \&\ Y\vdash^* Z\Rightarrow X\vdash^* Z$$

重写信息系统的公理 5，这样可以清楚地知道，公理 5 表达了推导关系的传递性。

对信息系统符号的任一子集 X，记

$$\overline{X}=_{def}\{a\mid\exists Z\subseteq X.\ Z\vdash a\}$$

注意，$\overline{\emptyset} = \emptyset$ 因为只有对非空集合 X 才有 $X \vdash b$ 成立。

12.3 闭族与斯科特前域

本节描述信息系统元素形成的完全偏序的性质。本节不影响以后本书的阅读，所以可以略过。但它引入了现在广泛使用的概念斯科特域。首先，我们精确地描述这族作为信息系统元素的子集的性质。

定义 集合的闭族是可数集合的子集的集合 $\mathscr{F}$，它满足：

1. 如果 $x \in \mathscr{F}$，则 $x \neq \emptyset$。
2. 如果 $x_0 \subseteq x_1 \subseteq \cdots \subseteq x_n \subseteq \cdots$ 是 $\mathscr{F}$ 中的 ω 链，则 $\bigcup_{n \in \omega} x_n \in \mathscr{F}$。
3. 如果 U 是 $\mathscr{F}$ 的非空子集，且 $\cap U \neq \emptyset$，则 $\cap U \in \mathscr{F}$。 228

现在讨论信息系统和闭族之间一一对应关系。

定理 12.5

(i) 设 $\mathscr{A}$ 是信息系统，则 $|\mathscr{A}|$ 是集合的闭族。

(ii) 设 $\mathscr{F}$ 是集合的闭族，定义

$$A_{\mathscr{F}} = \bigcup \mathscr{F}$$

$$X \in \mathrm{Con}_{\mathscr{F}} \Longleftrightarrow X = \emptyset \text{ 或 } (\exists x \in \mathscr{F}. X \subseteq^{fin} x)$$

$$X \vdash_{\mathscr{F}} a \Longleftrightarrow \emptyset \neq X \in \mathrm{Con}_{\mathscr{F}} \,\&\, a \in A_{\mathscr{F}} \,\&\, (\forall x \in \mathscr{F}. X \subseteq x \Rightarrow a \in x)$$

则 $\mathcal{I}(\mathscr{F}) = (A_{\mathscr{F}}, \mathrm{Con}_{\mathscr{F}}, \vdash_{\mathscr{F}})$ 是一个信息系统。

(iii) 映射 $\mathscr{A} \mapsto |\mathscr{A}|$ 和 $\mathscr{F} \mapsto \mathcal{I}(\mathscr{F})$ 是给定的信息系统和闭族之间一一对应的互逆关系：如果 $\mathscr{A}$ 是信息系统，则 $\mathcal{I}(|\mathscr{A}|) = \mathscr{A}$；如果 $\mathscr{F}$ 是闭族，则 $|\mathcal{I}(\mathscr{F})| = \mathscr{F}$。

证明

(i) 设 $\mathscr{A} = (A, \mathrm{Con}, \vdash)$ 是信息系统。我们要证明 $|\mathscr{A}|$ 是一闭族。命题 12.3 可以证明 $|\mathscr{A}|$ 满足闭族的前两个条件。假设 $\emptyset \neq U \subseteq |\mathscr{A}|$ 且 $\cap U$ 非空，我们来证明 $\cap U \in |\mathscr{A}|$。取 $u \in U$，因为 $\cap U \subseteq u$，所以 $\cap U$ 是一致的。假设 $X \subseteq \cap U$ 且 $X \vdash a$，则对所有 $u \in U$，有 $X \subseteq u$。因为每一个 $u \in U$ 是 $\vdash$ 封闭的，所有 $a \in U$。于是，$a \in \cap U$。因此，$\cap U$ 是非空的、一致的并且 $\vdash$ 封闭的，所以，$\cap U \in |\mathscr{A}|$，这样就证明了 $|\mathscr{A}|$ 是集合的闭族。

(ii) 设 $\mathscr{F}$ 是闭族，则 $\mathcal{I}(\mathscr{F})$ 是信息系统。其证明留给读者。

(iii) 设 $\mathscr{A} = (A, \mathrm{Con}, \vdash)$ 是信息系统。为了证明 $\mathcal{I}(|\mathscr{A}|) = \mathscr{A}$，我们需要证明

$$A = \cup\, |\mathscr{A}|$$

$$X \in \mathrm{Con} \Longleftrightarrow X = \emptyset \vee (\exists x \in |\mathscr{A}|. X \subseteq^{fin} x)$$

$$X \vdash a \Longleftrightarrow \emptyset \neq X \in \mathrm{Con} \,\&\, a \in A \,\&\, (\forall x \in |\mathscr{A}|. X \subseteq x \Rightarrow a \in x)$$

很明显，由信息系统的公理 2，得 $A = \cup\, |\mathscr{A}|$。

设 $X \subseteq^{fin} A$。如果 $X \in \mathrm{Con}$，那么或者 $X = \emptyset$，或者 $X \subseteq \overline{X} = \{a \mid X \vdash a\} \in |\mathscr{A}|$。反过来，如果 $X = \emptyset$ 或 $X \subseteq^{fin} x$，其中 $x \in |\mathscr{A}|$，那么，由这样的元素 x 的定义，我们必然有 $X \in \mathrm{Con}$。

假设 $X \in \mathrm{Con}$ 并且 $a \in A$。很清楚，如果 $X \vdash a$，那么，由 $\mathscr{A}$ 的元素的定义，对任意

$x \in |\mathscr{A}|$，我们必然有 $X \subseteq x \Rightarrow a \in x$。假设（$\forall x \in |\mathscr{A}|. X \subseteq x \Rightarrow a \in x$），则 $\bar{X} = \{b \mid X \vdash b\} \in |\mathscr{A}|$，所以 $X \vdash a$。因而 $\mathcal{I}(|\mathscr{A}|) = \mathscr{A}$。

229

设 $\mathscr{F}$ 是闭族。我们要证明 $|\mathcal{I}(\mathscr{F})| = \mathscr{F}$。如果 $x \in \mathscr{F}$，则直接由 $\mathcal{I}(\mathscr{F})$ 的一致性和推导关系的定义得到 $x \in |\mathcal{I}(\mathscr{F})|$。所以 $\mathscr{F} \subseteq |\mathcal{I}(\mathscr{F})|$。现在我们来证明 $|\mathcal{I}(\mathscr{F})| \subseteq \mathscr{F}$。如上所述，记 $\mathcal{I}(\mathscr{F}) = (A_{\mathscr{F}}, \mathrm{Con}_{\mathscr{F}}, \vdash_{\mathscr{F}})$，假设 $\emptyset \neq X \in \mathrm{Con}_{\mathscr{F}}$，则由 $\mathrm{Con}_{\mathscr{F}}$ 的定义，可得 $U = \{y \in \mathscr{F} \mid X \subseteq y\}$ 是 $\mathscr{F}$ 的非空子集，而且，由 $\vdash_{\mathscr{F}}$ 的定义，我们得到 $\bar{X} = \cap U$。因为 $\mathscr{F}$ 是闭族且 $\cap U$ 非空，所以 $\bar{X} \in \mathscr{F}$。为了完成证明，设 $x \in |\mathcal{I}(\mathscr{F})|$。假设可能为 $\cup \mathscr{F}$ 的集合 x 的元素的一个特殊的可数枚举为

$$e_0, e_1, \cdots, e_n, \cdots$$

因此，x 是可数集合。对 $n \in \omega$，现在 x_n 形成了 $\mathscr{F}$ 中的一条 ω 链，其中我们定义 $x_n = \bar{X}_n$，其中：

$$X_0 = \{e_0\}$$
$$X_{n+1} = X_n \cup \{e_{n+1}\}$$

由于 $\mathscr{F}$ 是闭族，$\cup_n x_n \in \mathscr{F}$，显然有 $\cup_n x_n = x$。于是，$|\mathcal{I}(\mathscr{F})| \subseteq \mathscr{F}$。从而，我们最后得到 $|\mathcal{I}(\mathscr{F})| = \mathscr{F}$。

对所有信息系统 $\mathscr{A}$，有 $\mathcal{I}(|\mathscr{A}|) = \mathscr{A}$；对所有的闭族 $\mathscr{F}$，有 $|\mathcal{I}(\mathscr{F})| = \mathscr{F}$，这表明信息系统和闭族之间有一一对应关系。 □

练习 12.6 证明上述(ii)。 □

现在我们考虑信息系统表示的一类完全偏序。实际上，含底的完全偏序恰好是一类著名的完全偏序，称为斯科特域（以 Dana Scott 的姓氏命名）。

定义 完全偏序 D 的元素 x 是有限的，当且仅当对每一条满足 $x \sqsubseteq \bigsqcup_{n \in \omega} d_n$ 的 ω 链 $d_0 \sqsubseteq \cdots \sqsubseteq d_n \cdots$，存在 $n \in \omega$，有 $x \sqsubseteq d_n$。我们用 D^0 表示 D 的有限元素的集合。

完全偏序 D 是 ω 代数的当且仅当有限元素集合 D^0 是可数的，并且对任意 $x \in D$，存在有限元素的 ω 链 $e_0 \sqsubseteq \cdots \sqsubseteq e_n \cdots$，使得 $x = \bigsqcup_{n \in \omega} e_n$。

如果 D 中存在 X 的上界，则我们称对于完全偏序 D 的子集 X，X 是有界的。如果 D 的每一个非空有界子集都有最小上界，那么称 D 是完全有界的。

如果完全偏序是含底元素的、完全有界的和 ω 代数的，则经常称其为斯科特域。通常，当它不含底元素时，我们将称完全有界的、ω 代数的完全偏序为斯科特前域。

230

练习 12.7 试证明在斯科特前域中，有限元素的有限集合的最小上界如果存在，则也是有限的。 □

命题 12.8 设 $\mathscr{A} = (A, \mathrm{Con}, \vdash)$ 是一个信息系统，它的元素 $|\mathscr{A}|$ 按包含关系排序形成一个斯科特前域。它的有限元素是形为 $\bar{X} = \{a \in A \mid X \vdash a\}$ 的元素，其中 $\emptyset \neq X \in \mathrm{Con}$。

证明 设 $\mathscr{A} = (A, \mathrm{Con}, \vdash)$ 是具有元素 $|\mathscr{A}|$ 的信息系统。因为 $|\mathscr{A}|$ 是闭族，所以 $|\mathscr{A}|$ 是按包含关系排序的完全偏序。

我们要证明 $|\mathscr{A}|$ 是完全有界的，即对非空集合 $V \subseteq |\mathscr{A}|$ 和 $y \in |\mathscr{A}|$，如果 $\forall x \in V. x \subseteq y$，则 $|\mathscr{A}|$ 中存在一个最小上界 V。然而，如果 $\forall x \in V. x \subseteq y$，则 $U = \{y \mid \forall x \in V. x \subseteq y\}$ 是闭

族 $|\mathscr{A}|$ 的非空子集。因为 V 是非空的,所以 V 含有 $|\mathscr{A}|$ 的非空元素 v。由于 $v \subseteq \cap U$,这就确保 $\cap U$ 是非空的。因此,由闭族定义的性质3,得 $\cap U \in |\mathscr{A}|$,$\cap U$ 显然是 V 的最小上界。

我们现在证明按包含关系排序的 $|\mathscr{A}|$ 是代数完全偏序。首先,我们观察一个关于 $|\mathscr{A}|$ 的所有元素的一个事实。设 $x \in |\mathscr{A}|$,假设 A 是可数集,则可以取 x 的可数枚举的 a_0, $a_1, \cdots, a_n, \cdots$。如前所述,定义 $x_n = \overline{X}_n$,其中

$$X_0 = \{a_0\}$$
$$X_{n+1} = X_n \cup \{a_{n+1}\}$$

于是,$x = \cup_n x_n$。我们现在进一步证明:对 $X \in \mathrm{Con}$,完全偏序 $|\mathscr{A}|$ 的有限元素确切地是 $\overline{X}$ 中的那些元素。因此,可以得到每个元素是有限元素的 ω 链的最小上界。

特别地,设 $x \in |\mathscr{A}|$ 是有限的,我们有 $x = \cup_n x_n$,如上所述,$x = \cup_n x_n$ 蕴涵着对某个 n, $x = x_n$。从而对某个 Con 中的 $X_n \subseteq^{fin} x$,有 $x = \overline{X}_n$。相反,假设对某个 $X \in \mathrm{Con}$,x 为形为 $\overline{X}$ 的一个元素。假设对完全偏序 $|\mathscr{A}|$ 的某条链 $x_0 \subseteq \cdots \subseteq x_n \subseteq \cdots$,有 $x \subseteq \cup x_n$,那么,对某个 n,有 $X \subseteq x_n$,从而也有 $x \subseteq x_n$。这就说明完全偏序 $|\mathscr{A}|$ 的有限元素确切地是 $\overline{X}$ 中的那些元素,其中 $\emptyset \neq X \in \mathrm{Con}$。

因此,我们得到 $(|\mathscr{A}|, \subseteq)$ 是完全有界的、ω 代数的完全偏序,所以是斯科特前域。 □

任意一个斯科特前域都自然地和一个信息系统相联系。直观地,我们把一个有限元素理解为在有限时间内实现计算(使用或产生)得到的信息,所以自然地把符号作为一个有限元素。于是,一致性和推导关系可以由最初始的域逐步诱导出。如果有限元素的有限集合 X 有界,则它是一致的,如果有限元素 X 的有限集合的最小上界控制一个元素,则它推导出该元素。 231

定义 设 $(D, \sqsubseteq)$ 是斯科特前域,定义 $IS(D) = (D^0, \mathrm{Con}, \vdash)$,其中 D^0 是 D 的有限元素的集合,Con 和 $\vdash$ 定义如下:

$$X \in \mathrm{Con} \Longleftrightarrow X \subseteq^{fin} D^0 \ \&\ (X = \emptyset \text{ 或 } X \text{ 有界})$$
$$X \vdash e \Longleftrightarrow \emptyset \neq X \in \mathrm{Con} \ \&\ e \sqsubseteq \bigsqcup X$$

命题 12.9 *设 D 是斯科特前域,则 $IS(D)$ 是一个信息系统,其元素是按包含关系排序的与 D 同构的完全偏序。同构序偶是:*

$$\theta: D \to |IS(D)| \text{ 由 } \theta: d \mapsto \{e \in D^0 \mid e \sqsubseteq d\} \text{ 给出}$$
$$\varphi: |IS(D)| \to D \text{ 由 } \varphi: x \mapsto \bigsqcup x \text{ 给出}$$

练习 12.10 试证明上述命题。 □

因此,一个信息系统确定了元素的斯科特前域,反之,一个前域确定了一个信息系统,其元素是一个与前域同构的完全偏序。所以我们用信息系统去表示斯科特前域。注意,如果信息系统允许含有空元素(作为完全偏序的底元素),则它们表示斯科特域。

下面的练习有一个重要的结论:任意斯科特前域的函数空间不是斯科特前域,因此不能表示为信息系统。(然而,我们在信息系统 $\mathscr{A}, \mathscr{B}$ 之间能定义一个提升函数空间构造 $\mathscr{A} \to \mathscr{B}_\perp$,它具有与 $[|\mathscr{A}| \to |\mathscr{B}|_\perp]$ 同构的完全偏序。)

练习 12.11 设 **N** 和 **T** 分别是整数和真值的（离散）斯科特前域。试证明它们的函数空间，即完全偏序[**N**→**T**]，不是斯科特前域，因此，不能表示为信息系统。

（提示：它的有限元素是什么？这些元素是否构成可数集？） □

练习 12.12 完全偏序有时可以用有向集的概念表示以代替 ω 链表示。偏序 D 的有向集是 D 的非空子集 S，满足：如果 $s,t\in S$，则存在 $u\in S$，使得 $s,t\sqsubseteq u$。有时，把完全偏序看作具有所有有向集的最小上界的偏序。在这种框架下，完全偏序的有限元素看作是满足下列条件的元素 e：对有向集 S，如果 $e\sqsubseteq\bigsqcup S$，则存在 $s\in S$，使得 $e\sqsubseteq s$。于是，ω 代数的完全偏序是指完全偏序 D：给定任意 $x\in D$，集合 $S=\{e\sqsubseteq x\mid e$ 是有限的$\}$ 是具有最小上界 x 的有向集；如果有限元素的集合是可数的，则它是 ω 代数的。试证明：这个意义上的 ω 代数完全偏序和前面的定义相同。同样，试证明：如果在闭族定义中，用

232

$$\text{“如果 } S \text{ 是}(\mathscr{F},\subseteq)\text{ 的有向子集，则 } \cup S\in\mathscr{F}\text{”}$$

去替换闭族定义的条件 2，则两个定义是一样的。 □

12.4 信息系统的完全偏序

因为我们已经研究了完全偏序的具体表示方法，所以我们可以用信息系统的完全偏序上的不动点构造来求解递归域方程。信息系统上的序关系用 $\trianglelefteq$ 表示，其直观的意义是指一个信息系统是另一个信息系统的子系统或子结构。

定义 设 $\mathscr{A}=(A,\mathrm{Con}_A,\vdash_A)$ 和 $\mathscr{B}=(B,\mathrm{Con}_B,\vdash_B)$ 是信息系统，定义 $\mathscr{A}\trianglelefteq\mathscr{B}$ 当且仅当

1. $A\subseteq B$
2. $X\in\mathrm{Con}_A\Longleftrightarrow X\subseteq A\ \&\ X\in\mathrm{Con}_B$
3. $X\vdash_A a\Longleftrightarrow X\subseteq A\ \&\ a\in A\ \&\ X\vdash_B a$

对两个信息系统 $\mathscr{A}$ 和 $\mathscr{B}$，当 $\mathscr{A}\trianglelefteq\mathscr{B}$ 时，我们称 $\mathscr{A}$ 是 $\mathscr{B}$ 的子系统。

于是，如果 $\mathscr{A}$ 的符号包含在 $\mathscr{B}$ 的符号内，并且 $\mathscr{A}$ 的一致性关系和推导关系是较大信息系统 $\mathscr{B}$ 的相应关系的一个简单限制，则称信息系统 $\mathscr{A}$ 是信息系统 $\mathscr{B}$ 的子系统。我们观察到：

命题 12.13 设 $\mathscr{A}=(A,\mathrm{Con}_A,\vdash_A)$ 和 $\mathscr{B}=(B,\mathrm{Con}_B,\vdash_B)$ 是信息系统。如果它们的符号集相等（即 $A=B$）并且 $\mathscr{A}\trianglelefteq\mathscr{B}$，则 $\mathscr{A}=\mathscr{B}$。

证明 由 $\trianglelefteq$ 的定义显然可以证明。 □

子系统的这个定义几乎给出了一个含底元素的信息系统的完全偏序。存在一个惟一的最小的信息系统，其符号为空集。按 $\trianglelefteq$ 递增的，每一条信息系统的 ω 链具有一个最小上界，它的符号、一致性关系和推导关系分别是链上各信息系统相应部分的并集。但信息系统不会形成一个集合，仅仅因为这个原因，它们不会形成完全偏序。我们可以称其形成了一个大完全偏序。这就是我们所需要的。

233

定理 12.14 关系 $\trianglelefteq$ 是一个把 $\mathbf{0}=_{def}(\emptyset,\{\emptyset\},\emptyset)$ 作为最小元素的偏序关系。而且，如果 $\mathscr{A}_0\trianglelefteq\mathscr{A}_1\trianglelefteq\cdots\trianglelefteq\mathscr{A}_i\trianglelefteq\cdots$ 是信息系统 $\mathscr{A}_i=(A_i,\mathrm{Con}_i,\vdash_i)$ 的 ω 链，则存在最小上界

$$\bigcup_i \mathscr{A}_i = (\bigcup_i A_i,\ \bigcup_i \mathrm{Con}_i,\ \bigcup_i \vdash_i)$$

（这里以及今后我们用并符号去表示信息系统的最小上界。）

证明 由定义知，$\trianglelefteq$是自反的和传递的。由命题 12.13 知，$\trianglelefteq$是反对称的。因此，$\trianglelefteq$是一个偏序关系，且 $\mathbf{0}$ 显然是$\trianglelefteq$最小信息结构。

设 $\mathscr{A}_0 \trianglelefteq \mathscr{A}_1 \trianglelefteq \cdots \trianglelefteq \mathscr{A}_i \trianglelefteq \cdots$是信息系统 $\mathscr{A}_i = (A_i, \mathrm{Con}_i, \vdash_i)$的递增的 ω 链。记 $\mathscr{A} = (A, \mathrm{Con}, \vdash) = (\cup_i A_i, \cup_i \mathrm{Con}_i, \cup_i \vdash_i)$。下面证明 $\mathscr{A}$ 是一个信息系统。

先证明 $\mathscr{A}$ 是链的上界：显然，每一个 A_i 是符号集 A 的一个子集；很明显，$\mathrm{Con}_i \subseteq \mathrm{Con}$，而反过来，如果 $X \subseteq A_i$ 且 $X \in \mathrm{Con}$，则对于某个 $j \geqslant i$，有 $X \in \mathrm{Con}_j$，但因为 $A_i \trianglelefteq A_j$，所以 $X \in \mathrm{Con}_i$；很显然，$\vdash_i \subseteq \vdash$，反过来，如果 $X \subseteq A_i$，$a \in A_i$ 且 $X \vdash a$，则对于某个 $j \geqslant i$，有 $X \vdash_j a$，但因为 $\mathscr{A}_i \trianglelefteq \mathscr{A}_j$，所以 $X \vdash_i a$。

再证明 $\mathscr{A}$ 是链的最小上界：假设 $\mathscr{B} = (B, \mathrm{Con}_B, \vdash_B)$是链的一个上界。显然，有 $A = \cup_i A_i \subseteq B$。很明显，$\mathrm{Con} = \cup_i \mathrm{Con}_i \subseteq \mathrm{Con}_B$。同样如果 $X \subseteq A$ 且 $X \in \mathrm{Con}_B$，那么，因为 X 是有限的，对某个 i，有 $X \subseteq A_i$。又因为 $\mathscr{A}_i \trianglelefteq \mathscr{B}$，所以 $X \in \mathrm{Con}_i \subseteq \mathrm{Con}$。于是，$X \in \mathrm{Con} \Longleftrightarrow X \subseteq A \ \&\ X \in \mathrm{Con}_B$，同样，$X \vdash a \Longleftrightarrow X \subseteq A \ \&\ a \in A \ \&\ X \vdash_B a$。于是，$\mathscr{A} \trianglelefteq \mathscr{B}$，从而 $\mathscr{A}$ 是链的最小上界。 □

我们考虑信息系统上的连续操作，用它们去递归地定义信息系统。我们像从前一样，信息系统不会形成集合，它不影响我们的讨论。信息系统上的操作 F 称为（对应于$\trianglelefteq$）单调的，当且仅当对所有的信息系统 $\mathscr{A}$ 和 $\mathscr{B}$ 有

$$\mathscr{A} \trianglelefteq \mathscr{B} \Rightarrow F(\mathscr{A}) \trianglelefteq F(\mathscr{B})$$

操作 F 称为（对应于$\trianglelefteq$）连续的，当且仅当它是单调的且对任意一条递增的信息系统 ω 链

$$\mathscr{A}_0 \trianglelefteq \mathscr{A}_1 \trianglelefteq \cdots \trianglelefteq \mathscr{A}_i \trianglelefteq \cdots$$

我们有

$$\bigcup_i F(\mathscr{A}_i) = F\Big(\bigcup_i \mathscr{A}_i\Big)$$

234

（因为 F 是单调的，所以$\cup_i F(\mathscr{A}_i)$存在。）正如从前在讨论完全偏序的最小不动点时那样，我们知道，信息系统上的任一连续操作 F 都有最小不动点 $fix(F)$，它由递增的 ω 链 $\mathbf{0} \trianglelefteq F(\mathbf{0}) \trianglelefteq F^2(\mathbf{0}) \trianglelefteq \cdots \trianglelefteq F^n(\mathbf{0}) \trianglelefteq \cdots$的最小上界$\cup_i F^i(\mathbf{0})$给出。

下面的引理非常有助于证明操作的连续性。通常容易证明：一元操作对应于$\trianglelefteq$是单调的，并且在符号集合上是连续的。现在我们精确地给出概念。

定义 称信息系统上一元操作 F 在符号集合上是连续的当且仅当对任何 ω 链 $\mathscr{A}_0 \trianglelefteq \mathscr{A}_1 \trianglelefteq \cdots \trianglelefteq \mathscr{A}_i \trianglelefteq \cdots$，$F(\cup_i \mathscr{A}_i)$的每一符号都是$\cup_i F(\mathscr{A}_i)$的一个符号。

引理 12.15 设 F 是信息系统上的一元操作，则 F 是连续的当且仅当 F 对应于$\trianglelefteq$是单调的且在符号集合上是连续的。

证明

“$\Rightarrow$”：显然成立。

"$\Leftarrow$":设 $\mathscr{A}_0 \trianglelefteq \mathscr{A}_1 \trianglelefteq \cdots \trianglelefteq \mathscr{A}_i \trianglelefteq \cdots$ 是信息系统的 ω 链。很清楚,因为假设 F 是单调的,所以 $\cup_i F(\mathscr{A}_i) \trianglelefteq F(\cup_i \mathscr{A}_i)$。所以由假设得到,$\cup_i F(\mathscr{A}_i)$ 的符号和 $F(\cup_i \mathscr{A}_i)$ 的符号是相同的。因此由命题 12.13 得,它们是两个相同的信息系统。 □

通常,信息系统上的操作以一组信息系统为变量,以一组信息系统为结果。但正如以前对于一般的完全偏序那样,在论证操作的单调性和连续性时,我们只需考虑一个输入,一个输出。引理 8.8 和引理 8.10 可以用很直观的方式予以推广。这意味着信息系统上的这样一个操作对应于 $\trianglelefteq$ 是连续的,当且仅当它对每一个变量分别是连续的,即将之看成其他变量固定的任意一个变量的函数。类似地,函数是连续的当且仅当函数对每一个输出都是连续的。因此,验证一个操作是否连续,就简化为验证一些一元操作对应于子系统关系 $\trianglelefteq$ 是否连续。

序 $\trianglelefteq$ 也许不是首先想到的。为什么不把信息系统的完全偏序关系建立在以下更为简单的包含次序之上呢?

$$(A, \mathrm{Con}_A, \vdash_A) \subseteq (B, \mathrm{Con}_B, \vdash_B) \text{ 当且仅当 } A \subseteq B \ \&\ \mathrm{Con}_A \subseteq \mathrm{Con}_B \ \&\ \vdash_A \subseteq \vdash_B$$

我们不这样做是因为下一节介绍的信息系统上的提升函数空间对左边的变量甚至不是单调的(见练习 12.34)。

235

练习 12.16 这个练习将信息系统间的子系统关系与集合族和完全偏序上的对应关系联系起来。设 $\mathscr{A} = (A, \mathrm{Con}_A, \vdash_A)$ 和 $\mathscr{B} = (B, \mathrm{Con}_B, \vdash_B)$ 是信息系统。

(i) 假设 $\mathscr{A} \trianglelefteq \mathscr{B}$,试证明映射 $\theta: |\mathscr{A}| \to |\mathscr{B}|$ 和 $\varphi: |\mathscr{B}| \to |\mathscr{A}| \cup \{\emptyset\}$,其中

$$\theta(x) = \{b \in B \mid \exists X \subseteq x.\, X \vdash_B b\}$$
$$\varphi(y) = y \cap A$$

对应于包含关系是连续的,并且满足:对所有的 $x \in |\mathscr{A}|$ 和 $y \in |\mathscr{B}|$,有

$$\varphi \circ \theta(x) = x \text{ 且 } \theta \circ \varphi(y) \subseteq y$$

(ii) 对信息系统 $\mathscr{A}$ 和 $\mathscr{B}$,试证明

$$\mathscr{A} \trianglelefteq \mathscr{B} \Longleftrightarrow |\mathscr{A}| = \{y \cap A \mid y \in |\mathscr{B}| \ \&\ y \cap A \neq \emptyset\}$$

(这里指出了解递归域方程的另一个方法,使用完全偏序之间的一对连续函数 $\theta: D \to E$ 和 $\varphi: E \to D_\perp$,即嵌套投影序偶,的逆极限。函数满足对所有的 $d \in D, e' \in E_\perp$,

$$\varphi \circ \theta(d) = \lfloor d \rfloor, \text{且 } \theta_\perp \circ \varphi^*(e') \sqsubseteq e'$$

请读者回忆 8.3.4 节中,$\varphi^*: E_\perp \to D_\perp$ 满足 $\varphi^*(e') = let\ e \Leftarrow e'.\ \varphi(e)$,同时 $\theta_\perp: D_\perp \to E_\perp$,定义为 $\theta_\perp(d') = let\ d \Leftarrow d'.\ \lfloor \theta(d) \rfloor$。) □

下一节我们将看到许多信息系统上操作的例子,以及我们如何用子系统关系的完全偏序来获得递归定义的信息系统的解。因为函数操作的变量不止一个信息系统,所以它能用于同时定义几个信息系统。

12.5 构造

本节我们给出积、提升函数空间、提升以及和的信息系统的构造。它们诱导完全偏序上相

应的构造。我们适当选择这些构造，使得它们关于$\trianglelefteq$也是连续的。这样，我们能产生用这些构造表示的信息系统乃至完全偏序的递归方程的解。实际上，D 域的提升 $D_\perp$ 可以从其他构造（即$[\mathbf{1}\to D_\perp]$）的同构中得到，这里我们使用定义在信息系统上的提升函数空间和空积 $\mathbf{1}$。然而，使用本节给出的更直接的定义，我们的工作变得更方便。 236

12.5.1 提升

我们的目的是定义信息系统上的提升，它反映了完全偏序上的提升。

定义 信息系统 $\mathscr{A}=(A,\mathrm{Con},\vdash)$ 的提升定义为 $\mathscr{A}_\perp=(A',\mathrm{Con}',\vdash')$，其中：

1. $A'=\mathrm{Con}$
2. $X\in\mathrm{Con}'\Longleftrightarrow X\subseteq\mathrm{Con}\ \&\ \cup X\in\mathrm{Con}$
3. $X\vdash' b\Longleftrightarrow \emptyset\neq X\in\mathrm{Con}'\ \&\ \cup X\vdash^* b$

直观上，提升函数把原来的一组符号进行扩充而包含一个符号，上述构造中的空集，甚至在没有原来的输入值时，输出为真。正如我们期望的，提升以单一的附加符号 $\emptyset$ 为元素，开始构造族。

定义 定义 $\mathbf{1}=\mathbf{0}_\perp$。

信息系统 $\mathbf{1}$ 有一个符号 $\emptyset$，一致性集合为 $\emptyset$ 和 $\{\emptyset\}$，推导关系为 $\{\emptyset\}\vdash\emptyset$，它的惟一元素为 $\{\emptyset\}$。

命题 12.17 设 $\mathscr{A}$ 是一个信息系统，则 $\mathscr{A}_\perp$ 是一个信息系统，满足

$$y\in|\mathscr{A}_\perp|\Longleftrightarrow y=\{\emptyset\}\ 或\ \exists x\in|\mathscr{A}|.\,y=\{b\mid b\subseteq^{fin}x\}$$

证明 设 $\mathscr{A}=(A,\mathrm{Con},\vdash)$ 是一个信息系统。

我们很容易检验 $\mathscr{A}_\perp=(A',\mathrm{Con}',\vdash')$ 是一个信息系统。在所有公理中，这里我们只验证公理 5 成立。假设对所有 $b\in Y$，$X\vdash' b$，$Y\vdash' c$。首先，注意到 $X\neq\emptyset$，因为如果 X 为空，则 Y 也为空，从而使 $Y\vdash' c$ 不成立。现在注意到对所有 $b\in Y$，$\cup X\vdash^* b$。因此，$\cup X\vdash^*\cup Y$。由于 $Y\vdash' c$，我们得到 $\cup Y\vdash^* c$。因此，对 $\mathscr{A}$，公理 5 成立，得 $\cup X\vdash^* c$。因为 $X\neq\emptyset$，我们得到 $X\vdash' c$。

现在我们证明

$$y\in|\mathscr{A}_\perp|\Longleftrightarrow y=\{\emptyset\}\ 或\ \exists x\in|\mathscr{A}|.\,y=\{b\mid b\subseteq^{fin}x\}$$

"$\Leftarrow$"：容易证明 $\{\emptyset\}$ 是一致的和 $\vdash'$ 封闭的；所以，如果 $y=\{\emptyset\}$，则 $y\in|\mathscr{A}_\perp|$。现在对 $x\in|\mathscr{A}|$，假设 $y=\{b\mid b\subseteq^{fin}x\}$。显然 $\emptyset\in y$，所以 $y\neq\emptyset$。假设 $X\subseteq^{fin}y$，很清楚，有 $\cup X\subseteq^{fin}x$。于是 $X\subseteq\mathrm{Con}$，$\cup X\in\mathrm{Con}$，所以 $X\in\mathrm{Con}'$。假设 $X\subseteq y$ 且 $X\vdash' b$，则 $\cup X\vdash^* b$ 且 $\cup X\subseteq x$。因此，$b\subseteq^{fin}x$，从而 $b\in y$。这样我们证明了 $y\in|\mathscr{A}_\perp|$。 237

"$\Rightarrow$"：假设 $y\in|\mathscr{A}_\perp|$ 和 $y\neq\{\emptyset\}$。取 $x=\cup y$。我们必须证明 $x\in|\mathscr{A}|$ 和 $y=\{b\mid b\subseteq^{fin}x\}$。

首先观察到：由于 y 既不为空，也不是 $\{\emptyset\}$，所以，$x\neq\emptyset$。如果 $Z\subseteq^{fin}x$，则对某个 $X\subseteq^{fin}y$ 有 $Z\subseteq\cup X$。这样就得到如果 $Z\subseteq\cup x$，则 $Z\in\mathrm{Con}$。假设 $Z\subseteq x$，$Z\vdash a$，从而 $Z\neq\emptyset$，则对某个 $X\subseteq^{fin}y$，有 $Z\subseteq\cup X$。由于 $Z\neq\emptyset$，得 $X\neq\emptyset$。从而 $X\vdash'\{a\}$，这样必然有 $\{a\}\in y$，因此，$a\in x$。

这样,我们证明了 $x \in |\mathscr{A}|$。

显然,有 $y \subseteq \{b \mid b \subseteq^{fin} x\}$。下面证明 $y \supseteq \{b \mid b \subseteq^{fin} x\}$。因为 $y \neq \emptyset$,所以存在某个 $b \in y$。由定义 $\{b\} \vdash' \emptyset$。因此$\emptyset \in y$。假设$\emptyset \neq b \subseteq^{fin} x$。那么对某个 $X \subseteq^{fin} y$,有 $b \subseteq \cup X$。因为 $b \neq \emptyset$,所以必然 $X \neq \emptyset$。显然,$\cup X \vdash^* b$。所以 $X \vdash' b$,因此 $b \in y$。这就建立了逆包含关系,即 $y \supseteq \{b \mid b \subseteq^{fin} x\}$,从而得到 $y = \{b \mid b \subseteq^{fin} x\}$。 □

现在用信息系统的提升来推出信息系统元素的完全偏序上的提升。

推论 12.18 设 $\mathscr{A}$ 是一个信息系统,则存在完全偏序的同构

$$|\mathscr{A}_\perp| \cong |\mathscr{A}|_\perp$$

其定义由下式给出:

$$x \mapsto \begin{cases} \perp, & x = \{\emptyset\} \\ \lfloor \cup x \rfloor, & \text{其他} \end{cases}$$

定理 12.19 操作 $\mathscr{A} \mapsto \mathscr{A}_\perp$ 是按 $\trianglelefteq$ 排序的信息系统上的连续操作。

证明 我们用引理 12.15 进行证明。先证明提升是单调的。假设对信息系统 $\mathscr{A} = (A, \mathrm{Con}_A, \vdash_A)$ 和 $\mathscr{B} = (B, \mathrm{Con}_B, \vdash_B)$,有 $\mathscr{A} \trianglelefteq \mathscr{B}$。记 $\mathscr{A}_\perp = (A', \mathrm{Con}'_A, \vdash'_A)$,$\mathscr{B}_\perp = (B', \mathrm{Con}'_B, \vdash'_B)$。我们要证明 $\mathscr{A}_\perp \trianglelefteq \mathscr{B}_\perp$。

显然,$A' = \mathrm{Con}_A \subseteq \mathrm{Con}_B = B'$。我们有:

$$\begin{aligned} X \in \mathrm{Con}'_A &\Longleftrightarrow X \subseteq \mathrm{Con}_A \ \&\ \bigcup X \in \mathrm{Con}_A \\ &\Longleftrightarrow X \subseteq \mathrm{Con}_A \ \&\ \bigcup X \in \mathrm{Con}_B \\ &\Longleftrightarrow X \subseteq A' \ \&\ X \in \mathrm{Con}'_B \end{aligned}$$

同样,

$$\begin{aligned} X \vdash'_A c &\Longleftrightarrow X \subseteq A' \ \&\ X \neq \emptyset \ \&\ c \in A' \ \&\ \bigcup X \vdash^*_A c \\ &\Longleftrightarrow X \subseteq A' \ \&\ X \neq \emptyset \ \&\ c \in A' \ \&\ \bigcup X \vdash^*_B c \\ &\Longleftrightarrow X \subseteq A' \ \&\ c \in A' \ \&\ X \vdash'_B c \end{aligned}$$

238

所以,$\mathscr{A}_\perp \trianglelefteq \mathscr{B}_\perp$。因此,$(-)_\perp$ 是单调的。剩下来我们要证明对符号集合,它是连续的。设 $\mathscr{A}_0 \trianglelefteq \mathscr{A}_1 \trianglelefteq \cdots \trianglelefteq \mathscr{A}_i \trianglelefteq \cdots$ 是信息系统 $\mathscr{A}_i = (A_i, \mathrm{Con}_i, \vdash_i)$ 的 ω 链。然而,$(\cup_i \mathscr{A}_i)_\perp$ 和 $\cup_i(\mathscr{A}_{i\perp})$ 的符号集合都等于 $\cup_i \mathrm{Con}_i$。所以,由引理 12.15 知,提升是按 $\trianglelefteq$ 排序的信息系统上的连续操作。 □

练习 12.20 试写出 $\mathbf{1}_{\perp_\perp}$ 和 $\mathbf{1}_{\perp_{\perp_\perp}}$ 元素的域。 □

练习 12.21 因为提升对应于 $\trianglelefteq$ 是连续的,所以它有最小不动点 $\Omega = \Omega_\perp$。试计算符号集合并且证明其元素的完全偏序 $|\Omega|$ 和以无限元素为最小上界的 ω 链组成的完全偏序是同构的。 □

练习 12.22 设 $\mathscr{A}$ 是一个信息系统。设 X 是 $\mathscr{A}_\perp$ 的一致性集合,b 是 $\mathscr{A}$ 的符号。试证

明

$$b \in \times \bar{X} \Longleftrightarrow \times U \vdash_A b \qquad \square$$

12.5.2 和

我们已经看到一些类似于空和 **0** 的特殊和构造。通常,用信息系统的和(即把两个信息系统的不相交复件并列在一起)来表示斯科特前域的和。于是,符号与一个分量或另一个分量的断言相对应。

和构造依赖于下列简单的操作。

记号 对两个集合 A 和 B,设 $A \uplus B$ 是 A 和 B 的不相交并,定义为 $A \uplus B = (\{1\} \times A) \cup (\{2\} \times B)$。内射 $inj_1 : A \to A \uplus B$ 和 $inj_2 : B \to A \uplus B$ 取为:对 $a \in A, inj_1 : a \mapsto (1,a)$ 和对 $b \in B, inj_2 : b \mapsto (2,b)$。

定义 设 $\mathscr{A} = (A, \mathrm{Con}_A, \vdash_A)$ 和 $\mathscr{B} = (B, \mathrm{Con}_B, \vdash_B)$ 是信息系统,和 $\mathscr{A} + \mathscr{B}$[⊖] 定义为 $\mathscr{C} = (C, \mathrm{Con}, \vdash)$,其中:

1. $C = A \uplus B$
2. $X \in \mathrm{Con} \Longleftrightarrow \exists Y \in \mathrm{Con}_A . X = inj_1 Y \ \vee \ \exists Y \in \mathrm{Con}_B . X = inj_2 Y$
3. $X \vdash c \Longleftrightarrow (\exists Y, a . X = inj_1 Y \ \& \ c = inj_1(a) \ \& \ Y \vdash_A a) \vee$
   $(\exists Y, b . X = inj_2 Y \ \& \ c = inj_2(b) \ \& \ Y \vdash_B b)$ 239

例 设 $\mathscr{T}$ 是和 $\mathbf{1} + \mathbf{1}$。则 $|\mathscr{T}|$ 和真值的离散完全偏序同构。$\mathscr{T}$ 的符号是 $(1,\emptyset)$ 和 $(2,\emptyset)$,正好由单元素集 $\{(1,\emptyset)\}$ 和 $\{(2,\emptyset)\}$ 构成。 □

命题 12.23 设 $\mathscr{A}$ 和 $\mathscr{B}$ 是信息系统,则和 $\mathscr{A} + \mathscr{B}$ 是一个信息系统,使得

$$x \in |\mathscr{A} + \mathscr{B}| \Longleftrightarrow (\exists y \in |\mathscr{A}| . x = inj_1 y) \text{ 或 } (\exists y \in |\mathscr{B}| . x = inj_2 y)$$

证明 我们要验证:如果 $\mathscr{A}$ 和 $\mathscr{B}$ 是信息系统,则它们的和 $\mathscr{A} + \mathscr{B}$ 也是信息系统。因为 $\mathscr{A}$ 以及 $\mathscr{B}$ 满足性质 1 到 5,所以易得 $\mathscr{A} + \mathscr{B}$ 满足性质 1 到 5。读者试证明 $\mathscr{A} + \mathscr{B}$ 的元素由 $\mathscr{A}$ 和 $\mathscr{B}$ 的元素的不相交的复件组成(留为练习)。 □

可以证明,信息系统和的元素的完全偏序与信息系统的元素的完全偏序的和同构。

推论 12.24 设 $\mathscr{A}$ 和 $\mathscr{B}$ 是信息系统,则存在一个完全偏序的同构

$$|\mathscr{A} + \mathscr{B}| \cong |\mathscr{A}| + |\mathscr{B}|$$

其定义为

$$x \mapsto \begin{cases} in_1(y), & x = inj_1 y \\ in_2(y), & x = inj_2 y \end{cases}$$

定理 12.25 操作 + 是按 $\trianglelefteq$ 排序的信息系统上的连续操作。

证明 我们要证明 + 对应于 $\trianglelefteq$ 是连续的。根据连续性的定义,我们必须证明 + 在每一个

⊖ 原文为 $\mathscr{A}_1 + \mathscr{A}_2$,疑误。——译者注

变量上分别是连续的。我们先证明 + 在第一个变量上连续。于是,根据对称性,+ 在第二个变量上也连续。

首先,我们证明 + 在第一个变量上是单调的。设 $\mathscr{A}=(A,\mathrm{Con}_A,\vdash_A)$,$\mathscr{A}'=(A',\mathrm{Con}'_A,\vdash'_A)$ 以及 $\mathscr{B}=(B,\mathrm{Con}_B,\vdash_B)$ 是满足 $\mathscr{A}\trianglelefteq\mathscr{A}'$ 的信息系统。记 $\mathscr{C}=(C,\mathrm{Con},\vdash)=\mathscr{A}+\mathscr{B}$ 以及 $\mathscr{C}'=(C',\mathrm{Con}',\vdash')=\mathscr{A}'+\mathscr{B}$。我们要证明 $\mathscr{C}\trianglelefteq\mathscr{C}'$,即

1. $C\subseteq C'$
2. $X\in\mathrm{Con}\Longleftrightarrow X\subseteq C\ \&\ X\in\mathrm{Con}'$
3. $X\vdash a\Longleftrightarrow X\subseteq C\ \&\ a\in C\ \&\ X\vdash' a$

240

1. 根据 + 的定义和假设 $\mathscr{A}\trianglelefteq\mathscr{A}'$,我们得 $C\subseteq C'$。

2. "$\Rightarrow$"。设 $X\in\mathrm{Con}$,那么对某个 $X_1\in\mathrm{Con}_A$,有 $X=\{1\}\times X_1$,或者对某个 $X_2\in\mathrm{Con}_B$,有 $X=\{2\}\times X_2$。假设 $X=\{1\}\times X_1$,因为 $\mathscr{A}\trianglelefteq\mathscr{A}'$,所以 $X\subseteq C$ 和 $X_1\in\mathrm{Con}'_A$。从而,由 + 的定义,$X\in\mathrm{Con}'$。现在我们假设 $X=\{2\}\times X_2$,其中 $X_2\in\mathrm{Con}_B$,直接由 + 的定义,得 $X\in\mathrm{Con}'$。

"$\Leftarrow$"。假设 $X\in\mathrm{Con}'$ 和 $X\subseteq C$。于是,对某个 $X_1\in\mathrm{Con}'_A$,有 $X=\{1\}\times X_1$,或者对某个 $X_2\in\mathrm{Con}_B$,有 $X=\{2^{\ominus}\}\times X_2$。在前者 $X_1\subseteq A$ 的情况下,由于 $\mathscr{A}\trianglelefteq\mathscr{A}'$,所以得 $X_1\in\mathrm{Con}_A$。同样道理,在后者的情况下,我们能得到 $X\in\mathrm{Con}$。

3. 证明和 2 类似。

这样,我们证明了 + 在第一个变量上是单调的。接下来我们要证明 + 在符号集上是连续的。设 $\mathscr{A}_0\trianglelefteq\mathscr{A}_1\trianglelefteq\cdots\trianglelefteq\mathscr{A}_i\trianglelefteq\cdots$ 是信息系统 $\mathscr{A}_i=(A_i,\mathrm{Con}_i,\vdash_i)$ 的 ω 链。$(\bigcup_i\mathscr{A}_i)+\mathscr{B}$ 的符号集是 $(\bigcup_{i\in\omega}A_i)\uplus B$,它等于 $\bigcup_i(\mathscr{A}_i+\mathscr{B})$ 的符号集 $\bigcup_{i\in\omega}(A_i\uplus B)$。

于是 + 在第一个变量上是连续的,由对称性得,+ 在第二个变量上也是连续的,所以,+ 操作是连续的。 □

例 因为 + 是连续的,所以我们可以构造最小的信息系统 $\mathscr{N}$,使得 $\mathscr{N}=\mathbf{1}+\mathscr{N}$。它的元素形成了一个和整数同构的离散完全偏序,它的符号为

$$(1,\{\emptyset\}),(2,(1,\{\emptyset\})),\cdots,(2,(2,\cdots(2,(1,\{\emptyset\}))\cdots),\cdots$$ □

12.5.3 积

完全偏序的积构造是它们元素的序偶上坐标方式的次序。理想的效果是在信息系统上通过下列方式获得的:形成符号集的积,并且如果有限集合的投影是一致的,则它们是一致的;如果一致的集合的投影推导出一个符号合适分量,那么该集合可推导出这个符号。

积构造依赖于下列简单的操作。

记号 集合 A 和 B 的积记为 $A\times B$,而投影 $proj_1:A\times B\to A$ 和 $proj_2:A\times B\to B$ 的定义为 $proj_1(a,b)=a$ 且 $proj_2(a,b)=b$。

定义 设 $\mathscr{A}=(A,\mathrm{Con}_A,\vdash_A)$ 和 $\mathscr{B}=(B,\mathrm{Con}_B,\vdash_B)$ 是信息系统,定义它们的积 $\mathscr{A}\times\mathscr{B}$ 是信息系统 $\mathscr{C}=(C,\mathrm{Con},\vdash)$,其中:

⊖ 原文为 1,疑误。——译者注

1. $C = A \times B$
2. $X \in \mathrm{Con} \Longleftrightarrow proj_1 X \in \mathrm{Con}_A \ \&\ proj_2 X \in \mathrm{Con}_B$ 241
3. $X \vdash a \Longleftrightarrow proj_1 X \vdash_A proj_1(a) \ \&\ proj_2 X \vdash_B proj_2(a)$

两个信息系统的积的元素有两个分量，每一个分量对应各自信息系统的元素。直观上讲，积 $\mathscr{A}_1 \times \mathscr{A}_2$ 的符号是关于两个相应分量的断言的序偶。

命题 12.26 设 $\mathscr{A}$ 和 $\mathscr{B}$ 是信息系统，则积 $\mathscr{A} \times \mathscr{B}$ 是一个信息系统，使得

$$x \in |\mathscr{A} \times \mathscr{B}| \Longleftrightarrow \exists x_1 \in |\mathscr{A}|, x_2 \in |\mathscr{B}|. x = x_1 \times x_2$$

证明 我们要证明两个信息系统的积是一个信息系统。

对某个 $x_1 \in |\mathscr{A}|$ 和 $x_2 \in |\mathscr{B}|$，我们去证明

$$x \in |\mathscr{A} \times \mathscr{B}| \Longleftrightarrow x = x_1 \times x_2$$

"$\Leftarrow$"：如果 $x_1 \in |\mathscr{A}|, x_2 \in |\mathscr{B}|$，直接得到它们的积 $x_1 \times x_2 \in |\mathscr{A} \times \mathscr{B}|$。

"$\Rightarrow$"：假设 $x \in |\mathscr{A} \times \mathscr{B}|$，定义 $x_1 = proj_1 x$ 和 $x_2 = proj_2 x$。那么容易证明 $x_1 \in |\mathscr{A}|$ 且 $x_2 \in |\mathscr{B}|$。显然，有 $x \subseteq x_1 \times x_2$。反过来，为了证明逆包含关系，假设 $(a,b) \in x_1 \times x_2$，则一定存在 a', b'，使得 (a,b') 和 $(a',b) \in x$。根据积中的推导的定义，我们有 $\{(a',b),(a,b')\} \vdash (a,b)$，从而得到 $(a,b) \in x$。因此 $x = x_1 \times x_2$。 □

由此得到，信息系统积的元素的完全偏序与元素的完全偏序的积同构。

推论 12.27 设 $\mathscr{A}$ 和 $\mathscr{B}$ 是信息系统，则存在一个完全偏序的同构

$$|\mathscr{A} \times \mathscr{B}| \cong |\mathscr{A}| \times |\mathscr{B}|$$

其定义 $x \mapsto (proj_1 x, proj_2 x)$ 具有互逆 $(x_1, x_2) \mapsto x_1 \times x_2$。

定理 12.28 操作 $\times$ 是按 $\trianglelefteq$ 排序的信息系统上的连续操作。

证明 我们要证明积操作对符号集是单调的和连续的。根据引理 12.15，我们知道，它对应于 $\trianglelefteq$ 是连续的。

单调性：设 $\mathscr{A} \trianglelefteq \mathscr{A}'$ 和 $\mathscr{B}$ 是信息系统。显然，$\mathscr{A} \times \mathscr{B}$ 的符号形成了 $\mathscr{A}' \times \mathscr{B}$ 的符号的子集。假设 X 是 $\mathscr{A} \times \mathscr{B}$ 的符号的子集。于是，X 在 $\mathscr{A} \times \mathscr{B}$ 中是一致的当且仅当 $proj_1 X$ 和 $proj_2 X$ 在 $\mathscr{A}$ 和 $\mathscr{B}$ 中分别是一致的。因为 $\mathscr{A} \trianglelefteq \mathscr{A}'$，这等价于 X 在 $\mathscr{A}' \times \mathscr{B}$ 中是一致的。假 242
设 X 是 $\mathscr{A} \times \mathscr{B}$ 的符号的有限集，c 是 $\mathscr{A} \times \mathscr{B}$ 的一个符号，则在 $\mathscr{A} \times \mathscr{B}$ 中 $X \vdash c$ 当且仅当 $c = (a_1, a_2)$、$proj_1 X \vdash_A a_1$ 且 $proj_2 X \vdash_B a_2$。又因为 $\mathscr{A} \trianglelefteq \mathscr{A}'$，这等价于在 $\mathscr{A}' \times \mathscr{B}$ 中 $X \vdash c$。因而，$\mathscr{A} \times \mathscr{B} \trianglelefteq \mathscr{A}' \times \mathscr{B}$，于是，$\times$ 在第一个变量上是单调的。

对符号集合的连续性：现在，设 $\mathscr{A}_0 \trianglelefteq \mathscr{A}_1 \trianglelefteq \cdots \trianglelefteq \mathscr{A}_i \trianglelefteq \cdots$ 是信息系统的 ω 链。显然，对某个 $i \in \omega$，$(\bigcup_i \mathscr{A}_i) \times \mathscr{B}$ 的符号是 $\mathscr{A}_i \times \mathscr{B}$ 中的符号，所以也是 $\bigcup_i (\mathscr{A}_i \times \mathscr{B})$ 中的符号。

由引理 12.15，$\times$ 在第一个变量上是连续的。同样，$\times$ 在第二个变量上也是连续的。所以，$\times$ 是对应于 $\trianglelefteq$ 的信息系统上的连续操作。 □

表示单元素域的信息系统 **1** 可以认为是信息系统的空积，是积构造的特殊情况。

12.5.4 提升函数空间

设 $\mathscr{A}$ 和 $\mathscr{B}$ 是信息系统，对任一 $\mathscr{B}$，表示连续函数 $|\mathscr{A}| \to |\mathscr{B}|$ 的函数空间是不可能的（参考前面的练习 12.11）。不能期望解诸如

$$X \cong [X \to 2]$$

这样的域方程，其中 2 是两个元素的离散完全偏序。但是，指称语义的函数空间大多数形如 $D \to E_\perp$，这里值域被提升。可以在任意一个信息系统上构造这个操作。

定义 设 $\mathscr{A} = (A, \mathrm{Con}_A, \vdash_A)$ 和 $\mathscr{B} = (B, \mathrm{Con}_B, \vdash_B)$ 是信息系统，它们的提升函数空间 $\mathscr{A} \to \mathscr{B}_\perp$ 是信息系统 $(C, \mathrm{Con}, \vdash)$，其中

1. $C = ((\mathrm{Con}_A \setminus \{\emptyset\}) \times \mathrm{Con}_B) \cup \{(\emptyset, \emptyset)\}$
2. $\{(X_1, Y_1), \cdots, (X_n, Y_n)\} \in \mathrm{Con} \Longleftrightarrow$
   $\forall I \subseteq \{1, \cdots, n\}. \cup \{X_i \mid i \in I\} \in \mathrm{Con}_A \Rightarrow \cup \{Y_i \mid i \in I\} \in \mathrm{Con}_B$
3. $\{(X_1, Y_1), \cdots, (X_n, Y_n)\} \vdash (X, Y) \Longleftrightarrow$
   $\{(X_1, Y_1), \cdots, (X_n, Y_n)\} \neq \emptyset \ \& \ \cup \{Y_i \mid X \vdash_A^* X_i\} \vdash_B^* Y$

函数空间的符号 (X, Y) 断言：一个函数如果输入满足 X，则其输出满足 Y。我们要证明这个构造确实给出了信息系统，并且给出信息系统函数空间的元素的另一个特征。

引理 12.29 设 $\mathscr{A} = (A, \mathrm{Con}_A, \vdash_A)$ 和 $\mathscr{B} = (B, \mathrm{Con}_B, \vdash_B)$ 是信息系统，则 $\mathscr{A} \to \mathscr{B}_\perp$ 是一个信息系统。

243

我们有 $r \in |\mathscr{A} \to \mathscr{B}_\perp|$ 当且仅当 $r \subseteq \mathrm{Con}_A \times \mathrm{Con}_B$，因此，$r$ 是一个关系（我们用中缀的方式书写），它满足：对所有 $X, X' \in \mathrm{Con}_A, Y, Y' \in \mathrm{Con}_B$，有

(a) $\emptyset r Y \Longleftrightarrow Y = \emptyset$

(b) $XrY \ \& \ XrY' \Rightarrow Xr(Y \cup Y')$

(c) $X' \vdash_A^* X \ \& \ XrY \ \& \ Y \vdash_B^* Y' \Rightarrow X' r Y'$

证明 设 $\mathscr{A}$、$\mathscr{B}$ 是信息系统。我们首先检验 $\mathscr{A} \to \mathscr{B}_\perp$ 是一个信息系统。比较困难的是要证明定义信息系统中的公理 3 和 5，剩下的公理的证明留给读者。

公理 3。假设 $\{(X_1, Y_1), \cdots, (X_n, Y_n)\} \vdash (X, Y)$。我们需要证明

$$\{(X_1, Y_1), \cdots, (X_n, Y_n), (X, Y)\} \in \mathrm{Con}$$

于是，我们需要证明，如果 $J \subseteq \{1, \cdots, n\}$ 和 $\cup \{X_j \mid j \in J\} \cup X \in \mathrm{Con}_A$，则

$$\bigcup \{Y_j \mid j \in J\} \cup Y \in \mathrm{Con}_B$$

假设 $\cup \{X_j \mid j \in J\} \cup X \in \mathrm{Con}_A$，则

$$\bigcup \{X_j \mid j \in J\} \cup \bigcup \{X_i \mid X \vdash_A^* X_i\} \in \mathrm{Con}_A$$

现在，因为 $\{(X_1, Y_1), \cdots, (X_n, Y_n)\} \in \mathrm{Con}$，这就使得

$$\bigcup \{Y_j \mid j \in J\} \cup \bigcup \{Y_i \mid X \vdash_A^* X_i\} \in \mathrm{Con}_B$$

但是$\cup\{Y_i \mid X \vdash_A^* X_i\} \vdash_B^* Y$,因为$\{(X_1,Y_1),\cdots,(X_n,Y_n)\} \vdash (X,Y)$。因此,

$$\bigcup \{Y_j \mid j \in J\} \cup \bigcup \{Y_i \mid X \vdash_A^* X_i\} \vdash_B^* Y$$

所以,$\cup\{Y_j \mid j \in J\} \cup Y \in \mathrm{Con}_B$,公理3得证。

公理5。假设

$$\{(X_1,Y_1),\cdots,(X_n,Y_n)\} \vdash^* \{(Z_1,V_1),\cdots,(Z_{m-1},V_{m-1})\} \vdash (U,W)$$

我们需要证明$\{(X_1,Y_1),\cdots,(X_n,Y_n)\} \vdash (U,W)$,即

$$\bigcup \{Y_i \mid U \vdash_A^* X_i\} \vdash_B^* W$$

244

假设$U \vdash_A^* Z_j$。于是,因为$\cup\{Y_i \mid Z_j \vdash_A^* X_i\} \vdash_B^* V_j$,所以我们有$\cup\{Y_i \mid U \vdash_A^* X_i\} \vdash_B^* V_j$。因此

$$\bigcup \{Y_i \mid U \vdash_A^* X_i\} \vdash_B^* \bigcup \{V_j \mid U \vdash_A^* Z_j\} \vdash_B^* W$$

由$\vdash_B^*$的传递性得所需结果,公理5得证。

下面证明$\mathscr{A} \to \mathscr{B}_\perp$的元素可以用满足(a)、(b)和(c)的关系刻画。

"$\Rightarrow$":假设r是$\mathscr{A} \to \mathscr{B}_\perp$的元素,于是,$r$非空,所以$r$含有某个$(X,Y)$。由$\mathscr{A} \to \mathscr{B}_\perp$的继承关系的定义,我们有$(\emptyset,\emptyset) \in r$。这样,(a)的"$\Leftarrow$"成立。反过来,因为在提升函数空间中形为$(\emptyset,Y)$的惟一符号是$(\emptyset,\emptyset)$,所以(a)的"$\Rightarrow$"成立。性质(b)、(c)可直接由$\mathscr{A} \to \mathscr{B}_\perp$定义中的2和3得到。

"$\Leftarrow$":假设$r \subseteq \mathrm{Con}_A \times \mathrm{Con}_B$满足(a)、(b)和(c)。当然$r$是$(\mathrm{Con}_A \setminus \{\emptyset\}) \times \mathrm{Con}_B \cup \{(\emptyset,\emptyset)\}$的非空子集。为了证明$r \in |\mathscr{A} \to \mathscr{B}_\perp|$,我们需要证明$r$是一致的和$\vdash$封闭的。

假设$\{(X_1,Y_1),\cdots,(X_n,Y_n)\} \subseteq r$,又设$I \subseteq \{1,\cdots,n\}$,$X =_{def} \cup\{X_i \mid i \in I\} \in \mathrm{Con}_A$。于是对所有的$i \in I$,我们有$X \vdash_A^* X_i$,其中由(c),$(X_i,Y_i) \in r$确保$(X,Y_i) \in r$。使用(b),我们得$\cup\{Y_i \mid i \in I\} \in \mathrm{Con}_B$。所以,$r$是一致的。

现在我们证明r是$\vdash$封闭的。假设$\{(X_1,Y_1),\cdots,(X_n,Y_n)\} \subseteq r$且$\{(X_1,Y_1),\cdots,(X_n,Y_n)\} \vdash (X,Y)$。我们需要证明$(X,Y) \in r$。由(c),因为$(X_i,Y_i) \in r$,所以如果$X \vdash_A^* X_i$,那么$(X,Y_i) \in r$。那么由(b)的几次应用,得到$(X,Y') \in r$,其中$Y' =_{def} \cup\{Y_i \mid X \vdash_A^* X_i\}$。但现在由$\vdash$的定义,我们得$Y' \vdash_B^* Y$。所以,由(c)得到$(X,Y) \in r$。 □

斯科特称这种关系是逼近映射。直观上,逼近映射表示一个信息系统中的信息如何推导出另一个信息系统中的信息。对逼近映射$r \in |\mathscr{A} \to \mathscr{B}_\perp|$,$XrY$表示$\mathscr{A}$中的信息$X$推导出$\mathscr{B}$中的信息$Y$。特别地,从$\mathscr{A}$给定的输入得出$\mathscr{B}$中的输出的计算诱导出关系$r$。实际上,逼近映射,即

$$|\mathscr{A} \to \mathscr{B}_\perp|$$

和连续函数

$$[\,|\mathscr{A}| \to |\mathscr{B}|_\perp\,]$$

形成一一对应;对应关系确定了以包含关系排序的元素 $|\mathscr{A}\to\mathscr{B}_\perp|$ 和逐点排序的连续函数 [245] $[|\mathscr{A}|\to|\mathscr{B}|_\perp]$ 之间的有序同构。

容易证明,这种对应关系是表示提升信息系统元素的完全偏序的特殊方法。读者回忆 8.3.4 节,完全偏序 D 上提升构造 $D_\perp$ 时需要元素 $\perp$ 和一一对应函数 $\lfloor - \rfloor$,满足:对所有的 $x\in D$,

$$\perp \neq \lfloor x \rfloor$$

于是,提升的完全偏序 $D_\perp$ 是 D 的副本,它由 $x\in D$ 的对应元素 $\lfloor x\rfloor$ 和 $\perp$ 组成。当提升一个由信息系统的元素形成的完全偏序 $|\mathscr{A}|$ 时,$\mathscr{A}$ 的元素永远非空,我们可以利用这一事实并选择 $\perp=\emptyset$ 和 $\lfloor x\rfloor = x$。下面的命题设定提升采用了这样特殊的选择。这个选择简化了 8.3.4 节引入的操作 $(-)^*$。假设 $f:|\mathscr{A}|\to|\mathscr{B}|_\perp$ 是信息系统 $\mathscr{A}$ 和 $\mathscr{B}$ 的元素的完全偏序之间的连续函数。函数 f 扩展为具有 $\perp$ 和 $\lfloor - \rfloor$ 选择的函数

$$f^*: |\mathscr{A}|_\perp \to |\mathscr{B}|_\perp$$

其定义为

$$f^*(z) = \begin{cases} \emptyset, & z=\emptyset \\ f(z), & \text{其他} \end{cases}$$

定理 12.30 设 $\mathscr{A}$ 和 $\mathscr{B}$ 是信息系统。定义

$$|-|: |\mathscr{A}\to\mathscr{B}_\perp| \to [|\mathscr{A}|\to|\mathscr{B}|_\perp]$$
$$`-': [|\mathscr{A}|\to|\mathscr{B}|_\perp] \to |\mathscr{A}\to\mathscr{B}_\perp|$$

其定义为

$$|r| = \lambda x\in|\mathscr{A}|.\ \bigcup\{Y \mid \exists X\subseteq x.\ (X,Y)\in r\}$$
$$`f' = \{(X,Y)\in \mathrm{Con}_A\times\mathrm{Con}_B \mid Y\subseteq f^*(\overline{X})\}$$

于是,$|-|$ 和 '-' 是关于同构 $|\mathscr{A}\to\mathscr{B}_\perp|\cong[|\mathscr{A}|\to|\mathscr{B}|_\perp]$ 的互逆函数。

函数 '-' 满足:

$$`f' = \{(X,Y) \mid \emptyset\neq X\in\mathrm{Con}_A \ \&\ Y\subseteq^{fin} f(\overline{X})\}\cup\{(\emptyset,\emptyset)\}$$

证明 容易证明 $|-|$ 是良定义的,即 $|-|$ 给出的值是连续函数。证明 '-' 产生 $\mathscr{A}\to\mathscr{B}_\perp$ 的元素留作练习(见练习 12.31)。由 $|-|$ 和 '-' 定义知,它们是单调的。

[246] 现在我们宣称:对 $r\in|\mathscr{A}\to\mathscr{B}_\perp|$,$X\in\mathrm{Con}_A$,$Y\in\mathrm{Con}_B$,有

$$(X,Y)\in r \Longleftrightarrow Y\subseteq |r|^*(\overline{X})$$

"$\Rightarrow$"的证明直接由 $|-|$ 的定义得到。"$\Leftarrow$"用 r 的性质(b)和(c)以及引理 12.29 去证明:

假设对 $X\in\mathrm{Con}_A$,$Y\in\mathrm{Con}_B$,有 $Y\subseteq|r|^*(\overline{X})$。由 $|r|$ 的定义,一定存在

$$(X_1,Y_1),\cdots,(X_n,Y_n)\in r$$

使得

$$X_1,\cdots,X_n \subseteq \bar{X}, \text{即} X \vdash^* X_1 \cup \cdots \cup X_n \tag{1}$$

以及

$$Y \subseteq Y_1 \cup \cdots \cup Y_n, \text{所以} Y_1 \cup \cdots \cup Y_n \vdash^* Y \tag{2}$$

因为 $X_1 \cup \cdots \cup X_n \vdash^* X_i$ 和 $X_i r Y_i$，由(c)，我们得到 $(X_1 \cup \cdots \cup X_n) r Y_i, 1 \leqslant i \leqslant n$。所以，重复使用(b)，得

$$(X_1 \cup \cdots \cup X_n) r (Y_1 \cup \cdots \cup Y_n)$$

但现在由(c)，从(1)和(2)我们得到 XrY。这就是上述宣称所要证明的。

现在我们证明 $|\text{-}|$ 和 '-' 给出了一一对应关系。对 $r \in |\mathscr{A} \to \mathscr{B}_\perp|$，$X \in \mathrm{Con}_A$ 和 $Y \in \mathrm{Con}_B$，直接由 '-' 的定义，我们得到

$$\begin{aligned}(X,Y) \in r &\Longleftrightarrow Y \subseteq |r|^*(\bar{X}) \\ &\Longleftrightarrow (X,Y) \in `|r|'\end{aligned}$$

所以，$r = `|r|'$。同时也看到，对 $f \in [|\mathscr{A}| \to |\mathscr{B}|_\perp]$，$X \in \mathrm{Con}_A$ 和 $Y \in \mathrm{Con}_B$，直接从 '-' 的定义以及上面的宣称得到

$$Y \subseteq f^*(\bar{X}) \Longleftrightarrow (X,Y) \in `f' \Longleftrightarrow Y \subseteq |`f'|^*(\bar{X})$$

对 $X \in \mathrm{Con}_A$，连续函数由 $|\mathscr{A}|$ 中形如 $\bar{X}$ 的有限元素的值惟一确定：任一元素 x 是 ω 链 $\bar{X}_0 \subseteq \bar{X}_1 \subseteq \cdots$ 的最小上界，由连续性，得 $f(x) = \cup_n f(\bar{X}_n)$。所以 $f = |`f'|$。

我们得到 $|\text{-}|$ 和 '-' 确定了一个同构。

从 $f: |\mathscr{A}| \to |\mathscr{B}|_\perp$ 的扩展到 f^* 的定义，可以得到 '-' 的另一个特征。 □

练习 12.31 试证明：定理 12.30 定义的函数 '-' 是一个良定义的函数，即，给定连续函数 $f: |\mathscr{A}| \to |\mathscr{B}|_\perp$，则 247

$$`f' = \{(X,Y) \in \mathrm{Con}_A \times \mathrm{Con}_B \mid Y \subseteq f^*(\bar{X})\}$$

是 $\mathscr{A} \to \mathscr{B}_\perp$ 的元素。 □

练习 12.32 对信息系统 $\mathscr{A}$ 和 $\mathscr{B}$，试描述 $\mathscr{A} \to \mathscr{B}_\perp$ 的底元素中的符号。 □

定理 12.33 提升函数空间的操作是按 $\trianglelefteq$ 排序的信息系统上的连续操作。

证明 我们将用引理 12.15 来证明，提升函数空间对 $\trianglelefteq$ 的信息系统在每一个变量上分别是连续操作。

首先我们证明这种构造在第一个变量上是单调的。设 $\mathscr{A} \trianglelefteq \mathscr{A}'$ 和 $\mathscr{B}$ 是信息系统。记 $\mathscr{C} = (C, \mathrm{Con}, \vdash) = \mathscr{A} \to \mathscr{B}_\perp$ 和 $\mathscr{C}' = (C', \mathrm{Con}', \vdash') = \mathscr{A}' \to \mathscr{B}_\perp$。我们要证明 $\mathscr{C} \trianglelefteq \mathscr{C}'$，因此，通过检查 $\trianglelefteq$ 定义的 1,2,3 来证明 $\mathscr{C} \trianglelefteq \mathscr{C}'$：

1. 显然，$\mathscr{C}$ 的符号包含在 $\mathscr{C}'$ 的符号中。

2. 设 $(X_1,Y_1),\cdots,(X_n,Y_n)$ 是 $\mathscr{C}$ 的符号。因为 $\mathscr{A} \trianglelefteq \mathscr{A}'$，我们得，对任何子集 $I \subseteq \{1,\cdots,n\}$，$\cup_{i \in I} X_i \in \mathrm{Con}_A$ 当且仅当 $\cup_{i \in I} X_i \in \mathrm{Con}_{A'}$。所以，通过检查提升函数的一致性谓词的定义，得

$$\{(X_1,Y_1),\cdots,(X_n,Y_n)\} \in \mathrm{Con} \text{ 当且仅当 } \{(X_1,Y_1),\cdots,(X_n,Y_n)\} \in \mathrm{Con}'$$

3. 设$(X_1,Y_1),\cdots,(X_n,Y_n)$和$(X,Y)$是$\mathscr{C}$的符号。因为$\mathscr{A} \trianglelefteq \mathscr{A}'$,得$X \vdash_A^* X_i$当且仅当$X \vdash_{A'}^* X_i$。所以,通过检查提升函数空间的推导关系定义,得

$$\{(X_1,Y_1),\cdots,(X_n,Y_n)\} \vdash (X,Y) \text{ 当且仅当 } \{(X_1,Y_1),\cdots,(X_n,Y_n)\} \vdash' (X,Y)$$

所以,$\mathscr{C} \trianglelefteq \mathscr{C}'$。因此,函数提升空间在第一个变量上是单调的。

现在我们证明,提升函数空间在第一个变量的符号集上是连续的。设$\mathscr{A}_0 \trianglelefteq \mathscr{A}_1 \trianglelefteq \cdots \trianglelefteq \mathscr{A}_i \trianglelefteq \cdots$是信息系统$\mathscr{A}_i = (A_i, \mathrm{Con}_i, \vdash_i)$的$\omega$链,又设$(X,Y)$是$(\cup_i \mathscr{A}_i) \to \mathscr{B}_\perp$的符号。那么,$X$是$\cup_i \mathscr{A}_i$的一致的集合。但对某个$i$,有$X \in \mathrm{Con}_i$,所以$(X,Y)$是$\mathscr{A}_i \to \mathscr{B}_\perp$的符号。所以,正如我们希望的,$(X,Y)$是$\cup_i(\mathscr{A}_i \to \mathscr{B}_{i\perp})$的符号。

由引理 12.15,我们可以推得提升函数空间在第一个变量上连续,同样,在第二个变量上也连续,因此,提升函数空间是连续操作。 □

现在,我们从信息系统 **0** 开始通过提升、和、乘积以及提升函数空间的组合来给出信息系统的定义。因为这些操作对应于$\trianglelefteq$是连续的,所以该定义可以是递归的。这些构造可以用来给出具有递归类型的语言的语义。

248

例 操作$X \mapsto (X \to X_\perp)$是信息系统上的连续操作。它有最小不动点$\mathscr{L} = (\mathscr{L} \to \mathscr{L}_\perp)$。这个信息系统有元素的完全偏序$D = |\mathscr{L}|$,使得以下同构关系链成立:

$$D = |\mathscr{L}| = |\mathscr{L} \to \mathscr{L}_\perp| \cong [\,|\mathscr{L}| \to |\mathscr{L}|_\perp\,] = [D \to D_\perp]$$

这来自下面的事实:提升函数空间的信息系统的构造和相应的完全偏序的构造同构,即$D \cong [D \to D_\perp]$。 □

练习 12.34 我们为什么用关系$\trianglelefteq$去构造完全偏序,而不是用一个信息系统(在坐标方式下)和另一个信息系统的简单包含关系?这是一个偏序,并且确实给出另一个大型完全偏序。试验证它的一个主要缺点:对信息系统的提升函数空间构造对应于这个包含次序,虽然对右变量是连续的,但对左变量甚至是不单调的。 □

12.6 进一步阅读资料

本章介绍的内容主要基于文献[103],另外,我们推荐 Dana Scott 在[90]中介绍的信息系统。注意,通常信息系统表示含底的斯科特域。最近出版的关于域论的书[87]在信息系统的基础上介绍了域论,内容清晰易懂,适合数学专业本科生阅读。在[19]中,Gordon Plotkin 的讲义用信息系统的一个变体表示前域(不必含底)。信息系统可以作为"无点拓扑"中的特殊邻域(参考[53,98]),其中主要关心邻域而不是点,这种观点在拓扑和逻辑中都十分有用。若把符号作为建立语法的命题,则信息系统能给出更多的逻辑特征。结合这个思想,Samson Abramsky 通过把空间及其邻域表示方之间的对偶性提出了域论逻辑[2]。为了处理 Plotkin 幂域,要求推广域的概念,以便可以表示更大的域(SFP 对象)。[2]和[108]介绍了对域的推广。在 20 世纪 70 年代后期,Gérard Berry 发现了另一种"稳定"的域论,它给出了大部分指称

语义的基础。这里,把完全偏序限制在特殊的斯科特域——dI 域,函数是稳定且连续的。该域论有自己的表示方法,把信息系统的符号的角色转化为“事件”的角色,而这里关于信息系统上的工作可以转化到“事件结构”上进行(参考[104,105])。 249

第13章 递归类型

我们把第11章介绍的函数式语言的语法、操作语义和指称语义扩展到包含递归类型的函数式语言。指称语义使用信息系统来指称递归类型。考察活性语言和惰性语言的自然数、表等递归类型和λ演算模型的类型。信息系统的使用会带来额外的性质。它导致了适用性的相对简单的证明和活性λ演算、惰性λ演算中的不动点算子的刻画。这些讨论为我们提供了论证活性函数式语言(例如标准ML)以及惰性函数式语言(例如Miranda[⊖]、Orwell或Haskell)的数学基础。

13.1 活性语言

在上一章中,我们理解了递归定义类型的方法。用这种思想,我们把递归定义类型的机制引入到第11章的语言中。类型表达式τ形如:

$$\tau ::= \mathbf{1} \mid \tau_1 * \tau_2 \mid \tau_1 -> \tau_2 \mid \tau_1 + \tau_2 \mid X \mid \mu X.\, \tau$$

其中,X是类型变量的无限集合,$\mu X.\, \tau$是递归定义类型。这里有我们熟悉的类型构造符:积、函数空间以及和。存在惟一的基本类型**1**,它由空元组()这一个值组成。以后会定义像数和表的其他类型及其操作。类型表达式的自由变量和约束变量用标准方式定义。通常,当类型表达式的所有变量都是约束时,则称该类型表达式是封闭的。

无类型的项的语法为

$$\begin{aligned} t ::= {} & () \mid (t_1, t_2) \mid \mathbf{fst}(t) \mid \mathbf{snd}(t) \mid \\ & x \mid \lambda x.\, t \mid (t_1 t_2) \mid \\ & \mathbf{inl}(t) \mid \mathbf{inr}(t) \mid \mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).\, t_1, \mathbf{inr}(x_2).\, t_2 \mid \\ & \mathbf{abs}(t) \mid \mathbf{rep}(t) \mid \\ & \mathbf{rec}\, f.\, (\lambda x.\, t) \end{aligned}$$

其中,x, x_1, x_2, f是**Var**中的变量。语法中包含了类似于第11章的操作。两个新的操作**abs**和**rep**用于递归定义类型,以后将会解释。语法中没有包括构造

$$\mathbf{let}\ x \Leftarrow t_1\ \mathbf{in}\ t_2$$

251

但可以把它定义为$((\lambda x.\, t_2)\, t_1)$的简写。

我们假设每一个变量x都有惟一的封闭类型$type(x)$。为了变量够用,我们假设对每一个封闭的类型τ,

⊖ Miranda是Research Software Ltd的商标。

$$\{x \in \mathbf{Var} \mid type(x) = \tau\}$$

是无限的。

把类型赋予项可以扩展为一般的类型检查断言 $t: \tau$,其中 t 是项,τ 是封闭类型,类型规则如下:

类型规则

变量: $\dfrac{}{x:\tau}$ 如果 $type(x) = \tau$

积: $\dfrac{}{(\,):\mathbf{1}}$

$$\frac{t_1:\tau_1 \quad t_2:\tau_2}{(t_1, t_2):\tau_1 * \tau_2}$$

$$\frac{t:\tau_1 * \tau_2}{\mathbf{fst}(t):\tau_1} \quad \frac{t:\tau_1 * \tau_2}{\mathbf{snd}(t):\tau_2}$$

函数类型: $\dfrac{x:\tau_1 \quad t:\tau_2}{\lambda x.\, t:\tau_1 -> \tau_2} \quad \dfrac{t_1:\tau_1 -> \tau_2 \quad t_2:\tau_1}{(t_1 \quad t_2):\tau_2}$

和: $\dfrac{t:\tau_1}{\mathbf{inl}(t):\tau_1 + \tau_2} \quad \dfrac{t:\tau_2}{\mathbf{inr}(t):\tau_1 + \tau_2}$

$$\frac{t:\tau_1 + \tau_2 \quad x_1:\tau_1 \quad x_2:\tau_2 \quad t_1:\tau \quad t_2:\tau}{\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).\, t_1,\ \mathbf{inr}(x_2).\, t_2:\tau}$$

递归类型: $\dfrac{t:\tau[\mu X.\,\tau/X]}{\mathbf{abs}(t):\mu X.\,\tau} \quad \dfrac{t:\mu X.\,\tau}{\mathbf{rep}(t):\tau[\mu X.\,\tau/X]}$

rec: $\dfrac{f:\tau \quad \lambda x.\, t:\tau}{\mathbf{rec}\, f.\, (\lambda x.\, t):\tau}$

和以前一样,对某一类型 τ,当 $t: \tau$ 时,t 称为有类型的。有类型的项 t 的自由变量 $FV(t)$ 的定义和第 11 章中的定义完全一样(见 11.1 节)。因此,我们主要讨论有类型的项。

252

本章语言允许递归类型定义,如自然数

$$N \equiv_{def} \mu X.\, (\mathbf{1} + X)$$

或自然数的表

$$L \equiv_{def} \mu Y.\, (\mathbf{1} + N * Y)$$

或更不寻常的类型,如

$$\Lambda \equiv_{def} \mu Z.\, (Z -> Z)$$

这是一个(活性)λ 演算模型。项构造符 **abs** 和 **rep** 用于类型 $\mu X.\,\tau$ 及其展开式 $\tau[\mu X.\,\tau/X]$ 之间的同构关系。它们在递归类型上定义有用的操作时起重要作用。构造符 **rep** 将元素 $t: \mu X.\,\tau$ 映为 $\mathbf{rep}(t): \tau[\mu X.\,\tau/X]$,构造符 **abs** 将 $u: \tau[\mu X.\,\tau/X]$ 这样的表示映为它的抽象 **abs**

$(u):\mu X.\tau$。为了理解 **abs** 和 **rep** 的用途，我们看两个简单的类型，自然数和表，以及如何定义涉及它们的函数。

例　自然数

自然数的类型定义为

$$N \equiv_{def} \mu X.(\mathbf{1}+X)$$

对这种类型，**rep** 可以看成是映射

$$N \xrightarrow{\mathbf{rep}} \mathbf{1}+N$$

而 **abs** 看成是映射

$$N \xleftarrow{\mathbf{abs}} \mathbf{1}+N$$

常量 *Zero* 定义为

$$Zero \equiv_{def} \mathbf{abs}(\mathbf{inl}())$$

对任意项 $t:N$，后继操作定义为

$$Succ(t) \equiv_{def} \mathbf{abs}(\mathbf{inr}(t))$$

于是，后继函数为项

$$\lambda x.\,Succ(x):N \text{->} N$$

253

其中，x 是类型为 N 的变量。这些操作允许我们把类型为 N 的“数”定义成

$$\begin{array}{r} Zero \\ Succ(Zero) \\ Succ(Succ(Zero)) \\ \vdots \end{array}$$

我们通常借助于 *Case* 构造来定义自然数的函数

$$\begin{aligned}&\mathbf{Case}\ x\ \mathbf{of}\ Zero.\,t_1,\\ &\qquad Succ(z).\,t_2\end{aligned}$$

它在 x 为 *Zero* 的情况下产生 t_1，在 x 为后继 $Succ(z)$ 的情况下产生 t_2（常常依赖于 z）。这也可以定义为下列表达式的简写

$$\begin{aligned}&\mathbf{Case}\ \mathbf{rep}(x)\ \mathbf{of}\ \mathbf{inl}(w).\,t_1\\ &\qquad \mathbf{inr}(z).\,t_2\end{aligned}$$

现在我们定义加法

$$\begin{aligned}add \equiv_{def} \mathbf{rec}\,f.\,(\lambda x.\,\lambda y.\,&\mathbf{Case}\ x\ \mathbf{of}\ Zero.\,y,\\ &Succ(z).\,Succ((fz)y)\end{aligned}$$

其类型为 $(N\text{->}(N\text{->}N))$。 □

例 表

自然数 N 上的表定义为

$$L \equiv_{def} \mu Y.\ (\mathbf{1} + N * Y)$$

我们能实现通常的表构造,空表定义为

$$Nil \equiv_{def} \mathbf{abs}(\mathbf{inl}(\,))$$

对任意 $p: N * L$, $Cons$ 操作定义为

$$Cons(p) \equiv_{def} \mathbf{abs}(\mathbf{inr}(p))$$

$Cons$ 操作作用在由项 $n: N$ 和 $l: L$ 组成的序偶 $(n,l): N * L$ 上,以产生表 $Cons(n,l)$,其"头"为 n,"尾"为 l。与其相关联的函数项表示为

254

$$\lambda x.\ Cons(x): N * L \to L$$

其中,x 是类型为 $N * L$ 的变量。表上的函数习惯上用 *case* 构造定义。表上的一般 *case* 构造

$$\begin{aligned}&\mathbf{Case}\ l\ \mathbf{of}\ Nil.\ t_1\\ &\quad Cons(x,l').\ t_2\end{aligned}$$

在表 l 为空表的情况产生 t_1,而在表 l 为 $Cons(x,l')$ 的情况下产生 t_2。也可以定义为

$$\begin{aligned}&\mathbf{case}\ \mathbf{rep}(l)\ \mathbf{of}\ \mathbf{inl}(w).\ t_1\\ &\qquad \mathbf{inr}(z).\ t_2[\mathbf{fst}(z)/x, \mathbf{snd}(z)/l']\end{aligned}$$

□

13.2 活性操作语义

和以前一样,活性求值用有类型的封闭项 t 和标准型 c 之间的关系

$$t \to c$$

来表示。类型为 τ 的标准型 C_τ 是由下列规则给出的封闭项:

$$\frac{}{(\,) \in C_1}$$

$$\frac{c_1 \in C_{\tau_1}\quad c_2 \in C_{\tau_2}}{(c_1, c_2) \in C_{\tau_1 * \tau_2}}$$

$$\frac{\lambda x.\ t: \tau_1 \to \tau_2 \quad \lambda x.\ t\ \text{是封闭的}}{\lambda x.\ t \in C_{\tau_1 \to \tau_2}}$$

$$\frac{c \in C_{\tau_1}}{\mathbf{inl}(c) \in C_{\tau_1+\tau_2}} \qquad \frac{c \in C_{\tau_2}}{\mathbf{inr}(c) \in C_{\tau_1+\tau_2}}$$

$$\frac{c \in C_{\tau[\mu X.\ \tau/X]}}{\mathbf{abs}(c) \in C_{\mu X.\ \tau}}$$

产生递归类型的标准型的惟一规则是最后一条规则，它表示类型为 $\mu X.\tau$ 的标准型是 $\tau[\mu X.\tau/X]$ 的标准型的 $\mathbf{abs}(c)$。因为，通常 $\tau[\mu X.\tau/X]$ 不小于 $\mu X.\tau$，所以，标准型不能用类型上的结构归纳法定义——其原因是它们有归纳定义。 255

例 自然数

自然数的类型 $N \equiv \mu X.(1+X)$ 是与和的两个分量有关的标准型。存在与和的左边部分有关的单一的标准型

$$Zero \equiv_{def} \mathbf{abs}(\mathbf{inl}())$$

与和的右边部分有关的标准型为

$$\mathbf{abs}(\mathbf{inr}(c))$$

其中，c 是 N 的标准型。我们使用简写

$$Succ(t) \equiv \mathbf{abs}\ \mathbf{inr}(t)$$

这样，可以得到 N 的标准型：$Succ(Zero)$，$Succ(Succ(Zero))$，…。

作为数字的标准型是由 $Zero$ 通过重复应用后继操作而得到的。在指称语义中，N 指称信息系统，其元素与自然数的离散完全偏序同构。 □

例 表

对标准型 $n:N$ 和 $l:L$，由 $L \equiv_{def} \mu Y.(\mathbf{1}+N*Y)$ 定义的自然数上的表类型具有标准型

$$Nil \equiv \mathbf{abs}(\mathbf{inl}()):L$$
$$Cons(n,l) \equiv \mathbf{abs}(\mathbf{inr}(n,l))$$

换句话说，类型 L 的标准型或者是空表或者是自然数的有限表 $[n_1,n_2,\cdots]$，该表为 $Cons(n_1, Cons(n_2, Cons(\cdots)\cdots))$。 □

有类型的封闭项 t 和标准型 c 之间的活性求值关系由以下规则定义： 256

求值规则

$$\frac{}{c \to c} \quad \text{如果 } c \text{ 是标准型}$$

$$\frac{t_1 \to c_1 \quad t_2 \to c_2}{(t_1,t_2) \to (c_1,c_2)}$$

$$\frac{t \to (c_1,c_2)}{\mathbf{fst}(t) \to c_1} \qquad \frac{t \to (c_1,c_2)}{\mathbf{snd}(t) \to c_2}$$

$$\frac{t_1 \to \lambda x.t_1' \quad t_2 \to c_2 \quad t_1'[c_2/x] \to c}{(t_1\ \ t_2) \to c}$$

$$\frac{t \to \mathbf{inl}(c_1) \quad t_1[c_1/x_1] \to c}{\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).t_1,\mathbf{inr}(x_2).t_2 \to c} \qquad \frac{t \to \mathbf{inr}(c_2) \quad t_2[c_2/x_2] \to c}{\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).t_1,\mathbf{inr}(x_2).t_2 \to c}$$

$$\frac{t \to c}{\mathbf{abs}(t) \to \mathbf{abs}(c)} \quad \frac{t \to \mathbf{abs}(c)}{\mathbf{rep}(t) \to c}$$

$$\overline{\mathbf{rec}\, f.\ (\lambda x.\ t) \to \lambda x.\ t[\mathbf{rec}\, f.\ (\lambda x.\ t)/f]}$$

求值是确定的,并且类型是一致的。

命题 13.1 设 t 是有类型的封闭项,c, c_1, c_2 是标准型,则

(i) $t \to c$ & $t : \tau \Rightarrow c : \tau$,

(ii) $t \to c_1$ & $t \to c_2 \Rightarrow c_1 \equiv c_2$。

证明 用规则归纳法证明。 □

13.3 活性指称语义

一个有类型的封闭项可以求值到一个标准型或者发散。相应地,我们把项的指称作为 $(V_\tau)_\perp$ 的元素,其中 V_τ 是类型 τ 的包括标准型指称值在内的完全偏序。这样我们的语言允许递归定义类型。我们用上一章的方法来定义每个类型值的信息系统。

257

类型环境 χ 是从类型变量到信息系统的函数。施结构归纳于类型表达式,定义

$$\begin{aligned}
\mathcal{V}[\mathbf{1}]\chi &= (\emptyset, \{\emptyset\}, \emptyset)_\perp && \text{(也称为 } \mathbf{1}\text{)} \\
\mathcal{V}[\tau_1 * \tau_2]\chi &= (\mathcal{V}[\tau_1]\chi) \times (\mathcal{V}[\tau_2]\chi) \\
\mathcal{V}[\tau_1 -> \tau_2]\chi &= (\mathcal{V}[\tau_1]\chi) \to (\mathcal{V}[\tau_2]\chi)_\perp \\
\mathcal{V}[\tau_1 + \tau_2]\chi &= (\mathcal{V}[\tau_1]\chi) + (\mathcal{V}[\tau_2]\chi) \\
\mathcal{V}[X]\chi &= \chi(X) \\
\mathcal{V}[\mu X.\ \tau]\chi &= \mu I.\ \mathcal{V}[\tau]\chi[I/X]
\end{aligned}$$

所有在语义定义子句右边的操作是对信息系统的操作。在环境 χ 中,类型表达式 $\mu X.\ \tau$ 指称到信息系统完全偏序中

$$I \mapsto \mathcal{V}[\tau]\chi[I/X]$$

的 $\sqsubseteq$ 最小不动点。

于是,有类型的封闭项 τ 和信息系统

$$\mathcal{V}[\tau]\chi = (\mathrm{Tok}_\tau, \mathrm{Con}_\tau, \vdash_\tau)$$

相关,其元素构成了值

$$V_\tau =_{def} |\mathcal{V}[\tau]\chi|$$

的完全偏序,其中类型环境 χ 是任意的,不影响指称值。对应于环境中的自由变量,类型 τ 的

项指称$(V_\tau)_\perp$的元素。为简单起见,我们将$\perp$和提升函数$\lfloor - \rfloor : V_\tau \to (V_\tau)_\perp$作如下解释。因为信息系统的元素永远非空,如果我们取$\perp = \emptyset$(空集),且对所有的$x \in V_\tau$,取$\lfloor x \rfloor = x$,则满足$\lfloor - \rfloor$和$\perp$的条件。

环境集合 **Env** 的完全偏序由

$$\rho : Var \to \bigcup \{V_\tau \mid \tau \text{ 是封闭的类型表达式}\}$$

组成,使得$\rho(x) \in V_{type(x)}$,它是逐点排序的。

在表示指称语义时,使用某些相等符号是有帮助的。在考虑信息系统元素的完全偏序的和构造与信息系统和的元素完全偏序同构(正如推论 12.24 所表达的那样)时,我们完全可以假设这两个完全偏序是相等的。也就是说,对信息系统$\mathscr{A}$和$\mathscr{B}$,我们取 258

$$|\mathscr{A}| + |\mathscr{B}| = |\mathscr{A} + \mathscr{B}|$$

其内射函数$in_1 : |\mathscr{A}| \to |\mathscr{A}| + |\mathscr{B}|$,$in_2 : |\mathscr{B}| \to |\mathscr{A}| + |\mathscr{B}|$定义为

$$in_1(x) =_{def} inj_1 x = \{(1,a) \mid a \in x\}$$
$$in_2(x) =_{def} inj_2 x = \{(2,b) \mid b \in x\}$$

更值得注意的是,在积的情况下有类似的相等符号。信息系统$|\mathscr{A}|$和$|\mathscr{B}|$的完全偏序的乘积$|\mathscr{A}| \times |\mathscr{B}|$和$|\mathscr{A} \times \mathscr{B}|$相等。为了强调,写作

$$|\mathscr{A}| \times |\mathscr{B}| = |\mathscr{A} \times \mathscr{B}|$$

在引理 12.27 中,元素$x \in |\mathscr{A}|$和$y \in |\mathscr{B}|$的序偶表示为元素$x \times y \in |\mathscr{A} \times \mathscr{B}|$。所以,$|\mathscr{A}| \times |\mathscr{B}|$和$|\mathscr{A} \times \mathscr{B}|$相等意味着$|\mathscr{A}| \times |\mathscr{B}|$中序偶的操作表示为集合的积

$$(x,y) = x \times y$$

其中$x \in |\mathscr{A}|$,$y \in |\mathscr{B}|$。投影函数$\pi_1 : |\mathscr{A}| \times |\mathscr{B}| \to |\mathscr{A}|$和$\pi_2 : |\mathscr{A}| \times |\mathscr{B}| \to |\mathscr{B}|$定义为[⊖]

$$\pi_1(z) =_{def} proj_1 z = \{a \mid \exists b.\ (a,b) \in z\}$$
$$\pi_2(z) =_{def} proj_2 z = \{b \mid \exists a.\ (a,b) \in z\}$$

有了这些相等符号,我们在与和及积类型有关的语义定义中,避免了明确指出同构的繁琐。然而,我们不能用逼近映射来表示连续函数,因为这样太会引起混淆。对信息系统$\mathscr{A}$和$\mathscr{B}$,我们使用定理 12.30 的同构

$$|-| : |\mathscr{A} \to \mathscr{B}_\perp| \to [|\mathscr{A}| \to |\mathscr{B}|_\perp]$$
$$`-' : [|\mathscr{A}| \to |\mathscr{B}|_\perp] \to |\mathscr{A} \to \mathscr{B}_\perp|$$

读者回想一下,这些函数定义为

⊖ 我们的约定仅当将信息系统的完全偏序的积中的序偶(x,y)表示为$x \times y$时为真;特别地,不针对类似于这里的(a,b)的序偶(它们用集合论中的通常的配对操作定义)。

$$|r| = \lambda x \in |\mathscr{A}|.\ \bigcup \{Y \mid \exists X \subseteq x.\ (X,Y) \in r\}$$

259 $$`f' = \{(X,Y) \mid \emptyset \neq X \in \mathrm{Con}_A \ \&\ Y \subseteq^{fin} f(\overline{X})\} \cup \{(\emptyset,\emptyset)\}$$

正如我们所期望的，具有非递归类型的项的指称语义本质上和第 11 章（11.3 节）中的活性语言是一样的。与函数类型相关的元素不是函数，而是用逼近映射来表示这些元素，这样有一些表面上的差别，所以，有时在语义定义中，说明这种与表示有关的同构。我们使用信息系统，意味着可以用另一种更具体的方法来表示语义定义的子句。下面加以解释说明。

指称语义

$$
\begin{array}{lll}
[\![()]\!] & =_{def} & \lambda\rho.\ \lfloor\{\emptyset\}\rfloor \\
 & = & \lambda\rho.\ \{\emptyset\} \\
[\![(t_1,t_2)]\!] & =_{def} & \lambda\rho.\ let\ v_1 \Leftarrow [\![t_1]\!]\rho, v_2 \Leftarrow [\![t_2]\!]\rho.\ \lfloor(v_1,v_2)\rfloor \\
 & = & \lambda\rho.\ [\![t_1]\!]\rho \times [\![t_2]\!]\rho \qquad (1) \\
[\![\mathbf{fst}(t)]\!] & =_{def} & \lambda\rho.\ let\ v \Leftarrow [\![t]\!]\rho.\ \pi_1(v) \\
 & = & \lambda\rho.\ proj_1[\![t]\!]\rho \qquad (2) \\
[\![\mathbf{snd}(t)]\!] & =_{def} & \lambda\rho.\ let\ v \Leftarrow [\![t]\!]\rho.\ \pi_2(v) \\
 & = & \lambda\rho.\ proj_2[\![t]\!]\rho \\
[\![x]\!] & =_{def} & \lambda\rho.\ \lfloor\rho(x)\rfloor \\
 & = & \lambda\rho.\ \rho(x) \\
[\![\lambda x.\ t]\!] & =_{def} & \lambda\rho.\ \lfloor`(\lambda v \in V_{type(x)}.\ [\![t]\!]\rho[v/x])'\rfloor \\
 & = & \lambda\rho.\ \{(U,V) \mid \emptyset \neq U \in \mathrm{Con}_{type(x)}\ \&\ V \subseteq^{fin} [\![t]\!]\rho[\overline{U}/x]\} \cup \\
 & & \qquad \{(\emptyset,\emptyset)\} \qquad (3) \\
[\![t_1 t_2]\!] & =_{def} & \lambda\rho.\ let\ r \Leftarrow [\![t_1]\!]\rho, v \Leftarrow [\![t_2]\!]\rho.\ |r|(v) \\
 & = & \lambda\rho.\ \cup\{V \mid \exists U \subseteq [\![t_2]\!]\rho.\ (U,V) \in [\![t_1]\!]\rho\} \qquad (4) \\
[\![\mathbf{inl}(t)]\!] & =_{def} & \lambda\rho.\ let\ v \Leftarrow [\![t]\!]\rho.\ \lfloor in_1(v)\rfloor \\
 & = & \lambda\rho.\ inj_1[\![t]\!]\rho \qquad (5) \\
[\![\mathbf{inr}(t)]\!] & =_{def} & \lambda\rho.\ let\ v \Leftarrow [\![t]\!]\rho.\ \lfloor in_2(v)\rfloor \\
 & = & \lambda\rho.\ inj_2[\![t]\!]\rho \\
[\![\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).\ t_1, \mathbf{inr}(x_2).\ t_2]\!] & & \\
 & =_{def} & \lambda\rho.\ let\ v \Leftarrow [\![t]\!]\rho. \\
 & & \qquad case\ v\ of\ in_1(v_1).\ [\![t_1]\!]\rho[v_1/x_1] \mid in_2(v_2).\ [\![t_2]\!]\rho[v_2/x_2] \\
[\![\mathbf{abs}(t)]\!] & =_{def} & [\![t]\!] \qquad (6) \\
[\![\mathbf{rep}(t)]\!] & =_{def} & [\![t]\!] \\
[\![\mathbf{rec}\ f.\ (\lambda x.\ t)]\!] & =_{def} & \lambda\rho.\ \lfloor\mu r.\ `(\lambda v.\ [\![t]\!]\rho[v/x, r/f])'\rfloor \\
 & = & \lambda\rho.\ \mu r.\ [\![\lambda x.\ t]\!]\rho[r/f] \qquad (7)
\end{array}
$$

260

注释

（1）读者回想一下，信息系统积中的元素 v_1, v_2 的序偶用集合的积 $v_1 \times v_2$ 表示。所以，以我们对提升的理解，

$$[\![(t_1,t_2)]\!]\rho = let\ v_1 \Leftarrow [\![t_1]\!]\rho, v_2 \Leftarrow [\![t_2]\!]\rho.\ v_1 \times v_2$$

当$[\![t_1]\!]\rho$或$[\![t_2]\!]\rho$为$\emptyset$时，$[\![(t_1,t_2)]\!]\rho$返回一个底元素$\emptyset$，所以它等于

$$[\![t_1]\!]\rho \times [\![t_2]\!]\rho$$

(2) 对信息系统$\mathscr{A}$和$\mathscr{B}$，由我们对积$|\mathscr{A}| \times |\mathscr{B}|$的理解可知，因为在投影$proj_1$中，$\emptyset$的投影是$\emptyset$，所以有

$$\begin{aligned}[\![\mathbf{fst}(t)]\!]\rho &= let\ v \Leftarrow [\![t]\!]\rho.\ proj_1 v \\ &= proj_1 [\![t]\!]\rho\end{aligned}$$

(3) 读者回忆定理12.30中两个函数$|-|$和'-'给出的逼近映射和连续函数之间的同构关系

$$|\mathscr{A} \to \mathscr{B}_\perp| \cong [\,|\mathscr{A}| \to |\mathscr{B}|_\perp\,]$$

我们有

$$\begin{aligned}[\![\lambda x.\ t]\!]\rho &= \lfloor\text{`}(\lambda v.\ [\![t]\!]\rho[v/x])\text{'}\rfloor \quad (\text{由定义}) \\ &= \text{`}(\lambda v.\ [\![t]\!]\rho[v/x])\text{'} \quad (\text{由我们对提升的理解}) \\ &= \{(U,V) \mid \emptyset \neq U \in \mathrm{Con}_{type(x)}\ \&\ V \subseteq^{fin} [\![t]\!]\rho[\overline{U}/x]\} \cup \{(\emptyset,\emptyset)\}\end{aligned}$$

(4) 假设$t_1 : \sigma \to \tau$，$t_2 : \sigma$。根据定理12.30，在$[\![t_1]\!]\rho = \lfloor r \rfloor$和$[\![t_2]\!]\rho = \lfloor v \rfloor$的情况下，我们有

$$\begin{aligned}[\![t_1 t_2]\!]\rho &= |r|(v) \\ &= \cup\ \{V \mid \exists U \subseteq v.\ (U,V) \in r\}\end{aligned}$$

于是，在这种情况下，(4)的两个表达式是一致的。而且，当$[\![t_1]\!]\rho$或者$[\![t_2]\!]\rho$为$\emptyset$时，它们是相同的，均为$\emptyset$。 261

(5) 根据11.11节的讨论，我们可以规定$\mathbf{inl}(t)$的类型$\tau_1 + \tau_2$——因为τ_2没有明确定义可能会影响$\mathbf{inl}(t)$的指称。但因为我们对信息系统$\mathscr{A}$和$\mathscr{B}$的和$|\mathscr{A}| + |\mathscr{B}|$的内射用特殊的表示，所以我们可以不定义成分$\tau_2$；不论$\tau_2$如何，$\mathbf{inl}(t)$的指称是一样的。定义简化为

$$\begin{aligned}[\![\mathbf{inl}(t)]\!]\rho &= let\ v \Leftarrow [\![t]\!]\rho.\ \lfloor in_1(v) \rfloor \\ &= let\ v \Leftarrow [\![t]\!]\rho.\ inj_1 v \\ &= inj_1 [\![t]\!]\rho\end{aligned}$$

(6) 由$\mu X.\ \tau$和$\tau[\mu X.\ \tau/x]$指称的信息系统之间的同构关系的两个对应部分分别用**abs**和**rep**表示，它们是相等的。

(7) 从与提升有关的操作选择中，我们可简化

$$\begin{aligned}[\![\mathbf{rec}\ f.\ (\lambda x.\ t)]\!]\rho &=_{def} \lfloor \mu r.\ \text{`}(\lambda v.\ [\![t]\!]\rho[v/x, r/f])\text{'} \rfloor \\ &= \mu r.\ \lfloor \text{`}(\lambda v.\ [\![t]\!]\rho[v/x, r/f])\text{'} \rfloor \\ &= \mu r.\ [\![\lambda x.\ t]\!]\rho[r/f]\end{aligned}$$

指称语义满足下面所期望的性质。

指称只和项的自由变量有关。

引理 13.2 如果 ρ',ρ 关于 t 的自由变量是一致的,则 $[\![t]\!]\rho = [\![t]\!]\rho'$。

证明 用结构归纳法证明。 □

标准型指称值。

引理 13.3 如果 $c \in C_\tau$,则对任一环境 ρ,有 $[\![c]\!]\rho \neq \emptyset$。

证明 施结构归纳于 c 去证明。 □

13.4 活性语义的适用性

不论项的计算是否收敛,操作语义和指称语义是一致的。对有类型的封闭项 t,定义

$$t\downarrow \quad \text{当且仅当} \quad \exists c.\, t \to c$$

262

$$t\Downarrow \quad \text{当且仅当} \quad [\![t]\!]\rho \neq \emptyset$$

其中 ρ 为任意环境。所以,$t\downarrow$ 意味着封闭项 t 的求值终止于一个标准型,而 $t\Downarrow$ 意味着 t 的指称不为底元素。

借助于代入引理,按照 11.4 节的方法可以证明对指称语义遵循求值关系。

引理 13.4(代入引理) 设 s 为有类型的封闭项,使得 $[\![s]\!]\rho \neq \emptyset$,则

$$[\![t[s/x]]\!]\rho = [\![t]\!]\rho[[\![s]\!]\rho/x]$$

证明 由结构归纳法证明。 □

引理 13.5 如果 $t \to c$,则 $[\![t]\!]\rho = [\![c]\!]\rho$,其中 t 是任意有类型的封闭项,c 是标准型,ρ 是任意环境。

证明 用规则归纳法证明。 □

练习 13.6 试证明上述规则归纳法中对应于和与递归类型的规则的情形。 □

由引理 13.3,标准型指称值。所以,对任一有类型的封闭项 t,有

$$\text{如果 } t\downarrow \text{,则 } t\Downarrow$$

与通常情况相同,反方向较难证明,我们可以使用类似于第 11 章的逻辑关系去证明。但现在我们有更复杂的递归类型。在第 11 章中,我们对类型 τ 用结构归纳法定义了逻辑关系 $\lesssim_\tau$。当类型递归定义时,我们不能这样处理。我们不能直接用 $\lesssim_{\tau[\mu X.\tau/X]}$ 去定义 $\lesssim_{\mu X.\tau}$,因为这样的定义不是良基的。幸好,我们可以用信息系统的表示给出关系 $\lesssim_\tau$ 的简单定义。对符号 $a \in \mathrm{Tok}_\tau$,类型 τ,标准型 $c \in C_\tau$,用良基递归(见 10. 4 节)定义合适的关系

$$a\ \varepsilon_\tau\ c$$

对 $d \in (V_\tau)_\perp$ 和 $t : \tau$,我们取

$$d \lesssim_\tau t \text{ 当且仅当 } \forall a \in d\, \exists c \in C_\tau.\, t \to c \ \&\ a\varepsilon_\tau c$$

用符号的大小来定义关系 ε。 263

定义 对从空集开始递归构造的有限子集,第一个分量为 1 或 2 的序偶和其他序偶定义:

$$
\begin{aligned}
size(\emptyset) &= 1 \\
size(X) &= 1 + \Sigma_{a\in X}\, size(a) \qquad (X\text{ 是有限非空子集}) \\
size((a,b)) &= 1 + size(a) + size(b) \\
size((1,a)) &= 1 + size(a) \\
size((2,b)) &= 1 + size(b)
\end{aligned}
$$

引理 13.7 对每一个封闭类型 τ,在符号集合 V_τ 和标准型集合 C_τ 之间存在具有如下性质的关系 ε_τ:

- $\emptyset\ \varepsilon_1()$
- $(a,b)\varepsilon_{\tau_1*\tau_2}(c_1,c_2)$ 当且仅当 $a\ \varepsilon_{\tau_1}c_1\ \&\ b\ \varepsilon_{\tau_2}c_2$
- $(U,V)\varepsilon_{\tau_1->\tau_2}\lambda x.\,t$ 当且仅当 $\forall c\in C_{\tau_1}.\,U\lesssim_{\tau_1} c\Rightarrow V\lesssim_{\tau_2} t[c/x]$
- $(1,a)\varepsilon_{\tau_1+\tau_2}\mathbf{inl}(c)$ 当且仅当 $a\ \varepsilon_{\tau_1}\ c$
  $(2,b)\varepsilon_{\tau_1+\tau_2}\mathbf{inr}(c)$ 当且仅当 $b\ \varepsilon_{\tau_2}\ c$
- $a\varepsilon_{\mu X.\tau}\ \mathbf{abs}(c)$ 当且仅当 $a\ \varepsilon_{\tau[\mu X.\tau/X]}\ c$

其中,对 V_τ 的符号子集 U 和封闭项 $s:\tau$,有

$$U\lesssim_\tau s$$

当且仅当

$$\forall b\in U\exists c\in C_\tau.\,(b\ \varepsilon_\tau\ c\ \&\ s\to c)$$

证明 通过字典序对由符号的大小和标准型构成的序偶进行排列,由良基递归法可知,关系 ε 存在。更精确地,对符号 a,a',标准型 c,c',定义

$$
\begin{aligned}
(a,c) < (a',c') \text{ 当且仅当} & size(a) < size(a') \vee \\
& (size(a) = size(a') \text{ 且 } c \text{ 是 } c' \text{ 的真子项})
\end{aligned}
$$

该定义生成了一个良基集。对典型的成员 (a,c),用良基递归定义类型 τ,使得 $a\ \varepsilon_\tau\ c$ 成立。

□

引理 13.8 假设 t 是类型为 τ 的封闭项,$U,V\in \mathrm{Con}_\tau$,则

$$U\vdash_\tau^* V\ \&\ U\lesssim_\tau t\Rightarrow V\lesssim_\tau t$$ 264

证明 充分必要条件是:对任一 $U\in\mathrm{Con}_\tau$,$a\in\mathrm{Tok}_\tau$ 和 $c\in C_\tau$,有

$$U\vdash_\tau a\ \&\ (\forall b\in U.\,b\ \varepsilon_\tau\ c)\Rightarrow a\ \varepsilon_\tau\ c$$

施良基归纳于由 $size(U\cup\{a\})$ 以及 c 的字典序构成的结构进行证明。根据 τ 的形式来证明。

例如,设 $\tau\equiv\tau_1 -> \tau_2$,在这种情况下,假设

$$\{(X_1,Y_1),\cdots,(X_n,Y_n)\}\vdash_{\tau_1->\tau_2}(X,Y) \tag{1}$$

和

$$(X_i, Y_i)\varepsilon_{\tau_1 \to \tau_2}\lambda z.\, t \quad (1 \leqslant i \leqslant n) \tag{2}$$

为了完成归纳步骤,我们需要证明$(X,Y)\varepsilon_{\tau \to \tau_2}\lambda z.\, t$,即

$$\forall c_1 \in C_{\tau_1}.\, X \lesssim_{\tau_1} c_1 \Rightarrow Y \lesssim_{\tau_2} t[c_1/z]$$

假设$X \lesssim_{\tau_1} c_1$且$c_1 \in C_{\tau_1}$。如果$X \vdash^*_{\tau_1} X_i$,则根据良基归纳法,有$X_i \lesssim c_1$。因此由(2)可以得到$Y_i \lesssim_{\tau_2} t[c_1/z]$。于是

$$\bigcup \{Y_i \mid X \vdash^*_{\tau_1} X_i\} \lesssim_{\tau_2} t[c_1/z]$$

现在,由(1),

$$\bigcup \{Y_i \mid X \vdash^*_{\tau_1} X_i\} \vdash^*_{\tau_2} Y$$

由良基归纳法得

$$Y \lesssim_{\tau_2} t[c_1/z]$$

对τ的其他情况的证明比较简单;当$\tau \equiv \mu X.\, \sigma$时,良基归纳法依赖于字典序中第二个分量的不断下降。 □

定理 13.9 对任意有类型的封闭项t,

$$\text{如果 } t\Downarrow, \text{则 } t\downarrow$$

证明 施结构归纳于项t进行证明:

如果t有类型τ以及自由变量$x_1 : \tau_1, \cdots, x_k : \tau_k$,并且对$v_1 \in V_{\tau_1}, \cdots, v_k \in V_{\tau_k}$以及封闭项$s_1, \cdots, s_k$,

265

$$v_1 \lesssim_{\tau_1} s_1, \cdots, v_k \lesssim_{\tau_k} s_k$$

则

$$[t]\rho[v_1/x_1, \cdots, v_k/x_k] \lesssim_\tau t[s_1/x_1, \cdots, s_k/x_k]$$

我们考虑结构归纳法的两种情况,剩下的留给读者去证明。

$(t_1\ t_2)$的情况:归纳假设$t_1 : \sigma \to \tau, t_2 : \sigma$满足上述性质。假设$(t_1\ t_2)$中含有的自由变量$x_1 : \tau_1, \cdots, x_k : \tau_k$,并且对$v_1 \in V_{\tau_1}, \cdots, v_k \in V_{\tau_k}$和封闭项$s_1, \cdots, s_k$,与$v_1 \lesssim_{\tau_1} s_1, \cdots, v_k \lesssim_{\tau_k} s_k$。

假设$b \in [t_1\ t_2]\rho[v_1/x_1, \cdots]$。我们要证明存在一个标准型$c$,使得$b\ \varepsilon_\tau\ c$和$(t_1\ t_2)[s_1/x_1, \cdots] \to c$。由$[t_1\ t_2]$的指称知,对某个$U$和满足$b \in V$的$V$,有

$$U \subseteq [t_2]\rho[v_1/x_1, \cdots]\ \&\ (U, V) \in [t_1]\rho[v_1/x_1, \cdots]$$

由归纳得

$$U \lesssim_\sigma t_2[s_1/x_1, \cdots]$$

以及

$$\{(U,V)\} \lesssim_{\sigma\to\tau} t_1[s_1/x_1,\cdots]$$

由逼近映射的性质，V 非空确保 U 非空，所以存在标准型 c_2 和 $\lambda y.\ t_1$ 使得

$$U \lesssim_\sigma c_2 \ \&\ t_2[s_1/x_1,\cdots] \to c_2,\text{且}$$
$$(U,V)\ \varepsilon_{\sigma\to\tau}\ \lambda y.\ t_1' \ \&\ t_1[s_1/x_1,\cdots] \to \lambda y.\ t_1'$$

现在，由 $\varepsilon_{\sigma\to\tau}$的定义，得到

$$V \lesssim_\tau t_1'[c_2/y]$$

特别地，

$$\{b\} \lesssim_\tau t_1'[c_2/y]$$

所以，对某个标准型 c，有

$$b\ \varepsilon_\tau\ c \ \&\ t_1'[c_2/y] \to c$$

把求值关系的各种事实结合起来，由关于应用的求值的规则，得到

$$(t_1\quad t_2)[s_1/x_1,\cdots] \to c$$

266

$\lambda y.\ t$ 的情况：在有类型的抽象 $\lambda y.\ t$ 中，设 $y:\sigma$ 和 $t:\tau$。假设 $\lambda y.\ t$ 中含有的自由变量 $x_1:\tau_1,\cdots,x_k:\tau_k$，并且对 $v_1\in V_{\tau_1},\cdots,v_k\in V_{\tau_k}$和封闭项 $s_1,\cdots,s_k$，有

$$v_1 \lesssim_{\tau_1} s_1,\cdots,v_k \lesssim_{\tau_k} s_k$$

我们要证明$[\![\lambda y.\ t]\!]\rho[v_1/x_1,\cdots]$的任何符号（必有形式 U,V）满足

$$(U,V)\varepsilon_{\sigma\to\tau}(\lambda y.\ t)[s_1/x_1,\cdots]$$

假设$(U,V)\in[\![\lambda y.\ t]\!]\rho[v_1/x_1,\cdots]$。如果 $U=\emptyset$，那么 $V=\emptyset$，从而确保$(U,V)\varepsilon_{\sigma\to\tau}\ \lambda y.\ t[s_1/x_1,\cdots]$。否则，假设 $U\neq\emptyset$，由 $\varepsilon_{\sigma\to\tau}$的定义，我们要求

$$\forall c\in C_\sigma.\ U\lesssim_\tau c \Rightarrow V\lesssim_\tau t[c/y][s_1/x_1,\cdots]$$

设对某个 $c\in C_\sigma$，$U\lesssim_\tau c$。于是，由引理 13. 8 得，$\overline{U}\lesssim_\sigma c$。由 $\lambda y.\ t$ 的指称得

$$V\subseteq^{fin}[\![t]\!]\rho[v_1/x_1,\cdots][\overline{U}/y]$$

但由归纳假设得

$$[\![t]\!]\rho[v_1/x_1,\cdots][\overline{U}/y] \lesssim_\tau t[c/y][s_1/x_1,\cdots]$$

它可推出

$$V\lesssim_\tau t[c/y][s_1/x_1,\cdots]$$

所以当 $U\neq\emptyset$时，有

$$(U,V)\ \varepsilon_{\sigma\to\tau}(\lambda y.\ t)[s_1/x_1,\cdots]$$ □

练习 13. 10 试用结构归纳法证明定理 13. 9 中项为 **rec** $x.\ t$ 的情况。 □

推论 13.11 对任意有类型封闭项 t,有

$$t\downarrow \quad \text{当且仅当} \quad t\Downarrow$$

13.5 活性 λ 演算

在活性语言中,我们可以定义递归类型

267

$$\Lambda \equiv \mu X.\ (X \to X)$$

该类型指称⊴最小信息系统 $\mathscr{D}$ 使得 $\mathscr{D}=\mathscr{D}\to\mathscr{D}_\perp$——一个信息系统等于它自己的提升函数空间。从类型 Λ 开始不用 **rec** 构造的项可以很简单地描述。它们是由下式给出的项:

$$t ::= x \mid t_1.\ t_2 \mid \boldsymbol{\lambda} x.\ t$$

其中 x 是类型为 Λ 的变量。我们采用下列简写为

$$t_1.\ t_2 \equiv (\mathbf{rep}(t_1)\, t_2)$$
$$\boldsymbol{\lambda} x.\ t \equiv \mathbf{abs}(\lambda x.\ t)$$

我们用类型规则容易检验,如果 t, t_1, t_2 是类型为 Λ 的项,则应用 $t_1.\ t_2$ 和抽象 $\boldsymbol{\lambda} x.\ t$ 也是类型为 Λ 的项。这是 λ 演算的语法,在 λ 演算中,可以做像把函数应用到自己这样的事情,甚至可以定义不动点算子。

项中仅有的标准型是那些封闭的抽象 $\boldsymbol{\lambda} x.\ t$。它们求值到本身,可由下列求值规则刻画

$$\frac{}{\boldsymbol{\lambda} x.\ t \to \boldsymbol{\lambda} x.\ t} \tag{1}$$

当然,它可从操作语义中推导得出。下面介绍如何计算应用$(t_1.\ t_2)$。从操作语义中,我们得到推导

$$\frac{\dfrac{t_1 \to \mathbf{abs}(\lambda x.\ t_1') \equiv \boldsymbol{\lambda} x.\ t_1'}{\mathbf{rep}(t_1) \to \lambda x.\ t_1'} \quad t_2 \to c_2 \quad t_1'[c_2/x] \to c}{(t_1.\ t_2) \equiv (\mathbf{rep}(t_1)\, t_2) \to c}$$

它可简化为导出规则

$$\frac{t_1 \to \boldsymbol{\lambda} x.\ t_1' \quad t_2 \to c_2 \quad t_1'[c_2/x] \to c}{(t_1.\ t_2) \to c} \tag{2}$$

不难看出,确定 λ 演算中项求值的操作语义的所有推导都能从这些推导规则中构造出来。第 2 个导出规则表示应用$(t_1.\ t_2)$用活性计算方式进行求值。这些项构成了*活性 λ 演算*。

268

活性 λ 演算继承了更大的活性语言的指称语义。通过简单地限制项的指称语义,我们得到:

$$[\![x]\!]\rho = \rho(x)$$
$$[\![t_1.\ t_2]\!]\rho = [\![t_1]\!]\rho.\ [\![t_2]\!]\rho$$

其中 $\varphi \in |\mathscr{D}|_\perp$ 到 $d \in |\mathscr{D}|_\perp$ 的应用 $\varphi.\ d$ 定义为

$$\varphi.\ d =_{def} \bigcup \{V \mid \exists U \subseteq d.\ (U,V) \in \varphi\}$$

$$[\lambda x.\, t]\rho = \{(U,V) \mid \emptyset \neq U \in \mathrm{Con}_\Lambda \;\&\; V \subseteq^{fin} [t]\rho[\overline{U}/x]\} \cup \{(\emptyset,\emptyset)\}$$

如果把规则(1)和(2)作为求值规则,用上面的定义刻画指称语义(尽管环境只包括类型为 Λ 的变量),则我们可以用不同的方法从头开始定义活性 λ 演算的语法、操作语义和指称语义。通过对整个语言的适用性结果进行限制,可以得到活性 λ 演算的适用性结果:活性 λ 演算的封闭项指称不为底(即非空元素)当且仅当它对应于上述规则(1)和(2)给出的操作语义收敛。

13.5.1 等式理论

通常,我们认为相同类型的两个项是等价的当且仅当它们有相同的指称,即对相同类型的 t_1, t_2,定义

$$t_1 = t_2 \text{ 当且仅当 } [t_1] = [t_2]$$

即 t_1 和 t_2 是等价的当且仅当对任何环境 ρ,$[t_1]\rho = [t_2]\rho$。类似地,我们定义

$$t \downarrow \text{ 当且仅当 } \forall \rho.\, [t]\rho \neq \emptyset$$

有类型的项 t 使上式成立当且仅当它在任何环境中收敛。

现在我们检查哪一些规则满足 = 关系和 $\downarrow$ 关系,为了简洁,我们只考虑活性 λ 演算中的项。首先,= 关系是一个等价关系——它是自反的、对称的和传递的。= 关系也是可代入的:如果两个项有相同的指称,则在任意上下文中用一个项去代入另一个项都产生相同的指称。为了更一般地说明这样一个性质,我们需要讨论不封闭的项的代入中所涉及的问题。

代入 λ 演算中一个变量 x 在项 t 中的出现是*约束的*,如果它在形如 $\lambda x.\, t'$ 的某个子项中;否则它是*自由的*。我们用 $t[u/x]$ 表示用 u 代入 t 中每一个 x 的自由出现而得到的项。然而,代入时必须十分小心,请看下面列举的例子。$\lambda y.\, x$ 和 $\lambda w.\, x$ 表示的两个函数是任何环境中都相同的常量函数;我们有 269

$$\lambda y.\, x = \lambda w.\, x$$

但是,用 y 代入 x 的自由出现,得

$$(\lambda y.\, x)[y/x] \equiv \lambda y.\, y$$

在这种情况下,它指称恒等函数,而

$$(\lambda w.\, x)[y/x] \equiv \lambda w.\, y$$

在这种情况下,它是我们期望的常量函数。当然,

$$\lambda y.\, y = \lambda w.\, y$$

不成立。这里的问题是在第一种情况下由于代入导致自由变量 y 变成约束变量。仅当 u 的自由变量不能变成 t 中的约束变量时,代入 $t[u/x]$ 服从语义要求。

现在我们介绍与代入有关的等价规则。

等价规则:

$$(\mathrm{refl})\ \frac{}{t = t} \qquad (\mathrm{sym})\ \frac{t_1 = t_2}{t_2 = t_1} \qquad (\mathrm{tran})\ \frac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3}$$

$$(\text{eq1})\ \frac{t_1 = t_2}{t[t_1/x] = t[t_2/x]} \qquad (\text{eq2})\ \frac{t_1 = t_2 \quad t_1 \downarrow}{t_2 \downarrow}$$

上述规则的条件是 t_1, t_2 的自由变量代入后不能变成 t 的约束变量。最后一条规则说明，如果 t_1 始终收敛而且 t_1 和 t_2 有相同的指称，则 t_2 也始终收敛。

具有类型 Λ 的变量和抽象是收敛的。

收敛规则：

$$\frac{}{x \downarrow}\ \text{如果 } x \text{ 是类型 } \Lambda \text{ 的变量，} \qquad \frac{}{\lambda x.\, t \downarrow}$$

读者回想一下，在环境 ρ 下一个变量的指称是值 $\rho(x)$，它必然是收敛的。这样就解释了为什么变量始终是收敛的。

270 剩下的规则是经典 λ 演算的转换规则的变形，我们只考虑活性计算。

转换规则：

$$(\alpha)\ \frac{}{\lambda x.\, t = \lambda y.\, (t[y/x])}\quad (\text{如果 } y \text{ 不在 } t \text{ 中（自由或约束）出现})$$

$$(\beta)\ \frac{u \downarrow}{(\lambda x.\, t)u = t[u/x]}\quad (\text{如果 } u \text{ 的自由变量不会变成 } t \text{ 的约束变量})$$

$$(\eta)\ \frac{t \downarrow}{t = \lambda x.\, (t.\, x)}\quad (\text{如果 } x \text{ 不是 } t \text{ 的自由变量})$$

第一条规则(α)说明了在不引起误会的情况下，我们可以给约束变量换名。第二条规则(β)表达了活性计算的实质，应用需要先对参数求值。在检查指称语义时，最后一条规则(η)的可靠性是显然的。

练习 13.12 试由指称语义证明(η)规则的可靠性。 □

练习 13.13 试证明：如果 s 的自由变量不会变成 t_1, t_2 或 t 的约束变量，以下两条规则也是可靠的：

$$\frac{t_1 = t_2 \quad s \downarrow}{t_1[s/x] = t_2[s/x]} \qquad \frac{t \downarrow \quad s \downarrow}{t[s/x] \downarrow}$$

并解释为什么使用这些规则以及上面列举的规则系统得到的任何推导也可以用原先的系统推导得到。 □

练习 13.14 试证明以下两条“严格性”规则的可靠性：

$$\frac{t.\, u \downarrow}{t \downarrow} \qquad \frac{t.\, u \downarrow}{u \downarrow}$$

并解释为什么使用这些规则以及下面列举的规则系统得到的任何推导也可以用原先的系统推导得到。 □

271 **练习 13.15** 试给出完全活性语言（不只是活性 λ 演算）的 = 规则和 $\downarrow$ 规则。 □

13.5.2 不动点算子

活性 λ 演算像 λ 演算那样,其表达能力特别强。它可以把自然数及其上的可计算的操作等编码为活性 λ 演算中的项。特别地,它含有作用于类似不动点算子的项 $\mathscr{Y}$。

$$\mathscr{Y} \equiv \lambda f.(\lambda x.\lambda y.(f.(x.x).y)).(\lambda x.\lambda y.(f.(x.x).y))$$

(我们约定 $f.g.h$ 表示 $(f.g).h$)。设想我们把 $\mathscr{Y}$ 作用到项 $F \equiv \lambda g.(\lambda z.h)$——因此,$F$ 是以 g 为参数,返回可能包含 g 的函数 $(\lambda z.h)$。使用上一节的等价定律,我们推得

$$\begin{aligned}\mathscr{Y}.F &= (\lambda x.\lambda y.F.(x.x).y).(\lambda x.\lambda y.F.(x.x).y) \quad (\text{因为 } F\downarrow\text{,由}(\beta)\text{ 得}) \qquad (1)\\ &= \lambda y.(F.((\lambda x.\lambda y.F.(x.x).y)(\lambda x.\lambda y.F.(x.x).y)).y)\\ &\quad (\text{因为}(\lambda x.\lambda y.F.(x.x).y)\downarrow\text{,由}(\beta)\text{ 得})\\ &= \lambda y.(F.(\mathscr{Y}.F).y) \quad (\text{使用}(1)\text{,由}(eq1)\text{ 得})\end{aligned}$$

特别地,由 $(eq2)$ 得 $\mathscr{Y}.F\downarrow$。所以,由 (β) 得

$$F.(\mathscr{Y}.F) = (\lambda z.h)[\mathscr{Y}.F/g]$$

因为它是抽象 $(\lambda z.h)[\mathscr{Y}.F/g]\downarrow$,因此由 $(eq2)$ 得 $F(\mathscr{Y}.F)\downarrow$。于是由 (η) 得

$$\lambda y.(F.(\mathscr{Y}.F).y) = F.(\mathscr{Y}.F)$$

我们得到

$$\mathscr{Y}.F = F.(\mathscr{Y}.F)$$

换句话说,$\mathscr{Y}.F$ 是 F 的不动点。

练习 13.16

(i) 试证明对活性 λ 演算的任一封闭项 F,$\mathscr{Y}_1.F$ 的操作语义发散,其中

$$\mathscr{Y}_1 \equiv \lambda f.(\lambda x.f.(x.x)).(\lambda x.f.(x.x))$$

(ii) 假设 F 是活性 λ 演算的项 $\lambda g.\lambda z.h$,设

$$\mathscr{Y}' \equiv \lambda f.(\lambda x.f.(\lambda y.x.x.y)).(\lambda x.f.(\lambda y.x.x.y))$$

试证明 $\mathscr{Y}'.F = F.(\mathscr{Y}'.F)$。 □ 272

为了说明 $\mathscr{Y}$ 和最小不动点算子 fix 的关系,我们考虑 $\mathscr{Y}.f$ 的指称,其中变量 $f:\Lambda$ 在环境 ρ 中指称的值是 φ,显然 $\rho(f) = \varphi \in |\mathscr{L}|$。由 $\mathscr{L}$ 的定义,自动有 $\varphi \in |\mathscr{L} \to \mathscr{L}_\perp|$。所以 $|\varphi| : |\mathscr{L}| \to |\mathscr{L}|_\perp$。就目前而言我们尚不能得到 φ 的最小不动点。然而,注意到 $|\mathscr{L}|$ 有由下式给出底元素 $\perp_{|\mathscr{L}|}$,

$$\perp_{|\mathscr{L}|} = \{(X,\emptyset) \mid X \in \mathrm{Con}_\Lambda\}$$

所以我们能定义连续函数

$$down : |\mathscr{L}|_\perp \to |\mathscr{L}|$$

其定义为

$$down(d) = \begin{cases} d & d \in |\mathscr{L}| \\ \perp_{|\mathscr{L}|} & d = \emptyset \end{cases}$$

或者,等价地,*down* 函数描述为:对任一 $d \in |\mathscr{L}|_\perp$,

$$down(d) = d \cup \perp_{|\mathscr{L}|}$$

函数

$$down \circ |\varphi| : |\mathscr{L}| \to |\mathscr{L}|$$

有最小不动点。这是 $[\mathscr{Y}.f]\rho$ 在满足 $\rho(f) = \varphi$ 的环境 ρ 中的指称,我们宣称:

$$[\mathscr{Y}.f]\rho = fix(down \circ |\rho(f)|)$$

证明前,我们先研究在模型 $|\mathscr{L}|_\perp$ 中活性 λ 演算的应用的性质。前面我们把模型 $|\mathscr{L}|_\perp$ 中的应用定义为

$$\varphi.d = \bigcup \{V \mid \exists U \subseteq d.(U,V) \in \varphi\}$$

其中 $\varphi, d \in |\mathscr{L}|_\perp$。

引理 13.17 对 $\varphi, d \in |\mathscr{L}|_\perp$,符号 b,符号子集 V,有

$$V \subseteq^{fin} \varphi.d \Longleftrightarrow (V = \emptyset \vee \exists U \subseteq d.(U,V) \in \varphi)$$

证明 我们引用引理 12.29 中逼近映射的性质。由 $\varphi.d$ 的定义,对某些满足 (U_1,V_1), $\cdots,(U_k,V_k) \in \varphi$ 的 $U_1,\cdots,U_k \subseteq d$,我们有

273

$$V \subseteq^{fin} \varphi.d \Longleftrightarrow V = \emptyset \vee V \subseteq V_1 \cup \cdots \cup V_k$$

在后面一种情况下,令 $U = U_1 \cup \cdots \cup U_k$,因为 φ 是逼近映射,我们得到 $(U,V) \in \varphi$。 □

函数 *down* 与下列性质相关联:如果对一个项进行抽象,则该项不被求值。

引理 13.18 设活性 λ 演算项 t 不包含变量 y 的自由出现,则

$$[\lambda y.(t.y)]\rho = down([t]\rho)$$

证明 一旦我们证明,对符号 b 和任意环境 ρ,有

$$b \in [\lambda y.(t.y)]\rho \Longleftrightarrow (\exists U \in \mathrm{Con}_\Lambda.\, b = (U,\emptyset)) \vee b \in [t]\rho \qquad (\dagger)$$

可直接由 *down* 的定义证明该等式。

为此,由语义得知

$$(U,V) \in [\lambda y.(t.y)]\rho \Longleftrightarrow U = V = \emptyset \vee \emptyset \neq U \in \mathrm{Con}_\Lambda \;\&\; V \subseteq^{fin} [t.y]\rho[\overline{U}/y]$$

由下面的等价式可把上式简化成(†):

$$V \subseteq^{fin} [t.y]\rho[\overline{U}/y] \Longleftrightarrow V \subseteq^{fin} [t]\rho.\overline{U}$$

$$
\begin{aligned}
&\quad (\text{因为 } y \text{ 不是 } t \text{ 的自由变量,见引理 13.2}) \\
&\Longleftrightarrow V = \emptyset \ \vee \\
&\qquad \exists U' \subseteq \bar{U}.\ (U', V) \in [\![t]\!]\rho \quad (\text{由引理 13.17 得}) \\
&\Longleftrightarrow V = \emptyset \ \vee \\
&\qquad \exists U'.\ U \vdash^{*} U' \ \&\ (U', V) \in [\![t]\!]\rho \\
&\Longleftrightarrow V = \emptyset \ \vee \ (U, V) \in [\![t]\!]\rho \\
&\quad (\text{由逼近映射的性质})
\end{aligned}
$$

□

设 f 是类型为 Λ 的变量。由类似上面的等式推理,我们推出

$$\mathscr{Y}.f = \lambda y.f.(\mathscr{Y}.f).y \text{ 且 } \mathscr{Y}.f \downarrow$$

这样可以直接得到,对任意环境 ρ,

$$[\![\mathscr{Y}.f]\!]\rho = [\![\lambda y.f.(\mathscr{Y}.f).y]\!]\rho \neq \emptyset$$

274

所以根据引理 13. 18,由指称语义,我们得到

$$
\begin{aligned}
[\![\mathscr{Y}.f]\!]\rho &= down([\![f.(\mathscr{Y}.f)]\!]\rho) \\
&= down \circ |\rho(f)|([\![\mathscr{Y}.f]\!]\rho)
\end{aligned}
$$

因此,$[\![\mathscr{Y}.f]\!]\rho$ 是 $down \circ |\rho(f)|$ 的不动点。所以得到

$$fix(down \circ |\rho(f)|) \subseteq [\![\mathscr{Y}.f]\!]\rho$$

正如下面所宣称的那样,反过来的包含关系也成立。

定理 13.19 设

$$\mathscr{Y} \equiv \boldsymbol{\lambda} f.(\boldsymbol{\lambda} x.\boldsymbol{\lambda} y.f.(x.x).y)(\boldsymbol{\lambda} x.\boldsymbol{\lambda} y.f.(x.x).y)$$

则对任意环境 ρ,有

$$[\![\mathscr{Y}.f]\!]\rho = fix(down \circ |\rho(f)|)$$

证明 在证明过程中,我们假设一个特定的环境 ρ。对应于环境 ρ,项用它的指称标识。例如,对 $b \in [\![t]\!]\rho$,记作 $b \in t$。我们把 $fix(down \circ |\rho(f)|)$ 简记为 $Fixf$。注意 $Fixf$ 可归纳定义为最小集合 d,使得

$$d = \bigcup \{V \mid \exists U \subseteq d.\ (U, V) \in f\} \cup \bot_{|\mathscr{E}|}$$

显然,由前面的讨论可知,只剩下证明 $\mathscr{Y}.f \subseteq Fixf$。由规则($\beta$),得

$$\mathscr{Y}.f = (\boldsymbol{\lambda} x.\boldsymbol{\lambda} y.f.(x.x).y).(\boldsymbol{\lambda} x.\boldsymbol{\lambda} y.f.(x.x).y)$$

因而有

$$
\begin{aligned}
V \subseteq^{fin} \mathscr{Y}.f &\Longleftrightarrow V \subseteq^{fin} (\boldsymbol{\lambda} x.\boldsymbol{\lambda} y.f.(x.x).y).(\boldsymbol{\lambda} x.\boldsymbol{\lambda} y.f.(x.x).y) \\
&\Longleftrightarrow V = \emptyset \ \vee \\
&\qquad \exists U \subseteq (\boldsymbol{\lambda} x.\boldsymbol{\lambda} y.f.(x.x).y).(U, V) \in (\boldsymbol{\lambda} x.\boldsymbol{\lambda} y.f.(x.x).y)
\end{aligned}
$$

为了证明 $\mathscr{Y}.f \subseteq Fixf$，只要证明：对所有的 $U \in \mathrm{Con}_\Lambda$，性质 $P(U)$ 成立，其中

$$P(U) \Longleftrightarrow_{def}$$

275 $$\forall V.[U \subseteq (\lambda x.\lambda y.f.(x.x).y) \ \&\ (U,V) \in (\lambda x.\lambda y.f.(x.x).y)] \Rightarrow V \subseteq Fixf$$

这可以施归纳于 U 的大小去证明。

设 $U \in \mathrm{Con}_\Lambda$。假设对所有满足 $size(U') < size(U)$ 的 $U' \in \mathrm{Con}_\Lambda$，$P(U')$ 成立。我们要证明对任何 V，有

$$[U \subseteq (\lambda x.\lambda y.f.(x.x).y) \ \&\ (U,V) \in (\lambda x.\lambda y.f.(x.x).y)] \Rightarrow V \subseteq Fixf$$

当 V 为空时，显然成立。实际上，由下面的论证知，需要证明它不仅 V 为非空的情况成立，而且对其中 $V \cap \bot_{|\mathscr{L}|} = \emptyset$ 的情况也成立。当然，通常 $V = V_0 \cup V_1$，其中 $V_0 \cap \bot_{|\mathscr{L}|} = \emptyset$ 且 $V_1 \subseteq \bot_{|\mathscr{L}|}$。于是很显然，而由逼近映射的性质 $V_1 \subseteq Fixf$，得，$(U,V_0) \in (\lambda x.\lambda y.f.(x.x).y)$。原来的证明简化为证明

$$[U \subseteq (\lambda x.\lambda y.f.(x.x).y) \ \&\ (U,V_0) \in (\lambda x.\lambda y.f.(x.x).y)] \Rightarrow V_0 \subseteq Fixf$$

其中 $V_0 \cap \bot_{|\mathscr{L}|} = \emptyset$。

假设

$$U \subseteq (\lambda x.\lambda y.f.(x.x).y) \ \&\ (U,V) \in (\lambda x.\lambda y.f.(x.x).y)$$

其中我们假设 V 为非空且 $V \cap \bot_{|\mathscr{L}|} = \emptyset$，即 $V \cap \{(X,\emptyset) \mid X \in \mathrm{Con}_\Lambda\} = \emptyset$。在这些假设下，我们有

$$\begin{aligned}
(U,V) \in (\lambda x.\lambda y.f.(x.x).y) &\Longleftrightarrow V \subseteq [\lambda y.f.(x.x).y]\rho[\overline{U}/x] \quad (\text{由语义}) \\
&\Longleftrightarrow V \subseteq down([f.(x.x)]\rho[\overline{U}/x] \quad (\text{由引理 } 13.18) \\
&\Longleftrightarrow V \subseteq [f.(x.x)]\rho[\overline{U}/x] \cup \bot_{|\mathscr{L}|} \\
&\Longleftrightarrow V \subseteq [f.(x.x)]\rho[\overline{U}/x] \quad (\text{因为 } V \cap \bot_{|\mathscr{L}|} = \emptyset) \\
&\Longleftrightarrow V \subseteq \rho(f).(\overline{U}.\overline{U}) \quad (\text{由语义}) \\
&\Longleftrightarrow \exists W \subseteq (\overline{U}.\overline{U}).(W,V) \in f \\
&\quad (\text{因为 } V \neq \emptyset, \text{由引理 } 13.17)
\end{aligned}$$

于是，我们推导得到，存在 $W \in \mathrm{Con}_\Lambda$，使得

$$W \subseteq (\overline{U}.\overline{U}) \ \&\ (W,V) \in f$$

因为 V 非空且 (W,V) 为一个符号，所以 W 也非空。从 $W \subseteq (\overline{U}.\overline{U})$ 中，我们得到

$$\exists X \subseteq \overline{U}.(X,W) \in \overline{U}$$

即

$$\exists X.U \vdash_\Lambda^* X \ \&\ U \vdash_\Lambda (X,W)$$

但根据 $\mathscr{L} = \mathscr{L} \to \mathscr{L}_\bot$ 的推导关系的性质，这可以简化为

276 $$U \vdash_\Lambda (U,W)$$

然而,它正好表示

$$\bigcup \{Y \mid \exists Z.\, U \vdash_{\Lambda}^{*} Z \,\&\, (Z,Y) \in U\} \vdash_{\Lambda}^{*} W$$

考察任意的 Z 和 Y,使得

$$U \vdash_{\Lambda}^{*} Z \,\&\, (Z,Y) \in U$$

于是 $size(Z) < size(U)$,所以由归纳假设,$P(Z)$ 成立。由假设

$$U \subseteq (\lambda x.\, \lambda y.\, f.\, (x.\, x).\, y)$$

于是,由于指称是 $\vdash_{\Lambda}$ 封闭的,得到

$$(Z,Y) \in (\lambda x.\, \lambda y.\, f.\, (x.\, x).\, y) \text{ 且 } Z \subseteq (\lambda x.\, \lambda y.\, f.\, (x.\, x).\, y)$$

又由 $P(Z)$,我们得到 $Y \subseteq Fixf$。因为 Y,Z 是任意的,所以

$$Fixf \supseteq \bigcup \{Y \mid \exists Z.\, U \vdash_{\Lambda}^{*} Z \,\&\, (Z,Y) \in U\} \vdash_{\Lambda}^{*} W$$

又因为 $Fixf$ 是 $\vdash_{\Lambda}$ 封闭的,所以 $W \subseteq Fixf$。

回忆前面 $Fixf$ 的归纳刻画。因为

$$W \subseteq Fixf \text{ 且 } (W,V) \in f$$

所以最后得到 $V \subseteq Fixf$。这样,就完成了施归纳于 U 的大小的证明。 □

练习 13.20 设

$$\Omega \equiv (\lambda x.\, x.\, x).\, (\lambda x.\, x.\, x)$$

为活性 λ 演算的项,试证明

$$[\![\Omega]\!]\rho = \emptyset$$

即,对应于任何环境 ρ,Ω 指称 $|\mathscr{L}|_{\perp}$ 的底元素。

(提示:用证明定理 13.19 的方法证明,先假设 Ω 的指称非空,所以,有非空的 $V \subseteq^{fin} \Omega$,这等价于:对于某些 $U \in \mathrm{Con}_{\Lambda}$

$$U \subseteq (\lambda x.\, x.\, x) \,\&\, (U,V) \in (\lambda x.\, x.\, x) \qquad (\dagger)$$

然后证明

$$(U,V) \in (\lambda x.\, x.\, x) \Rightarrow U \vdash_{\Lambda} (U,V)$$

最后,通过检查 $\vdash_{\Lambda}$ 的定义得到,对于某个 V,存在一个满足性质(†)的最小元素 U,得出矛盾。) □ 277

13.6 惰性语言

在介绍惰性求值语言时,对 13.1 节的语法做些修改是合适的。类型与活性语言一样,但有一个小的修改:在惰性语言中,最小类型是 **0** (不是 **1**)。类型 **0** 没有值;类型为 **0** 的所有项

都发散。类型定义为：

$$\tau ::= \mathbf{0} \mid \tau_1 * \tau_2 \mid \tau_1 -> \tau_2 \mid \tau_1 + \tau_2 \mid X \mid \mu X.\tau$$

其中，X 是类型变量的无限集合，$\mu X.\tau$ 是递归定义的类型。用指称发散的、计算的类型为 **0** 的项 • 代替活性语言中的( )的作用，• 并不是标准型。在惰性语言中时，无类型的项的语法为：

$$\begin{aligned} t ::= \ & \bullet \mid (t_1, t_2) \mid \mathbf{fst}(t) \mid \mathbf{snd}(t) \mid \\ & x \mid \lambda x.t \mid (t_1 t_2) \mid \\ & \mathbf{inl}(t) \mid \mathbf{inr}(t) \mid \mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).t_1, \mathbf{inr}(x_2).t_2. \mid \\ & \mathbf{abs}(t) \mid \mathbf{rep}(t) \mid \\ & \mathbf{rec}\ x.t \end{aligned}$$

其中，x, x_1, x_2 是 **Var** 中的变量。和活性语言的惟一区别是用 • 代替( )，而且递归定义形式更一般。和第 11 章一样的是，惰性语言中递归的定义为 **rec** $x.t$；与活性语言不同的是，这里不坚持要求体 t 是一个抽象。

另外，任意一个封闭类型与该类型的无限多个项变量相联系。类型规则做适当的修改：

$$\frac{}{\bullet : \mathbf{0}} \qquad \frac{x : \tau \quad t : \tau}{\mathbf{rec}\ x.t \ : \tau}$$

其他的项构造的类型规则和活性语言一样。项的自由变量的定义以及封闭项的概念和以前一样。

13.7 惰性操作语义

278 类型 τ 的标准型 C_τ 由如下规则给出[⊖]：

$$\frac{t_1 : \tau_1 \quad t_2 : \tau_2 \quad t_1 \text{ 和 } t_2 \text{ 是封闭的}}{(t_1, t_2) \in C_{\tau_1 * \tau_2}}$$

$$\frac{\lambda x.t : \tau_1 -> \tau_2 \quad \lambda x.t \text{ 是封闭的}}{\lambda x.t \in C_{\tau_1 -> \tau_2}}$$

$$\frac{t_1 : \tau_1 \quad t_1 \text{ 是封闭的}}{\mathbf{inl}(t_1) \in C_{\tau_1 + \tau_2}} \qquad \frac{t_2 : \tau_2 \quad t_2 \text{ 是封闭的}}{\mathbf{inr}(t_2) \in C_{\tau_1 + \tau_2}}$$

$$\frac{c \in C_{\tau[\mu X.\tau / X]}}{\mathbf{abs}(c) \in C_{\mu X.\tau}}$$

标准型可能有未求值的分量。除了最后一条规则外，其他规则已经在第 11 章介绍过。递归类型的标准型按活性语言中的方式处理。

例 惰性自然数

考察惰性语言中的类型

⊖ 在本章剩下的部分，我们用与活性语言相同的符号表示惰性语言。由于两种情况是分开讨论的，所以不会引起混淆。

$$nat \equiv_{def} \mu X.\ (\mathbf{0} + X)$$

它的标准型与和的左分量以及和的右分量有关。

与左分量有关的标准型是

$$\mathbf{abs}(\mathbf{inl}(t_1))$$

其中 t_1 是类型为 $\mathbf{0}$ 的封闭项。事实上,存在无限多的封闭项 $t_1 : \mathbf{0}$ (请读者思考为什么?);尽管它们都指称相同的元素 $|\mathbf{0}_\perp|$,即底元素,除此之外没有别的。特别是,$\bullet : \mathbf{0}$ 指称底元素。我们用它定义

$$Zero \equiv \mathbf{abs}(\mathbf{inl}(\bullet))$$

于是,$Zero : nat$ 是一个标准型。在 $nat \equiv \mu X.\ (\mathbf{0} + X)$ 中与和的右分量有关的标准型为

$$\mathbf{abs}(\mathbf{inr}(t_2))$$

其中 t_2 是类型为 nat 的封闭项。如果我们用 $Succ(t_2)$ 简记 $\mathbf{abs}(\mathbf{inr}(t_2))$,则我们能生成标准型:

$$Zero, Succ(Zero), Succ(Succ(Zero)), \cdots$$

279

这些标准型从 $Zero$ 开始重复应用"后继函数"而得到

$$\lambda x.\ Succ(x) : nat \to nat$$

这样的标准型对应于自然数。然而,还有许多其他的标准型:由 $Succ(\mathbf{rec}\ x.\ Succ(x))$ 给出的标准型对应于一个"无限"数,而像 $Succ(Succ(\mathbf{rec}\ x.\ x))$(其中 $x : nat$)那样的标准型对应于部分自然数,这些将在下面的指称语义中讨论。 □

我们通过规则定义封闭项和标准型之间的求值关系。

求值规则

$$\frac{}{c \to c}\ \text{如果 } c \text{ 是标准型}$$

$$\frac{t \to (t_1, t_2) \quad t_1 \to c}{\mathbf{fst}(t) \to c} \qquad \frac{t \to (t_1, t_2) \quad t_2 \to c}{\mathbf{snd}(t) \to c}$$

$$\frac{t_1 \to \lambda x.\ t_1' \quad t_1'[t_2/x] \to c}{(t_1\ t_2) \to c}$$

$$\frac{t \to \mathbf{inl}(t') \quad t_1[t'/x_1] \to c}{\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).\ t_1, \mathbf{inr}(x_2).\ t_2 \to c} \qquad \frac{t \to \mathbf{inr}(t') \quad t_2[t'/x_2] \to c}{\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).\ t_1, \mathbf{inr}(x_2).\ t_2 \to c}$$

$$\frac{t \to c}{\mathbf{abs}(t) \to \mathbf{abs}(c)} \qquad \frac{t \to \mathbf{abs}(c)}{\mathbf{rep}(t) \to c}$$

$$\frac{t[\mathbf{rec}\ x.\ t/x] \to c}{\mathbf{rec}\ x.\ t \to c}$$

求值计算是确定的和类型一致的。

命题 13.21 设 t 是封闭项，c, c_1, c_2 是标准型，则

(i) $t \to c$ & $t : \tau$ 蕴涵 $c : \tau$

(ii) $t \to c_1$ & $t \to c_2$ 蕴涵 $c_1 \equiv c_2$

280 **证明** 用规则归纳法进行证明。 □

13.8 惰性指称语义

对每一个类型 τ，我们把类型 τ 的值和信息系统的元素相关联。类型 τ 可能含有自由类型变量，所以我们需要类型环境 χ，它给每一个变量都赋予一个信息系统。我们用结构归纳法定义由 τ 指称的信息系统：

$$\begin{aligned}
\nu[\mathbf{0}]\chi &= (\emptyset, \{\emptyset\}, \emptyset) \quad (\text{也称之为 } \mathbf{0}) \\
\nu[\tau_1 * \tau_2]\chi &= (\nu[\tau_1]\chi)_\perp \times (\nu[\tau_2]\chi)_\perp \\
\nu[\tau_1 -> \tau_2]\chi &= (\nu[\tau_1]\chi)_\perp \to (\nu[\tau_2]\chi)_\perp \\
\nu[\tau_1 + \tau_2]\chi &= (\nu[\tau_1]\chi)_\perp + (\nu[\tau_2]\chi)_\perp \\
\nu[X]\chi &= \chi(X) \\
\nu[\mu X.\tau]\chi &= \mu I.\ \nu[\tau]\chi[I/X]
\end{aligned}$$

所有右边的操作都是对信息系统的操作。在环境 χ 中递归类型表达式 $\mu X.\tau$ 指称到信息系统的完全偏序中

$$I \mapsto \nu[\tau]\chi[I/X]$$

的 $\sqsubseteq$ 最小不动点。

对任何类型环境 χ，一个封闭类型 τ 是由值组成的信息系统

$$\nu_\tau =_{def} \nu[\tau]\chi$$

我们把它写成

$$\nu_\tau = (\mathrm{Tok}_\tau, \mathrm{Con}_\tau, \vdash_\tau)$$

相应的值的完全偏序是 $|\nu_\tau|$。对应于项的自由变量的环境，项指称到值的提升完全偏序的元素。这一次，把类型 τ 的完全偏序表示为信息系统变得更简单，我们定义

$$\nu_{\tau_\perp} := (\nu_\tau)_\perp$$

记为

$$\boldsymbol{\nu}_{\tau_\perp} = (\mathrm{Tok}_{\tau_\perp}, \mathrm{Con}_{\tau_\perp}, \vdash_{\tau_\perp})$$

对应于环境 ρ: **Var**$\rightarrow |\boldsymbol{\nu}_{\tau_\perp}|$,类型为 τ 的项 t 指称元素

$$[t]\rho \in |\boldsymbol{\nu}_{\tau_\perp}|$$

281

我们选择 $\perp$ 和提升函数 $\lfloor - \rfloor$: $|\boldsymbol{\nu}_\tau| \rightarrow |\boldsymbol{\nu}_{\tau_\perp}|$ 的如下解释:如果我们取

$$\perp = \{\emptyset\}$$

即空集组成的单一元素,并且对所有 $x \in \boldsymbol{\nu}_\tau$,取

$$\lfloor x \rfloor = Fin(x)$$

即 x 的所有有限子集组成的集合,则满足 8.3.4 中完全偏序的提升构造所需要的条件。提升与操作 $f \mapsto f^*$ 有关,当元素 $|\mathscr{B}|$ 含底元素 $\perp_B$ 时,这个操作把连续函数 f: $|\mathscr{A}| \rightarrow |\mathscr{B}|$ 扩展到 f^*: $|\mathscr{A}_\perp| \rightarrow |\mathscr{B}|$。我们选择的提升构造导致 f^* 的以下刻画和与之紧耦合的 *let* 符号。

命题 13.22 设 $\mathscr{A}$ 和 $\mathscr{B}$ 是信息系统。假设 $|\mathscr{B}|$ 含底元素 $\perp_B$, f: $|\mathscr{A}| \rightarrow |\mathscr{B}|$ 是连续函数。它的扩展

$$f^*: |\mathscr{A}_\perp| \rightarrow |\mathscr{B}|$$

对 $x \in |\mathscr{A}_\perp|$,由下式给出

$$f^*(x) = \begin{cases} f(\cup x) & x \neq \{\emptyset\} \\ \perp_B & x = \{\emptyset\} \end{cases}$$

因此,

$$(let\ v \Leftarrow x.\ f(v)) = \begin{cases} f(\cup x) & x \neq \{\emptyset\} \\ \perp_B & x = \{\emptyset\} \end{cases}$$

证明 扩展 f^* 定义在 $x \in |\mathscr{A}_\perp|$ 上,所以

$$f^*(x) = \begin{cases} f(v) & x = \lfloor v \rfloor \\ \perp_B & x = \{\emptyset\} \end{cases}$$

然而,$x = \lfloor v \rfloor$ 等价于 $x = Fin(v)$,这蕴涵着 $v = \cup x$。注意,这里对某个 v, $x = \lfloor v \rfloor$ 和 $x \neq \{\emptyset\}$ 是一致的,所以我们得到了命题中给出的刻画。最后由定义,得

$$(let\ v \Leftarrow x.\ f(v)) = f^*(x)$$

□ 282

注记 函数 f: $|\mathscr{A}| \rightarrow |\mathscr{B}|$ 扩展到函数 f^*: $|\mathscr{A}_\perp| \rightarrow |\mathscr{B}|$ 常用在 f 是对集合的操作的情况下,其中 $f(\emptyset) = \emptyset$。这时 $f^*(x) = f(\cup x) \cup \perp_B$。

在表示指称语义时,对信息系统 $\mathscr{A}$ 和 $\mathscr{B}$,我们再一次用 $|\mathscr{A} + \mathscr{B}|$ 标识完全偏序的和

$|\mathscr{A}|+|\mathscr{B}|$,用$|\mathscr{A}\times\mathscr{B}|$标记完全偏序的积$|\mathscr{A}|\times|\mathscr{B}|$。惰性函数空间类型的处理将使用下列信息系统元素和连续函数之间的同构关系。

命题 13.23 设$\mathscr{A}$和$\mathscr{B}$是信息系统。定义

$$\|-\|:|\mathscr{A}_\perp\to\mathscr{B}_\perp|\to[|\mathscr{A}_\perp|\to|\mathscr{B}_\perp|]$$
$$\text{“-”}:[|\mathscr{A}_\perp|\to|\mathscr{B}_\perp|]\to|\mathscr{A}_\perp\to\mathscr{B}_\perp|$$

如下:

$$\|r\|=\lambda x\in|\mathscr{A}_\perp|.\{Y\mid\exists X\subseteq x.(X,Y)\in r\}$$
$$\text{“}f\text{”}=\{(X,Y)\mid\emptyset\neq X\in\mathrm{Con}_{A_\perp}\ \&\ Y\in f(\overline{X})\}\cup\{(\emptyset,\emptyset)\}$$

则$\|-\|$和“-”是互逆函数,给出同构

$$|\mathscr{A}_\perp\to\mathscr{B}_\perp|\cong[|\mathscr{A}_\perp|\to|\mathscr{B}_\perp|]$$

证明 由定理 12.30,我们有互逆函数

$$|-|:|\mathscr{A}_\perp\to\mathscr{B}_\perp|\to[|\mathscr{A}_\perp|\to|\mathscr{B}|_\perp]$$
$$\text{‘-’}:[|\mathscr{A}_\perp|\to|\mathscr{B}|_\perp]\to|\mathscr{A}_\perp\to\mathscr{B}_\perp|$$

其定义为

$$|r|=\lambda x\in|\mathscr{A}_\perp|.\bigcup\{Y\mid\exists X\subseteq x.(X,Y)\in r\}$$
$$\text{‘}f\text{’}=\{(X,Y)\mid\emptyset\neq X\in\mathrm{Con}_{A_\perp}\ \&\ Y\subseteq^{fin}f(\overline{X})\}\cup\{(\emptyset,\emptyset)\}$$

另外,存在着由互逆函数$Fin:|\mathscr{B}|_\perp\to|\mathscr{B}_\perp|$和$\cup:|\mathscr{B}_\perp|\to|\mathscr{B}|_\perp$给出的$|\mathscr{B}|_\perp$和$|\mathscr{B}_\perp|$之间的同构。所以,定义$\|r\|=Fin\circ|r|$和“$f$”=‘$\cup\circ f$’产生了$|\mathscr{A}_\perp\to\mathscr{B}_\perp|$和$[|\mathscr{A}_\perp|\to|\mathscr{B}_\perp|]$之间的同构序偶(“-”,$\|-\|$)。由$|-|$的定义,我们有

$$\begin{aligned}\|r\|(x)&=Fin(|r|(x))\\&=Fin(\bigcup\{Y\mid\exists X\subseteq x.(X,Y)\in r\})\\&=\{Y\mid\exists X\subseteq x.(X,Y)\in r\}\end{aligned}$$

283

由‘-’的定义,我们有

$$\begin{aligned}\text{“}f\text{”}&=\text{‘}\bigcup\circ f\text{’}\\&=\{(X,Y)\mid\emptyset\neq X\in\mathrm{Con}_{A_\perp}\ \&\ Y\subseteq^{fin}\bigcup f(\overline{X})\}\cup\{(\emptyset,\emptyset)\}\\&=\{(X,Y)\mid\emptyset\neq X\in\mathrm{Con}_{A_\perp}\ \&\ Y\in f(\overline{X})\}\cup\{(\emptyset,\emptyset)\}\end{aligned}$$

□

更精确地说,环境的完全偏序由

$$\rho:\mathbf{Var}\to\bigcup\{|\nu_{\tau_\perp}|\mid\tau\text{是封闭类型}\}$$

组成,其中ρ满足$\rho(x)\in|V_{type(x)_\perp}|$,并且逐点排序。我们把第 11 章(见 11.7 节)的指称语义扩充到递归类型上。我们用信息系统来定义指称语义。

指称语义

$$
\begin{array}{lll}
[\![\bullet]\!] & =_{def} & \lambda\rho.\ \{\emptyset\} \qquad (1)\\
[\![(t_1,t_2)]\!] & =_{def} & \lambda\rho.\ \lfloor([\![t_1]\!]\rho,[\![t_2]\!]\rho)\rfloor\\
 & = & \lambda\rho.\ \lfloor[\![t_1]\!]\rho\times[\![t_2]\!]\rho\rfloor \qquad (2)\\
[\![\mathbf{fst}(t)]\!] & =_{def} & \lambda\rho.\ let\ v\Leftarrow[\![t]\!]\rho.\ \pi_1(v)\\
 & = & \lambda\rho.\ (proj_1\cup[\![t]\!]\rho)\cup\{\emptyset\} \qquad (3)\\
[\![\mathbf{snd}(t)]\!] & =_{def} & \lambda\rho.\ let\ v\Leftarrow[\![t]\!]\rho.\ \pi_2(v)\\
 & = & \lambda\rho.\ (proj_2\cup[\![t]\!]\rho)\cup\{\emptyset\}\\
[\![x]\!] & =_{def} & \lambda\rho.\ \rho(x)\\
[\![\lambda x.\ t]\!] & =_{def} & \lambda\rho.\ \lfloor\text{“}(\lambda d\in|\boldsymbol{\nu}_{type(x)_\perp}|.\ [\![t]\!]\rho[d/x])\text{”}\rfloor\\
 & = & \lambda\rho.\ \lfloor\{(U,V)\mid\emptyset\neq U\in \mathrm{Con}_{type(x)_\perp}\ \&\ V\in[\![t]\!]\rho[\overline{U}/x]\}\cup\\
 & & \qquad\{(\emptyset,\emptyset)\}\rfloor \qquad (4)\\
[\![t_1t_2]\!] & =_{def} & \lambda\rho.\ let\ r\Leftarrow[\![t_1]\!]\rho.\ \|r\|([\![t_2]\!]\rho)\\
 & = & \lambda\rho.\ \{V\mid\exists U\subseteq[\![t_2]\!]\rho.\ (U,V)\in\bigcup[\![t_1]\!]\rho\}\cup\{\emptyset\} \qquad (5)\\
[\![\mathbf{inl}(t)]\!] & =_{def} & \lambda\rho.\ \lfloor in_1([\![t]\!]\rho)\rfloor\\
 & = & \lambda\rho.\ \lfloor inj_1[\![t]\!]\rho\rfloor \qquad (6)\\
[\![\mathbf{inr}(t)]\!] & =_{def} & \lambda\rho.\ \lfloor in_2([\![t]\!]\rho)\rfloor\\
 & = & \lambda\rho.\ \lfloor inj_2[\![t]\!]\rho\rfloor
\end{array}
$$

284

$$
\begin{array}{lll}
[\![\mathbf{case}\ t\ \mathbf{of}\ \mathbf{inl}(x_1).\ t_1,\mathbf{inr}(x_2).\ t_2]\!] & & \\
 & =_{def} & \lambda\rho.\ case[\![t]\!]\rho\ of\ in_1(d_1).\ [\![t_1]\!]\rho[d_1/x_1]\mid in_2(d_2).\ [\![t_2]\!]\rho[d_2/x_2]\\
[\![\mathbf{abs}(t)]\!] & =_{def} & [\![t]\!] \qquad (7)\\
[\![\mathbf{rep}(t)]\!] & =_{def} & [\![t]\!]\\
[\![\mathbf{rec}\ x.\ t]\!] & =_{def} & \lambda\rho.\ \mu d.\ [\![t]\!]\rho[d/x]
\end{array}
$$

注释

(1) 项 $\bullet$ 的指称是 $|\mathbf{0}_\perp|$ 的惟一元素,即底元素 $\{\emptyset\}$。

(2) 我们用 $[\![t_1]\!]\rho\times[\![t_2]\!]\rho$ 标识序偶 $([\![t_1]\!]\rho,[\![t_2]\!]\rho)$。

(3) 指称 $[\![\mathbf{fst}(t)]\!]\rho$ 的刻画依赖于命题 13.22。由该命题得

$$
\begin{aligned}
let\ v\Leftarrow[\![t]\!]\rho.\ \pi_1(v) &= \begin{cases}\pi_1(\bigcup[\![t]\!]\rho) & [\![t]\!]\rho\neq\{\emptyset\}\\ \{\emptyset\} & [\![t]\!]\rho=\{\emptyset\}\end{cases}\\
&= \begin{cases}proj_1\ \bigcup[\![t]\!]\rho & [\![t]\!]\rho\neq\{\emptyset\}\\ \{\emptyset\} & [\![t]\!]\rho=\{\emptyset\}\end{cases}\\
&= (proj_1\ \bigcup[\![t]\!]\rho)\cup\{\emptyset\}
\end{aligned}
$$

其中,最后一步由 $proj_1\ \emptyset=\emptyset$ 推出。

(4) 该等式由命题 13.23 得到。

(5) 由命题 13.22 中 *let* 构造的刻画得到。

$$
\begin{aligned}
let\ r \Leftarrow [t_1]\rho.\ \|r\|([t_2]\rho) &= \begin{cases} \|\bigcup [t_1]\rho\|([t_2]\rho) & [t_1]\rho \neq \{\emptyset\} \\ \{\emptyset\} & [t_1]\rho = \{\emptyset\} \end{cases} \\
&= \begin{cases} \{V \mid \exists U \subseteq [t_2]\rho.\ (U,V) \in \bigcup [t_1]\rho\} & [t_1]\rho \neq \{\emptyset\} \\ \{\emptyset\} & [t_1]\rho = \{\emptyset\} \end{cases} \\
&= \{V \mid \exists U \subseteq [t_2]\rho.\ (U,V) \in \bigcup [t_1]\rho\} \cup \{\emptyset\} \\
&\quad (\text{因为当}[t_1]\rho = \{\emptyset\}\text{ 时,第一分量为}\emptyset)
\end{aligned}
$$

(6) 和的内射 $in_1(d_1)$ 和 $in_2(d_2)$ 分别用 $inj_1 d_1$ 和 $inj_2 d_2$ 标识。

(7) 由 $\mu X.\ \tau$ 和 $\tau[\mu X.\ \tau/x]$ 指称的信息系统之间的同构的两个分量分别用 **abs** 和 **rep** 表示,它们是相等的。

285

例 惰性自然数

惰性自然数指称的信息系统

$$nat \equiv \mu X.\ (\mathbf{0} + X)$$

是

$$\mathscr{L} = \mathbf{0}_\perp + \mathscr{L}_\perp$$

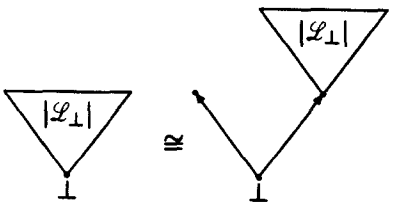
的 $\trianglelefteq$ 最小解。类型为 *nat* 的项指称 $\mathscr{L}_\perp$ 的元素,其中

$$\mathscr{L}_\perp = (\mathbf{0}_\perp + \mathscr{L}_\perp)_\perp$$

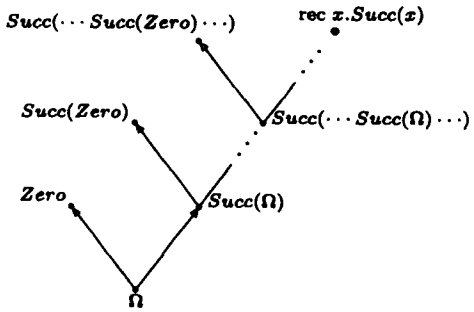
元素的完全偏序是

$$|\mathscr{L}_\perp| \cong (|\mathbf{0}|_\perp + |\mathscr{L}_\perp|)_\perp$$

我们用图表示元素的完全偏序:



实际上,完全偏序 $|\mathscr{L}_\perp|$ 形为:



上图表示了类型为 *nat* 的各种项的指称。我们把满足 x: *nat* 的项 **rec** $x.\ x$ 记为 Ω。元素

$$Zero, Succ(Zero), Succ(Succ(Zero)), \cdots$$

286

表示自然数,而 **rec** $x.\ Succ(x)$ 指称的最大元素可以认为是"无限"自然数

$$Succ(Succ(\cdots Succ\cdots))$$

它是"部分"自然数

$$\Omega, Succ(\Omega), Succ(Succ(\Omega)), \cdots$$

的最小上界。实际上,除了底元素之外,所有元素都由标准型指称——"无限自然数"是标准型 $Succ(\mathbf{rec}\ x.\ Succ(x))$ 的指称。惰性自然数上加法操作可以和活性语言中的情况一样定义。 □

练习 13.24 试解释为什么不可能把提升自然数的完全偏序 $\mathbf{N}_\perp$ 定义成与惰性语言类型相关的值或指称的完全偏序(一般惰性语言把这种类型作为基本类型)。 □

例 惰性表

设 α 是某个封闭的类型表达式,例如把 α 作为惰性自然数的类型。α 上的惰性表的类型由类型项

$$L \equiv \mu Y.\ (\mathbf{0} + \alpha * Y)$$

给出。假设 $\mathscr{A}$ 是 α 指称的信息系统。该类型项指称满足

$$\mathscr{L} = \mathbf{0}_\perp + \mathscr{A}_\perp \times \mathscr{L}_\perp$$

的 $\trianglelefteq$ 最小信息系统。类型为 L 的项指称 $D = |\mathscr{L}_\perp|$ 的元素,即 $|\mathscr{A}|_\perp$ 上的惰性表的域,其中

$$D \cong (|\mathbf{0}|_\perp + |\mathscr{A}|_\perp \times D)_\perp$$

惰性程序设计语言把常量 *Nil* 定义为标准型

$$Nil \equiv_{def} \mathbf{abs}(\mathbf{inl}(\bullet)) : L$$

把表构造符 *Cons* 定义为

$$Cons \equiv_{def} \lambda x.\ \mathbf{abs}(\mathbf{inr}(x)) : \alpha * L \to L$$

其中 x 是类型为 L 的变量。在惰性语言中我们也可以定义无限表。例如,项

$$\mathbf{rec}\ l.\ Cons(a,l)$$

定义了一个无限表,它的每一个分量是 $a:\alpha$。 □

287 **练习 13.25** 试对类型 α 上的惰性表类型的标准型进行分类,并指出它们的指称形式。 □

13.9 惰性语言的适用性

设 $t:\tau$ 是封闭项。我们称 t 的求值对应于操作语义收敛当且仅当它求值到某个标准型,即

$$t\downarrow \quad \text{当且仅当} \quad \exists c.\ t\rightarrow c$$

正如我们所期望的,如果项的指称不是完全偏序 $|V_{\tau_\perp}|$ 的底元素,则 t 收敛。我们知道,$|V_{\tau_\perp}|$ 的底元素是 $\{\emptyset\}$,这等价于:

$$t\Downarrow \quad \text{当且仅当} \quad \text{对任何环境}\rho,\ \bigcup\ [\![t]\!]\rho\neq\emptyset$$

对有类型的封闭项 t,直接可以证明 $t\downarrow$ 蕴涵 $t\Downarrow$。有关引理如下。

引理 13.26 如果 ρ 和 ρ' 在 t 的自由变量上是一致的,则 $[\![t]\!]\rho=[\![t]\!]\rho'$。

证明 施结构归纳于 t 去证明。 □

引理 13.27 如果 $c\in C_\tau$,则 $c\Downarrow$。

证明 用规则归纳法证明。 □

引理 13.28(代入引理) 设 s 是满足 $s:\sigma$ 的封闭项,x 是满足 $x:\sigma$ 的变量。假设 $t:\tau$,则 $t[s/x]:\tau$ 且 $[\![t[s/x]]\!]=[\![t]\!][[\![s]\!]/x]$。

证明 施结构归纳于 t 去证明。 □

引理 13.29 如果对任何封闭项 t、标准型 c 和任意环境 ρ,有 $t\rightarrow c$,则 $[\![t]\!]\rho=[\![c]\!]\rho$。

证明 用规则归纳法进行证明。 □

另外一个方向的证明,即对有类型的封闭项 t,$t\Downarrow$ 蕴涵 $t\downarrow$,需要使用符号子集 $\mathcal{V}_\tau$ 和标准型集合 C_τ 之间的逻辑关系 $\lesssim_\tau$。从下面的引理中,能推导出关系 ε_τ。

引理 13.30 对每一个封闭类型 τ,存在着符号 Tok_τ 和标准型 C_τ 之间的关系 ε_τ,它有如下性质:

288
- $(a,b)\ \varepsilon_{\tau_1*\tau_2}(t_1,t_2)$ 当且仅当 $a\lesssim_{\tau_1}t_1\ \&\ b\lesssim_{\tau_2}t_2$
- $(U,V)\ \varepsilon_{\tau_1\rightarrow\tau_2}\ \lambda x.\ t$ 当且仅当 (对于任何封闭项 $s:\tau_1$,$\bigcup\ U\lesssim_{\tau_1}s\Rightarrow V\lesssim_{\tau_2}t[s/x]$)
- $(1,a)\ \varepsilon_{\tau_1+\tau_2}\ \mathbf{inl}(t)$ 当且仅当 $a\lesssim_{\tau_1}t$
- $(2,b)\ \varepsilon_{\tau_1+\tau_2}\ \mathbf{inr}(t)$ 当且仅当 $b\lesssim_{\tau_2}t$
- $a\ \varepsilon_{\mu X.\tau}\ \mathbf{abs}(c)$ 当且仅当 $a\varepsilon_{\tau[\mu X.\tau/x]}c$

其中,对 $\mathcal{V}_\tau$ 的符号子集 U 和封闭项 t,我们记

$$U \lesssim_\tau t$$

当且仅当

$$\forall b \in U \exists c \in C_\tau.\ (b\ \varepsilon_\tau\ c\ \&\ t \to c)$$

证明 通过施良基归纳于按字典序排列的由符号的大小和标准型结构构成的序偶去证明该关系存在。 □

引理 13.31 对 $U \in \mathrm{Con}_{\tau_\perp}$ 和封闭项 $t: \tau$,有

$$\bigcup U \lesssim_\tau t \Rightarrow \bigcup \bar{U} \lesssim_\tau t$$

证明 对 $U \in \mathrm{Con}_{\tau_\perp}, a \in \mathrm{Tok}_{\tau_\perp}$ 和 $c \in C_\tau$,由

$$\bigcup U \lesssim_\tau c\ \&\ U \vdash_{\tau_\perp} a \Rightarrow a \lesssim_\tau c$$

可推出引理。而这可通过施良基归纳于按字典序排列的由 $size(U \cup \{a\})$ 和 c 的结构构成的序偶去证明。证明按 τ 的不同形式进行。 □

引理 13.32 对每一个有类型的封闭项 t,如果 $t\Downarrow$,则 $t\downarrow$。

证明 施结构归纳于项去证明:对所有含有自由变量 $z_1: \sigma_1, \cdots, z_k: \sigma_k$ 的项 $t: \tau$,如果对 $d_i \in |V\sigma_{i\perp}|$ 和封闭项 s_i,有 $\cup d_1 \lesssim_{\sigma_1} s_1, \cdots, \cup d_k \lesssim_{\sigma_k} s_k$,则

$$\bigcup [t]\rho[d_1/z_1, \cdots, d_k/z_k] \lesssim_\tau t[s_1/z_1, \cdots, s_k/z_k]$$

当 t 是抽象时,借助于引理 13.31 去证明。

设 t 是封闭的,则由 $\lesssim_\tau$ 的定义,得到:如果 $t\Downarrow$,即 $\cup [t]\rho \neq \emptyset$,则对某个标准型 c,有 $t \to c$。 289

□

13.10 惰性 λ 演算

在惰性语言中,我们能定义递归类型

$$\Lambda \equiv \mu X.\ (X -> X)$$

这种类型指称满足

$$\mathscr{L} = \mathscr{L}_\perp \to \mathscr{L}_\perp$$

的 $\trianglelefteq$ 最小信息系统 $\mathscr{L}$,这个信息系统等于它本身的惰性函数空间。这蕴涵着类型为 Λ 的项的指称处在完全偏序 $D = |\mathscr{L}_\perp|$ 中,D 满足

$$D \cong [D \to D]_\perp$$

和活性语言中的类似,类型 Λ 具有构成 λ 演算的项:

$$t ::= x \mid t_1.\ t_2 \mid \lambda x.\ t$$

其中 x 是类型为 Λ 的变量,这里我们再一次使用了简化符号

$$t_1 . t_2 \equiv ((\mathbf{rep}(t_1) t_2)$$
$$\boldsymbol{\lambda} x . t \equiv \mathbf{abs}(\lambda x . t)$$

我们继承了完整语言的操作语义和指称语义。项中间仅有的标准型是封闭的抽象 $\boldsymbol{\lambda} x . t$。由操作语义我们可以推导求值规则:

$$\frac{}{\boldsymbol{\lambda} x . t \to \boldsymbol{\lambda} x . t} \qquad \frac{t_1 \to \boldsymbol{\lambda} x . t_1' \quad t_1'[t_2/x] \to c}{(t_1 . t_2) \to c}$$

这两条规则足以推导求值关系 $t \to c$ 的任一实例,其中 t 是 λ 演算中的封闭项。由应用的求值方式,所以这种求值下的项形成了 λ 演算。

把指称语义限制到 λ 演算的项,我们得到:

$$[x]\rho = \rho(x)$$
$$[t_1 . t_2]\rho = [t_1]\rho . [t_2]\rho$$

其中 $\varphi \in |\mathscr{D}_\perp|$ 到 $d \in |\mathscr{D}_\perp|$ 的应用 $\varphi . d$ 定义为

$$\varphi . d =_{def} \{V \mid \exists U \subseteq d . (U,V) \in \bigcup \varphi\} \cup \{\emptyset\}$$

290 $$[\boldsymbol{\lambda} x . t]\rho = \lfloor \{(U,V) \mid \emptyset \neq U \in \mathrm{Con}_{\Lambda_\perp} \ \& \ V \in [t]\rho[\overline{U}/x]\} \cup \{(\emptyset,\emptyset)\} \rfloor$$

至此我们考察的 λ 演算中,和环境 ρ 惟一有关的是它如何把变量 x: Λ 作为元素 $|\mathscr{D}_\perp|$。

13.10.1 等式理论

我们认为惰性 λ 演算的两个项等价当且仅当它们有相同的指称,即对有相同类型的项 t_1 和 t_2,定义

$$t_1 = t_2 \text{ 当且仅当 } [t_1] = [t_2]$$

我们可以定义

$$t \downarrow \text{ 当且仅当 } \forall \rho . \ \bigcup [t_1]\rho \neq \emptyset$$

我们列出了 = 关系和 $\downarrow$ 关系成立的规则。这些规则和活性 λ 演算中规则的不同之处在于它们的变量不必收敛(因为惰性情况下变量不仅仅指称值)并且(β)转换与参数收敛无关。

等价规则:

$$(\mathrm{refl})\ \frac{}{t = t} \qquad (\mathrm{sym})\ \frac{t_1 = t_2}{t_2 = t_1} \qquad (\mathrm{tran})\ \frac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3}$$

$$(eq1)\ \frac{t_1 = t_2}{t[t_1/x] = t[t_2/x]} \qquad (eq2)\ \frac{t_1 = t_2 \quad t_1 \downarrow}{t_2 \downarrow}$$

上式中,假设 t_1 和 t_2 的自由变量不会因代入 t 而变成约束变量。

收敛规则:

$$\frac{}{\boldsymbol{\lambda} x . t \downarrow}$$

转换规则：

$$(\alpha)\ \frac{}{\lambda x.\,t = \lambda y.\,(t[y/x])}\quad (\text{如果 } y \text{ 不在 } t \text{ 中(自由或约束)出现})$$

$$(\beta)\ \frac{}{(\lambda x.\,t)u = t[u/x]}\quad (\text{如果 } u \text{ 的自由变量不会变成 } t \text{ 的约束变量})$$

$$(\eta)\ \frac{t\downarrow}{t = \lambda x.\,(t.\,x)}\quad (\text{如果 } x \text{ 不是 } t \text{ 的自由变量})$$

291

练习 13.33 试用指称语义证明规则的可靠性。 □

练习 13.34 试证明“严格性”规则

$$\frac{t.\,u\downarrow}{t\downarrow}$$

的可靠性。 □

练习 13.35 试提出在完整惰性语言中 = 关系和 $\downarrow$ 关系的规则。 □

13.10.2 不动点算子

惰性 λ 演算的不动点算子比活性 λ 演算的不动点算子要简单一些，因为它不必通过对参数进行抽象而不对该参数进行求值。定义

$$\mathscr{Y} \equiv \lambda f.\,(\lambda x.\,f.\,(x.\,x)).\,(\lambda x.\,f.\,(x.\,x))$$

由等式推理，得

$$\begin{aligned}\mathscr{Y}.f &= (\lambda x.\,f.\,(x.\,x)).\,(\lambda x.\,f(x.\,x)) \quad (\text{由}(\beta)\text{ 得}) \qquad (1)\\ &= f.\,((\lambda x.\,f.\,(x.\,x)).\,(\lambda x.\,f.\,(x.\,x))) \quad (\text{由}(\beta)\text{ 得})\\ &= f.\,(\mathscr{Y}.f) \quad (\text{由}(eq1)\text{，使用}(1)\text{ 得})\end{aligned}$$

为了理解 $\mathscr{Y}$ 的指称，我们引入使用 $|\mathscr{L}|$ 的底元素定义的函数 $down: |\mathscr{L}_\perp| \to |\mathscr{L}|$。因为

$$\mathscr{L} = \mathscr{L}_\perp \to \mathscr{L}_\perp$$

并且由约定知

$$\mathscr{L}_\perp = (\mathrm{Tok}_{\Lambda_\perp}, \mathrm{Con}_{\Lambda_\perp}, \vdash_{\Lambda_\perp})$$

$\mathscr{L}$ 的底元素是

$$\perp_{|\mathscr{L}|} = \{(U, \emptyset) \mid U \in \mathrm{Con}_{\Lambda_\perp}\}$$

定义函数 $down: |\mathscr{L}_\perp| \to |\mathscr{L}|$ 为

$$down(d) = (\bigcup d) \cup \perp_{|\mathscr{L}|}$$

引理13.36 设 $\varphi, d \in |\mathscr{L}_\perp|$，则

$$\varphi.\,d = \|down(\varphi)\|(d)$$

292

证明 设 $\varphi, d \in |\mathscr{L}_\perp|$，由 $\|\cdot\|$ 的定义，得到

$$\| down(\varphi) \| (d) = \{V \mid \exists U \subseteq d. (U,V) \in down(\varphi)\}$$

由定义,$down(\varphi) = (\cup\varphi) \cup \{(U,\emptyset) \mid U \in \mathrm{Con}_{\Lambda_\perp}\}$。因此

$$\| down(\varphi) \| (d) = \{V \mid \exists U \subseteq d. (U,V) \in \bigcup \varphi\} \cup \{\emptyset\} = \varphi. d \qquad \square$$

现在,根据引理 13.36,由事实 $\mathscr{Y}.f = f. (\mathscr{Y}.f)$ 得到

$$[\mathscr{Y}.f]\rho = \rho(f). [\mathscr{Y}.f]\rho = \| down(\rho(f)) \| ([\mathscr{Y}.f]\rho)$$

于是,$[\mathscr{Y}.f]\rho$ 是函数 $\| down(\rho(f)) \| : |\mathscr{L}_\perp| \to |\mathscr{L}_\perp|$ 的不动点,所以

$$fix(\| down(\rho(f)) \|) \subseteq [\mathscr{Y}.f]\rho$$

现在我们证明反方向也成立,所以两者相等。

定理 13.37 设

$$\mathscr{Y} \equiv \lambda f. (\lambda x. f. (x. x)). (\lambda x. f. (x. x))$$

则对任意环境 ρ,有

$$[\mathscr{Y}.f]\rho = fix(\| down(\rho(f)) \|)$$

证明 所要求证明的反方向的证明和定理 13.19 很类似。我们采用类似的简记符号。在整个证明过程中我们假设 ρ 是一个特定的环境。我们把 $fix(\| down(\rho(f)) \|)$ 简记为 $Fixf$。对应于环境 ρ,项用它的指称标识:对 $b \in [t]\rho$,记为 $b \in t$,对 $b \in \bigcup [t]\rho$,记作 $b \in \bigcup t$。

在证明前,我们注意到 $Fixf$ 可以刻画为最小的 $d \in |\mathscr{L}_\perp|$,使得 $d = \rho(f). d$,即

$$d = \{V \mid \exists U \subseteq d. (U,V) \in \bigcup f\} \cup \{\emptyset\}$$

规则(β)产生

$$\mathscr{Y}.f = (\lambda x. f. (x. x)). (\lambda x. f. (x. x))$$

293 因此,我们有

$$\begin{aligned} V \in \mathscr{Y}.f &\Longleftrightarrow V \in (\lambda x. f. (x. x)). (\lambda x. f. (x. x)) \\ &\Longleftrightarrow V = \emptyset \vee \exists U \subseteq (\lambda x. f. (x. x)). (U,V) \in \bigcup (\lambda x. f. (x. x)) \end{aligned}$$

如果 $V = \emptyset$,显然有 $V \in Fixf$,所以只需证明对所有 $U \in \mathrm{Con}_{\Lambda_\perp}$,性质 $P(U)$ 成立,其中

$$\begin{aligned} &P(U) \Longleftrightarrow_{def} \\ &\forall V. [U \subseteq (\lambda x. f. (x. x)) \ \& \ (U,V) \in \bigcup (\lambda x. f. (x. x))] \Rightarrow V \in Fixf \end{aligned}$$

这可以施归纳于 U 的大小进行证明。

设 $U \in \mathrm{Con}_{\Lambda_\perp}$。假设对所有满足 $size(U') < size(U)$ 的 $U' \in \mathrm{Con}_{\Lambda_\perp}$,归纳假设 $P(U')$ 成立。假设

$$U \subseteq (\lambda x. f. (x. x)) \ \& \ (U,V) \in \bigcup (\lambda x. f. (x. x))$$

如果 $V=\emptyset$,显然有 $V\in Fixf$。否则,假设 $V\neq\emptyset$。因为(U,V)是符号,所以 $U\neq\emptyset$。在这个假设下,我们有

$$
\begin{aligned}
(U,V)\in\bigcup(\lambda x.f.(x.x)) &\Longleftrightarrow V\in[\![f.(x.x)]\!]\rho[\bar{U}/x] \\
&\quad (\text{由 13.10 节的指称语义}) \\
&\Longleftrightarrow V\in\rho(f).(\bar{U}.\bar{U}) \quad (\text{由指称定义}) \\
&\Longleftrightarrow \exists W\subseteq(\bar{U}.\bar{U}).(W,V)\in\bigcup f
\end{aligned}
$$

于是,从假设$(U,V)\in\bigcup(\lambda x.f.(x.x))$,我们推得存在 $W\in\mathrm{Con}_{\Lambda_\perp}$,使得

$$(W,V)\in\bigcup f \text{ 且 } \forall C\in W.C\in(\bar{U}.\bar{U})$$

下面由证明 $W\subseteq Fixf$ 来证明 $V\in Fixf$。

为了证明 $W\subseteq Fixf$,设 $C\in W$。如果 $C=\emptyset$,则显然有 $C\in Fixf$,所以,仅设 $C\neq\emptyset$,由 $C\in(\bar{U}.\bar{U})$,得

$$\exists Z\subseteq\bar{U}.(Z,C)\in\bigcup\bar{U}$$

但

$$(Z,C)\in\bigcup\bar{U}\Longleftrightarrow\bigcup U\vdash_\Lambda(Z,C)$$

——信息系统上提升构造的一般性质的实例(练习 12.22)。所以

$$\exists Z.U\vdash^*_{\Lambda_\perp}Z\ \&\ \bigcup U\vdash_\Lambda(Z,C)$$ 294

因此

$$\bigcup U\vdash_\Lambda(U,C)$$

读者回想一下,Λ 指称$\mathscr{D}=\mathscr{D}_\perp\to\mathscr{D}_\perp$,它是一个信息系统的提升函数空间。由推导关系的定义得

$$\bigcup\{Y\mid\exists Z.U\vdash^*_{\Lambda_\perp}Z\ \&\ (Z,Y)\in\bigcup U\}\vdash^*_\Lambda C$$

考察满足

$$U\vdash^*_{\Lambda_\perp}Z\ \&\ (Z,Y)\in\bigcup U$$

的任意 Z,Y。于是 $size(Z)<size(U)$,所以由归纳假设得 $P(Z)$成立。根据假设

$$U\subseteq(\lambda x.f.(x.x))$$

所以由于 $\lambda x.f.(x.x)$的指称对推导关系是封闭的,

$$Z\subseteq(\lambda x.f.(x.x))$$

并且

$$(Z,Y) \in \bigcup (\lambda x.f.(x.x))$$

又由 $P(Z)$，得到 $Y \in Fixf$。因为 Y,Z 是任意的，所以

$$Fixf \supseteq \{Y \mid \exists Z.\, U \vdash^{*}_{\Lambda_\perp} Z \;\&\; (Z,Y) \in \bigcup U\} \vdash_{\Lambda_\perp} C$$

又因为 $Fixf$ 对推导关系是封闭的，所以 $C \in Fixf$。但 C 是 W 的任意成员，所以我们推得 $W \subseteq Fixf$。

从 $Fixf$ 的刻画，我们最后得到 $V \in Fixf$。这样就完成了施于 U 的大小的归纳证明。 □

13.11 进一步阅读资料

Wikström 的书[101]讲述了标准 ML 的活性语言，Bird 和 Wadler 的书[22]讲述了惰性函数式语言，它们对递归类型作了清晰、透彻的解释。适用性的证明中使用的技术与 Gordon Plotkin 的讲义中的证明联系密切——Per Martin-Löf 在它的类型理论的论域解释(1983)中和 Samson Abramsky 在[1]中都使用了类似的证明方法，同样的证明方法也用来证明含有多态类
295 型的扩充语言的适用性，可参见学生项目[17]。Plotkin 很早就在[77]中研究 λ 演算求值的不同模式。13.5.1 节中的活性 λ 演算规则本质上是 Eugenio Moggi 在[66]中的 λ_p 演算规则。Abramsky 在[1]中研究了惰性 λ 演算，13.10.1 节中惰性 λ 演算规则与 Chih-Hao Ong 在[71]中的规则相对应。[87]中也研究了惰性 λ 演算，它包含定理 13.37 的另一种证明。Andrew Pitts 的论文[76]讲述了递归论域性质的证明方法的最新进展。Barendregt 的[14]是关于 λ 演
296 算的经典著作，这方面也可以参考 Hindley 和 Seldin 的[45]。Gordon 的书[42]介绍了 λ 演算的基本知识。

第 14 章　不确定性和并行性

本章介绍不确定的和并行的(或并发的)程序和系统、它们的语义以及逻辑。我们首先从通过共享变量通信开始，从迪杰斯特拉(Dijkstra)的卫式命令语言到与 Occam 和霍尔(Hoare)的 CSP 紧密相关的语言，然后到米尔纳(Milner)的 CCS。后面的几种语言只通过同步交换值进行通信。接着介绍由含递归的简单模态逻辑组成的规范语言，以及导出检查有限状态进程是否满足规范的算法。Edinburgh-Sussex Concurrency Workbench 和 Aalborg TAV 系统是支持并行系统验证的工具。本章还介绍在并行进程的语义和逻辑方面的其他方法和一些最新的研究。

14.1　引言

介绍并行程序设计语言中一些基本问题的简单方法是通过并行复合操作扩充第 2 章中的 **IMP** 语言。对命令 c_0, c_1，它们的并行复合 $c_0 \parallel c_1$ 希望 c_0 和 c_1 一起执行，而不是偏爱哪一个命令。如果 c_0 和 c_1 处在对同一个变量赋值的位置，会发生什么情况呢？一方可能在另一方之前执行赋值。这样一方的赋值会影响另一方的执行。这意味着我们不能使用命令格局和最终状态之间的关系来精确建立并行执行命令的模型。我们必须用表示单一的不可中断的关系来刻画命令的执行关系，并允许一条命令影响和它并行执行的另一条命令的状态。

在第 2 章中，我们提到了如何考虑单一的不可中断的命令执行关系。这由为执行命令而写的规则以及表达式的求值规则来确定。假设命令的并行复合执行的规则如下：

$$\frac{\langle c_0, \sigma\rangle \rightarrow_1 \sigma'}{\langle c_0 \parallel c_1, \sigma\rangle \rightarrow_1 \langle c_1, \sigma'\rangle} \qquad \frac{\langle c_0, \sigma\rangle \rightarrow_1 \langle c_0', \sigma'\rangle}{\langle c_0 \parallel c_1, \sigma\rangle \rightarrow_1 \langle c_0' \parallel c_1, \sigma'\rangle}$$

$$\frac{\langle c_1, \sigma\rangle \rightarrow_1 \sigma'}{\langle c_0 \parallel c_1, \sigma\rangle \rightarrow_1 \langle c_0, \sigma'\rangle} \qquad \frac{\langle c_1, \sigma\rangle \rightarrow_1 \langle c_1', \sigma'\rangle}{\langle c_0 \parallel c_1, \sigma\rangle \rightarrow_1 \langle c_0 \parallel c_1', \sigma'\rangle}$$

297

先看前两条规则。它们表示命令 c_0 的单步执行如何扩展到 $c_0 \parallel c_1$ 的单步执行——这两条规则分别对应 c_0 单步执行结束和没有结束。对称地，存在并行复合右半部分的执行规则。如果并行复合的两个部分 c_0 和 c_1 有公共单元，那么它们相互影响对方的执行。它们之间通过共享存储单元进行通信。我们给出并行复合中通过共享变量进行通信的例子。

并行复合规则的对称性引入了命令的行为的不可预测性。例如，考察在初始状态下执行程序($X := 0 \parallel X := 1$)。当程序终止时，x 的值应为多少？更一般地，程序

$$(X := 0 \parallel X := 1);\ \textbf{if}\ X = 0\ \textbf{then}\ c_0\ \textbf{else}\ c_1$$

或者执行 c_0 或者执行 c_1，但我们不知道程序究竟执行哪一个。

不可预测性称为不确定性。我们用于说明不确定性的程序是人为的，可能会给人可以避免的感觉。但不确定性确实客观存在。除了程序的不确定性，人和计算机并行的工作也会导

致不确定性。在理解并行性前,我们应先理解不确定性。

14.2 卫式命令

人们惊奇地发现不确定性的严格使用可以使得算法的表示更直接。这是因为目标的实现并不依赖于几个任务的执行。日常生活中我们指示某些人做这做那,但并不关心做哪个。迪杰斯特拉的卫式命令使用不确定结构帮助程序员从复杂的解决方法中解放出来。迪杰斯特拉的语言有算术表达式 $a \in \mathbf{Aexp}$ 和布尔表达式 $b \in \mathbf{Bexp}$(这些和 **IMP** 一样)。还有两个新的语法集合——命令 c 和卫式命令 gc。它们的抽象语法由以下规则给出:

$$c ::= \mathbf{skip} \mid \mathbf{abort} \mid X := a \mid c_0;c_1 \mid \mathbf{if}\ gc\ \mathbf{fi} \mid \mathbf{do}\ gc\ \mathbf{od}$$

298

$$gc ::= b \rightarrow c \mid gc_0 \ [\!]\ gc_1$$

用来形成卫式命令 $gc_0 \ [\!]\ gc_1$ 的构造符 $[\!]$ 称为选择构造。卫式命令的典型形式为

$$(b_1 \rightarrow c_1) \ [\!]\ \cdots \ [\!]\ (b_n \rightarrow c_n)$$

在这个上下文中,我们称布尔表达式为卫式,如果卫式 b_i 计算为真,则执行相应的命令体 c_i,如果在一个状态下,没有一个卫式的结果为真,则整个卫式命令失败,在这种情况下,卫式命令不产生终止状态。否则,卫式命令不确定地执行某个卫式 b_i 为真的相应命令 c_i。我们已经介绍过 **IMP** 的 **skip**、赋值和顺序复合。新的命令 **abort** 表示从任一初始状态都不会产生终止状态。如果卫式命令 gc 不会失败,则命令 **if** gc **fi** 的执行和命令 gc 的执行一样,否则,**if** gc **fi** 像 **abort** 命令一样。命令 **do** gc **od** 表示如果 gc 执行成功,则重复地执行 gc,当 gc 失败时整个命令就终止;如果一开始执行失败,那么整个命令和 **skip** 一样。

现在我们用规则对命令和卫式命令做一些非形式化的解释。**Aexp** 和 **Bexp** 的求值关系和第 2 章的 **IMP** 一样。为了在后面的小节中介绍并行性,我们描述命令和卫式命令的一步执行。对命令 c 和状态 σ,命令格局形为$\langle c,\sigma\rangle$或 σ。

对卫式命令 gc 及状态 σ,卫式命令的初始格局是序偶$\langle gc,\sigma\rangle$。执行一步可能导致另一个格局或者新的称为 **fail** 的格局。执行规则如下。

命令规则:

$$\langle \mathbf{skip},\sigma\rangle \rightarrow \sigma$$

$$\frac{\langle a,\sigma\rangle \rightarrow n}{\langle X := a,\sigma\rangle \rightarrow \sigma[n/X]}$$

299

$$\frac{\langle c_0,\sigma\rangle \rightarrow \sigma'}{\langle c_0;c_1,\sigma\rangle \rightarrow \langle c_1,\sigma'\rangle} \qquad \frac{\langle c_0,\sigma\rangle \rightarrow \langle c_0',\sigma'\rangle}{\langle c_0;c_1,\sigma\rangle \rightarrow \langle c_0';c_1,\sigma'\rangle}$$

$$\frac{\langle gc,\sigma\rangle \rightarrow \langle c,\sigma'\rangle}{\langle \mathbf{if}\ gc\ \mathbf{fi},\sigma\rangle \rightarrow \langle c,\sigma'\rangle}$$

$$\frac{\langle gc,\sigma\rangle \rightarrow \mathbf{fail}}{\langle \mathbf{do}\ gc\ \mathbf{od},\sigma\rangle \rightarrow \sigma} \qquad \frac{\langle gc,\sigma\rangle \rightarrow \langle c,\sigma'\rangle}{\langle \mathbf{do}\ gc\ \mathbf{od},\sigma\rangle \rightarrow \langle c;\ \mathbf{do}\ gc\ \mathbf{od},\sigma'\rangle}$$

卫式命令规则：

$$\frac{\langle b,\sigma\rangle \to \mathbf{true}}{\langle b \to c,\sigma\rangle \to \langle c,\sigma\rangle}$$

$$\frac{\langle gc_0,\sigma\rangle \to \langle c,\sigma'\rangle}{\langle gc_0 \;[\!]\; gc_1,\sigma\rangle \to \langle c,\sigma'\rangle} \qquad \frac{\langle gc_1,\sigma\rangle \to \langle c,\sigma'\rangle}{\langle gc_0 \;[\!]\; gc_1,\sigma\rangle \to \langle c,\sigma'\rangle}$$

$$\frac{\langle b,\sigma\rangle \to \mathbf{false}}{\langle b \to c,\sigma\rangle \to \mathbf{fail}} \qquad \frac{\langle gc_0,\sigma\rangle \to \mathbf{fail}\langle gc_1,\sigma\rangle \to \mathbf{fail}}{\langle gc_0 \;[\!]\; gc_1,\sigma\rangle \to \mathbf{fail}}$$

选择 $gc_0 \;[\!]\; gc_1$ 的规则引入了不确定性——这样的卫式命令可以执行 gc_0，也可以执行 gc_1。注意这里缺少了 **abort** 命令的规则，以及卫式命令 gc 失败时 **if** gc **fi** 的规则。这种情况下，语句的执行不产生终止状态。在文献[36]中，迪杰斯特拉引入新的命令格局 **abortion** 以显式表示这种非正常终止。[⊖]

例如，把 X、Y 的最大值赋给 MAX：

if
$X \geqslant Y \to MAX := X$
$[\!]$
$Y \geqslant X \to MAX := Y$
fi

300

在传统的 **IMP** 程序中，X 和 Y 的对称性被丢失了。

求两个数的最大公约数的欧几里得算法，用卫式命令表示为：

do
$X > Y \to X := X - Y$
$[\!]$
$Y > X \to Y := Y - X$
od

和 3.3 节用 **IMP** 写的笨拙程序做比较（由于条件的两个分支语句的不对称性导致了程序的笨拙），这里的程序更具对称性。请读者参见迪杰斯特拉的著作[36]，其中有更多的以卫式命令语言设计的例子。

练习 14.1 试给出卫式命令语言的操作语义，其中规则确定了格局和最终状态之间的形如 $\langle c,\sigma\rangle \to \sigma'$ 和 $\langle gc,\sigma\rangle \to \sigma'$ 的转换。 □

练习 14.2 试解释为什么下面的程序终止：

⊖ 读者也许会感到好奇。正如语法表示的那样，规则中存在不必要的一般性。从卫式命令的规则中发现，在所涉及的转移 $\langle gc,\sigma\rangle \to \langle c,\sigma'\rangle$ 中状态保持不变，即 $\sigma = \sigma'$。所以，在所有的其前提形为转换 $\langle gc,\sigma\rangle \to \langle c,\sigma'\rangle$ 的规则中，我们可以用 σ 代替 σ'。当然，现在采用的一般性不会使我们丢失任何东西，但更重要的是，当我们扩展卫式命令使其具有更多的作用时，这种一般性是很必要的。

$$\textbf{do}\ (2 \mid X \to X := (3 \times X)/2)\ [\!]\ (3 \mid X \to X := (5 \times X)/3)\ \textbf{od}$$

其中,$3 \mid X$ 表示 3 整除 X,$(5 \times X)/3$ 表示 $5 \times X$ 除以 3。 □

练习 14.3 设 c 是命令或卫式命令,A 和 B 是关于状态的断言,称部分正确性断言 $\{A\}c\{B\}$ 是有效的,如果从 A 为真的任何状态出发且如果 c 的执行终止,则 c 终止于使 B 为真的最终状态。试写出迪杰斯特拉语言的部分正确性断言的可靠证明规则。在什么意义下,你可以期望证明规则是完备的?作为测试这些规则完备性的例子,试用这些规则来证明欧几里得算法(见练习 6.16)的部分正确性。假设初始状态下存储单元中存放正整数,如何证明它终止? □

练习 14.4 设正则命令 c 的语法如下:

$$c := \textbf{skip} \mid X := e \mid b? \mid c;c \mid c + c \mid c^*$$

其中 X 是一组存储单元,e 是整数表达式,b 是布尔表达式。状态 σ 是从存储单元到整数的函数。假设由语义函数定义了整数表达式和布尔表达式的语义,因此 $I[\![e]\!]\sigma$ 是表达式 e 在状态 σ 下计算得到的整数值,而 $B[\![b]\!]\sigma$ 是 b 在状态 σ 下计算得到的布尔值。一般正则命令 c 的含义用形如

301

$$\langle c,\sigma\rangle \to \sigma'$$

的关系给出,它表示在状态 σ 下 c 的执行产生终止状态 σ'。而关系由下列规则确定:

$$\langle \textbf{skip},\sigma\rangle \to \sigma \qquad \frac{I[\![e]\!]\sigma = n}{\langle X := e,\sigma\rangle \to \sigma[n/X]}$$

$$\frac{B[\![b]\!]\sigma = \mathit{true}}{\langle b?,\sigma\rangle \to \sigma} \qquad \frac{\langle c_0,\sigma\rangle \to \sigma'' \quad \langle c_1,\sigma''\rangle \to \sigma'}{\langle c_0;c_1,\sigma\rangle \to \sigma'}$$

$$\frac{\langle c_0,\sigma\rangle \to \sigma'}{\langle c_0 + c_1,\sigma\rangle \to \sigma'} \qquad \frac{\langle c_1,\sigma\rangle \to \sigma'}{\langle c_0 + c_1,\sigma\rangle \to \sigma'}$$

$$\langle c^*,\sigma\rangle \to \sigma \qquad \frac{\langle c,\sigma\rangle \to \sigma'' \quad \langle c^*,\sigma''\rangle \to \sigma'}{\langle c^*,\sigma\rangle \to \sigma'}$$

(i) 试写出一个和 while 循环

$$\textbf{while}\ b\ \textbf{do}\ c$$

具有相同作用的正则命令,其中 b 是布尔表达式,c 是正则规则。你的命令 C 和 while 循环在下述意义下有相同的作用,即

$$\langle C,\sigma\rangle \to \sigma' \text{ 当且仅当 } \langle \textbf{while}\ b\ \textbf{do}\ c,\sigma\rangle \to \sigma'$$

(由此可以容易得到 while 规则。)

(ii) 对两个正则命令 c_0 和 c_1,对任何状态 σ 和 σ',当 $\langle c_0,\sigma\rangle \to \sigma'$ 当且仅当 $\langle c_1,\sigma\rangle \to \sigma'$ 时,记作 $c_0 = c_1$。试对任一正则命令 c,用规则证明

$$c^* = skip + c;c^*$$

(iii) 试写出正则命令的指称语义;正则命令 c 的指称应该等于关系

$$\{(\sigma,\sigma') \mid \langle c,\sigma\rangle \to \sigma'\}$$

302

试简单描述证明该语义的确为真的方法。

(iv) 试给出关于正则命令 $b?, c_0 + c_1$ 和 c^* 的部分正确性断言的证明规则。 □

14.3 通信进程

20 世纪 70 年代后期霍尔和米尔纳各自提出了相同的通信原语。有独立存储空间的处理器系统变得越来越重要。通信原语和通信介质无关,不论它们通过共享内存或其他方式通信,人们都可以用进程来对它们建模。霍尔和米尔纳提出了用于数据交换的同步的原子动作,这些动作是通信的核心原语。

他们的描述有些不同。这里,我们假设进程通过通道和其他进程通信。我们允许隐蔽通道,这样,两个或更多的进程可以在特殊的通道上进行局部通信。一个进程可以在通道上准备输入或输出。然而,只有环境中另一个相伴进程和它一起进行输出或输入的补动作时,才能成功。这里没有自动缓存,一个输入或输出通信将延迟到对方进程执行相应的输出或输入动作。一旦通信成功,将输出进程中的值被复制到输入进程中。

现在我们给出通信进程语言的语法。语法中除了一组存储单元 $X \in \mathbf{Loc}$、布尔表达式 $b \in \mathbf{Bexp}$ 和算术表达式 $a \in \mathbf{Aexp}$ 外,我们假设:

通道名:$\alpha,\beta,\gamma,\cdots \in \mathbf{Chan}$

输入表达式:$\alpha?X$,其中 $X \in \mathbf{Loc}$

输出表达式:$\alpha!a$,其中 $a \in \mathbf{Aexp}$

命令:

$$c ::= \mathbf{skip} \mid \mathbf{abort} \mid X := a \mid \alpha?X \mid \alpha!a \mid c_0;c_1 \mid \mathbf{if}\ gc\ \mathbf{fi} \mid \mathbf{do}\ gc\ \mathbf{od} \mid c_0 \parallel c_1 \mid c\backslash\alpha$$

卫式命令:

$$gc ::= b \to c \mid b \wedge \alpha?X \to c \mid b \wedge \alpha!a \to c \mid gc_0 [\!] gc_1$$

不是所有的命令和卫式命令都是合式的。并行复合 $c_0 \parallel c_1$ 只有在 c_0 和 c_1 不含公共存储单元的情况下才是合式的。一般来说,如果命令 $c_0 \parallel c_1$ 的所有子命令是合式的,那么该命令也是合式的。限制 $c\backslash\alpha$ 表示隐蔽了通道 α,因此,α 上只有 c 的内部通信。 303

我们如何对通信进程语言的行为形式化呢?正如早期那样,状态是存储单元到它们包含的值的函数,对命令 c 和状态 σ,命令的格局形为 $\langle c,\sigma\rangle$ 或 σ。我们对单步执行形式化。考察形为

$$\langle \alpha?X;c,\sigma\rangle$$

的命令格局。它表示首先在通道 α 上为变量 X 接收同步通信的值。它的执行与否取决于并行的另一进程是否完成向通道 α 输出值。语义中应表达环境中这种可能性。我们用类似自动机理论的方法。对转换作标记。对一组标记,我们取

$$\{\alpha?n \mid \alpha \in \mathbf{Chan} \ \& \ n \in \mathbf{N}\} \cup \{\alpha!n \mid \alpha \in \mathbf{Chan} \ \& \ n \in \mathbf{N}\}$$

特别地,现在我们希望语义产生有标记的转换

$$\langle \alpha?X;c_0,\sigma\rangle \xrightarrow{\alpha?n} \langle c_0,\sigma[n/X]\rangle$$

它表示命令 $\alpha?X;c_0$ 可以在通道 α 上接收一个值 n 并且存储于 X 中,然后修改状态。$\alpha!n$ 的标记表示向通道 α 输出值 n 的能力。如果$\langle e,\sigma\rangle \to n$,则我们期望有转换

$$\langle \alpha!e;c_1,\sigma\rangle \xrightarrow{\alpha!n} \langle c_1,\sigma\rangle$$

一旦有了这些转换规则,我们希望描述两个并行执行的命令的通信的可能性:

$$\langle (\alpha?X;c_0) \parallel (\alpha!e;c_1),\sigma\rangle \to \langle c_0 \parallel c_1,\sigma[n/X]\rangle$$

这次我们没有标记转换,因为两条命令通过内部通信,而不对环境有影响。我们也希望有别的转换,毕竟在环境中可能有别的进程准备通过通道 α 发送或者接收数据。所以,为了不排除这种可能性,我们最好也包含转换

304

$$\langle (\alpha?X;c_0) \parallel (\alpha!e;c_1),\sigma\rangle \xrightarrow{\alpha?n} \langle c_0 \parallel (\alpha!e;c_1),\sigma[n/X]\rangle$$

和

$$\langle (\alpha?X;c_0) \parallel (\alpha!e;c_1),\sigma\rangle \xrightarrow{\alpha!n} \langle (\alpha?X;c_0) \parallel c_1,\sigma[n/X]\rangle$$

前者表示可能第一部分先从环境中接收到值,而不是从第二部分接收到值。后者表示第二部分发送由环境接收的值而不是由第一部分接收的值。

现在系统地使用规则给出完整的语义。我们假设算术表达式和布尔表达式具有和 **IMP** 一样的形式并继承了来自 **IMP** 的求值规则。

卫式命令和前面一样处理,但是允许在卫式中通信。卫式命令有时可能在一个状态上失败。

为了控制规则的数量,我们引入一些约定。为了统一地表示标记和未标记的转换,我们用 λ 表示 $\alpha?n$ 、$\alpha!n$ 以及空标记。另一约定是为了用同样的方法处理命令格局$\langle c,\sigma\rangle$和 σ。我们把格局 σ 看作是$\langle *,\sigma\rangle$,而把 $*$ 视为空命令。在这种约定下,$*$ 满足定律

$$*;c \equiv c;* \equiv * \parallel c \equiv c \parallel * \equiv c \text{ 和 } *;* \equiv * \parallel * \equiv (*\backslash\alpha) \equiv *$$

例如,$* \parallel c$ 表示 c 的一段语法。

命令规则

$$\langle \mathbf{skip},\sigma\rangle \to \sigma \qquad \frac{\langle a,\sigma\rangle \to n}{\langle X := a,\sigma\rangle \to \sigma[n/X]}$$

$$\langle \alpha?X,\sigma\rangle \xrightarrow{\alpha?n} \sigma[n/X] \qquad \frac{\langle a,\sigma\rangle \to n}{\langle \alpha!a,\sigma\rangle \xrightarrow{\alpha!n} \sigma}$$

$$\frac{\langle c_0,\sigma\rangle \xrightarrow{\lambda} \langle c_0',\sigma'\rangle}{\langle c_0;c_1,\sigma\rangle \xrightarrow{\lambda} \langle c_0';c_1,\sigma'\rangle}$$

$$\frac{\langle gc,\sigma\rangle \xrightarrow{\lambda} \langle c,\sigma'\rangle}{\langle \textbf{if } gc \textbf{ fi},\sigma\rangle \xrightarrow{\lambda} \langle c,\sigma'\rangle}$$

305

$$\frac{\langle gc,\sigma\rangle \xrightarrow{\lambda} \langle c,\sigma'\rangle}{\langle \textbf{do } gc \textbf{ od},\sigma\rangle \xrightarrow{\lambda} \langle c;\textbf{do } gc \textbf{ od},\sigma'\rangle} \qquad \frac{\langle gc,\sigma\rangle \rightarrow \textbf{fail}}{\langle \textbf{do } gc \textbf{ od},\sigma\rangle \rightarrow \sigma}$$

$$\frac{\langle c_0,\sigma\rangle \xrightarrow{\lambda} \langle c_0',\sigma'\rangle}{\langle c_0 \parallel c_1,\sigma\rangle \xrightarrow{\lambda} \langle c_0' \parallel c_1,\sigma'\rangle} \qquad \frac{\langle c_1,\sigma\rangle \xrightarrow{\lambda} \langle c_1',\sigma'\rangle}{\langle c_0 \parallel c_1,\sigma\rangle \xrightarrow{\lambda} \langle c_0 \parallel c_1',\sigma'\rangle}$$

$$\frac{\langle c_0,\sigma\rangle \xrightarrow{\alpha?n} \langle c_0',\sigma'\rangle \quad \langle c_1,\sigma\rangle \xrightarrow{\alpha!n} \langle c_1',\sigma\rangle}{\langle c_1 \parallel c_1,\sigma\rangle \rightarrow \langle c_0' \parallel c_1',\sigma'\rangle}$$

$$\frac{\langle c_0,\sigma\rangle \xrightarrow{\alpha!n} \langle c_0',\sigma\rangle \quad \langle c_1,\sigma\rangle \xrightarrow{\alpha?n} \langle c_1',\sigma'\rangle}{\langle c_0 \parallel c_1,\sigma\rangle \rightarrow \langle c_0' \parallel c_1',\sigma'\rangle}$$

$$\frac{\langle c,\sigma\rangle \xrightarrow{\lambda} \langle c',\sigma'\rangle}{\langle c\backslash\alpha,\sigma\rangle \xrightarrow{\lambda} \langle c'\backslash\alpha,\sigma'\rangle} \quad (\text{如果 } \lambda \equiv \alpha?n \text{ 和 } \lambda \equiv \alpha!n \text{ 都不成立。})$$

卫式命令规则

$$\frac{\langle b,\sigma\rangle \rightarrow \textbf{true}}{\langle b \rightarrow c,\sigma\rangle \rightarrow \langle c,\sigma\rangle} \qquad \frac{\langle b,\sigma\rangle \rightarrow \textbf{false}}{\langle b \rightarrow c,\sigma\rangle \rightarrow \textbf{fail}}$$

$$\frac{\langle b,\sigma\rangle \rightarrow \textbf{false}}{\langle b \wedge \alpha?X \rightarrow c,\sigma\rangle \rightarrow \textbf{fail}} \qquad \frac{\langle b,\sigma\rangle \rightarrow \textbf{false}}{\langle b \wedge \alpha!a \rightarrow c,\sigma\rangle \rightarrow \textbf{fail}}$$

$$\frac{\langle gc_0,\sigma\rangle \rightarrow \textbf{fail} \ \langle gc_1,\sigma\rangle \rightarrow \textbf{fail}}{\langle gc_0 \ [\!] \ gc_1,\sigma\rangle \rightarrow \textbf{fail}}$$

$$\frac{\langle b,\sigma\rangle \rightarrow \textbf{true}}{\langle b \wedge \alpha?X \rightarrow c,\sigma\rangle \xrightarrow{\alpha?n} \langle c,\sigma[n/X]\rangle} \qquad \frac{\langle b,\sigma\rangle \rightarrow \textbf{true} \ \langle a,\sigma\rangle \rightarrow n}{\langle b \wedge \alpha!a \rightarrow c,\sigma\rangle \xrightarrow{\alpha!n} \langle c,\sigma\rangle}$$

$$\frac{\langle gc_0,\sigma\rangle \xrightarrow{\lambda} \langle c,\sigma'\rangle}{\langle gc_0 \ [\!] \ gc_1,\sigma\rangle \xrightarrow{\lambda} \langle c,\sigma'\rangle} \qquad \frac{\langle gc_1,\sigma\rangle \xrightarrow{\lambda} \langle c,\sigma'\rangle}{\langle gc_0 \ [\!] \ gc_1,\sigma\rangle \xrightarrow{\lambda} \langle c,\sigma'\rangle}$$

306

例 下面的例子揭示了语言和进程的各种性质(在霍尔的论文[49]中有更多例子):一个进程重复地从 α 通道中接收数据,然后把数据发送到通道 β 中去:

$$\textbf{do } (\textbf{true} \wedge \alpha?X \rightarrow \beta!X) \textbf{ od}$$

能存放两个数据的缓存从 α 接受数据并且向 γ 传送数据:

$$(\textbf{do } (\textbf{true} \wedge \alpha?X \rightarrow \beta!X) \textbf{ od} \parallel \textbf{do } (\textbf{true} \wedge \beta?Y \rightarrow \gamma!Y) \textbf{ od})\backslash\beta$$

注意限制操作隐蔽通道 β,所以所有的通信都是内部的。

选择构造的一个用途是允许进程同时侦听两个通道,并且从其中一个读入;这时不确定地

从其中一个通道读入值：

$$\textbf{if}\ (\textbf{true} \wedge \alpha?X \rightarrow c_0)\ [\!]\ (\textbf{true} \wedge \beta?Y \rightarrow c_1)\ \textbf{fi}$$

假设该进程从环境中读取数据。那么如果不是 c_0 或 c_1 执行失败，则该进程不会死锁（即不会到达一个非正常终止的状态）。而下面的进程会死锁：

$$\textbf{if}\ (\textbf{true} \rightarrow (\alpha?X;c_0))\ [\!]\ (\textbf{true} \rightarrow (\beta?Y;c_1))\ \textbf{fi}$$

它自主地准备从 α 或 β 通道中接收数据。如果它选择了右边分支而其环境只能向 α 通道发送数据，那么发生死锁。其实死锁的发生更为复杂。迪杰斯特拉的“哲学家进餐问题”的死锁是由通常很难预测的复杂环境链而引起的（见文献[49]）。 □

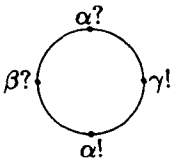
我们所考虑的程序设计语言与 Occam 紧密相关，但它不包含 Occam 的所有特性，例如 *prialt* 操作符，*prialt* 操作符类似于选择构造 [\!]，但它给左边的卫式命令的执行以更高的优先级。另一方面，它允许在卫式中有输出 $\alpha!e$，而 Occam 为了提高效率而不允许使用该语句。我们的语言和霍尔的顺序通信进程（Communicating Sequential Processes，CSP）[49]也有所不同。根本的不同是顺序通信进程用进程名代替通道名；在顺序通信进程中 $P?X$ 是从进程 P 接收值并把值放入存储单元 X 中的指令，$P!5$ 表示向进程 P 发送值 5。

307

14.4 米尔纳的 CCS

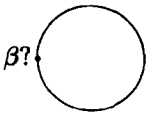
米尔纳的通信系统演算（CCS）对并行性的研究的基础有重要的影响。从上一节的语言中除去命令式特征就得到了 CCS。CCS 没有用状态来作为进程参数。实际上，CCS 的进程本身表示存储单元。

CCS 进程通过连在其端口上的通道与环境通信。进程 p 准备在 α 通道和 β 通道上输入，并准备在 α 通道和 γ 通道上输出，进程 p 在端口上适当地加以标记，表示为

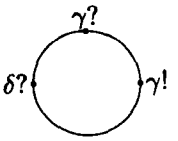


进程 q 在 α 上输入，在 β 和 δ 上输出，可以将进程 p 和进程 q 的并行复合 $p \parallel q$ 看成是含有端口 $\alpha?,\alpha!,\beta?,\beta!,\gamma!,\delta!$ 的进程。

限制操作隐蔽了指定的一组端口。例如 $p\backslash\{\alpha,\gamma\}$ 表示隐蔽了进程 p 的由一组标记 $\{\alpha,\gamma\}$ 指定的端口，这样进程只从通道 β 上输入，如下图所示。



通常复制进程是有用的，但要更改通道名。重命名函数是通道名的函数。假设重命名函数 f 为 $f(\alpha)=\gamma$，$f(\beta)=\delta$ 和 $f(\gamma)=\gamma$，则重命名后进程 p 变成与环境有如下接口的 $p[f]$：



除了在通道 α 上的通信 $\alpha?n$ 和 $\alpha!n$ 外，我们增加了动作 τ，τ 和先前的 **skip** 一样，用来表示内部通信的动作。因为我们去掉通常的赋值语句，所以我们不需要先前的状态 σ，直接使用存储单元中的变量 $x,y,\cdots$。为了命名进程，在语法中我们使用进程标识符 $P,Q,\cdots$。特别地，我们可以递归地定义进程的行为。假设算术表达式 a 和布尔表达式 b 的语法中用变量而不用存储单元。进程 $p,p_0,p_1,\cdots$ 的语法为： 308

$$
\begin{aligned}
p ::= & \mathbf{nil} \mid \\
& (\tau \to p) \mid (\alpha!a \to p) \mid (\alpha?x \to p) \mid (b \to p) \\
& p_0 + p_1 \mid p_0 \parallel p_1 \mid \\
& p\backslash L \mid p[f] \mid \\
& P(a_1,\cdots,a_k)
\end{aligned}
$$

其中，a 和 b 分别为算术表达式和布尔表达式，x 是有值的变量，L 是通道名子集，f 是重命名函数，P 表示含有参数 $a_1,\cdots,a_k$ 的进程——当参数为空时，我们就简写成 P。

$\alpha?x \to p$ 像 x 的 λ 抽象，如果 p 中变量 x 不在形如 $\beta?x \to q$ 的子项中出现，那么就受 $\alpha?x$ 约束。不受约束的变量称为*自由变量*。进程标识符 P 的定义记为：

$$P(x_1,\cdots,x_k) \stackrel{\text{def}}{=} p$$

其中，p 的所有自由变量出现于不同变量的 $x_1,\cdots,x_k$ 表中。一个进程的行为对应于该进程包含的所有进程标识符来定义。注意，进程标识符 P 可以由包含 P 的 p 递归定义。事实上，例如，如果

$$
\begin{aligned}
P(x_1,\cdots,x_k) &\stackrel{\text{def}}{=} p \\
Q(y_1,\cdots,y_l) &\stackrel{\text{def}}{=} q
\end{aligned}
$$

可以联立递归定义，其中，p 和 q 中提到 P 和 Q。

在后面给出的操作语义中，我们只规定了与不含自由变量的进程相关的转换。这样我们在操作语义中无需为变量而使用环境，用关系 $a \to n$ 和 $b \to t$ 来描述不含变量的表达式的求值。除此之外，操作语义没有其他特殊的地方。我们用 λ 来代替动作 $\alpha?n$、$\alpha!n$ 和 τ。

nil 进程 没有规则。 309

卫式进程：

$$(\tau \to p) \xrightarrow{\tau} p$$

$$\frac{a \to n}{(\alpha!a \to p) \xrightarrow{\alpha!n} p} \qquad \frac{}{(\alpha?x \to p) \xrightarrow{\alpha?n} p[n/x]}$$

$$\frac{b \rightarrow \mathbf{true} \quad p \xrightarrow{\lambda} p'}{(b \rightarrow p) \xrightarrow{\lambda} p'}$$

（所谓 $p[n/x]$ 是指 p 中用 n 来代入变量 x。更一般的代入 $p[a_1/x_1,\cdots,a_k/x_k]$ 表示在进程项 p 中用算术表达式 a_i 代入变量 x_i。）

和：

$$\frac{p_0 \xrightarrow{\lambda} p_0'}{p_0 + p_1 \xrightarrow{\lambda} p_0'} \qquad \frac{p_0 \xrightarrow{\lambda} p_1'}{p_0 + p_1 \xrightarrow{\lambda} p_1'}$$

复合：

$$\frac{p_0 \xrightarrow{\lambda} p_0'}{p_0 \parallel p_1 \xrightarrow{\lambda} p_0' \parallel p_1} \qquad \frac{p_0 \xrightarrow{\alpha?n} p_1' \quad p_1 \xrightarrow{\alpha!n} p_1'}{p_0 \parallel p_1 \xrightarrow{\tau} p_0' \parallel p_1'}$$

$$\frac{p_1 \xrightarrow{\lambda} p_1'}{p_0 \parallel p_1 \xrightarrow{\lambda} p_0 \parallel p_1'} \qquad \frac{p_0 \xrightarrow{\alpha!n} p_0' \quad p_1 \xrightarrow{\alpha?n} p_1'}{p_0 \parallel p_1 \xrightarrow{\tau} p_0' \parallel p_1'}$$

限制：

$$\frac{p \xrightarrow{\lambda} p'}{p/L \xrightarrow{\lambda} p'/L}$$

其中，如果 $\lambda \equiv \alpha?n$ 或 $\lambda \equiv \alpha!n$，则 $\alpha \notin L$。

重命名：

$$\frac{p \xrightarrow{\lambda} p'}{p[f] \xrightarrow{f(\lambda)} p'[f]}$$

标识符：

$$\frac{p[a_1/x_1,\cdots,a_k/x_k] \xrightarrow{\lambda} p'}{P(a_1,\cdots,a_k) \xrightarrow{\lambda} p'}$$

310

其中 $P(x_1,\cdots,x_k) \stackrel{\text{def}}{=} p$。

考察不含自由变量的进程已经足够了，所以在操作语义中不考虑环境。考察进程

$$(\alpha?x \rightarrow (\alpha!x \rightarrow \mathbf{nil}))$$

该进程在通道 α 上接收值 n 然后输出 n。由规则我们直接得到

$$(\alpha?x \rightarrow (\alpha!x \rightarrow \mathbf{nil})) \xrightarrow{\alpha?n} (\alpha!x \rightarrow \mathbf{nil})[n/x]$$

即

$$(\alpha?x \rightarrow (\alpha!x \rightarrow \mathbf{nil})) \xrightarrow{\alpha?n} (\alpha!x \rightarrow \mathbf{nil})$$

于是

$$(\alpha!n \rightarrow \mathbf{nil}) \xrightarrow{\alpha!n} \mathbf{nil}$$

正如我们所见到的,当推导子进程($\alpha!x \rightarrow \mathbf{nil}$)的转换时,自由变量 x 先前已约束到一个特别的数 n。

14.5 纯 CCS

米尔纳的工作有一个更为基础的称为纯 CCS 的演算。粗略地说,它从 CCS 中删除变量而得到。

我们已假设同步通信的值是整数。当然,也可以选择指称其他类型值的表达式。但为了修改表达式的需要,具体步骤是相同的。假设值位于有限集合

$$V = \{v_1, \cdots, v_k\}$$

扩展 CCS 使其允许输入动作 $\alpha?n$,其中 α 是通道,$v \in V$。进程

$$(\alpha?n \rightarrow p)$$

首先从通道 α 中输入指定的值 v,然后执行进程 p;其行为可以用以下规则描述:

$$\overline{(\alpha?n \rightarrow p) \xrightarrow{\alpha?n} p}$$

311

不难看出,在这些假设下,$\alpha?x \rightarrow p$ 的转换与

$$(\alpha?n_1 \rightarrow p[v_1/x]) + \cdots + (\alpha?n_k \rightarrow p[v_k/x])$$

的转换相同。这两个进程的行为是相同的。这样,我们可以消去进程项中的变量。然而,当整数形成无限集合以及当值的集合为无限时,我们不能用有限和代替项 $\alpha?x \rightarrow p$。我们在进程语法中引入任意和来解决这个问题。对用集合 I 加标的一组进程项 $\{p_i \mid i \in I\}$,假设进程项取为

$$\sum_{i \in I} p_i$$

于是,即使在值的集合无限时,我们也可以写

$$\sum_{m \in \mathbf{N}} (\alpha?n \rightarrow p[m/x])$$

来代替($\alpha?x \rightarrow p$)。

当有变量 x 时,存在输入值和输出值不同的情况。一旦我们消除了变量,这种不同纯粹是形式上而已;输入动作写成 $\alpha?n$,而输出动作写成 $\alpha!n$。事实上,在纯 CCS 中,值的作用被包含在端口名中。例如 $\alpha?n$,即在端口 α 上输入值 v,被认为是纯同步的,在端口 $\alpha?n$ 上没有交换任何数据。

在纯CCS中有三类动作名。动作ℓ(对应于动作$\alpha?n$或$\alpha!n$),补动作$\bar{\ell}$(对应于$\alpha?n$的补是$\alpha!n$而$\alpha!n$的补是$\alpha?n$)和内部动作τ。对补动作,$\bar{\bar{\ell}}$和ℓ是一样的。$\bar{\bar{\ell}}$强调了输入和输出之间关系的对称性。

在纯 CCS 的语法中,我们令λ表示ℓ、$\bar{\ell}$和τ形式的动作,其中ℓ属于给定的一组动作标记。纯 CCS 的进程$p, p_0, p_1, p_i, \cdots$的项形如:

$$p ::= \mathbf{nil} \mid \lambda.p \mid \sum_{i\in I} p_i \mid (p_0 \parallel p_1) \mid p\backslash L \mid p[f] \mid P$$

项$\lambda.p$是卫式进程($\lambda \rightarrow p$)的更符合习惯的记法。我们已经引入了加标进程$\{p_i \mid i \in I\}$的和$\sum_{i\in I} p_i$。当$I=\{0,1\}$时,记为p_0+p_1。L是标记的子集。我们把补操作扩充到这样的集合$\bar{L} =_{\text{def}} \{\bar{\ell} \mid \ell \in L\}$。$f$表示动作的重命名函数。重命名函数满足$f(\bar{\ell}) = \overline{f(\ell)}$和$f(\tau)=\tau$。此外,$P$还是表示进程标识符。进程标识符$P$的定义典型的形如

312

$$P \stackrel{\text{def}}{=} p$$

如前所述,这些定义支持递归和联立递归的定义。

纯 CCS 的操作语义规则特别简单:

nil:没有规则。

卫式进程:

$$\lambda.p \xrightarrow{\lambda} p$$

和:

$$\frac{p_j \xrightarrow{\lambda} q}{\sum_{i\in I} p_i \xrightarrow{\lambda} q} j \in I$$

复合:

$$\frac{p_0 \xrightarrow{\lambda} p_0'}{p_0 \parallel p_1 \xrightarrow{\lambda} p_0' \parallel p_1} \qquad \frac{p_1 \xrightarrow{\lambda} p_1'}{p_0 \parallel p_1 \xrightarrow{\lambda} p_0 \parallel p_1'}$$

$$\frac{p_0 \xrightarrow{l} p_0' \quad p_1 \xrightarrow{\bar{l}} p_1'}{p_0 \parallel p_1 \xrightarrow{\tau} p_0' \parallel p_1'}$$

限制:

$$\frac{p \xrightarrow{\lambda} q}{p\backslash L \xrightarrow{\lambda} q\backslash L} \lambda \notin L \cup \bar{L}$$

重命名:

$$\frac{p \xrightarrow{\lambda} q}{p[f] \xrightarrow{f(\lambda)} q[f]}$$

标识：

$$\frac{p \xrightarrow{\lambda} q}{P \xrightarrow{\lambda} q}, \text{其中 } P \stackrel{\text{def}}{=} p$$

由于我们看到的其他进程语言能变换成纯 CCS，所以我们主要研究纯 CCS。现在，我们用表的形式表示如何把 CCS 中的封闭项 t 变换成保持其行为的纯 CCS 的项 $\hat{t}$。 313

$(\tau \rightarrow p)$	$\tau.\hat{p}$
$(\alpha!a \rightarrow p)$	$\overline{\alpha m}.\hat{p}$，其中 a 指称值 m
$(\alpha?x \rightarrow p)$	$\sum_{m \in \mathbf{N}}(\alpha m.p[\overline{m/x}])$
$(b \rightarrow p)$	$\hat{p}$，如果 b 指称 **true** **nil**，如果 b 指称 **false**
$p_0 + p_1$	$\hat{p}_0 + \hat{p}_1$
$p_0 \parallel p_1$	$\hat{p}_0 \parallel \hat{p}_1$
$p \backslash L$	$\hat{p} \backslash \{\alpha m \mid \alpha \in L \ \& \ m \in \mathbf{N}\}$
$P(a_1, \cdots, a_k)$	$P_{m_1, \cdots, m_k}$，其中 $a_1, \cdots, a_k$ 分别指称值 $m_1, \cdots, m_k$

在 CCS 中，进程标识符的定义为 $P(x_1, \cdots, x_k) \stackrel{\text{def}}{=} p$，其中 p 有自由变量 $x_1, \cdots, x_k$，我们在纯演算中有由 $m_1, \cdots, m_k \in \mathbf{N}$ 加标的一组定义

$$P_{m_1, \cdots, m_k} \stackrel{\text{def}}{=} p[\overline{m_1/x_1, \cdots, m_k/x_k}]$$

练习 14.5 对于封闭进程项 p 和 q，试通过证明

$$p \xrightarrow{\lambda} q \quad \text{当且仅当} \quad \hat{p} \xrightarrow{\hat{\lambda}} \hat{q}$$

来验证上面的表，其中

$$\widehat{\alpha?n} = \alpha n, \quad \widehat{\alpha!n} = \overline{\alpha n}. \qquad \square$$ 314

递归定义 在应用中，使用进程标识符及定义方程是十分有用的。但有时在研究 CCS 时，使用显式的进程递归定义比使用定义方程更加方便。于是，我们使用递归定义

$$\text{rec}(P = p)$$

而不使用诸如 $P \stackrel{\text{def}}{=} p$ 的定义方程。这些附加项的转换由以下规则给出：

$$\frac{p[\mathrm{rec}(P = p)/P] \xrightarrow{\lambda} q}{\mathrm{rec}(P = p) \xrightarrow{\lambda} q}$$

练习 14.6 试使用操作语义从进程项 $\mathrm{rec}(P = a.\ b.\ P)$ 推出上述转换系统是可达的。 □

练习 14.7 设另一进程语言的语法为：

$$p ::= 0 \mid a \mid p;p \mid p + p \mid p \times p \mid P \mid \mathrm{rec}(P = p)$$

其中，a 是集合 Σ 中的动作符号，P 是用来递归定义进程 $\mathrm{rec}(P = p)$ 的进程变量。进程依次执行动作，动作由封闭进程项和有限序列 $s \in \Sigma^*$ 之间的执行关系 $p \rightarrow s$ 精确指定；$p \rightarrow s$ 表示进程 p 完整执行一系列的动作 s。注意序列 s 可以是空序列 ε，我们使用 st 来表示串 s 和串 t 的连接。下列规则给出执行关系：

$$0 \rightarrow \varepsilon \qquad a \rightarrow a \qquad \frac{p \rightarrow s \quad q \rightarrow t}{p;q \rightarrow st}$$

$$\frac{p \rightarrow s}{p + q \rightarrow s} \qquad \frac{q \rightarrow s}{p + q \rightarrow s}$$

$$\frac{p \rightarrow s \quad q \rightarrow s}{p \times q \rightarrow s} \qquad \frac{p[\mathrm{rec}(P = p)/P] \rightarrow s}{\mathrm{rec}(P = p) \rightarrow s}$$

记号 $p[q/P]$ 表示用 q 代入 p 中所有自由出现的 P 而得到的项。

315 此外，我们给出进程的指称语义。取环境 ρ 是变量 Var 到按包含关系排序的序列集的幂集 $P(\Sigma^*)$ 的函数。我们定义：

$$\begin{aligned}
&[\![0]\!]\rho = \{\varepsilon\} \quad [\![a]\!]\rho = \{a\} \\
&[\![p;q]\!]\rho = \{st \mid s \in [\![p]\!]\rho \text{ 且 } t \in [\![q]\!]\rho\} \\
&[\![p + q]\!]\rho = [\![p]\!]\rho \cup [\![q]\!]\rho \quad [\![p \times q]\!]\rho = [\![p]\!]\rho \cap [\![q]\!]\rho \\
&[\![X]\!]\rho = \rho(X) \\
&[\![\mathrm{rec}(P = p)]\!]\rho = S = [\![p]\!]\rho[S/P] \text{ 的最小解 } S
\end{aligned}$$

记号 $\rho[S/P]$ 表示用 S 修改 P 而更新的环境 ρ。

(i) 假设 a 和 b 是动作符号，试写出任一环境中语言 $\{a,b\}^*$ 的具有指称的封闭进程。

(ii) 对所有进程项 p 和 q，q 是封闭的，ρ 为环境，试用结构归纳法证明：

$$[\![p[q/P]]\!]\rho = [\![p]\!]\rho[[\![q]\!]\rho/P]$$

(iii) 试证明如果 $p \rightarrow s$，则 $s \in [\![p]\!]\rho$，其中 p 是封闭进程项，$s \in \Sigma^*$，ρ 为任一环境。明确指明你使用的归纳原理。 □

14.6 规范语言

现在我们讨论论证并行进程的方法。从历史上看，最早的论证方法是霍尔的逻辑方法。米尔纳的 CCS 开发基于等价定律的进程之间的等价的概念。如果两个进程用这些定律证明是相等的，则它们确实是等价的，在这个意义上，这些定律是可靠的。对有限状态进程，这些定

律也是完备的，即如果任何两个有限状态进程是等价的，则可以用这些定律进行证明。等价定律可以构造进程代数。进程的不同语言和不同的等价关系产生不同的进程代数。其他著名的方法可参考本章的结论。

米尔纳的等价是基于进程之间的互模拟概念的。早期，在研究互模拟的性质时，米尔纳(Milner)和亨纳斯(Hennessy)发现这种核心等价关系的逻辑性质。两个进程是互模拟的当且仅当它们正好满足模态逻辑中相同的断言(称为亨纳斯-米尔纳逻辑)。这种逻辑的语法为：

$$A := T \mid F \mid A_0 \wedge A_1 \mid A_0 \vee A_1 \mid \neg A \mid \langle \lambda \rangle A$$ 316

断言$\langle \lambda \rangle A$是模态断言（读作“diamond λ A”），它包含动作名λ，它被任何能执行一个λ动作后变成满足A的进程满足。更具体地，我们允许λ表示纯CCS的任何动作。其他形式的断言都是我们熟悉的断言。T表示真，F表示假，用合取($\wedge$)、析取($\vee$)和否定($\neg$)来构成更复杂的断言。所以，$(\neg\langle a\rangle T) \wedge (\neg\langle b\rangle T)$表示任何一个即不能执行$a$动作又不能执行$b$动作的进程。我们用逻辑定义对偶模态。取

$$[\lambda]A$$

读作“box λ A”，是$\neg\langle\lambda\rangle\neg A$的简写。该断言被任何不能通过执行$\lambda$动作后变为不能满足$A$的进程满足。换句话说，$[\lambda]A$被只要执行$\lambda$动作就满足$A$的进程满足。特别地，根本不能执行任何$\lambda$动作的进程满足这个断言。注意$[c]F$被拒绝执行$c$动作的那些进程满足。在写断言时，我们假设模态操作符$\langle a\rangle$和$[a]$比布尔操作符的约束更强，例如，$([c]F \wedge [d]F)$与断言$(([c]F) \wedge ([d]F))$相同。又例如，

$$\langle a\rangle\langle b\rangle([c]F \wedge [d]F)$$

被先执行a、b之后拒绝执行c或d动作的任一进程满足。

亨纳斯-米尔纳逻辑给出了互模拟等价的性质(见本节末尾的练习)，同时用上述有限性语言描述进程规范时有明显的缺陷；单个断言只能定义有限深度的进程的行为，而不能表达进程总是能够执行无限行为动作。为了改进，我们先考虑如何表达在分析并行进程行为中十分重要的特殊性质。

我们给出死锁进程的断言。如果进程进入非正常终止的状态，我们称该进程死锁。对死锁的含义存在几种不同的解释。例如，死锁取决于“非正常终止”是指整个进程终止还是部分进程终止。为了简化，我们假设非正常终止是指整个进程非正常终止，这样使得“非正常终止”的概念精确化。假设进程正常终止的断言是*terminal*。用显式的递归方法，施结构归纳于纯CCS的结构，这个性质的特征函数的合理定义如下： 317

$$\begin{aligned}
&terminal(\mathbf{nil}) = \mathbf{true}\\
&terminal(\lambda.p) = \mathbf{false}\\
&terminal(\sum_{i\in I} p_i) = \begin{cases}\mathbf{true}, terminal(p_i) = \mathbf{true}, \text{对于所有 } i \in I\\ \mathbf{false}, \text{其他}\end{cases}\\
&terminal(p_0 \parallel p_1) = terminal(p_0) \wedge_T terminal(p_1)\\
&terminal(p\backslash L) = terminal(p)\\
&terminal(p[f]) = terminal(p)\\
&terminal(P) = \mathbf{false}
\end{aligned}$$

$$terminal(\mathrm{rec}(P = p)) = terminal(p)$$

这已经强调了一种方法,用这种方法我们可以扩展逻辑,即加入常断言来表示正常终止的特殊进程。现在,我们说进程表示非正常终止当且仅当它不是正常终止并且不能执行任何动作。我们该如何表达这个断言呢?当然,对特殊的动作 a,断言 $[a]F$ 恰好被那些不能执行 a 的进程满足。类似地,断言

$$[a_1]F \wedge \cdots \wedge [a_k]F$$

被那些不能执行动作集合 $\{a_1, \cdots, a_k\}$ 的中任一动作的进程满足。但这样的动作集合是已知的有限集合,我们不能写出恰好被能执行(或不能执行)任何动作的那些进程满足的断言。这就促使我们研究断言的另一种扩充。以下形式的新的断言:

$$\langle . \rangle A$$

恰好表示能执行任意动作后变成满足 A 的进程。同样定义断言

$$[.]A \equiv_{def} \neg \langle . \rangle \neg A$$

它恰好表示只要执行一个动作,就变成满足 A 的进程精确为真。断言 $[.]F$ 被不能做任何动作的进程满足。现在直接死锁的性质可写为

318

$$Dead \equiv_{def} ([.]F \wedge \neg\ terminal)$$

断言 *Dead* 表示非正常终止的概念。如果进程执行一系列动作,它达到一个满足 *Dead* 的进程,那么该进程死锁。下面用无限析取表示死锁的可能性:

$$Dead \vee \langle . \rangle Dead \vee \langle . \rangle \langle . \rangle Dead \vee \langle . \rangle \langle . \rangle \langle . \rangle Dead \vee \cdots \vee (\langle . \rangle \cdots \langle . \rangle Dead) \vee \cdots$$

但是这不是真正的断言,因为在形成断言中只允许有限的析取。由于存在执行许多步之后才死锁的进程,因此我们不可能把该进程化简成有限的析取,那么我们该如何写断言呢?

在断言语言中我们需要另一种原语。我们选择一个新的定义断言的方法,能足以定义死锁的可能性和许多其他的性质,而不是碰到我们想表达的进一步性质时才引进一个附加的特定的原语。无限的析取看作是在刻画连续函数不动点中链的最小上界。的确,我们扩展的断言语言允许递归地定义性质。死锁的可能性用最小不动点表示为

$$\mu X. (Dead \vee \langle . \rangle X)$$

它直观地展开成无限"断言"

$$Dead \vee \langle . \rangle (Dead \vee \langle . \rangle (Dead \vee \langle . \rangle (\cdots$$

更一般地,我们可以写成

$$possibly(B) \equiv_{def} \mu X. (B \vee \langle . \rangle X)$$

它表示通过执行一系列的动作后,达到满足 B 的进程的那些进程。其他的性质的构造也可以表达。我们也许只对进程最终是否变成满足断言 B 有兴趣,而不管它执行什么动作序列。这能表示为

$$eventually(B) \equiv_{def} \mu X.(B \vee (\langle . \rangle T \wedge [.]X))$$

正如这个例子指出，我们并不总是清楚如何用断言定义性质。即使在下一节中我们给出递归定义性质的数学证明方法，通常也很难证明递归定义的特殊断言表达了所期望的性质。然而，对一些有用的性质，我们只需做一次。因为它们是用同样的递归机制定义的，所以这里我们集中讨论证明的方法和工具。 319

事实上，在我们以后的工作中最大不动点（而不是最小不动点）有更重要的作用。与最小不动点相反，用其他术语可以定义最大不动点。用最大不动点定义的断言可以看作是无限个合取。最大不动点 $\nu X.(B \wedge [.]X)$ 展开为

$$B \wedge [.](B \wedge [.](B \wedge [.](B \wedge \cdots$$

它被不论进程执行什么动作都满足 B 的那些进程满足。类似地，我们可以表示进程在执行无限计算序列中，都满足 B 的断言：

$$\nu X.(B \wedge \langle . \rangle X)$$

练习 14.8 试指出下面的断言表达什么意思？

(i) $\mu X.(\langle a \rangle T \vee [.]X)$

(ii) $\nu Y.(\langle a \rangle T \vee (\langle . \rangle T \wedge [.]Y))$

（通过展开定义进行非形式讨论。后面用练习 14.13 说明如何证明断言至少表达有限状态的进程的某一性质。） □

练习 14.9 在文献[63]中，米尔纳定义强互模拟为具有以下性质的 CCS 进程之间二元关系 R：如果 pRq，则

$$\text{(i)}\ \forall a,p'.\ p \xrightarrow{a} p' \Rightarrow \exists q'.\ q \xrightarrow{a} q' \text{ 且}$$

$$\text{(ii)}\ \forall a,q'.\ q \xrightarrow{a} q' \Rightarrow \exists p'.\ p \xrightarrow{a} p'$$

这样，强互模拟等价关系 ~ 定义为

$$\sim = \bigcup \{R \mid R \text{ 是强互模拟}\}$$

用亨纳斯-米尔纳逻辑可以诱导出另一个等价，包括可能的无限合取，其中断言 A 定义为

$$A := \bigwedge_{i \in I} A_i \mid \neg A \mid \langle a \rangle A$$

其中，I 是断言 A_i 的加标集合（可能为空），a 是动作。在施结构归纳于断言 A 而定义的关系 $p \models A$ 中，表示进程 p 满足断言 A 的概念被形式化： 320

$$p \models \bigwedge_{i \in I} A_i \text{ 当且仅当对于所有 } i \in I, p \models A_i$$

$$p \models \neg A \text{ 当且仅当 not } p \models A$$

$$p \models \langle a \rangle A \text{ 当且仅当对于某个 } q, p \xrightarrow{a} q \ \&\ q \models A$$

(空合取表示 **true**,因为所有的进程自动为真。)现在,我们定义 $p \asymp q$ 当且仅当对任何亨纳斯-米尔纳逻辑的断言 A 有 $(p \models A) \Leftrightarrow (q \models A)$。试证明 $\asymp$ 和强互模拟是一致的,即 $\asymp = \sim$:

(i) 施结构归纳于 A 证明

$$\forall p,q \cdot p \sim q \Rightarrow (p \models A \Leftrightarrow q \models A)$$

这样就证明了 $\asymp \supseteq \sim$。

(ii) 试证明 $\asymp$ 是强互模拟。

(由 $\sim$ 的定义可以证明 $\asymp \subseteq \sim$。提示:假设 $\asymp$ 不是互模拟,然后导出矛盾。) □

14.7 模态 ν 演算

现在,我们对 14.6 节的规范语言做形式描述。

设 $\mathscr{P}$ 表示纯 CCS 的进程集合。断言确定了进程的性质。性质对进程或者为真或者为假,所以性质可以用满足它的进程集合 $\mathscr{P}$ 的子集来标识。实际上,我们把断言理解为描述进程子集的记号。我们使用下述方法来构建断言:

- 常量:任何进程子集 $S \subseteq \mathscr{P}$ 被认为是常量断言,若一个进程属于 S,则常量断言对该进程取真,否则取假。(我们也可以使用它们的有限描述,如 *terminal* 和 *Dead*。我们把这样的描述和满足它们的进程子集同等对待。)
- 逻辑连接词:特殊常量 T 和 F 分别表示真和假。如果 A 和 B 是断言,则 $\neg A$(“非 A”),$A \wedge B$(“A 与 B”),$A \vee B$(“A 或 B”)也是断言。
- 模态算子:如果 a 是动作符号,A 是断言,则 $\langle a \rangle A$ 是断言。如果 A 是断言,则 $\langle . \rangle A$ 也是断言。(模态断言 $[a]A$ 和 $[.]A$ 分别是 $\neg \langle a \rangle \neg A$ 和 $\neg \langle . \rangle \neg A$ 的简写。)

321

- 最大不动点:如果 A 是正出现变量 X(即对 X 的每个出现,其前面的否定符号个数为偶数)的断言,则 $\nu X. A$(A 的最大不动点)是断言。(最小不动点 $\mu X. A$ 可以理解为 $\neg \nu X. \neg A[\neg X/X]$ 的简写。)

在论证断言时,我们经常使用断言的大小($size$)。断言的大小用结构归纳法定义为:

$$size(S) = size(T) = size(F) = 0, \text{其中 } S \text{ 是常量}$$
$$size(\neg A) = size(\langle a \rangle A) = size(\nu X. A) = 1 + size(A)$$
$$size(A \wedge B) = size(A \vee B) = 1 + size(A) + size(B)$$

断言是描述进程子集的记号。例如,$A \wedge B$ 表示满足断言 A 和断言 B 的那些进程,所以,$A \wedge B$ 可以取为交集 $A \cap B$。下面我们给出不同类型的断言所表示的进程子集,其中左边的断言表示右边的集合:

$$\begin{aligned}
S &= S \quad \text{其中 } S \subseteq \mathscr{P} \\
T &= \mathscr{P} \\
F &= \emptyset \\
A \wedge B &= A \cap B \\
A \vee B &= A \cup B \\
\neg A &= \mathscr{P} \backslash A
\end{aligned}$$

$$\begin{aligned}\langle a\rangle A &= \{p \in \mathscr{P} \mid \exists q.\, p \xrightarrow{a} q \text{ 且 } q \in A\}\\ \langle . \rangle A &= \{p \in \mathscr{P} \mid \exists a, q.\, p \xrightarrow{a} q \text{ 且 } q \in A\}\\ \nu X.\, A &= \bigcup \{S \subseteq \mathscr{P} \mid S \subseteq A[S/X]\}\end{aligned}$$

注意,因为与断言有关的集合是以与严格更小的断言有关的集合来定义的,所以这是一个好的定义。定义中大多数子句是显然的;例如,$\neg A$ 应该被所有不满足 A 的进程满足,$\neg A$ 也解释了为什么它取 A 的补;模态 $\langle a\rangle A$ 被执行 a 转换导致进程满足 A 的任一进程 p 满足。如果 X 只在 A 中正出现,则函数

$$S \longmapsto A[S/X]$$

对按关系 $\subseteq$ 排序的 $\mathscr{P}$ 的子集是单调的。克纳斯特-塔尔斯基定理(见 5.5 节)把该函数的最大不动点刻画成

$$\bigcup \{S \subseteq \mathscr{P} \mid S \subseteq A[S/X]\}$$

322

它是函数 $S \mapsto A[S/X]$ 的所有后缀不动点的并集。断言 $A[S/X]$ 表示用进程的子集 S 来代替 X 的所有出现。

练习 14.10 试证明最小不动点 $\mu X.\, A$ 等于 $\neg\, \nu X.\, \neg\, A[\neg\, X/X]$,其中

$$\mu X.\, A = \bigcap \{S \subseteq \mathscr{P} \mid A[S/X] \subseteq S\}$$

(提示:证明取反操作提供了函数 $S \mapsto A[S/X]$ 的前缀不动点和函数 $S \mapsto \neg\, A[\neg\, S/X]$ 的后缀不动点之间的一一对应关系。) □

练习 14.11 试证明 $[a]A = \{p \in \mathscr{P} \mid \forall q \in \mathscr{P}.\, p \xrightarrow{a} q \Rightarrow q \in A\}$。通过考察下面的例子进行证明。例如,考虑进程 $\Sigma_{n\in\omega} a.\, p_n$,其中 $p_n, n \in \omega$ 是各自不同的,证明函数 $S \mapsto [a]S$ 对应于包含关系不是连续的(但它是单调的)。 □

现在我们定义满足断言 A 的进程 p。我们定义如果 $p \in A$,则满足性断言 $p \models A$ 为 **true**,否则为 **false**。

自动检查有限状态进程 p 是否满足断言 A 是可能的。(Concurrency-Workbench/TAV 的命令之一可以检查进程 p 是否满足断言 A;对无限状态的进程,尽管给予足够的时间和空间,但也可能不终止。对有限状态进程则终止。)为了看一看这为什么是实际可行的,设 p 是有限状态进程,这意味着一组由 p 可达的进程集合

$$\mathscr{P}_p =_{def} \{q \in \mathscr{P} \mid p \xrightarrow{\cdot}{}^{*} q\}$$

是有限的,其中用 $p \xrightarrow{\cdot} q$ 表示对某动作 a 有 $p \xrightarrow{a} q$。在判定 p 是否满足断言时,我们只需考虑可达进程 $\mathscr{P}_p$ 的性质。我们用 $\mathscr{P}_p$ 代替 $\mathscr{P}$。再一次通过施归纳于断言的大小,给出如下定义:

$$
\begin{aligned}
S|_p &= S \cap \mathscr{P}_p, \text{其中 } S \subseteq \mathscr{P} \\
T|_p &= \mathscr{P}_p \\
F|_p &= \emptyset \\
A \wedge B|_p &= A|_p \cap B|_p \\
A \vee B|_p &= A|_p \cup A|_p \\
\neg A|_p &= \mathscr{P}_p \backslash (A|_p) \\
\langle a \rangle A|_p &= \{ r \in \mathscr{P}_p \mid \exists q \in \mathscr{P}_p . r \xrightarrow{a} q \text{ 且 } q \in A|_p \} \\
\langle . \rangle A|_p &= \{ r \in \mathscr{P}_p \mid \exists a, q \in \mathscr{P}_p . r \xrightarrow{a} q \text{ 且 } q \in A|_p \} \\
\nu X. A|_p &= \bigcup \{ S \subseteq \mathscr{P}_p \mid S \subseteq A[S/X]|_p \}
\end{aligned}
$$

323

幸好,下面的引理表示了断言的“全局”含义和“局部”含义之间的简单关系。

引理 14.12 对所有断言 A 和进程 p,有

$$A|_p = A \cap \mathscr{P}_p$$

证明 我们先观察

$$A[S/X]|_p = A[S \cap \mathscr{P}_p/X]|_p$$

施归纳于断言 A 的大小容易证明该观察。

进一步对断言的大小进行归纳就会产生所需的结果。我们考虑最大不动点的情况。我们要证明

$$\nu X. A|_p = (\nu X. A) \cap \mathscr{P}_p$$

假设引理表达的性质直觉上对断言 A 成立。因为

$$
\begin{aligned}
\nu X. A &= \bigcup \{ S \subseteq \mathscr{P} \mid S \subseteq A[S/X] \} \text{ 且} \\
\nu X. A|_p &= \bigcup \{ S' \subseteq \mathscr{P}_p \mid S' \subseteq A[S'/X]|_p \}
\end{aligned}
$$

假设 $S \subseteq \mathscr{P}$ 和 $S \subseteq A[S/X]$。于是

$$
\begin{aligned}
S \cap \mathscr{P}_p &\subseteq A[S/X] \cap \mathscr{P}_p \\
&= A[S/X]|_p && \text{(由归纳法)} \\
&= A[S \cap \mathscr{P}_p/X]|_p && \text{(由观察)}
\end{aligned}
$$

所以,$S \cap \mathscr{P}_p$ 是 $S' \mapsto A[S'/X]|_p$ 的后缀不动点,因而,$S \cap \mathscr{P}_p \subseteq \nu X. A|_p$。因此,$\nu X. A \cap \mathscr{P}_p \subseteq \nu X. A|_p$。

为了证明反过来也成立,假设 $S' \subseteq \mathscr{P}_p$ 且 $S' \subseteq A[S'/X]|_p$。那么,由归纳法得 $S' \subseteq A[S'/X] \cap \mathscr{P}_p$。所以 $S' \subseteq A[S'/X]$,S' 是 $S \mapsto A[S/X]$ 的后缀不动点,即 $S' \subseteq \nu X. A$。这样有 $\nu X. A|_p \subseteq \nu X. A$。

因此，我们得到 $\nu X.A\mid_p=(\nu X.A)\cap\mathscr{P}_p$。 □

对 $\mathscr{P}_p$ 限制的一个优点是对大小为 n 的有限集合，有

$$\begin{aligned}\nu X.A\mid_p &= \bigcap_{0\le i\le n} A^i[T/X]\mid_p\\ &= A^n[T/X]\cap\mathscr{P}_p\end{aligned}$$

324

其中 $A^0=T, A^{i+1}=A[A^i/X]$。从含有底元素的完全偏序上的连续函数的最小不动点的性质，我们得到：函数 $S\mapsto A[S/X]\mid_p$ 是单调的，因此，在有限完全偏序 $(\mathscr{P}ow(\mathscr{P}_p),\supseteq)$ 上是连续的——对应于这个完全偏序的最小不动点，当然按相反的关系 $\subseteq$，对应的有最大不动点。

用这种方法，最大不动点可以从我们想检验的 $p\models A$ 的断言 A 中去掉。假设去掉以后的结果为 $\langle a\rangle B$，于是我们将检验是否存在进程 q 使得 $p\xrightarrow{a}q$，并且 $q\models B$。另一方面，如果结果是合取 $B\wedge C$，我们要检验是否有 $p\models B$ 和 $p\models C$。不管断言的形式如何，一旦去掉最大不动点，我们就把检验进程是否满足断言的问题简化为检验进程是否满足严格意义下更小的断言的问题，直到简化为检验进程是否满足常量断言的问题。如果常量断言是代表判定的性质，那么用这种方法最终我们可以确定 $p\models A$ 是否成立。但这是一个很费时的方法。只有通过大量的较小规模的断言才能起到去掉最大不动点的作用。Emerson 和 Lei 在文献[37]中介绍了一个类似的、优化的有效模型检查方法。

然而，由 Stirling 和 Walker 提出了称为“局部模型检查”的方法，他们考虑了更多的断言结构，不总是找出最大不动点集 $\nu X.A\mid_p$。“局部模型检查”方法是基于 Concurrency Workbench 和 TAV 系统中算法的方法。

练习 14.13

(i) 设 $\mathscr{S}$ 是大小为 k 的有限集合，$\Phi:\mathscr{P}ow(\mathscr{S})\to\mathscr{P}ow(\mathscr{S})$ 是单调操作符。试证明

$$\begin{aligned}\mu X.\Phi(X) &= \cup_{n\in w}\Phi^n(\emptyset) = \Phi^k(\emptyset)\\ \nu X.\Phi(X) &= \cap_{n\in w}\Phi^n(\mathscr{S}) = \Phi^k(\mathscr{S})\end{aligned}$$

(ii) 设 p 是有限状态进程。试证明 p 满足 $\nu X.(\langle a\rangle X)$ 当且仅当 p 可以执行 a 转换的无限链。请问 $\mu X.(\langle a\rangle X)$ 的意思是什么？试证明之。

练习的剩下部分假设所考虑的进程是有限状态的(所以可以用(i))。读者回想一下，p 是有限状态的当且仅当集合 $\mathscr{P}_p$ 是有限的，即 p 只能到达有限个进程。

(iii) 试证明满足断言 A 的那些进程 p 满足 $\nu X.(A\wedge[.]X)$，即对所有 $q\in\mathscr{P}_p$，q 满足 A。 325

(iv) 如何用模态 ν 演算表示下列性质：最终到达满足断言 A 的状态的那些进程满足？请证明该性质。(提示：见上文或练习 14.15。) □

在下面的练习中假设进程是有限状态的。

练习 14.14

(i) 在时态逻辑中有一个复杂的模态算子，称作 until 算子。下面解释如何用进程转换系统描述 until 算子：

进程 p 满足 A until B(其中，A 和 B 都是断言)当且仅当对所有的转换序列

$$p = p_0 \dot{\rightarrow} p_1 \dot{\rightarrow} \cdots \dot{\rightarrow} p_n$$

它满足

$$\forall i(0 \leqslant i \leqslant n).\, p_i \models A$$
$$\exists i(0 \leqslant i \leqslant n).\, (p_i \models B \ \&\ \forall j(0 \leqslant j \leqslant i).\, p_j \models A)$$

试把 until 算子描述为最大不动点断言(提示:见练习 14.15)。

(ii) 下面的断言(表示所谓的"强 until")的含义是什么?

$$\mu X.\, (B \vee (A \wedge \langle . \rangle T \wedge [.]X))$$ □

练习 14.15 下面的断言的含义是什么?其中 A 和 B 是断言。

(i) $inv(A) \equiv \nu X.\, (A \wedge [.]X)$

(ii) $ev(A) \equiv \mu X.\, (A \vee (\langle . \rangle T \wedge [.]X))$

326 (iii) $un(A,B) \equiv \nu X.\, (B \vee (A \wedge [.]X))$ □

练习 14.16 称进程 p 是对应于动作 a 不公平的当且仅当存在无限的转换链

$$p = p_0 \xrightarrow{a_0} p_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} p_n \xrightarrow{a_n} \cdots$$

使得

(a) 对所有的 $i \geqslant 0$, $\exists q.\, p_i \xrightarrow{a} q$,且

(b) 对所有的 $i \geqslant 0, a_i \neq a$。

非形式地说,存在一条无限转换链,在链中 a 总是出现但永远不执行。

(i) 在模态 ν 演算中,把不公平的进程表示为断言,并证明任何有限状态进程 p 满足这个断言当且仅当 p 是对应于 a 不公平的。

(ii) 进程 p 称为对应于 a 弱不公平的当且仅当存在一条无限转换链,在该链中 a 无限次地出现但决不执行。试用模态 ν 演算写出断言来表达该性质。 □

14.8 局部模型检查

我们的兴趣在于有限状态进程 p 是否满足递归的模态断言 A,即判定 $p \models A$ 的真假。我们给出将这样的一个满足性断言归约到真假的算法。主要的引理是归约引理,可由 5.5 节的克纳斯特-塔尔斯基定理进行证明。

引理 14.17(归约引理) 设 φ 是幂集 $\mathscr{P}ow(\mathscr{S})$ 上的单调函数. 对 $S \subseteq \mathscr{S}$,有

$$S \subseteq \nu X.\, \varphi(X) \Leftrightarrow S \subseteq \varphi(\nu X.\, (S \cup \varphi(X)))$$

证明

"$\Rightarrow$"假设 $S \subseteq \nu X.\, \varphi(X)$,则

$$S \cup \varphi(\nu X.\, \varphi(X)) = S \cup \nu X.\, \varphi(X) = \nu X.\, \varphi(X)$$

所以,$\nu X.\, \varphi(X)$ 是 $X \mapsto S \cup \varphi(X)$ 的后缀不动点。因为 $\nu X.\, (S \cup \varphi(X))$ 是后缀不动点中最大的不动点,有

$$\nu X.\varphi(X) \subseteq \nu X.(S \cup \varphi(X))$$ 327

由单调性得

$$\nu X.\varphi(X) = \varphi(\nu X.\varphi(X)) \subseteq \varphi(\nu X.(S \cup \varphi(X)))$$

但是,$S \subseteq \nu X.\varphi(X)$,所以 $S \subseteq \varphi(\nu X(S \cup \varphi(X)))$。

"$\Leftarrow$"假设 $S \subseteq \varphi(\nu X.(S \cup \varphi(X))$。因为 $\nu X.(S \cup \varphi(X))$ 是 $X \mapsto S \cup \varphi(X)$ 的不动点,有

$$\nu X.(S \cup \varphi(X)) = S \cup \varphi(\nu X.(S \cup \varphi(X)))$$

所以,由假设

$$\nu X.(S \cup \varphi(X)) = \varphi(\nu X.(S \cup \varphi(X)))$$

即 $\nu X.(S \cup \varphi(X))$ 是不动点,所以是 φ 的后缀不动点。所以,由于 $\nu X.\varphi(X)$ 是最大后缀不动点,得

$$\nu X.(S \cup \varphi(X)) \subseteq \nu X.\varphi(X)$$

显然,$S \subseteq \nu X.(S \cup \varphi(X))$,所以 $S \subseteq \nu X.\varphi(X)$,得证。 □

我们特别关注引理中 S 为单元素集合 $\{p\}$ 的情况。这种情况下引理为

$$p \in \nu X.\varphi(X) \Leftrightarrow p \in \varphi(\nu X.(\{p\} \cup \varphi(X)))$$

这个等价性表示进程 p 满足递归定义的性质当且仅当进程满足这个递归定义性质的某种展开。展开方法比较特殊,因为我们不是用原来的递归定义代入到递归的体中,而是代入更大的含有 p 的递归定义的体。因为 $p \in \varphi(\nu X.(\{p\} \cup \varphi(X)))$ 比 $p \in \nu X.\varphi(X)$ 更容易证明,所以提供了一种判定进程中递归定义断言真假的方法。

通过把断言的语法扩展到包含更一般形式的递归断言,我们允许进程出现于断言中,把进程的有限集合和变量的约束出现结合在一起:

如果 A 是出现变量 X 的断言,$p_1,\cdots,p_n$ 是进程,则 $\nu X\{p_1,\cdots,p_n\}A$ 是一个断言;可以理解为它和 $\nu X.(\{p_1,\cdots,p_n\} \vee A)$ 的性质是一样的。

(因为后一断言可以包含进程集作为常量,所以后一个断言是有意义的。)

我们允许进程集 $\{p_1,\cdots,p_n\}$ 为空;在这种情况中 $\nu X\{\}A$ 等于 $\nu X.A$。事实上,到现在为止我们所写的 $\nu X.A$ 都可理解为 $\nu X\{\}A$ 的简写。 328

练习 14.18 试证明如果 $p \in \{p_1,\cdots,p_n\}$,则 $(p \models \nu X\{p_1,\cdots,p_n\}A) = \mathbf{true}$。 □

借助于这些附加的断言,我们可以提出一个判定 $p \models A$ 是 **true** 还是 **false** 的算法。假设有 n 个真值上的布尔操作。记真值上的否定操作为 $\neg_T$,因此 $\neg_T(\text{true}) = \text{false}$ 和 $\neg_T(\text{false}) = \text{true}$。记 T 上的二元的合取操作为 $\wedge_T$,于是,如果 t_0 和 t_1 都为真,则 $t_0 \wedge_T t_1$ 为真;否则,$t_0 \wedge_T t_1$ 为假。记二元析取操作为 $\vee_T$,于是,如果 t_0 或 t_1 为真,则 $t_0 \vee_T t_1$ 为真;否则为假。更一般地,对 n 个 $t_1,\cdots,t_n$ 的析取,记作

$$t_1 \vee_T t_2 \vee_T \cdots \vee_T t_n$$

如果其中至少一个 $t_i(1 \leq i \leq n)$ 为真,则 $t_1 \vee_T t_2 \vee_T \cdots \vee_T t_n$ 为真;否则为假。我们把空析取理解为假。

由归约引理,我们得到下面的等式:

$$
\begin{aligned}
(p \models S) &= \textbf{true}, \text{如果 } p \in S \\
(p \models S) &= \textbf{false}, \text{如果 } p \notin S \\
(p \models T) &= \textbf{true} \\
(p \models F) &= \textbf{false} \\
(p \models \neg B) &= \neg_T (p \models B) \\
(p \models A_0 \wedge A_1) &= (p \models A_0) \wedge_T (p \models A_1) \\
(p \models A_0 \vee A_1) &= (p \models A_0) \vee_T (p \models A_1) \\
(p \models \langle a \rangle B) &= (q_1 \models B) \vee_T \cdots \vee_T (q_n \models B) \\
\text{其中} \{q_1, \cdots, q_n\} &= \{q \mid p \xrightarrow{a} q\} \\
(p \models \langle . \rangle B) &= (q_1 \models B) \vee_T \cdots \vee_T (q_n \models B) \\
\text{其中} \{q_1, \cdots, q_n\} &= \{q \mid \exists a. p \xrightarrow{a} q\} \\
(p \models \nu X\{\vec{r}\} B) &= \textbf{true}, \text{如果 } p \in \{\vec{r}\} \\
(p \models \nu X\{\vec{r}\} B) &= (p \models B[\nu X\{p, \vec{r}\} B / X]), \text{如果 } p \notin \{\vec{r}\}
\end{aligned}
$$

(在如果 p 没有推导的情况下,由它的推导加标的析取式为 **false**。)

除了最后两个等式,其他等式都是显然的。最后的等式是归约引理的特殊情况,而倒数第二个等式可以通过回忆"带标记的最大不动点的含义而得到(上面的练习要求证明这个等式)"。

329 归约等式建议归约规则,在这些规则中用相应的右边替换左边,尽管没有确保这种归约方式一定会终止。更精确地说,规约规则对从基本的满足性表达式通过布尔操作 $\wedge$、$\vee$、$\neg$ 构造的布尔表达式进行操作,对进程项 p 以及断言 A,表达式的语法形如 $p \vdash A$。布尔表达式形如

$$b ::= p \vdash A \mid \textbf{true} \mid \textbf{false} \mid b_0 \wedge b_1 \mid b_0 \vee b_1 \mid \neg b$$

语法 $p \vdash A$ 和真值 $p \models A$ 不同。

为了精确地归约,随着归约的进行,我们需要规定如何计算出现于满足性表达式之间的布尔操作。为了不局限于一种特殊的方法,使得能覆盖这种布尔表达式的不同求值方法,我们仅仅规定规则有以下性质:

否定:对任何真值 t,有

$$(b \rightarrow^* t \Leftrightarrow \neg b \rightarrow^* \neg_T t)$$

合取:如果 $b_0 \rightarrow^* t_0, b_1 \rightarrow^* t_1, t_0, t_1 \in T$,则对任何真值 t,有

$$(b_0 \wedge b_1) \rightarrow^* t \Leftrightarrow (t_0 \wedge_T t_1) = t$$

析取:如果 $b_0 \rightarrow^* t_0, b_1 \rightarrow^* t_1, t_0, t_1 \in T$,则对任何真值 t, 有

$$(b_0 \vee b_1) \rightarrow^* t \Leftrightarrow (t_0 \vee_T t_1) = t$$

更一般地,如果 $b_1,\cdots,b_n$ 有一个归约到 **true**,则析取 $b_1 \vee b_2 \vee \cdots \vee b_n$ 为 **true**,如果所有的 $b_1,\cdots,b_n$ 都归约到 **false**,那么该析取为 **false**。正如前面提到的,我们把空析取理解为假。

当然,不论求值是从左到右的、从右到左的或并行的,任何布尔表达式的求值规则都有上述的性质。有了布尔表达式的求值方法,下面可以用归约规则的形式介绍算法的核心:

$$(p \vdash S) \rightarrow \textbf{ture}, \text{如果 } p \in S$$
$$(p \vdash S) \rightarrow \textbf{false}, \text{如果 } p \notin S$$
$$(p \vdash T) \rightarrow \textbf{true}$$
$$(p \vdash F) \rightarrow \textbf{false}$$

330

$$(p \vdash \neg B) \rightarrow \neg(p \vdash B)$$
$$(p \vdash A_0 \wedge A_1) \rightarrow (p \vdash A_0) \wedge (p \vdash A_1)$$
$$(p \vdash A_0 \vee A_1) \rightarrow (p \vdash A_0) \vee (p \vdash A_1)$$
$$(p \vdash \langle a \rangle B) \rightarrow (q_1 \vdash B) \vee \cdots \vee (q_n \vdash B)$$

其中 $\{q_1,\cdots,q_n\} = \{q \mid p \xrightarrow{a} q\}$

$$(p \vdash \langle . \rangle B) \rightarrow (q_1 \vdash B) \vee \cdots \vee (q_n \vdash B)$$

其中 $\{q_1,\cdots,q_n\} = \{q \mid \exists a.\, p \xrightarrow{a} q\}$

$$(p \vdash \nu X\{\vec{r}\}B) \rightarrow \textbf{true}, \text{如果 } p \in \{\vec{r}\}$$
$$(p \vdash \nu X\{\vec{r}\}B) \rightarrow (p \vdash B[\nu X\{p,\vec{r}\}B/X]), \text{如果 } p \notin \{\vec{r}\}$$

(同样,在 p 没有推导的情况下,由它的推导加标的析取式为 **false**。)

这个思想是要找出左边满足断言的真值可以归约为要找出右边表达式的真值。除了最后的规则外,其他的从左到右进行处理会取得一些进展是显然的,对这些规则要么右边是真值,要么右边是比左边严格更小的断言。另一方面,最后的规则看上去使得归约不终止。事实上,我们可以证明它终止于正确的答案。粗略地讲,原因是我们是用有限状态的进程检查该断言的满足性,这意味着我们不能永远扩展那些标记递归的集合。

下面的定理表示在布尔表达式的求值假设下,归约规则是可靠的和完备的。(注意,这个定理蕴涵了归约是终止的。)

定理 14.19 设 $p \in \mathscr{P}$ 是有限状态进程,A 是封闭的断言。对任何真值 $t \in T$,有

$$(p \vdash A) \rightarrow^* t \text{ 当且仅当 } (p \models A) = t$$

证明 假设 p 是有限状态进程。如果对所有递归断言 $\nu X\{r_1,\cdots,r_k\}$,有 $r_1,\cdots,r_k \in \mathscr{P}_p$,即所有断言中的进程由从 p 的转换可达,则称断言是 p 断言。施良基归纳于 p 断言进行证明,关系为

$$A' < A \text{ 当且仅当 } A' \text{ 是 } A \text{ 的真子断言}$$
$$\text{或 } A, A' \text{ 形为}$$
$$A \equiv \nu X\{\vec{r}\}B \text{ 且 } A' \equiv \nu X\{p,\vec{r}\}B, \text{其中 } p \notin \{\vec{r}\}$$

331

因为 $\mathscr{P}_p$ 是有限集合，所以关系 $<$ 是良基的。

我们的兴趣是证明对所有封闭的 p 断言，性质

$$Q(A) \Leftrightarrow_{def} \forall q \in \mathscr{P}_p \forall t \in T. [(q \vdash A) \rightarrow^* t \Leftrightarrow (q \models A) = t]$$

成立。然而，证明中需要我们把性质 Q 扩展到具有自由变量集 $FV(A)$ 的 p 断言 A，我们用下面的方式进行证明：

对 p 断言 A，定义

$$Q^+(A) \Leftrightarrow_{def} \forall \theta, \text{从 FV}(A) \text{ 到封闭 } p \text{ 断言的一个代入},$$
$$[(\forall X \in FV(A). Q(\theta(X))) \Rightarrow Q(A[\theta])]$$

注意当 A 是封闭的时，$Q^+(A)$ 和 $Q(A)$ 逻辑上等价。这里，θ 表示代入 $B_1/X_1, \cdots, B_k/X_k$ 的简写，即诸如 $\theta(X_j)$ 这样的表达式和断言 B_j 对应。

我们施良基归纳于 $<$ 去证明对所有 p 断言 A，$Q^+(A)$ 成立。为此，设 A 是 p 断言，使得对所有 p 断言 $A' < A$，有 $Q^+(A')$。下面我们需要证明 $Q^+(A)$。所以，设 θ 是满足 $\forall X \in FV(A). Q(\theta(X))$ 的 $FV(A)$ 到封闭 p 断言的代入。对所有 A 的可能形式，我们要证明 $Q(A[\theta])$。我们选择几种情况进行证明：

$A \equiv A_0 \wedge A_1$：在这种情况下，$A[\theta] \equiv A_0[\theta] \wedge A_1[\theta]$。设 $q \in \mathscr{P}_p$，令 $(q \models A_0[\theta]) = t_0$ 和 $(q \models A_1[\theta]) = t_1$。因为 $A_0 < A$ 且 $A_1 < A$，所以，我们有 $Q^+(A_0)$ 和 $Q^+(A_1)$ 成立。因而，$Q(A_0[\theta])$ 和 $Q(A_1[\theta])$ 成立。因此，有 $(q \vdash A_0[\theta]) \rightarrow^* t_0$ 和 $(q \vdash A_1[\theta]) \rightarrow^* t_1$。现在，对 $t \in T$，

$$\begin{aligned}(q \vdash A_0[\theta] \wedge A_1[\theta]) \rightarrow^* t &\Leftrightarrow ((q \vdash A_0[\theta]) \wedge (q \vdash A_1[\theta])) \rightarrow^* t \\ &\Leftrightarrow t_0 \wedge_T t_1 = t \quad (\text{由合取计算的性质}) \\ &\Leftrightarrow (q \models A_0[\theta]) \wedge_T (q \models A_1[\theta]) = t \\ &\Leftrightarrow (q \models A_0[\theta] \wedge A_1[\theta]) = t\end{aligned}$$

所以，在这种情况下 $Q(A[\theta])$ 成立。

$A \equiv X$：在这种情况下，当 X 是变量时，根据对 Q 的假设，显然有 $Q(A[\theta])$ 成立。

$A \equiv \nu X\{\vec{r}\}B$：这时，$A[\theta] \equiv \nu X\{\vec{r}\}(B[\theta])$——因为 X 不是 A 的自由变量，所以 θ 对 X 无定义。设 $q \in \mathscr{P}_p$，或者 $q \in \{\vec{r}\}$ 或者 $q \notin \{\vec{r}\}$。如果 $q \in \{\vec{r}\}$，则对任一 $t \in T$，有

$$(q \vdash \nu X\{\vec{r}\}(B[\theta])) \rightarrow^* t \Leftrightarrow t = \mathbf{true}$$

并且 $(q \models \nu X\{\vec{r}\}(B[\theta])) = \mathbf{true}$。因此，当 $q \in \{\vec{r}\}$ 时，$Q(A[\theta])$ 成立。而如果 $q \notin \{\vec{r}\}$，$\nu X\{q, \vec{r}\}B < A$，所以有 $Q(\nu X\{q, \vec{r}\}(B[\theta]))$。定义一个从 $Y \in FV(B)$ 到封闭的 p 断言的代入 θ'，取

332

$$\theta'(Y) = \begin{cases} \theta(Y), & Y \not\equiv X \\ \nu X\{q, \vec{r}\}(B[\theta]), & Y \equiv X \end{cases}$$

显然，对所有的 $Y \in FV(B)$，有 $Q(\theta'(Y))$。因为 $B < A$，所以有 $Q^+(B)$。因此，有 $Q(B[\theta'])$。但是，$B[\theta'] \equiv (B[\theta])[\nu X\{q, \vec{r}\}(B[\theta])/X]$。所以由规约规则，

$$\begin{aligned}(q \vdash \nu X\{\vec{r}\}(B[\theta])) \rightarrow^* t &\Leftrightarrow (q \vdash (B[\theta])[\nu X\{q, \vec{r}\}(B[\theta])/X] \rightarrow^* t \\ &\Leftrightarrow (q \vdash B[\theta']) \rightarrow^* t\end{aligned}$$

$$\Leftrightarrow (q \models B[\theta']) = t \quad (\text{因为 } Q(B[\theta']))$$
$$\Leftrightarrow (q \models (B[\theta])[\nu X\{q,\vec{r}\}(B[\theta])/X]) = t$$
$$\Leftrightarrow (q \models \nu X\{\vec{r}\}(B[\theta])) = t \quad (\text{由归约引理})$$

所以,不论 $q \in \{\vec{r}\}$ 还是 $q \notin \{\vec{r}\}$,$Q(A[\theta])$都成立。

A 的其他可能形式的证明情况作为练习。用良基归纳法我们得到:对所有 p 断言 A,有 $Q^+(A)$。特别地,对所有封闭的断言 A,$Q(A)$成立,定理证毕。 □

例 考察 CCS 中由下式给出的两个元素的转换系统

$$P \stackrel{def}{=} a.Q$$
$$Q \stackrel{def}{=} a.P$$

它由两个转换 $P \xrightarrow{a} Q$ 和 $Q \xrightarrow{a} P$ 组成。下面证明:重写算法建立了明显为真的事实,即 P 能执行任意多次的 a 动作,形式地,$p \models \nu X.\langle a\rangle X$。读者回想一下,$\nu X.\langle a\rangle X$ 代表 $\nu X\{\}\langle a\rangle X$,由模型检查算法的归约,我们得到

$$\begin{aligned}
P \vdash \nu X\{\}\langle a\rangle X &\to P \vdash \langle a\rangle X[\nu X\{P\}\langle a\rangle X/X] \\
&\quad (\text{即 } P \vdash \langle a\rangle \nu X\{P\}\langle a\rangle X) \\
&\to Q \vdash \nu X\{P\}\langle a\rangle X \\
&\to Q \vdash \langle a\rangle X[\nu X\{Q,P\}\langle a\rangle X/X] \\
&\quad (\text{即 } Q \vdash \langle a\rangle \nu X\{Q,P\}\langle a\rangle X) \\
&\to P \vdash \nu X\{Q,P\}\langle a\rangle X \\
&\to \mathbf{true}
\end{aligned}$$

□ 333

所以,如果断言语言的常量断言限制到判定性质,则归约规则给出了判定进程是否满足断言的方法。我们集中考虑局部模型检查的正确性而不是算法的效率。因为算法没有充分挖掘共享数据的潜在性质,所以在最坏的情况下,算法是低效率的。下一节包含了更仔细、更有效的算法的参考文献。

练习 14.20 (i) 对由下式定义的 CCS 进程 P:

$$P \stackrel{def}{=} a.P$$

试证明上述算法把 $p \vdash \nu X.\langle a\rangle T \wedge [a]X$ 归约为 **true**。

(ii) 对下列 CCS 定义

$$P \stackrel{def}{=} a.Q$$
$$Q \stackrel{def}{=} a.P + a.\mathbf{nil}$$

试证明 $P \vdash \mu X.[a]F \vee \langle a\rangle X$ 归约为 **true**。 □

练习 14.21 试用 SML 或 Prolog 设计一个方法,从操作语义中抽取出有限状态进程的转换系统表。试编程实现模型检查算法。并用它研究下面的简单协议。 □

练习 14.22 在 CCS 中简单通信协议(摘自[72])描述为:

$$\begin{aligned}
\text{Sender} &= a.\ \text{Sender}' \\
\text{Sender}' &= \bar{b}.\ (\text{d. Sender} + c.\ \text{Sender}') \\
\text{Medium} &= b.\ (\bar{c}.\ \text{Medium} + \bar{e}.\ \text{Medium}) \\
\text{Receiver} &= e.f.\ \bar{d}.\ \text{Receiver} \\
\text{Protocol} &= (\text{Sender} \parallel \text{Medium} \parallel \text{Receiver}) \backslash \{b,c,d,e\}
\end{aligned}$$

试用练习 14.21 中给出的工具(或者 Concurrency Workbench 或 TAV 系统)证明:

334 进程 Protocol 不满足 $Inv([a](ev\langle f\rangle T))$。

Protocol 满足 $Inv([f](ev\langle a\rangle T))$。

(提示:$Inv(A) \equiv \nu X.\ (A \wedge [.]X)$,$ev(A) \equiv \mu X.\ (A \vee (\langle . \rangle T \wedge [.]X))$,其中总是满足 A 的那些进程满足 $Inv(A)$,而最终满足 A 的那些进程满足 $ev(A)$。) □

练习 14.23(互模拟测试) 强互模拟可以用最大不动点表达(见练习 14.9)。试测试两个有限状态进程间的互模拟可以和局部模型检查一样自动进行。如何用 SML 或 Prolog 程序来实现?(尽管该方法和 Concurrency Workbench 不同,但和 TAV 系统接近。) □

14.9 进一步阅读资料

如何对并行系统建模和论证的数学理论还没有解决。本章所作的简要介绍显然是不完整的。

我们讨论过取自[36]的迪杰斯特拉的卫式命令语言,霍尔把它扩展为通信顺序进程(CSP)[49],米尔纳研究了通信系统演算(CCS)。米尔纳关于 CCS 的书[63]很适合作为本科生的教材。米尔纳的手册[64]简要地讨论了该书的理论内容。霍尔的书[50]集中讨论了另一种等价(“失败”等价),代表了另一种有影响的研究方向。亨纳斯的书[48]介绍了这方面的数学描述。程序设计语言 Occam[70]是基于 CCS 和 CSP 的。Kozen 在[55]中讲述了模态 ν 演算。到 1992 年为止,这包含了逻辑公理完备性方面最好的结果——完全逻辑的公理完备性还在研究!该逻辑更传统地称为(模态)μ 演算。因为我们研究最大不动点而不是最小不动点,所以在名字上有所不同。

最好的 CCS 方面的经典著作是由 Edinburgh-Sussex Concurrency Workbench[30]和 Aalborg
335 TAV 系统这样的工具实现的[46][⊖]。Walker 的论文[100]很好地将 Concurrency Workbench 用

⊖ Concurrency Workbench 可以从爱丁堡大学或北卡罗莱纳州立大学获得:
George Cleland, LFCS, Dept. of Computer Science, University of Edinburgh, The King's Buildings, Edinburgh EH9 3JZ, Scotland。
电子邮件:lfcs@ ed. ac. uk,
匿名 FTP:ftp. dcs. ed. ac. uk(IP 地址:129.215.160.150)。
Rance Cleaveland, Dept. of Computer Science, N. C. State University, Raleigh, NC27695-8206, USA。
电子邮件:rance@ adm. csc. ncsu. edu,
匿名 FTP:science. csc. ncsu. edu(IP 地址:152.1.61.34)。
TAV 系统可以从 Kim G. Larsen 或 Arne Skou 处获得:
Institute for Electronic Systems, Dept. of Mathematics and Computer Science, Aalborg University Centre, Fredrik Bajersvej 7, 9200 Aalborgϕ, Denmark。
电子邮件:kgl@ iesd. auc. dk。

于研究并行算法很好的描述。[52]中把 Concurrency Workbench 扩展,使其能处理 Occam 语言的优先级;论文[18]给出了推广的互模拟等式证明系统。[93,25]中可以找到 Concurrency Workbench 的理论基础源于[57](本章的模型检查部分参考[106])。最近几年模型检查成为研究的热门。在本书写作时(1992),Edinburgh-Sussex Concurrency Workbench 对公式和转换系统的大小成指数增长(即使只有一个不动点操作)。这里的算法有同样的缺陷。它们不能尽可能多地重用计算中获得的信息。按照"选择深度"(断言的最小不动点和最大不动点关联度的度量值) TAV 系统与公式和转换系统的大小是多项式的关系。到现在为止,大多数的局部模型检查的算法是选择深度 2(见文献[6,7])。还有其他许多模型检查的方法([37])是基于逻辑的(见[24])。

除了本章外,本书介绍了程序设计语言的操作语义和指称语义。我们没有给出进程语言的指称语义,因为它涉及域论中的"幂域"理论,而这不是本书讨论范围。幂域是类似于幂集的完全偏序,这些幂集能使指称表示可能输出的集合。Plotkin 在[79]中提出了这些概念并介绍了它们的用法(论文[92]和[102]也有更浅显的介绍)。

最近 Apt 和 Olderog 的书[9]把霍尔逻辑扩展成并行程序。时态逻辑被强烈提倡为论证并行进程的方法(见[60,56])。

在效果上,我们这里的并行性表示形式处理了复合性的问题,将其看作是复合成为的原子动作的不确定交叉的描述。还有其他模型,如 Petri 网和事件结构,它们将并行性显式地表示为动作之间的一种独立性,于是把并行进程和纯不确定性进程区别开来。[85]中介绍了 Petri 网。最近用结构化操作语义表达了 Petri 网的数学框架。[107]对不同的并行进程模型都给出了简要的论述。 336

附录 A　不完备性和不可判定性

本附录对可计算性理论作简要的介绍[⊖]，包括可计算（部分递归）函数、递归可枚举和可判定集合等基本概念。可以证明，“停机问题”是不可判定的，并由此可知 **Assn** 的有效断言不是递归可枚举的。特别是它充实了 7.3 节中的哥德尔不完备性定理的证明。通过对“通用 **IMP** 程序”的讨论还可以给出另一种证明。最后，在更为严格地检查后得出，有关 **Assn** 断言成真（成假）的问题不是递归可枚举的。

A.1　可计算性

一个 **IMP** 命令 c 可以和 $\mathbf{N}$ 上的一个部分函数相关联。我们假定，将所有的存储单元排列为 $X_1, X_2, X_3, \cdots$。设在状态 σ_0 下，所有的存储单元都被赋值为 0。对于 $n \in \mathbf{N}$，定义

$$\{c\}(n) = \begin{cases} \sigma(X_1) & \sigma = \mathscr{C}[c]\sigma_0[n/X_1] \\ \text{未定义} & \mathscr{C}[c]\sigma_0[n/X_1] \text{ 是未定义的} \end{cases}$$

任一 $\mathbf{N} \rightharpoonup \mathbf{N}$ 的部分函数，若对某一命令 c，能使 $n \mapsto \{c\}(n)$，其中，$n \in \mathbf{N}$，则称该函数是 **IMP** 可计算的。这样的函数也被称为是“部分递归的”，如果是全函数，则称为“递归的”。更一般地，我们可以将一个命令与一个 k 元部分函数相关联，从而定义从 $\mathbf{N}^k$ 到 $\mathbf{N}$ 的 **IMP** 可计算函数。对 $n_1, \cdots, n_k \in \mathbf{N}$，定义

$$\{c\}(n_1, \cdots, n_k) = \begin{cases} \sigma(X_1) & \sigma = \mathscr{C}[c]\sigma_0[n_1/X_1, \cdots, n_k/X_k] \\ \text{未定义} & \mathscr{C}[c]\sigma_0[n_1/X_1, \cdots, n_k/X_k] \text{ 是未定义的} \end{cases}$$

为了证明 **IMP** 可计算函数的复合也是 **IMP** 可计算函数，我们先介绍一个概念——整洁命令，即该命令执行终止时将除 X_1 以外的所有存储单元设置为 0。

定义　称 **IMP** 命令 c 是整洁的当且仅当对所有的状态 σ 和数 n，有

$$\mathscr{C}[c]\sigma_0[n/X_1] = \sigma \Rightarrow \sigma[0/X_1] = \sigma_0$$

练习 A.1　试证明：如果 f 是 **IMP** 可计算的，则对所有的 m, n，存在一个整洁 **IMP** 命令使得 $f(n) = m$ 当且仅当 $\{c\}(n) = m$。 □

现在，容易证明下面的命题是成立的。

命题 A.2　设 c_0 和 c_1 是命令，且 c_0 是整洁的。对任意 $n, m \in \mathbf{N}$，有

⊖ 本附录是以 Albert Meyer 在 MIT 的本科课程讲义为基础的，讲义是对 1～7 章内容的补充，非常感谢 Albert 允许我无偿地使用他的讲义。

$$\{c_1\}(\{c_0\}(n)) = m \text{ 当且仅当 } \{c_0;c_1\}(n) = m$$

记号 对部分函数 f 和变量 n，我们用记号 $f(n)\downarrow$ 表示 $\exists m.f(n)=m$，即结果有定义；同时用记号 $f(n)\not\downarrow$ 表示结果未定义。

注意，记号 $\{c\}(n)\downarrow$ 与命令 c 从开始状态 $\sigma_0[n/X_1]$ 执行后终止是一致的。**N** 的子集 M 是 **IMP** 可检查的，当且仅当有一个 **IMP** 命令 c，使得

$$n \in M \text{ 当且仅当 } \{c\}(n)\downarrow$$

也就是说，给定一个输入 n（其中 n 在存储单元 X_1 中，其他存储单元初始化为零）后，命令 c "检查" n 是否在 M 中，如果检查过程成功则停止；如果 n 不在 M 中，则命令将继续检查下去（如此将永远不会成功）。可检查集合通常也称为"递归可枚举"集。

与之关系密切的另一个概念是 **IMP** 可判定集。一个子集 $M\subseteq\mathbf{N}$ 是 **IMP** 可判定的，当且仅当有一个 **IMP** 命令 c 使得

$$n \in M \text{ 蕴涵 } \{c\}(n) = 1$$

且

$$n \notin M \text{ 蕴涵 } \{c\}(n) = 0$$

也就是说，给定一个输入 n，命令 c 检查是否 $n\in M$。若是，则将 1 放入 X_1 作为输出返回，否则返回输出 0。命令对所有的输入都会以这样一个输出终止。可判定集有时也称作"递归"集。

如果 c 是 M 的"判定者"，则

$$c;\ \textbf{if } X_1 = 1 \textbf{ then skip else } \text{Diverge}$$

是 M 的"检查者"，其中，Diverge = **while true do skip**。因此，我们有下面的引理。

引理 A.3 如果 M 是可判定的，则 M 是可检查的。

练习 A.4 试证明：如果 M 是可判定的，则补集 $\overline{M}=\mathbf{N}\backslash M$ 也是可判定的。 □

练习 A.5 试证明：如果 M 是可检查的，则存在 M 的检查者 c，使得对所有的 $n\in\mathbf{N}$，$\{c\}(n)\downarrow$ 蕴涵 $\mathscr{E}[c]\sigma_0[n/X_1]=\sigma_0$。换言之，$c$ 只有在它"清理了所有的存储单元"后才停机（参考练习 A.1）。 □

338

反之，如果 c_1 是 M 的检查者，c_2 是 $\overline{M}$ 的检查者，则可以通过使 c_1 和 c_2 "分时"或"榫接"来构造命令 c，从而得到 M 的判定者。

下面，更详细地描述 c 的构造过程。设 T,F,S 是不属于 $\mathbf{Loc}(c_1)\cup\mathbf{Loc}(c_2)$ 的"新"存储单元，将 $\mathbf{Loc}(c_i)\backslash\{X_1\}$ 清 0 的赋值命令序列简记为"Clear_i"，则 c 可以写成：

```
T := X1;                    % 将 X1 保存到 T 中
F := 0;                     % F 为标志
S := 1;                     % 执行的步数
[while F=0 do
```

```
Clear1;X1 := T;
"执行 c1 S 次,或直到 c1 停机";
if "c1的执行次数 ≤ S 且停机" then
                    F := 1;              % 执行结束
                    X1 := 1;             % T 属于 M
                else S := S + 1;         % 步计数器加 1
if F = 1 then skip else
  Clear2;X1 := T;
  "执行 c2 S 次,或直到 c2 停机";
  if"c2 的执行次数 ≤ S 且停机" then
                    F := 1;              % 执行结束
                    X1 := 0;             % T 不属于 M
                else S := S + 1];        % 步计数器加 1
Clear1;Clear2;T := 0;F := 0;S := 0       % 清理 X1 以外的存储单元
```

练习 A.6 试描述如何将一条命令 c_1 转换为符合描述"执行 c_1 S 次,或直到 c_1 停机(无论哪一种情况先发生)。"的命令。 □

于是,我们有下面的定理。

定理 A.7 集合 M 是可判定的当且仅当 M 和 $\overline{M}$ 都是可检查的。

A.2 不可判定性

通过对命令进行编码,将它们转换为一些数,我们可以将它们作为其他命令的输入。我们用下面的方法将命令编码为一个数#c。设 mkpair 是整数对的配对函数,例如,

$$\text{mkpair}(n,m) = 2^{\text{sg}(n)} \cdot 3^{|n|} \cdot 5^{\text{sg}(m)} \cdot 7^{|m|}$$ 339

其中

$$\text{sg}(n) = \begin{cases} 1 & n \geqslant 0 \\ 0 & n < 0 \end{cases}$$

配对函数的细节无关紧要,重要的是有两个函数 left 和 right 使得

$$\text{left}(\text{mkpair}(n,m)) = n$$
$$\text{right}(\text{mkpair}(n,m)) = m$$

此外,还有一些类似于赋值语句的 **IMP** 命令,它们的形式是

$$X := \text{mkpair}(Y,Z)$$
$$X := \text{left}(Y)$$
$$X := \text{right}(Y)$$

练习 A.8

(i) 试给出 **IMP** 命令 Mkpair、Left、Right,实现上面的函数,即,对所有的 $n,m \in \mathbf{N}$,

$$\{\text{Mkpair}\}(n,m) = \text{mkpair}(n,m)$$
$$\{\text{Left}\}(n) = \text{left}(n)$$
$$\{\text{Right}\}(n) = \text{right}(n)$$

(ii) 设 c 是一条 **IMP** 命令形式的文本,但 c 包含"$X := \text{left}(Y)$"形式的赋值语句。试说明怎样构造一个真实的 **IMP** 命令 $\hat{c}$,使得 $\hat{c}$ 可以使用临时存储单元模拟 c。

(iii) 假设将 **Aexp** 的定义,当然也对 **IMP** 的定义,进行修改,使得 **Aexp** 可以是"mkpair(a_1,a_2)","left(a)"和"right(a)"的形式,其中 a,a_1,a_2 本身也是修改后的 **Aexp** 中的表达式。称修改后的语言为 **IMP′**。试说明,如何将每一个 $c' \in \mathbf{Com}'$ 转换成 $c \in \mathbf{Com}$ 使得 c 模拟 c'。 □

为了将命令编码为数,我们要利用存储单元集合 **Loc** 上的对 $X_1,X_2,X_3,\cdots$ 的编号。我们用0 作为"存储单元标志",并定义

$$\#(X_i) = \text{mkloc}(i) = \text{mkpair}(0,i)$$

同样,也对数编码,用1 作为"数的标志":

340

$$\#(n) = \text{mknum}(n) = \text{mkpair}(1,n)$$

用2、3、4 分别作为和、差、积的标志对 **Aexp** 中的算术表达式进行编码,例如:

$$\#(a_1 + a_2) = \text{mksum}(\#a_1,\#a_2) = \text{mkpair}(2,\text{mkpair}(\#a_1,\#a_2))$$

用5、6、7、8、9 分别作为≤, =, ∧, ∨, ¬ 的标志对 **Bexp** 中的布尔表达式进行编码,例如:

$$\#(a_1 \leq a_2) = \text{mkleq}(\#a_1,\#a_2) = \text{mkpair}(5,\text{mkpair}(\#a_1,\#a_2))$$
$$\#(b_1 \vee b_2) = \text{mkor}(\#b_1,\#b_2) = \text{mkpair}(8,\text{mkpair}(\#b_1,\#b_2))$$

最后,用10 ~ 14 分别作为:=、**skip**、**if**、顺序、**while** 等命令的标志对 **Com** 进行编码,例如:

$$\begin{aligned}\#(\textbf{if } b \textbf{ then } c_0 \textbf{ else } c_1) &= \text{mkif}(\#b,\#c_0,\#c_1)\\ &= \text{mkpair}(12,\text{mkpair}(\#b,\text{mkpair}(\#c_0,\#c_1)))\end{aligned}$$

这种对语法或有限结构化对象的编码方法由哥德尔于20 世纪30 年代最先使用,因此$\#(c)$也称为 c 的*哥德尔数*。

当命令被编码后,我们就可以讨论将一个命令(即它的编码)作为另一个命令的输入了。我们称命令的子集 S 是可检查的(或可判定的),如果它们的编码集合

$$\{\#c \mid c \in S\}$$

是可检查的(或可判定的)。

练习 A.9 试描述如何写出一个 **IMP** 命令,它能判定一个数是否为某个 **IMP** 中合式命令的编码。并证明集合$\{c \mid c \in \mathbf{Com}\}$是可判定的。 □

设 H 是“自停机”的命令集：

$$H = \{c \mid \{c\}(\#c)\downarrow\}$$

记

$$\overline{H} =_{def} \{c \in \mathbf{Com} \mid c \notin H\}$$

定理 A.10 $\overline{H}$ 不是 **IMP** 可检查的。

证明 假设 C 是一个检查 $\overline{H}$ 的 **IMP** 命令，即，对所有的命令 c，

$$c \in \overline{H} \text{ 当且仅当 } \{C\}(\#c) \text{ 有定义}$$

341

特别地，由于 C 自身也是一个命令，再由 $\overline{H}$ 的定义，有

$$\{C\}(\#C)\not\downarrow \text{ 当且仅当 } \{C\}(\#C)\downarrow$$

矛盾。因此，$\overline{H}$ 不能是可检查的。 □

推论 A.11（停机问题） 集合 H 是不可判定的。

其他性质的不可判定性都可以通过停机问题的不可判定性得到。定义

$$H_0 = \{c \in \mathbf{Com} \mid \mathscr{E}[c]\sigma_0 \neq \perp\}$$

注意

$$H_0 = \{c \in \mathbf{Com} \mid \{c\}(0)\downarrow\}$$

由于 $\overline{H}$ 不是可检查的，所以 $\overline{H}_0 = \{c \in \mathbf{Com} \mid c \notin H_0\}$ 也不是可检查的。

定理 A.12（零状态停机问题） 集合 $\overline{H}_0$ 不是可检查的。

证明 证明利用了一个实现函数 g 的命令，使得对任一命令 c，

$$g(\#c) = \#(X_1 := \#c; c)$$

这样的函数可以由定义

$$g(n) = \mathrm{mkseq}(\mathrm{mkassign}(\mathrm{mkloc}(1), \mathrm{mknum}(n)), n)$$

得到，其中 $n \in \mathbf{N}$。然而，由练习 A.8 可知，存在着一个命令 G，使得

$$\{G\}(n) = g(n)$$

又由练习 A.1，我们设命令 G 是整洁的。

为了推出矛盾，我们假设 $\overline{H}_0$ 是可检查的，即，存在一个命令 C，对任一命令 c，都有

$$c \in \overline{H}_0 \text{ 当且仅当 } \{C\}(\#c)\downarrow$$

则

$$\begin{aligned} c \in \overline{H} &\text{ 当且仅当 } \{c\}(\#c)\not\downarrow \\ &\text{ 当且仅当 } \{X_1 := \#c; c\}(0)\not\downarrow \\ &\text{ 当且仅当 } (X_1 := \#c; c) \in \overline{H}_0 \end{aligned}$$

$$\text{当且仅当 } \{C\}(\#(X_1 := \#c;c)) \downarrow$$
$$\text{当且仅当 } \{C\}(g(\#c)) \downarrow$$
$$\text{当且仅当 } \{C\}(\{G\}(\#c)) \downarrow$$
342
$$\text{当且仅当 } \{G;C\}(\#c) \downarrow$$

其中,最后一个步骤可由命题 A.2 得到。但这就使得命令 $G;C$ 成为了 $\bar{H}$ 的一个检查者,矛盾。因此,$\bar{H}_0$ 不是可检查的。 □

练习 A.13

(i) 试描述一个 **IMP** 命令 C,对于任一命令 c 的哥德尔数$\#c$,由命令 C 输出 c 的存储单元 X_k 的最大值 k。由此可证明:存在一个 **IMP** 可计算函数,对输入$\#c$,可得到清除所有出现在 c 中存储单元的命令

$$X_1 := 0;X_2 := 0;\cdots;X_k := 0$$

的哥德尔数。

(ii) 设

$$D = \{c \in \mathbf{Com} \mid \forall \sigma.\ \mathscr{E}[\![c]\!]\sigma = \perp\}$$

利用(i),由 $\bar{H}_0$ 不是可检查的这一事实出发,证明 D 也不是可检查的。 □

A.3 哥德尔不完备性定理

如果存在一个能力足够强的定理证明系统,可以证明所有的(当然也可以是仅有的)**Assn** 有效断言,那么,我们就希望写一个这样的程序,当给定一个输入断言 A(的编码)后,该程序就毫无遗漏地去搜索 A 的证明,并且停机当且仅当找到了这样一个证明。这个程序就是一个有效性的检查者。

更详细一些,设想我们用类似于对表达式和命令的方法,也对断言进行编码并得到它的哥德尔数$\#A$。任何理所当然地称为"定理证明器"的系统应能够提供一种方法用来判定一个结构化有限对象(一般情况下是一个 **Assn** 断言的有限序列)是否为给定断言的一个证明。一个有证明能力的检查者会详尽地搜索这些结构化有限对象,从中找到一个证明对象。因此,为了与"定理证明器"的名字相称,我们有理由要求集合

$$\mathbf{Provable} = \{A \in \mathbf{Assn} \mid \vdash A\}$$

是 **IMP** 可检查的。如前所述,对于命令,称一个断言的子集是可检查的当且仅当与之对应的哥德尔数的集合是可检查的。设有效断言构成的集合为

$$\mathbf{Valid} = \{A \in \mathbf{Assn} \mid \models A\}$$

有效性的定理证明器应使该集合是可检查的。但是,我们有下面的定理。

343 **定理 A.14** **Valid** 不是可检查的。

证明 设函数 h 对所有的命令 c 有

$$h(\#c) = \#(w[c,\mathbf{false}][\vec{0}/\mathbf{Loc}(c)])$$

(不言而喻,以上记号的含义是将 c 的每一个存储单元代换为0,从而将所有出现在断言中的存储单元代换为0。)

证明中要利用一个能实现函数 h 的命令 W,命令的存在性可由定理7.5的构造性证明得到;其中给出了对命令 c 和断言 A 构造一个表示最弱前置条件的断言 $w[c,A]$ 的方法,这样,原则上我们同样可以写出用于哥德尔数的 **IMP** 命令。下面的证明依赖于存在这样一个命令 W,使得

$$\{W\}(\#c) = \#(w[c,\mathbf{false}][\vec{0}/\mathbf{Loc}(c)])$$

在此,我们不给出命令 W 的构造细节。可设 W 是一个整洁命令。

假设 **Valid** 是可检查的,即,有一个命令 C,使得对任一断言 A,

$$A \in \mathbf{Valid} \text{ 当且仅当 } \{C\}(\#A)\downarrow$$

令 $A \equiv w[c,\mathbf{false}][\vec{0}/\mathbf{Loc}(c)]$,则

$$\begin{aligned} c \in \bar{H}_0 \text{ 当且仅当 } & A \in \mathbf{Valid} \\ \text{当且仅当 } & \{C\}(\#A)\downarrow \\ \text{当且仅当 } & \{C\}(\{W\}(\#c))\downarrow \\ \text{当且仅当 } & \{W;C\}(\#c)\downarrow \quad (\text{由命题 A.2}) \end{aligned}$$

这样,$\bar{H}_0$ 是可由命令 $W;C$ 检查的,矛盾。因此,**Valid** 不是可检查的。 □

以上证明同样可用于这样一个特殊的有效断言子集:它是封闭的而且无存储单元(即不涉及任何存储单元),这些断言要么为真要么为假,且不依赖于任何状态和解释。我们设

$$\mathbf{Truth} = \{A \in Assn \mid A \text{ 封闭 } \& \text{ 无存储单元 } \& \models A\}$$

注意到断言"$w[c,\mathbf{false}][\vec{0}/\mathbf{Loc}(c)]$"在上面的证明中封闭且无存储单元,因此,用同样的论证可证明 **Truth** 不是 **IMP** 可检查的。这样,对所有的定理证明器均有 **Provable** $\neq$ **Truth**。由于我们想得到一个可靠的证明系统,故我们至多有 **Provable** $\subsetneq$ **Truth**。所以,对任何一个定理证明器,如果它能证明的断言都为真,则必有某些真的断言它不能证明。因而定理证明器不可能完备地证明真的断言。这就是哥德尔(第一)不完备性定理,可用一个抽象的形式简述如下。 344

定理 A.15 **Truth** 不是可检查的。

哥德尔不完备性定理的证明基于表示最弱前置条件的断言的构造。在下一节中将给出另一种证明,它是基于一个"通用程序"的存在性的。

A.4 一个通用程序

今天,为 **IMP** 命令写一个"模拟器"是一个平常的事(但在20世纪30年代却是一个惊人的想像);事实上,可以用 **IMP** 实现这个模拟器。在此,我们需要一个命令 *SIM*,给定一个输入数对$(\#c,n)$,它输出的结果与 c 在输入 n 上运行后的输出结果相同。更准确的说明为:对任一命令 c,以及 $n,m \in \mathbf{N}$,

$$\{SIM\}(\#c,n) = m \text{ 当且仅当 } \{c\}(n) = m$$

(注意,由练习 A.9,我们可以排除那些不是命令编码的数。)

定理 A.16 (通用程序定理) *存在一个 IMP 命令 SIM,满足上面的说明。*

证明 留作练习。这是一个较长的构造 SIM 的编程练习,它的正确性证明会更长、更复杂。 □

推论 A.17 *自停机集合 H 是 IMP 可检查的。*

证明 命令"$X_2 := X_1; SIM$"是一个 H 的 IMP 检查者。 □

集合 $M \subseteq \mathbf{N}$ 是可表达的当且仅当有一个无存储单元的 $A \in \mathbf{Assn}$ 且仅含有一个整数自由变量 i,使得

$$\models A[n/i] \text{ 当且仅当 } n \in M$$

换言之,A 意为"i 在 M 中"。只要用一个具体的数(例如 7)替换 i,则断言 $A[7/i]$ 为真或者为假(依赖于是否 $7 \in M$)不依赖于确定其真值的状态 σ 或解释 I。

345 **定理 A.18** *每一个 IMP 可检查的集合 $M \subseteq \mathbf{N}$ 都是可表达的。*

证明 设 $c \in \mathbf{Com}$ 是 M 的一个检查者,令 $w[c,\mathbf{false}]$ 表示 c 和 **false** 的最弱前置条件,则

$$(\neg\, w[c,\mathbf{false}])[i/X_1][\vec{0}/\mathbf{Loc}(c)]$$

表达了 M。 □

一旦我们用类似于对 **Com** 的方法为 **Assn** 断言指派哥德尔数,我们得到的编号方式就会有一个重要性质:对无存储单元且仅有一个自由整数变量 i 的断言 A,设 $f(n) = \#(A[n/i])$;那么,我们就声称存在一个能实现 f 的 **IMP** 命令 S,即,对任一 $n \in \mathbf{N}$,

$$\{S\}(n) = f(n)$$

要做到这一点,一种方法是假定 A 具有下面的形式

$$\exists j.\, j = i \wedge A'$$

其中 A' 中没有 i 的自由出现。这种假定不失一般性,因为任一 $A \in \mathbf{Assn}$ 都能等价地表示为一个具有以上形式的断言。现在,我们有

$$f(n) = \text{mkexistential}(\#(j), \text{mkand}(\text{mkeq}(\#(j), \text{mknum}(n)), \#(A')))$$

所以 $f(n)$ 可用一个带有"mkpair"运算的 **Aexp** 断言定义,因此,由前面的练习 A.8 可知,对所有的 n,存在一个 **IMP** 命令 S,使得 $\{S\}(n) = f(n)$。再由练习 A.1,不妨设 S 是整洁的。

在下面的不完备性定理的另一种证明中,这个性质是我们惟一用到的有关封闭断言的事实。

不完备性定理的另一种证明:

假设 $C \in \mathbf{Com}$ 是 **Truth** 的一个检查者,由于自停机集合 H 是可检查的,故存在一个断言 B,使得对所有的命令 c,

$$c \in H \text{ 当且仅当} \models B[\#c/i]$$

令 A 为 $\neg B$,我们有

$$\begin{aligned} c \in \bar{H} &\text{ 当且仅当} \models A[\#c/i] \\ &\text{当且仅当} A[\#c/i] \in \mathbf{Truth} \\ &\text{当且仅当} \{C\}(\#(A[\#c/i]))\downarrow \\ &\text{当且仅当} \{C\}(\{S\}(\#c))\downarrow \\ &\text{当且仅当} \{S;C\}(\#c)\downarrow \end{aligned}$$

346

其中 S 是完成对 A 进行代入的整洁命令。

从而“$S;C$”是 $\bar{H}$ 的一个检查者,矛盾。 □

练习 A.19 试证明 **Trūth** 也不是可检查的。 □

练习 A.20 对于陈述“可判定的(可检查的,可表达的)集合对补(并,交)运算是封闭的”,试给出证明或反例。注意,这是九个问题而不是三个。 □

练习 A.21 试证明 $H_0 = \{c \in \mathbf{Com} \mid \mathscr{C}[c]\sigma_0 \neq \perp\}$ 是可检查的。 □

A.5 马蒂雅塞维奇定理

现在,我们更加严密地考察究竟是什么使得 **Assn** 断言成真(成假)问题不是可检查的,更不用说是可判定的。看起来问题的根源似乎是量词“∀”和“∃”,因为对它们完成一次检查好像要无限地搜索下去。其实,这是直觉的误导。**Assn** 断言的“困难部分”更多的是与数的加法和乘法性质的交互作用有关,而不是量词。特别地,我们设 *PlusAssn* 为不含有乘法符号的断言,同样,*TimesAssn* 为不含有加法或减法符号的断言,那么,*PlusAssn* 和 *TimesAssn* 的有效性实际上都是可判定的,而且存在着常见的证明所有有效 *PlusAssn* 断言的逻辑系统,*TimesAssn* 也是同样的。这些事实一点也不明显,在这里,我们不打算给出这些过长的、富有创造性的证明。

另一方面,当我们局限于不带量词的 **Assn** 断言时,也就是 **Bexp** 断言时,其结果是有效性仍然不是可检查的。这也是“希尔伯特第 10 问题”(给出 $a \in \mathbf{Aexp}$,判定 a 是否有一个整数向量根)不可判定性的一个直接推论,更准确地,令

$$H_{10} = \{a \in \mathbf{Aexp} \mid \text{对某些 } \sigma \in \Sigma\ \sigma \models a = 0\}$$

这可以理解为集合

$$\{\#a \mid \text{对某些 } \sigma \in \Sigma\ a \in \mathbf{Aexp} \text{ 且 } \sigma \models a = 0\}$$

不是 **N** 的一个可判定子集。

定理 A.22(马蒂雅塞维奇,1970) H_{10} 不是可判定的。 347

该定理是 20 世纪在数学和逻辑学上取得的重大成果之一。马蒂雅塞维奇(Matijasevic)(俄罗斯人)在美国人 Davis、Putnam 和 Robinson 早期工作的基础上,利用整数多项式研究如何“编程”时,获得了该定理。定理的证明只用了初等数论,却需要花好几个星期的时间,用一系列讲座来讲解。

练习 A.23 试说明为什么 H_{10} 是可检查的，而 $\overline{H}_{10} = \mathbf{Aexp} \backslash H_{10}$ 不是可检查的。 □

马蒂雅塞维奇事实上证明了以下的一般结论。

定理 A.24（多项式程序设计） 设 M 是一个非负整数的递归可枚举集，则存在一个 $a \in \mathbf{Aexp}$，使得 M 为 a 的值域内的非负整数集合。

回忆 $a \in \mathbf{Aexp}$ 可以看作是描述了一个整数上的多项式函数。特别地，a 的值域是 $\mathrm{Rge}(a) =_{def} \{\mathscr{A}[\![a]\!]\sigma \mid \sigma \in \Sigma\}$。

练习 A.25

(i) 根据多项式程序设计定理，证明：

$$\{a \in \mathbf{Aexp} \mid \#a \in \mathrm{Rge}(a)\}$$

不是可检查的。

(ii) 试说明为什么希尔伯特第 10 问题的不可判定性可以由多项式程序设计定理得出。

□

现在，我们可以得出：形如“$\neg(a = 0)$”的 **Assn** 断言的有效性问题不是可检查的。设

$$\mathbf{ValidNonEq} = \{\neg(a = 0) \mid a \in \mathbf{Aexp} \text{ 且} \models \neg(a = 0)\}$$

推论 A.26 **ValidNonEq** 不是可检查的。

证明 我们有 $a \in \overline{H}_{10}$ 当且仅当 $\neg(a = 0) \in \mathbf{ValidNonEq}$。因此，如果 c 是 **ValidNonEq** 的一个检查者，则

$$X_1 := \mathrm{mkneg}(\mathrm{mkeq}(X_1, \mathrm{mknum}(0)));c$$

348 也就是 $\overline{H}_{10}$ 的检查者。 □

另一方面，再看一个简单而有益的例子——有效方程，即形如“$a_1 = a_2$”的 **Assn** 断言，它是可判定的，而且可以恰当地公理化。

我们先给出“可证明”方程的归纳定义，用 $\vdash e$ 表示方程 e 是可证明的。

$$\vdash a = a \qquad \text{（自反性）}$$

$$\frac{\vdash a_1 = a_2}{\vdash a_2 = a_1} \qquad \text{（对称性）}$$

$$\frac{\vdash a_1 = a_2 \quad \vdash a_2 = a_3}{\vdash a_1 = a_3} \qquad \text{（传递性）}$$

$$\frac{\vdash a_1 = a_2}{\vdash a_1 \ \mathbf{op}\ a = a_2 \ \mathbf{op}\ a} \qquad \text{（右同余）}$$

$$\frac{\vdash a_1 = a_2}{\vdash a \ \mathbf{op}\ a_1 = a \ \mathbf{op}\ a_2} \qquad \text{（左同余）}$$

其中 $\mathbf{op} \in \{+, -, \times\}$

$$\vdash (a_1 \ \mathbf{op}'\ a_2)\mathbf{op}'\ a_3 = a_1 \ \mathbf{op}'\ (a_2 \ \mathbf{op}'\ a_3) \qquad \text{（结合律）}$$

$$\vdash a_1 \ \mathbf{op}'\ a_2 = a_2 \ \mathbf{op}'\ a_1 \qquad \text{（交换律）}$$

其中 $\mathbf{op}' \in \{+, \times\}$

$$\vdash a + 0 = a \qquad (+\text{ 单位元})$$
$$\vdash a \times 1 = a \qquad (\times\text{ 单位元})$$
$$\vdash a - a = 0 \qquad (\text{加法逆元})$$
$$\vdash a - b = a + ((-1) \times b) \qquad (\text{负 1})$$
$$\vdash a_1 \times (a_2 + a_3) = (a_1 \times a_2) + (a_1 \times a_3) \qquad (\text{分配律})$$
$$\vdash (-n) = (-1) \times n \qquad (\text{负数})$$
$$\vdash 1 + 1 = 2,\ \vdash 2 + 1 = 3,\ \vdash 3 + 1 = 4, \cdots \qquad (\text{后继})$$

349

定理 A.27 $\vdash a_1 = a_2$ 当且仅当 $\models a_1 = a_2$。

证明 （$\Rightarrow$:证明系统的可靠性）由“$\vdash$”的归纳定义可直接得到。我们注意到,这些熟悉的规则（公理可看作是没有前提的规则）都保持了有效性。

（$\Leftarrow$:证明系统的完备性）用我们所选择的公理和规则完全可以将每一个表达式 a 归约成一个“标准型”$\hat{a}$，并且满足性质

$$\models a_1 = a_2 \text{ 当且仅当 } \hat{a}_1 \equiv \hat{a}_2$$

一个标准型或者是0，或者是不同单项的和。每个单项（存储单元的乘积）的存储单元按下标的递增次序排列,并从左边加上括号。此外,每个单项有一个非零“系数”n,同时含非零系数的单项按其次数（即长度）的递减次序相加,次数相同的则按字母表次序排列,然后用加号连接起来。 □

例如,设 $a \in \mathbf{Aexp}$ 为

$$2 - ((X_3)^2 - ((X_2)^2((X_3 + 2X_4)X_2) + X_3X_4(X_3)^2))$$

则 $\hat{a}$ 可以表示为

$$(X_2)^3X_3 + 3(X_2)^3X_4 - (X_3)^2 + 2$$

用通常的数学简写式表示的 a 和 $\hat{a}$ 中省略了乘号和一些括号,用幂表示乘的次数,等等。标准型 $\hat{a} \in \mathbf{Assn}$ 可以更为形式地表示如下:

$$\begin{aligned}(((1 \times (((X_2 \times X_2) \times X_2) \times X_3)) &+ (3 \times (((X_2 \times X_2) \times X_2) \times X_4))) \\ &+ ((-1) \times (X_3 \times X_3))) + (2 \times 1)\end{aligned}$$

我们将 1 看作是次数为零的单项式。

这里的思路是,首先,减法可用（负 1）公理消去。反复应用分配律移去加法项上的乘积。最终得到由存储单元和数的乘积的和构成的表达式。在乘积项内部用结合律和交换律排序,在加法项中也可按此次序排列。同类项的系数可由分配律合并,单项将有一个它们各自系数的乘积的和,再用数和单位元公理以及结合律、交换律和分配律将其化简成一系列 1 之和,最后归约为一个数。说得够多了;因此我们有下面的引理。 350

引理 A.28 对每个 $a \in \mathbf{Aexp}$,都存在一个标准型 $\hat{a} \in \mathbf{Aexp}$,使得 $\vdash a = \hat{a}$。

现在我们陈述关于整数多项式的一个事实。

事实 如果 $\hat{a}_1$ 和 $\hat{a}_2$ 是语法上不相同的两个标准型,则 $\mathscr{A}[\![\hat{a}_1]\!] \neq \mathscr{A}[\![\hat{a}_2]\!]$。

练习 A.29 证明这个事实。 □

证明 （完备性） 我们现在可以证明完备性。设$\models a_1 = a_2$，即$\mathscr{A}[a_1] = \mathscr{A}[a_2]$。由引理，$\vdash a_i = \hat{a}_i$，由可靠性，对$i = 1,2$，有$\models a_i = \hat{a}_i$，故$\mathscr{A}[\hat{a}_1] = \mathscr{A}[a_1] = \mathscr{A}[a_2] = \mathscr{A}[\hat{a}_2]$。再由以上事实，$\hat{a}_1$与$\hat{a}_2$实际上语法等同，因此有

$$\vdash a_1 = \hat{a}_1 \text{ 且 } \vdash a_2 = \hat{a}_1$$

由对称性和传递性，我们得到$\vdash a_1 = a_2$。 □

A.6 进一步阅读资料

以上论述的内容基于 Albert Mayer 的讲义，经作者修改而成。关于马蒂雅塞维奇定理的证明可参阅文献[35]。第 7 章已提到的 Crossley[34]、Kleene[54]、Mendelson[61]和 Enderton[38]的著作以及 Kfoury、Moll 和 Arbib 等人的合著[11]中有关本附录内容的论述都是相近的。Cutland 的著作[20]是一本很好的书，对此有着更为传统的数学描述，可作为阅读 Rogers 的百科全书式著作[86]的预备读物。

351

参考文献

[1] Abramsky, S., "The lazy lambda calculus." In Research Topics in Functional Programming (ed. Turner,D.A.), The UT Year of Programming Series, Addison-Wesley, 1990.

[2] Abramsky, S., "Domain theory in logical form." In IEEE Proc. of Symposium on Logic in Computer Science, 1987. Revised version in Annals of pure and Applied Logic, 51, 1991.

[3] Abramsky, S., "A computational interpretation of linear logic." To appear in Theoretical Computer Science.

[4] Aczel, P., "An introduction to inductive definitions." A chapter in the **Handbook of Mathematical Logic**, Barwise, J., (ed), North Holland, 1983.

[5] Alagić, S., and Arbib, M., "The design of well-structured and correct programs." Springer-Verlag, 1978.

[6] Andersen, H.R., "Model checking and boolean graphs." Proc. of ESOP 92, Springer-Verlag Lecture Notes in Computer Science vol.582, 1992.

[7] Andersen, H.R., "Local computation of alternating fixed-points." Tehnical Report No. 260, Computer Laboratory, University of Cambridge, 1992.

[8] Apt, K.R., "Ten years of Hoare's Logic: a survey." TOPLAS, 3, pp. 431-483, 1981.

[9] Apt, K.R, and Olderog, E-R., "**Verification of Sequential and Concurrent Programs**," Springer-Verlag, 1991.

[10] Arbib, M., and Manes, E., "**Arrows, structures and functors**." Academic Press, 1975.

[11] Kfoury, A.J., Moll, R.N. & Arbib, M.A., "**A programming approach to computability**." Springer-Verlag, 1982.

[12] Backhouse, R., "**Program construction and verification**." Prentice Hall, 1986.

[13] de Bakker, J., "**Mathematical theory of program correctness**." Prentice-Hall, 1980.

[14] Barendregt, H., "**The lambda calculus, its syntax and semantics**." North Holland, 1984.

[15] Barr, M., and Wells, C., "**Category theory for computer science**." Prentice-Hall, 1990.

[16] Berry, G., Curien, P-L., and Lévy, J-J., "Full abstraction for sequential languages: the state of the art. In Nivat, M., and Reynolds, J., (ed), Algebraic Methods in Semantics, Cambridge University Press, 1985.

[17] Sørensen, B.B., and Clausen, C., "Adequacy results for a lazy functional language with recursive and polymophic types." DAIMI Report, University of Aarhus, submitted to Theoretical Computer Science.

[18] Camilleri, J.A., and Winskel, G., "CCS with priority choice." Proc. of Symposium on Logic in Computer Science, Amsterdam, IEEE, 1991. Extended version to appear in Information and Computation.

[19] Crole, R., "Programming metalogics with a fixpoint type." University of Cambridge Computer Laboratory Technical Report No. 247, 1992.

[20] Cutland, N.J., "Computability: an introduction to recursive function theory." Cambridge University Press, 1983.

[21] Bird, R., "**Programs and machines**." John Wiley, 1976.

[22] Bird, R., and Wadler, P., "**Introduction to functional programming**." Prentice-Hall, 1988.

[23] Clarke, E.M. Jr., "The characterisation problem for Hoare Logics" in Hoare, C.A.R. and Shepherdson, J.C. (eds.), " Mathematical logic and programming languages." Prentice-Hall, 1985.

[24] Clarke, E.M., Emerson, E.A., and Sistla, A.P., "Automatic verification of finite state concurrent sytems using temporal logic." Proc. of 10th Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, 1983.

[25] Cleaveland, R., "Tableau-based model checking in the propositional mu-calculus." Acta Informatica, 27, 1990.

[26] Clément, J., Despeyroux, J., Despeyroux, T., and Kahn, G., "A simple applicative language: mini-ML." Proc. of the 1986 ACM Conference on Lisp and Functional Programming, 1986.

[27] Cosmadakis, S.S., Meyer, A.R., and Riecke, J.G., "Completeness for typed lazy languages (Preliminary report)." Proc. of Symposium on Logic in Computer Science, Philadelphia, USA, IEEE, 1990.

[28] Despeyroux, J., "Proof of translation in natural semantics." Proc. of Symposium on Logic in Computer Science, Cambridge, Massachusetts, USA, IEEE, 1986.

[29] Despeyroux, T., "Typol: a formalism to implement natural semantics." INRIA Research Report 94, Roquencourt, France, 1988.

[30] Cleaveland, R., Parrow, J. and Steffen, B., "The Concurrency Workbench." Report of LFCS, Edinburgh University, 1988.
[31] Clocksin, W.F., and Mellish, C., "**Programming in PROLOG**." Springer-Verlag, 1981.
[32] Cohen, "**Programming for the 1990's**". Springer-Verlag, 1991.
[33] Cook, S.A., "Soundness and completeness of an axiom system for program verification." SIAM J. Comput. 7, pp. 70-90, 1978.
[34] Crossley, J.N., "**What is mathematical logic?**." Oxford University Press, 1972.
[35] Davis, M., "Hilbert's tenth problem is unsolvable." Am.Math.Monthly 80, 1973.
[36] Dijkstra, E.W., "**A discipline of programming**." Prentice-Hall, 1976.
[37] Emerson, A. and Lei, C., "Efficient model checking in fragments of the propositional mu-calculus." Proc. of Symposium on Logic in Computer Science, 1986.
[38] Enderton, H.B., "**A mathematical introduction to logic**." Academic Press, 1972.
[39] Enderton, H.B., "**Elements of set theory**." Academic Press, 1977.
[40] Girard, J-Y., Lafont, Y., and Taylor, P., "**Proofs and types**." Cambridge University Press, 1989.
[41] Good, D.I., "Mechanical proofs about computer programs." in Hoare, C.A.R., and Shepherdson, J.C. (eds.), "**Mathematical Logic and Programming Languages**." Prentice-Hall, 1985.
[42] Gordon, M.J.C., "**Programming language theory and its implementation**." Prentice-Hall, 1988.
[43] Gordon, M.J.C., HOL: A proof generating system for higher-order logic, in **VLSI Specification, Verification and Synthesis**, (ed. Birtwistle, G., and Subrahmanyam, P.A.) Kluwer, 1988.
[44] Gries, D., "**The science of programming**." Springer Texts and Monographs in Computer Science, 1981.
[45] Hindley, R., and Seldin, J.P, "**Introduction to combinators and lambda-calculus**." Cambridge University Press, 1986.
[46] Godskesen, J.C., and Larsen, K.G., and Zeeberg, M., "TAV (Tools for Automatic Verification) users manual." Technical Report R 89-19, Department of Mathematics and Computer Science, Aalborg University, 1989. Presented at the workshop on Automated Methods for Finite State Systems, Grenoble, France, June 1989.
[47] Halmos, P.R., "**Naive set theory**." Litton Ed Publ. Inc., 1960.
[48] Hennessy, M.C, "**Algebraic theory of processes**." MIT Press, 1988.
[49] Hoare, C.A.R., "Communicating sequential processes." CACM, vol.21, No.8, 1978.
[50] Hoare, C.A.R., "**Communicating sequential processes**." Prentice-Hall, 1985.
[51] Huet, G., "A uniform approach to type theory." In **Logical Foundations of Functional Programming** (ed. Huet,G.), The UT Year of Programming Series, Addison-Wesley, 1990.
[52] Jensen, C.T., "The Concurrency Workbench with priorities." To appear in the proceedings of Computer Aided Verification, Aalborg, 1991, Springer-Verlag Lecture Notes in Computer Science.
[53] Johnstone, P.T., "**Stone spaces**." Cambridge University Press, 1982.
[54] Kleene, S.C., "**Mathematical logic**." John Wiley, 1967.
[55] Kozen, D., "Results on the propositional mu-calculus," Theoretical Computer Science 27, 1983.
[56] Lamport, L., "The temporal logic of actions." Technical Report 79, Digital Equipment Corporation, Systems Research Center, 1991.
[57] Larsen, K.G., "Proof systems for Hennessy-Milner logic." Proc. CAAP, 1988.
[58] Loeckx, J. and Sieber, K. "**The foundations of program verification**." John Wiley, 1984.
[59] Manna, Z., "**Mathematical theory of computation**." McGraw-Hill, 1974.
[60] Manna, Z., and Pnueli, A., "How to cook a temporal proof system for your pet language." Proc. of 10th Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, 1983.
[61] Mendelson, E., "**Introduction to mathematical logic**." Van Nostrand, 1979.
[62] Milner, A.J.R.G., "Fully abstract models of typed lambda-calculi." Theoretical Computer Science 4, 1977.
[63] Milner, A.J.R.G., "**Communication and concurrency**." Prentice Hall, 1989.
[64] Milner, A.J.R.G., "Operational and algebraic semantics of concurrent processes." A chapter in **Handbook of Theoretical Computer Science**, North Holland, 1990.
[65] Mitchell, J.C., "Type systems for programming languages." A chapter in **Handbook of Theoretical Computer Science**, North Holland, 1990.
[66] Moggi, E., "Categories of partial morphisms and the lambda$_p$-calculus." In proceedings of Category Theory and Computer Programming, Springer-Verlag Lecture Notes in Computer Science vol.240, 1986.
[67] Moggi, E., "Computational lambda-calculus and monads." Proc. of Symposium on Logic in Computer Science, Pacific Grove, California, USA, IEEE, 1989.
[68] Mosses, P.D., "Denotational semantics." A chapter in **Handbook of Theoretical Computer Science**, North Holland, 1990.
[69] Nielson. H.R., and Nielson, F., "**Semantics with applications: a formal introduction**." John Wiley, 1992.
[70] inmos, "**Occam programming manual**." Prentice Hall, 1984.

[71] Ong, C-H.L., "The lazy lambda calculus: an investigation into the foundations of functional programming." PhD thesis, Imperial College, University of London, 1988.
[72] Parrow, J., "Fairness properties in process algebra." PhD thesis, Uppsala University, Sweden, 1985.
[73] Paulson,L.C., "**ML for the working programmer**." Cambridge University Press, 1991.
[74] Paulson, L.C., "**Logic and computation: interactive proof with Cambridge LCF**." Cambridge University Press, 1987.
[75] Pitts, A., "Semantics of programming languages." Lecture notes, Computer Laboratory, University of Cambridge, 1989.
[76] Pitts, A., "A co-induction principle for recursively defined domains." University of Cambridge Computer Laboratory Technical Report No.252, 1992.
[77] Plotkin, G.D., "Call-by-name, Call-by-value and the lambda calculus." Theoretical Computer Science 1, 1975.
[78] Plotkin, G.D., "LCF considered as programming language." Theoretical Computer Science 5, 1977.
[79] Plotkin, G.D., "A powerdomain construction." SIAM J. Comput.5, 1976.
[80] Plotkin, G.D., "The Pisa lecture notes." Notes for lectures at the University of Edinburgh, extending lecture notes for the Pisa Summerschool, 1978.
[81] Plotkin, G.D., "Structural operational semantics." Lecture Notes, DAIMI FN-19, Aarhus University, Denmark, 1981 (reprinted 1991).
[82] Plotkin, G.D., "An operational semantics for CSP." In Formal Description of Programming Concepts II, Proc. of TC-2 Work. Conf. (ed. Bjørner, D.), North-Holland, 1982.
[83] Plotkin, G.D., "Types and partial functions." Notes of lectures at CSLI, Stanford University, 1985.
[84] Prawitz, D., "**Natural deduction, a proof-theoretical study**." Almqvist & Wiksell, Stockholm, 1965.
[85] Reisig, W., "**Petri nets: an introduction**." EATCS Monographs on Theoretical Computer Science, Springer-Verlag, 1985.
[86] Rogers, H., "**Theory of recursive functions and effective computability**." McGraw-Hill, 1967.
[87] Roscoe,A.W., and Reed,G.M., "**Domains for denotational semantics**." Prentice Hall, 1992.
[88] Schmidt, D., "**Denotational semantics: a methodology for language development**." Allyn & Bacon, 1986.
[89] Scott, D.S., "Lectures on a mathematical theory of computation." PRG Report 19, Programming Research Group, Univ. of Oxford, 1980.
[90] Scott, D.S., "Domains for denotational semantics." In proceedings of ICALP '82, Springer-Verlag Lecture Notes in Computer Science vol.140, 1982.
[91] Scott, D.S., and Gunter, C., "Semantic domains." A chapter in **Handbook of Theoretical Computer Science**, North Holland, 1990.
[92] Smyth, M., "Powerdomains." JCSS 16(1), 1978.
[93] Stirling, C. and Walker D., "Local model checking the modal mu-calculus." Proc.of TAPSOFT, 1989.
[94] Stoughton, A., "**Fully abstract models of programming languages**." Pitman, 1988.
[95] Stoy, J., "**Denotational semantics: the Scott-Strachey approach to programming language theory**." MIT Press, 1977.
[96] Tarski, A., "A lattice-theoretical fixpoint theorem and its applications." Pacific Journal of Mathematics, 5, 1955.
[97] Tennent, R.D., "**Principles of programming languages**." Prentice-Hall, 1981.
[98] Vickers, S., "**Topology via logic**." Cambridge University Press, 1989.
[99] Vuillemin, J.E., "Proof techniques for recursive programs." PhD Thesis, Stanford Artificial Intelligence Laboratory, Memo AIM-218, 1973.
[100] Walker, D., "Automated analysis of mutual exclusion algorithms using CCS." Formal Aspects of Computing 1, 1989.
[101] Wikström, Å. "**Functional programming using Standard ML**." Prentice-Hall, 1987.
[102] Winskel, G., "On powerdomains and modality." Theoretical Computer Science 36, 1985.
[103] Winskel, G. and Larsen, K., "Using information systems to solve recursive domain equations effectively." In the proceedings of the conference on Abstract Datatypes, Sophia-Antipolis, France, Springer-Verlag Lecture Notes in Computer Science vol.173, 1984.
[104] Winskel, G., "Event structures." Lecture notes for the Advanced Course on Petri nets, September 1986, Springer-Verlag Lecture Notes in Computer Science, vol.255, 1987.
[105] Winskel, G., "An introduction to event structures." Lecture notes for the REX summerschool in temporal logic, May 88, Springer-Verlag Lecture Notes in Computer Science, vol.354, 1989.
[106] Winskel, G., "A note on model checking the modal nu-calculus." Theoretical Computer Science 83, 1991.
[107] Winskel, G., and Nielsen, M., "Models for concurrency." To appear as a chapter in the **Handbook of Logic and the Foundations of Computer Science**, Oxford University Press.
[108] Zhang, G-Q., "**Logic of domains**." Birkhäuser, 1991.

索　引

索引中的页码为英文原书页码，与书中页边标注的页码一致。

A

abstract syntax（抽象语法），12，26
Ackermann's function（Ackermann 函数），175
adequacy（适用性），191，216，262，288
application（应用），129
applicative（应用式），141
approximable mapping（逼近映射），245
axiomatic semantics（公理语义），77，89

B

Backus- Naur form（巴科斯-诺尔范式），11
Bekic's theorem（贝伊克定理），162，163
binary trees（二叉树），224
bisimulation（互模拟），316，317，320，335，336
BNF（巴克斯-诺尔范式），11
bound variable（约束变量），81

C

Calculus of Communicating Systems（通信系统演算），308
call-by-name（传名调用），142
call-by-need（传需调用），183
call-by-value（传值调用），142
canonical forms（标准型）
  call-by-name（传名调用），201
  call-by-value（传值调用），186
  eager（活性），186，255
  lazy（惰性），201
Cantor's diagonal argument（康托尔对角线方法），8
cases-notation for cpo's（完全偏序的 case 记号），134，138
category（范畴），139
cartesian closed（笛卡儿封闭），139
  coproducts（对偶积），139
  products（积），139
CCS（通信系统演算），308，335
  operational semantics（操作语义），313
  pure（纯），311
  recursive definition（递归定义），315
  syntax（语法），309，312
channel（通道），303
checkable（可检查的），338
closed family（闭族），228
closed under rules（对规则封闭），42
commands（命令），13
communicating processes（通信进程），303
Communicating Sequential Processes（顺序通信进程），307
communication by shared variables（共享变量进行通信），298
complete lattice（完全格），74
complete partial order（cpo），（完全偏序），68，70
compositional（复合），60
computability（可计算性），337
computationally feasible（计算可行性），122
concrete syntax（具体语法），12
concurrency（并发），297
Concurrency Workbench，335
conditional on cpo's（完全偏序条件），134
configuration（格局），14，19
context（上下文），218
continuous function（连续函数），68，120
continuous in each argument separately（在每个变量上分别连续），127
continuous in variables（对变量连续），136
convergence（收敛），
  eager（活性），191，262
  lazy（惰性），204，288
cpo（完全偏序），68，70，119

algebraic（代数），230
bounded complete（完全有界的），230
constructions（构造），123
function space（函数空间），128
lifting（提升），131
product（积），125
sum（和），133
discrete（离散的），120，124
discrete（flat）（离散(平坦)的），70
finite element（有限元素），230
injection function（内射函数），133
isomorphism（同构），124
omega algebraic（ω 代数），230
projection function（投影函数），125
with bottom（含底），70，119
CSP（顺序通信进程），307，335
currying（柯灵），129

D

deadlock（死锁），307，317
decidable（可判定的），338
declaration（声明），141
local（局部），161
denotational semantics（指称语义），55
higher types（高阶类型），
eager（活性），188
lazy（惰性），203
IMP，58
REC，
call-by-name（传名调用），154
call-by-value（传值调用），144
recursive types（递归类型），
eager（活性），257
lazy（惰性），281
derivation（推导），14，16
induction on（对推导的归纳），35
subderivation（子推导），35
deterministic evaluation（确定的求值），28
dI-domains（dI 域），249
Dijkstra's guarded commands（迪杰斯特拉的卫式命令），298
domain（域），119
domain theory（域论），119
metalanguage（元语言），135，172
application（应用），136
cases-notation（case 记号），138
lambda abstraction（λ 抽象），137
let-notation（let 记号），137
mu-notation（mu 记号），138
tupling（元组），136

E

eager evaluation（活性求值），183
eager language（活性语言），
recursive types（递归类型），251
embedding-projection pairs（嵌套投影序偶），236
environment（环境），144，154，188，203，258，284
for types（类型的环境），258，281
Euclid's algorithm（欧几里得算法），33，96，301
expressible set（可表达的集合），345
expressiveness（可表达性），100，101
extension of assertion（断言的扩充），86

F

fairness（公平），327
finite-state process（有限状态进程），323
Fixed-point induction（不动点归纳法），166
fixed-point operator（不动点算子），209
eager（活性），214，272，275
lazy（惰性），209，292，293
Fixed-Point Theorem（不动点定理），71，121
Floyd-Hoare rules（费洛伊德-霍尔规则），77
free variable（自由变量），81
full abstraction（完全抽象），215，221
function（函数），6
composition（复合），7
continuous（连续的），68，71，120
continuous in each argument separately（在每个变量上分别连续），127
direct image（正象），9
fixed point（不动点），71
identity（恒等），8
IMP computable（IMP 可计算的），337
inverse image（逆象），9

maximum fixed point（最大不动点），75
monotonic（单调的），71，120
order-monic（相对于序关系是内射的），170
partial（部分），7
prefixed point（前缀不动点），71
recursive（递归的），337
stable（稳定的），249
strict（严格的），132
total（完全），7
function space of cpo's（完全偏序的函数空间），128
function type（函数类型），
eager（活性），251
lazy（惰性），278
functional language（函数式语言），251，295
eager（活性），183，251
lazy（惰性），200，278

G

gcd（最大公约数），33，96
glb（最大下界），74
Godel number（哥德尔数），341
Godel's beta predicate（哥德尔 β 谓词），101，110
Godel's Incompleteness Theorem（哥德尔不完备性定理），99，110，343
greatest common divisor（gcd）（最大公约数），33
greatest lower bound（最大下界），74
guarded commands（卫式命令），298，335

H

halting problem（停机问题），342
Haskell，251
Hennessy-Milner logic（亨纳斯-米尔纳逻辑），316
Hilbert's Tenth Problem（希尔伯特第 10 问题），347
Hoare logic（霍尔逻辑），89，97
Hoare rules（霍尔规则），77，89
completeness（完备性），91，99
relative completeness（相对完备性），99，100
soundness（可靠性），91
HOL，93

I

IMP（命令式语言），11
checkable（可检查的），338
computable（可计算的），337
decidable（可判定的），338
denotational semantics（指称语义），60
evaluation rules（求值规则），14，17
execution rules（执行规则），20
syntax（语法），11
imperative language（命令式语言），11
inclusive in each argument separately（在每个变量上分别包含），171
inclusive predicates（包含谓词），167
constructions（构造），
logical operations（逻辑运算），169
substitution（代入），168
inclusive properties（包含性质），166
constructions（构造），
finite unions（有限并集），169
function space（函数空间），171
intersections（交集），169
inverse image（逆象），168
lifting（提升），171
products（积），170
sum（和），171
incompleteness（不完备性），337
induction（归纳法），27
inductive definition（归纳定义），41，54
information system（信息系统），223
consistency relation（一致性关系），226
constructions（构造），236
lifted function space（提升函数空间），243
lifting（提升），237
product（积），241
sums（和），239
cpo of information systems（信息系统的完全偏序），233
definition（定义），226
entailment relation（推导关系），226
tokens（符号），226
interpretation（解释），84
invariant（不变式），78，90

isomorphism（同构），124

K

Knaster-Tarski Theorem（克纳斯特-塔尔斯基定理），74，322

L

lambda calculus（λ演算），296
  eager（活性），267
  equational theory（等式理论），269
  eager typed（活性类型），183
  denotational semantics（指称语义），188
  operational semantics（操作语义），186
  lazy（惰性），290
  equational theory（等式理论），291
  lazy typed（惰性类型），200
  denotational semantics（指称语义），203
  operational semantics（操作语义），201
  lambda notation（λ记号），7
lazy evaluation（惰性求值），183
lazy language（惰性语言），
  recursive types（递归类型），278
lazy lists（惰性表），121，287
lazy natural numbers（惰性自然数），279，286
LCF，93，139
least common multiple（最小公倍数），84
least upper bound（lub）（最小上界），69
let-notation（let记号），132
lifting of cpo's（完全偏序的提升），131
lists（表），179，224，254，256
  append（添加），179
  cons，179
  lazy（惰性），287
  of integers（整数的），179
local model checking（局部模型检查），325，327
location（存储单元），11，39，48
logical operations（逻辑运算），1
logical relation（逻辑关系），217
  eager（活性），193，263
  lazy（惰性），205
lower bound（下界），74
lub（最小上界），69

M

mathematical induction（数学归纳法），27
Matijasevic Theorem（马蒂雅塞维奇定理），351
Matijasevic's Theorem，347
metavariables（元变量），11
Miranda，251
modal logic（模态逻辑），316
modal mu-calculus（模态μ演算），321
modal nu-calculus（模态ν演算），321
model checking（模型检查），325
  local（局部），325
monotonic function（单调函数），120
mu-calculus（μ演算），321，335

N

natural semantics（自然语义），16，26
Noetherian induction（诺特归纳法），32
nondeterminism（不确定性），297
nu-calculus（ν演算），321，335

O

observation（观察），215
Occam，307，335
omega chain（ω链），70
operational semantics（操作语义），11
  CCS（通信系统演算），309
  communicating processes（通信进程），303
  guarded commands（卫式命令），298
  higher types（高阶类型），
  eager（活性），186
  lazy（惰性），201
  IMP（命令式语言），13
  pure CCS（纯CCS），313
  REC（递归语言），
  call-by-name（传名调用），153
  call-by-value（传值调用），143
  recursive types（递归类型），
  eager（活性），255
  lazy（惰性），278
  shared-variable communication（共享变量通信），297

operator on sets（集合上算子），
least fixed point（最小不动点），59
operators on sets（集合上算子），52
continuous（连续的），54
fixed points（不动点），52
increasing（递增的），54
monotonic（单调），52
order-monic（相对于序关系是内射的），170
Orwell，251

P

parallel composition（并行复合），303
parallelism（并行性），297
Park induction（帕克归纳法），163
partial correctness（部分正确性），
proof rules（证明规则），89
partial correctness assertion（部分正确性断言），79
annotated（有注释的），113
partial correctness predicate（部分正确性谓词），115
partial order（偏序），69
partial recursive function（部分递归函数），337
Petri nets（Petri 网），336
Plotkin powerdomain（Plokin 幂域），249
polynomial programming（多项式程序设计），348
ports（端口），308
powerdomain（幂域），249，336
predicate calculus（谓词演算），81
predicate transformer（谓词转换器），115
predomain（前域），
Scott（斯科特），230
predomains（前域），70，249
product of cpo's（完全偏序的积），125
product type（积类型），
eager（活性），251
lazy（惰性），278

Q

quantifiers（量词），1

R

REC（递归语言），141
call-by-name（传名调用），
denotational semantics（指称语义），154
operational semantics（操作语义），153
call-by-value（传值调用），
denotational semantics（指称语义），144
operational semantics（操作语义），143
semantics equivalent（语义等价），149
syntax（语法），141
recursion equations（递归方程），141
recursive function（递归函数），337
recursive set（递归集），338
recursive types（递归类型），251，295
eager language（活性语言），251
denotational semantics（指称语义），257
operational semantics（操作语义），255
typing rules（类型规则），252
lazy language（惰性语言），278
operational semantics（操作语义），278
lazy lists（惰性表），287
lazy natural numbers（惰性自然数），279，286
lists（表），254，256
natural numbers（自然数），253，256
recursively enumerable（递归可枚举），338
relation（关系），6
composition（复合），7
direct image（正象），9
equivalence relation（等价关系），9
identity（恒等），10
inverse image（逆象），9
transitive closure（传递闭包），10
well-founded（良基的），31
restriction（限制），304
rule（规则），35
axiom（公理），35
conclusion（结论），35
finitary（有限的），35，71
instance（实例），15，35
premise（前提），35
set defined by rules（规则定义的集合），41
rule induction（规则归纳法），41
general principle（一般原理），41
special principle（特殊原理），44
rules（规则），

set closed under rules（对规则封闭的集合），42
Russell's paradox（罗素悖论），3

S

Scott closed（斯科特闭），167
Scott domain（斯科特域），228
Scott predomain（斯科特前域），230
Scott topology（斯科特拓扑），123
Scott's fixed-point induction（斯科特不动点归纳法），166
sequentiality（串行性），218
set（集合），
  closed under rules（对规则封闭），42
  defined by rules（由规则定义），41
  inductively defined（归纳定义），41
sets（集合），2
  constructions（构造），4
  foundation axiom（基本公理），6
  functions（函数），6
  relations（关系），6
  Russell's paradox（罗素悖论），3
SFP objects（SFP 对象），249
side effects（副作用），26
size of token（符号的大小），263
Standard ML(标准 ML 语言)，251
state（状态），13
state transformer（状态转换器），115
stoppered sequences（带终止符的序列），121，224
streams（流），121，224
strict extension of a function(函数的严格扩展)，132
strongest postcondition（最强后置条件），117
structural induction（结构归纳法），28
structural operational semantics（结构化操作语义），16，26
subderivation（子推导），35
substitution（代入），82，103，269
sum of cpo's（完全偏序的和），133
sum type（和类型），
  eager（活性），251
  lazy（惰性），278
sum types（和类型），219
  eager（活性），219
  lazy（惰性），219

T

Tarski's Theorem（塔尔斯基定理），74，322
TAV System（TAV 系统），335
temporal logic（时态逻辑），336
tidy command（整洁命令），337
transition relations（转换关系），21
transition systems（转换系统），21
truth values（真值），11
  cpo（完全偏序），122
  operations（运算），57
typable term（有类型的项），184
type environment（类型环境），258，281
type variables（类型变量），251，278
types（类型），183
  eager（活性），251
  function（函数），
  eager（活性），251
  lazy（惰性），278
  higher（高阶），183
  lazy（惰性），278
  product（积），
  eager（活性），251
  lazy（惰性），278
  recursive(递归)，223
  eager（活性），251
  lazy（惰性），278
  sum（和），
  eager（活性），251
  lazy（惰性），278
  typing rules（类型规则），185
typing rules（类型规则），252

U

undecidability(不可判定性)，337，339
universal program(通用程序)，345
until operator（until 操作符），326
upper bound(上界)，69

V

validity(有效性),87
values(值),186
  eager(活性),188,258
  lazy(惰性),203,281
verification condition(验证条件),112,113
  generator(生成器),115

W

weakest liberal precondition(最弱宽松前置条件),101
weakest precondition(最弱前置条件),100
well ordering(良序),181
well-founded induction(良基归纳法),31,174
  principle(原理),32
well-founded recursion(良基递归),40,176,264,289
well-founded relation (良基关系),
  inverse image(逆象),175
  lexicographic product(字典序积),175
  product(积),175
while programs (while 程序),11