Formal Semantics

形式语义学引论

（第二版）

周巢尘 詹乃军 著

Introduction to Formal Semantics

科学出版社

形式语义学引论

（第二版）

周巢尘　詹乃军　著

科学出版社
北京

内 容 简 介

本书为形式语义学入门参考书，简单介绍程序的操作语义、指称语义和公理语义。全书共 7 章：第 1 章介绍操作语义，第 2 章介绍指称语义，第 3 章介绍公理语义，第 4 章介绍过程调用的形式语义，第 5 章介绍非确定程序的形式语义，第 6 章介绍并发程序的形式语义，第 7 章介绍程序的时态语义。

本书可供计算机科学与技术、软件工程等专业的教师、研究生参考，也可供相关领域的科研和工程技术人员阅读参考。

图书在版编目 (CIP) 数据

形式语义学引论 / 周巢尘，詹乃军著. —2 版. —北京：科学出版社，2017.9

ISBN 978-7-03-053383-8

Ⅰ. ①形… Ⅱ. ①周… ②詹… Ⅲ. ①形式语义学－研究 Ⅳ. ①TP301.2

中国版本图书馆 CIP 数据核字 (2017) 第 134871 号

责任编辑：任 静 / 责任校对：桂伟利

责任印制：肖 兴 / 封面设计：迷底书装

科学出版社 出版

北京东黄城根北街 16 号

邮政编码：100717

http://www.sciencep.com

北京通州皇家印刷厂 印刷

科学出版社发行 各地新华书店经销

*

1985 年 8 月湖南科学技术出版社第一版

2017 年 9 月第 二 版 开本：720×1 000 1/16

2017 年 9 月第一次印刷 印张：10 1/4

字数：145 000

定价：68.00 元

（如有印装质量问题，我社负责调换）

再 版 说 明

作为形式语义学的一本入门书，本书的首次出版是1985年，距今已经三十多年。在过去三十多年里，形式语义研究取得了巨大进展，例如，基于博弈的语义、并发程序理论、概率程序理论、信息物理融合系统设计理论、量子计算、生物计算，等等。但是，形式语义学的主要基础部分仍旧为本书第一版本中介绍的内容。再版的目的仍旧是想为初学者提供一本入门书，因而仅仅纠正第一版本中发现的错误，未增加新的内容。但是为了方便读者了解最新进展，每章结尾增加一节，简单介绍该方向在过去三十多年的研究动态。如果读者想全面了解形式语义研究方面的最新进展，建议参考 Jan van Leeuwen 编著的《Handbook of Theoretical Computer Science (B 卷): Formal models and Semantics》（1991 年，Elsevier 出版社和 MIT 出版社）及陆汝钤先生的著作《计算系统的形式语义》（2016 年，上、下册，清华大学出版社）等。

前　言

1. 什么是形式语义学

形式语义学（Formal Semantics）是研究程序设计语言的语义的学问。它以数学为工具，运用符号和公式，严格地解释程序设计语言的语义，使语义形式化，故称形式语义学。

程序设计语言是人类用来和计算机系统进行通信，并控制其工作的人工语言。作为语言，人工语言和自然语言（如汉语、英语等）一样，有其语法（Syntax）、语义（Semantics）和语用（Pragmatics）范畴。程序设计语言的语法是指程序的组成规则；语义是指程序的含义；语用的所指，说法不一，大致包括程序的使用效果。

为了正确、有效地使用程序设计语言，必须了解语言中各个成分的含义，并且要求计算机系统执行这些成分所产生的效果和其含义完全一致。但是程序设计语言的语义通常是由设计者用一种自然语言非形式地解释，实施者和使用者依据各自的理解实现和使用这种语言。由于自然语言本身存在歧义现象，非形式的解释又不严格，故用这种方法解释语义容易造成设计者、用户和实施者对语义的不同理解，影响语言的正确实施和有效使用。程序设计语言中的过程调用语句就是这方面的一个典型例子。人们发现对过程调用语句的非形式的解释可能导致不同的理解，产生不同的效果。

人们对语义精确解释的要求产生了形式语义学，形式语义学的研究始于20世纪60年代初，在程序设计语言ALGOL 60的设计中，第一次明确区分了语言的语法和语义，并使用BNF范式成功地实现了语法的形式描述。语法的形式化大大刺激了语义形式化的研究，围绕ALGOL 60的语义出现了形式语义学早期的研究热潮。

美国斯坦福大学麦克阿瑟教授（J. McCarthy）在国际信息加工联合会1962年大会上作了著名的报告——“通往计算的数学科学”，系统地论述了程序设计语言语义形式化的重要性，和程序正确性、语言的正确实施等的关系，并提出在形式语义学研究中使用抽象语法和状态向量等基本方法。

几十年来，形式语义学的研究取得了巨大进展。它对程序设计语言的设

计、使用和实施产生了深刻的影响，语言的形式语法和形式语义（统称语言的形式定义）已经成为程序语言的必要组成部分。形式语义学也是软件工程学的基础理论之一。从形式语义学的观点看，软件工程学中的软件需求（Requirement）和软件规范（Specification）是在不同详尽程度上对程序语义的刻画，而软件正确性是讨论程序的语义和预期目标的一致；自动程序设计则是研究如何将程序的一种形式语义自动转换成另一种形式语义。

通常的程序设计语言的语法是规定程序组成方法的一些规则，称为具体语法。但在定义程序的语义时，必须首先识别给定的程序，需要分析程序的语法结构。因此，在形式语义学中，使用一种讨论程序分解的语法规则，这种语法称作抽象语法。不同的程序设计语言往往使用不同的记号和表示方式，形式语义学提供的方法则应适用于一切程序设计语言，故抽象语法采用的记号和表示方式也是具体语法的一种抽象。

在定义程序设计语言的语义时，需要一种定义语义的语言，这种语言叫作元语言（Metalanguage）。元语言可以采用已有的数学语言，也可以是专门设计的语言。当然，为了用元语言去定义程序语言的形式语义，必须首先严格定义元语言的语义。

用程序设计语言编写的程序，规定了对计算机系统中数据的一个加工过程。形式语义学的基本方法是用一种元语言将程序加工数据的过程及其结果形式化，从而定义程序的语义。由于形式化中侧重面和使用的数学工具不同，形式语义学可分为四大类：

操作语义学（Operational Semantics），着重模拟数据加工过程中计算机系统的操作；

指称语义学（Denotational Semantics），主要刻画数据加工的结果，而不是加工过程的细节；

代数语义学（Algebraic Semantics），可看作是指称语义学的一个分支，以使用代数学为特征；

公理语义学（Axiomatic Semantics），用公理化的方法描述程序对数据的加工。

2. 本书的选材和组织

鉴于形式语义学在程序设计语言和软件工程中的重要地位，故撰写此书。希望此书成为迫切渴望知识更新的我国软件工作者学习形式语义学的入门书，也可作为高等院校计算机科学等相关专业的教材和教学参考书。

形式语义学的内容极为广泛。国外的一些大学将每类语义学作为单独课程；甚至在讲授指称语义学时，还增设论域理论（Domain Theory）。但针对我国现况，编著一本入门书则为当务之急。当然，这不应该是一本通俗的科普书，只增加读者对一不熟悉领域的一般知识；或者是一本字典，罗列各种新名词，而不给予读者深入的了解。这本书应该介绍形式语义学的基本概念、基本理论和基本方法，使得掌握本书内容的读者易于深入到形式语义学的各个领域。

每类形式语义学都是建立在所用元语言的数学理论基础上的。借助于元语言的理论，才可能严格定义程序设计语言的语义，论证语义的合理性，讨论不同语义定义的优劣，以及语义的其他特性。通过各种语义学途径定义各式各样程序语言成分的语义时，也需要探讨表述方法，以求表述的语义简洁易懂。由于语言成分花样繁多，表述方法也就十分繁杂。集各类语义学的理论基础与表述方法于一书，篇幅过于浩大。作为入门书，只介绍表述方法，不介绍基础理论，又会使读者误认为形式语义学只是一种符号化了的“非形式语义学”。对于我国的软件工作者来说，形式语义学的理论部分比之表述方法恐怕更为陌生。故本书的前三章以一个简单的程序设计语言为例，介绍操作语义学、指称语义学和公理语义学的基本理论和方法，并侧重于基本理论。对其中的概念和事实都以必要的严格形式陈述和剖析，望初学者认真阅读。第 4、5、6 章讨论过程调用语句，非确定和并发程序设计语言的语义，介绍这些语言现象引起的语义学理论和方法的发展。这些方面的语义学研究远远没有完结，非确定性和并发性的研究是当前形式语义学研究的前沿领域。第 7 章介绍时态语义。时态逻辑（Temporal Logic）用于语义学的研究开始于 20 世纪 70 年代后期，近年来蓬勃发展，已成为公理语义学的重要分支。希望这后四章能为有志于深入学习形式语义学或研究形式语义学的读者打下基础。至于操作语义学中的互模拟（Bi-Simulation）的概念，在第一版时（1985 年）正处于萌芽状态，当时未能介绍，再版时也只能割爱了。

作为一本入门书，本书中不讨论语义学中待解决的问题，也不涉及语义学的应用。本书没有提供习题，但在各章内容中（尤其在后四章中）提出了若干问题或者留下若干待证的结论。有兴趣的读者可自己动手补足这些细节，这对于理解形式语义学的理论和方法是很有帮助的。

本书的再版得到国内诸多同行的鼓励和帮助，包括：刘志明、王戟和王怀民等；同时，应明生、王淑灵等帮助审阅了再版部分草稿，并提出许多宝贵修改建议，在此一并表示感谢！

本书再版得到国家自然科学基金委员会项目（NSFC-61625206 和 NSFC-61732001）、国家 973 项目（2014CB340701）、中德合作研究项目 CAP（GZ 1023）及中国科学院和国家外国专家局国际创新团队项目（The CAS/SAFEA International Partnership Program for Creative Research Teams）的资助。

由于著者对代数语义学并不熟悉，本书中只好略去这一内容。其他缺陷和不足也在所难免，望读者不吝指教。

目　　录

第 1 章　操作语义学

§1.1　引　　言

程序设计语言的实施者是按照语言的语义在具体的计算机系统中编制语言的解释程序或者编译程序的。但反过来看，语言在任何计算机系统中的任何一种实施一旦完成，那么对这个计算机系统而言，语言的含义也就完全确定了；也就是说，语言的一种实施可看作是语言语义的一种定义。用语言的实施作为语言的语义定义，也就是将语言成分所对应的计算机系统的操作作为语言成分的语义，故称之为操作语义。当然，语言的语义应该是标准的，不应依附于一个特定的计算机系统、一种具体的实施。因此操作语义学中使用抽象的机器和抽象的解释程序来定义语言的语义。

操作语义学的基本思想来源于程序设计语言的实施。但第一次系统地、严格地陈述这一途径的是英国学者兰丁（P. J. Landin），他于 1964 年 1 月发表了《表达式的机械化求值》一文。文中使用“栈-环境-控制-外贮”抽象机器（简称 SECD 机器），定义了表达式的操作语义。

IBM 公司维也纳实验室 20 世纪 60 年代在研究程序设计语言 PL/I 的形式定义时，提出了描述操作语义的一种元语言——维也纳定义语言（简称 VDL）。1974 年，欧洲计算机制造商联合会（ECMA）和美国国家标准局（ASI）正式建议使用 VDL 定义的 PL/I 的语义作为 PL/I 的一种标准。

1980 年前后，英国爱丁堡大学的计算机科学家们提出了结构化操作语义。它在更一般的数学结构（不必是抽象机器）上用数学的归约关系（Reduction Relation）建立语义的解释系统。这种语义学具有结构化特征，也就是语言中复合成分的语义是由其子成分的语义复合而成。结构化的语义定义可为软件工程中的结构化程序设计方法提供重要原理。

本章中将以一个简单程序设计语言 FLOW 为例，介绍兰丁式的抽象机语义定义和结构化的操作语义定义。FLOW 语言也将用于以后几章中。

§1.2　FLOW 语言

为了侧重讨论程序语言的一些基本控制结构的语义，FLOW 语言中不给出原子语句（如赋值语句）和布尔表达式的语法定义，视作待定参数。

FLOW 的语法：

（1）原子语句集 Asts，用 A 表示其中元素；

（2）布尔表达式集 Bexp，用 B 表示其中元素；

（3）语句集 Sts，用 S 表示其中元素，定义如下：

$$S ::= \text{skip} \mid A \mid S;S \mid (\text{if } B \text{ then } S \text{ else } S) \mid (\text{while } B \text{ do } S)$$

Asts 和 Bexp 是 FLOW 的两个参数，这里未作详细规定。但是作为原子语句和布尔表达式，自然要求从语法上就能和 FLOW 中的其他部分相区别，这一要求在以后的讨论中还会用到。一旦给定这两个参数，就可得到一个特定的语言。

例如，令

X——变元集，以 x 表示其中元素；

N——非负整数集，以 n 表示其中元素；

Aexp——算术表达式集，以 a 表示其中元素，定义如下：

$$a ::= n \mid x \mid +(a,a) \mid -(a,a) \mid *(a,a)$$

这里我们采用前缀式运算符，如 $-(a_1,a_2)$ 表示 a_1 减去 a_2。

这样，Asts 可定义为：

$$A ::= a \Rightarrow x$$

即将变元 x 赋以表达式 a。Bexp 定义为：

$$B ::= tt \mid ff \mid =(a,a) \mid <(a,a) \mid \wedge(B,B) \mid \neg B$$

这里我们也使用前缀式运算符，如 $<(a_1,a_2)$ 表示表达式 a_1 的值小于表达式 a_2 的值。

这一语言记为 AFLOW。

§1.3　栈-状态-控制机器

本节中我们定义一个抽象机器来解释执行 FLOW 的程序。为了解释执行 FLOW 程序，需要三个工作区：

（1）存储程序的控制区，以 c 表示；

（2）记载程序变元当前值的机制，称为状态，记作 s，全体状态的集合记为 State；

（3）记载中间结果的栈区，记作 st。

三元组（st,s,c）的一种取值构成解释执行 FLOW 程序的抽象机器的一个状态，称作为大状态（Grand State），为和 s 区分，将大状态记作 gs。抽象机的动作是由大状态间的转移规则（Transition Rule）规定的，转移规则形为

$$(st,s,c) \Rightarrow (st',s',c')$$

规定抽象机可从大状态 (st,s,c) 转移至大状态 (st′,s′,c′)。转移规则也可以是一个模式，它的每个实例规定了抽象机大状态的一个可能的转移。

下面我们就用转移规则模式来定义解释执行 FLOW 程序的抽象机的行为。

A. 分析

（1）$(st, s, (\text{if B then } S_1 \text{ else } S_2)c) \Rightarrow (S_2 : S_1 : st, s, B \text{ if } c)$

这条规则是说，当抽象机执行到条件语句时，先把分语句存入栈区，其中“:”表示分割符；然后准备求出布尔表达式 B 的值，即将 B 保存在控制区的首部；在求出 B 的值后，再执行该条件语句，即把条件语句标志保存在控制区的 B 的后面。而栈区、状态区和控制区的其他部分不变。

（2）$(st, s, (\text{while B do S})c) \Rightarrow (\text{skip: S;(while B do S): st, s, B while c})$

这条转移规则和分析条件语句的规则很相似，只是在栈区中已形成了当 B 取真值时转往执行语句 S;(while B do S)，即先执行 S，再执行此循环语句；以及当 B 取假值时转往执行语句 skip，即跳过此语句，准备顺序执行下条语句。

B. 求值

由于 FLOW 语言中布尔表达式是一种参数，没有详尽的定义，因此在建

立布尔表达式求值的转移规则时，必须预先假定布尔表达式求值的语义函数 $\mathscr{B}$。$\mathscr{B}$ 规定了任一布尔表达式的语义，即 $\mathscr{B}$ 是布尔表达式至其语义的一个函数。而布尔表达式的语义又可看作是程序变元当前值（即状态 s 的值）至布尔值的一个函数。令 T=(tt, ff)，其中 tt 表示真值，ff 表示假值。则

$$\mathscr{B} \in (\text{Bexp} \to (\text{State} \to \text{T}))$$

亦写作

$$\mathscr{B}: \text{Bexp} \to (\text{State} \to \text{T})$$

这里（L→R）表示由集合 L 至集合 R 的全体函数组成的函数空间。在语义学中，通常采用花体字表示语言成分的语义函数，而将语言成分用双重括号括起，故给定一个布尔表达式 B，有

$$\mathscr{B}[\![\text{B}]\!]: \text{State} \to \text{T}$$

对给定的状态 s，又有

$$\mathscr{B}[\![\text{B}]\!](\text{s}):\text{T}$$

即对给定的布尔表达式 B 和状态 s，由 $\mathscr{B}$ 可得到 B 在状态 s 下所取的布尔值。

以上这些记号法在本书中将不断使用，不再一一说明。使用语义函数 $\mathscr{B}$ 及上述记号布尔表达式求值的转移规则可定义如下：

$$(\text{st},\text{s},\text{Bc}) \Rightarrow (\mathscr{B}[\![\text{B}]\!](\text{s}):\text{st},\text{s},\text{c})$$

即抽象机按照 $\mathscr{B}$ 对给定 B 及 s 求出布尔值 $\mathscr{B}[\![\text{B}]\!](\text{s})$，并将其存入栈区。

由于 FLOW 语言中没有算术表达式，故算术表达式的求值规则不必在此讨论。

C. 执行

下列各规则中 c=; c′或者 c = c′ = ∅（∅表示空序列，在不致引起混淆时，本书中亦用来表示空集合）。

（1） $(\text{st},\text{s},\text{skip c}) \Rightarrow (\text{st},\text{s},\text{c}')$

即执行 skip 不导致 st 和 s 的改变，但导致抽象机转向执行其后继部分 c′。当 c 中内容非空时，c 应以顺序算子“;”为起始跟以后继部分 c′，故 c=; c′；当 c 中内容为空时，其后继部分亦为空，故 c = c′=∅。

（2） $(\text{st},\text{s},\text{Ac}) \Rightarrow (\text{st},\mathscr{A}[\![\text{A}]\!](\text{s}),\text{c}')$

这里 $\mathscr{A}$ 代表原子语句的语义函数，如 $\mathscr{B}$ 一样，此处作为预先假定的。原子语句设想为赋值语句，它按照程序变元的当前值求出表达式值，再改变被赋

程序变元的值。故假定其语义为状态空间至状态空间的一个函数，即

$$\mathscr{A}: \text{Asts} \to (\text{State} \to \text{State})$$

（3） $(\text{tt}:S_2:S_1:\text{st},\text{s},\text{if c}) \Rightarrow (\text{st},\text{s},S_1\text{c})$

（4） $(\text{ff}:S_2:S_1:\text{st},\text{s},\text{if c}) \Rightarrow (\text{st},\text{s},S_2\text{c})$

这两条规则是说，在求出布尔表达式值后条件语句的执行导致抽象机选择相应的分语句，并转向执行该分语句。对于循环语句亦有两条类似的规则。

（5） $(\text{tt}:S_2:S_1:\text{st},\text{s},\text{while c}) \Rightarrow (\text{st},\text{s},S_1\text{c})$

（6） $(\text{ff}:S_2:S_1:\text{st},\text{s},\text{while c}) \Rightarrow (\text{st},\text{s},S_2\text{c})$

其中，S_2 为 skip，S_1 为 S;(while B do S)。

至此，我们定义了解释执行 FLOW 程序的一个抽象机，也就给出了 FLOW 语言的一种操作语义。下面以 AFLOW 中的一个程序为例，在假定 $\mathscr{A}$，$\mathscr{B}$ 和通常理解一致的前提下，看其操作语义是否与通常理解亦一致。

例　计算 x!的程序 FAC。

$$1 \Rightarrow y;\quad x \Rightarrow z;\quad (\text{while} \neg = (z,0)\ \text{do} * (y,z) \Rightarrow y;\quad -(z,1) \Rightarrow z)$$

FAC 中只有程序变元 x，y 和 z，设 State 为非负整数三元组的集合。在 x，y 和 z 的初始值分别为 2，0，0 时解释执行 FAC 程序形成的大状态转移序列如下，其中转移号“⇒”上标以此转移所依据的规则号，而用到的 $\mathscr{A}$、$\mathscr{B}$ 函数假设为：

$$\mathscr{B}[\![\neg = (z,0)]\!](x,y,z) = \begin{cases} \text{tt}, & z \neq 0 \\ \text{ff}, & z = 0 \end{cases}$$

$$\mathscr{A}[\![1 \Rightarrow y]\!](x,y,z) = (x,1,z)$$

$$\mathscr{A}[\![x \Rightarrow z]\!](x,y,z) = (x,y,x)$$

$$\mathscr{A}[\![*(y,z) \Rightarrow y]\!](x,y,z) = (x,y*z,z)$$

$$\mathscr{A}[\![-(z,1) \Rightarrow z]\!](x,y,z) = (x,y,z \dot{-} 1)$$

这里“$\dot{-}$”为非负减，即

$$n \dot{-} m = \begin{cases} n-m, & n \geqslant m \\ 0, & n < m \end{cases}$$

$(\varnothing, s_0, \text{FAC})$　　$s_0 = (2,0,0)$

$\overset{C,(2)}{\Rightarrow} (\varnothing, \mathscr{A}[\![1 \Rightarrow y]\!](s_0), S_1)$　　设 $\text{FAC} = 1 \Rightarrow y; S_1$

C,(2)
$\Rightarrow (\varnothing, \mathscr{A}[\![x \Rightarrow z]\!](s_1), S_2)$　　$s_1 = (2,1,0)$

$\mathscr{A}[\![1 \Rightarrow y]\!](s_0) = s_1$

设$S_1 = x \Rightarrow z; S_2$

A,(2)
$\Rightarrow (\text{skip} : S_3; S_2, s_2, \neg = (z,0)\,\text{while})$　　$s_2 = (2,1,2)$

$\mathscr{A}[\![x \Rightarrow z]\!](s_1) = s_2$

设$S_2 = (\text{while} \neg = (z,0)\,\text{do}\, S_3)$

B,(1)
$\Rightarrow (\mathscr{B}[\![\neg = (z,0)]\!](s_2) : \text{skip} : S_3; S_2, s_2, \text{while})$

C,(5)
$\Rightarrow (\varnothing, s_2, S_3; S_2)$　　$\mathscr{B}[\![\neg = (z,0)]\!](s_2) = \text{tt}$

C,(2)
$\Rightarrow (\varnothing, \mathscr{A}[\![*(y,z) \Rightarrow z]\!](s_2), S_4; S_2)$

设 $S_3 = *(y,z) \Rightarrow z; S_4$

C,(2)
$\Rightarrow (\varnothing, \mathscr{A}[\![-(z,1) \Rightarrow z]\!](s_3), S_2)$　　$s_3 = (2,2,2)$

$\mathscr{A}[\![*(y,z) \Rightarrow z]\!](s_2) = s_3$

A,(2)
$\Rightarrow (\text{skip} : S_3; S_2, s_4, \neg = (z,0)\,\text{while})$　　$s_4 = (2,2,1)$

$\mathscr{A}[\![-(z,1) \Rightarrow z]\!](s_3) = s_4$

……　　省略若干次转移

A,(2)
$\Rightarrow (\text{skip} : S_3; S_2, s_5, \neg = (z,0)\,\text{while})$　　$s_5 = (2,2,0)$

B,(1)
$\Rightarrow (\mathscr{B}[\![\neg = (z,0)]\!](s_5) : \text{skip} : S_3; S_2, s_5, \text{while})$

C,(6)
$\Rightarrow (\varnothing, s_5, \text{skip})$　　$\mathscr{B}[\![\neg = (z,0)]\!](s_5) = \text{ff}$

C,(1)
$\Rightarrow (\varnothing, s_5, \varnothing)$

上述转移序列表明，只要$\mathscr{A}$，$\mathscr{B}$符合通常的理解，在 x 的初值为 2 时执行 FAC，程序可以终止，终止时变元 y 处存有所求的阶乘值 2。