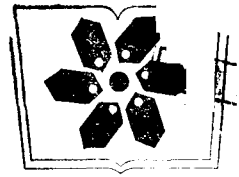
陆汝钤 编著

计算机语言的形式语义

科学出版社



中国科学院科学出版基金资助项目

计算机语言的形式语义

陆汝钤 编著

科学出版社

1992

（京）新登字 092 号

内 容 简 介

计算机语言的形式语义是目前计算机科学理论研究的两大方向之一，其研究成果对程序设计语言、编译技术、应用软件、分布式系统等分支领域有重大的实际意义．本书概述了形式语义学中的操作语义、指称语义、公理语义和代数语义四大流派的主要内容，并辟一章集中讨论了并发和分布式语义．本书内容自成体系，在开篇第一章即给出了阅读本书所必需的数学知识．全书内容丰富，结构严谨，集形式语义领域有关分支之大成，系统地反映了这个领域各方面的研究成果，特别是它的近代发展潮流和趋势，并对不同流派的理论和方法给予了分析和评价．

本书可作为计算机科学专业研究生、本科生有关课程的教材或教学参考书，也可供有关专业科技人员进修或作为工具书．

计算机语言的形式语义
陆汝钤 编著
责任编辑 刘晓融
科学出版社出版
北京东黄城根北街 16 号
邮政编码：100707
中国科学院印刷厂印刷
新华书店北京发行所发行 各地新华书店经售
*
1992 年 12 月第 一 版 开本：850×1168 1/32
1992 年 12 月第一次印刷 印张：28 1/8
印数：1—2 000 字数：744 000
ISBN 7-03-003022-2/TP · 223
定价：21.00 元

谨以本书纪念

吴允曾教授

前　言

语义学（semantics）一词出自希腊文 sēmantikos，它由动词 sēmainō 转化而来，表示语言的意义（相当于英文的 mean 或 signify）。一般认为语言学的研究可分为三部分：语法学（syntax），研究语言的形态结构；语义学，研究语言和它所指的对象之间的关系；语用学（pragmatics），研究语言和它的使用者之间的关系（当然也要涉及语言所指的对象）。三者合在一起，组成了指号学（semiotics）的最重要部分。在计算机语言的范围内，语法学的研究已经相当成熟，语用学是一个基本上还没有被人们所认识的陌生世界，只有语义学是一个正在蓬勃发展中的领域。在这个领域中既有已经取得的丰硕成果，又有尚待解决的重大问题，它是计算机科学中勇敢的探索者的乐园。

50 年代是计算机语言兴起的年代。在这一阶段的早期，计算机语言的设计往往主要强调其“方便”的一面，而比较忽略其“严格”的一面，因而对语言的语义，甚至是语法，未下严格的定义，从而语言设计者、语言实现者和语言使用者对同一语言的语义缺乏共同的理解，造成了一定程度的混乱。Chomsky 关于语言分层的理论，以及 Backus，Naur 关于上下文无关文法表示形式的研究成果推动了语法形式化的研究。其结果是，在 ALGOL60 的文本设计中第一次使用了 Backus-Naur 标准型表示语法，并且第一次在语言文本中明确地把语法和语义区分开来。后来，在 50 年代和 60 年代间，面向语法的编译自动化理论研究得到了很大发展，使语法形式化研究的成果达到实用化的地步。

语法形式化问题基本解决以后，人们逐渐把注意力集中到语义形式化方面。60 年代可以说是计算机语言形式语义学正式诞生的 10 年，形式语义学的四大流派皆渊源于这一时期。其中，

1964 年被认为是操作语义学和指称语义学的诞生年代，Landin 关于操作语义的奠基性文章“表达式的机械化处理”和 Strachey 关于指称语义的奠基性文章“关于形式语义学”都问世于这一年。1967 年则被认为是公理语义学诞生的年代，Hoare 关于公理语义的奠基性文章“计算机程序设计的一个公理学基础”就发表于这一年。最年轻的是代数语义学，它是在抽象数据类型理论的基础上发展起来的。虽然后者的思想源自 1967 年问世的 SIMULA 67，但正式把它提到抽象数据类型高度的是 Liskov 和 Zilles 在 1974 年的工作，而把它进一步上升为代数语义学则是 70 年代中期以后的事了。当时的主要代表人物有 Goguen 等人的 ADJ 小组及 Guttag 等人。

操作语义的基本思想是用抽象的方法描述语言中每一成分的执行效果，以免所描述的语义依赖于该语言实现时所用的具体计算机。通常的做法是设计一个抽象机，定义一组抽象状态，把语言的语法表示成抽象的形式，然后指明抽象机每加工一个语言成分时将对状态作何种改变。这种语义方法与语言实现的关系比较紧密，但是难以用数学方法处理，而且对语义描述者个人使用的实现方法依赖很大。

指称语义的基本思想是使语言的每一成分对应于一个数学对象，该对象称为该语言成分的指象，它不像操作语义那样涉及语言成分的执行过程，而是只考虑各成分执行的最终效果，并认为此最终效果应不依赖于其执行过程。由于语言成分可以是很复杂的，因此它的指象也可以很复杂，甚至达到令人怀疑这样的指象在数学上是否存在的程度。Scott 创建的论域理论解决了这个问题，从而为指称语义学奠定了坚实的理论基础。后来 Smyth 和 Plotkin 等人又建立了幂域理论，为不确定、并发和分布式程序的指称语义奠定了基础。

公理语义是在程序正确性验证的基础上发展起来的，它不像其它语义学方法那样，对程序语义作宏观的全局性描述，而只是给出一种方法，使人们能在给定的前提下，验证某种特定的性质是否

成立．例如，

$$\{P[e/x]\}x:=e\{P\}$$

表示：若在执行赋值语句 $x:=e$ 前条件 $P[e/x]$ 成立，则在执行后条件 P 成立．因此，公理方法的基础是一个逻辑系统，包括一组公理及其推理规则，它区别于经典逻辑的主要之点是把程序执行的效果也考虑进逻辑系统中．公理语义学的核心课题是研究这类逻辑系统的健康性和完备性．近年来，一些所谓非 Hoare 型的公理语义方法陆续出现，如模态逻辑、时序逻辑、动态逻辑、无穷逻辑、构造性逻辑等，大大丰富了这一领域的研究．

代数语义学的基本思想是把描述语义的逻辑体系和满足这个逻辑体系的模型区分开来．任何程序的操作语义或指称语义描述只给出该程序的一个语义模型，而公理语义描述则只给出逻辑系统，不深入探讨其可能的模型（近年来这种情况已有所改变，特别是对非 Hoare 型逻辑，例如参见本书 § 4.14）．代数语义方法的特点就是用代数方法来处理满足一个逻辑系统的各种模型，把模型的集合看成是一个代数结构．因此，在代数语义中，除了要研究类似于健康性和完备性等公理体系的概念外，还要研究模型之间的关系．简而言之，它要处理的是一整个模型族．

形式语义学文献中使用的技术和方法并不一定都能归入上面所说的四大流派，但这四大流派确实代表了形式语义学中最主要的四个研究方向，它们在本书中各自所占的篇幅大体上反映了该领域当前研究的活跃程度．相对地说，在公理语义和代数语义中，尚待研究和解决的问题比起操作语义和指称语义来要多一些．

当前研究兴趣的集中点之一是各种非 Hoare 型逻辑体系，在公理语义和代数语义中都有这种情况．至于哪一种逻辑最有前途，目前尚无定论．有人看好时序逻辑，有人欣赏无穷逻辑．近年来类型理论倍受青睐，似乎又预示着研究构造性逻辑的热潮即将兴起．总之，这里还是各种逻辑群雄逐鹿（也许更合适的词是百花齐放）的局面．

有分即有合．也有人研究如何构造一个超级逻辑体系，把各

种具体的逻辑系统都容纳进去，例如 Edinburgh 学派的机关(institution)理论．目前，这类研究主要是纯理论性的．

当前研究兴趣的另一个集中点是不确定、并发和分布式程序的形式语义．我们为后两者单设了第六章．一般认为，顺序式语言和顺序式程序的形式语义研究比较成熟，而上面提到的三个方面的形式语义研究则还有很长的一段路要走．例如，操作语义的偏序推导和变迁系统，指称语义的幂域理论和不动点理论，都是研究分布式语义的有力手段，但都还不完善．分布式程序的真并发语义，以及它的一些重要性质，如公平性和发散性，是近年来有关文献中的热门话题．如何用代数语义方法和更多的代数手段来处理分布式语义，也是许多研究者特别关心的课题．

操作语义和指称语义并没有失去其生命力．有两个情况使操作语义的研究保持着一定的势头．其一是 Plotkin 在 1981 年发明的结构化操作语义，给人以操作语义重新焕发青春之感；以结构化操作语义为基础的变迁系统已普遍用于各类语言，包括分布式语言的语义描述中．其二是在论述其它语义的正确性时，操作语义至今仍常被作为"参照系"而使用．至于说到指称语义，则在不少计算机科学家的心目中，它仍然是一种"标准"的语义描述方法．

四种语义方法融合的倾向值得注意．它的一种表现是各方法之间的边界逐渐模糊，互相采用对方的长处，有些语义方法已很难绝对地划入某一阵营．另一种表现是有人试图建立一种超级方法体系，把四种现有方法都纳入其中作为子方法，在程序的不同开发阶段根据各方法的特点量才录用．

由于本书涉及的内容比较多，为了方便读者，首先在第一章给出了必要的数学基础，在第二至六章的每一章前给出了一节概述．概述的目的是，没有时间细读整章内容的读者可以只读概述部分；细读整章内容的读者可以用它把各节内容串成一个整体．概述中还包括了一些不能独立成节的内容．

作者衷心感谢唐稚松教授、林惠民副教授和冯玉琳教授在本书审稿方面给予的宝贵支持．林惠民审阅了第一和第五章，冯玉

琳审阅了第二、三、四、六各章. 他们在百忙中花费了大量时间阅读原稿，指出了原稿中存在的许多不当之处，使本书内容得以进一步完善. 作者还十分感谢本书的第一位读者韦梓楚同志，他细读了本书初稿，发现了不少问题，提出了许多宝贵的建议.

作者十分怀念原北京大学教授吴允曾先生. 本书是在他的建议和鼓励下开始写作的. 吴先生离开我们已有多年，但他的音容笑貌和谆谆教诲却仍浮现在眼前. 作者也十分感谢原《计算机科学丛书》编委会的全体成员，特别是王湘浩教授和徐家福教授，感谢他们对本书出版所给予的热情支持. 我还特别感谢科学出版社第六编辑室编辑同志们的耐心和宽宏. 本书书稿的完成时间大大超过了原合同的规定，于今仍能顺利出版，这样的支持是难能可贵的. 当然，时间上的延长也使我有机会接触更多和更新的研究成果，然而，其结果之一是本书的篇幅也大大膨胀了.

本书的文献目录是冯方方、曹存根和孙荣平协助编辑和录入的. 张松懋帮助整理了全部书稿. 我在此向他们一并致以衷心的感谢.

形式语义学是一门发展中的学科，涉及面广，有较强的理论性. 尽管作者已几易其稿，但书中的错误和不妥之处仍在所难免，欢迎读者批评指正.

目　　录

第一章 数学基础

§1.1 λ 演 算

和组合逻辑一样，λ 演算的早期根源也在 20 世纪 20 年代，但常被引用的权威文献则是 Church 在 1941 年发表的 "The Calculi of Lambda Conversion"。许多数学工具都是以集合为基础的，而 λ 演算与众不同，它是以函数演算为基础的。它允许任意高阶的函数运算，也就是说，一个函数的输入和输出都可以是函数。这个特点使它特别适于用作语义描述的表示方法，尤其是在指称语义描述中更需要用到它。但是请注意，本节给出的实际上是 λ 演算的操作语义，因此，有的作者在使用 λ 演算于语义描述之前，首先要定义 λ 演算本身的指称语义，本书把这一点省略了。

人们通常把函数写为 $y = f(x)$ 的形式，这里涉及到三个因素：函数符号 f，变元 x 和值 y。通常，这三者可以属于不同的值域。例如 $y = \mathrm{ent}(x)$ 是定义在实轴上的一个全函数，表示取变元的整数部分。这是程序库中常见的初等函数，其变元的值域(即定义域)自然是实数集，而函数的值域却是整数集。这样，三个因素属于三种不同的值域，不得不分别加以考虑。然而，各式各样的值域是无穷之多的，我们不能统统拿来逐个加以研究，尤其是函数的变元和值都可以又是函数，如

$$g(t) = \int_{-\infty}^{t} f(x)dx = F(f,t) \qquad (1.1.1)$$

就出现了高阶函数，即泛函。在计算机语义的数学表示中，泛函的研究是主要难点之一。所以，我们不仅需要有一种能统一处理各类值域的函数表示方法，而且尤其需要能有效地处理泛函的函数表示方法。人们发现，在逻辑学的库藏中有着现成的处理这类

问题的方法，至少 λ 演算和组合逻辑中使用的表示法是可以考虑的. Church 的 λ 演算因直观、好用而被人们选中. Church 用来表示函数的方法叫 λ 表示法(注意,由于函数和其它类型的值在 λ 演算中是一视同仁的,我们不再区别函数和泛函这两个概念)，它的基本形式是

$$\lambda\langle\text{变元}\rangle.\langle\text{表达式}\rangle \tag{1.1.2}$$

用这种方法表示的函数叫 λ 表达式. 如

$$\lambda x.x^2+1 \tag{1.1.3}$$

就是一个 λ 表达式，其中 x 是变元. 函数值的表示方法是: 用一对括号把代表函数的 λ 表达式括起来，而把变元值置于 λ 表达式之后. 如,取变元值 x 为 1 ,则相应的函数值是

$$(\lambda x.x^2+1)1 \tag{1.1.4}$$

其计算方法是把变元值代入表达式中的变元，去掉前面的 λ〈变元〉,并按通常方式计算此表达式,在上例中

$$(\lambda x.x^2+1)1=1^2+1=2 \tag{1.1.5}$$

变元可以有多个,此时 λ 表达式的形式变元为

$$\{\lambda\langle\text{变元}\rangle.\}^{*}\langle\text{表达式}\rangle \tag{1.1.6}$$

例如,

$$\lambda x.\ \lambda y.\ x^3+y^2 \tag{1.1.7}$$

就是两个变元的 λ 表达式，把它作用于变元值 $x=1, y=2$ 时，得

$$((\lambda x.\lambda y.x^3+y^2)1)2=(\lambda y.1+y^2)2=5 \tag{1.1.8}$$

从这里可以看出 λ 表示法的一个优点，即函数的值可以是一个函数. 由于 $\lambda x.\lambda y.x^3+y^2$ 本是一个双变元的函数，当变元 x 被具体的值 1 代替时，原来的双变元函数就成了单变元函数 $\lambda y.1+y^2$. 因此，用 λ 表示法可以体现函数的层次关系，即函数的函数.

变元的次序会影响函数应用的值,如*

---

* 比较式(1.1.8).

$$((\lambda y.\lambda x.x^3 + y^2)1)2 = (\lambda x.x^3 + 1)2 = 9 \qquad (1.1.9)$$

在注意到这一点的前提下，可以把多变元的 λ 表达式简写为

$$\lambda\{\langle\text{变元}\rangle\}^{+}.\langle\text{表达式}\rangle \qquad (1.1.10)$$

我们的例子也就成为

$$\lambda xy.x^3 + y^2 \qquad (1.1.11)$$

注意式 (1.1.11) 只是一种简写，含意未变，它的内层依旧是 $\lambda y.x^3 + y^2$. 在这个子 λ 表达式中，变量 x 未在 λ 部分出现，它有什么意义呢？它叫做自由变量，而在 λ 部分出现的变量，如 y，则叫做约束变量. 我们可以按通常程序设计语言中的意义把约束变量理解为局部变量，但如果随之而要把自由变量理解成全局变量，则请谨慎为好！因为我们现在看到的只是一个 λ 表达式

$$\lambda y.x^3 + y^2,$$

除此以外我们什么也看不见！不过，大体上可以按数学函数中参数的意义来理解自由变量.

我们再看 $\lambda x.x^3 + y^2$. 在这里，x 和 y 的作用正好和它们在 $\lambda y.x^3 + y^2$ 中的作用倒了一个个儿. x 成了约束变量，而 y 则是自由变量. 如果把前面讲过的也合在一起，则可以看到，同是一个表达式 $x^3 + y^2$，其意义由于加了不同的 λ 首部 ($\lambda xy.$, $\lambda yx.$, $\lambda x.$, $\lambda y.$) 而不一样了. 这种在表达式前加上 λ 首部的操作，叫做抽象，是 λ 表示法中一个基本的概念.

实际上，我们刚才还涉及到了另外一些基本概念. 比如，“应用”就是 λ 演算中(乃至一切函数型系统中)最重要的概念之一，它表示把一个函数应用于一组具体的变元值(如像上例中的 $x = 1$, $y = 2$)，当然，变元值本身也可以是函数. 在下面将要考察的形式系统中，“应用”是唯一的一种操作，一切其它操作都以函数应用的形式出现.

此外，“置换”是又一个重要的基本概念，它不但是实现“应用”的手段，而且是整个 λ 演算的基石. 两个 λ 表达式是否相等就是靠“置换”来证明的，也正是依靠它，Church 证明了 λ 演算具有部分递归函数的计算能力，也就是相当于图灵机.

定义 1.1.1. 形式的 λ 演算系统. 一个形式的 λ 演算系统由下列几部分组成:

(1) 由变量组成的无穷集合 V;

(2) 由常量组成的(一般为有穷的)集合 C. 集合 V 和集合 C 中的元素统称原子;

(3) 由四个特殊符号 λ, ., (,) 组合的集合 S. 它们在形式系统中通常称为组合符. 集合 $\{V \cup C \cup S\}^+$ 中的任一元素皆能唯一地分解为 V, C 和 S 三个集合中元素的并列 (注意, 这隐含了 V, C 和 S 三个集合两两相交均为空集);

(4) 以 V, C 和 S 三集合中元素为语法元的一组语法公式;

(5) 一组转换规则. ∎

定义 1.1.2. 我们一般以 x, y, z, $\cdots$ 等表示 V 中元素, 以 a, b, c, $\cdots$ 等表示 C 中元素. 我们所讨论的形式系统的语法公式是

〈λ 表达式〉:: = λ〈变量〉.〈λ 表达式〉|〈表达式〉

〈表达式〉:: = 〈应用〉|〈原子〉

〈应用〉:: = 〈表达式〉〈原子〉

〈原子〉:: = 〈变量〉|〈常量〉|(〈λ 表达式〉) ∎

例 1.1.1. x, a, (y), xy, $(abc\ xyz)$, $x(ab)(yz)c$; $\lambda x.y$, $\lambda x.x$, $\lambda x.\lambda x.xx$, $yx(\lambda x.z)$, $(\lambda x.y)(\lambda y.z)(\lambda z.x)$ 等都是合乎定义的 λ 表达式. ∎

这个语法公式表明, λ 演算中的基本运算是"应用", 每个表达式应用于紧靠着它后边的原子, 应用方式取左结合, 即

$$abc = (a(b))c$$

这里没有结合律, 否则就要出二义性了. 并且, 本系统的"应用"只适用于单变元.

需要解释一下约束变量的作用域. 我们设想把 λ 表达式写成 $\lambda x.(e)$ 的形式, 则约束变量 x 的作用域是从 e 的左括号起, 到与它配对的右括号止.

定义 1.1.3. 令 e_1 和 e_2 皆为表达式, 若至少有下列条件之一成立, 则称 e_1 在 e_2 中出现:

(1) $e_1 \equiv e_2$;

(2) $e_2 \equiv e_3 e_4$, e_3 为表达式，e_4 为原子，e_1 在 e_3 或 e_4 中出现；

(3) $e_2 \equiv (e_3)$，e_3 为 λ 表达式，e_1 在 e_3 中出现；

(4) $e_2 \equiv \lambda y.e_3$，y 是任一变量，e_3 是 λ 表达式，e_1 在 e_3 中出现。 ∎

例 1.1.2. 变量 x 在 x, xy, $\lambda y.x$, $\lambda x.x$, $(\lambda x.x)(\lambda y.x)$ 等 λ 表达式中出现，但不在 $\lambda y.y$, $\lambda x.y$ 中出现。 ∎

对于前面已经提到过的自由变量和约束变量，我们也需要更确切的定义。

定义 1.1.4. 令 x 为变量，e_2 为表达式，称 x 在 e_2 中自由出现，若

(1) x 在 e_2 中出现；

(2) 当 x 是按定义 1.1.3 (4) 的规定在 e_2 中出现时，x 应在表达式 e_3 中自由出现，且 $x \neq y$。 ∎

定义 1.1.5. 若变量 x 在表达式 e 中出现，但这个出现不是自由出现，则称 x 在 e 中约束出现。 ∎

请读者注意，我们强调了“这个”两字。因为，同一个变量 x 可在同一个表达式 e 中同时有自由出现和约束出现。例如，x 在 $\lambda y.x(zt)$, $\lambda y.\lambda z.xt$, xzt 中自由出现，在 $\lambda x.x, \lambda x.t(\lambda y.x)$ 中约束出现，在 $(\lambda x.x)(\lambda y.x)$ 中既有自由出现，又有约束出现。

定义 1.1.6. 若变量 x 在表达式 Y 中仅有自由出现或仅有约束出现，则 x 分别称为该表达式的自由变量或约束变量。 ∎

前面已经提到，置换在 λ 演算中有着特别重要的意义，它的确切定义如定义 1.1.7.

定义 1.1.7. 令 x 是变量，M 和 N 是表达式，则置换 $[N/x, M]$ 表示用 N 代替 x 在 M 中的自由出现。置换所得的结果是

(1) M，若 x 不在 M 中自由出现；

(2) N，若 $M = x$；

(3) $[N/x, L][N/x, R]$，若 $M = LR$；

(4) $([N/x, P])$，若 $M = (P)$；