arXiv:2304.00358v1 [cs.LO] 1 Apr 2023

逻辑即代数

Steven Obua[0000−0002−4362−752X]

Practal
obua@practal.com
https://practal.com

摘要。 只要采用恰当的代数和恰当的逻辑，逻辑实际上就是代数。恰当的代数是抽象代数，恰当的逻辑是抽象逻辑。

关键词： 抽象逻辑 · 抽象代数 · 抽象代数 · 代数逻辑 · 泛代数 · 等式逻辑 · 一阶逻辑 · 高阶逻辑 · λ演算 · 简单类型论 · Practal

1 恰当的逻辑

什么是正确的逻辑？显然，这个问题的答案取决于你的具体情况。但如果你正在寻找一种作为数学和证明助手基础的通用逻辑，那么我有一个主张要告诉你：

正确的逻辑是抽象逻辑。

乍一听，这个说法似乎荒谬，尤其是当我接着告诉你，抽象逻辑（AL）比证明助手中使用的任何其他通用逻辑都更简单，例如一阶逻辑以及简单类型论或依赖类型论。如果它确实存在，那么这种“正确”的逻辑肯定早就被发现了，不是吗？

我不知道为什么直到现在才发现 AL。也许它其实已经被发现过了，只是我不知道。鉴于有关逻辑的文献浩如烟海，这种情况是可能的。即便如此，这也不会改变我撰写这篇论文的主要理由：每个人都应该了解抽象逻辑，并使用它。

要猜测为什么 AL 之前一直未被发现，或许可以回想一下它是如何被发现的：我当时正在为我的证明助手项目 *Practal* [6]寻找合适的基础。简单类型论感觉过于受限，依赖类型论又过于任意，而一阶逻辑甚至无法恰当地表达绑定构造，除了 ∀ 和 ∃。事实证明，后一项观察至关重要：AL 与一阶逻辑相似，但它不只是讨论数学宇宙 $\mathcal{U}$中的元素，以及定义在 $\mathcal{U}$上的函数和关系；AL 还允许讨论作用于 $\mathcal{U}$的<style id='10'>算子。算子是 $\mathcal{U}$上的泛函，即一个以定义在 $\mathcal{U}$上的函数为输入，并以 $\mathcal{U}$中的元素为输出的函数。全称量化 ∀ 和存在量化 ∃ 于是只是众多算子中的显著例子。$\mathcal{U}$中的元素，以及定义在 $\mathcal{U}$上的函数和关系，都是算子的特殊情况。

当然，认为讨论泛函是有用的这一想法远非新颖。在数学中，整个泛函分析分支正是建立在这一洞见之上的。不过，泛函分析通常并不将整个数学宇宙 $\mathcal{U}$ 作为考察主题，而是聚焦于特殊的（尽管也相当一般的）数学结构，例如向量空间、希尔伯特空间和拓扑空间。与逻辑的关系更为密切的是，数学家发展出了范畴论，计算机科学家发展出了类型论。因此，以下便是我猜测 AL 尽管简单，却至今并不为人所熟知的原因：

AL 在范畴论和类型论的漩涡中迷失了方向。

本文其余部分将介绍 AL，并论证 AL 是合适的逻辑。首先，将抽象代数引入为抽象代数结构的一种推广。文中将说明，抽象代数足以直接描述常见的数学概念，例如自然数、集合、函数，以及简单类型和依赖类型，即使这些概念都处于同一个抽象代数中也不例外。随后，将介绍 AL 极其简洁的语法、语义和证明论。文中将证明 AL 是可靠的；在对所讨论的特定逻辑作出适度条件假设的情况下，AL 也是完备的。最后，将探讨抽象逻辑不一致意味着什么。结尾将简要提及相关工作与未来工作。

2 抽象代数

设 A 是一个非空集合[1]。每个映射 $A^n \to A$ 都称为 n 元运算**运算**，其作用于 A，或称为元数为 n 的运算元数。记号 A_n 是 $A^n \to A$ 的缩写。零元运算被解释为 A 的常元，并称为值。因此，A_0 与 A 被视为相同。

每个映射 $A_{m_1} \times \cdots \times A_{m_n} \to A$ 都称为一个 n 元 **算子**，或称为元数为 n 的算子。列表 $[m_1, \ldots, m_n]$ 称为其（算子）形状。由于A_0 和 A 被视为相同，每个元数为 n 的运算也是一个元数为 n的算子，其形状只由零组成。一个值的形状为 $[]$。不是值的运算，以及不是运算的算子，称为**真**。

一个**抽象代数** $\mathcal{A}$ 是一个非空集合 $\mathcal{C}$（$\mathcal{A}$），即 $\mathcal{A}$的**载体**，以及一个集合 $\mathcal{O}$（$\mathcal{A}$），该集合由定义在 $\mathcal{C}$（$\mathcal{A}$）上的算子组成，即 $\mathcal{A}$的**算子**。

记号（A、o_1、.....、o_k）描述了一个抽象代数 $\mathcal{A}$，其载体为$\mathcal{C}$（$\mathcal{A}$）$= A$，算子为 $\mathcal{O}$（$\mathcal{A}$）$= \{o_1, \ldots, o_k\}$。

例*1*。 任意（**抽象**）**代数** [5, p. 287] 都是抽象代数 $\mathcal{A}$，其中每个 $\mathcal{A}$ 的算子都是运算。例子包括自然数代数（$\mathbb{N}$、**0**、**1**、$+$、$\cdot$）以及二元布尔代数（{T、F}、T、F $\wedge$、$\vee$、$\neg$）。

接下来的三个小节将介绍更多示例：*Peano*、集合 和 *FunSets*。

[1] 这里的“集合 ”一词是按朴素意义使用的。

2.1 *Peano*

设 *Peano* 为抽象代数

$$(\mathbb{N} \uplus \{\mathsf{T}, \mathsf{F}, \bot\}, N, 0, S, \equiv, \Rightarrow, \neg, \forall, \forall_N).$$

其中，T、F 和 $\bot$ 是三个互不相同的值。算子 0 是一个值，表示其自身。*Peano* 的其他算子定义如下：

$$\begin{aligned}
N(x) &= \begin{cases} \mathsf{T} & \text{if } x \in \mathbb{N} \\ \mathsf{F} & \text{otherwise} \end{cases} \\
S(x) &= \begin{cases} x+1 & \text{if } x \in \mathbb{N} \\ \bot & \text{otherwise} \end{cases} \\
x \equiv y &= \begin{cases} \mathsf{T} & \text{if } x = y \\ \mathsf{F} & \text{otherwise} \end{cases} \\
x \Rightarrow y &= \begin{cases} \mathsf{F} & \text{if } x = \mathsf{T} \text{ and } y \neq \mathsf{T} \\ \mathsf{T} & \text{otherwise} \end{cases} \\
\neg x &= x \Rightarrow \mathsf{F} \\
\forall f &= \begin{cases} \mathsf{T} & \text{if } f(x) = \mathsf{T} \text{ for all } x \in \mathcal{C}(\mathit{Peano}) \\ \mathsf{F} & \text{otherwise} \end{cases} \\
\forall_N f &= \forall\,(x \mapsto N(x) \Rightarrow f(x))
\end{aligned}$$

注意以下表达式

$$\mathsf{N}(0),\ \forall_N(a \mapsto a \equiv a),\ \forall_N\,(a \mapsto \forall_N\,(b \mapsto (a \equiv b) \Rightarrow (b \equiv a))),\ \forall_N\,(a \mapsto \forall_N\,(b \mapsto \forall_N\,(c \mapsto (a \equiv b) \Rightarrow ((b \equiv c) \Rightarrow (a \equiv c))))),\ \forall(a \mapsto \forall_N\,(b \mapsto (a \equiv b) \Rightarrow \mathsf{N}(a))),\ \forall_N(a \mapsto \mathsf{N}(\mathsf{S}(a))),\ \forall_N\,(a \mapsto \forall_N(b \mapsto \mathsf{S}(a) \equiv \mathsf{S}(b) \Rightarrow a \equiv b)),\ \forall_N\,(a \mapsto \neg(\mathsf{S}(a) \equiv 0)),\ K(0) \Rightarrow ((\forall_N\,(x \mapsto K(x) \Rightarrow K(\mathsf{S}(x)))) \Rightarrow \forall_N K)$$

它们与原始皮亚诺公理相对应 [7, p. 113],都取值为 T。在最后一个表达式中，单参数运算 K 可以任意选择。

2.2 集合

让集合成为抽象代数

$$(\textsc{Sets}_{\mathsf{T},\mathsf{F}} \uplus \{\bot\}, \equiv, \text{if-then-else}, \forall, \emptyset, :, \bigcup, \mathscr{P}, \mathscr{R}, \iota, \epsilon, S, \mathbb{N}).$$

载体是包含**原子**T 和 F 的集合类，以及一个独立的异常值 $\perp$。原子就是本身不是集合、但可以成为集合元素的值。因此，值 $\perp$ 既不是集合，也不是原子。运算 $\equiv$ 定义如下：如果 A 和 B 相等，则 $(A \equiv B) = \mathsf{T}$；否则 $(A \equiv B) = \mathsf{F}$。两个集合相等，当且仅当它们包含相同的元素；T、F 和 $\perp$ 是彼此不同的值，它们不等于任何集合，且只与自身相等。$\forall$算子的定义也与*Peano*中的定义相同，但当然需要考虑不同的载体。条件运算"条件选择"的定义如下

$$(\text{if } C \text{ then } a \text{ else } b) = \begin{cases} a & \text{if } C = \mathsf{T}, \\ b & \text{otherwise.} \end{cases}$$

值 $\emptyset$ 是空集。: 运算定义为：如果 D 是集合且 $x \in D$，则 $(x : D) = \mathsf{T}$；否则 $(x : D) = \mathsf{F}$。$\bigcup$ 运算以集合 D 为参数，并返回所有集合 $A \in D$ 的并集。如果 D 不是集合，则 $(\bigcup D) = \emptyset$。类似地，如果 D 是集合，则 $\mathscr{P}\, D$ 是 D的所有子集组成的集合；如果 D 不是集合，则 $(\mathscr{P}\, D) = \emptyset$ 。替换算子 $\mathscr{R}$ 以集合 D 和一元运算 f为参数，将 D的每个元素 x替换为 $f(x)$，从而得到另一个集合：

$$(\mathscr{R}\, D\, f) = \{\, y \mid \text{there is } x \in D \text{ such that } f(x) = y \neq \perp \,\}$$

如果 D 不是集合，则这意味着 $(\mathscr{R}\, D\, f) = \emptyset$。定描述算子 ι选出一个唯一的值，使某个运算的结果为 T：

$$\iota\, f = \begin{cases} x & \text{if there is a unique } x \in \mathcal{C}(\mathit{Sets}) \text{ such that } f(x) = \mathsf{T} \\ \perp & \text{otherwise} \end{cases}$$

选择运算 ϵ 从一个非空集合中任意选取一个元素：

$$\epsilon\, D = \begin{cases} x & \text{where } x \text{ is } \textit{some} \text{ value such that } x \in D \\ \perp & \text{if } D \text{ is not a set, or if } D \text{ is an empty set} \end{cases}$$

对于选取 x 中的哪一个 D 元素，并没有进一步的要求，当然，$D = E$ 蕴含 $(\epsilon\, D) = (\epsilon\, E)$ 这一点除外。注意，结合替换与选择，通过 $D \mapsto \mathscr{R}\, D\, \epsilon$ 即可满足选择公理。

自然数集合 $\mathbb{N}$ 在构造无限集合时起着重要作用：

$$\mathbb{N} = \{\emptyset, S(\emptyset), S(S(\emptyset)), S(S(S(\emptyset))), \ldots\}\,.$$

这里S 是后继运算，定义为$S(n) = n \cup \{n\}$。

如何定义集合的算子，有一定的灵活空间。归根结底，这取决于你打算将抽象代数用于什么用途。对于这个版本的集合，设计原则是引入较少的特殊情况，而不是捕捉更多潜在的规范错误。反过来采用这一设计原则，例如就会得到如下定义：

$$(\text{if } C \text{ then } a \text{ else } b) = \begin{cases} a & \text{if } C = \mathsf{T}, \\ b & \text{if } C = \mathsf{F}, \\ \perp & \text{otherwise.} \end{cases}$$

2.3 *FunSets*

抽象代数集合 只定义了一组核心算子。还可以仅基于这组核心算子定义更多算子。我们不断这样做，直到得到*FunSets*：一种载体与集合相同的抽象代数，但还包含以下附加算子，这些算子依次根据先前定义的算子来定义：

$\mathsf{T} = \quad \forall (x \mapsto x \equiv x)\ \mathsf{F} = \quad \forall (x \mapsto x) \bot \quad = \quad \iota (x \mapsto \mathsf{F})$
$A \Rightarrow B \quad = \text{如果 } A \text{那么 } B \text{否则 } \mathsf{T} \neg A \quad = \quad A \Rightarrow \mathsf{F}$
$A \not\equiv B \quad = \quad \neg (A \equiv B) A \wedge B \quad = \text{如果 } A \text{那么 } B \text{否则 } A$
$A \vee B \quad = \text{如果 } A \text{那么 } A \text{否则 } B A \Leftrightarrow B \quad = (A \Rightarrow B) \wedge ($
$B \Rightarrow A) \{a, b\} \quad = \quad \mathscr{R} (\mathscr{P} (\mathscr{P} \emptyset))\ (x \mapsto \text{如果 } x \equiv \emptyset \text{那么 } a \text{否则 } b)$
$\{a\} \quad = \quad \{a, a\} D \cup E \quad = \quad \bigcup \{D, E\} D \subseteq E \quad = \quad \forall ($
$x \mapsto (x : D) \Rightarrow (x : E)) \mathbb{B} \quad = \quad \{\mathsf{T}, \mathsf{F}\} \mathscr{S}\ D\ p \quad = \quad \mathscr{R}\ D$
$(x \mapsto \text{如果 } p(x) \text{ 那么 } x \text{否则 } \bot) \bigcap M \quad = \quad \mathscr{S} (\bigcup M)\ (x \mapsto \forall (D \mapsto (D :$
$M) \Rightarrow (x : D))) D \cap E \quad = \quad \mathscr{S}\ D (x \mapsto x : E) \exists p \quad = \quad \neg \forall$
$(x \mapsto \neg p(x))\ (a, b) = \text{如果 } a \not\equiv \bot \wedge b \not\equiv \bot \text{那么 } \{a, \{a, b\}\} \text{否则 } \bot \text{第一个}$
$u \quad = \quad \iota (x \mapsto \exists (y \mapsto u \equiv (x, y))) \text{第二个 } u \quad = \quad \iota (y \mapsto \exists ($
$x \mapsto u \equiv (x, y))) D \times E \quad = \quad \bigcup \mathscr{R}\ D\ (d \mapsto \mathscr{R}\ E\ (e \mapsto (d, e))) \text{ 定义域}$
$R \quad = \quad \mathscr{R}\ R \text{第一个 陪域 } R \quad = \quad \mathscr{R}\ R \text{第二个 泛函 } R \quad = \quad \forall (x \mapsto \forall ($
$a \mapsto \forall (b \mapsto ((x, a) : R \wedge (x, b) : R) \Rightarrow a \equiv b))) D \rightharpoonup E \quad = \quad \mathscr{S} (\mathscr{P} ($
$D \times E)) \text{ 泛函 } D \to E \quad = \quad \mathscr{S} (D \rightharpoonup E)\ (R \mapsto \text{定义域 } R \equiv D)$
$R \cdot x \quad = \quad \iota (y \mapsto (x, y) : R) \lambda\ D\ f \quad = \quad \mathscr{R}\ D\ (x \mapsto (x, f(x)))$
$\sum D\ t \quad = \quad \mathscr{S} (D \times \bigcup \mathscr{R}\ D\ t)\ (u \mapsto (\text{第二个 } u) : t(\text{第一个 } u))$
$\prod D\ t \quad = \quad \mathscr{S} (D \to \bigcup \mathscr{R}\ D\ t)\ (f \mapsto \forall (x \mapsto (x : D) \Rightarrow (f(x) : t(x))))$

根据这些定义，对于各自表达式中出现的自由变量，无论如何赋值，下面的示例表达式都等于 T：

- $\mathsf{F} \not\equiv \bot$, but also $\mathsf{F} \Leftrightarrow \bot$,
- $\{a, \bot\} \equiv \{a\}$, $\{\bot, a\} \equiv \{a\}$, $\{\bot\} \equiv \emptyset$, $(a, \bot) \equiv \bot$, $(\bot, a) \equiv \bot$,
- $(\bigcup \mathbb{N}) \equiv \mathbb{N}$, and $(\bigcup \mathbb{B}) \equiv \emptyset$,
- $\sum D\ (x \mapsto E) \equiv D \times E$, $\prod D\ (x \mapsto E) \equiv D \to E$,
- $(x : D) \wedge (F : D \to E) \Rightarrow (F \cdot x) : E$,
- $(x : D) \Rightarrow (\lambda\ D\ f) \cdot x \equiv f(x)$.

2.4 数据抽象

关于集合论，一个常见的批评是：它让**数据抽象**无法实现，因为所有东西都是集合。对于集合 和*FunSets*来说，这个问题没那么严重，因为人们可以直接“忘掉”某个对象的定义，转而依据它的抽象属性进行操作。例如，人们可以忘记 $\mathbb{N}$ 中的元素是如何定义的，而只像在 *Peano* 中那样使用它们。这样一来，就无法知道某个自然数是不是集合，因为它也可能是原子！具体来说，等式 $\bigcup \mathbb{N} = \mathbb{N}$ 在集合 中成立，但此时并不能确定它也成立。不过，对于某些应用而言，这种不确定性可能还不够。可能需要确定 $\mathbb{N}$ 的元素不是集合，这样 $\bigcup \mathbb{N} = \emptyset$ 才成立，正如 $\bigcup \mathbb{B} = \emptyset$ 已经成立一样。

基于简单类型论的证明助手（如 HOL4、Hol Light 和 Isabelle/HOL）提供了一种类型定义机制，通过建立新类型与现有类型子集之间的双射关系来引入新类型。同样的原理也可以应用于 *FunSets*。首先，假定有一个有序对运算$x\,y \mapsto \langle x\,y\rangle$，使得 $\langle x\,y\rangle$ 能够保证是一个原子。Dunne 等人在 [8] 中完成了这种构造，从而在 ZF 集合论[2]之上建立了一个抽象屏障。然后，为了基于现有集合 S创建一种新的数据类型，会创建一个新的标签 τ，并保证它不同于此前创建的标签；集合 $T = \{\langle \tau, s\rangle \mid s \in S\}$ 则构成新的数据类型 T。这样，就可以引入标签 bool、nat、pfun、fun、dpair 和 dfun，从而不仅为布尔值 $\mathbb{B}$创建抽象数据类型，也为自然数 $\mathbb{N}$、函数类型 $D \rightharpoonup E$和 $D \to E$以及依赖类型 $\sum D\,t$和 $\prod D\,t$创建抽象数据类型。

3 JustUse a LambdaTM

到目前为止，表达式一直以标准数学方式由算子构成。现在的任务是定义**项**，使得根据抽象代数对某个项求值时，它对应于这样的表达式。作为示例，请考虑 λ for*FunSets*的定义：

$$\lambda\, D\, f \quad = \quad \mathscr{R}\, D\, (x \mapsto (x, f(x))).$$

定义表达式由一个符号 $\mathscr{R}$ 构成，该符号命名了一个接受两个参数的算子：一个值 D，以及对 $\mathcal{C}$（*FunSets*）进行的一元运算，由表达式 $x \mapsto (x, f(x))$表示。人们很容易想要使用只需使用*Lambda*TM来定义能够直接表示此类运算的项，例如通过 $\Lambda\, x.(x, f(x))$。

问题在于，抽象代数的载体对应于一个完整的数学宇宙。这个数学宇宙由**值**组成。引入 Lambda 是试图将运算视为数学宇宙中的普通元素，就像值一样。但由于康托尔定理，运算的数量远多于值的数量（假设至少存在两个值）。这意味着我们不能简单地将运算与值归为一类，并将它们称为 2.0 版的值，因为那会导致悖论。

[2] 另请参阅我在 MathOverflow 上提出的问题：https://mathoverflow.net/questions/366014/data-abstraction-in-set-the

因此，试图将运算纳入数学宇宙，就需要引入**类型**，以便在数学宇宙中将值与运算区分开来。

但类型并不会带来任何东西。类型是对你自由组合算子的方式施加的任意限制，并且会完全禁止许多有用的算子。在引入类型之后，试图重新获得部分自由，便导致了日益复杂的类型系统，在 Barendregt 的 λ 立方体中激烈发展 [9]。但正如我们自集合论诞生以来所知，也正如这里通过 *FunSets*再次具体展示的那样，集合、函数和类型（作为集合）可以 在单一的数学宇宙中共存。只不过并非所有运算都可以。但当然，整个 λ 立方体都可以容纳在这个数学宇宙中，而类型规则也只会变成普通的定理。事实证明，那些你以为是通过接受类型而免费获得的定理 [10], 从来都不是免费的。你付出的代价正是自由本身。

类型系统的替代方案，是不让运算作为项独立存在。当然，仍然必须能够谈论运算，而这正是算子的作用。谈论运算，就是将算子应用于运算。因此，解决方案是以这样的方式定义项：直接表示算子应用，而不赋予运算（或者算子本身）脱离这种应用而存在的资格：

$$\lambda\, x.\, D\, f[x] \quad \equiv \quad \mathscr{R}\, x.\, D\, (x, f[x]).$$

这样，每个项都会求值得到一个值。接下来的章节将对此作出精确定义。

4 语法

签名 是由一组称为抽象的名称构成的集合，其中每个抽象都与一个（抽象）形状相关联。抽象形状是一个列表 $[p_1, \ldots, p_n]$，其中每个 p_i 都是 $\overline{m} = \{1, 2, \ldots, m\}$的子集，并且满足 $\overline{m} = \bigcup_{i=1}^{n} p_i$。接着， n 称为该形状／抽象的元数 ，而 m 称为它的价数。

$\mathfrak{S}$-代数是一个抽象代数 $\mathcal{A}$，并配有一个签名 $\mathfrak{S}$，使得每个形状为 $[p_1, \ldots, p_n]$ 的抽象 $a \in \mathfrak{S}$ 都与某个形状为 $[|p_1|, \ldots, |p_n|]$的算子$o \in \mathcal{O}(\mathcal{A})$相关联。通常，我们不会区分$\mathfrak{S}$-代数与抽象代数，而是用同一个名称指代二者；在本例中为 $\mathcal{A}$。

设 X 为一个由无限多个变量构成的集合。相对于 X 和给定的签名 $\mathfrak{S}$，一个项要么是变量应用，要么是抽象应用。

一个变量应用具有如下形式

$$x[t_1, \ldots, t_n],$$

其中， $x \in X$ 和 t_i 本身都是项。我们称 x 以元数 n出现。对于 $n = 0$ ，我们也可以直接写作 x ，而不写 $x[]$。

一个抽象应用的形式为

$$(a\, x_1 \ldots x_m . t_1 \ldots t_n),$$

其中， a 是属于该签名的抽象， m 是 a的价数，而 n是 a的元数。 t_i 是表示所要应用 a 的运算的项。 x_j 是互不相同的变量。其思想是，抽象的形状 $[p_1, \ldots, p_n]$ 决定了每个 t_i所绑定的变量。例如，对于$p_i = \{q_1 < \cdots < q_k\}$，项 t_i 绑定变量 x_{q_1}、.....、 x_{q_k}，并表示 k元运算 x_{q_1}、.....、 $x_{q_k} \mapsto t_i$。这里的记法 $\{q_1 < \cdots < q_k\}$ 是 $\{q_1, \ldots, q_k\}$的简写，其中 $q_1 < \cdots < q_k$。

在用抽象表示算子时，选择 p_i 的方式不止一种，至少对于具有 $n > 1$的适当 n 元算子而言是如此。假设算子的形状为 $[1, 1]$, ，则抽象的形状可以是 $[\{1\}, \{2\}]$，也可以是 [{2}、{1}], ，从而得到价数为 2；或者也可以是 $[\{1\}, \{1\}]$，从而得到价数为 1。例如，如果算子的含义是对两个数组 u 和 v 进行加法，那么价数为 1 的抽象似乎更合适，因为直观上加法是同步步进执行的：(add-arrays $i.\, u[i]\, v[i]$)。另一方面，如果算子的含义是执行加法，使得 u 的每个元素都与 v 的每个元素相加，并使结果形成一个矩阵，那么独立索引似乎更好：(matrix-add-arrays $i\, j.\, u[i]\, v[j]$)。不过，选择表示形式 (add-arrays $i\, j.\, u[i]\, v[j]$) 和 (matrix-add-arrays $i.\, u[i]\, v[i]$) 也完全可以，尽管这种选择的品味可能值得商榷。

不必写作 $(a\, x_1 \ldots x_m . t_1 \ldots t_n)$；对于具体情形，可以使用显而易见的自定义语法，例如用于 $n = 2$ 和 $m = 0$的二元运算语法。

5 语义

设 $\mathcal{A}$ 是一个 $\mathfrak{S}$-代数， X 是一个变量集。一个 赋值 ν 到 $\mathcal{A}$ 为每个变量 $x \in X$ 以及每个元数 n 指定一个 $\mathcal{C}(\mathcal{A})$上的 n元运算。

对于互不相同的变量 x_1、. . . 、 x_k 以及 $\mathcal{C}(\mathcal{A})$中的值 u_1、. . . 、 u_k，更新后的赋值定义如下:

$$\nu[x_1 \coloneqq u_1, \ldots, x_k \coloneqq u_k](y, n) = \begin{cases} u_i & \text{for } y = x_i \text{ and } n = 0, \\ \nu(y, n) & \text{otherwise.} \end{cases}$$

赋值 ν为 X中的所有变量赋予含义，而 $\mathcal{A}$ 为 $\mathfrak{S}$中的所有抽象赋予含义。这使得我们能够通过递归考察 t的结构，计算任意项t 相对于 ν和 $\mathcal{A}$ 的值 $[\![t]\!]_\nu$:

– 如果 $t = x[t_1, \ldots, t_n]$ 是一个变量应用，那么

$$[\![t]\!]_\nu = \nu(x, n)([\![t_1]\!]_\nu, \ldots, [\![t_n]\!]_\nu).$$

– 若 $t = (a\, x_1 \ldots x_m . t_1 \ldots t_n)$ 是一个抽象应用，则令 o为与 a 相关联的n元算子，并令 $[p_1, \ldots, p_n]$为 a 的形状。对于$p_i = \{q_1 < \cdots < q_k\}$，在 $\mathcal{C}(\mathcal{A})$ 上的 k元运算 f_i 定义如下

$$(u_1, \ldots, u_k) \mapsto [\![t_i]\!]_{\nu[x_{q_1} \coloneqq u_1, \ldots, x_{q_k} \coloneqq u_k]},$$

由此得到 $[\![t]\!]_\nu = o(f_1, \ldots, f_n)$。

变量的恒等式与其出现时的元数密不可分。项 $x[x]$ 实际上表示 $x^1[x^0]$，其中 x^1 是一个不同于 x^0 的变量。

如果对于所有 $\mathfrak{S}$-代数 $\mathcal{A}$ 以及所有从 ν 到 $\mathcal{A}$ 的赋值 ν，$[\![s]\!]_\nu = [\![t]\!]_\nu$ 都成立，那么称两个项 s 和 t 关于签名 $\mathfrak{S}$ 语义等价。此外，如果两个项用 **de Bruijn 索引** [3] 表示后完全相同，则称它们是 α-等价。可以证明，两个项语义等价，当且仅当它们是 α-等价的。

如果项 t 中存在一个未被任何外围抽象绑定的 x 元数为 n 的出现，则称变量 x 在项 t 中以元数 n 自由出现。由于抽象只绑定元数为 0 的变量，因此 t 中 $n > 0$ 元数的 x 任何出现都是自由的。

显然，t 的值仅取决于赋值 x 中对那些在 t 中自由出现的变量所作的赋值。因此，闭项 t（即不含任何自由变量的项）的值完全不依赖于赋值，而只取决于进行计算的抽象代数。

例 *2.* 考虑抽象代数*Peano*。签名 $\mathfrak{S}_{Peano}$ 通过将*Peano* 的算子名称重用为抽象名称而形成。于是

$$K[0] \Rightarrow ((\forall_N x.\, K[x] \Rightarrow K[(S.\, x)]) \Rightarrow (\forall_N x.\, K[x]))$$

对于取值于*Peano*中的任意赋值，其结果都是 T。

例*3.* 考虑抽象代数*FunSets*。签名 $\mathfrak{S}_{FunSets}$ 通过将*FunSets* 的算子名称重用为抽象名称而形成。于是

$$\begin{gathered}
\textstyle\sum x.\, D\, E \equiv D \times E, \\
\textstyle\prod x.\, D\, E \equiv D \to E, \\
(x : D) \wedge (F : D \to E) \Rightarrow (F \cdot x) : E, \\
(x : D) \Rightarrow (\lambda\, x.\, D\, f[x]) \cdot x \equiv f[x]
\end{gathered}$$

对于取值于*FunSets*中的任意赋值，它们的结果都是 T。

6 代换

一个 n元模板 的形式为 $[x_1 \ldots x_n.\, t]$，其中 x_i 是互不相同的变量，称为模板绑定变量，而 t 是一个项，称为模板体。该模板绑定 t 中元数为 0 的所有 x_i 的自由出现。赋值 ν 将一个 n元模板转化为一个 n元运算，通过

$$[\![[x_1 \ldots x_n.\, t]]\!]_\nu \quad = \quad (u_1, \ldots, u_n) \mapsto [\![t]\!]_{\nu[x_1 := u_1, \ldots, x_n := u_n]}.$$

将项上的语义等价和 α-等价概念扩展到模板上十分直接。若对于所有 $\mathfrak{S}$-代数 $\mathcal{A}$ 以及所有从 ν到 $\mathcal{A}$的赋值，模板 U 和 V所表示的是相同的运算，则称它们关于签名 $\mathfrak{S}$ 是语义等价的。若两个模板是 α-等价的，

则它们使用**de Bruijn 索引** [3] 的表示完全相同。与项一样，语义等价和 α-等价最终证明是相同的。

代换是一种定义在定义域上的函数，它将一个变量映射到一个元数为 n 的模板，前提是 (x,n) 属于 D。代换的目的是应用于一个项/模板，从而得到另一个项/模板作为代换的结果。原则上，这是通过将 t 中属于 σ 定义域的自由变量替换为对应的模板，然后再通过代换将这些模板应用于它们各自的参数来完成的。具体细节比较复杂，因为必须小心避免自由变量被约束变量捕获。也许最简单的方法是将项/模板转换为基于 de Bruijn 索引的表示形式，执行代换，再将结果转换回普通的项/模板。因此，应用代换并没有唯一确定的结果：结果只在 α-等价意义下确定。

代换的主要性质是：给定一个背景赋值 ν，任意代换 σ 都可以通过 $\nu_\sigma(x,n) = [\![\sigma(x,n)]\!]_\nu$，针对 $(x,n) \in D$，以及 $\nu_\sigma(x,n) = \nu(x,n)$，针对 $(x,n) \notin D$，转化为一个赋值 ν_σ。赋值 ν_σ 具有如下性质：对于任意项或模板 t，以下关系均成立：$[\![t]\!]_{\nu_\sigma} = [\![t/\sigma]\!]_\nu$。

7 逻辑、真值与模型

将抽象代数转化为逻辑并不难。这种转化是可能的，或许并不令人意外[3]，但肯定并非显而易见。毕竟，Church 曾尝试用无类型λ演算实现这一点，却失败了 [11]。只有引入类型之后，他才能将λ演算转化为逻辑 [12]。我想，Church 是第一个受害于只需使用*Lambda*TM 的人。

一个前提是一个模板，其中所有模板绑定变量都在其模板体中自由出现。若两个前提仅在模板绑定变量的顺序上有所不同，则视为相同。一个规则是一个有序对 (P, c)，其中 P 是一个有限的前提集合，c 是一个称为结论的项。如果 P 为空，我们也只写 c。

一个抽象逻辑 $\mathcal{L}$是一个签名 $\mathfrak{S}$ 以及一组称为推理规则的规则。没有前提的推理规则称为公理。

设 $\mathcal{A}$ 是一个 $\mathfrak{S}$-代数，且 $\mathsf{T} \in \mathcal{C}(\mathcal{A})$ 是该代数的一个指定值。若项 t 在 $\mathcal{A}$ 中对于 ν满足 $[\![t]\!]_\nu = \mathsf{T}$，则称其为真。若模板 U 在 $\mathcal{A}$ 中对于ν 处处等于 T，则称其为真。若一条规则在 $\mathcal{A}$ 中对于 ν 满足以下条件之一，则称其为真：其结论对于 ν 在 $\mathcal{A}$中为真，或者其某个前提对于 ν 在 $\mathcal{A}$中不为真。

对于 $\mathcal{A}$，赋值空间 $\mathcal{V}$ 是由取值于 $\mathcal{A}$ 的赋值构成的集合，满足：

– $\mathcal{V}$ 非空。– 如果 ν 是属于 $\mathcal{V}$的一个赋值，$u \in \mathcal{C}(\mathcal{A})$，且 x 是一个变量，那么 $\nu[x := u]$也属于 $\mathcal{V}$。– 如果 ν 是属于 $\mathcal{V}$的一个赋值，且 σ 是一个代换（相对于$\mathfrak{S}$），那么 ν_σ 也属于 $\mathcal{V}$。

[3] 不过，这让我很惊讶。请在 [4]中搜索“不一致”。

这种赋值空间的概念足够强大，可以证明抽象逻辑的可靠性，又足够弱，可以处理其完备性。当然，取值于 $\mathcal{A}$ 的所有 赋值所构成的空间显然是一个赋值空间。

对于 $\mathcal{L}$，一个模型 是一个三元组（$\mathcal{A}$,T $\mathcal{V}$），其中 $\mathcal{A}$ 是一个 $\mathfrak{S}$-代数，T $\in \mathcal{C}(\mathcal{A})$ 是表示真值的一个指定但任意的值，$\mathcal{V}$ 是 $\mathcal{A}$的赋值空间，并且对于所有 $\nu \in \mathcal{V}$，$\mathcal{L}$ 的所有推理规则在 $\mathcal{A}$ 中都为真。如果 $\mathcal{V}$ 是所有赋值到 $\mathcal{A}$中的空间，则该模型称为标准 模型；否则称为非标准模型。

如果 $\mathcal{A}$ 的载体只包含一个值（该值必然是 T），则称该模型为退化 模型。请注意，每种 抽象逻辑都有退化模型。

如果 $\mathcal{L}$ 的每个模型也是与 $\mathcal{L}$具有相同签名、且只有一个推理规则 r的逻辑的模型，那么我们称 r 在 $\mathcal{L}$ 中有效，并记作

$$\mathcal{L} \models r.$$

8 证明

如果两个规则的结论 α-等价，并且其中一个规则的每个前提都与另一个规则中的某个前提 α-等价，则称这两个规则为 α-等价。将代换 σ 应用于规则 r，得到规则 r/σ：将 r 应用于 r的前提（并通过删除未使用的模板绑定变量来规范化所得模板）和结论。

给定具有签名 $\mathfrak{S}$ 的逻辑 $\mathcal{L}$，规则 r 的证明 $\mathbf{p}_r$ in $\mathcal{L}$ 可以是**恒真式**、**代换**或**推理**:

– 恒真式 ***True()*** 证明 r，其中 r 是一条推理规则。– 代换 ***Subst(, p,)*** 证明 r，其中 p_s 是 s 的证明，且 r 与 α 等价于 s/σ。– 推理 ***Infer(, p, p)*** 证明 r，并满足以下假设：

- p_s 是 $s = (\{H_1, \ldots, H_n\}, c)$ 的证明，其中 $H_1 = [x_1 \ldots x_k . h]$,
- p_t 是 $t = (\{G_1, \ldots, G_m\}, h)$ 的证明，其中任何 G_j 的模板绑定变量都不等于任何 x_i, 并且
- $r = (\{G'_1, \ldots, G'_m, H_2, \ldots, H_n\}, c)$，其中 G_j 通过将那些在 G_j 的主体中以元数 0 自由出现的 x_i 加为模板绑定符，得到 G'_j。

如果存在 $\mathcal{L}$中的规则 r 的证明，我们就称 r 是 $\mathcal{L}$ 的定理，并记作

$$\mathcal{L} \vdash r.$$

9 可靠性

抽象逻辑是可靠的，即 $\mathcal{L}$ 的每个定理在 $\mathcal{L}$中也都是有效的：

$$\mathcal{L} \vdash r \quad \text{implies} \quad \mathcal{L} \models r \quad \text{for every rule } r.$$

我们通过对证明的结构进行归纳法，证明如果 p_r 是 $\mathcal{L}$ 中 r 的证明，那么 r 在 $\mathcal{L}$ 中有效：

– 假设 $\mathrm{p}_r = \mathrm{True}(r)$。那么 r 是一条推理规则，因此在 $\mathcal{L}$ 的每个模型中都为真。

– 假设 $\mathrm{p}_r = \mathrm{Subst}(r, \mathrm{p}_s, \sigma)$，其中 p_s 是 s 的一个证明，并且

$$r = (\{H_1, \ldots, H_m\}, d) \quad \text{and} \quad s = (\{G_1, \ldots, G_n\}, c).$$

这里 d 和 c/σ 是 α-等价的，因此也语义等价；并且对于每个 $i \in \overline{n}$，都存在一个 $j \in \overline{m}$，使得 H_j 和 G_i/σ 也语义等价。现在设 $(\mathcal{A}, \mathsf{T}\ \mathcal{V})$ 是 $\mathcal{L}$ 的一个模型，且 $\nu \in \mathcal{V}$。那么

$$[\![d]\!]_\nu = [\![c/\sigma]\!]_\nu = [\![c]\!]_{\nu_\sigma}, \quad \text{and} \quad [\![H_j]\!]_\nu = [\![G_i/\sigma]\!]_\nu = [\![G_i]\!]_{\nu_\sigma}.$$

由于 $\mathcal{V}$ 构成一个赋值空间，$\nu_\sigma \in \mathcal{V}$；又由于 s 有效，要么 $[\![c]\!]_{\nu_\sigma} = \mathsf{T}$，要么存在 $i \in \overline{n}$，使得对于某些 $u_1, \ldots, u_k \in \mathcal{C}(\mathcal{A})$，$[\![G_i]\!]_{\nu_\sigma}(u_1, \ldots, u_k) \neq \mathsf{T}$。但这意味着，要么 $[\![d]\!]_\nu = \mathsf{T}$，要么存在 $j \in \overline{m}$，使得 $[\![H_j]\!]_\nu(u_1, \ldots, u_k) \neq \mathsf{T}$。因此，$r$ 有效。

– 假设 $\mathrm{p}_r = \mathrm{Infer}(r, \mathrm{p}_s, \mathrm{p}_t)$，其中 $s = (\{H_1, \ldots, H_n\}, c)$、$H_1 = [x_1 \ldots x_k.\, h]$、$t = (\{G_1, \ldots, G_m\}, h)$ 和 $r = (\{G'_1, \ldots, G'_m, H_2, \ldots, H_n\}, c)$。令 $(\mathcal{A}, \mathsf{T}\ \mathcal{V})$ 为 $\mathcal{L}$ 的一个模型，并且 $\nu \in \mathcal{V}$。我们需要证明，对于 ν，r 为真；假设 s 和 t 在 $\mathcal{L}$ 中都有效。如果 $[\![c]\!]_\nu = \mathsf{T}$，那么 r 对于 ν 为真，证明即告完成。因此假设 $[\![c]\!]_\nu \neq \mathsf{T}$。由于 s 有效，存在某个 $i \in \overline{n}$，使得 H_i 相对于 ν 并非为真。如果 $i \geq 2$，那么 H_i 也是 r 的一个前提；于是 r 为真，证明再次完成。因此假设 $i = 1$。那么，对于某个 u_1，....., $u_k \in \mathcal{C}(\mathcal{A})$，以及 $\nu' = \nu[x_1 := u_1, \ldots\ldots, x_k := u_k]$，我们有 $[\![h]\!]_{\nu'} \neq \mathsf{T}$。由于 $\nu' \in \mathcal{V}$，且 t 有效，这意味着存在 $j \in \overline{m}$，使得 $[\![G_j]\!]_{\nu'} \neq \mathsf{T}$。对于满足 $\{y_1, \ldots, y_l\} \cap \{x_1, \ldots, x_k\} = \emptyset$ 的 $G_j = [y_1 \ldots y_l.\, g]$，这意味着存在 $v_1, \ldots\ldots, v_l \in \mathcal{C}(\mathcal{A})$，使得对于 $\nu'' = \nu'[y_1 := v_1, \ldots\ldots, y_l := v_l]$，我们有 $[\![g]\!]_{\nu''} \neq \mathsf{T}$。令 $x_{\alpha_1}, \ldots\ldots, x_{\alpha_q}$ 为 $\{\alpha_1 < \cdots < \alpha_q\} \subseteq \overline{k}$ 的那些变量，它们属于 $x_1, \ldots\ldots, x_k$，并且在 g 中以元数 0 自由出现；再假设 $G'_j = [x_{\alpha_1} \ldots x_{\alpha_q}\, y_1 \ldots y_l.\, g]$。那么 $[\![G'_j]\!]_\nu(u_{\alpha_1}, \ldots\ldots, u_{\alpha_q}, v_1, \ldots\ldots, v_l) = [\![[x_{\alpha_1} \ldots x_{\alpha_q}\, y_1 \ldots y_l.\, g]]\!]_\nu(u_{\alpha_1}, \ldots\ldots, u_{\alpha_q}, v_1, \ldots\ldots, v_l) = [\![g]\!]_{\nu''} \neq \mathsf{T}$。因此，$G'_j$ 相对于 ν 并非为真，这使得 r 对于 ν 为真。

10 使用等号的演绎逻辑

一个重要的抽象逻辑是带等号的演绎逻辑 $\mathcal{L}_E$。它的签名由四个抽象组成：真值 T、蕴含 $\Rightarrow$、等号 $\equiv$ 和全称量化 $\forall$。它具有以下 10 条推理规则，其中 8 条是公理：

Modus Ponens	$(\{A \Rightarrow B,\ A\}, B)$
Universal Introduction	$(\{[x.\, P[x]]\}, \forall x.\, P[x])$
Truth_1	T
Truth_2	$A \Rightarrow (A \equiv \mathsf{T})$
$\mathrm{Implication}_1$	$A \Rightarrow (B \Rightarrow A)$
$\mathrm{Implication}_2$	$(A \Rightarrow (B \Rightarrow C)) \Rightarrow ((A \Rightarrow B) \Rightarrow (A \Rightarrow C))$
$\mathrm{Universal}_1$	$(\forall x.\, A[x]) \Rightarrow A[x]$
$\mathrm{Universal}_2$	$(\forall x.\, A \Rightarrow B[x]) \Rightarrow (A \Rightarrow (\forall x.\, B[x]))$
$\mathrm{Equality}_1$	$x \equiv x$
$\mathrm{Equality}_2$	$(x \equiv y) \Rightarrow (A[x] \Rightarrow A[y])$

如果逻辑 $\mathcal{L}'$ 的每个抽象都是逻辑 $\mathcal{L}$ 的抽象，且还是逻辑 $\mathcal{L}$ 中形状相同的抽象；如果对于 $\mathcal{L}$ 的每条推理规则，都存在一条与之对应的、 α 等价的 $\mathcal{L}'$ 推理规则；并且 $\mathcal{L}'$ 中没有对应于 $\mathcal{L}$ 推理规则的每条推理规则实际上都是公理，那么就称逻辑 $\mathcal{L}'$ 是逻辑 $\mathcal{L}$ 的公理扩展。构造逻辑 $\mathcal{L}_{Peano}$ 和 $\mathcal{L}_{Sets}$ 并不困难，它们是 $\mathcal{L}_E$ 的公理扩展，并且（*Peano*，T $\mathcal{V}_{Peano}$）是 $\mathcal{L}_{Peano}$ 的模型，（集合，T $\mathcal{V}_{Sets}$）是 $\mathcal{L}_{Sets}$ 的模型。由于*FunSets* 是集合的定义扩展，因此通过将这些定义作为公理添加到 $\mathcal{L}_{Sets}$ 中来构造 $\mathcal{L}_{FunSets}$，而（*FunSets*，T $\mathcal{V}_{FunSets}$）是一个模型。这里，$\mathcal{V}_{Peano}$、$\mathcal{V}_{Sets}$ 和 $\mathcal{V}_{FunSets}$ 由分别映射到相应抽象代数中的所有赋值组成。

11 Rasiowa模型

给定任意逻辑 $\mathcal{L}$，它是 $\mathcal{L}_E$ 的 **公理扩展**，可以为 $\mathcal{L}$ 构造一个模型，我称之为 $\mathcal{L}$ 的 ***Rasiowa***模型 $\mathfrak{R}$。令 $\mathcal{L}$ 的签名为 $\mathfrak{S}$，并令关于 $\mathfrak{S}$ 的项集合为 $\mathcal{T}$。在 $\mathcal{T}$ 上定义关系 $\approx$ 如下:

$$s \approx t \quad \text{iff} \quad \mathcal{L} \vdash s \equiv t.$$

显然，$\approx$ 是等价关系。例如，根据 $\mathcal{L}_E$ 的等号公理 $_1$，$\approx$ 具有自反性。令 $t \in \mathcal{T}$ 关于 $\approx$ 的等价类为 $[t]$，并令所有这类等价类构成的集合为 $\mathcal{U}$。于是，Rasiowa模型

$$\mathfrak{R} = (\mathcal{A}_{\mathfrak{R}}, [\mathsf{T}], \mathcal{V}_{\mathfrak{R}})$$

以 $\mathcal{U}$ 作为 $\mathcal{A}_{\mathfrak{R}}$ 的载体。接下来需要定义 $\mathfrak{S}$-代数$\mathcal{A}_{\mathfrak{R}}$ 的运算，并定义赋值空间 $\mathcal{V}_{\mathfrak{R}}$。每个 n 元模板 $T = [x_1 \dots x_n . t]$ 都可以视为函数 $\Phi(T)$: $\mathcal{U}^n \to \mathcal{U}$，它将 $([r_1], \dots, [r_n])$ 映射为 $[t/\sigma]$，其中 $\sigma = \{r_1/x_1, \dots, r_n/x_n\}$ 是将 x_i 替换为 r_i 的代换。为说明 $\Phi(T)$ 定义良好，假设 $r_1 \approx s_1, \dots, r_n \approx s_n$，并注意到 $x_1 \equiv y_1 \Rightarrow$

$\cdots \Rightarrow x_n \equiv y_n \Rightarrow A[x_1, \dots, x_n] \equiv A[y_1, \dots, y_n]$ 是 $\mathcal{L}$的一个定理。将A 实例化为 $[x_1 \dots x_n . t]$、将 x_i 实例化为 r_i，并将 y_i 实例化为 s_i，然后结合 n 次肯定前件式的应用即可证明 $t/\{r_1/x_1, \dots, r_n/x_n\} \approx t/\{s_1/x_1, \dots, s_n/x_n\}$。对于$n = 0$，模板 T 只是一个项 t，而 $\Phi(T) = \Phi(t) = [t/\{\}] = [t]$。这促使我们将 $\approx$ 从项扩展到模板，使得 $S \approx T$ 当且仅当 $\Phi(S) = \Phi(T)$。

现在设 $a \in \mathfrak{S}$ 是形状为 $[p_1, \dots, p_n]$的抽象。与 a 相关联的算子 o 通过如下定义 $o(f_1, \dots, f_n)$来构造。若对于每个运算 f_i，都存在一个模板 G_i 使得 $\Phi(G_i) = f_i$，那么我们选择满足$G_i = [x_{\alpha_1^i} \dots x_{\alpha^i_{|p_i|}} . g_i]$的 G_i，其中$p_i = \{\alpha_1^i < \cdots < \alpha^i_{|p_i|}\}$。这里，$x_1$、$\dots$、$x_m$ 是任意互不相同的 m 变量，而 m 是 a 的价数。于是，$o(f_1, \dots, f_n) = [(a\, x_1 \dots x_m . g_1 \dots g_n)]$。这一定义再次是良定义的，因为即使我们碰巧选择了不同的 $G_i' = [y_{\alpha_1^i} \dots y_{\alpha^i_{|p_i|}} . g_i']$，仍然有 $G_i \approx G_i'$，因此$(a\, x_1 \dots x_m . g_1 \dots g_n) \approx (a\, y_1 \dots y_m . g_1' \dots g_n')$。否则，如果某个 f_i 不在 Φ的像中，则任意选择一个结果，例如令 $o(f_1, \dots, f_n) = [\mathsf{T}]$。

如果对于每个变量 x 和每个元数 n，都存在一个模板 G，使得 $\nu(x, n) = \Phi(G)$，则将赋值 ν 到 $\mathcal{A}_{\mathfrak{R}}$ 称为简单。描述简单

赋值的另一种方式是使用处处有定义的代换。对于这样的代换 σ，通过 $\sigma^*(x,n) = \Phi(\sigma(x,n))$ 定义赋值 σ^*。赋值 σ^* 显然是简单的。同样显然的是，每个简单赋值 ν 都可以表示为某个处处有定义的代换 σ 对应的 $\nu = \sigma^*$。以下简单赋值的重要性质很容易证明：$[\![t]\!]_{\sigma^*} = [t/\sigma]$。

空间 $\mathcal{V}_{\mathfrak{R}}$ 由那些简单的赋值构成。它确实是一个赋值空间。首先，$\mathcal{V}_{\mathfrak{R}}$ 非空：令 κ 为由 $\kappa(x,n) = [y_1 \ldots y_n.\, x[y_1,\ldots,y_n]]$ 定义的典范替换。于是 $\kappa^* \in \mathcal{V}_{\mathfrak{R}}$。其次，假设 $\nu \in \mathcal{V}_{\mathfrak{R}}$, $u \in \mathcal{U}$，并令 x 为一个变量。那么存在 σ 和 t，使得 $\nu = \sigma^*$ 且 $u = [t]$。因此，$\nu[x := u] = \sigma^*[x := [t]] = (\sigma[x := t])^* \in \mathcal{V}_{\mathfrak{R}}$。第三，假设 $\nu \in \mathcal{V}_{\mathfrak{R}}$，并令 θ 为一个定义域为 D 的代换。那么存在 σ，使得 $\nu = \sigma^*$。对于 $(x,n) \notin D$，$\nu_\theta(x,n) = \nu(x,n) = \Phi(\sigma(x,n))$。对于 $(x,n) \in D$，$\nu_\theta(x,n) = \Phi(\theta(x,n)/\sigma)$。综合起来，得到 $\nu_\theta \in \mathcal{V}_{\mathfrak{R}}$。

为了证明 $\mathfrak{R}$ 确实是 $\mathcal{L}$的模型，我们需要确认 $\mathcal{L}$ 的所有推理规则在 $\mathfrak{R}$中都为真。

要确认肯定前件式，假设对于赋值 $\nu = \sigma^* \in \mathcal{V}_{\mathfrak{R}}$，$A \Rightarrow B$ 和 A 在 $\mathfrak{R}$中都为真。这意味着 $[\mathrm{T}] = [\![A \Rightarrow B]\!]_\nu = [\![A \Rightarrow B]\!]_{\sigma^*} = [(A \Rightarrow B)/\sigma] = [A/\sigma \Rightarrow B/\sigma]$，同理 $[\mathrm{T}] = [A/\sigma]$。这意味着 $\mathrm{T} \equiv (A/\sigma \Rightarrow B/\sigma)$ 和 $\mathrm{T} \equiv A/\sigma$ 都是 $\mathcal{L}$中的定理，而且我们还可以推导出定理 $\mathrm{T} \equiv (\mathrm{T} \Rightarrow B/\sigma)$，然后是 $\mathrm{T} \Rightarrow B/\sigma$，接着是 B/σ。使用公理 Truth_2，我们得到 $B/\sigma \equiv \mathrm{T}$，这意味着 $[\![B]\!]_\nu = [\mathrm{T}]$。

要确认全称引入，假设对于赋值 $\nu = \sigma^* \in \mathcal{V}_{\mathfrak{R}}$，$[x.\, P[x]]$ 在 $\mathfrak{R}$ 中为真。这意味着运算 $u \mapsto [\![P[x]]\!]_{\nu[x:=u]}$ 处处等于 $[\mathrm{T}]$。因此，对于所有 $t \in \mathcal{T}$，我们都有 $[\mathrm{T}] = [\![P[x]]\!]_{\nu[x:=[t]]} = [\![P[x]]\!]_{(\sigma^*)[x:=[t]]} = [\![P[x]]\!]_{(\sigma[x:=t])^*} = [P[x]/\sigma[x := t]] = [\sigma(P,1)(t)]$，这意味着在 $\mathcal{L}$ 中，对于任意项 t，定理 $\mathrm{T} \equiv \sigma(P,1)(t)$ 都可以推导出来。因此，$[\![\forall x.\, P[x]]\!]_\nu = [\![\forall x.\, P[x]]\!]_{\sigma^*} = [(\forall x.\, P[x])/\sigma] = [\forall x.\, \sigma(P,1)(x)] = [\forall x.\, \mathrm{T}]$。由于 $\forall x.\, \mathrm{T}$ 是 $\mathcal{L}_E$ 的定理，这意味着 $[\![\forall x.\, P[x]]\!]_\nu = [\forall x.\, \mathrm{T}] = [\mathrm{T}]$。

为确认 $\mathcal{L}$ 的每个**公理** t 对于 $\nu = \sigma^* \in \mathcal{V}_{\mathfrak{R}}$ 的每种取值在 $\mathfrak{R}$ 中都为真，注意 $[\![t]\!]_\nu = [\![t]\!]_{\sigma^*} = [t/\sigma]$。但由于 t 是一条公理，$\mathcal{L} \vdash t$，因此 $\mathcal{L} \vdash t/\sigma$ 也成立。这意味着 $[\![t]\!]_\nu = [t/\sigma] = [\mathrm{T}]$。

因此，$\mathfrak{R}$ 是 $\mathcal{L}$ 的一个模型。

12 完备性与一致性

如果每个有效项也是定理，即满足以下条件，则称抽象逻辑 $\mathcal{L}$ 是完备的：

$$\mathcal{L} \models t \quad \text{implies} \quad \mathcal{L} \vdash t \quad \text{for every term } t.$$

如果 $\mathcal{L}$ 是 $\mathcal{L}_E$ 的公理扩展，那么它就是完备的。为此，构造 $\mathcal{L}$ 的 Rasiowa模型。如果 t 是一个有效项，那么 t 必定在 $\mathcal{L}$ 的每个模型中都为真，因此在 Rasiowa模型中也为真。于是，对于任意赋值 $\nu \in \mathcal{V}_{\mathfrak{R}}$，$[\![t]\!]_\nu = [\mathrm{T}]$。特别地，使用典范替换 κ，有 $[\mathrm{T}] = [\![t]\!]_{\kappa^*} = [t/\kappa] = [t]$。但这意味着 $\mathrm{T} \equiv t$ 是 $\mathcal{L}$ 的一个定理，因此 $\mathcal{L} \vdash t$。

如果所有项也都是定理，则抽象逻辑 $\mathcal{L}$ 被称为 不一致；如果至少存在一个不是定理的项，则称为 一致。

如果 $\mathcal{L}$ 是 $\mathcal{L}_E$的公理扩展，那么 $\mathcal{L}$ 的不一致性等价于$\mathcal{L} \vdash \forall x.\, x$。这一点很容易看出。假设 $\forall x.\, x$ 是 $\mathcal{L}$中的一个定理。将 $[x.\, x]$代入公理 Universal_1 中的 A，并应用肯定前件式，可推出 x 是一个定理。将任意项 t 代入 x，即可表明 t 是一个定理。

如果抽象逻辑 $\mathcal{L}$ 不一致，那么它的所有模型都是退化的。为说明这一点，假设 $\mathcal{L}$ 不一致。因此，对于任意变量 x，x 都是一个定理。这意味着，x 在 $\mathcal{L}$ 的任意模型 $(\mathcal{A}, \mathrm{T}\ \mathcal{V})$ 中都有效。特别是，对于 $\mathcal{V}$ 中的任意赋值 ν，我们都有 $[\![x]\!]_\nu = \mathrm{T}$。这样的 ν 确实存在，因为 $\mathcal{V}$ 非空。对于任意值 $u \in \mathcal{C}(\mathcal{A})$，赋值 $\nu[x := u]$ 也属于 $\mathcal{V}$，因此$u = [\![x]\!]_{\nu[x:=u]} = \mathrm{T}$。这意味着 $\mathcal{C}(\mathcal{A}) = \{\mathrm{T}\}$。因此，逻辑 $\mathcal{L}_{Peano}$、$\mathcal{L}_{Sets}$ 和 $\mathcal{L}_{FunSets}$都是一致的，因为它们都有非退化模型。

那么反过来呢？如果抽象逻辑 $\mathcal{L}$ 的每个模型都是退化的，那么对于任意项 t 和任意赋值 ν，$[\![t]\!]_\nu = \mathrm{T}$都成立，因此 $\mathcal{L} \models t$。如果 $\mathcal{L}$ 是完备的，这就意味着 $\mathcal{L} \vdash t$，从而使 $\mathcal{L}$ 不一致。

对于完备逻辑 $\mathcal{L}$，例如所有公理上扩展 $\mathcal{L}_E$ 的逻辑，$\mathcal{L}$ 的不一致性因此等价于 $\mathcal{L}$ 的所有模型都是退化的。

13 相关工作与未来工作

鉴于我主张 AL 是正确的逻辑，相关工作这一节或许本身就应该写成一篇论文。当然，这里不可能做到这一点。因此，我认为将 AL 与逻辑领域已有的研究联系起来属于未来工作。例如，将类型论和直觉主义逻辑表示为抽象逻辑似乎很直接。这对类型论和直觉主义逻辑意味着什么？将简单类型论表示为一种 AL 时，简单类型论的亨金模型与 AL 模型有何关联？和/或通过简单类型论对 AL 进行编码时又如何？诸如此类。因此，我只会简要谈谈与 AL 直接相关的两个研究领域。

抽象逻辑是对 Rasiowa 关于非标准命题逻辑的**代数语义**研究的推广 [13]。Rasiowa 仿照 Mostowski 的做法，将量词解释为最小上界和最大下界，从而把她的方法推广到谓词逻辑。在我看来，这似乎不是将 Rasiowa 的方法推广到命题逻辑之外的恰当方式。我认为抽象逻辑才是。

AL（以及 Practal）的一个重要灵感来源是 Isabelle 逻辑框架 [14]。它的元逻辑 $\mathcal{M}$ [15] 基于直觉主义高阶逻辑，而后者基于 λ-演算。因此，和大多数其他逻辑框架一样，Isabelle 也是一种只需使用*LambdaTM*。若要证明建立在 $\mathcal{M}$之上的对象逻辑是可靠且完备的，就必须针对每一种这样的对象逻辑分别给出论证。与此相反，每一种抽象逻辑相对于一种简单语义都是可靠的；如果它是在 $\mathcal{L}_E$的基础上以公理方式扩展而来，也能保证完备。尽管如此，AL 终究也只是代数，这似乎预示着它在实际应用中前景良好：AL 不仅可以作为证明助手的基础，也可以用于计算机代数系统、自动定理证明，以及任何需要基于简单语义的一般形式语言的其他应用。

14 致谢

Norbert Schirmer 曾对这篇论文的早期版本提出意见，并提出了一些切中要害的问题，帮助我构思这篇论文的叙述方式和结构。

参考文献

1. Steven Obua. 抽象逻辑：计算机数学的新基础。https://arxiv.org/abs/2207.05610，2022年7月。2. Steven Obua. 抽象逻辑哲学。https://doi.org/10.47757/pal.2，2021年12月。3. Steven Obua. 抽象逻辑。https://doi.org/10.47757/abstraction.logic.2，2021年11月。4. Steven Obua. 实用类型。https://doi.org/10.47757/practical.types.1，2021年7月。5. Helena Rasiowa. 现代数学导论。爱思唯尔，1973年。6. *Practal* — 实用逻辑：数学思维的自行车。https://practal.com，访问于2023年4月。7. Giuseppe Peano, Hubert C. Kennedy. *Giuseppe Peano* 选集。多伦多大学出版社，1973年12月。8. C. Dunne, J.B. Wells, F. Kamareddine. 为 *ZF* 集合论添加抽象屏障。智能计算机数学，2020年7月。9. Henk Barendregt. 广义类型系统导论。《函数式编程杂志》，1991年4月。10. Phil Wadler.免费的定理！第四届函数式编程语言与计算机体系结构国际会议论文集，1989年11月。11. S. C. Kleene, J. B. Rosser.某些形式逻辑的不一致性。《数学年刊》，1935年7月。12. Alonzo Church. 简单类型论的一种表述。《符号逻辑杂志》，1940年6月。13. Helena Rasiowa. 非经典逻辑的代数方法。北荷兰出版公司 / 爱思唯尔，1974年5月。14. Lawrence C. Paulson, Tobias Nipkow, Makarius Wenzel. 从 *LCF* 到 *Isabelle/HOL*。《计算的形式方面》，2019年9月。15. Lawrence C. Paulson. 通用定理证明器的基础。《自动推理杂志》，1989年9月。