arXiv:2304.08068v1 [cs.LO] 17 Apr 2023

一种理论无关的 Curry-De Bruijn-Howard 对应

Gilles Dowek

INRIA，法国巴黎 13 区 Cedex 13，意大利大道 23 号，CS 81321，邮编 75214。gilles.dowek@inria.fr

布劳威尔、海廷和科尔莫哥洛夫提出将构造性证明定义为算法，例如，将 $A \Rightarrow B$ 证明视为一种算法：以 A 的证明为输入，并返回 B 的证明。柯里、德布鲁因和霍华德进一步发展了这一思想。首先，他们提出用λ演算来表示这些算法，例如，将 $\lambda f^{A \Rightarrow A \Rightarrow B} \lambda x^{A}$ $(f\ x\ x)$写作命题（$A \Rightarrow A \Rightarrow B$） $\Rightarrow A \Rightarrow B$ 的证明：该算法以 f 的证明（$A \Rightarrow A \Rightarrow B$ ）和 x 的证明（ A ）为输入，并返回通过将 f 应用于 x 两次而得到的 B 的证明。然后，他们指出，由于 $A \Rightarrow B$的证明将 A 的证明映射为 B的证明，其类型 proof($A \Rightarrow B$) 是 proof(A) $\rightarrow$ proof(B)。因此，将命题映射到其证明类型的函数 proof 是一个态射，它将运算 $\Rightarrow$ 变换为运算 $\rightarrow$。同理，这个态射将证明中的切消去变换为λ项中的β-归约。

这种将证明表示为 λ项的方式已广泛应用于证明处理系统中，例如 Automath、Nuprl、Coq、Elf、Agda 等。与自然演绎或相继式演算证明树相比，λ演算能更紧凑地表示证明。这种表示方式很方便，例如可以将证明存储在磁盘上，并通过网络传输。

由此发展出了多种带类型 λ 演算：Automath、系统 F、系统 Fω、λ-Pi 演算、直觉主义类型论的 Martin-L¨、构造演算、归纳构造演算等。我们不禁会问，为什么需要这么多不同的演算。

在某些情况下，λ演算之间的差异反映了表达证明所用逻辑的不同：例如，有些演算表达构造性证明，另一些则表达经典证明。在其他情况下，这些差异反映了用于定义证明的归纳规则不同：有些演算以自然演绎为基础，另一些则以相继式演算为基础。但大多数时候，这些差异反映的是表达证明所用理论的不同：算术、类理论（又称二阶逻辑）、简单类型论（又称高阶逻辑）、直谓类型论等。

与其针对每种特定理论分别开发定制的类型化 λ演算，我们不妨尝试设计一种通用的参数化演算，以便表达任意理论的证明。这样一来，在 λ演算中表达证明的问题，就能与选择理论的问题完全分开。

一种实现方式是从 λ-Pi 演算出发，该演算旨在表达最小谓词逻辑中的证明，并以公理化的

方式为每条公理声明一个变量或常量。这就是逻辑框架 [8] 所采用的方法。然而，这种方法的一个局限是，在存在公理时，β-归约过于弱，因此我们需要添加特定于公理的证明归约规则，例如 G¨del 系统 T 中针对归纳公理的规则，以便在特定理论中模拟切消去。

我们在 [5] 提出了一种不同的方法：理论不是通过公理表达，而是像模态演绎 [6,7]那样通过重写规则表达。这催生了模 lambda-Pi 演算及其实现系统 Dedukti[2]。

尽管 Dedukti 只是一个证明检查器，但它是一个通用证明检查器 [3]。通过选择适当的重写规则，可以对模 lambda-Pi 演算进行参数化，以表达任何能够用模态演绎表达的理论的证明，例如算术、类理论、简单类型论、某些版本的集合论等。通过选择适当的重写规则，lambda-Pi 演算还可以模拟系统 F、系统 Fω、构造演算 [5], 、归纳构造演算 [4], 等。这促成了将证明从系统 Coq 翻译到 Dedukti[4] ，以及从系统 HOL 翻译到 Dedukti [1]的工具系统的开发。

这个通用证明检查器开辟了仍有待探索的新研究方向。首先，如果我们在理论 $\mathcal{T}_1$ 中证明命题 $A \Rightarrow B$ ，并在理论 $\mathcal{T}_2$中证明命题 A ，会发生什么？是否存在一种理论，可以在其中推导出 B？当然，如果理论 $\mathcal{T}_1$ 和 $\mathcal{T}_2$ 互不相容——例如带选择公理的集合论与带选择公理否定的集合论——那么无论在哪里推导出 B 都毫无意义。不过，也存在这样的情况：在 lambda-Pi- calculus modulo 中表示 $\mathcal{T}_1$ 和 $\mathcal{T}_2$ 的重写系统之一是另一个的子集，或者这两个系统的并集定义了一个相容理论，又或者一种理论的命题和证明可以转换到另一种理论中。在所有这些情况下，即使这些证明是在不同的理论和不同的系统中构建的，从 $A \Rightarrow B$ 和 A的证明中推导出 B也是有意义的。

更一般地说，尽管大多数证明处理系统都建立在强大的理论之上——如简单类型论、归纳构造演算等——但我们知道，在这些系统中构造的许多证明只用到了这类理论的一小部分能力。明确给出定义这些理论的公理或重写规则，有助于识别每个证明中使用了哪些公理或规则，这与我们大致了解非形式数学的哪些部分依赖选择公理、哪些部分不依赖选择公理的方式类似。

这种分析或许是开发证明库的第一步。在这些证明库中，证明的分类依据不再是它们在哪个系统中构造，而是它们使用了哪些公理和规则；这将实现证明系统之间真正的互操作性。

参考文献

1. A. Assaf, *Translating HOL in the lambda-Pi-calculus modulo*, 硕士论文，编写中，2012年。

Conception d'un noyau de vérification de preuves pour le lambda-
2. M. Boespflug, *Pi-calcul modulo* 2011 ,

博士论文，巴黎综合理工学院，。 3. M. Boespflug、Q. Carbonneaux 和 O. Hermant，《模 lambda-Pi 演算作为通用证明语言》，*Second International Workshop on Proof Exchange for Theorem Proving* 2012 ，。 4. M.

Boespflug 和 G. Burel，CoqInE：将归纳构造演算转换为模 lambda-Pi 演算，*Second International Workshop onProof Exchange for Theorem Proving*，2012年。 5. D. Cousineau 和 G. Dowek，将纯类型系统嵌入模 lambda-Pi 演算，载于 S. Ronchi Della Rocca，*Typed lambda calculi and applications*，《计算机科学讲义》4583，施普林格出版社，2007年，第102-117页。 6. G. Dowek、Th. Hardin 和 C. Kirchner，模定理证明，*Journal of Automated Reasoning* 31 2003 33 72 ，，，第-页。 7.

R. Harper、F. Honsell 和 G. Plotkin，模证明规范化，*The Journal of Symbolic Logic* 68 4 2003 1289 1316 ，，，，第-页。

8. R. Harper、F. Honsell 和 G. Plotkin，定义逻辑的框架，*The Journal of the ACM* 40 1 1993 ，，，。