

立方类型论入门

∞ -type Café 暑期学校讲座

Jun 19, 2023

Tesla Zhang

teslaz@cmu.edu

目录

1. 前置知识	2
1.1. 基础方面的假设和符号约定	2
2. 立方类型论的动机	3
3. 新的相等类型	4
3.1. 依值相等	5
3.2. 类型转换	7
4. 面映射	8
4.1. 立方子类型	9
5. 双层类型论	10
6. 补齐操作	11
6.1. 不完整类型	12
6.2. 补齐的计算	13
7. 泛等公理	14
7.1. 胶合类型的形成规则	15
7.2. 胶合类型其余的规则	16
7.3. 独立的宇宙补齐	17
7.4. V 类型	18
8. 高阶归纳类型	18
8.1. 立方类型论中的高阶归纳类型	19
9. 趣味问题一: 取反操作	20
9.1. 趣味应用: 中点问题	21
10. 趣味问题二: 缝合补齐	22
10.1. 趣味应用: Brunerie–Ljungström 数	23
11. 术语中英对照表	24
参考文献	25

全文使用香蕉空间风格的术语翻译和书写规范. 例如非中文的人名统一保留源语言中的拼写, 笔者不知道中文名写法的中文名也会使用英语写法. 简短的字母, 编程语言的名字和符号也不翻译. 比如, 相等类型的消去规则保留 J 规则这个名字, 数学公式中「胶合类型」也会写作 Glue . 标点符号全部使用半角. 作者水平有限, 讲义也是在仓促之间写就, 若读者发现本文有不连贯的地方, 遗漏或者错误, 请联系作者.

在开始之前, 笔者引用另一位主讲人狗先生的一段话 (标点符号保留原文)¹:

在寒冷的夜晚, 两个孩子生起了篝火, 越来越多迷路的孩子来到了火堆边. 他们感到无比幸福. 天亮了, 孩子们慢慢地找到了自己的路, 于是一个个地远去了. 但他们永远不会忘记那个曾经给予过他们温暖的篝火.

¹<https://infinity-type-cafe.github.io/ntype-cafe-summer-school>

另外感谢其它几位主讲人和暑校相关人员对本文档早期版本的建议. Alias Qli, 娜娜琪和面皮都给出了很有价值的建设性建议. 另外「无懈可击99」问了一些笔者原本打算省略掉的内容, 虽说丰富了讲义, 但也导致增加了很多工作负担, 我真是谢谢你.

1. 前置知识

读者应该了解同伦类型论中的视角和基本概念:

- 相等类型视作空间中的道路后, 类型具有的空间结构,
- 泛等公理的定义和基本用法,
- 常见的高阶归纳类型.

读者需要非常熟悉 Martin-Löf 类型论中的基本概念:

- 类型论的基本概念: 形成规则, 构造规则, 消去规则, 计算规则, 唯一性规则,
- 归纳类型的基本概念, 包括模式匹配的记号,
- 相等类型本身的定义, 包括 J 规则,
- 宇宙的概念, 包括宇宙的层次结构, 如何避免一致性问题等. 宇宙记作 \mathcal{U} .

这也意味着读者需要熟悉有类型 λ 演算中的基本概念:

- 基于 de Bruijn 编号的无名表示,
- 类型论和逻辑学的类比, 换言之即 Curry-Howard 对应,
- 常见的归纳类型的定义, 例如 \mathbb{N} , \mathbb{Z} , List 等,
- 语境和类型规则.

这些概念和资料可以从不计其数的渠道获取, 这里罗列一些资料:

- **推荐:** ∞ -type Café 暑期学校讲座的其它内容,
- 《类型论简史》, <https://github.com/Trebor-Huang/history>,
- 香蕉空间的类型论板块, <https://www.bananaspace.org/wiki/模板:类型论>,
- 《同伦类型论 – 泛等数学基础》[1] 和它配套的网站.

本讲义会穿插一些语义视角的讨论, 这些讨论会使用较多数学术语, 需要读者对范畴论和抽象同伦论有一定了解, 但尽可能保证这些文本被删除/忽略后, 剩下的内容也是连贯且可读的. 换言之, 本讲义的硬性前置知识基本上只有依值类型论的基本概念, 但如果读者有一些基本的数学背景, 将能学到更多.

本文将会使用相对非正式的语气书写, 不仅是为了降低阅读难度, 也是为了增加乐趣. 笔者最近看了很多柯洁的视频, 对他的把欢声笑语带到专业场合的天赋深感憧憬. 希望本讲义也能让读者认为立方类型论是一个有趣的研究课题.

1.1. 基础方面的假设和符号约定

- 避免关于替换操作的具体实现的讨论. 全文的类型论表达式都会使用带名字的变量, 并假设它们实际上是某种 de Bruijn 编号表示. 例如, $\lambda x. x$ 和 $\lambda y. y$ 是同一个表达式. 换言之, 采取「写作变量名, 读作无名表示」的策略.
- 避免关于宇宙分层的讨论. 避免宇宙导致的一致性非常简单, 不应该在这件事上浪费笔墨. 本讲义在写就时, 精神上遵循的是 Conor McBride 风格的宇宙分层策略 (原名「野蛮但好使的分层」), 参见 [2].
- 在 Russell/Tarski 风格的宇宙之间, 选择弱 Tarski 宇宙, 采取「写作 Russell 宇宙, 读作 Tarski 宇宙」的策略. 但这也不会成为讨论的重点.
- 在《类型论简史》中, 作者使用的「典范性」和「闭典范性」在香蕉空间被管理员否认, 并推荐分别翻译为「正规性」和「典范性」, 这导致「典范性」一词在香蕉空间和《类型论简史》中的含

义不同. 虽然笔者非常喜欢《类型论简史》作者使用的术语, 但考虑到香蕉空间是在线资源, 因此本讲义使用了香蕉空间的翻译.

关于符号, 采用如下约定:

- 使用 $=$ 表示命题等价, 类型写在下标处 (例如 $u =_x v$), 若很明显时会省略. \equiv 表示相等.
- 在书写类型规则时, 若语境在整条规则中都是显然的, 那么会省略类似 $\Gamma \vdash$ 或者 Γ 的前缀. 例如, 函数类型的构造规则写作:

$$\frac{x : A \vdash u : B}{\lambda x. u : A \rightarrow B}$$

- 依值类型的形成规则使用类似 Agda, Nuprl 代码中的语法, 例如 Π 类型写作 $(x : A) \rightarrow B$ 而不是 $\prod_{x:A} B$.
- 函数应用尽可能使用 $f(x)$ 这样的括号语法, 而不是 $f x$ 这样的空格语法.
- 使用 \mathcal{J} 代表任意类型论判断 (也就是写在 $\Gamma \vdash$ 右边的东西), 比如 $u : A, u \equiv v : A$ 这样的表达式都可以用 \mathcal{J} 代指, 并且定义 $\mathcal{J}[x \mapsto v]$ 为「将 \mathcal{J} 中全部的变量 x 都替换为 v 」的操作.
- 类型论规则定义时, 有时会出现多条规则的条件部分是相同的, 但是结论不同. 为了节约篇幅, 我们把这些结论写到一起. 例如, 积类型的消去规则写作:

$$\frac{u : A \times B}{u.1 : A \quad u.2 : B}$$

这个记法有非常重大的问题, 因为 Gentzen 风格相继式演算中有很类似的语法, 但是语义是正好相反的: 相继式演算中在结论处有多个公式的话表示这些公式中的一个成立, 而不是全部都成立. 但是这个语法实在是太方便了, 既方便阅读又节约空间. 我们姑且把它当作是一种自然演绎的变种.

- 积类型的构造规则使用 $\langle u, v, \dots \rangle$ 这个语法.
- 「计算规则」和 β 规则在本讲义中是同义词. 同理「唯一性规则」和 η 规则也是同义词.

2. 立方类型论的动机

同伦类型论中引入了大量的公理:

- 泛等公理: $(A = B) \simeq (A \simeq B)$,
- 高阶归纳类型的存在: 例如 S^1, T^2 , Klein 瓶等经典空间的同伦型.

这些公理破坏了类型论良好的计算性质, 例如它缺少典范性 — 通过泛等公理构造的相等证明不存在对应的表达式形态, 只能保持「对公理的调用」形式. 在这种状态下做形式化证明是不优雅的. 比如我们认为根据合流性应该成立的等式可能并不能直接成立, 导致需要人工证明一些看起来很显然的事情.

但是, 同伦类型论给出的诱惑实在是太大了, 比如它通过泛等公理给出了函数外延性的证明, 以及借助高阶归纳类型的思想给出了一些常用的集合论构造的类型论版本, 如商集 (等价类), 无序列表等. 因此把同伦类型论实现为一个可计算的理论是非常有价值的.

对于研究数学的人而言, 同伦类型论给出了一种描述同伦型的优雅语法, 它的表达力和经典同伦论语言近似 (都能描述基本群, 同伦群, 环路空间, 纬悬, Hopf 映射等常见构造, 也能描述选择公理), 却有着更好的性质, 例如它作为范畴总是有幂对象, 而不需要加上类似紧生成弱 Hausdorff 的条件. 这样的性质也是继承自类型论的优良传统: 能写出来的东西性质都很好.

对于需要发表数学论文的人而言, 同伦类型论作为一种新的语言, 能给许多经典的定理带来新的证明, 也能加深大家对同伦论的理解. 例如它引入的编码-解码法就是很好的例子, 传统的一般拓扑学中, 证明 $\pi_1(S^1)$ 同构于 \mathbb{Z} 的加法群用到了万有覆盖, 而同伦类型论中的证明则是借助了泛等

公理将 S^1 编码为类型. 笔者猜测这受到了类型论中使用宇宙证明 $\text{true} \neq \text{false}$ 的手法的启发, 而这种跨学科的启发是同伦类型论的重大贡献之一.

对于研究类型论的人而言, 立方类型论的设计和实现用到了大量的语义技术, 例如一般的类型论中只有形成, 构造, 消去, 计算, 唯一性这几种规则, 而立方类型论引入了新的 Kan 规则 (就是补齐和转换规则的统称). 除此之外, 立方类型论使用了语境范畴上的自函子 (也就是语境限制操作), 这是一种类似模态的技术, 可以作为入门抽象的模态类型论的动机. 通过了解这些做法, 可以消除对类型论的一些不正确的刻板印象 (例如语境总是由一组类型生成, 类型规则只有固定的几种等), 或许能启发出一些其它的类型论设计.

对于研究编程语言的人而言, 立方类型论展示了一种全新的使用编程语言的方式, 它的表达式在范畴语义中对应着任意维度的立方体, 而不是一维的字符串. 除此之外, 立方类型论可能提供了一种零知识证明的手法, 即给出大定理的证明的正规形式, 这个证明可以由计算机验证, 但立方类型论的证明的特色就是正规形式复杂度突破天际, 难以从中提取出人类可读的证明思路.

立方类型论是一种同伦类型论的典范「实现」. 有如下两种方式看待立方类型论:

- 语义视角: 将依值类型论中的所有对象全部升级为方形集, 这是一种类似单纯集的预层构造, 但使用方形范畴代替单形范畴.
- 语法视角: 通过重新定义 Martin-Löf 类型论中的相等类型以将同伦类型论中的公理转化为新的相等类型上的定理.

除此之外, 立方类型论甚至满足一些超过了 Martin-Löf 类型论的性质, 这也是很有价值的. 本讲义会介绍其中的一部分.

3. 新的相等类型

在经典同伦论中, 对于拓扑空间 X 以及 $x, y \in X$, 其间的道路 $p : x = y$ 表示为一条连续函数 $p : \mathbb{I} \rightarrow X$, 其中 \mathbb{I} 是实数上的单位区间 $[0, 1]$, 满足 $p(0) = x, p(1) = y$. 立方类型论在 Martin-Löf 类型论的基础上, 引入类型 \mathbb{I} 来模拟这个道路的定义, 但实数的理论是非常复杂的, 好在实数的大部分性质我们也不需要 (例如我们不需要它有不可数无穷多个元素), 因此立方类型论使用公理化的定义来模拟它.

定义 3.1:

- \mathbb{I} 是一个类型 (形成规则),
- $0, 1 : \mathbb{I}$ 是空语境下的值 (构造规则).

对 \mathbb{I} 类型的实例的操作必须保持连续. 最常见的反例是模式匹配, 我们不能在 \mathbb{I} 上使用模式匹配. 在这之上可以直接将公理翻译过来, 定义相等类型:

$$\frac{x : X \quad y : X}{x =_X y : \mathcal{U}}$$

在有了类型的定义后, 我们照搬同伦论的思路定义它的构造规则:

$$\frac{i : \mathbb{I} \vdash u : X \quad u[i \mapsto 0] \equiv x \quad u[i \mapsto 1] \equiv y}{\langle i \rangle u : x =_X y}$$

这个记号 $\langle i \rangle u$ 模拟 λ 表达式 $\lambda i. u$. 另一种写法是

$$\frac{i : \mathbb{I} \vdash u : X}{\langle i \rangle u : u[i \mapsto 0] =_X u[i \mapsto 1]}$$

消去规则作为构造规则的对偶, 需要完全反映构造规则给出的信息. 换言之, 若我们知道 $p : x =_X y$, 那么 p 是可以当成一个函数 $\mathbb{I} \rightarrow X$ 来看待的, 而且我们还知道 $p(0) = x, p(1) = y$. 这些信息通过消去规则和计算规则实现.

$$\frac{p : x =_X y \quad i : \mathbb{I}}{p @_{x,y} i : X}$$

消去规则中, 我们把相等类型的两个端点记录在表达式里, 并且在这些信息显然时忽略它们, 简写作 $p @ i$. 在实现立方类型论的编译器时, 若不在语法树中保存这一信息, 就需要在化简表达式时获取 p 的类型, 这是很麻烦的. 为了避免这一问题, 消去规则中显式地记录了 x 和 y (看不懂没关系, 后面不会用到).

$$\frac{p : x =_X y}{p @ 0 \equiv x \quad p @ 1 \equiv y}$$

通过这些规则, 可以证明一些明显的命题, 例如:

命题 3.1: 对于 $x : X$, 有 $x =_X x$. 该命题的证明记作 idp .

证明: $\langle i \rangle x$

□

命题 3.2: 对于 $x, y : X$ 和 $f : X \rightarrow Y$, 有 $x =_X y \rightarrow f(x) =_Y f(y)$. 该命题的证明记作 ap .

证明: $\lambda p. \langle i \rangle f(p @ i)$

□

这个证明性质非常好, 它满足一些 Martin-Löf 类型论版本中不满足的性质, 例如:

$$\text{ap}(f \circ g, p) \equiv \text{ap}(f, \text{ap}(g, p))$$

在 Martin-Löf 类型论中, 需要对 p 进行归纳才能证明这一性质.

命题 3.3: 对于 $f, g : X \rightarrow Y$, 有 $((x : X) \rightarrow f(x) =_Y g(x)) \rightarrow f =_{X \rightarrow Y} g$.

证明: $\lambda p. \langle i \rangle \lambda x. p(x) @ i$

□

但有了这些还不够, 比如我们暂时还没有 J 规则, 目前有的基本操作也不能给我们证明相等的传递性等等.

3.1. 依值相等

我们略微推广相等类型的定义, 允许两端点的类型不同, 但是要求它们的类型是同伦等价的:

$$\frac{A : \mathbb{I} \rightarrow \mathcal{U} \quad x : A(0) \quad y : A(1)}{x =_A y}$$

其它规则都是类似的. 在 Agda 中, 对应依值相等的类型叫做 `PathP`. 注意这不同于 Conor McBride 提出的异质相等类型 (也就是 Idris 语言使用的相等类型), 他的异质相等类型不需要两边的类型同伦等价, 和泛等公理不兼容, 我们不作讨论.

依值相等非常方便. 考虑定长列表类型 `Vec`:

$$\frac{n : \mathbb{N} \quad A : \mathcal{U}}{\text{Vec}(A, n) : \mathcal{U}}$$

容易定义它上面的拼接操作: $\oplus : \text{Vec}(A, n) \rightarrow \text{Vec}(A, m) \rightarrow \text{Vec}(A, n + m)$, 但要想表达它的结合性, 在 Martin-Löf 类型论中并不方便. 考虑如下变量:

$$\begin{aligned} \text{xs} &: \text{Vec}(A, n) \\ \text{ys} &: \text{Vec}(A, m) \\ \text{zs} &: \text{Vec}(A, o) \end{aligned}$$

那么有:

$$\begin{aligned} \text{xs} \oplus (\text{ys} \oplus \text{zs}) &: \text{Vec}(A, n + (m + o)) \\ (\text{xs} \oplus \text{ys}) \oplus \text{zs} &: \text{Vec}(A, (n + m) + o) \end{aligned}$$

要想表达这两个表达式相等的命题比较困难, 因为两边类型不匹配. 使用依值相等可以得到一种非常优雅的表达方式:

$$\text{xs} \oplus (\text{ys} \oplus \text{zs}) =_{\text{ap}(\lambda n. \text{Vec}(A, n), +\text{-assoc})} (\text{xs} \oplus \text{ys}) \oplus \text{zs}$$

其中 $+\text{-assoc} : n + (m + o) = (n + m) + o$ 是一个定理. 容易看出

$$\text{ap}(\lambda n. \text{Vec}(A, n), +\text{-assoc}) : \text{Vec}(A, n + (m + o)) = \text{Vec}(A, (n + m) + o)$$

在进行这类证明时, 可以使用命题 3.2 的依值版本, 证明一致:

命题 3.1.1: 对于 $x, y : X$ 和 $f : (x : X) \rightarrow Y(x)$, 有 $(p : x =_X y) \rightarrow f(x) =_{\text{ap}(Y \circ f, p)} f(y)$.

可以看出, 原本的相等类型就是依值相等的特例. 从原本的相等到依值相等的推广类似于从函数类型到 Π 类型的推广, 是很自然的. 本讲义在这之后将不再区分两者.

立方类型论和同伦类型论有一个至关重要的区别就是后者并没有这样的异质相等, 它顶多有 Σ 类型的相等, 和异质相等虽然能做的事情很接近却有着本质的区别.

例子 3.1.1: 考虑 $a, b, c, d : \bullet =_A \bullet$ 四个相等, 其中 $\bullet : A$ 是一个无关紧要的常量. 乍看之下我们只能描述 $a =_{\bullet} b$ 这样的相等, 但除此之外我们还能写出如下类型:

$$a =_{\lambda i. c @ i = d @ i} b$$

检查一下也没有问题: 令 $t \equiv \lambda i. c @ i = d @ i$, 那么不管是 $t(0)$ 还是 $t(1)$ 都等于 $\bullet = \bullet$, 但这个函数偏偏就不是常函数, 它在 i 不为常量时有一个不同的值.

数学直觉好的读者可能会提前意识到这是个方块, 但看不出来也不用着急, 后面会介绍如何解读这样的类型.

3.2. 类型转换

最重要的是我们需要如下「类型转换」操作, 该操作本来是 J 规则的推论, 但是对于立方类型论的计算来说至关重要, 比 J 更基本:

$$\frac{A : X =_{\mathcal{U}} Y}{\text{coe}_A : X \rightarrow Y}$$

也可写作:

$$\frac{A : \mathbb{I} \rightarrow \mathcal{U}}{\text{coe}_A : A(0) \rightarrow A(1)}$$

这两个版本的定义是等价的, 但是语法是不一样的, 后文将倾向于使用后者, 即会使用

$$\text{coe}_{\lambda x. -\text{坨}}(u)$$

而不是

$$\text{coe}_{\langle x \rangle -\text{坨}}(u)$$

注意我们现在还没给出这个函数的实现, 我们只是在畅想有了它之后的美好未来. 如果有了它, 可以证明相等类型的传递性和对称性.

命题 3.2.1: 对于 $x, y, z : X$ 和 $p : y =_X z$, 有 $x =_X y \rightarrow x =_X z$.

证明: 根据 p 的类型, 我们知道 y 和 z 命题等价. 由命题 3.2 可知, $x =_X y$ 和 $x =_X z$ 这两个类型命题等价 (令 $f \equiv \lambda o. x =_X o$), 再借助转换操作将前者转换成后者即可.

因此可以得到如下证明: $\text{coe}_{\text{ap}(\lambda o. x =_X o, p)}$

□

命题 3.2.2: 对于 $x, y : X$ 和 $p : x =_X y$, 有 $y =_X x$.

证明: 根据 p 的类型, 我们知道 x 和 y 命题等价. 由命题 3.2 可知, $x =_X x$ 和 $y =_X x$ 这两个类型命题等价. 根据命题 3.1 可知, $x =_X x$ 是可证的, 再借助转换操作将这个转换成后者即可.

□

证明的具体构造留作练习, 难度非常低.

类型转换的操作目前看来还是公理, 因为我们没给它的具体实现. 事实上, 对于某些类型, 它的实现还是很简单的:

例子 3.2.1:

- 积类型:

$$\text{coe}_{\lambda x. A \times B}(u) \equiv \langle \text{coe}_A(u. 1), \text{coe}_B(u. 2) \rangle$$

- 函数类型需要一种反转的操作, 即 $\text{coe}_A^{-1} : A(1) \rightarrow A(0)$, 这在大多数具体实现中都是存在的, 参见下文关于 Descartes 立方类型论的描述.

例子 3.2.2:

- 没有参数的归纳类型 D :

$$\text{coe}_{\lambda x.D}(u) \equiv u$$

虽然还没介绍但是这对于高阶归纳类型也是一样.

- 对于宇宙 \mathcal{U} , 行为和 D 一致.

使用 J 规则实现的 coe 操作还满足如下计算规则:

$$\text{coe}_{\text{idp}}(u) \equiv u$$

我们这个版本的 coe 能不能满足这个等式还是个未知数. 事实上, 「构造一个立方类型论使得该等式成立」是个开放问题. 该等式又叫做正则性. 但这个等式的命题形式一般是成立的. 这也是在给出 coe 定义时需要时刻注意的一点: 必须确保能构造出一条路径

$$p : \text{coe}_{\text{idp}}(u) = u$$

这也在定义 Π, Σ 等依值类型的 coe 实现时必须的.

一个简单的例子是 Descartes 立方类型论 (这是立方类型论的一个版本, 还有很多不同的其它版本) 中的 coe 操作, 类型如下:

$$\frac{A : \mathbb{I} \rightarrow \mathcal{U} \quad r, s : \mathbb{I}}{\text{coe}_A^{r \rightsquigarrow s} : A(r) \rightarrow A(s)}$$

换言之它将原本的 0 和 1 端点都改成了参数. 它满足如下计算规则 (这也是一类似正则性的性质):

$$\text{coe}_A^{i \rightsquigarrow i}(u) \equiv u$$

这样的话便有如下证明:

$$\langle i \rangle \text{coe}_A^{i \rightsquigarrow 1}(u) : \text{coe}_A^{0 \rightsquigarrow 1}(u) =_A u$$

另外, 这个设计也非常巧妙地解决了 coe^{-1} 的定义, 只需交换两参数即可.

4. 面映射

本节给出一些技术性的定义, 可能有些枯燥, 推荐带着丰富的空间想象力阅读. 本来应该多配点图的, 但是 Typst 缺少相关工具链. 如果实在是看有图的需求, 可以看作者的英语教程 [3].

立方类型论中, $i : \mathbb{I} \vdash u : X$ 对应一个 X 中的一维路径, 可以认为 $i : \mathbb{I}$ 是它的「坐标轴», 而在语境中再增加一个变量 $j : \mathbb{I}$ 对应了增加了一个维度的操作, 比如 $i, j : \mathbb{I} \vdash u$ 就是一个二维的方块, 我们可以取它的边, 比如 $u[i \mapsto 0]$ 就是它在 i 坐标轴的左边, 由于 u 是二维图形, 它的一个边就是一维的路径, 而很显然 $u[i \mapsto 0]$ 中不再具有变量 i , 只有 j , 符合一维的条件.

类似地, 若 j 坐标轴是从下往上的话, 那么 $u[i \mapsto 0][j \mapsto 0]$ 就是它的左下角. 为了描述这种「取出一个面」的操作, 我们引入类型 $\mathbb{F} : \mathcal{U}$, 构造规则如下:

$$\frac{\top : \mathbb{F} \quad \perp : \mathbb{F} \quad \varphi : \mathbb{F} \quad \psi : \mathbb{F}}{\varphi \wedge \psi : \mathbb{F} \quad \varphi \vee \psi : \mathbb{F}}$$

$$\frac{i : \mathbb{I}}{i = 0 : \mathbb{F} \quad i = 1 : \mathbb{F}}$$

注意这里面的 $= 0$ 是被视为关键字的, 这个 $=$ 符号不是一个二元运算符, 应该认为 $= 0$ 是一个整体, 不要和相等类型搞混了. 其中:

- $i = 0, i = 1$ 表示在 i 变量对应的坐标轴中取出两 endpoints,
- $\varphi \wedge \psi$ 表示取出 φ 和 ψ 的交集,
- $\varphi \vee \psi$ 表示取出 φ 和 ψ 的并集,
- \perp 表示取出空集, 一般是在遇到想要取 $i = 0 \wedge i = 1$ 的时候会变成这种情况, 不会有直接取空的需求,
- \top 表示取出全集, 也就是啥也不干.

我们称 \mathbb{F} 类型的实例为「语境限制」. 在某些文献中它也会被称为「余纤维化」, 但是这个名字本身还有一个更广义的数学中的含义, 因此我们避免使用它. 在部分立方类型论的变种里, 还有更丰富的表达式, 比如 $i = j$ 这种描述两个维度的对角线的操作, 就是前面说的 Descartes 立方类型论里面的至关重要的一环.

我们可以看出, \mathbb{F} 类型像是一种命题宇宙, 它的实例对应了一些命题, 而且对这些命题我们可以取合取, 取析取, 还有真命题和假命题等等. 在范畴语义中, \mathbb{F} 对应的是子对象分类子的一个子对象, 也很好印证了这一点.

立方类型论的类型规则中, 对于 $\varphi : \mathbb{F}$, 可以得到一个对语境的函子, 为了方便, 直接将「对于语境 Γ 应用这个函子」记作 Γ, φ , 于是可以写出如下表达式:

$$\Gamma, \varphi \vdash \mathcal{J}$$

其中 $\varphi : \mathbb{F}$. 我们规定:

- Γ, \top 和 Γ 相等,
- Γ, \perp 语境下任何等式成立,
- $\Gamma, \psi \wedge \varphi$ 和 Γ, ψ, φ 相等.

然后再规定, 对于 $r \in \{0, 1\}$:

$$\frac{\Gamma \vdash \mathcal{J}[i \mapsto r]}{\Gamma, i = r \vdash \mathcal{J}}$$

对于 $\varphi, \psi : \mathbb{F}$, 有:

$$\frac{\Gamma, \varphi \vdash \mathcal{J}}{\Gamma, \varphi \vee \psi \vdash \mathcal{J}} \quad \frac{\Gamma, \psi \vdash \mathcal{J}}{\Gamma, \varphi \vee \psi \vdash \mathcal{J}}$$

例子 4.1: 对于 $r, r' \in \{0, 1\}$, 有

$$\frac{\Gamma \vdash \mathcal{J}[i \mapsto r, j \mapsto r']}{\Gamma, i = r \wedge j = r' \vdash \mathcal{J}}$$

4.1. 立方子类型

我们引入如下类型来描述某类型「满足某些等式」的「子类型」, 注意这不是子类型多态意义上的子类型, 我们要求显式的转换:

$$\frac{A : \mathcal{U} \quad \varphi \vdash u : A}{\{A \mid \varphi \mapsto u\} : \mathcal{U}}$$

这里实际上是简写, 这个大括号里面还可以写多个这样的式子, 比如

$$\{A \mid \varphi \mapsto u, \psi \mapsto v\}$$

也是可以的. 这样的类型我们称之为立方子类型, 用于和一般的子类型区分. 它的构造规则, 便是要求实例满足这个条件, 然后转换过来:

$$\frac{v : A \quad \varphi \vdash u \equiv v : A}{\text{inS}(v) : \{A \mid \varphi \mapsto u\}}$$

如果有多条规则, 就是要求满足所有的条件. 消去规则就是取出这个实例, 且会根据条件化简:

$$\frac{v : \{A \mid \varphi \mapsto u\}}{\text{outS}(v) : A \quad \varphi \vdash u \equiv \text{outS}(v) : A}$$

构造规则和消去规则都是一元的, 满足显然的 β, η 规则, 不再赘述. 我们接下来将会假设我们的代码中会处处隐式地插入 inS 和 outS 的调用来满足类型规则, 即「写作子类型, 读作自动插入转换」.

立方子类型可以嵌套:

$$\{\{A \mid \varphi \mapsto u\} \mid \psi \mapsto v\}$$

在有了隐式转换后, 这和 $\{A \mid \varphi \mapsto u, \psi \mapsto v\}$ 没有本质上的区别.

可以看出, (依值)相等类型的类型规则可以用子类型表达:

$$x =_A y \coloneqq (i : \mathbb{I}) \rightarrow \{A(i) \mid i = 0 \mapsto x, i = 1 \mapsto y\}$$

然后它的构造规则和消去规则和计算规则都会自动继承自子类型的规则. 此时我们已经有了充分的工具解读例子 3.1.1 了, 首先将该类型转化为立方子类型的描述:

$$(i : \mathbb{I}) \rightarrow \{(c @ i =_A d @ i) \mid i = 0 \mapsto a, i = 1 \mapsto b\}$$

我们发现这还能再展开一层:

$$(i : \mathbb{I}) \rightarrow \left\{ \underbrace{(j : \mathbb{I}) \rightarrow \{A \mid j = 0 \mapsto c @ i, j = 1 \mapsto d @ i\}} \mid i = 0 \mapsto a, i = 1 \mapsto b \right\}$$

调整一下内外两层子类型, 会变成这样 (证明这两者在隐式转换下等价留作练习):

$$(i, j : \mathbb{I}) \rightarrow \left\{ A \mid \underbrace{j = 0 \mapsto c @ i}, \underbrace{j = 1 \mapsto d @ i}, \underbrace{i = 0 \mapsto a @ j}, \underbrace{i = 1 \mapsto b @ j} \right\}$$

可以看出, 这是一个二维的相等, 或者说是一个方块, 有上下左右四条边, i 和 j 分别是两个坐标轴. 换言之, 借助子类型的视角, 我们能看出依值相等类型能表达这种高维图形.

前面说的 Descartes 立方类型论的 coe 操作也可以用这种方式一次性描述它的类型和化简:

$$\text{coe}_A^{r \rightsquigarrow s} : (u : A(r)) \rightarrow \{A(s) \mid r = s \mapsto u\}$$

这实在是太方便了!

5. 双层类型论

到现在为止我们基本完成了立方类型论的描述, 看起来这是个振奋人心的类型系统, 拥有丰富的功能, 但是还有一些细节没有说清楚, 那就是关于 coe 这个操作.

这个操作的本质是什么? 它是一个函数吗? 乍看之下它是一个内部对每种类型分别给出实现的函数, 那么它对于相等类型是如何实现的? 换言之, 相等的传递性证明化简到典范形式后是什么样的? 它对 \mathbb{I}, \mathbb{F} 还有立方子类型这些类型也有实现吗?

答案是否定的. 事实上, coe 是形成规则上带有的结构 (结构就是能参与计算的性质), 而且不是所有形成规则都有这个性质 — 「具有 coe 结构」的性质叫做「纤性」, 而 \mathbb{I}, \mathbb{F} 还有立方子类型都不是纤类型, 这样的类型又叫做「外类型」.

为了区分外类型和纤类型, 我们将宇宙一分为二, 变成纤类型宇宙 \mathcal{U}_1 和外类型宇宙 \mathcal{U}_0 . 这个符号借鉴了 [4]. 于是我们可以将 coe 的类型写作:

$$\frac{A : \mathbb{I} \rightarrow \mathcal{U}_1}{\text{coe}_A : A(0) \rightarrow A(1)}$$

这样的话 $\text{coe}_{\lambda x. \{A \mid \dots\}}$ 就直接类型不合法了, 这是好的. 立方子类型本质上是用来描述纤类型的子类型的, 因此改为

$$\frac{A : \mathcal{U}_1 \quad \varphi \vdash u : A}{\{A \mid \varphi \vdash u\} : \mathcal{U}_0}$$

剩余的 \mathbb{I}, \mathbb{F} 直接扔进 \mathcal{U}_0 , 不再赘述. 对于 Π, Σ 类型, 规定它们在这两个宇宙中各自闭合. 这样的话, 从两个宇宙之间游走的方式就很有限了, 比如使用立方子类型的规则等.

值得注意的是, 相等类型是纤类型, 但是把它解构成立方子类型的表示后, 就不再是纤类型了, 因此这里需要一个 Tarski 宇宙的技巧来解决这个问题. 这个事情不是很重要, 因为我们完全可以用立方子类型来代替相等类型.

另外, 这个配置下会导致 $\mathbb{I} \rightarrow A$ 这个类型在 $A : \mathcal{U}_1$ 时不合法, 因为 \mathbb{I} 和 A 不一样, 是外类型. 但实际上我们是可以赋予 $\mathbb{I} \rightarrow A$ 纤类型的结构的, 把它当成平凡的相等类型就可以了. 一种优雅解决方法是将 \mathbb{I} 隔离进第三个宇宙 \mathcal{U}_I , 令 $\mathcal{U}_I : \mathcal{U}_0$, 然后再允许

$$\frac{i : \mathbb{I} \vdash A : \mathcal{U}_1}{(i : \mathbb{I}) \rightarrow A : \mathcal{U}_1}$$

即可. 这应该是 Evan Cavallo 想的, 参见².

宇宙分两层, 允许有限交互的类型论叫双层类型论. 关于双层类型论还有很多细节和延伸话题, 这里不多赘述.

6. 补齐操作

我们来看看相等类型的转换操作到底需要构造个啥出来, 最好的方法就是把需要构造的表达式先写出来, 然后再查成分. 写出来是这样:

$$\text{coe}_{\lambda x. a =_A b} : (a =_A b)[x \mapsto 0] \rightarrow (a =_A b)[x \mapsto 1]$$

其中需要如下数据 (就是依值相等类型的基本要素):

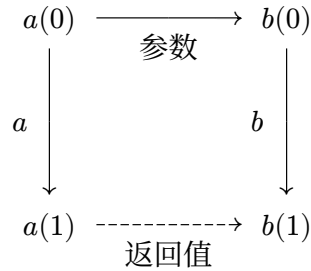
- $x : \mathbb{I} \vdash A : \mathbb{I} \rightarrow \mathcal{U}_1$,
- $x : \mathbb{I} \vdash a : A(0)$,
- $x : \mathbb{I} \vdash b : A(1)$.

显而易见的, A 现在是一个二维的类型! 把 x 参数化后, 可以这样看:

²<https://github.com/agda/agda/pull/5439>

- $A : (x, y : \mathbb{I}) \rightarrow \mathcal{U}_1$ 是一个方块,
- $a : (x : \mathbb{I}) \rightarrow A(x, 0)$,
- $b : (x : \mathbb{I}) \rightarrow A(x, 1)$,
- coe 操作的参数的类型为 $a(0) =_{\lambda y. A(0, y)} b(0)$,
- coe 操作的返回值的类型为 $a(1) =_{\lambda y. A(1, y)} b(1)$.

注意这里出现了 $A(x, 0), A(x, 1), A(0, y), A(1, y)$ 这样的类型, 这分明就是 A 这个正方形的四条边! 于是这些数据可以被画在图里:



我们需要的操作是: 给定如上的方块的三条边, 构造出第四条边. 这个操作可以被推广到高维: 给定 n 维立方体的全部 $n - 1$ 维边中去掉一个, 构造出那个被去掉的边.

一般来说, 在「参数」那个位置的边叫做「底面」, 尽管这里它是被画在上面的. 这是因为补齐操作一般被描述为「盖上盖子」, 而能被盖上盖子的东西一般开口是朝上的. 而画图的时候, 由于人阅读的顺序一般是从上往下的, 因此本文选择将参数放在上面.

例子 6.1:

- 令 $a \equiv \text{id}_p$, 那么这个补齐操作就对应了相等类型的传递性.
- 令 $b \equiv \text{参数} \equiv \text{id}_p$, 那么这个补齐操作就对应了相等类型的对称性.

立方类型论将这种操作的一种简化形式作为纤类型的定义的一部分引入类型论, 然后结合 coe 把以上操作模拟出来. 以上需要的操作又叫做异质补齐, 简化的操作叫做补齐或者同质补齐.

形式化地定义同质补齐需要有一个办法来描述上面那个方块的三条边, 然后将这个描述本身作为一个 A 类型的实例引入即可. 这需要引入一种新的外类型来描述, 叫做「不完整类型」.

6.1. 不完整类型

不完整类型的形成规则如下:

$$\frac{A : \mathcal{U}_1 \quad \varphi : \mathbb{F}}{\text{Partial}(\varphi, A) : \mathcal{U}_0}$$

构造规则稍微复杂, 如下所示:

$$\frac{\varphi_0 \vdash u_0 : A \quad \varphi_1 \vdash u_1 : A \quad \dots}{\varphi_0 \wedge \varphi_1 \vdash u_0 =_A u_1 \quad \dots}$$

$$\frac{\{\varphi_0 \mapsto u_0, \varphi_1 \mapsto u_1, \dots\} : \text{Partial}(\varphi_0 \vee \varphi_1 \vee \dots, A)}{\dots}$$

构造规则中,

- $\varphi_0 \vdash u_0 : A$ 这样的表达式就是一条边, 维度任意, 这是第一行想表达的信息,

- 此外我们需要需要确保任何两条边在相交处都是相等的, 这是第二行表达的信息,
- 若两条边不相交, 它们对应的 $\varphi_i \wedge \varphi_j$ 等价于 \perp , 而我们知道在这个限制下任何等式都成立, 因此不相交的边之间不需要满足任何等式即合法,
- 若给定两条完全重合的边, 即 $\varphi_i \equiv \varphi_j$, 那么它们必定得是无条件相等的.

有了不完整类型, 我们就可以定义同质补齐操作的类型了:

$$\frac{A : \mathcal{U}_1 \quad \varphi : \mathbb{F} \quad u : (i : \mathbb{I}) \rightarrow \text{Partial}(\varphi \vee i = 0, A)}{\text{hcom}(u) : \{A \mid \varphi \mapsto u(1)\}}$$

异质补齐就是一个简单的依值类型推广:

$$\frac{A : \mathbb{I} \rightarrow \mathcal{U}_1 \quad \varphi : \mathbb{F} \quad u : (i : \mathbb{I}) \rightarrow \text{Partial}(\varphi \vee i = 0, A(i))}{\text{com}(u) : \{A(1) \mid \varphi \mapsto u(1)\}}$$

用 $\text{hcom} + \text{coe}$ 来定义 com 的具体构造就省略了.

从补齐的类型中可以看出, 它有如下计算规则:

$$\text{hcom}(\lambda x. \{\top \mapsto m\}) \equiv m$$

换言之, 令 $\varphi \equiv \top$, 那么补齐就可以被消除. 对应的几何描述是: 若我需要补齐的立方体的全部侧面都由同一个值指定, 那么补齐出来的结果就是这个值.

在 Descartes 立方类型论中, 补齐也有类似 coe 的推广, 即从 0 到 1 改成从 r 到 s , 其中 $r, s : \mathbb{I}$ 是任意表达式:

$$\frac{A : \mathcal{U}_1 \quad \varphi : \mathbb{F} \quad r, s : \mathbb{I} \quad u : (i : \mathbb{I}) \rightarrow \text{Partial}(\varphi \vee i = r, A)}{\text{hcom}^{r \rightsquigarrow s}(u) : \{A \mid \varphi \vee r = s \mapsto u(s)\}}$$

异质补齐如下:

$$\frac{A : \mathbb{I} \rightarrow \mathcal{U}_1 \quad \varphi : \mathbb{F} \quad r, s : \mathbb{I} \quad u : (i : \mathbb{I}) \rightarrow \text{Partial}(\varphi \vee i = r, A(i))}{\text{com}^{r \rightsquigarrow s}(u) : \{A(s) \mid \varphi \vee r = s \mapsto u(s)\}}$$

注意这里有一个非常重要的事情: 补齐对它侧面的数量是没有限制, 即它接受任意的 $\varphi : \mathbb{F}$. 换言之, 令 $\varphi \equiv \perp$, 我们就不需要给出任何侧面, 也能进行合法的补齐. 这种补齐被称为「空补齐」.

6.2. 补齐的计算

除了类型里描述的规则之外, 补齐的常规计算规则 (也就是取决于具体类型的计算) 大体类似 coe , 不多赘述, 但是对于归纳类型的规则是不同的. 比如考虑自然数 \mathbb{N} , 我们有:

$$\begin{aligned} \text{hcom}(\lambda x. \{\varphi_i \mapsto \text{zero}\}) &\equiv \text{zero} \\ \text{hcom}(\lambda x. \{\varphi_i \mapsto \text{suc}(u_i)\}) &\equiv \text{suc}(\text{hcom}(\lambda x. \{\varphi_i \mapsto u_i\})) \end{aligned}$$

这要求我们在化简补齐的时候, 能进行「若每个给定的面都是由同一个构造子生成」这样的判断, 这是比较难实现的, 效率也比较差, 但却是典范性关键的一环.

我们也可以直接摆烂, 归纳类型的 hcom 就当成典范形式了, 这样的归纳类型叫弱归纳类型, 比如 `cooltt` 编程语言就是这样做的. 它给所有的归纳类型都新增了 hcom 这种形式的典范形式.

这些常规类型上的计算规则使得空补齐在类型已知的情况下基本能被化简为底面, 唯一的例外是宇宙 \mathcal{U}_1 . 如果宇宙上也能进行这样的化简, 那么这个宇宙就是「正则的」, 有正则宇宙的话空补齐就可以不管类型直接化简了, 这看起来很好看.

可惜正则性是个非常强的性质, 在 [5] 中, Carlo Angiuli 给出了一个由该化简规则蕴涵的一条非常难以算法化判定的类型论等式. 因此, 在不知道类型的时候, 空补齐目前无法被化简, 导致它成了立方类型论使用者深恶痛绝的对象³, 因为它们会在计算过程中永远地存在下去, 越堆越多, 无法消除, 就像是一种「内存泄漏」.

在一些立方类型论的文献中 (例如 [6]), 典范的 hcom 记作 fhcom , 这便于区分待计算的 hcom . 在下面我们会看到, 高阶归纳类型上的 hcom 几乎没有任何办法化简, 因此必须全部当成典范形式. 归纳类型的补齐的计算有一种巧妙的实现方式, 由 András Kovács 提出. 观察归纳类型和补齐的以下几个性质:

- 补齐必定要求提供底面, 即 $i = 0$ 的面 (在 Descartes 版本中则是需要提供 $i = r$ 的面),
- 由于不完整类型的构造规则, 整个立方体的相交部分必须重合,
- 一般的归纳类型里, 不同的构造子是互斥的 (Conor McBride 的「无混淆原则」).

因此若底面是由某个特定的构造子生成的, 假设形式为 $\text{con}(u_1, u_2, \dots)$, 那么侧面也都一定是由这个构造子生成的, 只不过表达式层面上不一定看得出来. 因此我们可以引入辅助运算 uncon , 它满足如下计算规则:

$$\text{uncon}(\text{con}(u_1, u_2, \dots)) \equiv \langle u_1, u_2, \dots \rangle$$

然后可以这样计算:

$$\begin{aligned} & \text{hcom}(\lambda x. \{x = 0 \mapsto \text{con}(u_1, u_2, \dots), \varphi_1 \mapsto v_1, \varphi_2 \mapsto v_2, \dots\}) \equiv \\ & \quad \text{let } a_1 \equiv \text{hcom}(\lambda x. \{x = 0 \mapsto u_1, \varphi_1 \mapsto \text{uncon}(v_1).1, \varphi_2 \mapsto \text{uncon}(v_2).1, \dots\}) \\ & \quad a_2 \equiv \text{hcom}(\lambda x. \{x = 0 \mapsto u_2, \varphi_1 \mapsto \text{uncon}(v_1).2, \varphi_2 \mapsto \text{uncon}(v_2).2, \dots\}) \\ & \quad a_3 \equiv \dots \\ & \quad \text{in } \text{con}(a_1, a_2, \dots) \end{aligned}$$

这个计算规则看起来很恐怖, 但是写在编译器里面是很直观的, 整个流程用一个循环里面新建语法树节点就可以实现, 而不需要对每个侧面进行形式的判断, 而且如果侧面在没有用到的话, 也不会触发计算, 因此性能上是很不错的.

但补齐的计算又带来了新的问题: 宇宙上的补齐怎么办? 我们直接给 \mathcal{U} 这个类型加了新的实例, 这意味着新的形成规则 (记作 Hcom), 因此我们还需要给它描述构造规则和消去规则. 这个事情还比较复杂, 有多种解决方法. 我们先来看看泛等公理的证明是怎么构造的.

7. 泛等公理

要定义泛等公理需要先给出等价 $A \simeq B$ 的定义. 我们把这个定义拆分成两部分:

$$A \simeq B \equiv (f : A \rightarrow B) \times \text{isEquiv}(f)$$

然后我们需要定义 $\text{isEquiv}(f)$. 这个就是同伦类型论的内容了, 读者应该已经熟悉, 因此我们跳过动机直接给出一个比较经典的版本 (可缩纤维等价):

$$\begin{aligned} \text{isContr}(A) & \equiv (a : A) \times ((b : A) \rightarrow a =_A b) \\ \text{fiber}(f, b) & \equiv (a : A) \times f(a) =_B b \\ \text{isEquiv}(f) & \equiv (b : B) \rightarrow \text{isContr}(\text{fiber}(f, b)) \end{aligned}$$

除此之外还有 András Kovács 推荐的半伴随等价, 但这些不同的定义起到的作用都是一样的.

³参见 Favonia 的讲座 <https://cse.umn.edu/cs/feature-stories/nullable-compositions-talk>

以下是几个读者不需要知道证明细节的引理, 帮助具有一定类型论背景的读者熟悉等价的性质:

引理 7.1 (研究生引理): 若 A 和 B 同构, 即存在 $f : A \rightarrow B, g : B \rightarrow A$ 以及

$$\begin{aligned} p : (x : A) \rightarrow g(f(x)) =_A x \\ q : (x : B) \rightarrow f(g(x)) =_B x \end{aligned}$$

那么 $\text{isEquiv}(f)$ 成立, 换言之 $A \simeq B$ 成立.⁴

引理 7.2: 给定 $p : A =_{\mathcal{U}_1} B$, 可证明 $\text{isEquiv}(\text{coe}_p)$. 记

$$\text{coe}_{\simeq} : A = B \rightarrow \text{isEquiv}(\text{coe}_p)$$

引理 7.3: 对于类型 $A : \mathcal{U}_1$, 有 $A \simeq A$. 该证明记作 idEquiv .

接下来就是正式的证明了. 证明泛等公理可以分为两步:

1. 构造出 $\text{fromEquiv} : A \simeq B \rightarrow A =_{\mathcal{U}_1} B$ 这个函数,
2. 证明该函数是 coe_{\simeq} 的逆.

第一步最具有类型论上的挑战性, 我们详细叙述. 第二步我们只关注它的一个简单推论, 剩下的都是技术细节:

命题 7.1: 对于 $e : A \simeq B$ (这意味着 $e.1 : A \rightarrow B$ 是它对应的函数),

$$\text{coe}_{\text{fromEquiv}(e)} =_{A \rightarrow B} e.1$$

根据函数外延性, 这也等价于

$$(a : A) \rightarrow \text{coe}_{\text{fromEquiv}(e)}(a) =_B e.1(a)$$

7.1. 胶合类型的形成规则

胶合类型是立方类型论中的核心构造, 身兼数职:

1. 它本身的构造能证明泛等公理的第一步, 其余规则证明了第二步,
2. 担任宇宙上的补齐操作 Hcom .

它作为一种类型, 形成规则如下:

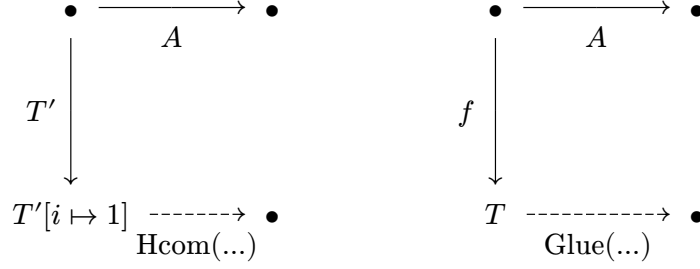
$$\frac{A : \mathcal{U}_1 \quad \varphi : \mathbb{F} \quad \varphi \vdash e : (T : \mathcal{U}_1) \times (T \simeq A)}{\text{Glue}(A)\{\varphi \mapsto e\} : \mathcal{U}_1}$$

⁴参见 <https://amelia.how/posts/the-complete-history-of-isotoequiv.html>, 对这个引理的名字给出了较为完整的考据. 这里的研究生可能是 Daniel Licata 和 Peter LeFanu Lumsdaine.

注意这里和补齐操作类似, 也是允许多个侧面的, 但为了写法简便, 就只写了一个. 方便起见, 记 $T \coloneqq e.1$. 作为对比, 我们用一种类似的结构写出宇宙上的补齐规则:

$$\frac{A : \mathcal{U}_1 \quad \varphi : \mathbb{F} \quad \varphi, i : \mathbb{I} \vdash T' : \mathcal{U}_1 \quad \varphi, i = 0 \vdash T' \equiv A}{\text{Hcom}(\lambda i. \{\varphi \mapsto T', i = 0 \mapsto A\}) : \mathcal{U}_1}$$

可以看出, 我们把补齐方块的两个侧面从「和底面在相交处兼容的一条道路 $i : \mathbb{I} \vdash T'$ 」换成了「和底面等价的类型 T 」。为了视觉上的直观, 我们把这两者在常见的情况的图画出来. 令横坐标轴为 j , 且 $\varphi \coloneqq j = 0$ (即忽略横坐标的右边, 不然画出来看不清楚):



注意左边的竖线是道路, 但右边的是等价.

定理 7.1.1: 给定 $A, B : \mathcal{U}_1$ 和 $e : A \simeq B$, 存在 $A = B$.

证明: 由引理 7.3 可得 $\text{idEquiv} : B \simeq B$. 于是可构造证明:

$$\langle i \rangle \text{Glue}(A) \{i = 0 \mapsto \langle A, e \rangle, i = 1 \mapsto \langle B, \text{idEquiv} \rangle\}$$

泛等公理第一步大功告成! □

接下来那些非平凡的类型上的相等就可以证明了, 这里给出一些简单的例子:

命题 7.1.1: 如下命题成立:

- $\mathbb{N} = \mathbb{Z}$ 以及 $\mathbb{N} \times \mathbb{N} = \mathbb{N}$
- $A \times B = B \times A$ 以及 $A \times (B \times C) = (A \times B) \times C$
- $(\text{Bool} \rightarrow A) = A \times A$ 以及 $(\text{Bool} \times A) = A + A$
- $\text{List}(A) = (\mathbb{N} \rightarrow A)$

7.2. 胶合类型其余的规则

我们可以把宇宙补齐 Hcom 「代理」到胶合类型上. 假设存在如下隐式转换:

$$\text{Partial}(\top, A) \rightarrow A$$

它取出「实际上完整的不完整类型」中的元素, 那么有

$$\frac{X : (i : \mathbb{I}) \rightarrow \text{Partial}(\varphi \vee i = 0, \mathcal{U}_1)}{\text{Hcom}(X) \coloneqq \text{Glue}(X(0)) \{\varphi \mapsto \langle X(1), \text{coe}_{\simeq}(\dots) \rangle\}}$$

其中 $\text{coe}_{\simeq}(\dots)$ 省略, 起到的作用是从 $\varphi \vdash X : (i : \mathbb{I}) \rightarrow \mathcal{U}_1$ 中得到 $X(0) \simeq X(1)$ 的证明.

至此, 补齐操作的最后一块拼图也完成了, 但我们还得定义胶合类型上的补齐操作和转换规则, 但好在这些步骤不再需要新的构造了.

胶合类型本身有类似补齐的计算规则:

$$\text{Glue}(A)\{\top \mapsto \langle T, e \rangle\} \equiv T$$

换言之, $\varphi \vdash \text{Glue}(A)\{\varphi \mapsto \langle T, e \rangle\} \equiv T$. 也可以使用子类型表示:

$$\text{Glue}(A)\{\varphi \mapsto e\} : \{\mathcal{U}_1 \mid \varphi \mapsto T\}$$

作为一种形成规则, 胶合类型也有配套的构造规则和消去规则. 构造规则虽然长, 但实际上很直观:

$$\frac{\varphi \vdash e : T \simeq A \quad \varphi \vdash t : T \quad u : \{A \mid \varphi \mapsto e.1(t)\}}{\text{glue}(u)\{\varphi \mapsto t\} : \text{Glue}(A)\{\varphi \mapsto \langle T, e \rangle\}}$$

它也有类似补齐的计算规则:

$$\text{glue}(u)\{\top \mapsto t\} \equiv t$$

这也可以用子类型表示:

$$\text{glue}(u)\{\varphi \mapsto t\} : \{\text{Glue}(A)\{\varphi \mapsto \langle T, e \rangle\} \mid \varphi \mapsto t\}$$

消去规则则依存于类似补齐的计算规则, 因为 φ 下有 $u : T$ 成立:

$$\frac{u : \text{Glue}(A)\{\varphi \mapsto \langle T, e \rangle\}}{\text{unglue}(u) : \{A \mid \varphi \mapsto e.1(u)\}}$$

至于 β, η 规则比较直观, 就姑且省略了.

通过这些构造, 可以定义胶合类型上的补齐操作和转换规则. 这两者本质上就是调用胶合类型的构造规则 `glue`, 然后传入适当的参数.

由于具体的构造步骤非常繁琐, 列出了可能会让读者头晕, 故省略. 对自己有信心或者想满足自己好奇心的读者可以参见 [7] 的 3.6 和 4.6 两节. 坚称自己是天选之人的读者可以考虑将这个构造作为练习.

7.3. 独立的宇宙补齐

借助胶合类型实现宇宙补齐有比较严重的性能问题, 因为相等在代码中的表示是「含有一个维度变量的值」, 这比起「一个转换函数配套它纤维可缩的证明」要小得多. 因此, 为了性能考量, 我们可以直接实现宇宙补齐, 而不是通过胶合类型来代理.

这样的话, 需要再额外引入一对构造规则和消去规则, 以及 β, η 规则. 在 [5] 的附录和 [8] 中便是这样的风格. 这里罗列一下规则, 形成规则如下:

$$\frac{\varphi : \mathbb{F} \quad A : (i : \mathbb{I}) \rightarrow \text{Partial}(\varphi \vee i = 0, \mathcal{U}_1)}{\text{Hcom}(A) : \{\mathcal{U}_1 \mid \varphi \mapsto A(1)\}}$$

于是对宇宙的补齐可以直接计算到这个形成规则:

$$\text{hcom}(A) ::= \text{Hcom}(A)$$

构造规则如下, 其实很类似 `glue`:

$$\frac{n : A(1) \quad m : \{A(0) \mid \varphi \mapsto \text{coe}_A^{-1}(n)\}}{\text{box}(m)\{\varphi \mapsto n\} : \text{Hcom}(A)}$$

注意这条规则中, 虽然 A 的值域是一个不完整类型, 但在 φ 下它是完整的, 因此我们可以在此插入一个隐式转换, 并使用它的转换规则. 消去规则如下所示, 也用到了同一个隐式转换, 形式上也很类似 `unglue`:

$$\frac{u : \text{Hcom}(A)}{\text{cap}(u) : \{A(0) \mid \varphi \mapsto \text{coe}_A^{-1}(u)\}}$$

计算规则和胶合类型一致. 至于它上面的补齐操作和转换规则, 请读者参阅 [8] 的第 3.2 和 3.4 两节, 本质上也都是调用构造规则, 然后传入适当的参数. 注意本讲义中讨论的 Hcom 是从 0 到 1 的, 而 [8] 讨论的是 Descartes 立方类型论, 将 0 和 1 推广为参数 r 和 s .

7.4. V 类型

既然宇宙的补齐有单独的实现了, 胶合类型就不需要做成方形的了 – 用胶合类型证明泛等公理时, 其中一边必定是平凡的等价, 那么直接删除这条边, 使用一个三角形就足够证明泛等公理了. 这个思路构造的类型叫 V 类型, 形成规则如下:

$$\frac{r : \mathbb{I} \quad r = 0 \vdash A : \mathcal{U}_1 \quad B : \mathcal{U}_1 \quad r = 0 \vdash e : A \simeq B}{V^r(e) : \{\mathcal{U}_1 \mid r = 0 \mapsto A, r = 1 \mapsto B\}}$$

很明显, 通过 $e : A \simeq B$ 生成 $\langle r \rangle V^r(A, B, e) : A = B$ 的过程就已经达成了证明泛等公理的第一步. 构造规则如下:

$$\frac{u : A \quad v : \{B \mid r = 0 \mapsto e.1(u)\}}{\text{Vin}^r(v)\{u\} : \{V^r(e) \mid r = 0 \mapsto u, r = 1 \mapsto v\}}$$

这里将 u 放在大括号里是为了和前面的 `glue` 中的 $\{\varphi \mapsto u\}$ 参数保持统一, 使得这两者的联系能一眼看出来. 消去规则如下:

$$\frac{u : V^r(e)}{\text{Vproj}^r(u) : \{B \mid r = 0 \mapsto e.1(u), r = 1 \mapsto u\}}$$

计算规则如下:

$$\text{Vproj}^r(\text{Vin}^r(v)\{\dots\}) \equiv v$$

其余规则篇幅所限, 不再罗列, 感兴趣的读者可以参阅 [8] 的第 3.1 和 3.3 两节. 值得一提的是, V 类型的转换规则和补齐操作也都是调用构造规则 `Vin`.

8. 高阶归纳类型

这里假定读者已经对高阶归纳类型有基本的概念, 即允许构造子的返回类型是归纳类型上的相等类型、这些相等类型上的相等类型、等等的归纳类型.

立方类型论对高阶归纳类型给出了一套相对比较统一的语法. 这里给出两个同伦型的例子和一个小学数学中的例子: 圆周 S^1 , 甜甜圈的表面 T^2 和对称的整数类型 \mathbb{Z} .

例子 8.1: 在同伦类型论中, S^1 由如下构造子生成:

$$\bullet : S^1 \quad \text{loop} : \bullet = \bullet$$

它的消去规则或者说模式匹配需要确保: \bullet 被映射到某个值 $x : X$ 、 loop 被映射到某个相等证明 $x =_X x$.

例子 8.2: 在同伦类型论中, 令相等类型的拼接操作为

$$\text{cat} : (x = y) \rightarrow (y = z) \rightarrow (x = z)$$

则 T^2 由如下构造子生成:

$$\bullet : T^2 \quad \text{l1, l2} : \bullet = \bullet$$

$$\text{surf} : \text{cat}(\text{l1}, \text{l2}) =_{\bullet} \text{cat}(\text{l2}, \text{l1})$$

它的消去规则或者说模式匹配需要确保: \bullet 被映射到某个值 $x : X$, $\text{l1}, \text{l2}$ 分别被映射到某个相等证明 $x =_X x$,

可以看出, 这些构造子的类型比较复杂, 相等类型可以多重嵌套, 因此要想设计一种通用的语法框架来定义它们是比较麻烦的.

例子 8.3: 对称的整数类型由如下构造子生成:

$$\frac{n : \mathbb{N}}{+n : \mathbb{Z} \quad -n : \mathbb{Z}} \quad \text{zeq} : +\text{zero} =_{\mathbb{Z}} -\text{zero}$$

它的消去规则或者说模式匹配需要确保: $+n$ 被映射到某个值 $f(n) : X$, $-n$ 被映射到某个值 $g(n) : X$, zeq 被映射到某个相等证明 $f(\text{zero}) =_X g(\text{zero})$.

这是一些常见的数学概念的高阶归纳类型定义, 除此之外还有在同伦类型论中非常重要的命题截断类型. 这也是一个非常重要的例子, 因为命题截断类型不仅是一个有参数的类型, 它的构造子还有递归的情况, 而立方类型论能正确处理它是一件很重要的事实. 令

$$\text{isProp}(A) ::= (x, y : A) \rightarrow (x =_A y)$$

为「类型 A 是一个命题」这一事实, 则定义命题截断类型如下:

例子 8.4: 对于类型 $A : \mathcal{U}_1$, 它的命题截断记作 $\|A\|$, 由如下构造子生成:

$$\frac{u : A}{\text{inj}(u) : \|A\|} \quad \text{trunc} : \text{isProp}(\|A\|)$$

8.1. 立方类型论中的高阶归纳类型

方便起见, 引入如下符号:

定义 8.1.1: 定义「边界」操作如下:

$$\begin{aligned}\partial : \mathbb{I} &\rightarrow \mathbb{F} \\ \partial(i) &::= i = 0 \vee i = 1\end{aligned}$$

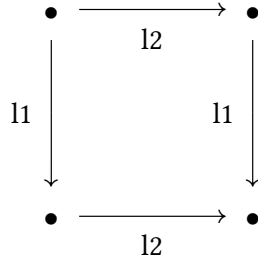
注意观察, 在立方类型论中, loop 的类型也可以看作

$$(i : \mathbb{I}) \rightarrow \{S^1 \mid \partial(i) \mapsto \bullet\}$$

而 surf 的类型更是可以使用更简洁的方法来表达:

$$(i, j : \mathbb{I}) \rightarrow \{T^2 \mid \partial(i) \mapsto l1 @ j, \partial(j) \mapsto l2 @ i\}$$

对应如下方块:



这种情况下, 我们只需额外允许构造子返回以下两种类中之一即可:

1. 它对应的高阶归纳类型 D ,
2. 某子类型 $\{D \mid \varphi \mapsto u\}$.

在模式匹配时, 若匹配到返回子类型的构造子, 则将子类型中的 u 应用到模式匹配中, 再检查兼容性即可. 这个说法有点抽象, 带入一个例子会容易许多: 考虑

$$\frac{m : X \quad n : (i : \mathbb{I}) \rightarrow X \quad u : S^1}{\text{elim}_{S^1}(u, m, n) : X}$$

这个规则要求 $n(0) \equiv n(1) \equiv m$, 而这里的 m 是由 $\partial(i) \vdash \text{elim}_{S^1}(u, m, n)$ 化简而来.

再考虑通用的情况, 令 D 为一个高阶归纳类型, $c : A \rightarrow D$ 为简单的构造子, $p : B \rightarrow \{D \mid \varphi \mapsto u\}$ 为高阶构造子. 函数

$$\begin{aligned}f : D &\rightarrow X \\ f(c(a)) &::= m \\ f(p(b)) &::= n\end{aligned}$$

中, 对于 p 构造子返回了 n , 因此它需要满足

$$\varphi \vdash f(u) \equiv n$$

而在 S^1 的例子中, $\varphi ::= \partial(i)$, 剩下的参数可以一一对应过来, 可以得到一样的「需要满足的等式」.

9. 趣味问题一: 取反操作

若我们引入如下运算符, 姑且称之为「取反」:

$$\neg : \mathbb{I} \rightarrow \mathbb{I}$$

并规定它满足如下等式:

$$\neg 1 \equiv 0 \quad \neg 0 \equiv 1 \quad \neg \neg i \equiv i$$

那么就会有一个新的方法证明相等类型的对称性:

命题 9.1: 若有 $p : x =_A y$, 则 $y =_A x$, 记作 p^{-1} .

证明: $\langle i \rangle p @ (\neg i)$

□

该证明性质比 Martin-Löf 类型论中的证明性质更强, 因为它满足如下等式:

$$(p^{-1})^{-1} \equiv p$$

在 Martin-Löf 类型论中, 需要对 p 进行归纳才能证明这一点.

取反操作能带来一些比较逆天的东西. 在 Descartes 立方类型论中, 加上取反操作后会出现这样的东西:

$$i = \neg i : \mathbb{F}$$

我们还可以这样补齐:

$$\text{hcom}^{r \rightsquigarrow \neg r}(\lambda x. \{r = \neg r \mapsto u, \dots\})$$

9.1. 趣味应用: 中点问题

在前面介绍的立方类型论和 Martin-Löf 类型论中, 拼接两个相等 $p : x =_A y, q : y =_A z$ 的时候, 拼出来的 $\text{cat}(p, q) : x =_A z$ 就不再能取出中间的那个 y 了, 但我们现在可以操作一番 (例子来自笔者另一篇笔记 [9]):

$$h(i) := \text{hcom}^{\neg i \rightsquigarrow i} \left(\lambda x. \begin{cases} \neg i = i \vee x = 0 \mapsto y \\ i = 0 \mapsto p @ \neg x \\ i = 1 \mapsto q @ x \end{cases} \right)$$

大致检查一下:

- 令 $x := 0$, 也就是底面 y , 我们考察它和另外两条是否兼容.
 - 若代入 $i = 0$ 分支, 得到 $p @ \neg 0 \equiv p @ 1 \equiv y$, 没有问题,
 - 若代入 $i = 1$ 分支, 得到 $q @ 0 \equiv y$, 更没有问题.
- $h : \mathbb{I} \rightarrow A$, 那么我们自然会好奇它的两个端点是什么.
 - 考虑 $h(0)$, 观察三条分支, $\neg i = i$ 不成立, $i = 0$ 成立, 而补齐返回的时候需要 $x := 1$, 因此给出 $p @ \neg 1 \equiv x$, 因此 $h(0) \equiv x$,
 - 考虑 $h(1)$, 只有 $i = 1$ 成立, 因此给出 $q @ 1 \equiv z$, 因此 $h(1) \equiv z$.

看起来 $h : x =_A z$, 所以从 p, q 构造 h 的过程是在拼接两个相等. 但事实真的如此简单吗? 考虑如下情况:

$$k = \neg k \vdash h(k) \equiv y$$

我们能够通过一些语境限制, 把这个拼接操作的中点拿出来.

看起来这个过程是允许我们取中点,但事实上我们可以在这个所谓的「中点」放任何东西,而不必是原本两个相等中的公共点,因为 $i = \neg i$ 这个等式和其它两条边是互斥的,我们只需要保证它和底面兼容即可,因此我们可以做这种事情:

$$h(i) ::= \text{hcom}^{\neg i \rightsquigarrow i} \left(\lambda x. \begin{cases} x = 0 \mapsto y \\ \neg i = i \mapsto b @ x \\ i = 0 \mapsto p @ \neg x \\ i = 1 \mapsto q @ x \end{cases} \right)$$

其中 $b : y = u$. 这样就得到了「左右两边分别是 x 和 z 但中间是 u 」的全新相等 h .

但即便如此,引入取反操作并不意味着存在某个值 $\frac{1}{2} : \mathbb{I}$, 这点还需要注意. 因为在空语境中, \mathbb{I} 还是只有 0 和 1 两个实例. 如果真的要引入 $\frac{1}{2} : \mathbb{I}$ 也不是不可以,但是这会引出额外的复杂问题,而且并不能解决立方类型论中的现有问题.

10. 趣味问题二: 缝合补齐

前面说过,我们痛恨空补齐,那么自然会有读者产生如下两个疑问:

1. 它到底有多坏?
2. 有没有办法防止它产生呢?

空补齐来自于所谓「可空补齐」,也就是不完整的补齐,例如 $\text{hcom}(\lambda x. \{i = 0 \mapsto u, x = 0 \mapsto v\})$. 而所谓的「不完整」,就是存在某个替换 σ 能把这个补齐变成一个空补齐,比如前面这个例子就可以通过 $\sigma ::= [i \mapsto 1]$ 这个替换变成空的. 因此,如果我们想要消灭空补齐,就首先需要消灭可空补齐.

在 [10] 和 Agda 中有一个思路类似的解决方法: 缝合补齐. 这是补齐操作的一种升级,它收取和 hcom 完全一样的输入,但转化为 hcom 时会找到缺失的面,并把底面给缝上去. 这样,就可以一定程度上消灭可空补齐.

一个感性的定义如下 (为了可读性,使用了和之前定义补齐时稍微不一样的写法):

$$\frac{i : \mathbb{I}, \varphi \vee i = 0 \vdash u : A}{\text{ghcom} \left(\lambda i. \left\{ \underbrace{\varphi \vee i = 0 \mapsto u} \right\} \right) ::= \text{hcom} \left(\lambda i. \left\{ \underbrace{\varphi \vee i = 0 \mapsto u, \neg \varphi \mapsto u[i \mapsto 0]} \right\} \right)}$$

这里「找到缺失的面」这个操作记作 $\neg \varphi$, 可以理解为 \mathbb{F} 的「取反」操作,因为它要找到一组面的「其它部分」. 注意和 \mathbb{I} 的取反区分,这两个操作完全不是一回事. 这个操作的定义有些复杂,涉及语义的细节,因此我们不在这里展开.

缝合补齐还有异质的版本,叫 gcom , 思路类似,也不再展开.

在 [10] 和 [5] (图表 4.2) 中 (定义几乎是一模一样的), 缝合补齐操作的定义是归纳的,而且没有用到这个取反操作. 这个版本的缝合补齐又叫「AFH 缝合补齐」,名字出自 [10] 的三位作者的姓氏首字母. 归纳定义使用文字叙述如下:

- 令 $\varphi ::= \perp$, 也就是空的缝合补齐, 这种情况会直接给出底面, 因为侧面全部被填上了底面,
- 对于 $r = s \mapsto u$ 这个面, 它首先给出 $r = 0 \wedge s = 0 \mapsto u$ 和 $r = 1 \wedge s = 1 \mapsto u$, 然后再对于 $r = 1 \wedge s = 0$ 和 $r = 0 \wedge s = 1$ 分别进行缝合补齐, 这样一路到头变成空的之后就可以走上一条规则.

代码实现可以参考 András Kovács 的 cctt 项目⁵。[10] 的作者之一 Favonia 说这个缝合补齐非常的恐怖, Jonathan Sterling 认为实现这个版本的缝合补齐非常的痛苦。笔者推测可能的原因是它在论文里的版本只考虑了 $r = s \mapsto \dots$ 这样的边界, 没有考虑 $r = s \wedge r' = s' \mapsto \dots$ 等情况, 明显这些情况处理起来更麻烦。总之 András Kovács 认为它棒极了。

在 Agda 中, 缝合补齐的实现不太一样, 但大体思路是一致的, 感兴趣的读者可以查阅论文 [11] 的第 5.2 节。注意: Agda 的立方类型论功能有两篇标题完全一样的论文, 一篇发表于 ICFP 会议, 一篇发表于 JFP 期刊, 后者是最新版本, 请大家尽量阅读后者。本文的参考文献中, 引用的也是后者。

10.1. 趣味应用: Brunerie-Ljungström 数

在同伦类型论的早期研究中, Guillaume Brunerie 证明了以下命题:

命题 10.1.1: 存在 $n : \mathbb{Z}$ 使得 $\pi_4(\mathbb{S}^3) \cong \mathbb{Z} / n\mathbb{Z}$ 。

这个数被称为 Brunerie 数。经典同伦论表明 n 等于 ± 2 , Brunerie 也在同伦类型论中证明了这一点。需要强调的是, Brunerie 数的定义是构造主义的, 且他构造了一串函数, 复合后会输出这个 2。这个证明本身意义并不明显, 但是在构造这个证明的过程中需要用到大量抽象同伦论的结论, Guillaume Brunerie 为了证明这一结论, 大力发展了综合同伦论 (也就是同伦类型论中的同伦论), 这是非常重要的贡献。

而在同伦类型论的计算机实现中, 我们可以写出这个程序, 然后让计算机自动计算出该数的值。对于立方类型论而言, 这是一个很好的测试用例。Guillaume Brunerie 的博士论文中跳过了一些比较麻烦的步骤, 所幸人们基本补上了这些步骤, 在各个不同的版本的立方类型论实现中都定义了这个程序。

很遗憾, 所有计算这个程序的尝试都失败了, 例如 Favonia 使用了一台 1TB 内存的电脑来运行, 最终该程序在 90 小时后内存溢出。而 Minghao Liu 使用 Scala 实现了一个能动态编译 (也就是将立方类型论中的闭包和替换操作全部提升到 Java 虚拟机中的函数和函数应用, 而 Java 虚拟机可以将之进一步编译为机器码) 的立方类型论实现⁶, 并且支持了独立的宇宙补齐, 但依然内存溢出。

Axel Ljungström 对这个程序进行了简化, 得到了一个等价但更简单的程序, 参见 [12]。本文为了区分, 将这个新的程序称为 Brunerie-Ljungström 数。这个数字在实现了缝合补齐的 Agda 中可以快速算出结果, 算出来是 -2 。值得一提的是, Agda 版本的 Brunerie-Ljungström 数在计算时, 如果去掉缝合补齐而改为直接使用补齐, 会出现类型错误。András Kovács 发现可以通过人工插入一些胶合类型相关的规则来解决这个问题⁷。

在 András Kovács 的实现中 (用到了他发明的胶合求值法), 在没有缝合补齐的时候, Brunerie-Ljungström 数可以在 50 秒左右算出来。此时, 每次使用胶合类型的转换规则都会产生两个空补齐。该计算中途产生了 6 千万个空补齐。当然, 这个时候用到了他人工插入的那些胶合类型的规则。

而加上了缝合补齐后, 这个程序只需要 0.5 毫秒就能算出来, 不再有类型错误, 而且中途只会产生 700 个空补齐。这大约是十万倍的性能提升, 且空补齐的数量减少到了十万分之一。足以看出空补齐多么可恨!

至此, Brunerie 数的传奇已经基本告一段落, 不知道还有没有勇士会去试图计算原版的 Brunerie 数, 不过这基本不能带来任何收益, 因为我们已经有足够优化的立方类型论实现, 有了更好

⁵<https://github.com/AndrasKovacs/cctt/blob/c2576da9afd7050d78b672f9954d8e6b7bca5020/src/Core.hs#L1103-L1147>

⁶<https://github.com/molikto/mlang>

⁷https://github.com/AndrasKovacs/cctt/blob/c2576da9afd7050d78b672f9954d8e6b7bca5020/tests/new_brunerie_adjusthcom.cctt#L970-L1031, 十分推荐阅读, 因为实在是太好笑了。

的定义, 而且我们在写这个程序之前就已经知道答案. 若要做出新的结果, 需要计算更高阶的同伦群, 而编写这个程序的难度也会大大增加. 目前 Brunerie 数基本上已经变成一个梗.

11. 术语中英对照表

本讲义对许多目前没有标准翻译的类型论术语按照香蕉空间的规范给出了中文翻译, 以下是一个对照参考表. 有很多术语的英语版本非常的混乱, 笔者在翻译时擅自进行了整理, 并且欢迎读者提出更好的建议, 不过需要尽可能遵循本讲义的命名原则.

同伦论名词:

同伦型 homotopy type, anima, ∞ -groupoid
经典同伦论 classical homotopy theory
环路空间 loop space
纬悬 suspension
紧生成弱 Hausdorff compactly generated weakly Hausdorff, CGWH
万有覆盖 universal cover

道路 path
单纯集 simplicial set
方形范畴 category of cubes
单形范畴 category of simplices
圆周 circle
甜甜圈的表面 torus

范畴论以及范畴语义相关名词:

幂对象 exponential object
语境范畴 category of contexts
自函子 endofunctor
预层 presheaf

子对象分类子 subobject classifier
子对象 subobject
结构 structure
性质 property

数学或同伦类型论名词:

立方类型论 cubical type theory
Descartes 立方类型论 Cartesian cubical type theory
同伦类型论 homotopy type theory
编码-解码 encode-decode
模态 modality
模态类型论 modal type theory
商集 quotient set
泛等公理 univalence axiom

归纳 induction
高阶归纳类型 higher inductive type
相继式演算 sequent calculus
自然演绎 natural deduction
研究生引理 graduate lemma
命题截断 propositional truncation
可缩纤维等价 contractible fiber equivalence
半伴随等价 half-adjoint equivalence
综合同伦论 synthetic homotopy theory

编程语言名词:

de Bruijn 编号 de Bruijn index
无名表示 nameless representation
语法树 syntax tree
模式匹配 pattern matching
化简 reduction
类型转换 coercion, cast

子类型 subtype
胶合求值法 glued evaluation
闭包 closure
动态编译 just-in-time compilation, JIT
替换操作 substitution

类型论规则:

形成规则 formation rule
构造规则 introduction rule
消去规则 elimination rule

计算规则 computation rule
唯一性规则 uniqueness rule

类型论性质:

一致性 consistency
合流性 confluence
正规性 normalization
正规形式 normal form

典范性 canonicity
典范形式 canonical form

类型论名词:

相等 (无修饰语) judgmental equality
相等类型 identity type, equality type
相等证明 identity proof, equality proof
函数外延性 function extensionality
命题等价 propositional equality
依值类型 dependent type
依值相等 polymorphic equality
异质相等 heterogeneous equality
归纳类型 inductive type
构造子 constructor

积类型 product type
宇宙 universe
宇宙层次 universe hierarchy
野蛮但好使的分层 crude but effective strat-
ification
判断 judgment
语境 context
传递性 transitivity
对称性 symmetry

立方类型论和双层类型论名词:

面映射 face map
语境限制 context restriction
命题宇宙 universe of propositions
余纤维化 cofibration
对角线 diagonal
双层类型论 two-level type theory, 2ltt
纤性 fibrancy
外类型 exo-type
不完整类型 partial type
补齐 composition
同质补齐 homogeneous composition, hcom

异质补齐 heterogeneous composition, com
空补齐 null composition
可空补齐 nullable composition
缝合补齐 generalized composition
正则性 regularity
胶合类型 glue type
边界 boundary
取反 (针对 \mathbb{I}) involution
取反 (针对 \mathbb{F}) negation
中点 midpoint

参考文献

- [1] T. Univalent Foundations Program, *Homotopy Type Theory — Univalent Foundations of Mathematics, First Edition*, 2023.
- [2] K.-B. Hou (Favonia), C. Angiuli, and R. Mullanix, “An order-theoretic analysis of universe polymorphism,” *Proc. ACM Program. Lang.*, vol. 7, no. POPL, 2023, doi: 10.1145/3571250. [Online]. Available: <https://doi.org/10.1145/3571250>
- [3] T. Zhang, “A tutorial on implementing de morgan cubical type theory,” 2023. [Online]. Available: <https://arxiv.org/abs/2210.08232>

- [4] A. Kovács, “Staged compilation with two-level type theory,” *Proc. ACM Program. Lang.*, vol. 6, no. ICFP, Aug. 2022, doi: 10.1145/3547641. [Online]. Available: <https://doi.org/10.1145/3547641>
- [5] C. Angiuli, “Computational semantics of cartesian cubical type theory,” Thesis, Carnegie Mellon Univ., Pittsburgh, PA, USA, 2019.
- [6] E. Cavallo, and R. Harper, “Higher inductive types in cubical computational type theory,” *Proc. ACM Program. Lang.*, vol. 3, no. POPL, Jan. 2019, doi: 10.1145/3290314. [Online]. Available: <https://doi.org/10.1145/3290314>
- [7] S. Huber, “A cubical type theory for higher inductive types,” 2017. [Online]. Available: <https://simhu.github.io/misc/hcomp.pdf>
- [8] C. Angiuli, E. Cavallo, K.-B. Hou (Favonia), and J. Sterling, “A cool cartesian cubical type theory,” 2019. [Online]. Available: <https://github.com/Infinity-Type-Cafe/ntype-cafe-summer-school/blob/main/resources/kan.pdf>
- [9] T. Zhang, “Cartesian \boxtimes category with involution,” 2023. [Online]. Available: <https://vixra.org/abs/2306.0017>
- [10] C. Angiuli, K.-B. H. (Favonia), and R. Harper, “Computational higher type theory III: univalent universes and exact equality,” 2017. [Online]. Available: <http://arxiv.org/abs/1712.01800>
- [11] A. Vezzosi, A. Mörtberg, and A. Abel, “Cubical agda: a dependently typed programming language with univalence and higher inductive types,” *J. Functional Program.*, vol. 31, 2021, doi: 10.1017/S0956796821000034.
- [12] A. Ljungström, and A. Mörtberg, “Formalizing and computing a brunerie number in cubical agda,” 2023. [Online]. Available: <https://arxiv.org/abs/2210.08232>