

一点儿综合同伦论

A Bit of Synthetic Homotopy Theory

kang

July 4, 2023

$$\left[\begin{array}{l} \text{data } S^1 : \text{Type where} \\ \text{base} : S^1 \\ \text{loop} : \text{base} = \text{base} \end{array} \right] = \text{[Diagram of a circle with a base point]}$$

目录

1	What is... 同伦论?	3
2	类型与空间	4
2.1	类型的语义 (可以) 是空间!	4
2.2	类型论的句法	7
2.3	类型规则	9
2.4	函数类型与纤维化	10
3	同伦与同伦等价	12
3.1	点的同伦	12
3.2	映射的同伦	15
3.3	同伦等价	16
4	立方体	17
4.1	立方体的子复形	18
4.2	部分元素与扩张类型	20
4.3	HELP	22
4.4	Univalence 公理	24
5	简单的空间性质	26
5.1	类型与命题	26
5.2	可缩空间	28
5.3	截断	31
6	S^1	32
6.1	连通性	32
6.2	环路空间	33
6.3	同伦群	37

1 What is... 同伦论?

同伦论是拓扑学的主要分支,而拓扑学是一门研究形状的学科,更准确地说,它探究的是在连续变形下仍然保持不变的几何性质的学科.为了进行严谨的数学表述,数学家们首先创立了点集拓扑,也称为一般拓扑学.事实上,点集拓扑最初是为了研究分析学而创造的,但后来成为了几何学的基石之一.基于此,我们可以定义我们所研究的几何对象,即拓扑空间.以及一种被称为同胚的等价关系,这种等价的概念不受连续形变的影响.

数学家们常说,拓扑学的基本课题是研究在同胚下保持不变的特征,也就是所谓的拓扑不变量.长期以来,最重要的拓扑不变量都是通过应用代数学的方法获得的代数对象,比如一些群或环,由此产生了代数拓扑的概念.有趣的是,这些不变量往往在一种更一般的等价关系——同伦等价下仍能保持不变.随着研究的深入,数学家们开始意识到同伦在拓扑学中具有更基础的地位.于是同伦论诞生了,它转而研究空间在同伦等价下的不变性质.虽然严格来说,同伦论应该被看作是代数拓扑的一个分支,但如今人们常常将两者看作是同义词.

虽然拓扑学总被认为是非常抽象的理论,但其中许多几何上的直观概念,即使对于缺乏训练的普通人而言也是容易理解的.然而,严格地定义空间及其不变量是一项困难的任务.更加艰巨的挑战在于理解空间之间的同伦等价关系,并建立一套完整的技术体系来进行构造和证明.


2 类型与空间

2.1 类型的语义 (可以) 是空间!

如果有人信誓旦旦地向你保证, 同伦类型论可以用来研究同伦论, 那个人究竟在说些什么? 简而言之, 同伦类型论是一门语言, 它可以以空间的同伦论作为语义, 就像中文可以以现实世界中的事物作为语义. 换句话说, 同伦类型论中的任何语句都可以被视为关于空间的陈述, 或者是同伦论中的理论表达.

为了阐明这一点, 首先我们需要理解什么是语言、什么又是语义. 语言是由一系列符号的组合构成的. 然而, 我们不能随意地将这些符号组合在一起, 而必须遵循一定的语法规则. 在形式语言中, 这些组合规则被称为**句法** (syntax). 我们必须严格依照句法规则来构建表达式. 当我们描述类型论时, 我们是在描述可以使用的符号以及组合这些符号的句法. 而当我们使用类型论时, 我们是在按照句法规则书写各种不同长度的语句.

当然, 除了逻辑学家和语言学家, 很少有人仅仅为了一门语言的句法而去学习它. 我们更感兴趣的是语言能够表达什么, 也就是语言的**语义** (semantics). 然而, 语言和语义经常会被混淆. 举个例子: 在中文中, 我们使用词语 **鸭子** 来代表鸭鸭这种可爱的动物. 然而, 一个单词 (作为一串符号) 和它所指代的事物 (也就是它的含义或语义) 实际上是不同的概念. 为了清晰地区分它们, 我们使用类型论学家的括号 $\llbracket - \rrbracket$ 来扩住一种语言中的语句, 以表示它所指代的含义. 在刚刚的例子中, 单词 **鸭子** 是中文中的一个表达式, 而 $\llbracket \text{鸭子} \rrbracket$ 则表示这个词所指代的动物:

$$\llbracket \text{鸭子} \rrbracket = \text{鸭子}$$


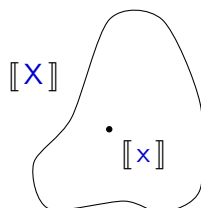
让我们回到类型论与同伦论. 类型论是一门形式语言, 而在这门语言中, 最基本的句式如下:

$$x : X$$

这同样是最重要的句式, 我们在使用类型论时几乎只会构造出这一种句子. 对于这类句子, 我们会说 X 是一个**类型** (type), 而 x 是类型 X 的一个**项** (term). 需要强调的是, 术语“类型”和“项”很大程度上仅仅是语言学的概念, 或者说句法的一部分. 这类似于在分析中文句子时:

鸭子在游泳。

我们会称 **鸭子** 是主语, 而 **游泳** 是谓语. 这种纯粹的语法概念可能会使你感到困惑. 幸运的是, 为了理解一门语言, 人们没有必要完全从形式的规则出发, 而可以从它们所表达的含义出发, 或者说从语义入手. 就像主语往往指向现实世界中的事物, 而谓语则代表它们的动作或行为一样, 我们认为**每个类型都指向一个几何空间, 而该类型的项则指代该空间中的一个点**:



一旦我们理解了这一点, 我们就能够理解为什么类型论可以用来表述同伦论. 这其实很简单, 不是吗?

如果我们需要表达 **X** 是一个类型, 可以这样写:

X : Type

这里 **Type** 是**所有类型的类型**, 通常被称为**类型宇宙** (type universe). 而作为类型宇宙的项, **X** 自然就是一个类型了. 换句话说, 一个类型可以看成是某个总体类型的项. 那么 **Type** 指的是什么空间呢? 用先前的术语来说, 我们在问其语义 $[[\text{Type}]]$ 是什么. 答案并不令人意外——它指代的是**所有空间的空间**, 被称为**宇宙** (universe) 或者**对象分类器** (object classifier), 我们用 \mathcal{U} 来表示它:

$$[[\text{Type}]] = \mathcal{U}$$

要理解所有空间的空间是什么, 需要花一些功夫. 这个概念使我们能够将一个空间本身看作是某个总体空间中的一个点. 同时, 由于我们可以在空间中自由连续地移动其中的点, 因此这个总体空间允许我们对一个空间进行连续的变化. 这种变化就是所谓的同伦. 我们稍后会更详细地讨论这个概念, 但现在你只需要知道这样的想法是有意义的.

小心 \triangle 2.1. 你可能已经注意到, 讨论由所有空间组成的空间是一件危险的事情. 就像讨论所有集合组成的集合一样, 这些概念都会引发所谓的罗素悖论. 实际上, 并不存在由所有空间组成的空间, 我们只能选择其中一部分作为总体. 这一部分被称为**小空间** (*small space*), 而其余的则被称为**大空间** (*large space*). 然而, 在数学实践中, 数学家们并不关心这个问题, 这是因为

从技术上讲，考虑所有空间和仅考虑小空间并没有太大的区别，所以我们也不会过多强调这一点。

2.2 类型论的句法

因为我们会使用类型论的语言来研究同伦论，所以有必要先讨论一下类型论这种语言本身。在类型论中，完整的句子被称为**判断** (judgement)，判断和它们的规则一般以**自然演绎** (natural deduction) 的方式被写下来。我们使用的类型论里只有三种类型的判断：

$$\Gamma \text{ context} \qquad \Gamma \vdash x : X \qquad \Gamma \vdash x = y$$

这三种判断从左至右分别是：

“ Γ 是一个合法的语境。”

“在语境 Γ 下， x 是类型 X 的项。”

“在语境 Γ 下，项 x 与 y 依判断相等。”

在上一节中，我们已经提到了中间这种类型的判断。然而，你肯定会注意到这里多了一个叫做语境的概念。让我们来解释一下这些术语和判断的含义。**语境** (context)，又叫**上下文**，一般用大写的希腊字母来表示。语境指的是我们在写下一句话时所有需要用到的自由变量，或者说前提。语境是按照下面的规则归纳构造的：

$$\frac{}{\cdot \text{ context}} \qquad \frac{\Gamma \vdash X : \text{Type}}{\Gamma (x : X) \text{ context}}$$

符号 \cdot 指**空语境** (empty context)，在这个语境下的表达式不包含任何变量。要求 Γ 是合法语境的判断在这里被省略了，但实际上我们总是要求任何写出的 Γ 都必须是合法的。这些规则意味着，语境可以以列表形式被写下来，这被称为 **telescope**：

$$(x_1 : X_1) (x_2 : X_2) \dots (x_n : X_n)$$

在 telescope 中，每一个 X_i 都是类型。在这个语境下写出的表达式就可以使用这些 **bound variable** x_i 。

当我们在真正使用类型论而不是研究它们的句法时，我们往往不会明确地指出语境。这和我们日常使用中文是一样的，没有人在说一句话之前会把这句话的一切前提都列举一遍，这些背景条件往往是靠人们自行从上下文里推断。而如果你在使用基于类型论的程序语言或者 proof assistant，表达式的语境则会由计算机根据语法自动推导出来。

判断相等 (judgemental equality), 或者称为**定义相等** (definitional equality), 是类型论句法中所包含的一种等价关系. 在通常的语言中, 语句是由遵循一些规则的字符串. 然而, 在类型论中, 更好的理解方式将语句视为字符串的**等价类**, 其中需要商掉的关系就是判断相等. 换句话说, 彼此间判断相等的表达式就是“同一个”表达式. 这是类型论和通常语言的一个区别.

在类型论中, 我们要求所有对表达式的操作和演绎都必须保持判断相等, 尽管很多时候我们不会明确地写出这些判断相等的规则.

对于类型论中的表达式, **代换** (substitution) 是非常重要的操作. 所谓的代换就是把一个表达式里中的某个自由变量全部替换为另一个表达式, 表示为

$$\{\dots\}[a/x]$$

这里的 $\{\dots\}$ 是一个包含自由变量 x 的表达式. 我们会把其中的 x 全部替换为表达式 a . 下面的 (自然语言中的) 例子可以这一点:

$$“x 真的很可爱”[鸭鸭/x] = “鸭鸭真的很可爱”$$

最后我们有两类重要的规则, 它们分别是代换规则和弱化(weakening)规则. 代换规则指的是: 如果在语境 Γ 下的表达式中的自由变量能够被全部代换成语境 Δ 下的表达式, 那么代换后的就是语境 Δ 下合法的表达式. 而弱化规则指的是: 在某个语境 Γ 下的表达式, 仍然是将 Γ 做扩充之后得到的语境下的表达式. 同时, 所有表达式间的判断相等都被这些操作所保持.

2.3 类型规则

在前面的部分中，我们介绍了**结构规则** (structural rule)，它们是在类型论中进行表达和推理的基本要素和模式。然而，作为一种“类型的理论”，我们却还没有明确定义一个真正的类型。

构造和表达类型以及它们项的规则叫做**类型规则** (typing rule)。每种类型都对应一系列类型规则。尽管准确描述类型规则很困难，但它们通常遵循一定的模式甚至套路，主要有以下几类：

1. **类型构造规则** (type formation rule) 告诉我们如何构造类型。通常，我们会使用现有的项和类型来组合生成新的类型表达式。
2. **项引入规则** (term introduction rule) 告诉我们如何编写特定类型的项。这通常包括该类型本身固有的项，以及如何通过组合现有的项和类型生成新的项。
3. **项消去规则** (term elimination rule) 告诉我们如何使用特定类型的项。通常，这些规则允许我们编写到其他类型的函数。
4. **计算规则** (computation rule) 或者有时称为 β 规则 (β -rule)，告诉我们当把消去规则应用于引入规则所引入的项时会得到什么。
5. 当我们对一个项使用消去规则时，会得到许多其他结果。**唯一性原理** (uniqueness principle) 或者有时称为 η 规则 (η -rule) 告诉我们，如果再次使用引入规则，从这些结果重新构造出一个项时，得到的项与最初的项之间是什么关系。许多类型并没有唯一性原理，但该原理在理论上具有重要价值。

注 2.2. 很多时候，类型构造规则可以被视为类型宇宙 **Type** 的项引入规则。

当然，从上面的表述中很难准确理解这些规则的含义。在接下来的章节中，我们将逐步引入新的类型并给出它们的类型规则。您很快就会理解这些规则的概念。

2.4 函数类型与纤维化

对于任意两个类型 X 与 Y ，类型论的句法允许我们构造**函数类型** (function type)，其写作：

$$X \rightarrow Y$$

这是最重要的类型之一，其语义是从 $\llbracket X \rrbracket$ 到 $\llbracket Y \rrbracket$ 所有连续映射组成的空间。连续映射就是连续地将 $\llbracket X \rrbracket$ 中的点变成 $\llbracket Y \rrbracket$ 中点的变换方式。函数类型的项被称为从 X 到 Y 的**函数**，其对应于一个连续映射。

$$f : X \rightarrow Y$$

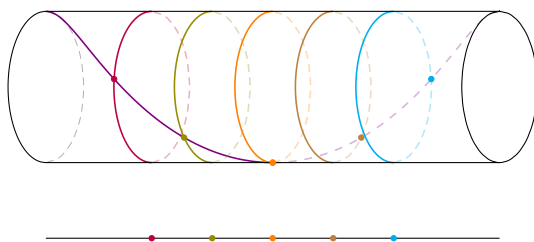
定义函数的句法规则通常被称为 λ -**抽象** (λ -abstraction)：

$$f = \lambda x \mapsto \{ \dots \}$$

其中 $\{ \dots \}$ 表示包含参数 x 的表达式。或者我们也可以将其写成以下形式：

$$f x = \{ \dots \}$$

类型论可以直接地表述几何学中的一个重要概念，即**纤维化** (fibration)。下图展示了一个纤维化，还有它的一个涂成紫色的**截面** (section)。这个纤维化的**底空间** (base space) 是下方的直线，其所有**纤维** (fiber) 都是圆周，而**全空间** (total space) 是圆柱的表面。我们在底空间上选择了五个点，并分别为这些点处的纤维以及截面对应的值粉刷了五种不同的颜色。如果你把横向摆放的圆柱面切割为无数个纵向的圆周，就可以得到这个纤维化，而截面则是围绕圆柱面旋转一周形成的螺旋线。



让我们更仔细地解释这些术语，你可以对照上面的图加以理解。在纤维化中，我们连续地为**底空间**上的每个点指定一个空间，这个空间称为该点处的**纤维**。纤维化的一个**截面**就是为底空间的每个点连续地选取其纤维中的一个点。此外，我们可以沿着底空间将所有纤维合并，得到的空间称为**全空间**。

很快你就会发现，虽然用语言描述纤维化相当繁琐，但这个概念其实非常直观. 纤维化是几何学和拓扑学中最基本且最重要的概念之一.

那么如何在类型论中表述纤维化呢? 非常简单，一个纤维化就是一个到 \mathbf{Type} 的函数:

$$P : X \rightarrow \mathbf{Type}$$

正如我们在上一节中提到的, $\llbracket \mathbf{Type} \rrbracket$ 是所有空间组成的空间. 因此, 这里的 $\llbracket P \rrbracket$ 可以被理解为从 $\llbracket X \rrbracket$ 到所有空间组成的空间的连续函数, 或者说, 关于 $\llbracket X \rrbracket$ 的每个点连续地选择一个空间. 这与纤维化的定义是一致的, 实际上我们可以认为 $\llbracket X \rightarrow \mathbf{Type} \rrbracket$ 就是由 $\llbracket X \rrbracket$ 上的纤维化所组成的空间.

纤维化的相关概念也可以用类型论的语言来表达. 对于 X 中的项 x , 对应于 $\llbracket P \rrbracket$ 在 $\llbracket x \rrbracket$ 处纤维的类型就是 $P\ x$. 而对应于 $\llbracket P \rrbracket$ 的所有截面的类型被称为 Π 类型 (Π -type), 一般写作

$$\Pi(x : X) P\ x \quad \text{或者} \quad (x : X) \rightarrow P\ x$$

此外, 对应于 $\llbracket P \rrbracket$ 全空间的类型被称为 Σ 类型 (Σ -type), 一般写作

$$\Sigma(x : X) P\ x \quad \text{或者} \quad (x : X) \times P\ x$$

3 同伦与同伦等价

3.1 点的同伦

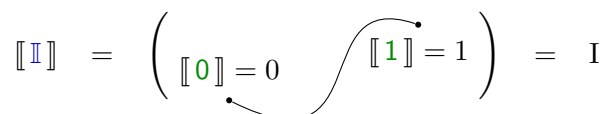
同伦是描述连续移动或变换的概念. 最简单的同伦是点的同伦, 简单到我们甚至不会提到同伦这个词, 而称其为**道路** (path). 在空间中, 道路指的是一个点从一处连续移动到另一处所形成的轨迹. 通常情况下, 我们将道路定义为从**区间** (interval) $I = [0, 1]$ 到空间 X 的连续映射:

$$p : I \rightarrow X$$

尽管在这个定义中涉及一些技术细节, 但实际上它非常直观. 可以想象一下, 一个点从区间 I 的左端点 0 连续移动到右端点 1 , 并且这个点的运动在映射 p 下保持不间断, 其最终在空间 X 中便形成一条道路. 另一种理解方式是, 用两只手分别握住一条绳子的两端, 然后随意摆弄绳子, 这条绳子就是我们所处的三维空间中的一条道路.

这个概念乍看上去十分平凡, 但它是整个同伦论的起点, 并且可以引出非常复杂和深刻的思想. 不过在进一步深入之前, 让我们先考虑如何在类型论的语言中表述这个概念.

实际上, 类型论中有不止一种表述道路的方式, 在这里我们选择一种与几何本身非常接近的方法. 我们直接引入一个区间类型 \mathbb{I} 来指代区间 I . 它有两个特殊的项 0 和 1 , 分别代表区间的两个端点:

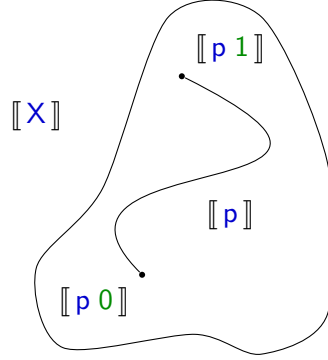
$$\mathbb{I} = \left(\begin{array}{c} \text{[[0]]} = 0 \quad \text{[[1]]} = 1 \end{array} \right) = I$$


注 3.1. 这里再次强调, 作为类型的区间 \mathbb{I} 和作为空间的区间 I 是不一样的. 前者是句法的一部分, 而后者是作为语义的数学对象.

有了区间类型 \mathbb{I} , 类型论中的道路就可以定义为从 \mathbb{I} 出发的函数:

$$p : \mathbb{I} \rightarrow X$$

它的语义自然就是空间中的道路:



我们希望 \mathbb{I} 的句法规则能够非常简单. 这意味着区间的精细结构 (例如在分析学或点集拓扑中的性质) 将被完全抛弃, 只有少部分能够描述同伦论的组合结构会被保留. 这也意味着能够对区间类型 \mathbb{I} 做的事情要比真正的区间空间少得多. 我们的 \mathbb{I} 拥有所谓的 **de Morgan 代数** 结构, 也就是三个基本运算: 将区间颠倒 (左右翻转), 以及两种将正方形压扁成区间的操作. 更准确地说, 给定变量 $i, j : \mathbb{I}$, 它们分别指代:

$$[[\sim i]] = 1 - [[i]]$$

$$[[i \wedge j]] = \min [[i]] [[j]]$$

$$[[i \vee j]] = \max [[i]] [[j]]$$

在数学中, 这里的两种二元运算被称为**连接** (connection).

区间类型应该被赋予哪些结构并没有标准答案, 目前至少有两种方案——**de Morgan 立方类型论**和 **Cartesian 立方类型论**. 我们的区间类型接近于前者. 在这套体系中, 类型 \mathbb{I} 在句法上其实只包含两个项 0 、 1 以及三种运算 \sim 、 \wedge 和 \vee . 虽然 \mathbb{I} 的句法非常简单, 但已经足够让我们研究整个同伦论了.

在大多数情况下, 我们其实只会关注连接给定端点的道路. 给定一个类型 X 的两个项 x 与 y , 句法规则允许我们构造连接这两个项的**道路类型** (path type), 其写作

$$\text{Path } x \ y \quad \text{或者} \quad x = y$$

它的语义自然就是以 $[[x]]$ 和 $[[y]]$ 为起点和终点的道路所形成的空间.

值得注意的是, 我们使用了相等符号 $=$ 来表示道路类型. 这是因为我们可以在类型论中定义**相等类型** (identity type), 它由一个被称为 **J 规则** (J rule) 的公理所刻画, 而道路类型正满足这个规则:

$$\begin{aligned} J : \{X : \text{Type}\} \{x \ y : X\} (P : X \rightarrow \text{Type}) \\ (p : x = y) \rightarrow P \ x \rightarrow P \ y \end{aligned}$$

注 3.2. 这里的 J 规则其实只是一个特例，但它已经能够解释相等的概念。完整的版本将在后面给出。

两个事物何时是相等的？这是一个哲学问题。哲学家们对此有各种不同的观点，其中之一被称为**不可分者同一性原理** (identity of indiscernibles). 后来，Per Martin-Löf 将这个思想引入了类型论，并将其命名为 J 规则。不可分者同一性原理的陈述是：两个事物 x 和 y 相等，当且仅当

对任何性质 P ，只要 x 满足 P ，那么 y 也满足 P 。

通过令性质 P 为“和 x 相等”，我们可以直接证明这个原理中“仅当”的部分。你可以从上述 J 的类型中看出， J 规则就是这个原理中“当”的部分。总的来说，相等类型的定义是：如果存在这样一种关系，只要 x 和 y 满足它，就没有任何东西能够真正区分 x 和 y ，那么这个关系其实就是“ x 与 y 相等”！

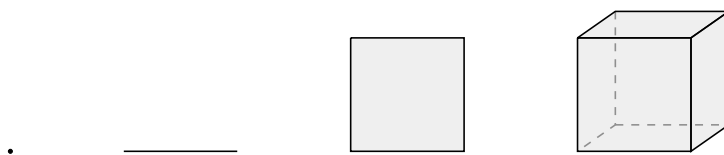
有趣的是，这其实已经暗示了我们的类型论语言正在描述同伦论——两点的同伦和两点的相等是一回事！这正是从同伦论视角看待事物的方式。实际上， J 是所谓同伦提升性质的推论，我们之后再讨论这一点。

3.2 映射的同伦

3.3 同伦等价

4 立方体

一个 n 维的立方体可以被认为是 n 维欧几里得空间 \mathbb{R}^n 中, 所有坐标取值在单位区间 $x_i \in [0, 1]$ 中的点所组成的子空间. 在低维情形下, 立方体是我们熟悉的物体: 0 维时是点, 1 维时是区间, 2 维时是正方形而 3 维时就是人们常说的“立方体”.



我们的 (类型论) 语言包含区间类型 \mathbb{I} , 因而能够表达道路的概念. 任何包含参数 $i : \mathbb{I}$ 的表达式可以被看成是在一条道路上连续分布的语句. 但事实上, 我们可以为我们的语境引入任意多个区间类型的参数:

$$i_1 \dots i_n : \mathbb{I}$$

当我们写出的表达式依赖于这 n 个参数时, 可以认为这个表达式实际上被一个 n 维的立方体所参数化. 换句话说, 由于类型论的基本规则, 只需要一个区间类型, 我们实际上就可以表述任意维度的立方体了.

同样, 区间类型所拥有的项和运算也都对立方体有意义. 这使得我们可以对立方体进行操作, 表达一个立方体的任意面, 并定义立方体之间复杂的映射. 特别地, 我们可以定义很多退化的立方体。

4.1 立方体的子复形

一个立方体的面 (face) 指的是其边界上的那些维度更低的子立方体. 需要强调的是, 在我们的用法中, 面不仅仅是余一维的子立方体, 还包括顶点、连接顶点的线段、线段围成的方形等等 ...

术语**复形** (complex) 一般指的是一种能够被非常规整地表示成某些非常简单的图形组合的几何体. **立方复形** (cubical complex) 则是由立方体沿着它们的面拼接而成的空间. 在我们的讨论中, 我们将只使用一个特例——通过将立方体的一些面并起来得到的子复形.

正如之前提到的那样, 当我们的语境 Γ 中有 n 个类型为区间 $i : \mathbb{I}$ 的变量时, 我们可以认为我们的语境本身包含一个 n 维的立方体. 为了能够更深入地表达同伦论的概念, 我们必须有能力将我们的语境限制在这个立方体的某个子复形上, 这意味着需要引入新的句法.

让我们仔细解释一下表达“语境立方体”子复形的句法规则——**余纤维化** (cofibration). 余纤维化 ϕ 是一类表达式, 它们与“语境立方体”所有的子复形一一对应:

$$\phi : \mathbf{Cof}$$

如何写出一个真正的余纤维化呢? 首先, 在句法中有两个特殊的余纤维化, 分别对应于空集和整个立方体:

$$0 : \mathbf{Cof}$$

$$1 : \mathbf{Cof}$$

其次, 对于语境中每个区间类型的自由变量 $i : \mathbb{I}$, 我们可以写出对应于坐标 $\llbracket i \rrbracket$ 等于 0 和 1 时的那一对面的余纤维化:

$$(i = 0) : \mathbf{Cof}$$

$$(i = 1) : \mathbf{Cof}$$

由此, “语境立方体”中所有的余一维面都可以被表示出来. 接下来, 我们注意到立方体的所有低维面实际上可以写成余一维面的交集. 因此我们引入一个“交”运算, 对于任意余纤维化 $\phi \psi : \mathbf{Cof}$, 它可以写成:

$$\phi \sqcap \psi : \mathbf{Cof}$$

方便起见, 很多时候我们会省略运算符而直接写 $\phi \psi$ 来代表“交”运算. 最后, 一个立方体的子复形就是其面的并集, 因此我们再引入一个“并”运算, 对于

任意余纤维化 $\phi \psi : \text{Cof}$, 它可以写成:

$$\phi \cup \psi : \text{Cof}$$

借助于这些基本的余纤维化和运算, 我们的类型论获得了表达“语境立方体”任意子复形的能力.

注 4.1. 在同伦论中, 余纤维化 (cofibration) 是一个相当糟糕的名字, 它其实和纤维化是完全不一样的概念. 这个术语最初是用来描述性质良好的子空间或者说“性质良好的单映射”的. 因为它的“良好性质”与“性质好的满映射”——也就是纤维化 (fibration) 的“良好性质”相对偶, 所以才被这样命名.

我们可以随时将一个余纤维化加入语境:

$$\Gamma, \phi$$

需要注意的是, 这里的 ϕ 不能被当作一个自由变量来使用, 相反它限制了允许被写出的表达式的形式. 这和将 $(\phi : \text{Cof})$ 加入语境是完全不一样的. 当同时加入数个余纤维化时, 我们的语境就被限制在了这些子复形的交上. 为了体现这个性质, 我们的句法中会包含判断相等:

$$\Gamma, \phi_1, \dots, \phi_n = \Gamma, \phi_1 \cap \dots \cap \phi_n$$

到目前为止, 我们仍然没有给出在子复形上构造语句的方法. 这将在下一节中解释.

ToDo: 如何对余纤维化中的区间类型变量做代换.

4.2 部分元素与扩张类型

在一个给定的“语境子复形” ϕ 上,我们该怎么写出一个属于类型 $X : \text{Type}$ 的项呢? 这样的项被称为 X 在 ϕ 上的**部分元素** (partial element), 记作:

$$\{ \dots \} : \text{Partial } \phi X$$

部分元素是通过模式匹配来定义的. 简单地说, 当我们的余纤维化是一系列余纤维化的并时:

$$\phi = \phi_1 \cup \dots \cup \phi_n$$

我们就可以通过下面的写法引入一个在 ϕ 上的部分元素:

$$\lambda \left\{ \begin{array}{l} \phi_1 \mapsto x_1 \\ \dots \\ \phi_n \mapsto x_n \end{array} \right\}$$

这里的 x_i 是 X 分别在每个 ϕ_i 上的部分元素. 它们需要满足拼图一样的关系——在对应子复形的交集上它们必须是相同的元素 (这样才能拼成一个整体!). 更准确地说, 我们要求在每个 $\phi_i \cap \phi_j$ 上有判断相等:

$$x_i = x_j$$

现在, 只要我们知道在基本的纤维化及其交上如何构造部分元素, 就足以涵盖所有的情形下的构造了. 在空子集 $\mathbf{0}$ 上只有一种元素, 表示没有任何内容:

$$\lambda () : \text{Partial } \mathbf{0} X$$

而在整个立方体 $\mathbf{1}$ 上, X 的部分元素就是 X 本身的项, 换句话说:

$$X = \text{Partial } \mathbf{1} X$$

最后, 在 ϕ 对应余一维的面交时, 我们的语境仍然是一个立方体, 只不过维度变低了. 此时 X 的部分元素依然是 X 的项, 但是所有出现在 ϕ 表达式中的区间类型变量都不再是自由的, 而会受限於 ϕ 对它们的约束.

例 4.2. 假设语境里有自由变量

$$i \ j \ k : \mathbb{I}$$

然后我们将其限制到如下的子复形上

$$\phi = (i = 0) \cap (k = 1)$$

这样，在试图编写一个部分元素时

$$\{\dots\} : \text{Partial } \phi X$$

我们能使用的自由变量就只有 j . 如果表达式 $\{\dots\}$ 里出现了 i 或 k , 那么它们就会看成是已经被代换为了 0 或 1 .

为了研究同伦论, 我们还需要另一个非常重要的概念. 对于任意类型 $X : \text{Type}$ 和其在某个余纤维化 ϕ 上的部分元素 $x : \text{Partial } \phi X$, 我们可以定义扩张类型 (extension type):

$$X \{ \phi \mapsto x \}$$

扩张类型的项就是 X 的项, 但是这些项限制在 ϕ 上后必须与 x 判断相等. 换句话说, 这些项就是将 x (作为定义在 ϕ 子复形上的元素) 扩展到整个语境上所得到的元素. 固定端点的路径类型可以使用扩张类型来定义.

定义 4.3. 给定类型的道路

$$X : \mathbb{I} \rightarrow \text{Type}$$

以及它两端的项

$$a : X \ 0 \quad b : X \ 1$$

在 X 之上连接 a 和 b 的道路类型是:

$$\text{PathP } X \ a \ b = (i : \mathbb{I}) \rightarrow X \ i \left\{ \begin{array}{l} (i = 0) \mapsto a \\ (i = 1) \mapsto b \end{array} \right\}$$

作为特例, 给定类型 $X : \text{Type}$ 和项 $a \ b : X$, 连接 a 和 b 的道路类型是:

$$\text{Path}_X \ a \ b = \text{PathP } (\lambda i \mapsto X) \ a \ b$$

部分元素和扩张类型是立方类型论的基本概念, 自始至终我们都会不断使用它们.

4.3 HELP

当一位拓扑学家需要帮助时, **同伦扩张与提升性质** (homotopy extension and lifting property) 就会出现. 在类型论的语言中, **HELP** 对应着一个名为 **fill** 的运算.

句法 4.4. 在我们的语言中, 包含着一个 **fill** 运算符, 其引入规则如下:

$$\begin{aligned} \text{fill} : \{ \phi : \mathbf{Cof} \} (X : \mathbb{I} \rightarrow \mathbf{Type}) \\ (x : (i : \mathbb{I}) \rightarrow \mathbf{Partial} (\phi \cup (i = 0)) (X i)) \\ (i : \mathbb{I}) \rightarrow X i \{ \phi \cup (i = 0) \mapsto x i \} \end{aligned}$$

注 4.5. 依据同伦论的术语, **fill** 运算存在等价于说当 ϕ 的表达式中不含自由变量 $i : \mathbb{I}$ 时, 余纤维化 $\phi \cup (i = 0)$ 是一个平凡余纤维化 (acyclic cofibration). 同时我们的类型满足 **Kan** 条件 (Kan condition).

有趣的是, 一旦存在 **fill** 运算, 这样的运算就是唯一的. 这是一个非常重要的性质. 可以注意到, 唯一性本身也是通过使用 **fill** 证明的.

定理 4.6 (**fill** 的唯一性). 任取两个部分元素 x 的扩张:

$$a \ b : (i : \mathbb{I}) \rightarrow X i \{ \phi \cup (i = 0) \mapsto x i \}$$

它们都是相等的:

$$a = b$$

特别地, 它们都等于:

$$\text{fill } X \ x$$

证明. 连接 a 和 b 的道路 p 可以这么定义:

$$p \ j = \text{fill } X \ \lambda \ i \mapsto \lambda \ \left\{ \begin{array}{l} (j = 0) \mapsto a \\ (j = 1) \mapsto b \\ \phi \cup (i = 0) \mapsto x \ i \end{array} \right\}$$

□

另一个重要的运算是 **comp**, 它实际上是 **fill** 的终点:

$$\text{comp } X \ x = \text{fill } X \ x \ 1$$

运算 `comp` 有两个常用的特例. 一个是当类型 X 是常值的情形, 此时对应的运算被称为 `hcomp`, 这里参数 X 可以被省略:

$$\text{hcomp } \{X\} x = \text{comp } (\lambda i \mapsto X) x$$

另一个特例是当 ϕ 为空时, 此时对应的运算被称为 `transport`:

$$\begin{aligned} \text{transport} &: (X : \mathbb{I} \rightarrow \text{Type}) \rightarrow X \ 0 \rightarrow X \ 1 \\ \text{transport } X \ x &= \text{comp } X \ (\lambda i \mapsto \lambda \{ (i = 0) \rightarrow x \}) \end{aligned}$$

注 4.7. 我们使用的表述平行于经典同伦论. 然而, 在 *de Morgan* 立方类型论中, 我们通常反过来使用 `comp` 来定义 `fill`. 而 `comp` 又常常是被 `hcomp` 和 `transp` 所定义的. 这里的 `transp` 是 `comp` 的另一个特例.

和 `fill` 一样, 我们这里定义的 `comp`、`hcomp` 与 `transport` 都有对应于自身的唯一性.

利用 `transport` 我们可以证明道路类型满足之前章节中提到的 `J` 规则. 完整的 `J` 规则包含如下的 `J`, 它可以看作是相等类型的消去规则:

$$\begin{aligned} \text{J} &: \{X : \text{Type}\} \{x : X\} \\ & (P : (y : X) \rightarrow x = y \rightarrow \text{Type}) (d : P \ x \ \text{refl}) \\ & \{y : X\} (p : x = y) \rightarrow P \ y \ p \end{aligned}$$

还有 `J` 需要满足的 β 规则, 它可以看作是相等类型的计算规则:

$$\text{J}\beta : \text{J } _ \ d \ \text{refl} = d$$

定理 4.8. 道路类型满足 `J` 规则.

证明.

$$\text{J } P \ d \ p = \text{transport } (\lambda i \mapsto P \ _ \ (\lambda j \mapsto p \ (i \wedge j))) \ d$$

$$\text{J}\beta = \text{transportRefl}$$

□

这里的 `transportRefl` 指的是沿着 `refl` 做 `transport` 得到的元素等于原先的元素. 这可以由 `fill` 的唯一性证明.

4.4 Univalence 公理

立方类型论引入了一种名为 **Glue** 的新类型, 你可以将其视为 **Type** 的 cube constructor. 它允许我们从类型的同伦等价和现有的立方体构造出新的立方体. **Glue** 的引入使得 univalence 成为一个定理.

句法 4.9. 在我们的语言中, 包含着一个 **Glue** 类型, 其构造规则如下:

$$\begin{aligned} \text{Glue} : \{ \phi : \text{Cof} \} \{ X : \text{Type} \} \{ T : \text{Partial } \phi \text{ Type} \} \\ (f : \text{Partial } \phi (T \simeq X)) \rightarrow \text{Type } \{ \phi \mapsto Y \} \end{aligned}$$

为了更清晰地表示 **Glue** 类型的参数, 我们通常将其写为:

$$\text{Glue } \{ \phi \mapsto (T, f) \} X = \text{Glue } \{ \phi \} \{ X \} \{ T \} f$$

从另一个角度来说, **Glue** 本身也是一个 type constructor. 它的 term introduction rule 与 term elimination rule 分别由一个 constructor 和一个 destructor 给出:

$$\begin{aligned} \text{glue} : (t : \text{Partial } \phi T) \rightarrow X \{ \phi \mapsto f t \} \rightarrow (\text{Glue } \{ \phi \mapsto _ \} X) \{ \phi \mapsto t \} \\ \text{unglue} : \text{Glue } \{ \phi \mapsto _ \} X \rightarrow X \end{aligned}$$

类似的, 我们会把 **glue** 写成

$$\text{glue } \{ \phi \mapsto t \} x = \text{glue } t x$$

Glue 类型的 computation rule 与 uniqueness principle 如下, 你可以认为这两个规则就是在表达 **glue** 和 **unglue** 是一对互逆的函数:

$$\begin{aligned} \text{unglue } (\text{glue } \{ \phi \mapsto t \} x) &= x \\ \text{glue } \{ \phi \mapsto t \} (\text{unglue } u) &= u \end{aligned}$$

正如本节开头提到的, 我们可以使用 **Glue** 类型来证明 univalence.

$$\text{ua} : \{ X Y : \text{Type} \} \rightarrow X \simeq Y \rightarrow X = Y$$

首先, 我们可以这样定义 **ua**:

$$\text{ua } \{ X \} \{ Y \} f i = \text{Glue } \left\{ \begin{array}{l} (i = 0) \mapsto (X, f) \\ (i = 1) \mapsto (Y, \text{id}_Y) \end{array} \right\} Y$$

接下来, 我们需要证明 β 规则:

$$\text{ua} \beta : \text{transport } (\text{ua } f) = f$$

定理 4.10. Univalence 在我们的类型论中成立.

证明. 证明 $\mathbf{ua}\beta$ 相当于证明对任意 $x : X$ 有

$$\mathbf{transport} (\mathbf{ua} f) x = f x$$

由 $\mathbf{transport}$ 的唯一性, 我们只需要构造一条在 $\mathbf{ua} f$ 上方连接 x 和 $f x$ 的道路即可. 这条道路可以使用 \mathbf{glue} 得到:

$$\lambda i \mapsto \mathbf{glue} \left\{ \begin{array}{l} (i = 0) \mapsto x \\ (i = 1) \mapsto f x \end{array} \right\} (f x)$$

□

5 简单的空间性质

5.1 类型与命题

在使用类型论来表达数学理论时，有一个基本的哲学，你可能已经了解到或者将在本讲义中看到：

类型即命题

实际上，这是一个非常简单的观点。但如果你不是从小就接触类型论并用它来理解数学，一开始可能会感到十分困惑。

在数学训练中，理解什么是“命题”以及什么是“命题”的“证明”是非常重要的环节。这也是令许多数学教师非常头疼的教学任务。在传统的数学基础之下，“命题”和“数学对象”本质上处在两个不同的世界之中。“命题”陈述了“数学对象”的某些性质，这件事不可能反过来，我们也不会声称“证明”了一个“数学对象”。当然，我们可以证明某些“数学对象”的存在性或者其他什么性质。但请注意，这里证明的实际上都是“命题”。

然而，在类型论中，“命题”和“数学对象”在语法上属于同一类事物，它们都是我们所说的“类型”。而“证明”一个“命题”和为一类“数学对象”构造实例是同一件事，它们都是为一个类型编写一个项。

让我们来举个例子。整数 \mathbb{Z} 是一个类型：

$\mathbb{Z} : \text{Type}$

众所周知，整数有加法运算 $+$ 。断言“整数的加法是交换的”的命题也是一个类型（我们将其命名为 Comm ）：

$\text{Comm} : \text{Type}$

$\text{Comm} = (m\ n : \mathbb{Z}) \rightarrow m + n = n + m$

构造类型 \mathbb{Z} 的项就是给出一个整数，这可以非常简单：

$0 : \mathbb{Z}$

也可以非常复杂：

超难的数 $: \mathbb{Z}$

超难的数 = “用 20000 页高深数学写出的表达式”

而构造类型`Comm`的项则是在证明“整数的加法是交换的”这个命题。当然，这个证明可以很简单，也可以故意写得晦涩难懂。

```
proof : Comm
proof = “整数加法交换性的证明”
```

无论乍看上去如何，“类型即命题”是类型论中非常自然的现象。一旦你熟悉了类型论的语言，也必定会欣赏这件事。

5.2 可缩空间

我们说一个空间是可缩的, 如果这个空间中存在一个点, 使得整个空间可以通过道路收缩到这个点上.

定义 5.1. 给定类型 $X : \text{Type}$, 如果存在 X 的一个点, 使得从该点到 X 的任意点都存在一条道路, 我们就称 X 是可缩的.

$$\text{isContr } X = \Sigma (a : X) ((x : X) \rightarrow (a = x))$$

这个定义在最原始的 HoTT 中也是有意义的. 然而, 一旦我们的语言中有立方体的概念, 就可以给出一个 (乍看上去) 更强的定义, 这个定义使用起来更加灵活且具有更大的威力.

定理 5.2. 类型 $X : \text{Type}$ 可缩当且仅当可以定义如下的 contr 运算:

$$\text{contr} : \{\phi : \text{Cof}\} (x : \text{Partial } \phi X) \rightarrow X \{ \phi \mapsto x \}$$

注 5.3. 定理中的 contr 运算可以这么理解: 对于任何立方体, 类型 X 定义在这个立方体任意部分上的元素, 都可以被无条件地扩张到整个立方体上.

证明. 首先, 我们使用运算 contr 来证明 X 的可缩性. 我们构造如下的项:

$$p : \text{isContr } X$$

它的两个分量都是 contr 的直接应用. 首先, 注意到当我们取 ϕ 为空子集时, 运算 contr 会给出 X 的一个项. 我们就取这个项作为收缩的中心:

$$p.1 = \text{contr } \lambda ()$$

随后, 我们利用 contr 来将其连接到任意的项 $x : X$:

$$p.2 \ x \ i = \text{contr } \lambda \left\{ \begin{array}{l} (i = 0) \mapsto p.1 \\ (i = 1) \mapsto x \end{array} \right\}$$

反之, 假设 X 可缩:

$$(a, q) : \text{isContr } X$$

我们来构造一个 `contr` 运算. 对于任意的部分元素 $x : \text{Partial } \phi \ X$, 我们首先利用收缩 q 把 x 变成一个常值的元素, 然后将常值元素进行扩张 (扩张成定义域更大的常值元素即可), 最后再使用 `hcomp` 将收缩逆转.

$$\text{contr } \{\phi\} x = \text{hcomp } \lambda i \mapsto \lambda \left\{ \begin{array}{l} \phi \mapsto q \ x \ (\sim i) \\ (i = 0) \mapsto a \end{array} \right\}$$

□

利用这个性质, 我们可以轻松地证明可缩性天生就是一个命题.

定理 5.4. 类型 `isContr` X 是命题.

证明. 对于任意的两个项 $u_0 \ u_1 : \text{isContr } X$, 我们假设它们形如:

$$\begin{aligned} u_0 &= a_0, q_0 \\ u_1 &= a_1, q_1 \end{aligned}$$

现在, 我们来构造这两个项之间的道路:

$$\text{path} : u_0 = u_1$$

我们可以使用模式匹配来定义道路的两个分量. 注意, 我们已经假设了 X 的可缩性 (u_0 和 u_1 在我们当前的语境中可用), 因此我们可以自由地使用 `contr` 运算.

首先, `contr` 可以直接连接 a_0 和 a_1 :

$$\text{path } i \ .1 = \text{contr } \lambda \left\{ \begin{array}{l} (i = 0) \mapsto a_0 \\ (i = 1) \mapsto a_1 \end{array} \right\}$$

接下来需要构造这条道路之上的 q_0 和 q_1 间的道路. 按定义, 我们需要对每个 $x : X$ 构造一个给定边界的正方形, 这同样可以用 `contr` 做到:

$$\text{path } i \ .2 \ x \ j = \text{contr } \lambda \left\{ \begin{array}{l} (i = 0) \mapsto q_0 \ j \\ (i = 1) \mapsto q_1 \ j \\ (j = 0) \mapsto \text{path } i \ .1 \\ (j = 1) \mapsto x \end{array} \right\}$$

□

定理 5.5. 类型 $X : \text{Type}$ 可缩当且仅当存在 X 的项 $a : X$ 使得如下的 `extend` 运算可以被定义:

$$\text{extend} : \{Y : X \rightarrow \text{Type}\} \rightarrow Y \rightarrow a \rightarrow (x : X) \rightarrow P \ x$$

定理 5.6. 对类型 $X : \text{Type}$, 以下性质是等价的:

1. 类型 X 可缩;
2. 类型 X 是一个命题, 并且存在一个点 $a : X$;
3. 类型 X 等价于单位类型 `1`.

5.3 截断

6 S^1

6.1 连通性

让我们使用类型论证明下面这个非常基本的性质.

定理 6.1. 在类型 S^1 中, 我们可以 merely 用道路连接 `base` 和任意点.

$$(x : S^1) \rightarrow \parallel \text{base} = x \parallel$$

注 6.2. 这个性质比连通性 `isConnected` 更强.

证明. 我们直接进行模式匹配来构造这样一个项:

$$\text{proof} : (x : S^1) \rightarrow \parallel \text{base} = x \parallel$$

对于基点 `base` 的情形, 最简单的选择当然是常值道路:

$$\text{proof base} = \mid \text{refl} \mid$$

接下来, 在处理 `loop i` 的情形时, 需要一点点 (但非常常见的) 技巧. 对于每个 $i : \mathbb{I}$, 一个想当然的选择是道路

$$\lambda j \mapsto \text{loop } (i \wedge j)$$

遗憾的是, 当 $i = 1$ 时, 这条道路成为了 `loop` 而不是 `refl`, 与先前在 `base` 处的定义不一致. 不过请注意, 这里的构造不是在道路空间本身, 而是在道路空间的命题截断中进行的. 因此我们可以使用 `squash` 来连接不同的项, 然后再配合 `hcomp` 修正错开的端点即可:

$$\text{proof } (\text{loop } i) = \text{hcomp } \lambda k \mapsto \lambda \left\{ \begin{array}{l} (i = 0) \mapsto \mid \text{refl} \mid \\ (i = 1) \mapsto \text{squash } \mid \text{loop} \mid \mid \text{refl} \mid k \\ (k = 0) \mapsto \mid \lambda j \mapsto \text{loop } (i \wedge j) \mid \end{array} \right\}$$

□

推论 6.3. 类型 S^1 是连通的.

$$\text{isConnected } S^1$$

6.2 环路空间

环路 (loop) 就是两端连接相同点的道路, 这个点叫做环路的**基点** (base). 环路空间 (loop space) 指的是一个空间中某点处所有环路构成的空间. 在这一节中, 我们使用类型论来研究 S^1 的环路空间.

$$\Omega S^1 = \Omega(S^1, \text{base}) = \text{base} = \text{base}$$

定理 6.4. 类型 S^1 在基点 **base** 处的环路类型 ΩS^1 等于整数类型 \mathbb{Z} .

$$\Omega S^1 = \mathbb{Z}$$

对于单个的道路类型, 我们可以基于 **HELP** 来构造许多道路 (term introduction), 但却不能直接使用道路 (term elimination). 这意味着, 虽然定义到道路类型的函数相对容易, 但是定义从道路类型出发的函数却十分困难. 即使我们知道道路类型的具体模型应该是什么, 也很难证明它们的确是等价的.

为了解决这个问题, 我们可以采用一种称为 encode-decode 的方法. 这种方法的核心思想是, 虽然很难对单个的道路类型进行消去操作 (elimination), 但整个**路径丛** (path fibration) 却具有天然的消去规则, 即 **J** 规则. 因此, 我们不再局限于单个道路类型的模型, 转而构造整个路径丛的模型, 并试图证明它们作为丛的等价性.

证明梗概. 首先, 我们需要构造路径丛的模型, 通常被称为 **Code**:

$$\text{Code} : S^1 \rightarrow \text{Type}$$

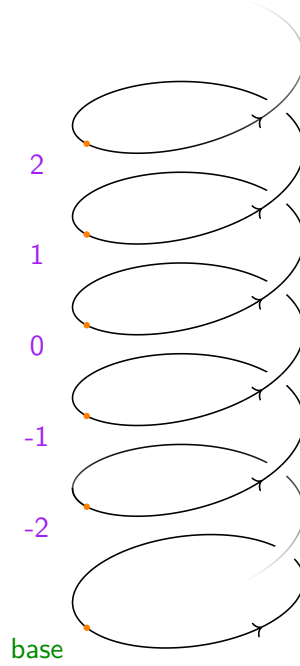
接下来, 我们定义 **Code** 和路径丛之间的一对函数, 通常被称为 **encode** 和 **decode**:

$$\begin{aligned} \text{encode} &: (x : S^1) \rightarrow \text{base} = x \rightarrow \text{Code } x \\ \text{decode} &: (x : S^1) \rightarrow \text{Code } x \rightarrow \text{base} = x \end{aligned}$$

最后, 我们证明它们互为逆函数即可.

构造 Code 我们可以使用模式匹配来定义类型族 **Code**:

$$\begin{aligned} \text{Code } \text{base} &= \mathbb{Z} \\ \text{Code } (\text{loop } i) &= \text{ua succ } i \end{aligned}$$



构造 `encode` 正如之前提到的，我们可以轻松地通过 `J` 来定义 `encode`：

$$\text{encode } p = J \text{ Code } p \ 0$$

基点 `base` 处的 `encode` 函数通常被称为**卷绕数** (winding number)：

$$\begin{aligned} \text{winding} &: \Omega S^1 \rightarrow \mathbb{Z} \\ \text{winding} &= \text{encode } \text{loop} \end{aligned}$$

实际上，这个函数相当于在计算一条环路绕着圆周旋转了几圈。这个性质可以通过 univalence 的 β 规则 `ua β` 来证明。

$$\text{winding } \underbrace{(\text{loop} \cdot \dots \cdot \text{loop})}_{n \text{ 个 } \text{loop} \text{ 的复合}} = n$$

注 6.5. 在这里，绕 0 圈应该理解被为常值道路 `refl`。而绕负数圈则是沿着反方向的 `loop-1` 旋转。

构造 `decode` 函数 `decode` 的定义需要更具体的构造了。首先，我们定义基点 `base` 处的 `decode`：

$$\text{looping} : \mathbb{Z} \rightarrow \Omega S^1$$

直观上, `looping` 的定义是满足下面等式的函数. 换句话说, 它将整数 n 映射到沿着 `loop` 绕 n 圈的环路.

$$\text{looping } n = \underbrace{\text{loop} \cdot \dots \cdot \text{loop}}_{n \text{ 个 } \text{loop} \text{ 的复合}}$$

注意到根据前面的陈述, `winding` 和 `looping` 应该是一对互逆的函数, 因此下面的等式成立:

$$\text{winding } (\text{looping } n) = n$$

一旦完成 `looping` 的定义, 我们就可以进行模式匹配了:

$$\begin{aligned} \text{decode } \text{base} &= \text{looping} \\ \text{decode } (\text{loop } i) &= \{\dots\} \end{aligned}$$

需要注意的是, 我们还没有给出 `looping` 的严格定义, 也没有补全模式匹配中 $\{\dots\}$ 具体的表达式, 以及证明函数的互逆性. 我们把需要用到的构造和性质概括在下面的引理中:

引理 6.6. 我们可以完成下面的任务:

1. 构造满足上面描述的 `looping` 函数;
2. 证明对每个 $n : \mathbb{Z}$ 有:

$$\text{winding } (\text{looping } n) = n$$

3. 补全 `decode` 的模式匹配.

我们之所以这样做, 是为了提供清晰的直观理解, 而避免陷入繁琐的细节中. 这些构造依赖于我们如何定义整数类型 \mathbb{Z} , 以及在我们的类型论中包含多少判断相等. 如果我们将 \mathbb{Z} 定义为归纳类型, 那么上面关于 `looping` 的陈述可以很容易地被严格化. 而引理中的第 1 点和第 2 点但可以通过简单的归纳法建立.

现在我们来解释引理中的第 3 点. 一种理解这个问题的方法是注意到, 补全 `decode` 的模式匹配等价于找到这样的道路:

$$\text{transport } (\lambda i \mapsto \text{Code } (\text{loop } i) \rightarrow \text{base} = \text{loop } i) \text{ looping} = \text{looping}$$

换句话说, 我们需要证明沿着 `loop` 将 `looping` 函数 `transport` 一周后得到的函数仍然等于它自身. 利用 Π 类型和道路类型的 `transport` 规则, 以及

univalence 的 β 规则, 左侧的函数可以被更具体地表述实际上, 左侧的函数可以写成:

$$\lambda n \mapsto \text{looping} (\text{pred } n) \cdot \text{loop}$$

根据我们之前对 `looping` 的描述, 我们可以证明:

$$\text{looping} (\text{pred } n) = \text{looping } n \cdot \text{loop}^{-1}$$

现在使用道路的乘法结构即可证明我们所需的相等性.

证明互逆性 接下来我们需要验证 `encode` 和 `decode` 是一对互逆的函数. 验证其中一侧的互逆性非常容易, 只需要直接使用 `J` 即可:

$$\begin{aligned} _ : \{x : S^1\} (p : \text{base} = x) &\rightarrow \text{decode } _ (\text{encode } _ p) = p \\ _ &= J _ \{...\} \end{aligned}$$

这里的 `{...}` 应该填写一条道路:

$$\text{looping} (J _ \text{refl } 0) = \text{refl}$$

这并不困难, 我们首先利用 $J\beta$, 将左侧化简为 `looping 0`. 根据 `looping` 的定义, 我们知道这就是常值道路,

接下来我们处理另一侧. 事实上, 我们的定理只用到了在 `base` 处的互逆性, 而这就是引理6.6的第 3 条性质.

一般情形可以被视为推论. 我们可以利用 `encode` 在 `base` 处是等价, 以及 S^1 的连通性来推出 `encode` 在整个 S^1 上都是等价:

$$(x : S^1) \rightarrow \text{isEquiv} (\text{encode } x)$$

因为等价的单侧逆就是双侧逆, 使用 `decode` 是 `encode` 的单侧逆即可完成全部证明.

□

注 6.7. 这个定理的语义是, 空间 S^1 的环路空间同伦等价于整数的集合 \mathbb{Z} .

$$\Omega S^1 \simeq \mathbb{Z}$$

6.3 同伦群

几乎在每一门代数拓扑课程中, 计算 S^1 的基本群都是第一个真正意义上的非平凡定理.

定理 6.8. 类型 S^1 是群胚.

`isGroupoid S^1`