Family 与 Fibration 与类型论

本文将会以“family 与 fibration 的联系/等价性”为主线（第一节将会解释这是什么），去讨论/推导范畴论和 dependent type 的范畴语义中的一些构造。具体来说：

- 第一节将会在集合论的背景下讨论 family 与 fibration，并用范畴的语言描述它们。
- 第二节将会把逐步集合拓展到任意范畴，得到 category of elements、grothendieck construction 和 grothendieck fibration 的定义。
- 第三节将会从以“如何表示 type family 和 context extension”为主线，介绍类型论的范畴语义的几种常见构造
- 第四节将会从具体地用范畴语言翻译 dependent type 中最基本的一些构造

每一节需要的背景知识是：

- 第一、第二节：最最基本的范畴论（范畴的定义、functor、product/pullback 的定义，怎么看交换图）
- 第三节：最最基本的 dependent type （知道 dependent product=function 和 dependent sum=pair 大概是什么即可），以及 Simply Typed λ Calculus 与 cartesian closed category 的联系（你可以看我之前写的 推导 Simply Typed Lambda Calculus 的范畴语义）
- 第四节：前面的全部，但范畴论浓度稍高一些

本文是以具备上述背景知识的读者，只需要按顺序慢慢地读一遍就能理解全部内容为目的写作的。我希望它实现了这一目的。

本文中一些记号的约定如下：

- 对于哪些字母代表什么没有明确区分，交由上下文决定。但我会尽量避免同一个字母有不同含义/某个字母的含义与常见含义不同
- 对于形如 交换图 A $\overset{F}{\Longrightarrow}$ 交换图 B 的交换图，它们的意义是交换图 A 被 functor F 送到了交换图 B。这些交换图中有时会带有颜色，A 中某种颜色的东西会被 F 映射到 B 中有相同颜色的东西去

- 如果 $\mathcal{C}$ 是一个范畴，那么 $Ob(\mathcal{C})$ 是 $\mathcal{C}$ 中对象的集合，$c \in \mathcal{C}$、$c \in Ob(\mathcal{C})$ 表示 c 是 $\mathcal{C}$ 中的一个对象
- 如果 $\mathcal{C}$ 是一个范畴，$c, d \in \mathcal{C}$，那么 $\mathtt{Hom}_{\mathcal{C}}(c, d)$ 表示 $\mathcal{C}$ 中从 c 到 d 的箭头的集合，$f : c \to d \in \mathcal{C}$、$c \xrightarrow{f} d \in \mathcal{C}$、$f \in \mathcal{C}(c, d)$ 表示 f 是 $\mathcal{C}$ 中一个从 c 到 d 的箭头

1 Family 与 fibration，set theoretically

1.1 family 与宇宙

什么是 family? 让我们首先从 family of sets 开始。一个 family of sets 无非就是一堆集合。为了将这一堆集合里的不同集合区分开，我们可以用一个集合 I 作为 index。所以，给定一个集合 I，以 I 作为 index 的一个 family of sets 就是一堆集合 $A_i (i \in I)$。Family of sets 几乎无处不在。例如，在一个类型系统里，每个类型 τ 都对应一些以它为类型的表达式 $t : \tau$，那么此时“每个类型的合法表达式”就构成一个 family $\mathtt{Tm}(\tau)_{\tau \in \mathtt{Types}}$，其中 $\mathtt{Tm}(\tau) = \{t \mid t : \tau\}$。

那么，我们应该如何表示一个 family 呢? 如果每个 A_i 都包含在一个更大的“集合宇宙”$\mathcal{U}$ 中的话，那么 $A_i (i \in I)$ 这个 family 就可以看成一个 $I \to \mathcal{U}$ 的函数。如果对于某个集合 D，所有 A_i 都满足 $A_i \subseteq D$ 的话，那么 $P(D)$（D 的幂集）就是一个可能的 $\mathcal{U}$ 的选择。例如，对于上面的 $\mathtt{Tm}(\tau)_{\tau \in \mathtt{Types}}$ 的例子，D 可以取 `Terms`，全体表达式（不管类型是什么、有没有类型）的集合。

1.2 family 与 fibration

看上去，关于 family 的研究到这里就可以结束了。然而，当我们试图用范畴论的语言来研究 family 时，问题出现了。考虑 `Set`，全体在某个“宇宙”$\mathcal{U}$ 中的“小”集合与它们之间的函数构成的范畴。如果我们想要在这个范畴里表示一个 family $A_i (i \in I)$，我们需要将它表示成一个 $I \xrightarrow{A} \mathcal{U}$ 的函数。但是 $\mathcal{U}$ 不能是 `Set` 中的对象，否则就会导致 $\mathcal{U} \in \mathcal{U}$——罗素悖论!

所以，“射向宇宙的箭头”这种表示法是有其局限性的。当“宇宙”$\mathcal{U}$ 本身无法直接拿到时，这种表示法就无法使用。幸运的是，我们可以在不提及宇宙的情况下表示一个 family：对于一个 family $A_i (i \in I)$，我们可以用它的“total space”：

$$\sum_{i \in I} A_i = \{(i, a) \mid i \in I, a \in A_i\}$$

来表示这个 family 自身。total space 中的每个元素都额外存储了它自身的 index，所以我们可以用一个函数 $\Sigma_{i \in I} A_i \xrightarrow{\pi} I$ 来取出这个 index。给定一个 i_0，我们可以从 $\sum_{i \in I} A_i \xrightarrow{\pi} I$ 中提取出 A_{i_0}：

$$A_{i_0} \cong \{x \in \Sigma_{i \in I} A_i \mid \pi(x) = i_0\}$$

（严格来说，这个集合并不真的是 A_{i_0}，因为它的元素额外存储了 index i_0。但它们是等价的）所以，total space 的确可以正确地表示一个 family。

于是我们得到了一种不使用宇宙来表示 family 的办法。不过，上面的 total space 是一个非常具体的集合论构造。这并不理想，我们还希望从 total space 和外部的互动与性质来刻画它。忽略具体的构造方式，那么 total space 包含的结构是 $\bullet \xrightarrow{\pi} I$：一个集合，以及一个从它到 I 的函数 π，用于提取出 index。但这看上去有些过于 general 了：任何一个指向 I 的函数都符合这一定义！所以，一个很自然的问题是：每个指向 I 的函数都可以看成一个以 I 为 index 的 family 吗？事实上，的确如此。考虑一个 $X \xrightarrow{f} I$ 的函数，我们可以通过 inverse image 构建出一个 family $f_i^{-1}(i \in I)$：

$$f_i^{-1} = \{x \in X \mid f(x) = i\} \qquad i \in I$$

所以，我们得到了在 family 和函数之间的一对变换：

$$\begin{aligned} -^{-1} &: \text{函数} \to \text{family} \\ f_i^{-1} &= \{x \in X \mid f(x) = i\} \\ \Sigma &: \text{family} \to \text{函数} \\ \Sigma A_i &= \Sigma_{i \in I} A_i \xrightarrow{\pi} I \end{aligned}$$

而且它们是互逆的，因为：

$$\begin{aligned} \Sigma(f^{-1}) &= \Sigma_{i \in I} f_i^{-1} \xrightarrow{\pi} I \\ &= \{(i, x) \mid i \in I, x \in X, f(x) = i\} \xrightarrow{\pi} I \\ &= \{(f(x), x) \mid x \in X\} \xrightarrow{\pi} I \\ &\cong f \\ (\Sigma A_i)_i^{-1} &= \{x \in \Sigma_{i \in I} A_i \mid \pi(x) = i\} \\ &= \{(i, a) \mid a \in A_i\} \\ &\cong A_i \end{aligned}$$

所以，在 `Set` 里，在集合论的背景下：以 I 为 index 的 family 和指向 I 的函数是一一对应的。我们可以用一个指向 I 的函数/箭头来表示 family，从而不需要提到宇宙 $\mathcal{U}$。而一个指向 I 的箭头 $\bullet \xrightarrow{\pi} I$，就称为一个 **fibration**。它的 domain、total spce，就称为这个箭头的 fiber。Fibration 与 fiber 的概念起源于拓扑学，同样是用于刻画 family。不过，在本文档中，“fibration”将会指代“箭头”，而本文档的主线、family 与 fibration 的联系，指的正是：

> 每个以 I 为 index 的 family，都可以用一个指向 I 的 fibration（箭头）来表示

1.3 fibration 与宇宙

最后，不妨将宇宙再搬出来，看看 fibration 与宇宙这两种表示一个 family 的方法之间的联系。假设 $\mathcal{U}$ 是一个集合宇宙，$I \xrightarrow{A} \mathcal{U}$ 是 $\mathcal{U}$ 中的一个 family，$X \xrightarrow{\pi} A$

是这个 family 对应的 fibration，它们可以被放进这样的一张图里：

$$\begin{array}{ccc} X & & \\ \downarrow{\scriptstyle \pi} & & \\ I & \xrightarrow{A} & \mathcal{U} \end{array}$$

但这张图并不能给我们提供 π 和 A 应该满足的性质。π 只会把 index 取出来，但关于 X 中、各个 A_i 中的元素，这张图完全没有提及。我们希望表达如下的事实：

对于 X 中的每个 (i, a)，$a \in A_i$

所以我们需要某种方式来表达集合上的属于关系。事实上，可以通过一个特殊的 fibration/family 来表达所有从属关系。定义：

$$\hat{\mathcal{U}} = \Sigma_{S \in \mathcal{U}} S = \{(S, s) \mid S \in \mathcal{U}, s \in S\}$$

从 $\hat{\mathcal{U}}$ 到 $\mathcal{U}$ 有一个 projection ϖ，它把 (S, s) 中的集合 S 取出。利用这个特殊的 fibration，我们可以刻画出任何一个其他 family 的 fibration 该满足的性质：

$$\begin{array}{ccc} X & \dashrightarrow & \hat{\mathcal{U}} \\ \vdots{\scriptstyle \pi} & & \downarrow{\scriptstyle \varpi} \\ I & \xrightarrow{A} & \mathcal{U} \end{array}$$

这张交换图说明了什么呢？沿着左下走，我们从 total space 中拿出了某个元素对应的 index i，并通过 A 得到了它在 family $A_i (i \in I)$ 中对应的集合。沿着右上走，我们从 total space 中拿出了某个元素 a，并同时记录它所属的一个集合 S，随后单独把 S 拿出来考虑。由于这张交换图 commute，沿着两条路径的结果一致，$S = A(i)$。从而我们得到了 $a \in A(i)$——关于 total space 中元素的性质！最后，我们希望 X 中没有其他多余的信息，所以我们要求上述交换图是一个 pullback square。于是，我们得到了集合中宇宙和 fibration 需要满足的性质。事实上，在任何一个范畴中，利用这些性质，我们可以定义出抽象的宇宙与 fibration：

- 在一个范畴中，一个“宇宙”是两个对象 $\mathcal{U}$、$\hat{\mathcal{U}}$ 以及一个箭头 $\hat{\mathcal{U}} \xrightarrow{\varpi} \mathcal{U}$
- 并不是随便一个箭头都是宇宙。宇宙应该满足的性质是：对于它之中的每个 family（一个 $I \xrightarrow{A} \mathcal{U}$ 的箭头），都有对应的 fibration 存在。换言之，对于每个 $I \xrightarrow{A} \mathcal{U}$，存在下面的 pullback square：

$$\begin{array}{ccc} \bullet & \dashrightarrow & \hat{\mathcal{U}} \\ \vdots & \lrcorner & \downarrow{\scriptstyle \varpi} \\ I & \xrightarrow{A} & \mathcal{U} \end{array}$$

- 而如果一个箭头 $\bullet \to I$ 是某个上面那样的 pullback square 中左侧的箭头，那么它就是 $\mathcal{U}$ 中的一个 fibration=family。

不妨将集合的例子代入，来验证这一定义。如果 $\mathcal{U}$ 是一个集合宇宙（或者任意一个集合的集合），那么可以把 $\hat{\mathcal{U}}$ 定义成 $\Sigma_{S\in\mathcal{U}}S$，$\varpi$ 是显然的 projection。对于每个 $I \xrightarrow{A} \mathcal{U}$ 的函数/family A，利用 pullback 可以看成满足一些等式的 product 的特点，我们可以直接构造出 $I \dashleftarrow \bullet \dashrightarrow \hat{\mathcal{U}}$：

$$\begin{aligned}\bullet &= \{(i,S,s) \mid i\in I, (S,s)\in\hat{\mathcal{U}}, A(i)=S\}\\ &= \{(i,S,s) \mid i\in I, S\in\mathcal{U}, s\in S, A(i)=S\}\\ &\cong \{(i,s) \mid i\in I, s\in A(i)\}\\ &= \sum_{i\in I} A(i)\end{aligned}$$

其中，红色的条件是使 pullback square 交换的条件。这种“满足等式的 product”的构造法本身保证了构造出的的确是 pullback。所以，集合论中的 family 与 fibration 的确是范畴中抽象的宇宙/fibration 定义的一个特例。

2 Family 与 fibration，category theoretically

前面，我们讨论了以**集合**为 index 的、**集合**的 family。通过这个例子，我们建立起了 family $\sim$ fibration 的第一个例子与最初的直觉。下面，我们将逐步把这一现象拓展到任意的范畴。

2.1 以范畴为 index 的集合、category of elements

首先，我们将作为 index 的集合 I 拓展为一个范畴。什么是以一个范畴 I 为 index 的 family of sets？一个以范畴 I 为 index 的 family，就是一个 $I \xrightarrow{F} \mathtt{Set}$ 的 functor。把这个定义展开，首先，对于 I 中的每个 object i，有一个对应的集合 $F(i)$。此外，对于 I 中的箭头 $i \xrightarrow{f} i'$，$\mathtt{Set}$ 中有一个函数 $F(f) : F(i) \to F(i')$。所以，以范畴为 index 的 family 会额外考虑不同 index 之间的联系，因而是以集合为 index 的 family 的一个拓展。

于是，我们可以开始考虑，这些拓展后的 family 对应的 fibration 是怎样的。同第一节的例子一样，我们可以先尝试给出一个具体的“total space”的构造。暂且忘记 I 中的箭头，那么 I 就变成了一个集合，所以，这个“total space”（记作 $\int_I F$）的对象集合应该是：

$$\sum_{i\in Ob(I)} F(i) = \{(i,a) \mid i\in Ob(I), a\in F(i)\}$$

那么，这个 total space 要如何反映不同 index 间的联系呢？让我们考虑 I 中的一个箭头 $i \xrightarrow{f} i'$：

- 在 $\mathtt{Set}$ 中，它对应 $F(i)\to F(i')$ 的一个函数 $F(f)$
- 对于 $(i,a)\in\int_I F$，函数 $F(f)$ 可以将它送到 $\int_I F$ 中的另一个元素 $(i', F(f)(a))$

所以，I 中的箭头，会对应 $\int_I F$ 中一些元素之间的联系，我们可以把 $\int_I F$ 定义成一个范畴：

- $Ob(\int_I F) = \sum_{i \in Ob(I)} F(i)$
- $\int_I F$ 中，两个元素 (i, a)、(i', a') 之间的箭头是 I 中的一个箭头 $f : i \to i'$，使得 $F(f)(a) = a'$

有一个显然的投影 functor $\pi : \int_I F \to I$。它把 (i, a) 送到 i，箭头 f 送到 f 自身。所以，family/fibration 的联系在从集合 index 拓展到范畴 index 后，依然是有效的。上述 $\int_I F$ 的构造，称为 F 的 *category of elements*，有很多重要的应用。

在第一节中，我们看到了宇宙和 fibration 之间的互动。这里，对于拓展后的 fibration，我们同样可以试图找到对应的“宇宙”。在集合里，充当宇宙的是 $\hat{\mathcal{U}} \xrightarrow{\varpi} \mathcal{U}$，其中 $\hat{\mathcal{U}} = \Sigma_{S \in \mathcal{U}} S$。现在，把 $\mathcal{U}$ 设置为 $\mathtt{Set}$，全体集合的范畴（忽略集合大小的问题），那么我们需要找到对应的 $\hat{\mathtt{Set}}$。回顾范畴中抽象的宇宙的定义，$\hat{\mathtt{Set}} \xrightarrow{\varpi} \mathtt{Set}$ 可以看成 $\mathtt{id}_{\mathtt{Set}}$ 这个以 $\mathtt{Set}$ 自身为 index 的 family 对应的 fibration。利用 category of elements，我们可以得到 $\hat{\mathtt{Set}}$ 的构造：

- $Ob(\hat{\mathtt{Set}}) = \Sigma_{S \in \mathtt{Set}} S$
- 对于 $\hat{\mathtt{Set}}$ 中的 (S, s) 和 (S', s')，它们之间的一个箭头是一个函数 $f : S \to S'$，满足 $f(s) = s'$

对于任意的 $F : I \to \mathtt{Set}$，有下面的 pullback square：

$$\begin{array}{ccc} \int_I F & \dashrightarrow & \hat{\mathtt{Set}} \\ {\scriptstyle \pi}\downarrow & \lrcorner & \downarrow{\scriptstyle \varpi} \\ I & \xrightarrow{F} & \mathtt{Set} \end{array}$$

至此，我们成功地将 fibration/family/宇宙间的所有联系拓展到了以范畴作为 index 的情况。

2.2 以范畴为 index 的范畴、grothendieck construction

下一步的拓展，是将被 index 的东西也从集合拓展到范畴。也就是说，我们将开始研究以范畴 index 的一个范畴的 family。宇宙依然是提供第一个定义的优秀起点。我们可以把一个以 I 为 index 的范畴的 family 表示成一个 functor $F : I \to \mathtt{Cat}$，其中 $\mathtt{Cat}$ 是全体（小）范畴与它们之间的 functor 构成的范畴（忽略集合/范畴大小问题）。沿着和前面的两个例子同样的顺序，接下来我们尝试构造 F 对应的 total space。

将这个 total space 依然记作 $\int_I F$。同之前一样，忽略所有箭头，$\int_I F$ 中的对象就是 $\Sigma_{i \in Ob(I)} F(i)$。接下来，考虑 I 中的一个箭头 $f : i \to i'$：

- 在 $\mathtt{Cat}$ 中，f 对应一个 **functor** $F(f) : F(i) \to F(i')$

- 对于 $\int_I F$ 中的一个对象 (i, a)，$F(f)$ 会产生一个新的对象 $(i', F(f)(a))$。所以，对于每个 $f \in I(i, i')$，在 $\int_I F$ 中应当有从 (i, a) 到 $(i', F(f)(a))$ 的箭头
- 但现在，每个 $F(i)$ 自身也是范畴，我们需要考虑它们中的箭头。对于 $i \in I$，$a \xrightarrow{\phi} a' \in F(i)$，在 $\int_I F$ 中也应当有从 (i, a) 到 (i, a') 的箭头
- 所以，$\int_I F$ 中的箭头似乎有两个来源。但幸运的是，上述两种定义都是一个更一般的定义的特例。对于 $\int_I F$ 中的 (i, a) 和 (i', a')，它们之间的一个箭头被定义为一个二元组 (f, ϕ)，其中 $f \in I(i, i')$ 是 I 中的箭头，ϕ 则是 $F(i')$ 中一个从 $F(f)(a)$ 到 a' 的箭头。可以用下面的图来可视化这一定义：

$$\begin{array}{cc} i & F(f)(a) \\ \downarrow{\scriptstyle f} & \downarrow{\scriptstyle \phi} \\ i' & a' \end{array}$$

- 一开始的两种箭头定义的确都是这个更一般的定义的特例。对于 $f \in I(i, i')$，$(f, \mathtt{id}_{F(f)(a)})$ 是一个从 (i, a) 到 $(i', F(f)(a))$ 的箭头。对于 $\phi \in F(i)(a, a')$，$(\mathtt{id}_i, \phi)$ 是一个从 (i, a) 到 (i, a') 的箭头
- $\int_I F$ 中箭头复合的定义如下。考虑 $(i_1, a_1) \xrightarrow{(f_1, \phi_1)} (i_2, a_2) \xrightarrow{(f_2, \phi_2)} (i_3, a_3)$，$(f_2, \phi_2) \circ (f_1, \phi_1)$ 定义为：

$$(f_2 \circ f_1, \phi_2 \circ F(f_2)(\phi_1))$$

于是，我们得到了以范畴为 index 的范畴 family 中，total space 的定义。上述 $\int_I F$ 被称为 *grothendieck construction*。

与前面的两个例子一样，接下来，我们考虑“以范畴为 index 的范畴”对应的“宇宙”：$\hat{\mathtt{Cat}} \xrightarrow{\varpi} \mathtt{Cat}$。把 $\mathtt{id}_{\mathtt{Cat}}$ 看作一个以 Cat 自身为 index 的 family，$\hat{\mathtt{Cat}}$ 就是 $\int_{\mathtt{Cat}} \mathtt{id}_{\mathtt{Cat}}$。把定义展开，可以得到：

- $Ob(\hat{\mathtt{Cat}}) = \Sigma_{\mathcal{C} \in \mathtt{Cat}} Ob(c)$，
- 在 $\hat{\mathtt{Cat}}$ 中，从 $(\mathcal{C}, c)$ 到 $(\mathcal{D}, d)$ 的一个箭头是一个二元组 (F, f)，其中 $F : \mathcal{C} \to \mathcal{D}$ 是一个 functor，$f : F(c) \to d$ 是 $\mathcal{D}$ 中的一个箭头

同样地，对于任意的 $I \xrightarrow{F} \mathtt{Cat}$，有下面的 pullback square：

$$\begin{array}{ccc} \int_I F & \dashrightarrow & \hat{\mathtt{Cat}} \\ {\scriptstyle \pi}\downarrow & \lrcorner & \downarrow{\scriptstyle \varpi} \\ I & \xrightarrow{F} & \mathtt{Cat} \end{array}$$

2.3 哪些 functor 才是 fibration?

在“以集合为 index 的集合”中，通过一个函数的 inverse image 的构造，我们发现每个指向 I 的函数都是一个 fibration，这些函数和以 I 为 index 的 family 是一一对应的。然而，在 $\mathtt{Cat}$ 中，这一点并不一定成立。事实上，并不是所有指向 I 的 functor 都是 fibration、都对应某个以范畴 I 为 index 的范畴/集合，于是，一个很自然的问题就是：哪些 functor 才是 fibration?

这里，我们直接考虑“以范畴为 index 的一堆范畴”的情况，“以范畴为 index 的一堆集合”是它的特例（把集合看作没有箭头的 discrete category）。给定一个 functor $p : E \to I$，我们尝试构造它的 inverse image $p^{-1} : I \to \mathtt{Cat}$，并观察哪一步会出现问题、要求 p 具有哪些性质可以解决这些问题。在构造过程中，除了直觉正确性之外，构造出的 p^{-1} 和 grothendieck construction 互逆也是一个重要的检验手段。碍于篇幅与大量的“技术细节”，这里不会严谨地证明这一互逆/等价性，只会用它做一个参考。下面开始逐步地构造 p 的 inverse image：

- 首先，对于 I 中的一个对象 i，考虑 $p^{-1}(i)$。它应当是一个范畴。参考集合上的 inverse image，应当有 $Ob(p^{-1}(i)) = \{e \in Ob(E) \mid p(e) = i\}$。我们可以用 grothendieck construction 来验证这一点：考虑一个 grothendieck construction $\int_I F \xrightarrow{\pi} I$：

$$\begin{aligned}\pi^{-1}(i) &= \{x \in Ob(\int_I F) \mid \pi(x) = i\} \\ &= \{(i, a) \mid a \in F(i)\} \\ &\cong Ob(F(i))\end{aligned}$$

  同样，p^{-1} 的 grothendieck construction 应该是 E 自身：

$$\begin{aligned}Ob(\int_I p^{-1}) &= \Sigma_{i \in I} Ob(p^{-1}(i)) \\ &= \{(i, e) \mid e \in Ob(E), p(e) = i\} \\ &\cong Ob(E)\end{aligned}$$

  所以，通过集合的 inverse image 来定义 $Ob(p^{-1}(i))$ 的确是合理的

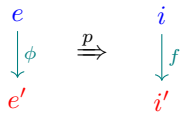
- 但每个 $p^{-1}(i)$ 都是范畴，它们除了对象集合还有箭头。考虑 $e_1, e_2 \in Ob(p^{-1}(i))$，它们之间的箭头应该是什么？不妨用 grothendieck construction 来作为引导。考虑 p 是 $\int_I F \xrightarrow{\pi} I$ 的情况，那么，每个 $\phi : a \to a' \in F(i)$ 都会对应 $\int_I F$ 中的一个箭头 $(\mathtt{id}_i, \phi)$，这个 $\int_I F$ 中的箭头又会被 π 送到 $\mathtt{id}_i$。所以，我们可以如此定义 $p^{-1}(i)$ 中的箭头：

$$\mathtt{Hom}_{p^{-1}(i)}(e_1, e_2) = \{\phi : e_1 \to e_2 \in E \mid p(\phi) = \mathtt{id}_i\}$$

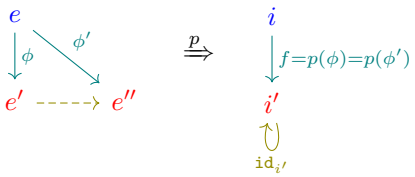
  不过，现在暂时无法通过对 p^{-1} 取 grothendieck construction 来恢复 E 中的所有箭头，因为不同的 $p^{-1}(i)$ 之间的 functor 还没有被定义出来

- 我们已经定义出了每个 $p^{-1}(i)$ （它们的确是范畴的验证留作极其简单的练习）。接下来，应该开始考虑 $p^{-1}(f)$，$f : i \to i' \in I$ 了。它应该是一个

由 $p^{-1}(i)$ 到 $p^{-1}(i')$ 的 functor。首先考虑 $p^{-1}(f)$ 这个 functor 在 object 上的定义。给定 $f : i \to i'$，$e \in p^{-1}(i)$，我们希望找到 $p^{-1}(i')$ 中的一个 e'，使得 $p^{-1}(f)(e) = e'$。e' 应该是由 e 和 f 共同确定的。所以，一个很自然的猜测是：e 和 e' 在 E 中是通过一个箭头 $\phi : e \to e'$ 联系起来的，而且 $p(\phi) = f$。以交换图表示的话：



- 但是，满足上述条件的 ϕ 可能有很多。我们必须从这之中选出一个最自然的 ϕ。什么是最自然的 ϕ 呢？我们希望 ϕ 可以用来表达所有其他的 ϕ'。如果对于其他每一个 ϕ'，ϕ' 都可以用 $\langle \texttt{bla} \rangle \circ \phi$ 的形式表达，那么 ϕ 就可以看作是一个最自然的选择。我们把这样的最自然的 ϕ 叫作 *weak cartesian morphism*。它的性质可以用交换图表示如下：

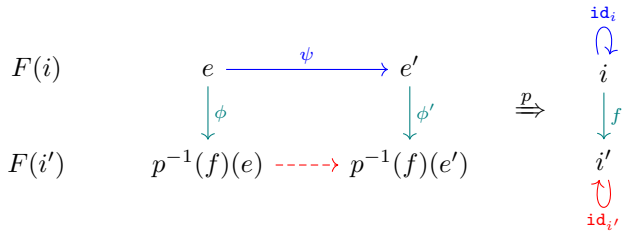


用文字表述的话，给定 $p : E \to I$、E 中的一个点 e 和一个 I 中的箭头 $f : p(e) \to i'$，$\phi : e \to e' \in E$ 是一个 weak cartesian morphism 意味着：

1. $p(\phi) = f$
2. 对于所有其他满足条件 1 的 ϕ'，存在一个唯一的 g 使得 $\phi' = g \circ \phi$

可以证明，weak cartesian morphism 如果存在，是 unique up to unique isomorphism 的。这意味着它不是严格唯一的。所以，定义 $p^{-1}(f)(e)$ 时，可能需要在许多不同的 weak cartesian morphism 中选择一个，并且可能需要选择公理

- 得到了 $p^{-1}(f)$ 在对象上的定义后，我们需要考虑它在箭头上的定义。给定 $\psi : e \to e' \in p^{-1}(i)$，可以用交换图画出已有的原料：

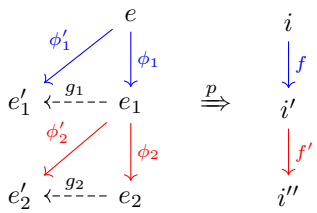


其中，虚线的箭头是我们希望填上的。然而，注意到 ϕ 是来自 e 的一个 weak cartesian morphism，而 $\phi' \circ \psi$ 同样是一个来自 e 的 morphism，而且

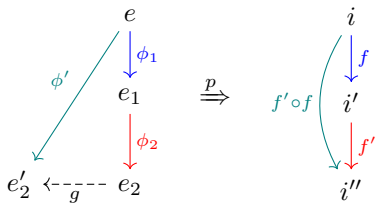
$p(\phi' \circ \psi) = p(\phi') \circ p(\psi) = f$（根据 $F(i)$ 中箭头的定义 $p(\psi) = \mathtt{id}_i$）。所以，根据 weak cartesian morphism 的定义，存在一个唯一的箭头 $p^{-1}(f)(e) \xrightarrow{g} p^{-1}(f)(e')$ 使得 $g \circ \phi = \phi' \circ \psi$：这正是我们想要的红色虚线箭头！

- 上面我们讨论了，对于一个具体的 $i \xrightarrow{f} i' \in C$，如何定义出它的 inverse image $p^{-1}(f)$：一个 functor。为了使 inverse image p^{-1} 成为定义在整个 I 上的一个 functor，我们要求对于每个 $e \in E$ 与 $p(e) \xrightarrow{f} i' \in I$，都存在一个 cartesian morphism $e \xrightarrow{\phi} e' \in E$，使得 $p(\phi) = f$。满足这种条件的 p 称为一个 weak grothendieck fibration

- 接下来，还需要验证 p^{-1} 满足 functor 的性质：保持单位箭头和复合。单位箭头都是 weak cartesian morphism，所以只需要再选择单位箭头对应的 weak cartesian morphism 时选取单位箭头即可。但 p^{-1} 保持复合的验证就没这么简单了考虑 $i \xrightarrow{f} i' \xrightarrow{f'} i'' \in I$ 与 $e \in p^{-1}(i)$，将 e、$e_1 = p^{-1}(f)(e)$、$e_2 = (p^{-1}(f') \circ p^{-1}(f))(e)$，以及它们是 weak cartesian morphism 的证明画入一张图中，有：



为了使 $p^{-1}(f') \circ p^{-1}(f) = p^{-1}(f' \circ f)$，$\phi_2 \circ \phi_1$ 应该也是一个 weak cartesian morphism：否则它就不可能成为 $p^{-1}(f' \circ f)(e)$ 的选择之一。然而，尽管 ϕ_1 和 ϕ_2 都是 cartesian morphism，它们的复合不一定是 weak cartesian morphism！考虑如下的配置：

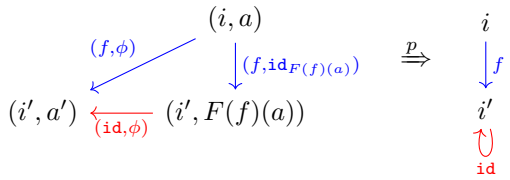


为了使 $\phi_2 \circ \phi_1$ 成为一个 weak cartesian morphism，我们需要找到图中的 g。为此，我们需要利用 ϕ_1、ϕ_2 都是 weak cartesian morphism 的性质。然而，它们的性质都不能直接使用，因为 $p(\phi') = f' \circ f$，即不是 f 也不是 f'！

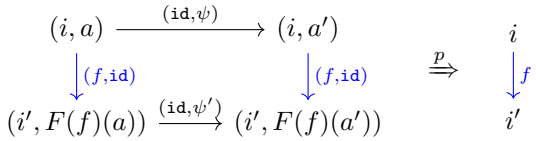
- 所以，为了使 p^{-1} 成为一个 functor，我们要求 E 中 weak cartesian morphism 的复合依然是 weak cartesian morphism。满足这种条件的 p 称为一个 grothendieck fibration（没有 weak 了！）。事实上，在一个 grothendieck fibration 中，每个 weak cartesian morphism 都满足一个更

强的条件，都是 cartesian morphism（没有 weak）。关于 cartesian morphism 的定义，这里不做讨论。感兴趣的读者可以参考 nLab 中对应的页面

- 终于，我们完成了 p^{-1} 的所有定义。最后，我们依然使用 grothendieck construction 来验证目前的定义。首先，假设 $p = \int_I F \xrightarrow{\pi} I$。考虑 $(i,a) \in \int_I F$、$i \xrightarrow{f} i' \in I$，$\pi^{-1}(f)$ 在 $\int_I F$ 中有一个很自然的选择，并且很容易验证它的确是一个 weak cartesian morphism：

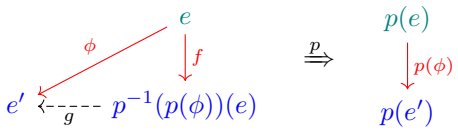


为了证明 π 是一个 grothendieck fibration，还需要证明 weak cartesian morphism 的复合依然是 weak cartesian morphism。用上面提到的 cartesian morphism 来证明这一点即可，感兴趣的读者可以当作练习。最后，注意到上面选取的 $\pi^{-1}(f)$ 会把每个 $a \in \pi^{-1}(i)$ 送到 $F(f)(a)$，对于 $\pi^{-1}(i)$ 中的箭头：



其中，根据 grothendieck construction 中箭头复合的定义，$\psi' = F(f)(\psi)$。所以 $\pi^{-1}(f)$ 就是 $F(f)$，对一个 grothendieck construction 取 inverse image 的确可以回到生成它的 functor 自身

- 下一步是考虑 inverse image 的 grothendieck construction。对于 E 中的一个箭头 $e \xrightarrow{\phi} e'$，令 $f = p^{-1}(p(\phi))$，有如下的交换图：



由于 f 是一个 cartesian morphism，存在唯一的 g，从而在 $\int_I p^{-1}$ 中有一个唯一的箭头 $(p(e), e) \xrightarrow{(f,g)} (p(e'), e')$，所以 E 中的箭头和 $\int_I p^{-1}$ 中的箭头是一一对应的！

这可能是这篇文档中最长的一节了。所以在漫长的推导过后，让我们回顾一下我们做了什么：

- 给定一个 functor $p : E \to I$，我们定义了 $p^{-1}(i)$ 对应的范畴（$i \in Ob(I)$）

- 对于每个 $f : i \to i' \in I$，我们需要定义出一个 functor $p^{-1}(f)$，为此，我们推导出了 p 需要满足一些性质，这些 p 称为一个 weak grothendieck fibration
- 为了让 p^{-1} 自身成为一个 $I \to \texttt{Cat}$ 的 functor，p 必须再满足一些其他性质，这些满足更多性质的 p 称为一个 grothendieck fibration
- 我们简单地验证了 grothendieck construction 和刚刚定义出的 inverse image 是一对互逆的操作

总结而言，我们**推导**出了哪些 functor 才可以算作 fibration，才会对应一个 family！

上面的推导过程与定义存在些许不严谨之处，虽然不影响理解，这里出于完整性考虑将它们列出：

- 本节中所有“两个箭头 isomorphic”的表述，指的都是 2-category 的 2-cell 意义上的
- 虽然 grothendieck fibration 保证了 weak cartesian morphism 的复合还是 weak cartesian morphism，并不是每个 inverse image 都是严格的 functor。在构造 inverse image 时，需要从许多 cartesian morphism 中选择一个，但 $p^{-1}(\texttt{id})$ 不一定真的是 $\texttt{id}$，$p^{-1}(f \circ g)$ 的选择也不一定真的是 $p^{-1}(f) \circ p^{-1}(g)$。我们不希望拒绝这些选择，因为不同的选择之间依然是 uniquely isomorphic 的。所以，我们把 inverse image 定义成一个 *pseudo functor*，pseudo functor 中 functor 需要满足的等式只在 isomorphism 意义上成立
- 事实上，inverse image 和 grothendieck construction 这对互逆的操作不仅在 family/fibration 上是互逆的，在 family 间的变换/fibration 间的变换上也是互逆的
- 在范畴论中，更多时候，我们关心的是形如 $I^{\mathrm{op}} \to \texttt{Cat}$ 的 family，此时上面的所有定义，包括 (weak) cartesian morphism 定义中的箭头方向需要反过来

3 Family 与 fibration，type theoretically I

终于，我们结束了数学上的讨论，可以进入类型论相关的内容了。在此之前，不妨回忆一下，本文档的核心内容是：

介绍 family/fibration 的联系，以及它在不同领域的体现与应用

而前面两节的内容，就是这一联系在集合与范畴上的应用。接下来的内容，则是关于这一联系在类型论，主要是 dependent type 上的应用。

3.1 从 STLC 到 dependent type

在进入正式的讨论之前，不妨复习一下类型系统与范畴论的联系。我们有著名的 Lambek Correspondence，Simply Typed Lambda Calculus（STLC）和 Cartesian Closed Category（CCC）是等价的。我们把类型系统中的 context 和类型翻译成 CCC 中的对象，把 substitution 和表达式翻译成 CCC 中的箭头。Context 的拓展/product type 对应范畴中的 product，函数类型对应范畴中的 exponential object。如果你不知道这部分内容，如文章开头所说过的那样，可以参考我写的 推导 Simply Typed Lambda Calculus 的范畴语义。

那么，如何将一个有 dependent type 的类型系统翻译到某个范畴中呢？Dependent type 比起 STLC 最大的不同，自然就是类型会依赖于值：除了 $\vdash A$ type 的 closed type 之外，还会有 $x : A \vdash B$ type 的 open type/type **family**（下面只考虑依赖一个 free variable 的 type family/open term，但这些定义都可以简单地拓展到多个 variable 的 context 和 substitution）。所以，我们需要面临的第一个考验就是如何将 type family 翻译到一个范畴中。除此之外，dependent type 中，判断一个 context 是否合法的规则也更复杂了：

$$\frac{\vdash \Gamma \text{ ctx} \qquad \Gamma \vdash A \text{ type}}{\vdash \Gamma, x : A \text{ ctx}}$$

所以，如何表示合法的 context extension 是另一个重要的挑战。这一节就将以**如何表示 type family 和如何定义 context extension** 为主线，展示 dependent type 范畴语义的各种框架。从此往后，将会假设一个以 context Γ, Δ 为对象，substitution $\Gamma \vdash \sigma : \Delta$ 为 $\Gamma \to \Delta$ 的箭头的范畴 $\mathtt{Ctx}$。其中，$\sigma : \Gamma \to \Delta$（$\Gamma \vdash \sigma : \Delta$）的意义是，$\sigma$ 的 domain 为 Δ，σ 产生的表达式需要在 Γ 中取得正确的类型。

3.2 全部使用 family：type category 与 CwF

首先，不妨依然采用最直观的构造方式：把 type family 直接构造成 family。在每个 context 下，都应该有对应的类型的集合 $\{A \mid \Gamma \vdash A \text{ type}\}$，记作 $\mathtt{Ty}(\Gamma)$。把一个 $\Gamma \xrightarrow{\sigma} \Delta$ 的 substitution 作用在一个类型 $\Delta \vdash A$ type 上时，会得到一个新的类型 $\Gamma \vdash A\sigma$ type。所以，σ 会对应一个 $\mathtt{Ty}(\Delta) \to \mathtt{Ty}(\Gamma)$ 的函数，因此，$\mathtt{Ty}$ 是一个 $\mathtt{Ctx}^{\mathrm{op}} \to \mathtt{Set}$ 的 functor。

那么，context extension 要怎么处理呢？由于 $\mathtt{Ty}$ 是单向的，这里只能暴力地把 context extension 定义为一个单独的操作 $- \cdot -$，使得如果 $\Gamma \in \mathtt{Ctx}$，$A \in \mathtt{Ty}(\Gamma)$，那么 $\mathtt{Ctx}$ 中有另一个 context $\Gamma \cdot A$。新的 context 与 $\Gamma \cdot A$ 与其他 context 之间的互动，例如忘记新的 A 的 weakening substitution $\Gamma \cdot A \to \Gamma$，也需要单独指定。

沿着这一思路定义出来的范畴模型，就叫作 *type category*。如果进一步地，对于每个 $A \in \mathtt{Ty}(\Gamma)$，定义一个集合 $\mathtt{Tm}(\Gamma, A)$，代表 $\{t \mid \Gamma \vdash t : A\}$ 的 well-typed terms 的集合，那么就得到了 *Category with Family (CwF)*。

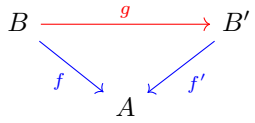
3.3 使用 fibration 其一：LCCC 与 display map catgeroy

采用直接的 family 定义出的 type category/CwF 虽然非常直观，但它们的 context extension 操作比较暴力，与 context 的范畴 `Ctx` 自身性质的联系不是那么直接。能否通过要求 `Ctx` 具有某种性质，使得各个 $\mathtt{Ty}(\Gamma)$ 可以在 `Ctx` 自身中得到表示呢？

$\Gamma \vdash A$ type 常常被叫作一个 type **family**，因为把 Γ 中 variable 替换成不同的表达式，就能得到不同的类型。但是，在 $\mathcal{T}$ 中并不存在一个表示“所有类型的类型”的宇宙的对象，所以我们不能把 A 表示成一个 $\Gamma \to \mathcal{U}$ 的函数。那么，我们能不能把 A 表示成一个 $\bullet \to \Gamma$ 的 **fibration** 呢？根据集合论中的直觉，这个 fibration 的 domain 可以取 A 的“total space”，$\Sigma_{\langle \mathrm{bla} \rangle : \Gamma} A$。但是，在一个 dependent type 的 context $\Gamma, x : A$ 中，“逗号”的意义和 dependent sum Σ 是一样的！所以，$\Sigma_{\langle \mathrm{bla} \rangle : \Gamma} A$ 就是 $\Gamma, x : A$，通过把 type **family** 看作 **fibration**，我们在 `Ctx` 自身当中同时得到了 type family 和 context extension！

那么，一个很自然的问题是，哪些箭头、哪些 substitution 才是 fibration，才能表示一个 type family 呢？最简单的情况下，如果每个箭头都是 fibration，那么“全体指向 Γ 的箭头”就是“所有 Γ 下的合法类型”。而通过一个叫 slice category 的构造，我们就能得到一个以这些箭头为对象的范畴 $\mathtt{Ctx}_{/\Gamma}$，它代表“假设了 Γ 后的世界”、“相对于 Γ 的世界”。具体来说，给定一个范畴 $\mathcal{C}$ 和 $\Gamma \in \mathcal{C}$，slice category $\mathcal{C}_{/A}$ 的定义如下：

- $\mathcal{C}_{/A}$ 中的对象是 $\mathcal{C}$ 中指向 A 的箭头 $\bullet \to A$
- 对于 $B \xrightarrow{f} A$、$B' \xrightarrow{f'} A \in \mathcal{C}_{/A}$，一个 $\mathcal{C}_{/A}$ 中从 f 到 f' 的箭头是一个 $\mathcal{C}$ 中的箭头 $B \xrightarrow{g} B'$，满足：



上面的交换图中，蓝色的是 $\mathcal{C}_{/A}$ 中的对象。红色的是 $\mathcal{C}_{/A}$ 中的箭头

对于 context 的范畴 `Ctx` 中的一个 context Γ，$\mathtt{Ctx}_{/\Gamma}$ 的对象、看作 fibration，就是 Γ 中的一个类型。$\mathtt{Ctx}_{/\Gamma}$ 中一个从 $\Gamma, x : A \xrightarrow{\pi_A} \Gamma$ 到 $\Gamma, x : B \xrightarrow{\pi_B} \Gamma$ 的箭头，就是 `Ctx` 中的一个 substitution $\Gamma, x : A \xrightarrow{\sigma} \Gamma, x : B$。但我们额外要求 $\sigma \circ \pi_B = \pi_A$，所以 σ 对于 context 中 Γ 的部分不做改动：它是一个 Γ-generic 的、相对于 Γ 的 substitution！

STLC 对应的是 CCC，CCC 意味着 function type 和 product type 存在。而如果在 `Ctx` 的每个 slice category $\mathtt{Ctx}_{/\Gamma}$ 也都是 CCC，就能推导出 dependent product 和 dependent function 的存在（具体的讨论留待下一节）！如果一个范畴不仅是一个 CCC，而且它的每个 slice category 也是 CCC，那么这样的范畴称为一个 **Locally** Cartesian Closed Category (**L**CCC)。LCCC 是类型论范畴

语义最为简单的模型之一，与 topos 等其他数学与范畴论中的领域也有很多联系。

尽管非常简单，LCCC 有一个致命的缺陷，这一缺陷来自于用它作为类型系统语义的一个大前提：每个箭头都是一个 fibration。考虑一个 substitution $\Gamma \xrightarrow{f} \Delta$，简单起间，我们考虑 Γ、Δ 都只有一个变量的情况，那么此时 $x : A \xrightarrow{f} y : B$ 等价于一个 $A \xrightarrow{f} B$ 的 non-dependent function。如果任意的 f 都是 fibration，我们就需要从 f 中构造出一个 type family。如同之前一样，我们尝试使用 inverse image 的办法来构造这个 family。在集合论中，一个函数 $f : A \to B$ 的 inverse image 是：

$$f_b^{-1} = \{a \in A \mid f(a) = b\} \qquad b \in B$$

如果我们想要在类型论中模仿这一定义，就需要表达出 $f(a) = b$ 这一条件，所以，使用 LCCC 作为类型论语义的第一个条件是：类型系统中必须存在 identity type/propositional equality $t = u : A$。在有 identity type 的情况下，我们可以从任意 f 中构建出一个“inverse image”类型：

$$y : B \vdash \left(f^{-1} :\equiv \Sigma_{x:A}(f(x) = y : B)\right) \text{ type}$$

接下来，我们还需要验证 `Ctx` 中的箭头和 type family 的确是通过 fibration 与 inverse image 等价的。依然考虑 context 中只有一个变量的简单情况，一个 type family $x : A \vdash B$ type 可以用 fibration $\Sigma_{x:A} B \xrightarrow{\pi_B} A$ 来表示，我们有：

$$x : A \vdash \pi_B^{-1} : \sum_{p:\Sigma_{x:A} B} (\pi_B(p) = x : A)$$

但是 π_B^{-1} 和 B 不是严格相等的。即使想要证明它们等价，我们还需要证明 $\pi_B(p) = x : A$ 这一 propositional equality 的证明是“无关紧要”的，不会携带额外的数据：所以我们还需要（definitionally）proof irrelevant 的 identity type。这使得 LCCC 只能被用来描述特定的一类 extensional type theory ——而它们的 type checking 一般都是 undecidable 的！

所以，如果想要通过 fibration 来表达 type family 与 context extension，就必须考虑这么一个问题：

> *哪些箭头才是 fibration?*

我们可以要求 `Ctx` 中有一类特殊的箭头，称为 display map，只有 display map 才是 fibration、才会对应 type family。如此一来，就可以避免 LCCC 对 extensional identity type 的依赖。通过只对 display map 要求 LCCC 中的各种性质，同样可以得到 dependent product/sum 等构造。通过这种方式得到的范畴，就叫作 *display map category*。

3.4 fibration 的应用其二：comprehension category

上面我们看到了类型论的四种范畴语义。然而，上面的四种语义在“每个 context 下的类型们”这件事情上，都采用的是 family 的定义方法：

- type category/CwF 中，每个 context 下的类型们是通过一个 functor $\mathtt{Ty} : \mathtt{Ctx}^{\mathrm{op}} \to \mathtt{Set}$ 表示的
- 在 LCCC 中，每个 context Γ 下的类型们其实也是用一个 functor $\mathtt{Ty} : \mathtt{Ctx}^{\mathrm{op}} \to \mathtt{Cat}$ 定义的。只不过这个 functor 是一个具体的定义，它会把每个对象 $\Gamma \in \mathtt{Ctx}$ 送到它的 slice category $\mathtt{Ctx}_{/\Gamma}$ （Ctx 中的 substitution 在类型上的作用留待下一节）
- display map category 的情况与 LCCC 类似，只不过 Ty 会把 context 送到某个只包含 display map 的范畴，而非完整的 slice category

那么，$\mathtt{Ctx}^{\mathrm{op}} \xrightarrow{\mathtt{Ty}} \langle \mathrm{bla} \rangle$ 本身，是否也可以表示为一个 fibration 呢？从前面的讨论我们知道，通过 grothendieck construction 就可以把任何一个 family 变成一个 fibration。$\int_{\mathtt{Ctx}} \mathtt{Ty}$ 中的对象形如 $\Gamma \vdash A$：一个 context Γ 和 Γ 中的一个类型 A。而 $\int_{\mathtt{Ctx}} \mathtt{Ty}$ 中的箭头，则形如 $\Gamma \vdash A\sigma \xrightarrow{\sigma} \Delta \vdash A$（箭头的方向是反的，这是为了与 substitution 在 Ctx 中的方向一致），其中 $\Gamma \vdash \sigma : \Delta$。反过来，我们也可以抽象地用一个 grothendieck fibration $p : \mathcal{E} \to \mathtt{Ctx}$ 来代替 Ty，而 grothendieck fibration 的性质保证了 Ctx 中的 substitution 的确会对应 $\mathcal{E}$ 中的“类型”上的 substitution！

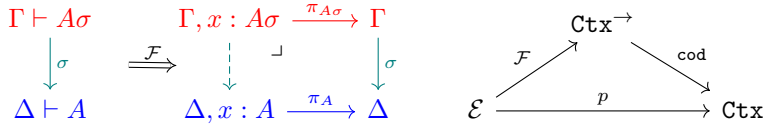
那么，fibration $\mathcal{E} \xrightarrow{p} \mathtt{Ctx}$ 要如何与 Ctx 互动呢？这里，不妨借用 LCCC/display map category 中的思路，把 $\mathcal{E}$ 中的类型送到 Ctx 中它们的 display map。方便起见，假设 $\mathcal{E} = \int_{\mathtt{Ctx}} \mathtt{Ty}$，我们希望将 $\mathcal{E}$ 中的每个 $\Gamma \vdash A$ 送到 Ctx 中的一个箭头 $\Gamma, x : A \vdash \Gamma$。为了将对象送到箭头，我们需要借助 arrow category 的概念。给定一个范畴 $\mathcal{C}$，它的 arrow category $\mathcal{C}^{\to}$ 定义如下：

- $\mathcal{C}^{\to}$ 中的对象是 $\mathcal{C}$ 中的箭头
- 给定 $A \xrightarrow{f} B$、$C \xrightarrow{f} D \in \mathcal{C}$，它们都是 $\mathcal{C}^{\to}$ 中的对象。$\mathcal{C}^{\to}$ 中从 f 到 g 的箭头是两个 $\mathcal{C}$ 中的箭头 (h, h')，使得：

$$\begin{array}{ccc} A & \xrightarrow{f} & B \\ \downarrow{\scriptstyle h} & & \downarrow{\scriptstyle h'} \\ C & \xrightarrow{g} & D \end{array}$$

交换。图中蓝色的是 $\mathcal{C}^{\to}$ 中的对象，红色的是 $\mathcal{C}^{\to}$ 中的箭头

有两个显然的 functor $\mathtt{dom}, \mathtt{cod} : \mathcal{C}^{\to} \to \mathcal{C}$，它们把每个箭头送到它的 domain/codomain。我们想把 $\mathcal{E}$ 中的每个 $\Gamma \vdash A$ 送到 Ctx 中的箭头，这可以表示为一个 functor $\mathcal{E} \xrightarrow{\mathcal{F}} \mathtt{Ctx}^{\to}$。我们希望每个 $\Gamma \vdash A$ 会被送到一个 $\Gamma, x : A \to \Gamma$ 的箭头，所以我们还要求 $\mathtt{cod} \circ \mathcal{F} = p$。最后，给定一个 $\Gamma \vdash A$，display map 的 domain 就是它产生的 extended context $\Gamma, x : A$，所以 $\Gamma, x : A$ 就是 $(\mathtt{dom} \circ \mathcal{F})(\Gamma \vdash A)$！我们可以把上述结构用如下交换图概括：



这种结构叫作 *comprehension category*。注意到上述交换图要求 $\mathcal{E}$ 中的箭头（substitution）会被送到 `Ctx` 中的 pullback，这一要求的意义将在下一节解释。

4 Family 与 fibration，type theoretically II

上一节中，我们粗略地介绍了 dependent type 的几种常见的范畴模型。这一节中，我们将具体地把 dependent type 中的基本构造翻译到范畴的语言。本节中使用的模型将会是 display map category 或者 comprehension category，因为本节中会使用 display map/fibration 表示 type family，但不会设计具体模型的太多技术细节。下面，当我说 $\Gamma, x : A \xrightarrow{\pi_A} \Gamma$ 是一个 type family/fibration 时，我指的是它是 `Ctx` 中的一个 display map（在 display map category 中），或者存在某个 $e \in \mathcal{E}$ 使得 $\mathcal{F}(e) = \pi_A$（在 comprehension category）中。

为了记号的简洁性，我不会区分一个类型系统中的表达式/类型和它的范畴翻译。但是，在范畴语言中的表达式/类型/context 都不会带有 free variable。例如，$\Gamma, x : A$ 会被写成 Γ, A。这是因为范畴语言中没有 free variable 的概念，也可以看作是在使用 de Brujin index 一类的没有 free variable 的表达式记法。

4.1 terms as sections

我们已经找到了类型系统中大部分构造的范畴对应：

- context 是 `Ctx` 中的对象
- substitution 是 `Ctx` 中的箭头
- 类型是 `Ctx` 中的 fibration

那么，剩下的就是表达式 $\Gamma \vdash t : A$ 了。我们要如何把它们翻译到 `Ctx` 中呢？在 STLC 中，一个表达式会被翻译成一个 $\Gamma \xrightarrow{t} A$ 的箭头。然而，在 dependent type 中，A 是一个 type family，它并不直接对应 `Ctx` 中的对象，而是对应一个 fibration $\Gamma, x : A \xrightarrow{\pi} \Gamma$。

所以，STLC 中表达式的表示方法在 dependent type 中不再适用了。但是，在介绍 LCCC 时我曾经提到，通过取 slice category，我们可以得到“相对于 Γ 的世界”。而 type family $\Gamma \vdash A$ type 则会成为 $\mathtt{Ctx}_{/\Gamma}$ 中的一个对象，这时我们就可以尝试使用 STLC 中的表示法了。在“相对于 Γ 的世界”$\mathtt{Ctx}_{/\Gamma}$ 中，t 会变成一个“closed”term，而 $\mathtt{Ctx}_{/\Gamma}$ 中的 terminal object 是 $\mathtt{id}_\Gamma$，所以 t 可以翻译成 $\mathtt{Ctx}_{/\Gamma}$ 中 $\mathtt{id}_\Gamma \to \pi_A$ 的一个箭头。把这个定义翻译回 `Ctx`，一个表达式 $\Gamma \vdash t : A$ 在 `Ctx` 中将对应：

> 一个 $\Gamma \to \Gamma, x : A$ 的箭头，满足 $\pi_A \circ t = \mathtt{id}_\Gamma$

这种箭头称为 π_A 的 *section*。在 `Ctx` 自身中，这个定义的意义是什么呢？首先，我们可以把 t 变成一个 substitution $\Gamma \mapsto \Gamma, x \mapsto t$，这是一个从 $\Gamma \to \Gamma, x : A$ 的 substitution。但是，并不是所有这样的 substitution 都对应一个 Γ 下的 term——所以我们要求 t 不能改动 Γ 中的内容，也就是 $\pi_A \circ t = \mathtt{id}$、$t$ 是 π_A 的 section。

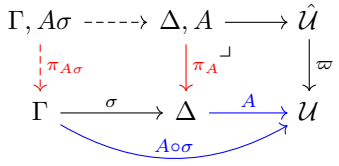
4.2 substitutions as pullback

Ctx 中的箭头是 substitution，那么，应该怎么将 substitution 应用在类型和表达式上呢？我们需要翻译类型系统中对应的规则/substitution lemma：

$$\frac{\Delta \vdash A \text{ type} \qquad \Gamma \vdash \sigma : \Delta}{\Gamma \vdash A\sigma \text{ type}} \qquad\qquad \frac{\Delta \vdash t : A \qquad \Gamma \vdash \sigma : \Delta}{\Gamma \vdash t\sigma : A\sigma}$$

首先考虑前者。翻译到范畴中后：$\Delta, x : A \xrightarrow{\pi_A} \Delta$ 是一个 fibration，而 $\sigma : \Gamma \to \Delta$：两者没有办法直接复合！原因在于，$A$ 使用了 fibration 的方式来表达，因此，A 的 context 变成了 A 的 codomain，而非 domain！

为了解决这个问题，我们不妨暂时假设 Ctx 中存在一个“类型的宇宙” $\hat{\mathcal{U}} \xrightarrow{\varpi} \mathcal{U}$，这么一来，$A$ 就可以表示为一个 $\Delta \to \mathcal{U}$ 的箭头，从而 $A\sigma$ 就可以直接计算为 $A \circ \sigma$ 了！我们还需要把 A 利用宇宙的表示方法和利用 fibration 的表示方法联系起来。不过，在1.3中我们已经看到了，fibration 和 universe 的表示法可以通过 pullback 联系起来！把各种原料放在一起：



图中，蓝色的箭头是使用宇宙表示的 type family，红色的箭头是使用 fibration 表示的 type family。π_A 是由 A 和 ϖ 生成的 pullback，而 $\pi_{A\sigma}$ 则是由 $A \circ \sigma$ 和 ϖ 生成的 pullback。所以，上图中右侧的小方格和外面的大方格都是 pullback，而根据 pullback 的性质，这**等价**于左侧、右侧的小方格都是 pullback！所以，只需要对 π_A 与 σ 取 pullback，或者说把 π_A 沿着 σ “拉回”，就能得到 $\pi_A\sigma$！所以，借助于外部的宇宙，我们得到了 Ctx 中 substitution 作用于 type family 的方法：pullback。即使宇宙 $\hat{\mathcal{U}} \xrightarrow{\varpi} \mathcal{U}$ 在 Ctx 中不存在，这一定义也依然是成立的：可以把它看成不在 Ctx 中的宇宙的一部分性质在 Ctx 中的“投影”。

事实上，我们可以从一个更大的角度来考察 substitutions as pullback 这个事实。在前面的讨论中，我们提到过，每种类型论的模型都在用不同的方式表达一个 family $\mathtt{Ty} : \mathtt{Ctx} \to \langle \text{bla} \rangle$，而 substitution $\sigma : \Gamma \to \Delta$ 会对应 $\mathtt{Ty}(\Delta) \to \mathtt{Ty}(\Gamma)$ 的函数/functor。在 displaymap category/comprehension category 中，由于采用了 display map 来表示 type family，$\mathtt{Ty}(\Gamma)$ 实际上对应着 $\mathtt{Ctx}_{/\Gamma}$（取只有 type family 的 sub-category）。所以，一个 substitution 对应着 $\Gamma \xrightarrow{\sigma} \Delta$ 应当一个 functor $\sigma^* : \mathtt{Ctx}_{/\Delta} \to \mathtt{Ctx}_{/\Gamma}$。当 Ctx 中有 pullback 时，$\sigma^*$ 可以定义为所谓的 pullback functor，它把每个箭头沿着 σ “拉回”。但是要求 Ctx 中有任意的 pullback 和 LCCC 一样，需要 identity type。但由于我们使用了 display map/comprehension category 来指明“哪些箭头是 fibration”，我们只需要要求这些 fibration 上存在任意的 pullback 即可。在 display map category 中，这表现为 display map 上有任意的 pullback。在 comprehension category 中，这表现为 $\mathcal{E} \xrightarrow{\mathcal{F}} \mathtt{Ctx}^{\to}$ 会把 $\mathcal{E}$ 中的每个箭头（substitution）送到一个 pullback。

一些 reference 与 further reading

- 无所不有的nLab，和本文内容比较有关的页面有 grothendieck construction、grothendieck fibration、dependent type 的各种范畴语义 dependent type 的各种范畴语义、LCCC、dependent sum、dependent product
- 另一份类型论范畴语义的介绍，从最基础的范畴论开始，主要讲了 comprehension categories，但也有涉及其他模型。结构更紧凑，定义非常全

Acknowledgement

感谢 Xu Huang，他教了我 inverse image 这一重要直觉，从而直接导致了这篇文章的诞生。感谢 Xu Huang 和 Tesla Zhang 的 proof reading 和反馈。