《前沿计算研究实践》报告

题目： 信息熵的若干性质探究、Lambda Calculus 简介、构造主义与 coq 语言简介

学号　　姓名: LWLAymh

二〇二五年 十二 月

目　录

第一章　前言

1.1　结构说明

我选取了" 信息" 作为主要重点阅读, 尝试以论文格式写了摘要 (见下页), 并在 [二] 中给出了一些信息熵的更详细的性质. 此外, 我选取了" 计算" 作为一个基本概念, 将在 [三] 中介绍一种图灵完备的语言:Lambda Calculus; 以此为基础, 我将以" 逻辑" 作为另一个基本概念, 并在 [四] 中介绍构造主义逻辑以及在此上面搭建的证明器 coq 语言.

本篇论文有相当一部分参考了我的博客[1][2].

1.2　有关 AIGC 的使用情况

本人在完成本项作业的过程中，未使用任何 AIGC 工具。本人确认，作业中的核心观点、论证逻辑、数据分析及最终文本均由本人独立完成，并对所有内容的原创性、准确性及可验证性负全部责任。本人知晓，将非本人完成的作品（包括由 AI 生成或实质性修改的作品）声称为自己的原创成果，将被视为学术不端行为，并将依据北京大学相关规定严肃处理。

[1] https://www.luogu.com.cn/user/661076/article
[2] https://lwlaymh.github.io/

第二章　信息熵的若干性质探究

作为一篇简单介绍信息熵的文章, 我们将以离散随机变量为基础, 探讨信息熵在尾不等式, 信道, 以及一些交互性的猜测游戏中的用处, 致力于给出一个关于信息熵的较为详细的提纲.

2.1　常用数学工具

在介绍信息熵之前, 我们首先需要介绍一些在研究过程中所常见的数学工具.

首先是函数 $f(x) = x\log x$, 只需对此求导就可以得知其是下凸函数, 此时琴生不等式给出:

$$f(\sum_i p_i x_i) \leq \sum_i p_i f(x_i), \text{when } \sum_i p_i = 1 \tag{2.1}$$

另一个很重要的工具是**对数求和不等式**. 对于任何非负实数 $a_1, \cdots, a_n$ 和正数 $b_1, \cdots, b_n$, 记 $a = \sum_i a_i, b = \sum_i b_i$, 则:

$$\sum_i a_i \log \frac{a_i}{b_i} \geq a \log \frac{a}{b} \tag{2.2}$$

其证明策略如下:

$$\begin{aligned}
\sum_i a_i \log \frac{a_i}{b_i} &= \sum_i b_i \frac{a_i}{b_i} \log \frac{a_i}{b_i} \\
&= \sum_i b_i f(\frac{a_i}{b_i}) \\
&= b \sum_i \frac{b_i}{b} f(\frac{a_i}{b_i}) \\
&\geq b f(\sum_i \frac{b_i}{b} \frac{a_i}{b_i}) \\
&= b f(\frac{a}{b}) \\
&= a \log \frac{a}{b}
\end{aligned}$$

2.2　信息熵的定义

离散情况下将熵定义为 $H[X]=E(\log\frac{1}{P(X)})=\sum_i P_i\log\frac{1}{P_i}$. 如果设 $|X|=|\{x|P(x)>0\}|$, 则容易见到 $0\le H[X]\le\log|X|$. 信息熵实际上是上凸函数, 对于任意分布 P,Q 和 $\lambda\in(0,1)$, 都有:

$$H[\lambda P+(1-\lambda)Q]\ge\lambda H[P]+(1-\lambda)H[Q] \tag{2.3}$$

原因是考虑:

$$f(\lambda p_i+(1-\lambda)q_i)\le\lambda f(p_i)+(1-\lambda)f(q_i)$$

接下来我们定义条件熵, 仿照信息熵, 考虑定义 $H[Y|X]=E(\log\frac{1}{P(Y|X)})$, 此时会有以下性质, 它们表明了条件熵 $H[Y|X]$ 其实可以理解为"(X,Y) 的信息去掉 X 的信息" 的结果:

$$H[Y|X]=H[X,Y]-H[X] \tag{2.4}$$

$$H[Y|X]\le H[Y] \tag{2.5}$$

$$H[Y|X]=\sum_x H[Y|X=x]P(X=x) \tag{2.6}$$

对于2.4, 我们有:

$$\begin{aligned}
H[Y|X]&=E(\log\frac{1}{P(Y|X)})\\
&=\sum_{x,y}P(Y=y,X=x)\log\frac{1}{P(Y=y|X=x)}\\
&=\sum_{x,y}P(Y=y,X=x)\log\frac{P(X=x)}{P(Y=y,X=x)}\\
&=\sum_{x,y}P(Y=y,X=x)\log\frac{1}{P(Y=y,X=x)}-\sum_x\log\frac{1}{P(X=x)}\sum_y P(X=x,Y=y)\\
&=H[X,Y]-H[X]
\end{aligned}$$

对于2.5我们有:

$$
\begin{aligned}
&H[X]-H[X|Y]\\
=&\sum_x\left(P(X=x)\log\frac{1}{P(X=x)}-\sum_y P(X=x,Y=y)\log\frac{1}{P(X=x|Y=y)}\right)\\
=&\sum_x\left(\sum_y P(X=x,Y=y)\log\frac{P(X=x|Y=y)}{P(X=x)}\right)\\
=&\sum_x\sum_y P(X=x,Y=y)\log\frac{P(X=x,Y=y)}{P(X=x)P(Y=y)}
\end{aligned}
$$

不妨令 $a_i=P(X=x,Y=y), b_i=P(X=x)P(Y=y)$. 容易见到 $a=\sum_i a_i=1, b=\sum_i b_i=1$. 于是上式变为:

$$
\begin{aligned}
&H[X]-H[X|Y]\\
=&\sum_i a_i\log\frac{a_i}{b_i}\\
=&-C\sum_i a_i\ln\frac{b_i}{a_i}\\
\geq&-C\sum_i a_i(\frac{b_i}{a_i}-1)\\
=&-C\sum_i(b_i-a_i)\\
=&0
\end{aligned}
$$

对于2.6则是:

$$
\begin{aligned}
H[Y|X]&=E(\log\frac{1}{P(Y|X)})\\
&=\sum_{x,y}P(Y=y,X=x)\log\frac{1}{P(Y=y|X=x)}\\
&=\sum_x P(X=x)\sum_y P(Y=y|X=x)\log\frac{1}{P(Y=y|X=x)}\\
&=\sum_x H[Y|X=x]P(X=x)
\end{aligned}
$$

我们还需要一条性质, 这条性质展示了对于一个确定性函数 g, 信息在经过它后不会增多, 有:

$$H[X] \geq H[g(X)] \tag{2.7}$$

上式取等当且仅当 g 是单射, 它的原因是:

$$\begin{aligned} H[X, g(X)] &= H[g(X)] + H[X|g(X)] \\ H[X] &= H[g(X)] + H[X|g(X)] \end{aligned}$$

2.3 互信息

定义互信息 $I(X;Y) = H[X] + H[Y] - H[X,Y]$. 根据2.4, 这正等于 $H[X] - H[X|Y]$, 又根据2.5, 可以得知总有 $I(X;Y) \geq 0$. 在此基础上定义 $I(X;Y \mid Z) = H[X|Z] - H[X|YZ]$, 注意: 它实际上在说的是 Z 条件下,X 和 Y 的互信息, 而不是 X 和 $Y|Z$ 的互信息. 我们不厌其烦地在下面罗列其性质:

$$I(X;YZ) = I(X;Z) + I(X;Y \mid Z) \tag{2.8}$$

$$I(X;Y \mid Z) \leq I(X;Y) + H(Z) \tag{2.9}$$

对于2.8, 留神到 $H[X] = I(X;Z) + H[X|Z]$, 考虑:

$$\begin{aligned} I(X;YZ) &= H[X] - H[X|YZ] \\ &= I(X;Z) + H[X|Z] - H[X|YZ] \\ &= I(X;Z) + I(X;Y \mid Z) \end{aligned}$$

对于2.9, 考虑:

$$\begin{aligned} I(X;YZ) &= I(X;Z) + I(X;Y \mid Z) \\ &= I(X;Y) + I(X;Z \mid Y) \\ I(X;Y \mid Z) &= I(X;Y) + I(X;Z \mid Y) - I(X;Z) \end{aligned}$$

然而 $I(X;Z \mid Y) = H[Z|Y] - H[X|YZ] \le H[Z]$, 而 $I(X;Z) \ge 0$, 于是显然.

最后我们要证明互信息的 data-processing 不等式: 当 X, Y, Z 满足 Markov 规则, 或者说 $P_{X,Y,Z} = P_X P_{Y|X} P_{Z|Y}$, 或说 $P_{Z|Y} = P_{Z|XY}$, 则:

$$I(X;Y) = I(X;Z) + I(X;Y|Z) \tag{2.10}$$

考虑:

$$\begin{aligned} I(X;YZ) &= I(X;Y) + I(X;Z \mid Y) \\ I(X;YZ) &= I(X;Z) + I(X;Y \mid Z) \end{aligned}$$

此外:

$$\begin{aligned} I(X;Y) + I(X;Z \mid Y) &= H[X] - H[X|Y] + H[Z|Y] - H[Z|YX] \\ &= H[X] - H[X|Y] + H[Z|Y] - H[Z|Y] \\ &= I(X;Y) \end{aligned}$$

于是 $I(X;Z \mid Y) = 0$, 这就证毕.

2.4 KL 散度

定义 $D(P||Q) = \sum_x P(x) \lg \frac{P(x)}{Q(x)}$. 其中如果 $P(x) \neq 0$ 而 $Q(x) = 0$ 的情况出现, 我们就说此时其为 $+\infty$. 我们想要证明:$D(P||Q) \ge 0$. 考虑:

$$\begin{aligned} D(P||Q) &= E_{x\sim Q}(\frac{P(X)}{Q(X)} \lg \frac{P(X)}{Q(X)}) \\ &\ge f\left(E_{x\sim Q}(\frac{P(X)}{Q(X)})\right) \\ &= f(1) \\ &= 0 \end{aligned}$$

从而这的确是某种衡量偏离程度的算子.

此外还应当定义条件 KL 散度. 考察:

$$\begin{aligned}
D(P_{X,Z}||Q_{X,Z}) &= \sum_{(x,z)} P_{X,Z}(x,z)\log\frac{P_{X,Z}(x,z)}{Q_{X,Z}(x,z)} \\
&= \sum_{(x,z)} P_{Z|X}(z|x)P_X(x)\log\frac{P_{Z|X}(z|x)P_X(x)}{Q_{Z|X}(z|x)Q_X(x)} \\
&= D(P_X||Q_X) + \sum_{(x,z)} P_{Z|X}(z|x)P_X(x)\log\frac{P_{Z|X}(z|x)}{Q_{Z|X}(z|x)}
\end{aligned}$$

将后面的部分定义为 $D(P_{Z|X}||Q_{Z|X} \mid P_X)$, 于是:

$$D(P_{X,Z}||Q_{X,Z}) = D(P_X||Q_X) + D(P_{Z|X}||Q_{Z|X} \mid P_X) \tag{2.11}$$

顺便应该有 $D(P_{X,Z}||Q_{X,Z}) \geq D(P_X||Q_X)$.

此外还应当证明 KL 散度凸性. 对于概率分布对 $(P_1,Q_1),(P_2,Q_2)$, 以及任意 $\theta \in [0,1]$, 令 $P = \theta P_1 + (1-\theta)P_2, Q = \theta Q_1 + (1-\theta)Q_2$. 事实上还有下凸的性质:

$$D(P||Q) \leq \theta D(P_1||Q_1) + (1-\theta)D(P_2||Q_2) \tag{2.12}$$

下面的定理将 KL 散度和互信息统一了起来:

$$I(X;Y) = D(P_{XY}||P_XP_Y) \tag{2.13}$$

因为:

$$\begin{aligned}
D(P_{XY}||P_XP_Y) &= E[\log\frac{P_{X,Y}(X,Y)}{P_X(X)P_Y(Y)}] \\
&= -E[\log\frac{1}{P_{X,Y}(X,Y)}] + E[\log\frac{1}{P_X(X)}] + E[\log\frac{1}{P_Y(Y)}] \\
&= H[X] + H[Y] - H[X,Y] \\
&= I(X;Y)
\end{aligned}$$

另外, 对任意 P_X, Q_X 和 kernel$P_{Y|X}$, 令 $P_Y = P_X \circ P_{Y|X}$,$Q_Y = Q_X \circ P_{Y|X}$. 我们有散度的 data-processing 不等式:

$$D(P_X||Q_X) \geq D(P_Y||Q_Y) \tag{2.14}$$

这听上去也很合理: 两个随机变量在被同一个核操作后, 它们的差异应该不会增大. 证明此的关键是留神:

$$D(P_{X,Y}||Q_{X,Y}) = D(P_{X|Y}||Q_{X|Y} \mid P_Y) + D(P_Y||Q_Y)$$
$$D(P_{X,Y}||Q_{X,Y}) = D(P_{Y|X}||P_{Y|X} \mid P_X) + D(P_X||Q_X)$$

然而 $D(P_{Y|X}||P_{Y|X} \mid P_X) = 0$ 而 $D(P_{X|Y}||Q_{X|Y} \mid P_Y) \geq 0$, 这就给出结论.

最后我们将来证明 KL 散度的一个下界, 引入统计距离的概念:

$$\Delta(P,Q) = \frac{1}{2}\sum_x |P(x) - Q(x)| = \max_{W\subseteq\Omega}(P(W) - Q(W)) \tag{2.15}$$

我们断言:

$$\sqrt{\frac{1}{2\log e}D(P||Q)} \geq \Delta(P,Q) \tag{2.16}$$

要证明此, 不得不先证明一个平凡的情况, 考虑设 $d(p||q) = D(\mathrm{Bern}(p)||\mathrm{Bern}(q))$, 则:

$$d(p||q) \geq 2\log e(p-q)^2 \tag{2.17}$$

这并不困难, 设 $f(p) = d(p||q) - 2\log e(p-q)^2$, 容易见到 $f(q) = \frac{\mathrm{d}f(p)}{\mathrm{d}p}(q) = 0$, 此外 $\frac{\mathrm{d}^2 f(p)}{\mathrm{d}p^2} = \log e\left(\frac{1}{p} + \frac{1}{1-p}\right) - 4\log e \geq 0$, 这就给出 $f(p) \geq 0$ 的结论.

接下来考虑证明对于任意的事件 E, 有 $D(P||Q) \geq 2\log e(P(E) - Q(E))^2$, 考虑一个 kernel $P_{Y|X}$, 它把原随机变量加工为一个伯努利分布: 如果 X 满足事件 E 则返回 1, 否则返回 0, 因此根据引理总有 $D(P_{X,Y}||Q_{X,Y}) \geq 2\log e(P(E) - Q(E))^2$, 然而 data-processing 不等式 [2.14] 给出 $D(P||Q) \geq D(P_{X,Y}||Q_{X,Y})$, 这就搞定.

如果我们就讲到这里, 那无非只是引入了一些无聊的定义. 接下来我们将见到信息熵的若干用处. 限于篇幅我们不会讲得太多, 只介绍其中之三: 基于信息熵的尾不等式, 信道编码, 以及一些交互式游戏的界证明.

2.5 Sanov Bound

Sanov Bound 的一个很大的动机是尝试判断最大似然估计的误差.

对于一个可能的空间 $\Omega = \{v_1, \cdots v_L\}$, 现在有一个分布 $P : \Omega \to [0,1]$, 记录 $p_i = P(v_i)$.

现在从 P 中独立取样 $x_1,\cdots,x_n$. 考虑对于一个特定的可重集合 S, 求 $Pr[\{x_1,\cdots,x_n\}=S]$. 回忆到可重集的定义为 $S:\Omega\to\mathbb{N}$, 不妨干脆记录 $s_i=S(v_i)$. 容易发现 $Pr[\{x_1,\cdots,x_n\}=S]=\binom{n}{s_1,\cdots,s_L}p_1^{s_1}\cdots p_L^{s_L}$.

现在考虑一个新的分布 $Q:\Omega\to[0,1]$, 其中 $q_i=\frac{s_i}{n}$. 如果我们采样出了 S, 那么 Q 就是当前的一个最大似然估计, 而且 S 是所有 Q 可能出现的情况中概率最大的. 我们断言:

$$\frac{\exp(-nD(Q||P))}{(n+1)^{L-1}}\le Pr[\{x_1,\cdots,x_n\}=S]\le\exp(-nD(Q||P)) \tag{2.18}$$

此时如果采样 $y_1,\cdots,y_n\sim Q$ 的时候, 先来看看 $Pr[\{y_1,\cdots,y_n\}=S]$.

容易见到 $|\Omega\to\mathbb{N}|\le\frac{1}{(n+1)^{L-1}}$, 从而见到以下简单估计:

$$\begin{aligned}\frac{1}{(n+1)^{L-1}}&\le Pr[\{y_1,\cdots,y_n\}=S]\le 1\\ \frac{1}{(n+1)^{L-1}}&\le\binom{n}{s_1,\cdots,s_L}\le\left((\frac{1}{q_1})^{q_1}\cdots(\frac{1}{q_L})^{q_L}\right)^n\\ \frac{\exp(nH(Q))}{(n+1)^{L-1}}&\le\binom{n}{s_1,\cdots,s_L}\le\exp(nH(Q))\end{aligned}$$

这给出了 $\binom{n}{s_1,\cdots,s_L}$ 的一个上下界. 然而天然有:

$$p_1^{s_1}\cdots p_L^{s_L}=\exp(n\sum_i q_i\log p_i)=\exp(-nD(Q||P)-nH(Q))$$

足够完成此估计.

感性上理解,Sanov Bound 告诉了我们真实分布和采样分布之间的误差可以由 KL 散度所刻画.

2.6 信道编码

定义信道为某种会” 污染” 信息的东西, 或者干脆写成条件概率分布 $P_{Y|X}$. 此外定义信道容量 $C=\max_{P_X}I(X;Y)$, 其中 $(X,Y)\sim P_XP_{Y|X}$.

现在我们考虑一个一般的信道编码, 取 $W\to\vec{X}\in\mathcal{X}^L\to\vec{Y}\in\mathcal{Y}^L\to\hat{W}$.

现在我们来证明以下性质:

- $I(\vec{X};\vec{Y})\le LC$
- data-processing 不等式:$I(W;\hat{W})\le I(\vec{X};\vec{Y})$

对于前者, 由于 Y_i 独立地依赖于 X_i, 考虑:

$$\begin{aligned} I(\vec{X};\vec{Y}) &= \sum_i I(\vec{X};Y_i \mid Y_1 \cdots Y_{i-1}) \\ &= \sum_i I(X_i;Y_i) \\ &\le LC \end{aligned}$$

至于后者, 实际上就是互信息的 data-processing 不等式.

接下来我们要搞定传送速率的问题. 不妨设 $n = H(W)$, 现在我们将要证明:

$$n \le \frac{LC + H(\epsilon)}{1-\epsilon} \tag{2.19}$$

其中 ϵ 是可接受的最大错误概率, 定义为 $\epsilon = \max_w Pr[\hat{W} \neq W | W = w]$.

怎么证明呢? 考虑取一个指示变量 Z, 当 $\hat{W} \neq W$ 的时候 $Z = 1$, 否则 $Z = 0$. 不妨直接让 Z 多错一点, 到达 $Pr[Z = 1|W = w] \equiv \epsilon$ 以方便我们下面的分析. 这样的话 Z 和 W 就独立了. 此时立刻见到:

$$(1-\epsilon) I(W;\hat{W} \mid Z = 0) \le I(W;\hat{W} \mid Z) \le I(W;\hat{W}) + H(Z)$$

而 $I(W;\hat{W} \mid Z = 0) = I(W;W) = H(W) = n$.

2.7 称球游戏

给定 n 个球 (其中有一个次品球, 重量和其它球不一样) 和一架天平, 求最少通过多少次操作才能找到这个球.

根据我们定义的条件熵, 我们知道当我们得知一条信息的时候, 要做的就是把这条信息的熵减去, 从而得到当前的不确定性具体是多少.

如果我们已知次品轻重, 由于一共有 n 个球, 每个球等概率成为次品, 因此总熵是 $\log n$, 每称一次能得到的信息有三种: 平衡, 左边重, 右边重, 因此称一次能得到的熵是 $\log 3$, 也就是说我们至少要猜 $\frac{\log n}{\log 3} = \log_3 n$ 次. 如果我们不知道次品的轻重, 那么至少要猜 $\frac{\log 2n}{\log 3} = \log_3 2n$ 次.

这里并不给出具体情况的详细上下界, 如果读者有兴趣可以参考我的另一篇博客①.

① 见https://lwlaymh.github.io下《贪心与构造》一文

第三章 Lambda Calculus 简介

3.1 什么是 Lambda Calculus

就在图灵以其惊人的天才洞察力发明图灵机的前后, 另一位天才数学家 Alonzo Church 在几乎同时提出了 Lambda Calculus 的概念. 后人已经证明了 Lambda Calculus 是图灵完备的. 在此之外, 比起图灵机的设计,Lambda Calculus 的设计极为简洁干净. 它的定义仅包括:

- 变量符号 x, y, z
- 定义函数 $\lambda x.f(x)$, 这是一个函数, 意为 $x \mapsto f(x)$

除此之外再无其它, 甚至不需要自然数等一系列规则. 举个例子, 最简单的函数当然是恒等函数, 我们记作 $\lambda x.x$, 它总将一个变量 x 映射回其本身.

在此基础上, 我们还有以下三条性质:

- α-Conversion: 绑定变量的名字不重要, 即:$(\lambda x.f(x)) = (\lambda y.f(y))$.
- β-Reduction: 其实就是带入值, 例如 $(\lambda x.x)a$ 就是 a, 而 $(\lambda x.y)a$ 是 y.
- η-Conversion: 是说外延相同的函数相同, 即 $(\lambda x.f(x)) = f$.

让我们来看更多的例子:

3.2 自然数系统

丘奇尝试以函数的迭代来定义自然数, 后人一般也管这种定义方式叫做丘奇数, 我们有:

- $0 = \lambda f.\lambda x.x$
- $1 = \lambda f.\lambda x.f(x)$
- $2 = \lambda f.\lambda x.f(f(x))$
- $3 = \lambda f.\lambda x.f(f(f(x)))$
- $\cdots$

如果我们想把这种定义方式变到 Peano 的定义, 就需要定义后继函数:$\mathrm{SUCC} = \lambda n.\lambda f.\lambda x.f(n\ f\ x)$ 对于一个 Lambda Calculus 的初学者来说, 这种形式看着未免太过诡异. 我们不妨仔细分析一下其中的每一项:

- n 是一个自然数, 根据我们上面所说的, 它形如一个 $\lambda f.\lambda x.f(\cdots f(x)\cdots)$ 的形式.
- 因此,$(n\ f\ x)$ 恰好就表示了 $f(\cdots f(x)\cdots)$, 其中 f 嵌套了 n 次.
- 因此,$f(n\ f\ x)$ 恰好就表示了 $f(f(\cdots f(x)\cdots))$, 其中 f 嵌套了 $n+1$ 次.

- 因此,$\lambda f.\lambda x.f(n\ f\ x)$ 就是 $n+1$.
- 因此,SUCC 是一个函数: 它把 n 变成 $n+1$.

既然都有自然数了,就需要构造自然数上的加法,我们定义 PLUS $= \lambda m.\lambda n.\lambda f.\lambda x.(m\ f\ (n\ f\ x))$ 请试着让我们对此进行一些分析:

- $(n\ f\ x)$ 如上所说,就代表了 $f(\cdots f(x)\cdots)$,其中 f 嵌套了 n 次.
- $m\ f\ (n\ f\ x)$ 呢? 由于 $m\ f\ a$ 意味着将 a 前面套上 m 个函数 f,因此这里相当于在 $f(\cdots f(x)\cdots)$ 前面再嵌套 m 个 f,总共嵌套了 $n+m$ 次.

因此这真的表示了加法! 现在来看乘法,一个显然但是无趣的定义是 MULT $= \lambda m.\lambda n.m(\text{PLUS}n)0$ 但是更有意思的定义是 MULT $= \lambda m.\lambda n.\lambda f.m\ (n\ f)$ 这是怎么回事呢:

- $(n\ f)$ 代表了一个函数,每收到一个 x,就在它前面套 n 个 f.
- $m\ (n\ f)$ 代表了一个函数,每收到一个 x,就在它前面套 m 次 $(n\ f)$,总共套了 $n\times m$ 次 f.

那么指数可以定义吗? 我们可以定义 EXP $= \lambda m.\lambda n.(n\ m)$ 等一下,这个是在干什么呢?

- m 代表了一个函数,它接受一个 f 和一个 x,然后在这个 x 前面套 m 个 f.
- $(n\,m)$ 代表了一个函数,它接受一个 f,然后在前面嵌套若干个 m,就得到了 $m(\cdots m(f)\cdots)$. 那么这个东西是什么呢? 它还是一个函数. 我们来仔细看看:
- $m(f)$ 是一个函数,它接受一个 x,然后得到一个 $f(\cdots f(x)\cdots)$.
- $m\ m(f)$ 于是也是一个函数,当它接受一个 x 后,它就把 $m(f)$ 做的事做 m 次,于是此时 x 前面一共套了 $m\times m$ 个 f.
- $m(m(m(f)))$ 于是也是一个函数,类似地,它将 x 前面套了 $m\times m\times m$ 个 f.
- 以此类推.

第四章　构造主义与 coq 语言简介

4.1　Syntax 和 Semantics

当逻辑课老师第一次讲到基本逻辑的构建的时候, 他一定会提出 Semantics 和 Syntax 两个概念, 并在随后介绍命题逻辑和一阶谓词逻辑的 Soundness 和 Completeness 定理, 以展示我们所定义的 Syntax(比如 Natural Deduction) 的确忠实地反映了我们的 Semantics. 然而, 当他随后展示 Godel 不完备定理时, 初学者就难免对此产生疑惑了: 为什么当我们站在 Semantics 层面时, 可以发现 Syntax 的 Completeness 定理; 但当我们下落到 Syntax 层面时, 却发现了不完备性. 如果他在此时深究 Godel 不完备定理的证明, 那它就会发现另一件令人惊讶的事情: 我们可以造出一个 P, 使得 P 本身说的是:"P 不可被证明", 却怎么也造不出一个命题 P, 使得 P 说的是:"P 命题是错误的命题".

就在逻辑学蓬勃发展的时候, 有一批学者开始质疑原本的思考方式. 他们开始着眼于纯粹的 Syntax 层面 (Kripke 发明 Kripke Semantics 则是后来的事). 其中可能算得上是最出名的一批学者, 后人称他们为构造主义者.

4.2　构造主义简介

构造主义者重新审视了 Natural Deduction 中的规则, 我们都知道 Natural Deduction 包含 $\wedge, \vee, \forall, \exists$ 四个基本符号的 I(introduction) 规则和 E(elimination) 规则, 此外定义了 $\neg P$ 就是 $P \to \bot$, 以及额外的两条:

- 归谬法 (RAA): 当我们可以证明 $\neg P$ 是错误的时候, 我们就说 P 是正确的. 这一条等价于双重否定消去律或者排中律.
- 爆炸规则 (exfalso): 错误命题可以证明任意命题.

构造主义者抛弃了 RAA. 尽管如此, 他们仍然能推出相当多的命题, 其中包括一些很多人直觉上认为" 无法绕过 RAA" 的命题:

- $P \to \neg\neg P$.
- $(P \to Q) \to (\neg Q \to \neg P)$.
- $\neg(P \vee Q) \to (\neg P \wedge \neg Q)$
- $\neg\neg(P \vee \neg P)$

常规的证明这些命题的方式一般是直接用 Natural Deduction 画证明树, 相信读者已经对此非常熟练了. 不过我们这里将介绍另一条优雅得多而且也有用得多的证明方式, 即在 coq 上构建的类型论.

4.3 coq 语言简介

作为一个定理证明器,coq 因其辅助了四色定理的证明而闻名于世. 在 coq 上有两类东西: 递归类型, 以及递归类型中的实例. 我们将以自然数的定义作为一个经典的递归类型的例子:

```
Inductive nat : Type :=
  | O
  | S (n : nat).
```

这段代码的意义就是 Peano 公理的规定: 一个自然数要么是 0, 要么是另一个自然数的后继. 我们有的时候也将 O 和 S 称作 constructor. 读者可以回忆一下我们在 [三] 里所提过的定义自然数的方式, 与这正是类似的. 事实上,coq 正是基于 Lambda Calculus 所开发出来的若干种函数式编程的一种. 然而问题在于,Lambda Calculus 是一个函数上的操作, 它是如何与" 证明" 扯上关系的呢? 为此就必须重新审视" 证明" 的过程与" 函数" 的关系. 我们知道在证明的时候,imply 符号 $\rightarrow$ 是至关重要的, 当我们得知 P 和 $P \rightarrow Q$ 的时候, 我们就可以使用 Elimination 规则得到 Q. 这个过程恰好与函数操作的过程一般无二. 当我们得知一个函数 $f : P \rightarrow Q$ 时, 如果我们知道了一个 $p \in P$, 我们就可以拿到 $f(p) \in Q$.

现在我们将一条命题看作一个" 类型", 而将它的一个证明看作该类型下的一个" 实例". 就可以发现: 当我们知道了 P 的一个证明 p, 以及 $P \rightarrow Q$ 的一个证明 f, 我们就可以组合它们得到一个 Q 的证明 $f(p)$. 于是我们原本的" 构造证明", 就在此转变为了" 构造函数".

既然如此,$\perp$ 当然也是一个命题啦, 那它是什么类型呢, 它的类型正是:

```
Inductive False : Type :=
.
```

乍一看像是我写错了或者漏写了后半部分, 但是没有.False 的定义正是没有定义. 没有任何 constructor 可以构造出 False, 这看上去像是一个不可达的孤点! 试想一下: 如果一个命题竟然可以达到这个孤点, 或者说我们可以证明 $P \rightarrow$ False, 也就是可以构造一个函数 $f : P \rightarrow$ False, 可是这个函数的值域是完完全全空白的, 那里什么也没有, 这意味着这个函数的定义域也什么都没有,P 里面不存在任何一个证明, 这就是为什么我们说 P 是假命题; 此外, 如果我们想要证明 False $\rightarrow P$, 这是轻而易举的, 因为构造函数 f : False $\rightarrow P$ 的方式就是不构造, 因为这个函数的定义域什么都没有, 我们自然不用耗费心力去看谁应当映射到谁.

相比之下, 真命题是:

```
Inductive True : Type :=
| I.
```

思路也很简单: 如果我们想要证明 $P \to \text{True}$, 这无非只需要让 P 里面的所有实例都映射到同一个元素就行了, 这个函数的构造方式也非常轻易.

4.4　一些证明例子

不妨试看这两个例子:

- $P \to \neg(\neg P)$.
- $(P \to Q) \to (\neg Q \to \neg P)$.
- 部分德摩根律:$\neg(P \vee Q) \to (\neg P \wedge \neg Q)$.

将上述按 $\neg$ 定义展开见得:

- $P \to (P \to \text{False}) \to \text{False}$.
- $(P \to Q) \to (Q \to \text{False}) \to P \to \text{False}$.
- 部分德摩根律:$((P \vee Q) \to \text{False}) \to ((P \to \text{False}) \wedge (Q \to \text{False}))$

(1) 立刻得证了: 我们知道了 P 和 $P \to Q$ 自然能推出 Q, 这只是定义.

(2) 也一样: 我们知道了 P 和 $P \to Q$ 就能推出 Q, 又知道了 $Q \to \text{False}$ 就能推出 False.

对于 (3), 也就是说, 我们拿到了:$(P \vee Q) \to \text{False}$, 需要分别证明 $P \to \text{False}$ 和 $Q \to \text{False}$, 两者类似只看前者, 前者意味着: 当我拿到 P 时, 我能导出 False. 的确如此: 因为当我拿到 P 和 $(P \vee Q) \to \text{False}$ 的时候,False 自然可以被导出.

此外, 虽然有的人经常误解构造主义是" 否定了排中律", 但这其实是不妥的, 更好的说法应该是" 抛弃了排中律". 原因是排中律是不可否定的. 我们可以证明 $\neg\neg(P \vee \neg P)$. 我们只需要证明 $(\neg(P \vee \neg P)) \to \text{False}$, 然而部分德摩根律给出 $(\neg(P \vee \neg P)) \to ((\neg P) \wedge (\neg\neg P))$.

观察 $(\neg P) \wedge (\neg\neg P)$, 实际上就是 $(\neg P) \wedge (\neg P \to \text{False})$, 这显然能推出 False, 于是结论的确成立了.