依值类型入门

∞-type Café 暑期学校讲座

Jul 01, 2023

Mepy
Minpows@outlook.com

目录

1. 前言

这里放置一些叙述约定以及预设要求.

本文使用 某些物 以表强调, 有时用以强调语法的结合性, 有时用以吸引读者的注意力.

本文会在关键术语的第一次出现处使用 术语(jargon) 来提及对应的术语外文, 附录处亦有部分术语的命名动机, 这主要是我的叙事风格可能不与他人一致, 例如所谓赋型规则(typing rule).

在数学与逻辑上, 本文要求读者至少有普通高中水平, 能理解自然数上的归纳法, 熟悉一些可从各类科普材料得来的朴素集合论知识, 能理解一阶逻辑. 理解是指你至少能够在阅读本文时对这些概念不感到困惑.

从类型论的预设来说, 本文要求读者熟悉简单类型 λ 演算, 从暑期学校的整体性来说, 请参考 Alias Qli 君的讲义. 简单类型 λ 演算是本文用以叙述依值类型的基底演算, 换言之, 在叙述含有依值类型的演算时, 我们总默认其是简单类型 λ 演算的一个拓展. 符号与之基本一致, 其中替换(substitution) $b[x \mapsto a]$ 用于表示将 b 中的自由变量 x 替换为 a, 小写字母x, y用于表示词项级别的变量, 大写字母A, B用于表示类型级别的变量, 花体字母$\mathcal{U}$用于表示宇宙级别的变量, 黑板体字母$\mathbb{N}$用于表示数学对象.

从工具的角度来说, 读者至少需要从元语言的角度熟悉如何使用相继式演算(sequent calculus)来描述类型论构造. 另外, 读者应当能够明显地区分元语言(自然语言及相继式演算)与目标语言(类型论)的差异, 特别是所谓的定义等同(definitional equality)与命题等同(propositional equality), 前者在本文中使用符号 $\equiv$, 这意味着你可以将二者在你的概念中随意混同, 因为他们是定义上的同一; 而后者是本文即将介绍的同一类型 $=_A$.

本文要求读者知道 Curry–Howard 对应, 即命题与类型的对应, 具体来说熟悉如下表格[1]:

类型含义	类型记号	命题含义	命题记号
单位类型	$\top$	恒真命题	$\top$
空类型	$\bot$	恒假命题	$\bot$
和类型	$A + B$	析取, 逻辑或	$A \vee B$
积类型	$A \times B$	合取, 逻辑与	$A \wedge B$
函数类型	$A \to B$	蕴含	$A \Rightarrow B$
	$A \to \bot$	否定, 逻辑非	$\neg A$

我们在类型论里叙述某命题, 实际上用的是某类型 $P : \mathcal{U}$. 其中 $\mathcal{U}$ 称为宇宙(universe), 类型是宇宙的词项(term)或说居留元(inhabitant), 即$P : \mathcal{U}$.

依值类型(depedent type)是指依赖于词项的类型(type dependent to term), 当我们在谈论某项$a : A$的时候, 我们所涉及的该项性质$P(a) : \mathcal{U}$便是依赖于a的类型.

本文使用如下表示依值类型的依赖情况, 其中 $P(x)$ 强调 P 中含有自由变量 x, 这严格地区别于函数调用$f\ x$; 在上下文清晰的情况下, 读者可以自行分辨含有哪些自由变量.

$$\frac{\Gamma, x : A \vdash P : \mathcal{U}}{\cdots}$$

下面是两节导入例子, 数学背景与计算机背景的读者各取所需.

1.1. YZLX

小标题 YZLX 是依值类型(Yi Zhi Lei Xing)与一致连续(Yi Zhi Lian Xu)的拼音首字母缩写, 他们并**不是** 同一个东西, 本节只是用(一致)连续来做个示例, 以显示出类型依赖于值的关系.

例子 1.1.1: 设实数之间的函数f在区间$[a, b] \subset \mathbb{R}$上有定义[2]:

- f称为在区间$[a, b]$上连续当且仅当

$$\forall x \in [a, b], \forall \varepsilon > 0, \exists \delta > 0, \forall y \in [a, b], |y - x| < \delta \Longrightarrow |f(y) - f(x)| < \varepsilon$$

- f称为在区间$[a, b]$上一致连续当且仅当

$$\forall \varepsilon > 0, \exists \delta > 0, \forall x \in [a, b], \forall y \in [a, b], |y - x| < \delta \Longrightarrow |f(y) - f(x)| < \varepsilon$$

读者在学习数学分析的时候应该有过如下的理解, 根据语境(context)中变量的引入顺序:

- 对于 f 在 $[a, b]$ 上连续, δ 的取值依赖于 x 与 ε 二者;
- 对于 f 在 $[a, b]$ 上一致连续, δ 的取值仅依赖于 ε.

---

[1] 默认地, 我们在讨论构造主义逻辑, 即排除 排中律(Law of Excluded Middle, LEM).

[2] 出于尊重数学习惯的缘故, 函数调用在此仍写作$f(x)$, 学习类型论的读者应该有能力自由切换叙事记号.

从类型论的角度来看，不只在 δ 的取值问题上，而且在后续的命题部分上，即 $\boxed{|y-x|<\delta \Longrightarrow |f(y)-f(x)|<\varepsilon}$ 也依赖于语境中的变量，这是本节使用这一示例的动机.

读者必须对 Curry–Howard 对应这一直观足够熟悉，命题应该被当作类型来看待，而证明是对应类型的词项. 因此，命题 $|y-x|<\delta \Longrightarrow |f(y)-f(x)|<\varepsilon$ 是依赖于$\varepsilon, \delta, x, y$ 的类型.

$$\frac{\Gamma, x:[a,b], \varepsilon:(0,+\infty), \delta:(0,+\infty), y:[a,b] \vdash \boxed{|y-x|<\delta \Longrightarrow |f(y)-f(x)|<\varepsilon} : \mathcal{U}}{\cdots}$$

$$\frac{\Gamma, \varepsilon:(0,+\infty), \delta:(0,+\infty), x:[a,b], y:[a,b] \vdash \boxed{|y-x|<\delta \Longrightarrow |f(y)-f(x)|<\varepsilon} : \mathcal{U}}{\cdots}$$

下面的练习在逻辑上要等到第 2 节后才进行，目前只是意义不明地堆放在此.

练习 1.1.1：使用 Π 类型与 Σ 类型写出连续与一致连续之定义对应的类型.

1.2. 并不那么实用的依值类型编程

在中文互联网上搜寻依值类型相关的例子时，时常会遇见所谓类型安全的 `printf` 一说，即所谓 `printf` 的后续参数类型依赖于第一个参数提供的值，方便起见，不如将 `printf` 透过 Curry 化看待，并且忽略 `printf` 的返回值，来看如下例子：

```
printf("(%d, %g)",   10,    0.4f   ); // 忽略返回值
//       cstr_t   ->  int -> double   ->    void
```

可以理解为如下断言，其中 `printf` 的类型 printf_t 依赖于 cstr 的值：

$$\frac{\Gamma, \text{cstr} : \text{cstr\_t} \vdash \boxed{\text{printf\_t}} : \mathcal{U}}{\cdots}$$

且当 cstr $\equiv$ "(%d, %g)" 时，printf_t $\equiv$ cstr_t $\to$ int $\to$ double $\to$ void.

事实上，编译器或 IDE 会在你输入错误参数的时候给予反馈信息：

```
root@linux:~$ gcc x.c
x.c: In function ‘main’:
x.c:4:19: warning: format ‘%g’ expects argument of type ‘double’, but argument 3 has
type ‘int’ [-Wformat=]
    4 |     printf("(%d, %g)", 10, 5);
      |                  ~^        ~
      |                   |        |
      |                   double   int
      |                  %d
```

C 语言并 **不是** 依值类型的，依值类型的编程语言，请去寻找证明助手，如 Coq, Agda, Idris 等等.

正如小标题所言，依值类型对于编程并不那么实用，依值类型更多地用来做证明.读者必须对 Curry–Howard 对应这一直观足够熟悉，命题应该被当作类型来看待，而证明是对应类型的词项. 例如证明红黑树的相关操作保持平衡，通过给红黑树rbt : rbt_t定义平衡balance这一类型/命题：

$$\frac{\Gamma, \text{rbt} : \text{rbt\_t} \vdash \boxed{\text{balance(rbt)}} : \mathcal{U}}{\cdots}$$

对于红黑树的插入操作insert : rbt_t → elem_t → rbt_t, 这无非是说从p : $\boxed{\text{balance}}$ 出发, 要构造p' : $\boxed{\text{balance}[\text{rbt} \mapsto \text{insert rbt } x]}$.

下面的练习在逻辑上要等到第 2 节后才进行, 目前只是意义不明地堆放在此.

练习 1.2.1: 使用 Π 类型与 Σ 类型写出红黑树 **任意** 插入元素均保持平衡所对应的类型.

2. 一阶逻辑与高阶逻辑

本节将介绍如下表所示的两个类型:

依值函数类型	dependent function type	$(x : A) \to B(x)$
依值对子类型	dependent pair type	$(x : A) \times B(x)$

2.1. 依值函数

依值函数类型是函数类型的依值版本, 差异在函数返回值型依赖于函数参数的值, 因此直观地写作 $\boxed{(x : A) \to B(x)}$. 下面的规则基本上只是在函数类型上做了一些依值的修改, 熟悉简单类型 λ 演算的读者应该很容易接受这一差异. 当返回类型B不依赖于$x : A$时, 依值函数类型自然就退化为一般的函数类型$A \to B$.

$$\frac{\Gamma, x : A \vdash B : \mathcal{U}}{\Gamma \vdash (x : A) \to B : \mathcal{U}} \to_{\text{形成}}$$

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda(x : A).\, b : (x : A) \to B} \to_{\text{构造}}$$

$$\frac{\Gamma \vdash f : (x : A) \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f\ \ a : \boxed{B[x \mapsto a]}} \to_{\text{消去}}$$

$$\frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash \boxed{\lambda(x : A).\, b}\ \ a \equiv b[x \mapsto a] : \boxed{B[x \mapsto a]}} \to_{\text{计算}}$$

在逻辑上, 依值函数类型对应全称量词语句$\forall x, P(x)$. 用自然语言叙述, 对于任意的x, 总有命题$P(x)$成立, 此一阶逻辑命题的证明意味着任意给定$x : A$, 总有类型为$P(x)$的元素, 这就构成了一个函数, 将$x : A$映射为命题$P(x)$之证明的函数f : $\boxed{(x : A) \to P}$.

2.2. 依值对子

类似地, 与特称量词语句$\exists x, P(x)$对应的类型, 是依值对子类型 $\boxed{(x : A) \times B(x)}$, 是对子类型$A \times B$ 的依值版本, 其中第 2 个元素的类型依赖于第 1 个元素的值.

用自然语言描述, 存在某$x : A$满足命题$P(x)$, 此一阶逻辑命题的证明意味着存在某个对子(a, p), 使得$a : A$并且命题$P[x \mapsto a]$有居留元p.

$$\frac{\Gamma, x : A \vdash B : \mathcal{U}}{\Gamma \vdash (x : A) \times B : \mathcal{U}} \times_{\text{形成}}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B[x \mapsto a]}{\Gamma \vdash (a, b) : (x : A) \times B} \times_{\text{构造}}$$

$$\frac{\Gamma \vdash p : (x : A) \times B}{\Gamma \vdash p.1 : A} \times_{\text{消去1}}$$

$$\frac{\Gamma \vdash p : (x : A) \times B}{\Gamma \vdash p.2 : B[x \mapsto p.1]} \times_{\text{消去2}}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B[x \mapsto a]}{\Gamma \vdash (a, b).1 \equiv a : A} \times_{\text{计算1}}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma, a : A \vdash b : B[x \mapsto a]}{\Gamma \vdash (a, b).2 \equiv b : B[x \mapsto a]} \times_{\text{计算2}}$$

2.3. 依值积与依值和

令初学者不免感到困惑的是, 依值函数类型与依值对子类型分别是函数类型和对子类型(即积类型)的依值版本, 那么和类型的依值版本到哪里去了?

事实上, 在相关文献中, 依值积类型(dependent product type)指的就是依值函数类型, 而依值和类型(dependet sum type)指的就是依值对子类型, 如下表所示:

依值函数类型, 依值积类型, Π类型	$\Pi(x : A)B(x), (x : A) \to B(x)$
依值对子类型, 依值和类型, Σ类型	$\Sigma(x : A)B(x), (x : A) \times B(x)$

现在考虑索引类型I, 更具象些, 可以取$I \equiv \text{Bool} \equiv \{0, 1\}$. 对于 Π 类型 $\Pi(i : I)X(i)$, 其元素 $t : \Pi(i : I)X(i)$ 事实上是由 I 索引的元组, $t\ \ i$ 是元组中的第i个元素. 来自数学背景的读者, 可以回忆集合论中关于集合的无限笛卡尔积是如何构造的, 例如拓扑空间的积拓扑与箱拓扑等内容中会涉及无限笛卡尔积, 用以编码无限笛卡尔积的正是映射; 来自计算机背景的读者, 可以回忆所使用过支持重载运算符的编程语言, 例如C++, 当重载 `operator[]` 时, 所编写函数的类型就是 Π 类型的一种退化情形, 即X不依赖于$i : I$. 不妨使用范畴话语, 我们使用态射对乘积对象进行编码.

类似地, 对于 Σ 类型 $\Sigma(i : I)X(i)$, 其元素 (i, x) 事实上是和. 来自数学背景的读者, 请回忆集合不交并的构造, 一种最粗暴的方法便是使用标记i以指示$x \in X_i$, 构成二元组(i, x); 来自计算机背景的读者, 只需注意到 i 是标记, 例子不胜枚举, 例如 OCaml 运行时区分指针和整数的方法是在 64 位的最低位存储标记位 i , 0 表示指针(对齐的内存地址), 1 表示整数, 其余 63 位用于存储数据 x .

2.4. 宇宙谱系漫游指南

不知阅读至此的读者, 是否对 $\Gamma, x : A \vdash P : \mathcal{U}$ 这一元语言产生疑惑: 为何不直接写成 $\Gamma \vdash P : A \to \mathcal{U}$? 使用闭项不是更方便吗?

这是因为, 一旦提及类型 $A \to \mathcal{U}$, 我们就必须考虑这一类型位于什么宇宙中. 若某类型论约定 $A \to \mathcal{U} : \mathcal{U}$, 则该类型论可被称为非直谓的(impredicate), 这样的类型论会带来类似于 Russell 悖论的悖论, 除非加以限制.

更一般地, 考虑宇宙$\mathcal{U}$本身作为类型, 应该位于何等宇宙之中, 一种最简单的想法是谱系(hierarchy), 即$\mathcal{U}_0 : \mathcal{U}_1 : \mathcal{U}_2 : \ldots : \mathcal{U}_n : \ldots$. 对于$A : \mathcal{U}_i, B : \mathcal{U}_j$, 令$A \to B : \mathcal{U}_{\max(i,j)}$的类型论称为直谓的(predicate). 直谓的宇宙谱系是安全的, 历史上这也是 Russell 最初处理悖论的方法.

假设引入宇宙谱系, 那么 $\Gamma \vdash P : A \to \mathcal{U}_i$ 便是可行的, 而此时 $\Gamma, x : A \to \mathcal{U}_i \vdash$ 中的变量 x 在逻辑上对应了命题变元, 因此类型论对应的逻辑系统便是高阶逻辑系统.

顺带一提, 假设宇宙可以出现在函数类型中, 那么依赖于类型的项 $f : \mathcal{U} \to A$ 以及依赖于类型的类型 $f : \mathcal{U} \to \mathcal{U}$ 也就实现了.

宇宙以及谱系的引入模糊了词项与类型之间的差异, 毕竟类型是宇宙的词项, 宇宙U_i又是更高一等宇宙$U_{\{i+1\}}$的词项. 这就未免让人对下列 词项 感到困惑:

$$\lambda(x:A).\,\Pi(y:A)P(x,y)$$

依照一种相对渡步的感受, $M \equiv \Pi(y:A)P(x,y) : \mathcal{U}$ 形如 词项 : 类型, 而 $\lambda(x:A).\,M : (x:A) \to \mathcal{U}$ 亦然. 尽管在依值类型论中, 类型是(宇宙的)词项, 但是相对而言, 读者必须区分词项与类型, 方能漫步在宇宙谱系中.

至此, 读者理应能区分依值函数类型$(x:A) \to B(x)$与 返回 类型 之函数 的类型 $A \to \mathcal{U}$.

由于宇宙谱系的引入会增添许多复杂的内容, 在不加以强调的情况下, 后文只使用单个宇宙 $\mathcal{U}$, 总使用 $\Gamma, x:A \vdash P:\mathcal{U}$ 这一元语言.

3. 归纳! 归纳! 归纳!

本节将介绍 Martin Löf 类型论(Martin Löf Type Theory, MLTT) 中的同一类型, 然后通过定义自然数类型并对其进行论述, 展示类型论作为基础框架, 如何用来研究特定理论, 并最终概括为各种各样的归纳子.

3.1. 同一:唯有自反

本小节将介绍同一类型(identity type) $a =_A a'$, 或称为相等类型(equality type), 即在命题意味上, 项a与a'相等. 在集合论中, 同一 $=$ 是经由属于 $\in$ 导出的, 即 $a = a' \iff \forall x \in a, x \in a', \forall x \in a', x \in a$; 但在类型论中, 同一类型 $a =_A a'$是直接形成的:

$$\frac{\Gamma \vdash a:A \qquad \Gamma \vdash a':A}{\Gamma \vdash a =_A a' : \mathcal{U}} =_{\text{形成}}$$

$$\frac{\Gamma \vdash a:A}{\Gamma \vdash 1_a : a =_A a} =_{\text{构造}}$$

其中 1_a 称为自反性(reflexivity), 是命题 $a = a$ 的证明.

在自然语言中, 我们轻而易举地就将同一的二者在概念或说定义中同一, 为了在类型论中机械而形式地刻画这种行为, 我们需要同一类型的消去规则, 即 J 规则[3]:

$$\frac{\Gamma, x:A, y:A, p:x =_A y \vdash C(x,y,p):\mathcal{U} \qquad \Gamma \vdash c:(z:A) \to C[x \mapsto z, y \mapsto z, p \mapsto 1_z]}{\Gamma, x:A, y:A, p:x =_A y \vdash \mathrm{ind}_=(x,y,p,c) : C(x,y,p)} =_{\text{消去}}$$

从命题的视角看, 关于 $x:A, y:A$ 且已知$p:x=y$ 的命题$C(x,y,p)$, 假定已在概念同一, 即定义等同($z \equiv z:A$)的情况下证明了该命题$c\ \ z:C[x \mapsto z, y \mapsto z, p \mapsto 1_z]$, 那么我们可以得到原命题的证明 $\mathrm{ind}_=(x,y,p,c) : C(x,y,p)$. 下面作为演示, 我们使用 J 规则来证明一个重要的引理:

引理 3.1.1: 设$a, a':A, p:a =_A a'$, 则对于任意的函数[4] $f:A \to B$, 总有$f\ \ a =_B f\ \ a'$.

证明: 记$C(a,a',p) \equiv$ $f\ \ a =_B f\ \ a'$, 则

[3] 名称 J 来自同一类型 Identity type 首字母 I 的下一字母, 这是历史原因.

[4] 注意此处并不是依值函数$f:(x:A) \to B$, 原因在于类型$B[x \mapsto a]$与$B[x \mapsto a']$, 即便假设宇宙谱系能证得二者命题同一, 但并非概念同一, 因而无法触发形成规则$f\ \ a =_{\boxed{?}} f\ \ a' : \mathcal{U}$.

$$\boxed{(z:A)\to C[x\mapsto z, y\mapsto z, p\mapsto 1_z]} \equiv (z:A)\to \boxed{f\ \ z =_B f\ \ z}$$

其中显然有居留元$\lambda(z:A).1_{f\ z}$, 定义$\mathrm{ap} \equiv \lambda a\ \ a'\ \ p\ \ f.\mathrm{ind}_{=}\left(a, a', p, \lambda z.1_{f\ z}\right)$得证. □

这事实上是一种归纳法(induction principle), 只是归纳分支唯有自反一支[5], 区别于自然数归纳法, 这是下一节的工作.

练习 3.1.1: 试证明同一类型$=_A$具有对称性, 即设$a, a' : A, p : a =_A a'$, 证明$a' =_A a$.

3.2. 自然数:最初的归纳

自然数的归纳法, 或者说第一数学归纳法, 用自然语言叙述如下:

命题 3.2.1: 对于任意与自然数n有关的命题$P(n)$, 若已知如下两支成立:
1. $P(0)$成立
2. 设任意的自然数n, 若$P(n)$成立能推出$P(n+1)$成立

则对于任意的自然数n, 命题$P(n)$都成立.

在类型论中, 我们不负责对这一命题进行证明, 毕竟 ZFC 集合论中的自然数构造过于冗长. 我们仅从 Peano 公理的角度进行思考, 这是 Peano 公理中的第 5 条, 其刻画了自然数最重要的直观, 保证了自然数只能是一条从 0 逐渐递增的单链.

下面定义类型论中的自然数类型:

$$\frac{}{\Gamma\vdash \mathbb{N} : \mathcal{U}}\mathbb{N}_{\text{形成}}$$

$$\frac{}{\Gamma\vdash 0 : \mathbb{N}}\mathbb{N}_{\text{构造0}}$$

$$\frac{\Gamma\vdash n : \mathbb{N}}{\Gamma\vdash S(n) : \mathbb{N}}\mathbb{N}_{\text{构造S}}$$

上述的两条构造规则对应着 Peano 公理的前两条, 告诉了我们哪些是自然数, 方便起见, 从自然语言的角度, 我们可以把$S(0)$写作 1, $S(1)$写作 2, 如此类推, 关于$S(n)$这一记号, 其他材料中亦有写作$\mathrm{succ}(n)$, $n++$, $n+1$.

对应于构造规则, 归纳法是如下所述的消去规则:

$$\frac{\Gamma, n:\mathbb{N}\vdash P(n):\mathcal{U} \qquad \Gamma\vdash p_0 : P(0) \qquad \Gamma\vdash p_s : (n:\mathbb{N})\to P(n)\to P(S(n))}{\Gamma, m:\mathbb{N}\vdash \mathrm{ind}_{\mathbb{N}}(m, p_0, p_s) : P(m)}\mathbb{N}_{\text{消去}}$$

与归纳法配套的是归纳定义, 在归纳定义的意味下, $P(n)$作为类型, 并不对应某个命题. 我们用自然数的加法来演示如何进行归纳定义, 下面是两种不同风格的伪代码:

```
let rec add (m:nat) (n:nat) : nat = match m
  | 0 => n
  | S(m) => S(add m n)
```

```
add : N -> N -> N
add   0    n := n
add  S(m)  n := S(add m n)
```

[5]尽管如此, 需要提及的是, 这不意味着$a =_A a$只有自反1_a一个居留元, 读者可以在同伦类型论中找到反例, 只不过这远超了本文所谈及的范围.

上述定义的步幅有些大, 事实上, 加法的定义与运算会被划分为两步, 首先是定义:

定义 3.2.1: 加法 $\mathrm{add} \equiv \lambda\ m\ n.\mathrm{ind}_{\mathbb{N}}(m, n, \lambda\ \_\ n.S(n)))$, 其中:

$$\frac{\Gamma \vdash \boxed{P \equiv \mathbb{N}} : \mathcal{U} \qquad \Gamma, n : \mathbb{N} \vdash \boxed{p_0 \equiv n} : \mathbb{N} \qquad \Gamma \vdash \boxed{p_s \equiv \lambda\ \_\ n.S(n)} : (\_ : \mathbb{N}) \to \mathbb{N} \to \mathbb{N}}{\Gamma, m : \mathbb{N}, n : \mathbb{N} \vdash \mathrm{ind}_{\mathbb{N}}(m, n, p_s) : \mathbb{N}}\mathbb{N}_{\text{消去}}$$

紧接着, 加法的运算, 更一般地, 归纳定义的计算规则如下所述:

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n) : \mathcal{U} \qquad \Gamma \vdash p_0 : P(0) \qquad \Gamma \vdash p_s : (n : \mathbb{N}) \to P(n) \to P(S(n))}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}}(0, p_0, p_s) \equiv p_0 : P(0)}\mathbb{N}_{\text{计算0}}$$

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n) : \mathcal{U} \qquad \Gamma \vdash p_0 : P(0) \qquad \Gamma \vdash p_s : (n : \mathbb{N}) \to P(n) \to P(S(n))}{\Gamma, n : \mathbb{N} \vdash \mathrm{ind}_{\mathbb{N}}(S(n), p_0, p_s) \equiv p_s\ n\ \boxed{\mathrm{ind}_{\mathbb{N}}(n, p_0, p_s)} : P(S(n))}\mathbb{N}_{\text{计算S}}$$

例子 3.2.1: $\mathrm{add}\ \ 3\ \ 1 \equiv 4$, 方便起见, 记$p_s \equiv \lambda\ _\ n.S(n))$

$$\begin{aligned}
\mathrm{add}\ \ 3\ \ 1 &\equiv \mathrm{ind}_{\mathbb{N}}(3, 1, p_s) \equiv p_s\ \ 2\ \ \mathrm{ind}_{\mathbb{N}}(2, 1, p_s) \\
&\equiv S(\mathrm{ind}_{\mathbb{N}}(2, 1, p_s)) \equiv S(p_s\ \ 1\ \ \mathrm{ind}_{\mathbb{N}}(1, 1, p_s)) \\
&\equiv S(S(\mathrm{ind}_{\mathbb{N}}(1, 1, p_s))) \equiv S(S(p_s\ \ 0\ \ \mathrm{ind}_{\mathbb{N}}(0, 1, p_s))) \\
&\equiv S(S(S(\mathrm{ind}_{\mathbb{N}}(0, 1, p_s)))) \equiv S(S(S(1))) \\
&\equiv 4
\end{aligned}$$

直观地说, $\mathrm{add}\ \ m\ \ n$ 的语义是将 $p_s\ \ \_ \equiv \lambda\ \ n.S(n)$ 这一函数作用在 n 上 m 次. 熟悉无类型 λ 演算的读者或许会立即想起 Church 编码的自然数 m 是将某函数应用到参数上 m 次的高阶函数, 写过解释器(interpreter)的读者还会有更深的洞见: 自然数类型是一类语法树(Abstract Syntax Tree, AST), 而归纳定义则是对语法树的解释(interpretation). 从此处出发, 还可以统一地刻画包括其余的归纳类型(inductive type), 即所谓的 W 类型, 本文不表. 像自然数理论一样, 定义归纳类型, 并对归纳类型进行相应的论证, 是使用类型论研究一些理论的一大通用途径.

练习 3.2.1: 利用add, 定义自然数的乘法$\mathrm{mul} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$.

下面是使用归纳法进行证明的一个例子:

引理 3.2.1: 对于任意的自然数$n : \mathbb{N}$, 总有$n + 0 = n$.

证明: 上述命题对应的类型为 $(n : \mathbb{N}) \to n + 0 =_{\mathbb{N}} n$, 取$P(n) \equiv \boxed{n + 0 =_{\mathbb{N}} n}$, 则归纳所需的分支为 $1_0 : 0 + 0 \equiv 0 =_{\mathbb{N}} 0$ 以及

$$p_s : (n : N) \to n + 0 =_{\mathbb{N}} n \to S(n) + 0 \equiv S(n + 0) = S(n)$$

从$n + 0 =_{\mathbb{N}} n$要证$S(n) + 0 \equiv S(n + 0) =_{\mathbb{N}} S(n)$, 注意到上一节中的引理ap, 取

$$p_s \equiv \lambda n\ \ p.\mathrm{ap}\ \ n+0\ \ n\ \ p\ \boxed{\lambda n.\, S(n)}$$

所得 $\lambda n.\,\mathrm{ind}_{\mathbb{N}}(n, 1_0, p_s)$ 为本引理的证明. □

练习 3.2.2: 试证明自然数加法的交换律, 即对于任意的$m, n : \mathbb{N}$, 总有$m + n = n + m$.

3.3. 依值消去:归纳子

在引入了归纳法这一术语后, 我们可以回到依值对子类型作为乘积类型与余积类型的依值推广上, 并给出如下的归纳原理:

$$\frac{\Gamma, z : (x : A) \times B \vdash C(z) : \mathcal{U} \qquad \Gamma \vdash c : (a : A) \to (b : B[x \mapsto a]) \to C[z \mapsto (a, b)] \qquad \Gamma \vdash p : (x : A) \times B}{\Gamma \vdash \mathrm{ind}_{\times}(c, p) : C[z \mapsto p]}\ \times_{\text{消去-归纳}}$$

$$\frac{\Gamma, z : (x : A) \times B \vdash C(z) : \mathcal{U} \qquad \Gamma \vdash c : (a : A) \to (b : B[x \mapsto a]) \to C[z \mapsto (a, b)] \qquad \Gamma \vdash a : A, b : B[x \mapsto a]}{\Gamma \vdash \mathrm{ind}_{\times}(c, (a, b)) \equiv c\ \ a\ \ b : C[z \mapsto (a, b)]}\ \times_{\text{计算-归纳}}$$

通过定义 $c_1 \equiv \lambda a\ \ b.\, a$ 就得到了对应的消去规则1与计算规则1:

$$\frac{\Gamma \vdash p : (x : A) \times B}{\Gamma \vdash p.1 : A}\ \times_{\text{消去1}}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B[x \mapsto a]}{\Gamma \vdash (a, b).1 \equiv a : A}\ \times_{\text{计算1}}$$

类似地, 我们也可以定义 $c_2 = \lambda a\ \ b.\, b$得到对应的规则2, 这是简单的; 当然, 作为余积类型的依值版本, 我们也可以得到对应的规则, 只不过我们需要假定有 Boolean 类型的归纳原理, 或者说我们能对 Boolean 做模式匹配(pattern matching). 不妨留作练习:

练习 3.3.1: 写出 Boolean 类型 $\mathbb{B}$ 的赋型规则(归纳子$\mathrm{ind}_{\mathbb{B}}$), 并因此给出对应的定义

$$c : (i : \mathbb{B}) \to (x : X(i)) \to C((i, x))$$

使得依值对子类型的归纳子$\mathrm{ind}_{\times}$ 退化为余积类型的归纳子 ind_{+}.

另外, 关于归纳(induction)/递归(recursion), 在相关文献中, 会将依值版本的消去子$\mathrm{elim}_?$称作归纳子$\mathrm{ind}_?$, 非依值版本的消去子$\mathrm{elim}_?$称为递归子$\mathrm{rec}_?$.

4. 未完的旅途

- 宇宙 的有趣 我才不在意
  - universes à la Tarski and that à la Russell
- 你真的懂 唯一 的定义
  - uniqueness rule of each type
  - the weak/strong computation rule of identity type
  - extentionality/intentionality around definitional/propositional equality
  - B[x ↦ a] and B[x ↦ a']
  - the univalent axiom and the funtion-extentionality axiom

5. 附录

5.1. 习题汇总

练习 5.1.1: 使用 Π 类型与 Σ 类型写出连续与一致连续之定义对应的类型.

练习 5.1.2: 使用 Π 类型与 Σ 类型写出红黑树 **任意** 插入元素均保持平衡所对应的类型.

练习 5.1.3: 试证明同一类型 $=_A$ 具有对称性, 即设$a, a' : A, p : a =_A a'$, 证明$a' =_A a$.

练习 5.1.4: 利用add, 定义自然数的乘法$\mathrm{mul} : \mathbb{N} \to \mathbb{N} \to \mathbb{N}$.

练习 5.1.5: 试证明自然数加法的交换律, 即对于任意的$m, n : \mathbb{N}$, 总有$m + n = n + m$.

练习 5.1.6: 写出 Boolean 类型 $\mathbb{B}$ 的赋型规则(归纳子$\mathrm{ind}_{\mathbb{B}}$), 并因此给出对应的定义

$$c : (i : \mathbb{B}) \to (x : X(i)) \to C((i, x))$$

使得依值对子类型的归纳子$\mathrm{ind}_{\times}$ 退化为余积类型的归纳子 ind_{+}.

5.2. 术语与命名动机

- **词项(term)**: 一般译作项, 出于音节对称的美感, 当与类型相对时, 选译为词项.
- **赋型规则(typing rule)**: 或译作类型规则, 但从词性的角度, typing 是动名词, 实用编程语言理论基础(Pratical Foundation of Programming Language, PFPL)的中译本译作定型规则, 意为确定类型, 考虑到元语言与目标语言之间叙述与被叙述的关系, 私以为用 $\boxed{\text{赋}}$ 更具有主观性.
- **定义等同(definitional equality)** 与 **命题等同(propositional equality)**: 这里的等同(equality)更倾向于同一(identity)这一含义, 其中定义等同在本文中也常常被叙述为概念同一; 命题等同则是类型论层面的同一, 即同一类型.
- **同一类型(identity type)**: 同一来自同一性(identity), 另外英文术语中也会使用 equality type 或 path type(在同伦类型论的语境中)表示, 对应译作相等类型或路径类型.
- **宇宙谱系(universe hierachy)**: 此词参考自李文威老师的代数学方法卷一, 原本用以描述集合论的 Grothendieck 宇宙.
- **形成规则(formation rule)** 与 **构造规则(introduction rule)**: 在依值类型的语境下讨论, 二者是相似而易混淆的, 前者在宇宙中形成类型, 后者在类型中构造词项, 读者须严格区分. 构造一词

有时也用作 construct 的翻译, 特别是在归纳类型的构造子(constructor)这一概念, 读者可以注意到构造子用以构造词项, 如自然数类型 $\mathbb{N}$ 的 0 与 S.

5.3. 赋型规则

5.3.1. 依值函数类型

$$\frac{\Gamma, x : A \vdash B : \mathcal{U}}{\Gamma \vdash (x : A) \to B : \mathcal{U}} \to_{\text{形成}}$$

$$\frac{\Gamma, x : A \vdash b : B}{\Gamma \vdash \lambda(x : A).\, b : (x : A) \to B} \to_{\text{构造}}$$

$$\frac{\Gamma \vdash f : (x : A) \to B \qquad \Gamma \vdash a : A}{\Gamma \vdash f\ \ a : \boxed{B[x \mapsto a]}} \to_{\text{消去}}$$

$$\frac{\Gamma, x : A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash \boxed{\lambda(x : A).\, b}\ \ a \equiv b[x \mapsto a] : \boxed{B[x \mapsto a]}} \to_{\text{计算}}$$

5.3.2. 依值对子类型

$$\frac{\Gamma, x : A \vdash B : \mathcal{U}}{\Gamma \vdash (x : A) \times B : \mathcal{U}} \times_{\text{形成}}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B[x \mapsto a]}{\Gamma \vdash (a, b) : (x : A) \times B} \times_{\text{构造}}$$

$$\frac{\Gamma \vdash p : (x : A) \times B}{\Gamma \vdash p.\, 1 : A} \times_{\text{消去1}}$$

$$\frac{\Gamma \vdash p : (x : A) \times B}{\Gamma \vdash p.\, 2 : B[x \mapsto p.\, 1]} \times_{\text{消去2}}$$

$$\frac{\Gamma, z : (x : A) \times B \vdash C(z) : \mathcal{U} \qquad \Gamma \vdash c : (a : A) \to (b : B[x \mapsto a]) \to C[z \mapsto (a, b)] \qquad \Gamma \vdash p : (x : A) \times B}{\Gamma \vdash \mathrm{ind}_{\times}(c, p) : C[z \mapsto p]} \times_{\text{消去-归纳}}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash b : B[x \mapsto a]}{\Gamma \vdash (a, b).\, 1 \equiv a : A} \times_{\text{计算1}}$$

$$\frac{\Gamma \vdash a : A \qquad \Gamma, a : A \vdash b : B[x \mapsto a]}{\Gamma \vdash (a, b).\, 2 \equiv b : B[x \mapsto a]} \times_{\text{计算2}}$$

$$\frac{\Gamma, z : (x : A) \times B \vdash C(z) : \mathcal{U} \qquad \Gamma \vdash c : (a : A) \to (b : B[x \mapsto a]) \to C[z \mapsto (a, b)] \qquad \Gamma \vdash a : A, b : B[x \mapsto a]}{\Gamma \vdash \mathrm{ind}_{\times}(c, (a, b)) \equiv c\ \ a\ \ b : C[z \mapsto (a, b)]} \times_{\text{计算-归纳}}$$

5.3.3. 同一类型

$$\frac{\Gamma \vdash a : A \qquad \Gamma \vdash a' : A}{\Gamma \vdash a =_A a' : \mathcal{U}} =_{\text{形成}}$$

$$\frac{\Gamma \vdash a : A}{\Gamma \vdash 1_a : a =_A a} =_{\text{构造}}$$

$$\frac{\Gamma, x : A, y : A, p : x =_A y \vdash C(x, y, p) : \mathcal{U} \qquad \Gamma \vdash c : (z : A) \to C[x \mapsto z, y \mapsto z, p \mapsto 1_z]}{\Gamma, x : A, y : A, p : x =_A y \vdash \mathrm{ind}_{=}(x, y, p, c) : C(x, y, p)} =_{\text{消去}}$$

5.3.4. 自然数类型

$$\frac{}{\Gamma \vdash \mathbb{N} : \mathcal{U}} \mathbb{N}_{\text{形成}}$$

$$\frac{}{\Gamma \vdash 0 : \mathbb{N}} \mathbb{N}_{\text{构造0}}$$

$$\frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash S(n) : \mathbb{N}} \mathbb{N}_{\text{构造S}}$$

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n) : \mathcal{U} \qquad \Gamma \vdash p_0 : P(0) \qquad \Gamma \vdash p_s : (n : \mathbb{N}) \to P(n) \to P(S(n))}{\Gamma, m : \mathbb{N} \vdash \mathrm{ind}_{\mathbb{N}}(m, p_0, p_s) : P(m)} \mathbb{N}_{\text{消去}}$$

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n) : \mathcal{U} \qquad \Gamma \vdash p_0 : P(0) \qquad \Gamma \vdash p_s : (n : \mathbb{N}) \to P(n) \to P(S(n))}{\Gamma \vdash \mathrm{ind}_{\mathbb{N}}(0, p_0, p_s) \equiv p_0 : P(0)} \mathbb{N}_{\text{计算0}}$$

$$\frac{\Gamma, n : \mathbb{N} \vdash P(n) : \mathcal{U} \qquad \Gamma \vdash p_0 : P(0) \qquad \Gamma \vdash p_s : (n : \mathbb{N}) \to P(n) \to P(S(n))}{\Gamma, n : \mathbb{N} \vdash \mathrm{ind}_{\mathbb{N}}(S(n), p_0, p_s) \equiv p_s \ \ n \ \boxed{\mathrm{ind}_{\mathbb{N}}(n, p_0, p_s)} : P(S(n))} \mathbb{N}_{\text{计算S}}$$