Coq、Lean 及构造演算入门

大西王

July 4, 2023

以依值类型论为基础的计算机证明助理有很多，其中最常见的三种为 Coq、Lean 和 Agda．Agda 基于 Martin-Löf 类型论（MLTT），其用法也已经在之前的课程中简单介绍，而 Coq 和 Lean 都基于**构造演算**（calculus of constructions, CoC）的变体．构造演算是 MLTT 的扩张，但其独特的特性使得它和 MLTT 有着不小的区别，并且和后续课程将介绍的同伦类型论（homotopy type theory, HoTT）不甚兼容．同时，Coq 缺乏一些后续课程中需要的特性，而 Lean 的一些特性使得其与 HoTT 不兼容，因此我们只在这里简单介绍 CoC 以及基于 CoC 的 Coq 和 Lean，并说明 CoC 及其变体与 MLTT、Coq 和 Lean 与 Agda 的不同之处.

1 Martin-Löf 类型论回顾

在之前的课程中, 各位应当已经学到了 Martin-Löf 类型论（MLTT）的一些基础知识，以及 MLTT 在证明助理 Agda 中的实现．我们在此简单回顾一下 Martin-Löf 类型论的相关知识.

Martin-Löf 类型论只有四类基本类型：**依值函数**（即 Π）类型、**依值对子**（即 Σ）类型、**同一类型**（identity type）以及**宇宙**（universe），但我们一般还允许定义新的**归纳类型** (inductive type)．这使得 MLTT 的表达力大大增强，因很多常见的结构都可作为归纳类型定义.

譬如自然数类型 $\mathbb{N}$，其为拥有两个构造器的归纳类型：其构造器分别为 0 和 $\mathsf{succ}(n : \mathbb{N})$．$\Sigma$ 类型和同一类型实际上亦可实现为归纳类型，因此，若我们允许定义新的归纳类型，则只需要 Π 类型和宇宙即足矣.

1.1 关于宇宙谱系

宇宙谱系（universe hierarchy）在之前的课程中已经简单提到，但我们在此更加详细地说明.

熟悉基础集合论的朋友应当清楚集合论中并不存在“包含所有集合的集合”，因为这种集合的存在将会导致理论不一致。这便是集合论中知名的罗素（Russell）悖论．同样，类型论中也有类似于罗素悖论的 Girard 悖论，因此下面的朴素的类型规则会导致不一致，因而并不可行：

$$\frac{\textsc{TypeInType}}{\vdash \mathcal{U} : \mathcal{U}}$$

Figure 1: 会导致不一致的朴素规则

对于这个问题，罗素提出了一个简单地解决方案：只要我们有可数多个叠套的宇宙，使得小的宇宙“包含”于大的宇宙中，问题便应刃而解．用现代的记号，我们可以将其写为如下的规则：

$$\frac{\textsc{Univ}}{\vdash \mathcal{U}_i : \mathcal{U}_{i+1}} \qquad\qquad \frac{\overset{\textsc{Cumul}}{\Gamma \vdash A : \mathcal{U}_i}}{\Gamma \vdash A : \mathcal{U}_{i+1}}$$

Figure 2: Russell 式的宇宙

设问 1.1. 试证明

$$\frac{\overset{\textsc{Cumul'}}{\Gamma \vdash A : \mathcal{U}_i \qquad i \le j}}{\Gamma \vdash A : \mathcal{U}_j}$$

规则是可容许的（即其可自其它规则推出）．

这种“套叠”的宇宙谱系被称为**渐增的**（cumulative），其特征是上面的 Cumul 规则．渐增的宇宙谱系很容易使用，但其实现相当困难，因为很难判断类型究竟应当居留于何层宇宙之中．因此，我们亦可以放弃渐增性，即舍弃 Cumul 规则而只保留 Univ 规则．这是可行的，但将导致一些定义过分繁琐；实际上，Agda 就缺乏渐增性，各位可能会在之后的课程中切身体会其导致的种种麻烦．

2 类型与命题

在之前的课程中，我们曾经提到如下的观点：

论 2.1 (Curry-Howard 对应)．在类型论中，“类型”对应着“命题”．

但实际上，这一观点有着一定的瑕疵．显然，并不是所有类型都能看作命题：$\mathbb{N}$ 固然就不适合也不应当看作命题．那么，究竟有哪些类型对应着命题呢？在类型论的历史中，不同的流派给出了迥然不同的答案，而在此我们将会介绍一个最简单直接的答案：在语法层面上，直接将对应命题的类型和不对应命题的类型做出区分；这便是构造演算（CoC）的开端．

2.1 命题的宇宙

刚刚已经提到，在 MLTT 中有可数多个宇宙，其构成一个宇宙谱系：$\mathcal{U}_0 : \mathcal{U}_1 : \ldots : \mathcal{U}_i : \ldots, i \in \mathbb{N}$．该宇宙谱系可能渐增亦可能并非渐增，在此并无显著区别．

CoC 在此基础上增加了一个新的宇宙 Prop（本文中亦写作 $\mathcal{P}$），谓之**命题宇宙**（universe of propositions）．顾名思义，该宇宙里的类型被称为“命题”，且都对应着并可以被看作是命题．Coq 和 Lean 中都有一个不同于类型宇宙 $\mathcal{U}_i$[1]的命题宇宙 Prop．例如，在 Coq 和 Lean 中，同一类型 $x =_A y$ 就居留在 Prop 之中．

细心的读者可能发现，上文提到的是“一个”命题宇宙而非“谱系”．这并不是疏漏或错误：Prop 的性质和 Type_i 迥然不同，因此并不需要宇宙谱系．在下面一节，我们将着重描述这点．

[1]本文中为了和命题宇宙更易区别，亦作 Type_i．

注. Coq 的宇宙谱系是渐增的，因此 Coq 的 Prop 宇宙实际也是宇宙谱系的一部分. 更具体地，在 Coq 中我们有 $\mathsf{Prop} : \mathsf{Type}_1$，而 Type_0 和 Prop 则相互独立. Lean 的宇宙谱系并非渐增的，因此在 Lean 中 Prop 确实是完全独立于 Type_i 宇宙谱系之外的.

此外，若我们构造的项的类型是命题，我们常将 Π 写为 $\forall$，Σ 写作 $\exists$，$\times$ 写作 $\wedge$. 在下文中，若一个项的类型是命题，则我们都使用 $\forall$、$\exists$ 和 $\wedge$，以利读者清晰区分. 类型为命题的项也常直接被叫做“证明”.

2.2 直谓性与非直谓性

让我们回到之前关于宇宙谱系的论述. 在之前的课程中，我们提到过：如果我们有 $T = \Pi(x : A).B$，则 T 所居留的宇宙谱系级不可能低于 A 所居留的宇宙谱系. 更具体地，假设 $A = \mathcal{U}_i$，那么 T 所居留的宇宙谱系将至少为 $i+1$. 如果我们构造一个取值于 $\mathcal{U}_i$ 的 Π 类型，则该类型本身不可能在 $\mathcal{U}_i$ 中，我们称 $\mathcal{U}$ 的这一性质为**直谓性**（predicativity）.

而 $\mathcal{P}$ 与此相反；$\mathcal{P}$ 是**非直谓**（impredicative）的. 换言之，一个取值于所有命题的类型仍然是一个命题. 让我们用一个例子说明之.

例 2.1. 考虑这个项：$\mathsf{conj} = \lambda(p : P).\lambda(q : Q).(p, q)$，这里 $P, Q : \mathcal{P}$. 它的类型是 $\forall(P : \mathcal{P}).\forall(Q : \mathcal{P}).P \wedge Q$. 这个类型取值于所有的 $\mathcal{P}$，但由于 $\mathcal{P}$ 的非直谓性，它仍然可以居留于 $\mathcal{P}$ 中.

换言之，量化命题的命题仍然是命题. 这当然是符合我们对命题的直觉的.

到此时，大部分读者可能心存疑惑：为何 Coq 和 Lean 能够允许非直谓的 Prop 而不导致不一致呢？这是因为 Coq 和 Lean 都有一个特殊的规则限制 Prop 的使用；这个规则常被称为**命题消去规则** (proposition elimination rule).

该规则的完整表述较为复杂，但其内涵比较简单，所以我们在此用自然语言描述它. 设有两个项 $a : A$ 和 $p : P$，其中 $P : \mathcal{P}$，且 P 是一归纳类型. 在 a 的定义中，我们可能使用了对 p 的归纳（即 a 含有 P 的归纳子 ind_P）. 此时，命题消去规则要求 $A : \mathcal{P}$；换言之，我们可以对证明进行归纳，但只能用来构造其它证明. 这防止了利用 $\mathcal{P}$ 的非直谓性来构造类似罗素悖论的不一致性的可能.

例如，在 Coq 里，$x =_A y : \mathcal{P}$，所以我们没法通过对 $p : x =_A y$ 进行归纳来构造函数，只能用来证明别的命题.

按. 实际上，Lean 对此规则提供了一个例外，称为单子消去规则（singleton elimination）. 该规则指出，若一个归纳定义的命题只有一个构造器，那么可以不受命题消去规则的限制. 虽然这个规则已被证明是一致的，但它和同伦类型论（HoTT）的公理并不兼容，因而也无法使用 Lean 来进行 HoTT 中的证明.

Prop 的非直谓性和命题消去规则的存在间接导致 Prop 具有一些特殊的性质，我们接下来将简单讨论.

2.3 证明无关性

考虑一个命题 $P : \mathcal{P}$ 及其两个证明 $p, q : P$. 在许多情况下，我们并不在意同一个命题的两个证明 p, q 到底有何不同，而只关心其是否同一命题的证明. 让我们考虑一个数学中较为现实的例子.

例 2.2. 考虑一个幺半群（或其它代数结构，为图简洁我们只考虑幺半群）. 在类型论中，我们可以把一个幺半群描述为这样一个类型：

$$\begin{aligned}\mathsf{Mon}_M =&(M : \mathcal{U})\\ &\times(\_\cdot\_ : M \to M \to M)\\ &\times(e : M)\\ &\times(\mathsf{assoc} : \forall(a, b, c : M).a \cdot (b \cdot c) =_M (a \cdot b) \cdot c))\\ &\times(1_l : \forall(a : M).e \cdot a =_M a)\\ &\times(1_r : \forall(a : M).a \cdot e =_M e).\end{aligned}$$

假设我们有两个类型 Mon_M 和 Mon_M'，它们的基类型都是 M，所携带的二元运算 $\_\cdot\_$ 和单位元 e 也相同，那么我们自然认为它们描述的是同一个幺半群. 但如果我们想证明 Mon_M 和 Mon_M' 是同一的类型，这将会非常繁琐，因为我们将不得不证明 $\mathsf{assoc} = \mathsf{assoc}'$ 等等！如果我们采取了不同方法证明 $\_\cdot\_$ 的结合性，那么我们可能无法证明 $\mathsf{assoc} = \mathsf{assoc}'$；因为采取了不同的证明方法，我们会得到两个不同（或者至少是无法证明同一）的幺半群. 显然，这不符合我们的数学直觉！

让我们注意到 assoc 的类型是命题（也就是说 assoc 是一个证明）. 那么，为了防止被这些无关紧要的细节妨碍，我们可以加入这样一条规则：

$$\textsc{ProofIrrel}\quad \frac{\Gamma \vdash P : \mathcal{P} \qquad \Gamma \vdash p, q : P}{\Gamma \vdash p \equiv q}$$

Figure 3: 定义证明无关

亦即一个命题的所有证明都是定义等同的. 这一规则被称为（定义的）**证明无关性**（proof irrelevance）. 显然，这需要我们有一个单独的命题宇宙（如果给非命题的类型也加上这样的规则，那当然是十分荒谬的）. 并且，由于命题消去规则的存在，在命题宇宙之外无法“看到”一个证明的内部，所以我们知道这条规则应当是一致的.

当然，我们也可以不加入这条规则，而是改用公理的形式，以避免改动类型论本身：

公理 2.3 (命题证明无关). 对于一个命题 P，其任两个证明均命题等同. 用形式的语言描述即：$\forall(P : \mathcal{P}).\forall(p, q : P).p =_P q$.

由于相同的原因，我们知道这个公理也当是一致的.

Lean 的 Prop 宇宙是定义证明无关的. Coq 的 Prop 宇宙没有这个规则，但有另一个宇宙 SProp 拥有这条规则（除此之外，SProp 和 Prop 并无二致）. 当然，也可以给 Coq 的 Prop 加入命题证明无关性，这当然是一致的.

3 语用学的区别

上面的一节已经大致概括 CoC 和 MLTT 的区别；这也是 Coq 和 Lean 与 Coq 在理论层面上的主要区别. 但实际上，Coq 和 Lean 与 Agda 最大的区别是语用

的：两者的实际操作范式有着截然的不同．由于我们主要关注理论上的区别，在此只做简单的叙述，辅助以实际演示．当然，这一区别也和理论层面上的不同不无关系．

Coq 和 Lean 的操作范式是通过**策略**（tactic）间接构造项，而 Agda 主要是通过直接书写项的内容来构造项．让我们先通过例子进行一个直观的说明．

例 3.1. 我们想证明自然数加法的交换律，即 $+\text{-}\mathsf{comm} : (m : \mathbb{N}) \to (n : \mathbb{N}) \to m + n = n + m$.

一个直接式（即 Agda 式）的证明可能如此：

证明.

$$\begin{aligned} +\text{-}\mathsf{comm}(0, 0) &= \dots \\ +\text{-}\mathsf{comm}(\mathsf{succ}(m'), 0) &= \dots \\ +\text{-}\mathsf{comm}(0, \mathsf{succ}(n')) &= \dots \\ +\text{-}\mathsf{comm}(\mathsf{succ}(m'), \mathsf{succ}(n')) &= \dots \end{aligned}$$

具体的证明留作练习. □

而一个间接式的证明可能如此：

证明. 同时对 n 和 m 做归纳.

1. $n = m = 0$，即证.
2. $n = 0, m = \mathsf{succ}(m')$，由归纳法有 $m' + 0 = 0 + m'$. 则我们有：

$$\begin{aligned} \mathsf{succ}(m') + 0 &= \mathsf{succ}(m' + 0) && \text{由加法的定义} \\ &= \mathsf{succ}(0 + m') && \text{由归纳假设} \\ &= 0 + \mathsf{succ}(m') && \text{由加法定义.} \end{aligned}$$

其余情形留作证明. □

可见前者更接近类型论本身，而后者更接近我们一般数学证明的范式．也因此，Coq 和 Lean 被广泛用于分析等传统（主流）数学的形式化当中，而 Agda 较多被用来形式化综合同伦论、范畴论等领域．