arXiv:2304.04574v1 [cs.PL] 10 Apr 2023

依赖类型下的去函数化*

YULONG HUANG，英国剑桥大学JERE
MY YALLOP，英国剑桥大学

消除程序中高阶函数的去函数化翻译是许多编译器的重要组成部分。然而，针对依赖类型语言的去函数化尚未得到形式化研究。

我们提出了首个针对依赖类型语言的形式化去函数化翻译，并确立了可靠性和类型保持等关键元理论性质。该翻译适合纳入依赖类型语言的保持类型的编译器中。

CCS 概念：• **计算理论** → 类型论；程序语义；• **软件及其工程** →**编译器**。

Addit ional 关键词和短语：编译、类型保持、类型系统、依赖类型

ACM 引用格式：
Yulong Huang 和 Jeremy Yallop。2023。依赖类型下的去函数化。 *ACM* 程序设计语言会议录7, PLDI，第 127 号文章（2023年6月），30 页。 https://doi.org/10.1145/3591241

1 引言

类型越来越多地不仅用于 分类（即用于识别具有理想属性的程序子集），还用于 编译。一个 保持类型的编译器，组织为两个或更多带类型语言之间的一系列翻译 [例如 Bowman and Ahmed 2018年； Bowman et al. 2018年； Morrisett et al. 1999； Tarditi et al. 1996； Xiand Harper 2001], 可以支持诸如将源程序进行类型驱动的精化以转化为更明确的核心演算 [Kovács 2020； Wadler and Blott1989], 将不同的源语言特性翻译为简单的统一核心语言 [Sulzmannet al. 2007]以及类型驱动的优化 [Tarditiet al. 1996]。更一般地，在中间语言之间进行保持类型的翻译可以增强人们对编译过程正确性的信心 [Patrignaniet al. 2019]。

随着类型系统日益复杂，定义保持类型的转换带来了新的挑战。其中一些最显著的挑战出现在依赖类型语言的编译中，例如 Agda [Bove et al. 2009], Idris [Brady 2013], 和 Coq [Coq 开发团队 2022], , 这些语言的类型系统表达能力足够强，能够支持任意计算。事实证明，要将诸如延续传递风格转换、闭包转换以及转换为管理范式等经过长期研究的转换方法适配到依赖类型环境中并非易事[Bartheet al.1999； Barthe and Uustalu 2002； Bowman and Ahmed 2018； Bowman et al.2018； Koronkevich et al.2022]。

另一种此类转换——去函数化——能够从程序中消除高阶函数，是若干高阶语言编译器的重要组成部分 [例如，Braßel 2011；

*我们用不同的颜色和字体区分不同的语言，并建议阅读本文时使用彩色显示。

作者地址：Yulong Huang，剑桥大学，英国剑桥，yh419@cam.ac.uk；Jeremy Yallop，剑桥大学，英国剑桥，jeremy.yallop@cl.cam.ac.uk。

特此授权免费为个人或课堂用途制作本作品的部分或全部数字或纸质副本，但不得为营利或商业目的制作或分发副本，且副本须包含本声明及首页上的完整引文。本作品中第三方组件的版权必须得到尊重。其他用途请联系所有者/作者。© 2023 版权归所有者/作者所有。2475-1421/2023/6-ART127https://doi.org/10.1145/3591241

Pettyjohn 等人 2005；Podlovics 等人 2021；Weeks 2006]。去函数化的保类型变体适用于多种类型系统 [Bell 等人 1997；Nielsen 2000；Pottier and Gauthier2004]。去函数化对于依赖类型语言（例如 Idris）的编译也很有用 [1]。然而，迄今尚未针对依赖类型语言开发出去函数化翻译的保类型变体。

这项工作满足了这一需求，为一种依赖类型语言引入了带类型的去函数化翻译，并确立了其基本性质。与此前将类似的程序翻译调整为支持依赖类型的工作一样，我们也遇到并解决了在简单类型设置中不会出现的各种困难。特别是，保留宇宙大小（依赖类型语言利用它来避免不一致性）以及保留归约（利用它来确立类型相等性）的需求，使得直接调整标准去函数化变得不可行。

1.1 贡献

本文的核心贡献是首次为依赖类型语言提出保类型的去函数化翻译。具体而言，

- §2 表明，用于简单类型语言（§2.1）的保类型去函数化翻译无法扩展到依赖类型环境（§2.3），并提出了一种适用于依赖类型语言的抽象翻译（§2.4）。
- §3 介绍了我们保持类型的去函数化翻译的技术开发。§3.3 形式化定义了抽象翻译，§3.4 以及§3.5 确立了可靠性、类型保持和一致性等关键元理论性质。

最后，§4 介绍了我们翻译的实现（作为补充材料提供），而 §5 总结了类型保持编译和去函数化方面的相关工作。

2 概述

2.1 去函数化

去函数化去函数化翻译会将高阶程序转换为一阶程序，方法是将函数箭头 - →- 替换为一阶数据类型 - ⇝ -。去函数化会将源程序中的每个抽象 $\lambda x.e_i$ 替换为构造子应用 $C_i\ \overline{y}$，其中 C_i 是 - ⇝ - 的构造子，而 y 是该抽象的自由变量，并将每个应用 $f\ x$ 替换为 $f\$\ x$，其中中缀运算符 $ 将 C_i 映射回 e_i。

下面是一个示例。多态的compose 函数包含三个抽象，分别标记为F1、F2和F3。

```
compose :: (b → c) → (a → b) → (a → c)
compose = λf → λg → λx → f (g x)
                  F3
          F2
F1
```

去函数化 组合 会生成一个数据类型 ⇝ 其中每个抽象对应一个构造子。此处 → 分隔构造子参数：F2 有一个类型为 b ⇝ c 的参数，对应于 f 在上面的 F2 中，而 F3 有两个参数，分别对应于 f 和 g 在 F3 中。

```
data (⇝) a b where
  F1 :: (b ⇝ c) ⇝ (a ⇝ b) ⇝ (a ⇝ c)
  F2 :: (b ⇝ c) → (a ⇝ b) ⇝ (a ⇝ c)
  F3 :: (b ⇝ c) → (a ⇝ b) → (a ⇝ c)
```

[1] https://github.com/idris-lang/Idris-dev/blob/v1.3.4/src/IRTS/Defunctionalise.hs

继承 Pottier 和 Gauthier [2004], 去函数化 ⇝ 产生的数据类型是广义代数数据类型 （GADT），其中每个构造子的返回类型都可以是类型参数的不同实例化形式，并且构造子类型可以涉及在返回类型中未出现的类型变量（例如b，它出现在F3的类型中）。

去函数化还会生成一个运算符 $，它将 ⇝ 的构造子映射到相应抽象的主体：

```
($) :: (a ⇝ b) → a → b
F1      $ f = F2 f
F2 f    $ g = F3 f g
F3 f g $ x = f $ (g $ x)
```

这里 $ 将 F1 和参数 x 映射为 F2 x，因为抽象 F1 的主体是 F2，其中 x 是自由变量。同理，它将 F3 映射为 f $ (g $ x)，因为 F3 的主体是 f (g x)。

最后，去函数化在 compose 本身中，将 → 替换为 ⇝，并将 F1 替换为 F1：

```
compose_ :: (b ⇝ c) ⇝ (a ⇝ b) ⇝ (a ⇝ c)
compose_ = F1
```

以往对类型化情境下去函数化的研究考察了多种语言，从简单类型语言 [Nielsen 2000] 以及可单态化语言 [Bell 等人 1997] 到完全多态语言 [Pottier 和 Ga 2004]。在每种情况下，去函数化都使用归纳数据类型来表示函数：从用于简单类型函数的简单代数数据类型，到用于多态性的更强大的广义代数数据类型。

同样的方法若扩展到更强大的数据类型，是否能够支持对依赖类型程序进行去函数化？多态函数对类型进行抽象，并去函数化为按类型索引的GADT。类比而言，依赖函数对表达式进行抽象，是否能够去函数化为按表达式索引的归纳族？

2.2 归纳族

我们简要回顾归纳族及其相关限制。归纳族 [Dybjer1994] 通过允许按表达式进行索引，将普通归纳数据类型和广义代数数据类型加以一般化。

每个归纳族的构造子都可以返回一个以任意索引实例化的族实例。例如，在下面的 Agda [Norell 2008] 定义中，有限集归纳族的构造子 有限集由自然数n索引：

```
数据 有限集 : ℕ → 集合₀ 其中
  fzero : {n : ℕ} → 有限集(suc n)
  fsuc : {n : ℕ} → 有限集 n → 有限集 (suc n)
```

构造子 fzero 构造出一个类型为 有限集 (suc n) 的元素，其中 n 为任意值；而 fsuc 则构造出一个类型为 有限集 (suc n) 的元素，其来源是一个类型为 有限集 n 的元素。

一般来说，归纳族（不带参数）具有如下形式：

```
数据 D : (y₁ : T₁) → ... → (yₙ : Tₙ) → 集合_d 其中
  c₁ : A₁
  c₁ : A₂
  ...
```

其中 D 是 Set_d 中由类型 T_1、 ...、 T_n索引的归纳族。对于每个构造器 c_i，它接受若干个参数 $(z_i : S_i)$ 并构造类型为 $D\, t_1 \ldots t_n$的元素，其中每个 t_i 都是类型 T_i的表达式。具体而言，每个 A_i 都具有以下形式：

$(z_1 : S_1) \to \ldots \to (z_m : S_m) \to D\ t_1 \ldots t_n$.

D 和 c 的参数具有依赖类型： T_{i+1} 可以提及 $y_1 \ldots y_i$，而 S_{i+1} 可以提及 $z_1 \ldots z_i$。

为确保归纳族 y 定义是一致的，A $g^{da\ im}$ p施加了额外的限制。

首先，宇宙检查会拒绝带有非谓词构造子的归纳定义——也就是说，其构造子所处的宇宙比数据类型本身更大的定义。更具体地说，对于上述这样的数据类型 D，每个参数类型 S_i（即 S_i 的类型）都应小于 Set_d，才能通过宇宙检查。如果没有这一限制，归纳族就可以用来编码 Girard 悖论。

其次，正性检查 会拒绝包含自身在非严格正位置中引用的归纳族。如果没有这一限制，诸如下列的归纳族Fix 可以用来构造违反一致性的递归定义，例如 bad：

```
数据 Fix : 集合 → 集合 其中
  fix : ∀ {a} → (Fix a → a) → Fix a

糟糕 : ∀ {a} → a
坏 = f (fix f) where f : ∀ {a} → Fix a → a
                     f (fix g) = g (fix g)
```

严格正性对归纳族定义A_i 的构造子类型施加了两个条件D。首先，在D 出现的地方，其索引不得由涉及D 自身的表达式构成。其次，在参数类型S_i中，D 不得出现在函数箭头的左侧。

严格正性和宇宙方面的这些要求，是许多支持归纳族的依赖类型语言所共有的，例如 Coq 的 Gallina [Coq 开发团队 2022],Lean [de Moura 等人 2015],以及 Timany 和 Sozeau 的 pCuIC [Timany 和 Sozeau 2017]。

2.3 将去函数化扩展到依赖类型的问题

乍看之下，将去函数化扩展为支持依赖函数，并以归纳族为目标，似乎相当直接。作为示例，我们考虑对下面这个完全依赖的 compose函数进行去函数化。该函数使用 Agda 编写，并为清晰起见显式列出所有参数：

```
compose : (A : Set) → (B : A → Set) → (C : x : A) → B x → Set) →
          (f : (y : A) → (z : B y) → C y z) → (g : (x : A) → B x) → (x : A) →
          C x (g x)
组合 = λ A → λ B → λ C → λ f → λ g → λ x → f x (g x)
```

借鉴Pottier和 Gauthier 的方案，我们首先定义一个归纳族 Π 来表示依赖函数，就像 GADT ⇝ 表示非依赖函数一样：

```
数据 Π : (A : Set) → (A → Set) → Set其中
```

原程序中的每个依赖函数类型 $\Pi x{:}A.fx$（写作`(x : A) → f x` 在 Agda 中）都将被去函数化为 `Π A f`。

接下来，我们为原始程序中的每个 λ 抽象在 Π 中添加一个构造子。例如，`F6` 构造子对应于最内层抽象，其自由变量为 `A`、`B`、`C`、`f` 和`g`：

```
F6: (A : 集合) →
    (B : Π A (λ _ →Set)) →
    (C : Π A (λ x → Π (B $ x) (λ _ → Set))) →
    (f : Π A (λ y → Π (B $ y) (λ z →C $ y $ z))) →
```

```
  (g : Π A (λ x → B $ x)) →
  Π A (λ x →
  C $ x $ (g $ x))
```

最后，我们为 $ 的定义添加每个构造子的一个分支：

```
F6 A B C f g $ x = f $ x $ (g $ x)
```

附录 A 给出了完整的定义。遗憾的是，尽管这些定义在类型上正确，却不满足 Agda 的额外检查要求。

首先，宇宙检查 会拒绝 F6 的定义，因为在 B 的类型中，Π （即 λ_ → Set）的第二个参数属于宇宙 Set_1，，而该宇宙大于 Set。

其次，正性检查 会拒绝 F6 的定义，因为在 C 的类型中，Π 由一个包含 Π 的表达式进行索引。

最后，Agda 的终止性检查 会拒绝 $ 的定义，因为针对 F6 的分支并非结构递归终止。

2.3.1 一个更简单的示例。上面的示例表明，尽管去函数化看起来可以自然地扩展到依赖类型，但这种扩展会遇到一致性问题。事实上，情况更加严重：即使我们不使用依赖性，宇宙和正性方面也会出现同样的问题。

例如，这里有一个简单类型的 组合函数，基于固定类型 A、 B和 C：

```
compose : (B → C) → (A → B) → (A → C)
compose = λ f → λ g → λ x → f (g x)
```

去函数化 compose会生成一个归纳族 ⇝ 以及相应的 *apply* 函数 $:data _⇝_ : Set → Set → Setwhere_ _

```
  F1: (B ⇝ C) ⇝ (A ⇝ B) ⇝ (A ⇝ C)
  F2: (B ⇝ C) → (A ⇝ B) ⇝ (A ⇝ C)
  F3: (B ⇝ C) → (A ⇝ B) → (A ⇝ C)
```

```
_$_ : ∀ {A B} → (A ⇝ B) → A
→ B F1 $ f =F2 f F2 f$ g =
F3 f g F3 f g $ x = f $ (g $ x)
```

遗憾的是，这个简单的定义 ⇝ 存在与依赖性更强的定义相同的问题 Π。首先，宇宙检查会拒绝构造子F1，因为类型B ⇝ C 属于宇宙Set_1，，而该宇宙比Set更大。其次，在F1的类型中，⇝ 以 ⇝ 自身为索引，因此该定义无法通过正性检查。最后，F3 分支中的$ 无法通过终止性检查，因为递归调用的参数在结构上并不比参数本身更小。

2.3.2 表达能力不匹配。值得注意的是，Agda 的限制只是对语义性质相当粗略的语法近似；违反这些限制的程序未必就“不正确”。其他人也采用了类似的方法，例如Ahrens 等人 [2018]（用于宇宙检查），以及Weirich 和 Casinghino [2010]（用于全部三项检查）。

然而，我们并不赞成以这种方式移除去函数化所生成代码中的安全防护机制。在我们看来，Agda 拒绝去函数化生成的归纳族这一事实表明，归纳族并不适合这项任务。例如，拒绝 Π 的构造子的宇宙限制并不适用于与之对应的闭包

$$
\begin{array}{lll}
\mathfrak{D} ::= & \mathfrak{L}_3(\{f : B \to C, g : A \to B\}, x : A & \mapsto \quad f\,@\,(g\,@\,x) : C), \\
& \mathfrak{L}_2(\{f : B \to C\}, g : (A \to B) & \mapsto \quad \mathfrak{L}_3\{f, g\} : A \to C), \\
& \mathfrak{L}_1(\{\}, f : (B \to C) & \mapsto \quad \mathfrak{L}_2\{f\} :\to (A \to B) \to A \to C) \\
\text{组合} & ::= \quad \mathfrak{L}_1\{\}
\end{array}
$$

图 1. 去函数化的简单类型组合

源程序中的那些构造子：没有任何要求抽象主体中的自由变量必须归属于比函数自身更小的宇宙。额外的限制源于表达能力不匹配：只有在归纳族未以类似闭包的方式使用时，才需要宇宙限制——例如，在提取构造子参数时。

2.4 抽象去函数化

上述示例表明，将去函数化扩展到依赖类型后是保持类型的。还可以证明它是保持含义的。正如Pottier 和 Gauthier [2006]所观察到的，当去函数化生成单个多态 apply 函数时，它就等同于无类型转换。Pottier 和Gauthier利用这一等同性，通过提升关于无类型转换的证明，证明了有类型转换是保持含义的。类似地，我们或许可以将该证明提升到依赖类型设置中，以确立扩展转换的正确性。

扩展后的去函数化翻译似乎能够保留类型和含义，因此令人遗憾的是，它未能通过 Agda 的各种限制检查。我们该如何构造一种不会违反这些检查的翻译?

我们选择沿着 Minamide 等人 [1996] 和 Bowman andAhmed[2018] 在 抽象闭包转换中采用的方向前进。抽象闭包转换研究的是如何将闭包转换为一种专用目标语言，该语言包含用于表示闭包和闭包类型的新构造。将闭包转换为这些构造，既抓住了翻译的本质，又避免了更具体场景所施加的不必要限制。类似地，我们将定义一种目标语言，称为 去函数化构造演算 (DCC)，它采用 λ 演算的风格，但加入了一种新构造，用于表示去函数化的 标签 (即指向 标签上下文的索引)，以取代 λ 抽象。

图 1 展示了将简单类型的compose 函数去函数化为 DCC [2]的结果，其外观和行为与 §2.1中介绍的传统去函数化相似。在我们向 DCC 的翻译中，每个抽象 $\lambda x.e_i$ 都会被替换为一个标签表达式 $\mathfrak{L}_i\{y\}$ 其中 $\mathfrak{L}_i$ 是标签的标识符，y 是该抽象的自由变量。函数主体 e_i 存储在一个独立的标签上下文 $\mathfrak{D}$ 中，该上下文以标签标识符为索引，并附有相应的类型信息。

在图 1中，标签上下文 $\mathfrak{D}$ 有三个条目，原始 compose函数中的每个抽象对应一个条目。每个条目对应传统去函数化中$ 函数的一种情形。例如，$\mathfrak{L}_3$ 源自对 $\lambda x.\ A.\ f(g\ x)$的翻译，并对应于$定义中的 F3情形：它有两个自由变量 f : B → C 和 g : A → B，一个绑定变量x，以及一个主体 f @ (g@ x)。正如我们将看到的，标签应用 $\mathfrak{L}_3\{f, g\}$ @ N 归约为 f @ (g @ N)，正如应用 (F3 f g) $x 归约为相应的右侧 f $ (g $ x)。

向这一 SCHEME 添加依赖类型很直接，但需要谨慎定义变换，并证明它具有所需的元理论性质。特别是，正如我们将看到的，该变换需要考虑整个推导树，而不仅仅是源语言表达式（§3.3.2），并且我们需要使用一个带有

[2]我们在此假设 A、B 和 C 是基本类型。

显式替换（§3.4），以使类型保持性证明得以成立。这些挑战只会出现在依赖类型环境下的去函数化中，而此前尚未对此进行研究。

3 使用依赖类型进行去函数化

在非正式介绍了关键概念并阐述了我们抽象的去函数化翻译的动机之后，现在我们转向技术细节。接下来的几节将介绍我们的源语言、构造演算（§3.1）、我们的目标语言、去函数化的构造演算（§3.2），以及将它们联系起来的去函数化翻译（§3.3）。然后，我们建立该翻译的可靠性（§3.4），并证明目标语言的一致性（§3.5）。

3.1 构造演算

我们的源语言是构造演算（CC）[Coquand 和 Huet 1988],的一种变体，这是一种表达力强的依赖类型 λ 演算，是若干编程语言和证明助手的基础。与 CC 的原始表述相比，我们的主要不同之处在于遵循 Luo [1990], 以及 Agda、Lean、Coq 和 F 等许多依赖类型语言所采用的方法，通过为 CC 扩展 Martin-Löf 风格的宇宙层级。

下面给出 CC 定义的一个例子：compose，它表示函数 f和 g 的完全依赖复合函数：

$$\begin{aligned}\text{compose} \quad ::= \quad & \lambda A{:}U_0.\ \lambda B{:}(\Pi x{:}A.U_0).\ \lambda C{:}(\Pi x{:}A.\Pi y{:}B\ x.U_0).\\ & \lambda f{:}(\Pi y{:}A.(\Pi z{:}B\ y.C\ y\ z)).\ \lambda g{:}(\Pi x{:}A.B\ x).\\ & \lambda x{:}A.\ f\ x\ (g\ x)\end{aligned}$$

该定义中的表达式部分 $\lambda A.\lambda B.\lambda C.\lambda f.\lambda g.\lambda x.f\ x\ (g\ x)$ 并无特别之处；真正有趣的是类型对参数的依赖关系。特别是，f 的结果类型 $C\ y\ z$ 取决于 f 的参数 y和 z，而 g 的结果类型 $B\ x$ 取决于 g的参数 x。（在实际的编程语言中，y 以及类型参数 A、B 和 C 都会隐式传递，但我们的最小演算不支持隐式参数。）

图 2a 展示了 CC 的语法。CC 的表达式包括变量 x（从无限个名称中取值）、宇宙 U、依赖函数类型 $\Pi x{:}A.B$、应用 $L\ M$以及抽象$\lambda x{:}A.M$。CC 上下文 Γ 是由变量—表达式对组成的望远镜。

CC 有四个判断：

（1）归约（图2b）

$$M \triangleright N$$

（2）类型归属（图 2c）

$$\Gamma \vdash M : A$$

（3）上下文构造（图 2d）

$$\vdash \Gamma$$

（4）等价（图2e）

$$\vdash A \equiv B$$

只有一条归约规则（图 2b），用于 β-归约，$(\lambda x{:}A.N)\ M \ \triangleright N\ [M/x]$。我们用$L \triangleright^* M$ 表示 L 经过零步或多步归约到 M。

CC 的类型判断规则（图 2c）和上下文构造规则（图 2d）由相互归纳定义。变量 x 的类型是 A，当且仅当 $x : A$ 出现在良构上下文 Γ （Ty-Var）中。宇宙 U_i的类型是 U_{i+1} （Ty-Universe），而 $\Pi x{:}A.B$的类型是以下两者中更高的宇宙

Expressions	A, B, L, M, N	::=	$x \mid U \mid \Pi x{:}A.B \mid L\ M \mid \lambda x{:}A.M$
Universes	U	::=	U_i
Contexts	Γ	::=	$\cdot \mid \Gamma, x : A$

(a) 语法

$\boxed{M \triangleright N}$ （归约）

$$\frac{}{(\lambda x{:}A.N)\ M \triangleright N[M/x]}\ \textsc{red-Beta}$$

(b) 归约

$\boxed{\Gamma \vdash M : A}$ *(Typing)*

$$\frac{x : A \in \Gamma \qquad \vdash \Gamma}{\Gamma \vdash x : A}\ \text{类型-变量} \qquad \frac{\vdash \Gamma}{\Gamma \vdash U_i : U_{i+1}}\ \text{类型-宇宙}$$

$$\frac{\Gamma \vdash A : U_i \qquad \Gamma, x : A \vdash B : U_j}{\Gamma \vdash \Pi x{:}A.B : U_{max(i,j)}}\ \Pi\text{规则} \qquad \frac{\Gamma \vdash M : \Pi x{:}A.B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[N/x]}\ \text{ty-Apply}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x{:}A.M : \Pi x{:}A.B}\ \text{类型 Lambda} \qquad \frac{\Gamma \vdash M : A \qquad \Gamma \vdash B : U \qquad \vdash A \equiv B}{\Gamma \vdash M : B}\ \text{ty-Eqiv}$$

(c) 类型判断

$\boxed{\vdash \Gamma}$ （良构性）

$$\frac{}{\vdash \cdot}\ \text{wf-Empty} \qquad \frac{\vdash \Gamma \qquad \Gamma \vdash A : U}{\vdash \Gamma, x : A}\ \text{wf-Cons}$$

(d) 上下文构造

$\boxed{\vdash M \equiv N}$ *(等价)*

$$\frac{L \triangleright^* N \qquad M \triangleright^* N}{\vdash L \equiv M}\ \text{eq-Reduce} \qquad \frac{L \triangleright^* \lambda x : A.L' \qquad M \triangleright^* M' \qquad \vdash L' \equiv M'\ x}{\vdash L \equiv M}\ \text{eq-Eta1} \qquad \frac{L \triangleright^* L' \qquad M \triangleright^* \lambda x{:}A.M' \qquad \vdash L'\ x \equiv M'}{\vdash L \equiv M}\ \text{eq-Eta2}$$

（e）等价

图 2. 构造演算（CC）

宇宙 A 和 B （Π规则）。如果 M 在扩展了 $x{:}A$的某个上下文 Γ 中具有类型 B，那么 λx:A.M具有依赖函数类型 Πx:A.B（ty-Lambda）。应用的类型为 $B[N/x]$，因为函数类型的输出可能依赖于参数 N（ty-Apply）。最后，如果表达式 M具有类型 A，且 A 等价于 B，那么 M 也具有类型 B （ty-Eqiv）。

如果上下文 Γ 是良构的（记作 $\vdash \Gamma$），则称其良构；也就是说，其中的每个变量都关联着一个有效类型——即该变量所关联表达式的类型在上下文 Γ中是一个宇宙。

我们使用两种简写：写作 $\Gamma \vdash A : U$ 表示 $\Gamma \vdash A : U_i$ 对于某个 i（这意味着 A 是一个类型），并用 $A \to B$ 表示 Π-类型 $\Pi x{:}A.B$，其中 B 不依赖于

x。为简便起见，我们从形式化定义中省略单位类型 1 和自然数 $\mathbb{N}$，但会在示例中自由使用它们。

在 CC 中，如果两个表达式归约为相同的表达式（eq-Reduce），或者如两个对称规则eq-Eta1和eq-Eta2 所定义的那样是 η-等价的，则它们是等价的（图2e）。在 eq-Eta1下，如果 L 归约为抽象 λx:A.L′，M归约为某个 M'，并且 $L' \equiv M'\,x$，则 L和 M等价；而 eq-Eta2 对称地对应于 [BowmanandAhmed 2018年; Bowman et al. 2018]。

CC 的一个有用性质如下：如果 $\Gamma \vdash M : A$，那么 $\Gamma \vdash A : U$。此外，CC 是类型安全且一致的，并且 CC 中的类型检查是可判定的 [Coquand 和 Huet 1988年; Luo 1990]。

3.2 去函数化构造演算

图 3a 展示了我们目标语言去函数化构造演算（DCC）的语法。与 CC 一样，DCC 表达式包括变量 x、宇宙 U、依赖函数类型Πx:A.B和应用 L @M。与 CC 不同，DCC 包含一等函数标签 $\mathfrak{L}\{M\}$，而非 λ 抽象。

一个标签表达式 $\mathfrak{L}\{M\}$是一个标签名 $\mathfrak{L}$ 配有一个包含零个或多个表达式的列表 M（表示 $M_1, \cdots, M_n$）分配给其自由变量。标签名 $\mathfrak{L}_1$，$\mathfrak{L}_2$，$\cdots$与变量名不相交，我们通过使用不同的字体来强调这一点。

DCC 中有两种上下文。与 CC 中一样，类型上下文 Γ 将变量 x 与类型 A 关联起来。标签定义上下文 $\mathfrak{D}$将标签名与其关联的数据配对：$\mathfrak{L}(\{x : A\}, x : A \mapsto M : B)$。这里，$\overline{x : A}$ 记录标签所接受的（可能为空的）自由变量望远镜的类型，$(x : A) \to B$ 指定标签类型，而 M 是应用于一个参数时该标签归约到的表达式。注意，类型上下文 Γ 中的类型可以引用标签上下文 $\mathfrak{D}$中的标签 $\mathfrak{L}_1, \mathfrak{L}_2, \cdots$，反之则不行。

DCC 有四个判断：

（1）归约（图3b）

$$\mathfrak{D} \vdash M \triangleright N$$

（2）类型成员关系（图 3c）

$$\mathfrak{D}; \Gamma \vdash M : A$$

（3）上下文构造（图 3d）

$$\vdash \mathfrak{D}; \Gamma$$

（4）等价（图3e）

$$\mathfrak{D} \vdash A \equiv B$$

3.2.1 归约。只有一条归约规则（图 3b），用于标签应用：将标签 $\mathfrak{L}\{M\}$ 应用于参数 N 归约为 L[M/x,N/x],，其中 L 是标签上下文中条目 $\overline{\mathfrak{L}}$的主体，而 M 是 $\mathfrak{L}$的闭包。归约序列记作 $\mathfrak{D} \vdash M \triangleright^* N$，表示 M 经过零步或多步 归约为 N 。

DCC 中针对变量、宇宙、 Π-类型和应用的替换遵循传统定义。针对标签的替换是

$$\mathfrak{L}\{\overline{M}\}[N/x] \triangleq \mathfrak{L}\{\overline{M[N/x]}\},$$

其中 $\overline{M[N/x]}$ 是以下内容的语法糖：$M_1[N/x], \cdots, M_n[N/x]$。

3.2.2 类型判断。DCC 的类型判断形式为 $\mathfrak{D}$；$\Gamma \vdash M :\ A$，类型规则见图3c。变量、宇宙、 Π-类型、应用和转换的规则均相同

宇宙	U	::=	U_i
表达式	A, B, L, M, N	::=	$x \mid U \mid \Pi x{:}A.B \mid L \mathbin{@} M \mid \mathfrak{L}\{\overline{M}\}$
类型上下文	Γ	::=	$\cdot \mid \Gamma, x : A$
标签上下文	$\mathfrak{D}$	::=	$\cdot \mid \mathfrak{D}, \mathfrak{L}(\{\overline{x : A}\}, x : A \mapsto M : B)$
DCC 上下文	$\mathfrak{D}; \Gamma$		

(a) Syntax

$\boxed{\mathfrak{D} \vdash M \triangleright N}$ *(归约)*

$$\frac{\mathfrak{L}(\{\overline{x} : \overline{A}\}, x{:}A \mapsto L : B) \in \mathfrak{D}}{\mathfrak{D} \vdash \mathfrak{L}\{\overline{M}\} \mathbin{@} N \triangleright L[\overline{M}/\overline{x}, N/x]}\ \textsc{d-red-Beta}$$

(b) 归约

$\boxed{\mathfrak{D};\ \Gamma \vdash M :\ A}$ *(类型判断)*

$$\frac{x : A \in \Gamma \quad \vdash \mathfrak{D}; \Gamma}{\mathfrak{D};\ \Gamma \vdash x :\ A}\ \text{d-ty-Var} \qquad \frac{\vdash \mathfrak{D}; \Gamma}{\mathfrak{D}; \Gamma \vdash U_i : U_{i+1}}\ \text{d-ty-Universe}$$

$$\frac{\mathfrak{D}; \Gamma \vdash A : U_i \quad \mathfrak{D};\ \Gamma,\ x :\ A \vdash B :\ U_j}{\mathfrak{D};\ \ \Gamma \vdash \Pi x :\ A_{\circ}\ B : U^{\max(i,j)}}\ \text{d-ty-Pi} \qquad \frac{\mathfrak{D}; \Gamma \vdash M : \Pi x{:}A.B \quad \mathfrak{D}; \Gamma \vdash N : A}{\mathfrak{D};\ \ \Gamma \vdash M \mathbin{@} N : B[N/x]}\ \text{d-ty-Apply}$$

$$\frac{\mathfrak{D};\ \Gamma \vdash \overline{M :\ A} \quad \mathfrak{L}(\{\overline{x} : \overline{A}\}, x{:}A \mapsto M : B) \in \mathfrak{D}}{\mathfrak{D}; \Gamma \vdash \mathfrak{L}\{\overline{M}\} : \Pi x{:}A[\overline{M}/\overline{x}].B[\overline{M}/\overline{x}]}\ \text{d-ty-Label} \qquad \frac{\mathfrak{D};\ \Gamma \vdash M :\ A \quad \mathfrak{D};\ \Gamma \vdash B :\ U_i \quad \mathfrak{D} \vdash A \equiv B}{\mathfrak{D}; \Gamma \vdash M : B}\ \text{d-ty-Eqiv}$$

(c) 类型判断

$\boxed{\vdash \mathfrak{D}; \Gamma}$ （良构性）

$$\frac{}{\vdash \cdot; \cdot}\ \text{d-wf-Empty} \qquad \frac{\mathfrak{D}; \overline{x} : \overline{A}, x : A \vdash M : B}{\vdash \mathfrak{D},\ \ \mathfrak{L}(\{\overline{x : A}\},\ x{:}A \mapsto M{:}B);\ \cdot}\ \text{d-wf-Label} \qquad \frac{\mathfrak{D}; \Gamma \vdash A : U_i}{\vdash \mathfrak{D}; \Gamma, x : A}\ \text{d-wf-类型}$$

(d) 上下文与标签上下文的构造

$\boxed{\mathfrak{D} \vdash M \equiv N}$ （等价）

$$\frac{\mathfrak{D} \vdash L \triangleright^* N \quad \mathfrak{D} \vdash M \triangleright^* N}{\mathfrak{D} \vdash L \equiv M}\ \text{d-eq-Reduce}$$

$$\frac{\mathfrak{D} \vdash L \triangleright^* \mathfrak{L}\{\overline{N}\} \quad \mathfrak{D} \vdash M \triangleright^* M' \quad \mathfrak{L}(\{\overline{x} : \overline{A}\}, x{:}A \mapsto N : B) \in \mathfrak{D} \quad \mathfrak{D} \vdash N[\overline{N}/\overline{x}] \equiv M' \mathbin{@} x}{\mathfrak{D} \vdash L \equiv M}\ \text{d-eq-Eta1}$$

$$\frac{\mathfrak{D} \vdash L \triangleright^* L' \quad \mathfrak{D} \vdash M \triangleright^* \mathfrak{L}\{\overline{N}\} \quad \mathfrak{L}(\{\overline{x} : \overline{A}\}, x{:}A \mapsto N : B) \in \mathfrak{D} \quad \mathfrak{D} \vdash L' \mathbin{@} x \equiv N[\overline{N}/\overline{x}]}{\mathfrak{D} \vdash L \equiv M}\ \text{d-eq-Eta2}$$

(e) 等价

图 3. 去函数化构造演算（DCC）

与 CC 中对应的规则相比，因此我们重点关注标签规则。标签项 $\mathfrak{L}\{M\}$ 在以下条件满足时是类型良好的：$\mathfrak{D}$；Γ。

（1）上下文 $\mathfrak{D}$；Γ 是良构的。（2）$\mathfrak{L}$（x：A），x：$A \mapsto M$：B）存在于 $\mathfrak{D}$。（3）两个列表 M 和 x：A 的长度相等。（4）M 中的所有表达式都是类型正确的，并且其类型与自由变量 A 的指定类型相匹配。

具体而言，条件（4）意味着：

$$\begin{aligned}&\mathfrak{D};\Gamma \vdash M_1 : A_1,\\ &\mathfrak{D};\Gamma \vdash M_2 : A_2[M_1/x_1],\\ &\cdots,\\ &\mathfrak{D};\Gamma \vdash M_n : A_n[M_1/x_1,\cdots,M_{n-1}/M_{n-1}].\end{aligned}$$

每个 A_{i+1} 都依赖于 $x_1,\cdots,x_i$，因此需要将 $M_1,\cdots,M_i$ 代入 A_{i+1} 中，以构成 M_{i+1} 的类型判断。$\mathfrak{L}\{M\}$ 的类型是 $\Pi x{:}A[M/x].B[M/x]$。

请注意，自由变量 M 的值会代入 Πx：A。B 是标签的指定类型。我们使用 $[M/x]$ 作为 $[M_1/x_1,\cdots,M_n/x_n]$ 的语法糖，并约定将条件（3）和（4）简写为 $\mathfrak{D}$；$\Gamma \vdash M$：A。

DCC 对良构上下文的判断为 $\vdash \mathfrak{D};\Gamma$，其规则见图3d。如果类型上下文中的每个变量都与一个有效类型（在前一个上下文 $\mathfrak{D};\Gamma$ 中）相关联，并且每个标签都与类型正确的数据相关联，则上下文是良构的。换句话说，如果我们有 $\mathfrak{L}(\{x : A\}, x{:}\ A \mapsto M : B)$，那么 M 应具有类型 B，正如由前一个标签上下文和 M 中的自由变量（即 $\mathfrak{D}; x : A, x : A$）构成的上下文所规定的那样。

项 L 和 M 等价（图3e），当且仅当它们在同一个归约序列中都归约为同一项 N，或者它们是 η-等价的。DCC 的 η-等价规则与 CC 的类似。规则 (d-eq-Eta1) 规定：如果 L 归约为一个标签 $\mathfrak{L}\{N\}$，M 归约为 M'，$\mathfrak{L}(\{x{:}\ A\}, x : A \mapsto N : B)$ 出现在标签上下文 $\mathfrak{D}$ 中，并且 $M'\ @\ x$ 与 $N[N/x]$ 等价，则 L 和 M 等价。规则 (d-eq-Eta1) 和 (d-eq-Eta2) 是对称的。

类型上下文和标签上下文都具有弱化性质：良类型表达式在扩展后的类型上下文或标签上下文中仍然是良类型的（通过对类型推导规则进行归纳可得）。

引理 3.1（类型弱化）。如果 $\mathfrak{D};\Gamma \vdash M : A, \mathfrak{D};\Gamma \vdash B : U_i$，且 x 是新鲜的，那么 $\mathfrak{D};\Gamma, x : B \vdash M : A$。

引理 3.2（标签弱化）。如果 $\mathfrak{D};\Gamma \vdash M : C$，$\mathfrak{D}; x : A, \overline{x : A} \vdash N : B$，且 $\mathfrak{L}_i$ 是新鲜的，那么 $\mathfrak{D}, \mathfrak{L}_i(\{\overline{x : A}\}, x{:}\ A \mapsto N : B); \Gamma \vdash M : C$。

DCC 是类型安全且一致的（我们在 §3.5 中证明这些性质）。此外，它具有足够的表达能力来支持 compose 函数，但我们必须以去函数化风格编写它，因为该演算不支持 λ 抽象。标签上下文中有一个条目 $\mathfrak{D}$ 对应于 λ CC 中 compose 的定义：

$$\begin{aligned}\mathfrak{D} \quad ::= \quad &\mathfrak{L}_5(\{A,B,C,f,g\}, x : A \mapsto (f\ @\ x)\ @\ g\ @\ x : (C\ @\ x)\ @\ g\ @\ x),\\ &\cdots,\\ &\mathfrak{L}_1(\{A : U_0\}, B : (\Pi x{:}A.U_0) \mapsto \mathfrak{L}_2\{A,B\} : \Pi C.\Pi f.\Pi g.\Pi x.(C\ @\ x)\ @\ g\ @\ x),\\ &\mathfrak{L}_0(\{\}, A : U_0 \mapsto \mathfrak{L}_1\{A\} : \Pi B.\Pi C.\Pi f.\Pi g.\Pi x.(C\ @\ x)\ @\ g\ @\ x)\end{aligned}$$

而 compose 本身的定义只是从上下文中投影出一个闭合标签：

$$\text{组合} ::= \quad \mathfrak{L}_0\{\}$$

完整定义见附录 B。

3.3 去函数化翻译

$\boxed{\Gamma \vdash M : A \rightsquigarrow \mathsf{M}}$ *(Expression transformation)*

$$\frac{}{\Gamma \vdash x : A \rightsquigarrow \mathsf{x}}\ \textsc{t-Var} \qquad \frac{}{\Gamma \vdash U_i : U_{i+1} \rightsquigarrow \mathsf{U_i}}\ \textsc{t-Universe}$$

$$\frac{\Gamma \vdash A : U_i \rightsquigarrow \mathsf{A} \quad \Gamma, x : A \vdash B : U_j \rightsquigarrow \mathsf{B}}{\Gamma \vdash \Pi x{:}A.B : U_{max(i,j)} \rightsquigarrow \Pi \mathsf{x}{:}\mathsf{A}.\mathsf{B}}\ \textsc{t-Pi} \qquad \frac{\Gamma \vdash L : \Pi x{:}A.B \rightsquigarrow \mathsf{L} \quad \Gamma \vdash M : A \rightsquigarrow \mathsf{M}}{\Gamma \vdash L\ M : B[M/x] \rightsquigarrow \mathsf{L}\ @\ \mathsf{M}}\ \textsc{t-Apply}$$

$$\frac{FV(\lambda^i x{:}A.M) = \overline{x}{:}\overline{A} \quad \Gamma \vdash \overline{x} : \overline{A} \rightsquigarrow \overline{\mathsf{x}}}{\Gamma \vdash \lambda^i x{:}A.M : \Pi x{:}A.B \rightsquigarrow \mathfrak{L}_\mathsf{i}\{\overline{\mathsf{x}}\}}\ \textsc{t-Lambda} \qquad \frac{\Gamma \vdash M : A \rightsquigarrow \mathsf{M} \quad \Gamma \vdash B : U \quad \vdash A \equiv B}{\Gamma \vdash M : B \rightsquigarrow \mathsf{M}}\ \textsc{t-Equiv}$$

(a) 表达式的去函数化

$\boxed{\Gamma \vdash M :\ A\ \rightsquigarrow_d \mathfrak{D}}$ *(函数提取)*

$$\frac{\Gamma \vdash A : \text{宇宙}\ \rightsquigarrow_d \mathfrak{D}}{\Gamma \vdash x : A\ \rightsquigarrow_d \mathfrak{D}}\ \text{d-变量} \qquad \frac{}{\Gamma \vdash U_i :\ \ U_{i+1} \rightsquigarrow_d .}\ \text{d-宇宙}$$

$$\frac{\Gamma \vdash \text{一个} :\ U_i \rightsquigarrow_d \mathfrak{D}_A \quad \Gamma, x : A \vdash B :\ U_j \rightsquigarrow_d \mathfrak{D}_B}{\Gamma \vdash \Pi x{:}A.B :\ \ U_{max(i,j)} \rightsquigarrow_d \mathfrak{D}_A \cup \mathfrak{D}_B}\ \textsc{d-Pi}$$

$$\frac{\Gamma \vdash A :\ U\ \rightsquigarrow_d \mathfrak{D}_A \quad \Gamma, x :\ A \vdash M :\ B\ \rightsquigarrow_d \mathfrak{D}_M \quad FV(\lambda^i x{:}A.M) = x{:}\overline{A} \quad \Gamma \vdash \overline{A} : U \rightsquigarrow \overline{\mathsf{A}} \quad \Gamma, x : A \vdash M : B \rightsquigarrow \mathsf{M} \quad \Gamma \vdash \Pi x{:}A.B : U \rightsquigarrow \Pi \mathsf{x}{:}\mathsf{A}.\mathsf{B}}{\Gamma \vdash \lambda^i x{:}A.M : \Pi x{:}A.B \rightsquigarrow_d \mathfrak{D}_A \cup \mathfrak{D}_M, \mathfrak{L}_\mathsf{i}(\{\overline{\mathsf{x}} : \overline{\mathsf{A}}\}, \mathsf{x}{:}\mathsf{A} \mapsto \mathsf{M} : \mathsf{B})}\ \text{d-Lambda}$$

$$\frac{\Gamma \vdash M :\ \ \Pi x{:}\ A.B\ \rightsquigarrow_d \mathfrak{D}_1 \quad \Gamma \vdash N : A\ \rightsquigarrow_d \mathfrak{D}_2 \quad \Gamma \vdash 乙[N/x] :\ \ U\ \rightsquigarrow_d \mathfrak{D}_3}{\Gamma \vdash M\ N :\ \ B[N/x] \rightsquigarrow_d \mathfrak{D}_1 \cup \mathfrak{D}_2 \cup \mathfrak{D}_3}\ \text{d-Apply} \qquad \frac{\Gamma \vdash M : A\ \rightsquigarrow_d \mathfrak{D} \quad \Gamma \vdash B :\ U\ \rightsquigarrow_d \mathfrak{D}_B \quad \vdash A\ \equiv B}{\Gamma \vdash M :\ \ B\ \rightsquigarrow_d \mathfrak{D} \cup \mathfrak{D}_B}\ \text{d-Eqiv}$$

(b) 函数定义的提取

图 4. 去函数化翻译

图 4展示了这种翻译。它由两部分组成：一个用于表达式的变换 $[\![-]\!]$，以及一个从源程序中提取函数定义的元函数 $[\![-]\!]_d$。表达式变换生成目标程序，而元函数 $[\![-]\!]_d$ 给出一个标签上下文。

3.3.1 表达式变换。

定义3.3。 表达式变换 $[[-]]$ 接受 *CC* 中一个类型良好的项，以及该项类型推导的一个隐式参数。我们定义 $[[M]]$， M，其中 M 由一种新形式的判断给出 $\Gamma \vdash M:\ A \rightsquigarrow \mathsf{M}$（图 *4a*）。

该变换只需将 CC 中的变量、宇宙、 Π-类型、应用、基本类型和值，组合式地转换为 DCC 中对应的内容。源语言中的函数会被转换为目标语言中的标签。

去函数化要求每个标签与每个源程序函数之间存在唯一对应关系。我们约定，变换输入中的每个 lambda M 都标记有唯一标识符 i （$i \in \mathbb{N}$），其对应标签的名称为$\mathfrak{L}_i$。

此变换将函数 $\lambda^i x{:}A.M$ 转换为标签$\mathfrak{L}_i\{\mathsf{x}\}$，其中 x 来自函数的自由变量 x (T-Lambda)。元函数 FV（见定义 3.4）计算良类型 CC 表达式中涉及的所有自由变量及其类型。注意，FV 与 fv 不同，后者是传统的自由变量函数，用于计算表达式中的所有未绑定变量。在依赖类型语言中，自由变量的类型可能包含其他自由变量，而这些变量的类型又可能包含其他自由变量，如此等等！因此，$\mathrm{FV}(M)$ 必须递归地找出使 M 类型正确所需的所有变量。

定义 3.4。 有限集(M) 将 $\Gamma \vdash M:\ M$ 的类型判断作为隐式参数。它首先计算 M 和 A 中所有未绑定的变量 x_1、$\cdots$ x_n，然后递归地对这些变量的类型调用自身，最后返回它找到的所有自由变量及其类型的并集。

$$\begin{array}{lll} FV(M) & = & FV(A_1) \cup \cdots \cup FV(A_n) \cup \Gamma_{fv} \\ & where & \mathrm{fv}\ (M) \cup \mathrm{fv}\ (A) = x_1, \cdots, x_n \\ & & \Gamma \vdash x_1 : A_1, \cdots, \Gamma \vdash x_n : A_n \\ & & \Gamma_{fv} \triangleq x_1 : A_1, \cdots, x_n : A_n. \end{array}$$

这里，两个类型上下文 $\Gamma_1 \cup \Gamma_2$ 与 Γ_1 的并集，是在 $\Gamma_1 \cup \Gamma_2$ 后追加仅出现在 Γ_2 中的所有变量-表达式对 $x{:}A$，并保持其顺序。直观而言， $FV(M)$ 计算出正确为 M 定型所需的所有变量。因此， M 在其自由变量上下文 $FV(M)$ 中仍然是良类型的。

引理 3.5. 若 $\Gamma \vdash M : A$，则 $FV(M) \vdash M : A$。

3.3.2 提取函数定义。

定义 3.6。 $[[-]]_d$ 接受一个类型正确的*CC* 项，并隐式地接受其类型推导。我们定义$[[M]]_d$，$\mathfrak{D}$其中 $\mathfrak{D}$ 由如下形式的新判断给出 $\Gamma \vdash M:\ A \rightsquigarrow_d \mathfrak{D}$ （图 *4b*）。

在简单类型系统中，唯一需要做的事情 $[[-]]_d$ 就是找到每个函数 $\lambda^i x{:}\ A$。M 在源程序中，并将它们以以下形式放入标签上下文$\mathfrak{D}$ 中。

$$\mathfrak{L}(\{\overline{\mathsf{x}} : \overline{\mathsf{A}}\}, \mathsf{x}{:}\mathsf{A} \mapsto \mathsf{M} : \mathsf{B})$$

其中，$\{\overline{\mathsf{x}}{:}\overline{\mathsf{A}}\}$、$\mathsf{x}{:}\mathsf{A}$、$\mathsf{M}$和$\mathsf{B}$ 分别对应函数中的自由变量$(x{:}\ A)$、绑定变量 $x{:}\ A$、函数主体 M和返回类型 B。

唉，在我们的依赖类型论中，类型可以由函数索引；函数也可能出现在表达式的类型中，即使该表达式本身并不包含那个函数！例如，考虑 CC 中的以下三元组（内置自然数和加法）：$(\Gamma,\ M,\ N)$。

$$\begin{array}{l} \Gamma \triangleq \cdot,\ A : (Nat \to Nat) \to U_0,\ a : \Pi f : (Nat \to Nat).A\ (\lambda n : Nat.1 + (f\ n)) \\ M \triangleq a\ (\lambda x : Nat.1 + x) \\ N \triangleq A\ (\lambda n : Nat.2 + n) \end{array}$$

A是一个由 $Nat \to Nat$函数索引的类型族，而 $a\ f$构造了一个类型为$A(\lambda n\colon Nat.1 + (f\ n))$ 的元素。根据规则（TY-APPLY）， M的推断类型是

$$(A\ (\lambda n\colon Nat.1 + (f\ n)))[(\lambda x\colon Nat.1 + x)/f]$$
$$= A\ (\lambda n\colon Nat.(1 + (\lambda x\colon Nat.1 + x)\ n)),$$

归约为 $A(\lambda n\colon\ Nat.2{+}n)$。我们有 $\Gamma \vdash M\colon\ N$，然而 N包含一个不在 Γ或M中的函数！我们应将这个新函数包含在 $\mathfrak{D}$中，因为这保证了我们永远不会处于需要在 $[[M]]_d$ 中使用不存在的标签来为 $[[M]]$归约类型的情形。换句话说，该变换去函数化的不仅是源语言表达式，还包括其完整的类型推导树。

因此，我们得到了图 4b中的规则。宇宙的类型推导完全不涉及函数（D-UNIVERSE）。变量 x中的函数定义就是其类型中的定义 A (D-VAR)。依赖函数类型 $\Pi x\colon A.B$ 中的定义是定义的并集，即 A 和 B (D-PI) 中定义的并集。此处的并集与上下文的并集定义相同（见定义 3.4），并且不会产生歧义，因为不同函数对应不同的标签名。

应用 M N中的定义是 M、 N和 $B[N/x]$ 中定义的并集，因为替换 $B[N/x]$ may会创建新的函数定义（d-Apply）。对于 λ抽象$\lambda^i x$:A.M，它所包含的定义是 M 中定义与 A 中定义的并集，再加上 $\mathfrak{L}_i$（即其自身的定义）（d-Lambda）。如果 M 根据转换规则具有类型 B，那么 Γ ⊢ M :B 的推导中涉及的定义是 Γ ⊢ M : A 的推导中的定义与 B 中定义的并集（d-Eqiv）。

我们定义标签上下文的子集关系，以便陈述后续的定义和定理。

定义 *3.7.* 对于两个良构标签上下文 $\mathfrak{D}_1$ *and* $\mathfrak{D}_2$, $\mathfrak{D}_1 \subseteq \mathfrak{D}_2$，如果对于所有 $\mathfrak{L}_i(\{x : A\}, \overline{x : A} \mapsto N : B)$ *in* $\mathfrak{D}_1$， $\mathfrak{L}_i(\{\overline{x : A}\}, x\colon A \mapsto N : B)$ 也在 $\mathfrak{D}_2$ 中。

子集的概念为 DCC 带来了更强的弱化性质：良类型表达式在更大的标签上下文中仍然是良类型的。

引理 3.8（标签上下文弱化（子集））。如果 $\mathfrak{D}_1; \Gamma \vdash M : A$， $\vdash \mathfrak{D}_2$，并且 $\mathfrak{D}_1 \subseteq \mathfrak{D}_2$，则$\mathfrak{D}_2; \Gamma \vdash M : A$。

由于该变换会对表达式的整个类型推导树进行去函数化，如果$\Gamma \vdash M\colon A$，那么 $[[A]]_d$ 中的所有元素也都在 $[[M]]_d$中。我们可以对类型推导规则进行归纳，证明这一性质。

引理 3.9。对于任意良类型表达式 $\Gamma \vdash M\colon A$ *in CC,* $[[A]]_d \subseteq [[M]]_d$。

表达式变换和提取函数定义的过程（$[[-]]$ 和$[[-]]_d$）在 CC 上下文上逐点进行。换言之，

$$[[\cdot]] \triangleq \cdot, \qquad [[\Gamma, x\colon A]] \triangleq [[\Gamma]], x\colon [[A]],$$
$$[[\cdot]]_d \triangleq \cdot, \qquad [[\Gamma, x\colon A]]_d \triangleq [[\Gamma]]_d \cup [[A]]_d.$$

现在，我们可以看到，（带标签的）复合函数 $\lambda^0 A.\lambda^1 B.\lambda^2 C.\lambda^3 f.\lambda^4 g.\lambda^5 x.f\ x(g\ x)$ 转换为 $\mathfrak{L}_0\{\}$，这是一个未提供自由变量的标签，因为该函数是闭合的。复合函数的标签上下文 $\mathfrak{D}$ 可以根据函数提取判断，从下方所示的推导树草图中推导出来。

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A,B,C,f,g \vdash \lambda^5 x.f\ x\ (g\ x) \rightsquigarrow_d \mathfrak{L}_5(\{A,B,C,f,g\}, x : \_ \mapsto (f\ @\ x)\ @\ g\ @\ x : \_)}{A,B,C,f \vdash \lambda^4 g.\lambda^5 x.f\ x\ (g\ x) \rightsquigarrow_d \cdots, \mathfrak{L}_4(\{A,B,C,f\}, g : \_ \mapsto \mathfrak{L}_5\{A,B,C,f,g\} : \_)}\ \textsc{d-Lambda}}{A,B,C \vdash \lambda^3 f.\lambda^4 g.\lambda^5 x.f\ x\ (g\ x) \rightsquigarrow_d \cdots, \mathfrak{L}_3(\{A,B,C\}, f : \_ \mapsto \mathfrak{L}_4\{A,B,C,f\} : \_)}\ \textsc{d-Lambda}}{A,B \vdash \lambda^2 C.\lambda^3 f.\lambda^4 g.\lambda^5 x.f\ x\ (g\ x) \rightsquigarrow_d \cdots, \mathfrak{L}_2(\{A,B\}, C : \_ \mapsto \mathfrak{L}_3\{A,B,C\} : \_)}\ \textsc{d-Lambda}}{A \vdash \lambda^1 B.\lambda^2 C.\lambda^3 f.\lambda^4 g.\lambda^5 x.f\ x\ (g\ x) \rightsquigarrow_d \cdots, \mathfrak{L}_1(\{A\}, B : \_ \mapsto \mathfrak{L}_2\{A,B\} : \_)}\ \textsc{d-Lambda}}{\cdot \vdash \lambda^0 A.\lambda^1 B.\lambda^2 C.\lambda^3 f.\lambda^4 g.\lambda^5 x.f\ x\ (g\ x) \rightsquigarrow_d \cdots, \mathfrak{L}_0(\{\}, A : \_ \mapsto \mathfrak{L}_1\{A\} : \_)}\ \textsc{d-Lambda}$$

3.4 可靠性

$$\textit{Expressions} \quad ::= \quad \cdots \mid M\{x \mapsto N\}$$

$\boxed{\Gamma \vdash M : A}$ *(Typing)*

$$\dfrac{\Gamma \vdash M :\ \Pi x{:}A.\,B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\ N : B\{x \mapsto N\}}\ \text{s-ty-Apply} \qquad \dfrac{\Gamma, x : A \vdash M :\ B \qquad \Gamma \vdash N : A}{\Gamma \vdash M\{x \mapsto N\} : B\{x \mapsto N\}}\ \text{s-ty-Subst}$$

$\boxed{M \triangleright N}$ (归约)

$$\dfrac{}{x\{y \mapsto N\} \triangleright x}\ \textsc{s-red-Var1} \qquad \dfrac{}{x\{x \mapsto N\} \triangleright N}\ \textsc{s-red-Var2} \qquad \dfrac{}{U_i\{x \mapsto N\} \triangleright U_i}\ \textsc{s-red-Universe}$$

$$\dfrac{}{(\lambda x{:}A.M)\ N \triangleright M\{x \mapsto N\}}\ \textsc{s-red-Beta} \qquad \dfrac{}{(L\ M)\{x \mapsto N\} \triangleright (L\{x \mapsto N\})\ (M\{x \mapsto N\})}\ \textsc{s-red-Apply}$$

$$\dfrac{}{((\lambda x{:}A.M)\{\overline{x} \mapsto \overline{M}\})\ N \triangleright (M\{\overline{x} \mapsto \overline{M}\})\{x \mapsto N\}}\ \textsc{s-red-Closure}$$

$\boxed{\vdash M \equiv N}$ *(等价)*

$$\dfrac{L \triangleright^* (\lambda x{:}A.N)\{\overline{y} \mapsto \overline{N}\} \qquad M \triangleright^* M' \qquad \vdash N\{\overline{y} \mapsto \overline{N}\} \equiv M'\ x}{\vdash L \equiv M}\ \text{s-eq-Closure1} \qquad \dfrac{M \triangleright^* (\lambda x{:}A.N)\{\overline{y} \mapsto \overline{N}\} \qquad L \triangleright^* L' \qquad \vdash L'\ x \equiv N\{\overline{y} \mapsto \overline{N}\}}{\vdash L \equiv M}\ \text{s-eq-Closure2}$$

图 5. CC_S 中的新语法和规则

我们认为依赖类型去函数化是正确的，如果对于所有基本类型 A、值 v以及类型为 A的程序 M，

$$\cdot \vdash M : A \wedge M \triangleright^* v \implies \mathfrak{D}_\Gamma \cup \mathfrak{D}_M \vdash \mathsf{M} \triangleright^* \mathsf{v}' \text{ where } \mathsf{v}' \equiv \mathsf{v}.$$

换言之，如果一个封闭程序 M 求值为一个基本类型值 v，那么 M 求值为一个基本类型值 v'，且该值等价于 v。这一性质是归约序列保持性的推论，它指出

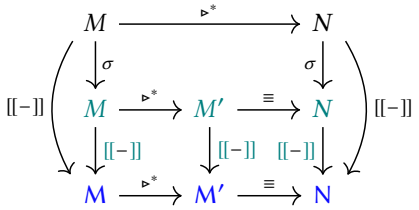
$$M \triangleright^* N \implies \mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N \vdash \mathsf{M} \triangleright^* \mathsf{M}', \quad \text{where } \mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N \vdash \mathsf{M}' \equiv \mathsf{N}.$$

理想情况下，我们可以通过证明该变换保留所有小步归约 $M \triangleright N$，然后对序列中的归约步数进行归纳，来证明这一性质。然而，CC 的元语言替换 $(\lambda x{:}A.M)[N/y]$ 会在将一个表达式替换到函数的自由变量中时创建一个新的函数定义。因此，对于 CC中的归约序列 $M_1 \triangleright \cdots \triangleright M_n$，某些 M_i 可能包含在 M_1 或 M_n 中不存在的函数定义。因此，无法

全部 M_i 转换为类型良好的 DCC 表达式 M_i 于 $(\mathfrak{D}_\Gamma \cup \mathfrak{D}_{M_1} \cup \mathfrak{D}_{M_n})$；Γ这使得标准方法不可行。此外，归约序列保持性是证明类型保持的关键引理，因为 CC 的类型规则涉及等价，而等价规则（EQ-REDUCE）是通过归约定义的。

幸运的是，在 CC 的归约序列中，元理论替换是创建新函数定义的唯一方式。如果源语言不将替换求值为函数，而是将其保留为原始表达式，就不会有问题。为运用这一观察，我们定义一个辅助语言 CC_S，它是在 CC 的基础上扩展了显式替换[Abadi 等人1991]。此外，CC_S 不会将表达式替换归约为函数。

由于 CC_S 扩展了 CC，因此每个 CC 表达式都平凡地是一个CC_S 表达式。我们将这一从 CC 到 CC 的平凡变换$_S$ 记作 σ。然后，我们以类似于从 CC 到 DCC 的方式定义从 CC_S 到 DCC 的去函数化变换——一个表达式变换 $[\![-]\!]$ 以及一个用于提取定义的元函数 $[\![-]\!]_d$。接下来，我们证明 σ 和针对 CC 的去函数化$_S$ 保持归约序列，并且它们与从 CC 到DCC的变换可交换。作为推论，从 CC 到 DCC的去函数化保持归约序列。换言之，我们证明对于所有 CC 表达式，以下图表 M 和 N（省略上下文）是可交换的。



CC_S 是 CC 的扩展，加入了新语法、类型推导规则、归约规则和等价规则（图 5）。为避免歧义，我们将 CC_S 表达式写在 *teal中 mathematical font*。CC_S 通过加入以下形式的语法替换来扩展 CC 语法：$M\{x \mapsto N\}$。

CC 中变量、宇宙、 Π-类型、函数和等价的类型规则$_S$ 与 CC 中的标准规则相同，但应用 $M\ N$ 的类型为 $B\{x \mapsto N\}$，其中使用语法替换。替换 $M\{x \mapsto N\}$ 的类型是 M 的类型，其中 x 被替换为 N (S-TY-SUBST)。

CC_S 有五条针对替换的归约规则，它们是将标准的元理论替换规则（针对变量、宇宙、 Π-类型和应用）内化到语言中的结果。请注意，CC 原有 β 归约规则中的元理论替换（$\lambda x{:}A.M$） $N \triangleright M\{x \mapsto N\}$也被语法替换取代。$CC_S$ 不会将替换归约到函数内部，但当替换应用于参数时，会对其进行 β-归约（s-red-Closure）。我们将$M\{x_1 \mapsto N_1, x_2 \mapsto N_2\}$记作先进行一次替换，再进行另一次替换（$(M\{x_1 \mapsto N_1\})\{x_2 \mapsto N_2\}$）；将$M\{\overline{y} \mapsto N\}$记作一连串替换（$(((M\{y_1 \mapsto N_1\})\{x_2 \mapsto N_2\})\cdots)\{y_n \mapsto N_n\}$。

如同在 CC 中，CC_S 中的两个项在它们 β-归约为同一表达式，或是η-等价时，彼此等价。此外，CC_S 有两个对称规则（s-eq-Closure1）和（s-eq-Closure2），用于确定将一串替换代入函数 $(\lambda x{:}A.M)\{\overline{y} \mapsto N\}$ 后何时与另一个表达式等价。本质上，这是与替换兼容的 η-等价规则的一种变体——$(\lambda x{:}A.M)\{\overline{y} \mapsto N\}$ 等价于 N，当把 N 应用于 x 后，其结果与函数主体 M（其中将 $\overline{y}$ 替换为 N）等价时。

现在，我们定义从 CC_S 到DCC的去函数化变换，该变换是将 CC 到DCC的变换扩展为以下两条规则。我们使用 $[\![-]\!]$ 和 $[\![-]\!]_d$ 表示

用于表达式变换和提取函数定义的元函数，并且我们像往常一样采用用唯一标识符标记 λ 抽象的约定 $i\,(i \in \mathbb{N})$。

$\Gamma \vdash M : A \rightsquigarrow \mathtt{M}$ （表达式变换）

$$\frac{\Gamma, x : A \vdash M : B \rightsquigarrow \mathtt{M} \quad \Gamma \vdash N : A \rightsquigarrow \mathtt{N}}{\Gamma \vdash M\{x \mapsto N\} : B\{x \mapsto N\} \rightsquigarrow \mathtt{M}[\mathtt{N}/\mathtt{x}]}\ \textsc{s-t-Subst}$$

$\Gamma \vdash M : A \rightsquigarrow_d \mathfrak{D}$ （函数提取）

$$\frac{\Gamma, x : A \vdash M : B \rightsquigarrow_d \mathfrak{D}_1 \quad \Gamma \vdash N : A \rightsquigarrow_d \mathfrak{D}_2}{\Gamma \vdash M\{x \mapsto N\} : B\{x \mapsto N\} \rightsquigarrow_d \mathfrak{D}_1 \cup \mathfrak{D}_2}\ \textsc{s-d-Subst}$$

该变换将 CC 中的语法替换$_S$ 转化为 DCC 中的元理论替换（S-T-Subst）；替换 $M\{x \mapsto N\}$ 中的函数定义，是 M 与 N (S-D-Subst) 中定义的并集。由于在 CC 中，对函数的替换不会归约$_S$，根据定义，从 CC 到 DCC 的这一变换具有以下强性质，而从 CC 到 DCC 的变换并不具备这些性质。

$$M \triangleright^* N \implies [\![N]\!]_d \subseteq [\![M]\!]_d \tag{1}$$

$$[\![M\{x \mapsto N\}]\!] = [\![M]\!][[\![N]\!]/\mathtt{x}]. \tag{2}$$

接下来，我们证明该变换保持 CC 中的小步归约$_S$ ——如果一个 CC$_S$ 程序 M 经过一步归约为 N，那么翻译后的程序 $\mathtt{M}$ 会经过一系列求值为 $\mathtt{N}$。

引理 3.10 （小步归约的保持性）。若 $\Gamma \vdash M : A$ 且 $M \triangleright N$，则 $[\![\Gamma]\!]_d \cup [\![M]\!]_d \vdash \mathtt{M} \triangleright^* \mathtt{N}$。

该变换保持归约序列；证明只需对序列中的小步数作平凡的归纳。

引理 3.11（归约序列保持性（CC$_S$））。若 $\Gamma \vdash M : A$ 且 $M \triangleright^* N$，则 $[\![\Gamma]\!]_d \cup [\![M]\!]_d \vdash \mathtt{M} \triangleright^* \mathtt{N}$。

该变换也是相容的，即它保持 CC$_S$ 中的等价关系。

引理 3.12 (相干性 (CC$_S$)). 如果 $\Gamma \vdash M : A$，$\Gamma \vdash N : A$，并且 $\vdash M \equiv N$，那么 $\mathfrak{D} \vdash \mathtt{M} \equiv \mathtt{N}$，其中 $\mathfrak{D} = [\![\Gamma]\!]_d \cup [\![M]\!]_d \cup [\![N]\!]_d$。

回忆一下，σ 表示从 CC 到 CC$_S$ 的平凡变换。根据定义，这个平凡变换与两个项变换可交换。

$$[\![\sigma(M)]\!] = [\![M]\!] \tag{3}$$

此外，$[\![\sigma(M)]\!]_d$中的函数定义是 $[\![M]\!]_d$中定义的子集，因为新的函数定义会作为替换的结果出现在 CC 的类型推导树中，但在 CC$_S$中不会发生这种情况。

$$[\![\sigma(M)]\!]_d \subseteq [\![M]\!]_d \tag{4}$$

我们证明，σ 也保持归约序列。作为约定，在没有歧义时，我们用 M 表示 $\sigma(M)$。

引理 3.13 (归约序列保持性 (σ)). 若 $\Gamma \vdash M \triangleright^* N$，则 $\Gamma \vdash M \triangleright^* M'$ 其中 $\Gamma \vdash M' \equiv N$。

最后，我们可以利用上述引理，证明依赖类型去函数化（从 CC 到 DCC）的约化序列保持性。

引理 3.14 （归约序列保持性）。对于所有 M 和 N，如果 $\Gamma \vdash M : A$ 且 $M \triangleright^* N$，则有

$$\mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N \vdash \mathsf{M} \triangleright^* \mathsf{M}', \tag{5}$$

$$\mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N \vdash \mathsf{M}' \triangleright^* \mathsf{N} \tag{6}$$

对于某个 M' 其中 $(\mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N) = ([[\Gamma]]_d, [[M]]_d, [[N]]_d)$ 且 $(\mathsf{M}, \mathsf{N}) = ([[M]], [[N]])$。

由于地类型和值不包含函数，$[[v]]_d = \cdot$，而变换的正确性只是引理3.14的一个特例。

推论 3.15. *(正确性)* 对所有地类型 A 和值 v，其类型为 A，

$$\cdot \vdash M : A \wedge M \triangleright^* v \Longrightarrow \mathfrak{D}_\Gamma \cup \mathfrak{D}_M \vdash \mathsf{M} \triangleright^* \mathsf{v}' \ \textit{where}\ \mathsf{v}' \equiv \mathsf{v}.$$

类型保持性的证明需要三个引理：替换、约化序列保持性，以及相干性。引理3.14 证明了依赖类型去函数化借助 CC 保持归约序列$_S$，现在我们以类似的方式证明其余两个引理。替换引理表明，去函数化与替换相容。

引理 3.16 (替换)。 若 $\Gamma, x{:}A \vdash M{:}B$ 且 $\Gamma \vdash N{:}A$，则 $\mathfrak{D} \vdash [[M\,[N/x]]] \equiv \mathsf{M}[\mathsf{N}/\mathsf{x}]$,其中 $\mathfrak{D} = \mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N \cup \mathfrak{D}_M[N/x]$。

相干性引理指出，去函数化与 CC 的相干性判断相容。

引理 3.17 （相干性）。若 $\Gamma \vdash M : A$，$\Gamma \vdash N : A$，且 $\vdash M \equiv N$，则 $\mathfrak{D} \vdash \mathsf{M} \equiv \mathsf{N}$，其中 $\mathfrak{D} = \mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N$。

F最后，我们通过对 CC 的类型推导规则进行归纳，证明类型保持。 .

定理 3.18 (类型保持)。 对于所有类型良构程序 M，

$$\Gamma \vdash M : A \Longrightarrow \mathfrak{D}_\Gamma \cup \mathfrak{D}_M; \Gamma \vdash \mathsf{M} : \mathsf{A},$$

其中 $(\mathfrak{D}_\Gamma, \mathfrak{D}_M) = ([[\Gamma]]_d, [[M]]_d)$ 且 $(\Gamma, \mathsf{M}, \mathsf{A}) = ([[\Gamma]], [[M]], [[A]])$.

3.5 DCC 的一致性与类型安全性

作为一种依赖类型论，DCC 作为编程语言运行时应当具有类型安全性，而被解释为逻辑时应当保持一致性。遵循Boulier 等人 [2017] 和 Bowman andAhmed[2018]，本节通过定义从 DCC 到 CC 的 逆向变换并证明其保持归约序列，来证明这些性质，从而在 DCC 中归约一个表达式等价于在 CC 中归约一个表达式。该变换保持类型，并将 DCC 中 假 的逻辑解释转换为CC中的逻辑解释，因此 DCC 中的有效证明（即类型良构程序）对应于 CC中的有效证明。这将证明 DCC 的类型安全性和一致性的问题，归约为证明 CC 的类型安全性和一致性；后者是一个标准结果 [Coquand 和 Huet 1988]。换言之，我们表明，DCC 可以以一致且保持含义的方式由 CC 建模。逆向变换的类型保持还需要 替换、 约化序列保持性以及 相干性 引理，这与定理 3.18的证明类似，其证明是直接的。

我们定义逆向变换 $[[-]]$，并引入一种形式为 $\mathfrak{D}; \Gamma \vdash \mathsf{M} : \mathsf{A} \rightsquigarrow_b \mathsf{M}$ 和 $[[\mathsf{M}]]$，M 的新判断（图 6）。该翻译将变量、宇宙、 Π-类型和应用映射回它们在 CC 中对应的形式，并将标签表达式 $\mathfrak{L}\{\mathsf{M}\}$（其中$\mathfrak{L}(\{\mathsf{x} : \mathsf{A}\}, \mathsf{x} : \mathsf{A} \mapsto \mathsf{M} : \mathsf{B}) \in \mathfrak{D}$）映射为 $\lambda \mathsf{x}{:}\mathsf{A}[\mathsf{M}/\mathsf{x}].\mathsf{M}[\mathsf{M}/\mathsf{x}]$——一个已代入其所有自由变量值的函数，其中 A、 M 和 ~~M~~ 分别表示 $[[\mathsf{A}]]$, $[[\mathsf{M}]]$，和 ~~$[[\mathsf{M}]]$~~

$\boxed{\mathfrak{D}; \Gamma \vdash M : A \rightsquigarrow_b M}$ （逆向变换）

$$\frac{}{\mathfrak{D}; \Gamma \vdash x : A \rightsquigarrow_b x}\ \textsc{b-Var} \qquad \frac{}{\mathfrak{D}; \Gamma \vdash U_i : U_{i+1} \rightsquigarrow_b U_i}\ \textsc{b-Universe}$$

$$\frac{\mathfrak{D}; \Gamma \vdash A : U_i \rightsquigarrow_b A \quad \mathfrak{D}; \Gamma, x : A \vdash B : U_j \rightsquigarrow_b B}{\mathfrak{D}; \Gamma \vdash \Pi x{:}A.B : U_{\max(i,j)} \rightsquigarrow_b \Pi x{:}A.B}\ \textsc{b-Pi}$$

$$\frac{\mathfrak{L}(\{\overline{x} : \overline{A}\}, x{:}A \mapsto M : B) \in \mathfrak{D} \quad \mathfrak{D}; \Gamma \vdash \overline{M} : \overline{A} \rightsquigarrow_b \overline{M} \quad \mathfrak{D}; \Gamma \vdash A : U \rightsquigarrow_b A \quad \mathfrak{D}; \Gamma, x : A \vdash M : B \rightsquigarrow_b M}{\mathfrak{D}; \Gamma \vdash \mathfrak{L}\{\overline{M}\} : \Pi x{:}A[\overline{M}/\overline{x}].B[\overline{M}/\overline{x}] \rightsquigarrow_b \lambda x{:}A[\overline{M}/\overline{x}].M[\overline{M}/\overline{x}]}\ \textsc{b-Label}$$

$$\frac{\mathfrak{D}; \Gamma \vdash L : \Pi x{:}A.B \rightsquigarrow_b L \quad \mathfrak{D}; \Gamma \vdash M : A \rightsquigarrow_b M}{\mathfrak{D}; \Gamma \vdash L\ @\ M : B[M/x] \rightsquigarrow_b L\ M}\ \textsc{b-Apply} \qquad \frac{\mathfrak{D}; \Gamma \vdash M : A \rightsquigarrow_b M \quad \mathfrak{D}; \Gamma \vdash B : U_i \quad \mathfrak{D} \vdash A \equiv B}{\mathfrak{D}; \Gamma \vdash M : B \rightsquigarrow_b M}\ \textsc{b-Equiv}$$

图 6. 逆向变换

(b-标签)。直观地说， $[[-]]$ 将标签反编译回它所表示的函数。逆向变换也逐点作用于类型上下文。

在 CC 中，逻辑假的解释是 $\Pi x{:}U_0.x$。不存在具有假类型的闭合表达式。在 DCC 中，假的解释是 $\Pi x{:}U_0.x$，因此根据定义，逆向变换保持了假的性质。

接下来，我们证明逆向变换与替换相容。本节中约定：若无歧义，我们记 M 为 $[[M]]$。

引理 3.19 （逆向变换与替换相容）。若 $\mathfrak{D}; \Gamma, x : A \vdash M : B$ 且 $\mathfrak{D}; \Gamma \vdash N : A$，则 $[[M[N/x]]] = M[N/x]$。

与 §3.4 节中的证明类似，我们通过证明变换保持小步归约，来证明归约序列保持性。利用这一点，我们证明逆向变换的相干性引理，然后证明类型保持。

Lemma 3.20. *If* $\mathfrak{D}; \Gamma \vdash M : A$ *and* $\mathfrak{D} \vdash M \triangleright^* N$, *then* $M \triangleright^* N$.

引理 3.21。若 $\mathfrak{D}; \Gamma \vdash M : A$，$\mathfrak{D}; \Gamma \vdash N : A$，且 $\mathfrak{D} \vdash M \equiv N$，则 $\vdash M \equiv N$。

引理 3.22。若 $\mathfrak{D}; \Gamma \vdash M : A$，则 $\Gamma \vdash M : A$。

作为引理3.20 和引理3.22的推论，DCC 是类型安全且一致的，因为 CC 也是如此。

Theorem 3.23 (Type safety). *If* $\mathfrak{D}; \cdot \vdash M : A$, *then* $\mathfrak{D} \vdash M \triangleright^* v$ *for some irreducible value* v.

也就是说，类型安全保证每个类型正确的封闭 DCC 项都能在有限步内归约为一个值。

定理3.24 （一致性）。不存在这样的标签上下文 $\mathfrak{D}$ 和 *DCC* 项 M，使得$\mathfrak{D}; \cdot \vdash M : \Pi A{:}U.A$。

将 DCC 解释为逻辑时，项 ΠA:U.A （它会产生任意类型 A 的项）对应于 假；它经反向变换得到 Π*A*:*U*.*A*，这是 假 在 CC 中的表示。DCC 的一致性意味着不存在类型为 ΠA:U.A 的闭项；若存在这样的项，翻译就会在 CC 中产生相应的项，而 CC 也会因此不一致。

4 实现

我们提供了 §3 中去函数化翻译的可移植独立实现。该实现以 OCaml 编写，并使用 `js_of_ocaml` [Vouillon 和 Balat2014] 编译为可在网页浏览器中运行的程序。该实现可对 CC（§3.1）和 DCC（§3.2）项进行类型检查，执行抽象去函数化（§3.3），并将 DCC 反向翻译为 CC（§3.5），让感兴趣的读者可以通过真实示例体验该翻译的效果。我们还提供了若干现成示例，包括依赖组合、依赖对和有限集。

5 相关工作

保类型编译。 保类型编译最初是为优化编译和验证编译后的代码而发展起来的；它已广泛应用于简单类型语言和多态语言的编译器中，偶尔也用于依赖类型语言。例如，Tarditi et al. [1996] 提出了 TIL（带类型的中间语言），这是一种 ML 编译器，具有面向类型的代码优化功能，可优化循环、垃圾回收和多态函数调用；而Morrisett etal. [1999] 研究了从 System F 到带类型汇编语言 TAL 的保类型翻译。Xi andHarper [2001] 随后将 TAL 扩展为 DTAL，这是一种具有有限形式依赖类型的汇编语言，可作为 Dependent ML 的编译目标。

Guillemette 和Monnier [2008] 还提出了一种源自System F的保持类型的编译器，但采用了不同的方法：使用广义代数数据类型构建带类型的中间表示，并利用宿主语言（GHC Haskell）的类型系统来验证编译器的每个阶段都保持类型。以这种方式嵌入带类型的变换是函数式编程社区中一种流行的技术，Carette等人的工作便是一个例子，Carette 等人 [2009], 该工作提出了嵌入式语言的保持类型的CPS 变换，以及基于部分求值的保持类型的优化。

Necula [1997]的携带证明代码是另一种早期的可靠可执行文件生成方法。它依赖外部逻辑框架来检查随代码附带的证明的正确性。

Bowman 及其合作者为依赖类型语言开发了多种保类型的翻译，包括 CPS 变换 [Bowman et al. 2018], 闭包转换 [Bowman 和 Ahme 2018] （基于 Minamide 等人 [1996]针对 System F 的类型化闭包转换，并转换为 A 范式 [Koronkevich 等人 2022]。

去函数化。 去函数化最初由 Reynolds [1972] 作为一种将高阶解释器转换为一阶解释器的编程技术提出 [Reynolds 1972]。它已用于多种应用，从 ML 编译器 [Cejtin 等人 2000; Chin 和 Darlington1996], 到类型安全垃圾回收器 [Wang 和 Appel 2001], 以及高阶类型多态的编码 [Yallop 和 White 2014]。

去函数化最初被提出为一种无类型转换。使用一系列单态 应用 函数来使简单类型去函数化保留类型，是文献中的一种标准变通方法 [Bell 等人 1997; Cejtin 等人 2000; Nielsen 2000; Tolmach 和 Oliva 1998]。

Danvy 和 Nielsen [2001] 调查了更多去函数化在实践中的例子。

迄今为止，对去函数化的形式化研究主要致力于证明变换的类型保持性和正确性。Bell 等人 [1997] 已证明，针对简单

类型化程序的翻译具有类型保持性。Nielsen [2000] 借助指称语义证明了其部分正确性，而 Banerjee 等人 [2001] 则使用操作语义确立了其完全正确性。Pottier 和 Gauthier [2004] 在扩展了 GADT 的 System F 中，形式化了具有类型保持性的多态去函数化。

闭包转换。与去函数化类似，闭包转换 变换也会将闭包表示为一阶值，该值将某种代码标识符与一组自由变量配对。上述文献中对闭包转换的表述 [Bowman and Ahmed 2018年; Minamide 等人 1996] 与去函数化的差异显著：去函数化涉及由代码标识符索引的全局定义映射（例如 *apply* 函数或我们的标签环境），而这些闭包转换则是在局部将函数转换为代码与环境对，并可通过标准消去规则应用这些对。不过，闭包转换的其他表述 [例如Appel 1992; Siek 2012] 还会将函数提升到顶层，使这种变换与去函数化更为相似。

闭包转换在许多函数式语言的编译器中发挥着关键作用，包括 Scheme [Steele Jr 1978],CAML [Mauny and Suárez 1986], Standard ML [Cejtin 等人 2000], Haskell [Leshchins 2006] 等人。近期工作着重于建立复杂的语义性质，例如在存在可变状态和控制效应的情况下，闭包转换的正确性（即使与外语代码链接时也是如此） [Mates 等人 2019], 以及时间和空间性质的保持 [Paraskevopoulou 和 Appel 2019]。

再函数化。我们的逆向翻译（参见 §3.5）与再函数化有关 [Danvy和 Millikin2009], 去函数化的左逆。在再函数化中，我们将目标应用M@ N 替换为源应用 M N，并将标签 ϑ{M} 替换为抽象 (λ x:A.M) [M/x]，其依据是它们在标签上下文中的实现 ϑ({x : A} , x: A ↦ M : B)。

致谢
感谢 David Sheets、András Kovács 和 Marcelo Fiore 提供有益的意见。

参考文献

Martín Abadi、Luca Cardelli、Pierre-Louis Curien 和 Jean-Jacques Lévy。1991 年 1，4 1991 年 375 416101017095679680000186 。显式替换。《函数式编程杂志》( )，–。https://doi.org/ 。 /SBenedikt Ahrens、Peter LeFanu Lumsdaine 和 Vladimir Voevodsky。2018年。单价基础中类型论的范畴结构。逻辑方法与计算机科学14, 3 (2018年)。https://doi.org/10.23638/LMCS-14(3:18)2018 Andrew W. Appel。1992年。使用续延进行编译。剑桥大学出版社。Anindya Banerjee、Nevin Heintze 和 Jon G. Riecke。2001年。基于控制流分析的程序变换的设计与正确性。载于计算机软件的理论方面：第 *4* 届国际研讨会 *TACS 2001*，仙台，日本，*2001* 年 *10* 月 *29—31* 日，论文集（计算机科学讲义，第 *2215* 卷），Naoki Kobayashi 和 Benjamin C. Pierce（编）。施普林格，420–447。https://doi.org/10.1007/3-540-45500-0_21 Gilles Barthe、John Hatcliff 和 Morten Heine Sørensen。1999年。CPS 转换与应用：立方体及其扩展。高阶符号计算12, 2 (1999年)，125–170。https://doi.org/10.1023/A:1010000206149 Gilles Barthe 和 Tarmo Uustalu。2002年。归纳类型和余归纳类型的 CPS 转换。载于*2002* 年 *ACM SIGPLAN* 部分求值与基于语义的程序操纵研讨会论文集（*PEPM '02*），美国俄勒冈州波特兰，*2002* 年 *1* 月 *14—15* 日，Peter Thiemann（编）。ACM，131–142。https://doi.org/10.1145/503032.503043 Jeffrey M. Bell、Françoise Bellegarde 和 James Hook。1997 年。类型驱动的去函数化。载于*1997* 年 *ACM SIGPLAN* 函数式编程国际会议论文集（*ICFP '97*），荷兰阿姆斯特丹，*1997* 年 *6* 月 *9—11* 日，Simon L. Peyton Jones、Mads Tofte 和 A. Michael Berman（编）。ACM，25–37。https://doi.org/10.1145/258948.258953 Simon Boulier、Pierre-Marie Pédrot 和 Nicolas Tabareau。2017年。类型论的下一个 700 个语法模型。载于第 *6* 届 *17*，*2017*年论文集，*2017*年，*16*18219410114530186103018620*ACM SIGPLAN* 认证程序与证明大会，*CPP*，法国巴黎，*1* 月—Yves Bertot 和 Viktor Vafeiadis（编）。ACM，–。https://doi.org/ 。 / 。

Ana Bove、Peter Dybjer 和 Ulf Norell。2009 年。Agda 简介——一种依赖类型函数式语言。载于《高阶逻辑定理证明》，22 20，2009 2009 年，第 17 5674 届国际会议，TPHOLs，德国慕尼黑，8 月 ——。会议录（《计算机科学讲义》，第 卷），Stefan Berghofer、Tobias Nipkow、Christian Urban 和 Makarius Wenzel（编）。施普林格，73–78。https://doi.org/10.1007/978-3-642-03359-9_6 William J. Bowman 和 Amal Ahmed。2018 年。构造演算的类型化闭包转换。载于第 39 届 ACM SIGPLAN 编程语言设计与实现会议论文集，PLDI 2018，美国宾夕法尼亚州费城，2018 年 6 月 18–22 日，Jeffrey S. Foster 和 Dan Grossman（编）。ACM，797–811。https://doi.org/10.1145/3192366.3192372 William J. Bowman、Youyou Cong、Nick Rioux 和 Amal Ahmed。2018 年。Σ和 Π类型的保持类型 CPS 转换并非不可能。ACM 程序设计语言会议录 2，POPL（2018），22:1–22:33。https://doi.org/10.1145/3158110 Edwin C. Brady。2013 年。Idris，一种通用的依赖类型编程语言：设计与实现。《函数式编程杂志》23，5（2013），552–593。https://doi.org/10.1017/S095679681300018X Bernd Braßel。2011 年。通过转换为纯函数式程序实现函数逻辑程序。博士论文。https://macau.uni-kiel.de/receive/diss_mods_00007056 Jacques Carette、Oleg Kiselyov 和 Chung-chieh Shan。2009 年。最终无标签、部分求值：用于更简单带类型语言的无标签分阶段解释器。《函数式编程杂志》19，5（2009），509–543。https://doi.org/10.1017/S0956796809007205 Henry Cejtin、Suresh Jagannathan 和 Stephen Weeks。2000 年。面向流的带类型语言闭包转换。载于《编程语言与系统》，第 9 届欧洲编程研讨会，ESOP 2000，作为欧洲软件理论与实践联合会议 ETAPS 2000 的一部分举行，德国柏林，2000 年 3 月 25 日–4 月 2 日，会议录（《计算机科学讲义》，第 1782 卷），Gert Smolka（编）。施普林格，56–71。https://doi.org/10.1007/3-540-46425-5_4 Wei-Ngan Chin 和 John Darlington。1996 年。一种高阶消除方法。LISP 符号计算 9，4（1996），287–322。Coq 开发团队。2022 年。Coq 参考手册。https://coq.inria.fr/distrib/current/refman/。Thierry Coquand 和 Gérard P. Huet。1988 年。构造演算。《信息与计算》76，2/3（1988），95–120。https://doi.org/10.1016/0890-5401(88)90005-3 Olivier Danvy 和 Kevin Millikin。2009 年。再函数化的实践。《科学计算程序》74，8（2009），534–549。https://doi.org/10.1016/j.scico.2007.10.007 Olivier Danvy 和 Lasse R. Nielsen。2001 年。去函数化实践。载于第 3 届 ACM SIGPLAN 声明式编程原理与实践国际会议论文集，2001 年 9 月 5–7 日，意大利佛罗伦萨。ACM，162–174。https://doi.org/10.1145/773184.773202 Leonardo Mendonça de Moura、Soonho Kong、Jeremy Avigad、Floris van Doorn 和 Jakob von Raumer。2015 年。Lean 定理证明器（系统描述）。载于自动演绎——CADE-25——第 25 届自动演绎国际会议，德国柏林，2015 年 8 月 1–7 日，会议录（《计算机科学讲义》，第 9195 卷），Amy P. Felty 和 Aart Middeldorp（编）。施普林格，378–388。https://doi.org/10.1007/978-3-319-21401-6_26 Peter Dybjer。1994 年。归纳族。《计算机形式化方面》6，4（1994），440–465。https://doi.org/10.1007/BF01211308 Louis-Julien Guillemette 和 Stefan Monnier。2008 年。Haskell 中的保持类型编译器。载于第 13 届 ACM SIGPLAN 函数式编程国际会议论文集，ICFP 2008，加拿大不列颠哥伦比亚省维多利亚，2008 年 9 月 20–28 日，James Hook 和 Peter Thiemann（编）。ACM，75–86。https://doi.org/10.1145/1411204.1411218 Paulette Koronkevich、Ramon Rakow、Amal Ahmed 和 William J. Bowman。2022 年。A 范式在外延相等意义下保持依赖类型。《函数式编程杂志》32（2022），e12。https://doi.org/10.1017/S0956796822000090 András Kovács。2020 年。带有一等隐式函数类型的 elaboration。ACM 程序设计语言会议录 4，ICFP（2020），101:1– 101:29。https://doi.org/10.1145/3408983 Roman Leshchinskiy、Manuel M. T. Chakravarty 和 Gabriele Keller。2006 年。高阶扁平化。载于《计算科学》——ICCS 2006，第 6 届国际会议，英国雷丁，2006 年 5 月 28–31 日，会议录，第二部分（《计算机科学讲义》，第 3992 卷），Vassil N. Alexandrov、G. Dick van Albada、Peter M. A. Sloot 和 Jack J. Dongarra（编）。施普林格，920–928。https://doi.org/10.1007/11758525_122 Zhaohui Luo。1990 年。扩展构造演算。博士论文。英国爱丁堡大学。https://hdl.handle.net/1842/12487 Phillip Mates、Jamie Perconti 和 Amal Ahmed。2019 年。受控之下：具有可变状态的组合正确闭包转换。载于第 21 届国际程序设计语言原理与实践研讨会论文集，PPDP 2019，葡萄牙波尔图，2019 年 10 月 7–9 日，Ekaterina Komendantskaya（编）。ACM，16:1–16:15。https://doi.org/10.1145/3354166.3354181 Michel Mauny 和 Ascánder Suárez。1986 年。在范畴抽象机中实现函数式语言。载于 1986 年 ACM LISP 与函数式编程会议论文集，LFP 1986，美国马萨诸塞州剑桥，1986 年 8 月 4–6 日，William L. Scherlis、John H. Williams 和 Richard P. Gabriel（编）。ACM，266–278。https://doi.org/10.1145/319838.319869 Yasuhiko Minamide、J. Gregory Morrisett 和 Robert Harper。1996 年。类型化闭包转换。载于 POPL'96 会议记录：第 23 届 ACM SIGPLAN-SIGACT 编程语言原理研讨会，在该会议上提交的论文，娱乐主管

研讨会，美国佛罗里达州圣彼得斯堡海滩，1996年1月21日至24日，Hans-Juergen Boehm 和 Guy L. Steele Jr.（编）。ACM出版社，271–283。https://doi.org/10.1145/237721.237791 J. Gregory Morrisett、David Walker、Karl Crary 和 Neal Glew。1999。从 System F 到类型化汇编语言。《ACM 程序语言系统汇刊》21卷3期（1999），527–568。https://doi.org/10.1145/319301.319345 George C. Necula。1997。携带证明的代码。载于《POPL'97会议记录：第24届 ACM SIGPLAN-SIGACT 编程语言原理研讨会，研讨会论文》，法国巴黎，1997年1月15日至17日，Peter Lee、Fritz Henglein 和 Neil D. Jones（编）。ACM出版社，106–119。https://doi.org/10.1145/263699.263712 Lasse R Nielsen。2000。去函数化的指称性探究。BRICS报告系列7，47（2000）。Ulf Norell。2008。Agda中的依值类型编程。载于《高级函数式编程》第6届国际暑期学校，AFP 2008，荷兰海延，2008年5月，修订版讲义（计算机科学讲义，第5832卷），Pieter W. M. Koopman、Rinus Plasmeijer 和 S. Doaitse Swierstra（编）。施普林格，230–266。https://doi.org/10.1007/978-3-642-04652-0_5 Zoe Paraskevopoulou 和 Andrew W. Appel。2019。闭包转换在空间使用方面是安全的。《ACM 程序设计语言会议录》3，ICFP（2019），83:1–83:29。https://doi.org/10.1145/3341687 Marco Patrignani、Amal Ahmed 和 Dave Clarke。2019。安全编译的形式化方法：完全抽象编译及相关工作的综述。《ACM计算评论》51卷6期（2019），125:1–125:36。https://doi.org/10.1145/3280984 Greg Pettyjohn、John Clements、Joe Marshall、Shriram Krishnamurthi 和 Matthias Felleisen。2005。来自广义栈检查的延续。载于《第10届ACM SIGPLAN函数式编程国际会议论文集，ICFP 2005》，爱沙尼亚塔林，2005年9月26日至28日，Olivier Danvy 和 Benjamin C. Pierce（编）。ACM，216–227。https://doi.org/10.1145/1086365.1086393 Peter Podlovics、Csaba Hruska 和 Andor Pénzes。2021。重新审视GRIN：一种优化函数式语言的后端。《网络学报》（2021年2月）。https://doi.org/10.14232/actacyb.282969 François Pottier 和 Nadji Gauthier。2004。多态类型化去函数化。载于《第31届ACM SIGPLAN-SIGACT编程语言原理研讨会论文集，POPL 2004》，意大利威尼斯，2004年1月14日至16日，Neil D. Jones 和 Xavier Leroy（编）。ACM，89–98。https://doi.org/10.1145/964001.964009 François Pottier 和 Nadji Gauthier。2006。多态类型化去函数化与具体化。《高阶符号计算》19卷1期（2006），125–162。https://doi.org/10.1007/s10990-006-8611-7 John C. Reynolds。1972。高阶编程语言的定义解释器。载于《ACM年度会议论文集，ACM 1972》，1972年，第2卷，John J. Donovan 和 Rosemary Shields（编）。ACM，717–740。https://doi.org/10.1145/800194.805852 Jeremy Siek。2012。闭包转换的本质。http://siek.blogspot.com/2012/07/essence-of-closure-conversion.html

.

Guy Lewis Steele Jr. 1978。RABBIT：SCHEME 编译器。硕士论文。麻省理工学院。Martin Sulzmann、Manuel M. T. Chakravarty、Simon L. Peyton Jones 和 Kevin Donnelly。2007。带类型相等强制转换的 System F。载于《TLDI'07 论文集：2007年 ACM SIGPLAN 语言类型与实现国际研讨会》，法国尼斯，2007年1月16日，François Pottier 和 George C. Necula（编）。ACM，53–66。https://doi.org/10.1145/1190315.1190324 David Tarditi、J. Gregory Morrisett、Perry Cheng、Christopher A. Stone、Robert Harper 和 Peter Lee。1996。TIL：一种面向类型的 ML 优化编译器。载于《ACM SIGPLAN'96 编程语言设计与实现大会（PLDI）论文集》，美国宾夕法尼亚州费城，1996年5月21日至24日，Charles N. Fischer（编）。ACM，181–192。https://doi.org/10.1145/231379.231414 Amin Timany 和 Matthieu Sozeau。2017。累积归纳构造的预示演算（pCuIC）的一致性。CoRR abs/1710.03912（2017）。arXiv:1710.03912 http://arxiv.org/abs/1710.03912 Andrew P. Tolmach 和 Dino Oliva。1998。从 ML 到 Ada：通过源代码转换实现强类型语言互操作性。《函数式编程杂志》8，4（1998），367–412。http://journals.cambridge.org/action/displayAbstract?aid=44181 Jérôme Vouillon 和 Vincent Balat。2014。从字节码到 JavaScript：Js_of_ocaml 编译器。《软件：实践与经验》44，8（2014），951–972。https://doi.org/10.1002/spe.2187 Philip Wadler 和 Stephen Blott。1989。如何让特设多态更不特设。载于《第十六届 ACM 编程语言原理年会论文集》，美国得克萨斯州奥斯汀，1989年1月11日至13日。ACM出版社，60–76。https://doi.org/10.1145/75277.75283 Daniel C. Wang 和 Andrew W. Appel。2001。类型保持垃圾回收器。载于《POPL 2001 论文集：第28届 ACM SIGPLAN-SIGACT 编程语言原理研讨会》，英国伦敦，2001年1月17日至19日，Chris Hankin 和 Dave Schmidt（编）。ACM，166–178。https://doi.org/10.1145/360204.360218 Stephen Weeks。2006。MLton 中的全程序编译。载于《2006年 ML 研讨会论文集》（美国俄勒冈州波特兰）（ML '06）。计算机协会，纽约，美国，1。https://doi.org/10.1145/1159876.1159877 Stephanie Weirich 和 Chris Casinghino。2010。依赖类型泛型编程。载于《泛型与索引编程——国际春季学校，SSGIP 2010》，英国牛津，2010年3月22日至26日，修订版讲义（讲义丛书中的...

计算机科学，第 *7470* 卷），Jeremy Gibbons（编）。施普林格，217–258。 https://doi.org/10.1007/978-3-642-32202-0_5
Hongwei Xi 和 Robert Harper。2001。依赖类型汇编语言。载于 第六届 *ACM SIGPLAN* 函数式编程国际会议（*ICFP '01*）论文集，意大利佛罗伦萨，*2001* 年 *9* 月 *3–5* 日，Benjamin C. Pierce（编）。ACM，169–180。 https://doi.org/10.1145/507635.507657

Jeremy Yallop 和 Leo White。2014 年。《轻量级高阶类型多态》。载于 函数式与逻辑编程——第 *12* 届国际研讨会 *FLOPS 2014*，*2014* 年 *6* 月 *4* 日至 *6* 日，日本金泽。论文集（计算机科学讲义，第 *8475* 卷），Michael Codish 和 Eijiro Sumii（编）。施普林格，119–135 页。 https://doi.org/10.1007/978-3-319-07151-0_8

Agda 中的依赖类型去函数化

```
{-# 选项 -type-in-type#-}

data Π : (A : 集合) → (A →集合) → 集合 infixl 9 _$
__$_ : ∀ {A : 集合} {p : A → 集合} → Π A p → (x :
A) → p x

{-# NO_POSITIVITY_CHECK #-}
data Π wher
    eF1 : Π Set (λ A →Π (Π A(λ _ →Set)) (λ B →Π
  (Π A (λ x → Π (B$ x) (λ _ → Set)))(λ C →Π (Π A (λ
  y → Π (B$ y)(λ z → C $y $ z)))(λ f →Π (Π A (λ x →
  B$ x)) (λ g →Π A (λ x →C $x $ (g$ x))))))) F2 :(A :
  Set) →F3: (A : Set) →(B : Π A (λ _ →Set)) →Π (Π
  A (λ x → Π (B$ x) (λ _ → Set)))(λ C →Π (Π A (λ y
  → Π (B$ y)(λ z → C $y $ z)))(λ f →Π (Π A (λ x →
  B$ x)) (λ g →Π A (λ x →C$ x $ (g $ x))))) F4: (A :
  Set) →(B : Π A (λ _ →Set)) →(C : Π A (λ x →Π (
  B $x) (λ _ → Set)))) →Π (Π A (λ y → Π (B$ y)(λ z →
  C $y $ z)))(λ f →Π (Π A (λ x →B$ x)) (λ g →Π A (λ
  x →C $x $ (g $ x)))) F5 : (A : Set) →
```

```
Π (Π A (λ _ → Set)) (λ B →
Π (B : Π A (λ _ → Set)) (λ C : Π A (λ x →Π (
Π (Π A (λ x → Π (B $ x) (λ z → C $ y $ z))) (λ f →
Π (Π A (λ x → B $ x)) (λ g →
Π A (λ x →
C $ x $ (g $ x))))))
```

```
      (B : Π A (λ _ →Set)) →(C : (Π A (λ x → Π
  (B $ x) (λ _ → Set)))) →(f : (Π A (λ y → Π (B $
  y) (λ z →C $ y $ z)))) →Π (Π A (λ x →B $ x)) (λ
  g →Π A (λ x →C $ x $ (g $ x))) F6 : (A : Set) →
  (B : Π A (λ _ →Set)) →(C : (Π A (λ x → Π (B $
  x) (λ _ → Set)))) →(f : (Π A (λ y → Π (B $ y) (λ z
  →C $ y $ z)))) →(g : (Π A (λ x → B $ x))) →Π
  A (λ x →C $ x $ (g $ x))
```

{-# 终止性#-}

```
F1 $ A =F2 A F2A$ B = F3 A
B F3 A B$C = F4 A BC F4 A B
C $ f =F5 A B C f F5 A B C f $
g = F6A B C f g F6 A B C f g $
x = f $ x $ (g $ x)
```

B 全依赖组合的去函数化

全依赖组合

$$\begin{aligned}\text{compose} \quad ::= \quad & \lambda A{:}U_0.\ \lambda B{:}(\Pi x{:}A.U_0).\ \lambda C{:}(\Pi x{:}A.\Pi y{:}B\ x.U_0).\\ & \lambda f{:}(\Pi y{:}A.(\Pi z{:}B\ y.C\ y\ z)).\ \lambda g{:}(\Pi x{:}A.B\ x).\\ & \lambda x{:}A.\ f\ x\ (g\ x)\end{aligned}$$

去函数化的非依赖组合。

$$\begin{aligned}\mathfrak{D} \quad ::= \quad & \mathfrak{L}_2(\{B : U_0, C : U_0, A : U_0, f : (\Pi\_ : B.C), g : (\Pi\_ : A.B)\}, x : A \mapsto f@(g@x) : C),\\ & \mathfrak{L}_1(\{B : U_0, C : U_0, A : U_0, f : (\Pi\_ : B.C)\},\\ & \quad g : (\Pi\_ : A.B) \mapsto \mathfrak{L}_2\{B, C, A, f, g\} : \Pi x : A.C),\\ & \mathfrak{L}_0(\{B : U_0, C : U_0, A : U_0\},\\ & \quad f : (\Pi\_ : B.C) \mapsto \mathfrak{L}_1\{B, C, A, f\} : \Pi g : (\Pi\_ : A.B).\Pi x : A.C)\\ \Gamma \quad ::= \quad & A : U_0, B : U_0, C : U_0\\ \text{compose} \quad ::= \quad & \mathfrak{L}_0\{\}\\ \text{composeT} \quad ::= \quad & \Pi f : (\Pi\_ : B.C).\Pi g : (\Pi\_ : A.B).\Pi x : A.C\end{aligned}$$

完全依赖型组合的去函数化。

$$
\begin{array}{rcl}
\mathfrak{D} & ::= & \mathfrak{L}_5(\{A : U_0, B : (\Pi\_ : A.U_0), C : (\Pi x : A.\Pi\_ : (B@x).U_0),\\
& & \quad f : (\Pi x : A.\Pi y : (B@x).(C@x)@y), g : (\Pi x : A.B@x)\},\\
& & \quad x : A \mapsto (f@x)@(g@x) : (C@x)@(g@x)),\\
& & \mathfrak{L}_4(\{A : U_0, B : (\Pi\_ : A.U_0), C : (\Pi x : A.\Pi\_ : (B@x).U_0),\\
& & \quad f : (\Pi x : A.\Pi y : (B@x).(C@x)@y)\},\\
& & \quad g : (\Pi x : A.B@x) \mapsto \mathfrak{L}_5\{A, B, C, f, g\} : \Pi x : A.(C@x)@(g@x)),\\
& & \mathfrak{L}_3(\{A : U_0, B : (\Pi\_ : A.U_0), C : (\Pi x : A.\Pi\_ : (B@x).U_0)\},\\
& & \quad f : (\Pi x : A.\Pi y : (B@x).(C@x)@y) \mapsto \mathfrak{L}_4\{A, B, C, f\} :\\
& & \quad \Pi g : (\Pi x : A.B@x).\Pi x : A.(C@x)@(g@x)),\\
& & \mathfrak{L}_2(\{A : U_0, B : (\Pi\_ : A.U_0)\}, C : (\Pi x : A.\Pi\_ : (B@x).U_0) \mapsto \mathfrak{L}_3\{A, B, C\} :\\
& & \quad \Pi f : (\Pi x : A.\Pi y : (B@x).(C@x)@y).\Pi g : (\Pi x : A.B@x).\Pi x : A.(C@x)@(g@x)),\\
& & \mathfrak{L}_1(\{A : U_0\}, B : (\Pi\_ : A.U_0) \mapsto \mathfrak{L}_2\{A, B\} : \quad \Pi C : (\Pi x : A.\Pi\_ : (B@x).U_0).\\
& & \quad \Pi f : (\Pi x : A.\Pi y : (B@x).(C@x)@y).\Pi g : (\Pi x : A.B@x).\Pi x : A.(C@x)@(g@x)),\\
& & \mathfrak{L}_0(\{\}, A : U_0 \mapsto \mathfrak{L}_1\{A\} : \Pi B : (\Pi\_ : A.U_0).\Pi C : (\Pi x : A.\Pi\_ : (B@x).U_0).\\
& & \quad \Pi f : (\Pi x : A.\Pi y : (B@x).(C@x)@y).\Pi g : (\Pi x : A.B@x).\Pi x : A.(C@x)@(g@x))\\
\text{compose} & ::= & \mathfrak{L}_0\{\}\\
\text{composeT} & ::= & \Pi A{:}U_0.\ \Pi B{:}(\Pi x{:}A.U_0).\ \Pi C{:}(\Pi x{:}A.\Pi y{:}B\ @\ x.U_0).\\
& & \Pi f{:}(\Pi x{:}A.\Pi y{:}B\ @\ x.(C\ @\ x\ @\ y)).\\
& & \Pi g{:}(\Pi x{:}A.B\ @\ x).\\
& & \Pi x{:}A.(C\ @\ x)\ @\ g\ @\ x
\end{array}
$$

去函数化变换的推导草图。

$$
\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{A, B, C, f, g \vdash \lambda^5 x.f\ x\ (g\ x) \rightsquigarrow_d \mathfrak{L}_5(\{A, B, C, f, g\}, x : \_ \mapsto (f\ @\ x)\ @\ g\ @\ x : \_)}{A, B, C, f \vdash \lambda^4 g.\lambda^5 x.f\ x\ (g\ x) \rightsquigarrow_d \cdots, \mathfrak{L}_4(\{A, B, C, f\}, g : \_ \mapsto \mathfrak{L}_5\{A, B, C, f, g\} : \_)}\ \textsc{d-Lambda}}{A, B, C \vdash \lambda^3 f.\lambda^4 g.\lambda^5 x.f\ x\ (g\ x) \rightsquigarrow_d \cdots, \mathfrak{L}_3(\{A, B, C\}, f : \_ \mapsto \mathfrak{L}_4\{A, B, C, f\} : \_)}\ \textsc{d-Lambda}}{A, B \vdash \lambda^2 C.\lambda^3 f.\lambda^4 g.\lambda^5 x.f\ x\ (g\ x) \rightsquigarrow_d \cdots, \mathfrak{L}_2(\{A, B\}, C : \_ \mapsto \mathfrak{L}_3\{A, B, C\} : \_)}\ \textsc{d-Lambda}}{A \vdash \lambda^1 B.\lambda^2 C.\lambda^3 f.\lambda^4 g.\lambda^5 x.f\ x\ (g\ x) \rightsquigarrow_d \cdots, \mathfrak{L}_1(\{A\}, B : \_ \mapsto \mathfrak{L}_2\{A, B\} : \_)}\ \textsc{d-Lambda}}{\cdot \vdash \lambda^0 A.\lambda^1 B.\lambda^2 C.\lambda^3 f.\lambda^4 g.\lambda^5 x.f\ x\ (g\ x) \rightsquigarrow_d \cdots, \mathfrak{L}_0(\{\}, A : \_ \mapsto \mathfrak{L}_1\{A\} : \_)}\ \textsc{d-Lambda}
$$

收稿于2022年11月10日；录用日期：2023年3月31日

第3 节的 C 证明（去函数化与依赖类型）

引理 3.9。对于任意良类型表达式 $\Gamma \vdash M : A$，在 CC 中，$[\![A]\!]_d \subseteq [\![M]\!]_d$。

证明。根据图中定义的规则进行归纳。4b。

情形（D-Lambda）。目标是证明 $[\![\Pi x{:}A.B]\!]_d \subseteq [\![\lambda^i x{:}A.M]\!]_d$。根据假设，

$$\Gamma \vdash \lambda^i x{:}A.M : \Pi x{:}A.B \rightsquigarrow_d \mathfrak{D}_A \cup \mathfrak{D}_M, \mathfrak{L}_i(\{\overline{x} : \overline{A}\}, x{:}A \mapsto M : B).$$

换言之，$[\![\lambda^i x{:}\mathrm{A.M}]\!]_d = [\![\mathrm{A}]\!]_d \cup [\![\mathrm{M}]\!]_d$，并附加了函数 λ^i 的定义。根据定义，$[\![\Pi \mathrm{x{:}A.B}]\!]_d = [\![\mathrm{A}]\!]_d \cup [\![\mathrm{B}]\!]_d$。根据归纳假设（因为 $\Gamma, \mathrm{x : A} \vdash \mathrm{M : B}$，$[\![\mathrm{B}]\!]_d \subseteq [\![\mathrm{M}]\!]_d$），我们有 $[\![\mathrm{A}]\!]_d \subseteq [\![\lambda^i x{:}\mathrm{A.M}]\!]_d$ 和 $[\![\mathrm{B}]\!]_d \subseteq [\![\lambda^i x{:}\mathrm{A.M}]\!]_d$。

其他情形要么是平凡的，要么可由定义直接推出。

□

引理 3.10（小步归约的保留性）。若 $\Gamma \vdash M : A$ 且 $M \triangleright N$，则 $[\![\Gamma]\!]_d \cup [\![M]\!]_d \vdash \mathrm{M} \triangleright^* \mathrm{N}$。

证明。对 CC_S 的归约规则进行归纳。

情况（S-RED-BETA）。假设（$\lambda^i x{:}\ A.M$） $N \triangleright M\{x \mapsto N\}$，目标是证明

$$\mathfrak{D} \vdash \mathfrak{L}_i\{\overline{x}\}\ @\ N \triangleright^* M[N/x],$$

其中，$\mathfrak{D} = [\![\Gamma]\!]_d \cup [\![M]\!]_d$和 $\overline{x}$对应于 λ^i 中的自由变量。根据 $[\![-]\!]_d$ 的定义，我们有 $\mathfrak{L}i(\{x : \_\}, x : \_ \mapsto M : \_) \in \mathfrak{D}$（忽略类型信息）。因此，$\mathfrak{D} \vdash \mathfrak{L}i\{x\}\ @\ N \triangleright M[\overline{x}/x, N/x] = \overline{M}[N/x]$。

情形（S-RED-BETA-ETA）。类似。

其他情形都是平凡的。

□

引理 3.12(相干性 (CC_S)). 若 $\Gamma \vdash M : A$，$\Gamma \vdash N : A$，且 $\vdash M \equiv N$，则 $\mathfrak{D} \vdash M \equiv N$，其中 $\mathfrak{D} = [\![\Gamma]\!]_d \cup [\![M]\!]_d \cup [\![N]\!]_d$。

证明。对 CC_S 的等价规则进行归纳。

情形 (S-EQ-REDUCE**).** 根据引理 3.11。

情形(S-EQ-ETA1**)**。假设我们有 $\Gamma \vdash M : \Pi x : A.B$，$\Gamma \vdash N : \Pi x : A.B$，并且根据 $\vdash M \equiv N$规则S-EQ-ETA1，那么我们必须有 $M \triangleright^* \lambda^i x : A.M'$，$N \triangleright^* N'$，以及 $\vdash M' \equiv N' x$。

根据引理3.11，我们有 $\mathfrak{D}_1 \vdash M \triangleright^* \mathfrak{L}i\{x\}$、$\overline{\mathfrak{D}_2} \vdash N \triangleright^* N'$，其中 $\mathfrak{D}_1 = [\![\Gamma]\!]_d \cup [\![M]\!]_d$、$\mathfrak{D}_2 = [\![\Gamma]\!]_d \cup [\![N]\!]_d$，以及 $\mathfrak{L}i(\{x : \_\}, x : \_ \mapsto M : \_) \in \mathfrak{D}_1$。根据归纳假设，$\mathfrak{D}_3 \vdash M' \triangleright^* N' @ x$，其中 $\mathfrak{D}_3 = [\![\Gamma]\!]_d \cup [\![M']\!]_d \cup [\![N']\!]_d$。

目标是证明$\mathfrak{D} \vdash M \equiv N$，使用规则 D-EQ-ETA1 （如下所示）。

$$\frac{\begin{array}{l}\mathfrak{D} \vdash L \triangleright^* \mathfrak{L}\{\overline{N}\} \\ \mathfrak{D} \vdash M \triangleright^* M' \\ \mathfrak{L}(\{\overline{x} : \overline{A}\}, x{:}A \mapsto N : B) \in \mathfrak{D} \\ \mathfrak{D} \vdash N[\overline{N}/\overline{x}] \equiv M'\ @\ x\end{array}}{\mathfrak{D} \vdash L \equiv M}\ \text{D-EQ-ETA1}$$

我们已经具备所有前提，例如 $M \triangleright^* \mathfrak{L}_i\{x\}$ 等，但它们并非在所需的标签上下文 $\mathfrak{D} = [\![\Gamma]\!]_d \cup [\![M]\!]_d \cup [\![N]\!]_d$ 中作出判断。我们只需证明 $\mathfrak{D}_1$、$\mathfrak{D}_2$ 和 $\mathfrak{D}_3$是 $\mathfrak{D}$的子集，然后应用弱化定理。$\mathfrak{D}_1$和 $\mathfrak{D}_2$显然是 $\mathfrak{D}$的子集。根据（1），我们有 $[\![M']\!]_d \subseteq [\![\Gamma]\!]_d \cup [\![M]\!]_d$和 $[\![N']\!]_d \subseteq [\![\Gamma]\!]_d \cup [\![N]\!]_d$。因此，$\mathfrak{D}_3$也是 $\mathfrak{D}$的子集。

情况（S-EQ-CLO1）。类似，但更繁琐 .

情况（S-EQ-ETA2**和** S-EQ-CLO2）。由对称性可知。

（注意，规则 S-EQ-REDUCE、S-EQ-ETA1和 S-EQ-ETA2 未列于图 5中，但它们与图 EQ-REDUCE、EQ-ETA1 和 EQ-ETA2中的规则完全相同，见图 2）。

□

引理 3 13 σ $\Gamma \vdash M \triangleright^* N$ $\Gamma \vdash M \triangleright^* M$ $\Gamma \vdash$。（归约序列保持性 $M \equiv N$ （ ））。若 ，则 ，其中 ，。

证明。 我们首先证明 Γ， $x{:}\ A \vdash M : B$和 $\Gamma \vdash N : A$蕴含 $\vdash \sigma(M[N/x]) \equiv M\{x \mapsto N\}$。我们通过对 M 的类型推导进行归纳来证明这一点。

情况（*ty-lambda*）。按定义，$(\lambda y{:}A.M)[N/x] = \lambda y{:}A[N/x].M[N/x]$。因此，$\sigma((\lambda y{:}A.M)[N/x]) = \lambda y{:}\sigma(A[N/x]).\sigma$ 我们有

$$((\lambda y{:}B.M)\{x \mapsto N\})\ y \triangleright (M\{x \mapsto N\})\{y \mapsto y\} \equiv M\{x \mapsto N\}.$$

根据归纳假设，$M\{x \mapsto N\} \equiv \sigma(M[N/x])$。因此，根据规则 s-eq-Eta1，我们有 $\sigma((\lambda y{:}B.M)[N/x]) \equiv ((\lambda y{:}B.M)\{x \mapsto N\})$。

其他情况都很简单。利用这个结果，我们有 $(\lambda x{:}A.M)\ N \triangleright M\{x \mapsto N\} \equiv \sigma(M[N/x])$，因此，$\sigma$保持小步归约。由此，归约序列的保持性也可立即得出。□

引理 3.14 (归约序列保持性)。对于所有 M 和 N，若 $\Gamma \vdash M : A$ 且 $M \triangleright^* N$，则有

$$\mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N \vdash \mathrm{M} \triangleright^* \mathrm{M}', \tag{5}$$

$$\mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N \vdash \mathrm{M}' \triangleright^* \mathrm{N} \tag{6}$$

对于某个 M'，其中 $(\mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N) = ([[\Gamma]]_d, [[M]]_d, [[N]]_d)$ 并且 $(\mathrm{M}, \mathrm{N}) = ([[M]], [[N]])$。

证明。 假设 $\Gamma \vdash M : A$ 且 $M \triangleright^* N$。那么，我们有 $\vdash M \triangleright^* M'$ 且 $\vdash M' \equiv N$ 对于某个 M'，因为 σ 保持归约序列。根据引理 3.11 和引理 3.12，

$$[[\Gamma]]_d \cup [[M]]_d \vdash \mathrm{M} \triangleright^* \mathrm{M}'$$

$$[[\Gamma]]_d \cup [[M]]_d \cup [[N]]_d \vdash \mathrm{M}' \equiv \mathrm{N}.$$

最后，由于 $([[\Gamma]]_d \cup [[M]]_d)$ 和 $([[\Gamma]]_d \cup [[M]]_d \cup [[N]]_d)$ 都是 $(\mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N)$ 的子集，因此有 $\mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N \vdash \mathrm{M} \triangleright^* \mathrm{M}'$ 和 $\mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N \vdash \mathrm{M}' \triangleright^* \mathrm{N}$，由弱化可得。 □

推论 3.15. *(正确性)* 对于所有地类型 A 和值 v 类型为 A，

$$\cdot \vdash M : A \wedge M \triangleright^* v \Longrightarrow \mathfrak{D}_\Gamma \cup \mathfrak{D}_M \vdash \mathrm{M} \triangleright^* \mathrm{v}' \ \textit{where}\ \mathrm{v}' \equiv \mathrm{v}.$$

引理 *3.16*（替换）。若 $\Gamma, x : A \vdash M : B$ 且 $\Gamma \vdash N : A$，则 $\mathfrak{D} \vdash [[M[N/x]]] \equiv M[N/x],,$ 其中 $\mathfrak{D} = \mathfrak{D}_\Gamma \cup \mathfrak{D}_M \cup \mathfrak{D}_N \cup \mathfrak{D}_M[N/x]$。

证明。根据引理 3.13 的证明，我们知道 $\Gamma, \mathrm{x} : \mathrm{A} \vdash \mathrm{M} : \mathrm{B}$ 且 $\Gamma \vdash \mathrm{N} : \mathrm{A}$ 蕴含 $\sigma(M[N/x]) \equiv M\{x \mapsto N\}$。这进一步蕴含 $\mathfrak{D}' \vdash [[\sigma(M[N/x])]] \equiv [[M\{x \mapsto N\}]],,$ 其中 $\mathfrak{D}' = [[\Gamma]]_d \cup [[M]]_d \cup [[N]]_d \cup [[\sigma\ M[N\ x]\ ]]_d$。

根据公式（4），$\mathfrak{D}' \subseteq \mathfrak{D}$。此外，根据（2）和（3），$[[\sigma(M[N/x])]] = [[M[N/x]]]$，$[[\Gamma]] = \Gamma$，以及 $[[M]][[[N]]/\mathrm{x}] = \mathrm{M}[\mathrm{N}/\mathrm{x}]$。因此，根据弱化，我们有 $\mathfrak{D} \vdash [[\mathrm{M}[\mathrm{N}/\mathrm{x}]]] \equiv \mathrm{M}[\mathrm{N}/\mathrm{x}]$。□

引理 3.17（相干性）。如果 $\mathfrak{D}_\Gamma \cup \mathfrak{D} \cup \mathfrak{D}$ $M : A$、$N : A$，且 MN，那么 MN，其中 MN。

证明。假设 $\Gamma \vdash \mathrm{M} : \mathrm{A}$，$\Gamma \vdash \mathrm{N} : \mathrm{A}$，且 $\vdash \mathrm{M} \equiv \mathrm{N}$。根据定义，我们有 $\vdash M \equiv N$，因为源语言不包含显式替换。根据引理 3.12，我们有 $\mathfrak{D}' \vdash \mathrm{M} \equiv \mathrm{N}$，其中 $\mathfrak{D}' = [[\Gamma]]_d \cup [[M]]_d \cup [[N]]_d$。由于 $\mathfrak{D}' \subseteq \mathfrak{D}$，根据弱化，我们有 $\mathfrak{D} \vdash \mathrm{M} \equiv \mathrm{N}$。□

定理 3.18 (类型保持)。对于所有类型良构程序 M，

$$\Gamma \vdash M : A \Longrightarrow \mathfrak{D}_\Gamma \cup \mathfrak{D}_M; \Gamma \vdash \mathrm{M} : \mathrm{A},$$

其中 $(\mathfrak{D}_\Gamma, \mathfrak{D}_M) = ([[\Gamma]]_d, [[M]]_d)$ 并且 $(\Gamma, \mathrm{M}, \mathrm{A}) = ([[\Gamma]], [[M]], [[A]])$。

证明。通过对互相定义的判断 $\vdash \Gamma$ 和 $\Gamma \vdash$ 进行同时归纳，共同证明以下两个陈述 $M :\ A$。

(1) $\vdash \Gamma \Longrightarrow \vdash \mathfrak{D}_\Gamma; \Gamma$.
(2) $\Gamma \vdash M : A \Longrightarrow \mathfrak{D}_\Gamma \cup \mathfrak{D}_M; \Gamma \vdash \mathrm{M} : \mathrm{A}$.

命题 1 可由归纳假设直接推出。我们考察命题 2 的各种情况。

情况（TY-VAR、TY-UNIVERSE**和** Π**规则**）。由归纳假设可得，显然成立。

情形 (ty-Apply) 假设 $\Gamma \vdash \mathrm{MN} : \mathrm{B}[\mathrm{N}/\mathrm{x}]$。目标是证明 $\mathfrak{D}$；$\Gamma \vdash \mathrm{M}\ @\mathrm{N} : [[B[N/x]]]$，其中 $\mathfrak{D} = (\mathfrak{D}_\Gamma \cup [[M\ N]]_d)$。根据归纳假设，我们有 $\mathfrak{D}_\Gamma \cup \mathfrak{D}\mathrm{M}$；$\Gamma \vdash \mathrm{M} : \Pi \mathrm{x}{:}\mathrm{A}.\mathrm{B}$ 以及 $\mathfrak{D}_\Gamma \cup \mathfrak{D}\mathrm{N}$；$\Gamma \vdash \mathrm{N} : \mathrm{A}$。根据规则 d-ty-Apply 和弱化，$\mathfrak{D}_\Gamma$；$\Gamma \vdash \mathrm{M}\ @\mathrm{N} : \mathrm{B}[\mathrm{N}/\mathrm{x}]$。根据替换引理，$\mathfrak{D}' \vdash \mathrm{B}[\mathrm{N}/\mathrm{x}] \equiv [[\mathrm{B}[\mathrm{N}/\mathrm{x}]]],,$ 其中 $\mathfrak{D}' = \mathfrak{D}_\Gamma \cup \mathfrak{D}\mathrm{B} \cup \mathfrak{D}\mathrm{N} \cup \mathfrak{D}\mathrm{B}[\mathrm{N}/\mathrm{x}]$。如果 $\mathfrak{D}' \subseteq \mathfrak{D}$，我们就得到目标。确实，

(1) $\mathfrak{D}_\Gamma \subseteq \mathfrak{D}$ 根据定义。（2） $\mathfrak{D}\mathrm{N} \subseteq \mathfrak{D}$ 因为 $\mathfrak{D}\mathrm{N} \subseteq [[\mathrm{M}\ \mathrm{N}\ ]]_d$ 根据定义。（3） $\mathfrak{D}\mathrm{B}[\mathrm{N}/\mathrm{x}] \subseteq \mathfrak{D}$ 因为 $\mathfrak{D}\mathrm{B}[\mathrm{N}/\mathrm{x}] \subseteq [[\mathrm{M}\ \mathrm{N}\ ]]_d$ 根据定义。（4） $\mathfrak{D}\mathrm{B} \subseteq \mathfrak{D}$，因为 $\mathfrak{D}\mathrm{N} \subseteq [[\Pi \mathrm{x}{:}\mathrm{A}.\mathrm{B}]]_d$ 根据定义，$[[\Pi \mathrm{x}{:}\mathrm{A}.\mathrm{B}]]_d \subseteq \mathfrak{D}\mathrm{M}$ 由 $\Gamma \vdash \mathrm{M} : \Pi \mathrm{x}{:}\mathrm{A}.\mathrm{B}$ 和引理 3.9 得出，并且 $\mathfrak{D}\mathrm{M} \subseteq \mathfrak{D}$ 根据定义。

[{"id":0,"output":"ACM 程序设计语言会议录，第 7 卷，第 PLDI 号，第 127 号文章。发布日期：2023年6月。"}]

情形（类型-λ）假设 $\Gamma \vdash \lambda^i x \;.\; \Pi \;.\; \mathfrak{D}\,\Gamma \vdash \mathfrak{L}\;\Pi \;.\; \mathfrak{D} =: \mathfrak{D}_\Gamma \cup \mathfrak{D}\;\mathfrak{L}\;, \mapsto \Gamma \vdash \overline{x}$ $AM:\ x:\ AB$。目标是; $_i\{x\} : x{:}AB$, 其中 $M_i(\{x:A\}\,x:A\,M:B)$。我们有 $x:A$，因为自由变量是类型正确的，因此，我们有 $\mathfrak{D};\Gamma \vdash x:A$，根据归纳假设和弱化。$\mathfrak{L}_i$ 的定义位于 $\mathfrak{D}$。

若 $\vdash \mathfrak{D};\ \Gamma$，则根据规则 D-TY-LABEL，即可得到目标。根据归纳假设以及 $\vdash \mathfrak{D}_\Gamma \cup \mathfrak{D}_M;\ \Gamma$ 的假设，我们有 $\Gamma,\ x:\ A \vdash M:\ B$。令 $\Gamma_{fv} =\cdot,\ x:\ A$，若以下陈述成立，则根据规则 $\vdash \mathfrak{D};\ \Gamma$ d-wf-Label 和弱化，我们即可得到。（

a) $\mathfrak{D}_\Gamma \cup \mathfrak{D}_M; \Gamma_{fv} \vdash \Pi x{:}A.B : U.$

(b) $\mathfrak{D}_\Gamma \cup \mathfrak{D}_M; \Gamma_{fv}, x:A \vdash M : B.$

根据引理3.5，$FV(\lambda^i x{:}A.M) \vdash \lambda^i x{:}A.M{:}\ \Pi x{:}A.B$，因此，$FV(\lambda^i x{:}A.M) \vdash \Pi x{:}A.B :U$，并且 $FV(\lambda^i x{:}A.M),\ x :A \vdash M{:}\ B$。因此，根据归纳假设和弱化，条件(a) 和 (b) 成立。情况 (ty-Eqiv)。根据相干性引理。

□

引理 3.19（与替换相容的逆向变换）。 如果 $\mathfrak{D};\ \Gamma,\ x:\ A \vdash M:\ B$ 且 $\mathfrak{D};\ \Gamma \vdash N:\ A$，则 $[[M[N/x]]] = M[N/x]$。。

证明。 对 M 的类型推导进行归纳。

情况（d-ty-标签）。假设 $\mathfrak{D};\ \Gamma, y:C \vdash \mathfrak{L}\{M\} : \Pi x{:}A.B,\ \mathfrak{L}(\{\Gamma\}, x:A \mapsto M:B) \in \mathfrak{D}$, and $\mathfrak{D};\ \Gamma \vdash N:C$。目标是证明 $[[\mathfrak{L}\{M\}[N/y]]] = (\lambda x{:}A[M/x].M[M/x])[N/y]$。确实，我们有

$$[[\mathfrak{L}\{\overline{M}\}[N/y]]] = [[\mathfrak{L}\{\overline{M}[N/y]\}]] = \lambda x{:}A[(\overline{M}[N/y])/\overline{x}].M[(\overline{M}[N/y])/\overline{x}]$$

根据归纳假设，并

$$\begin{aligned}(\lambda x{:}A[\overline{M}/\overline{x}].M[\overline{M}/\overline{x}])[N/y] &= \lambda x{:}((A[\overline{M}/\overline{x}])[N/y]).((M[\overline{M}/\overline{x}])[N/y]) \\ &= \lambda x{:}A[(\overline{M}[N/y])/\overline{x}].M[(\overline{M}[N/y])/\overline{x}]\end{aligned}$$

由于 y 在 A 和在 M 中不是自由变量（x 是其中的全部自由变量）。

其他情况要么是显然的，要么可直接由归纳假设得出。

□

引理 3.20。 若 $\mathfrak{D};\ \Gamma \vdash M:\ A$ 且 $\mathfrak{D} \vdash M \triangleright^* N$，则 $M \triangleright^* N$。

证明。 首先，我们通过归纳证明逆向变换保持小步归约。唯一的情况是规则D-RED-BETA。

情形（d-red-Beta）。假设 $\mathfrak{D};\ \Gamma \vdash \mathfrak{L}\{M\}@M': B,\ \mathfrak{L}(\{\Gamma\},\ x:A \mapsto N:B) \in \mathfrak{D}$，以及 $\mathfrak{D} \vdash \mathfrak{L}\{M\} @ N \triangleright^* N[M/x, M'/x]$。根据定义，我们有 $[[\mathfrak{L}\{M\}@M']] = \lambda x{:}A[M/x].M[M/x]$。事实上，我们有

$$\begin{aligned}(\lambda x{:}A[\overline{M}/\overline{x}].M[\overline{M}/\overline{x}])\ M' &\triangleright\ (N[\overline{M}/\overline{x}])[M'/x] \\ &= N[\overline{M}/\overline{x}, M'/x] \text{ since } x \text{ is not free in } \overline{M} \\ &= [[N[\overline{M}/\overline{x}, M'/x]]] \text{ by Lemma 3.19.}\end{aligned}$$

因此，逆向变换通过对归约序列中的小步数进行平凡归纳，保持了归约序列。

□

引理 3.21。 若 $\mathfrak{D};\ \Gamma \vdash M:\ A$，$\mathfrak{D};\ \Gamma \vdash N:\ A$，且 $\mathfrak{D} \vdash M \equiv N$，则 $\vdash M \equiv N$。

证明。通过对 DCC 的等价规则进行归纳。

情形（D-EQ-REDUCE）。根据引理3.20。

情形（D-EQ-ETA1）。假设我们有

$$\frac{\mathfrak{D} \vdash L \triangleright^* \mathfrak{L}\{\overline{N}\} \quad \mathfrak{D} \vdash M \triangleright^* M' \quad \mathfrak{L}(\{\overline{x} : \overline{A}\}, x{:}A \mapsto N : B) \in \mathfrak{D} \quad \mathfrak{D} \vdash N[\overline{N}/\overline{x}] \equiv M' @ x}{\mathfrak{D} \vdash L \equiv M}\ \text{D-EQ-ETA1}$$

我们的目标是证明 $\vdash L \equiv M$。根据引理 3.20，$L \triangleright^* (\lambda x{:}A[\overline{N}/\overline{x}].N\ [\overline{N}/\overline{x}])$ 且 $M \triangleright^* M'$。根据归纳假设和引理 3.19，$\vdash N'\ [\overline{N}/x] \equiv M'\ x$。因此，根据 CC 的规则 eq-Eta1，$\vdash L \equiv M$。情形（d-eq-Eta2）。由对称性可得。

□

引理 3.22。若 $\mathfrak{D}; \Gamma \vdash M : A$，则 $\Gamma \vdash M : A$。

证明。通过对相互定义的判断进行同时归纳，一并证明以下两个陈述 $\vdash \mathfrak{D}$；Γ 和 $\mathfrak{D}$；$\Gamma \vdash M : A$。

(1) $\vdash \mathfrak{D}; \Gamma \Longrightarrow \vdash \Gamma$.
(2) $\mathfrak{D}; \Gamma \vdash M : A \Longrightarrow \Gamma \vdash M : A$.

命题 1 可由归纳假设直接推出。我们考察命题 2 的各种情况。

情形（d-ty-Var、d-ty-Universe 和 d-ty-Pi）。由归纳法可知是平凡的。情形（d-ty-Apply）。假设 $\mathfrak{D}$；$\Gamma \vdash M@N:B[N/x]$, 目标是证明 $\Gamma \vdash MN$： $[[B[N/x]]]$。根据归纳假设，我们有 $\Gamma \vdash M : \Pi x{:}A.B$和 $\Gamma \vdash N: A$，这意味着 $\Gamma \vdash MN: B[N/x]$。根据引理3.19, $[[B[N/x]]] \equiv B[N/x]$。因此，根据等价规则，我们有 $\Gamma \vdash MN$： $[[B[N/x]]]$。情形（d-ty-Label）。类似。情形（d-ty-Eqiv）。根据引理 3.21

□

定理 3.23 (类型安全)。若 $\mathfrak{D}; \cdot \vdash M : A$，则 $\mathfrak{D} \vdash M \triangleright^* v$，其中 v 是不可约值。

定理 3.24 （一致性）。不存在由标签上下文 $\mathfrak{D}$ 和 *DCC* 项 M 组成的对 $\mathfrak{D}; \cdot \vdash M : \Pi A : U。A$。