四种分隔控制运算符的类型系统

Chiaki Ishio
ishio.chiaki@is.ocha.ac.jp
御茶之水大学 东京，日本

Kenichi Asai
asai@is.ocha.ac.jp 御
茶之水大学 日本东京

arXiv:2305.02852v1 [cs.PL] 4 May 2023

摘要

过去几十年中，控制算子的操作语义已得到全面研究，但控制算子的类型系统却鲜有探讨。shift、control 和 shift0 各自拥有不同的类型系统，彼此之间没有任何关联，而且一直没有能直接对应 control0 的类型系统。本文通过为这四种控制算子提供统一的类型系统，弥补了这一不足。遵循 Danvy 和 Filinski 的方法，我们从定义这四种控制算子操作语义的 CPS 解释器中推导出一个单态类型系统。通过在 Agda 中实现带类型的 CPS 解释器，我们证明了 CPS 翻译保持类型，并证明包含全部四种控制算子的演算是终止的。此外，我们还展示了我们的类型系统与先前针对 shift、control 和 shift0 的类型系统之间的关系。

CCS 概念： • **软件及其工程** → **控制结构**；解释器；函数式语言； • **计算理论** → 类型理论； 控制原语； •**计算数学** → λ 演算。

关键词： 类型系统，续延，带分隔的 control 运算符

ACM 参考格式：
Chiaki Ishio 和 Kenichi Asai。2022。《四种分隔控制算子的类型系统》。载于 第 *21* 届 *ACM SIGPLAN* 生成式编程：概念与经验国际会议论文集（*GPCE '22*），*2022* 年 *12* 月 *06–07* 日，新西兰奥克兰。ACM，纽约州纽约，14 页。https://doi.org/10.1145/3564719.35686

1 引言

续延表示计算的剩余部分。续延的概念用途广泛，因为它们可以被使用。

特此许可个人或课堂使用本作品的全部或部分内容，无需支付费用，前提是不得为营利或商业利益制作或分发副本，且副本首页须附有本声明及完整引文。本作品中由非ACM所有的组成部分，其版权必须得到尊重。允许在注明出处的情况下摘编。除上述情况外，如需复制、再版、发布到服务器或重新分发到列表，必须事先获得明确许可和/或支付费用。请通过 permissions@acm.org 申请许可。*GPCE '22*，*2022*年*12*月*6*日至*7*日，新西兰奥克兰© 2022 计算机协会。ACM
ISBN 978-1-4503-9920-3/22/12…$
15.00https://doi.org/10.1145/3564719.3568691

以暂停和恢复计算。其应用场景有很多，例如并发编程 [23] 和 Web 编程 [17]。

为了让程序员能够访问续延，人们提出了各种控制算子，以便在直接风格程序中处理续延。最早的是 Felleisen 提出的 control 和 prompt [11]，随后是 Danvy 和 Filinski 提出的 shift/reset [9]，、shift0/reset0 [8]，以及 control0/prompt0 [13]。这些算子大致可以分为两类，分类依据是控制算子能够捕获的计算范围是动态的还是静态的。

作为控制算子的替代方案，Plotkin 和 Pretnar [22] 提出了代数效应处理器来控制程序的执行流程。近年来，这一研究领域受到广泛关注，人们也为代数效应处理器 [3]的类型系统等方面付出了诸多努力。此外，代数效应处理器与分隔控制算子之间也存在密切联系。例如，shift0 [8] 与深层效应处理器 [12, 20] 相关，而 control0 [13] 与浅层效应处理器相关。

在这种情况下，厘清分隔控制运算符与代数效应处理器之间的类型层关系的需求日益迫切。为此，我们首先需要建立一个基础，使我们能够在类型层面讨论分隔控制运算符；本文将介绍这一基础。

本文的贡献如下：

- 我们从相应的续延传递风格解释器中推导出 control0 的单态类型规则（该规则允许答案类型修改）。
- 我们将这一推导方法扩展到完整的类型系统，从而推导出适用于 shift、shift0、control 和 control0 的类型系统。尽管其中一些运算符具有不同的动态行为，我们仍然精心设计了具有统一表示法的类型系统。
- 我们使用证明助手 Agda [19]形式化底层的续延传递风格解释器和类型系统。
- 我们厘清了现有研究中的类型系统与我们的类型系统之间的关系。特别是，我们表明，我们的类型系统涵盖了现有的类型系统。其中一些关系已在 Agda 中形式化。

本文其余部分安排如下。我们将在第2节介绍四种分隔控制算子及其差异，并在第3节展示底层语言的语法和语义。接下来，

91

我们将在第4节介绍四种分隔控制算子的类型系统，并在第5节比较以往研究与我们的工作。最后，我们将在第6 节讨论相关工作，并在第7节总结。我们将论文中提及的 Agda 代码作为补充材料提供[1]。

2 分隔控制算子

本节将介绍四种分隔控制算子：shift（S$k.\,e$）、control（F$\,k.\,e$）、shift0（S$_0k.\,e$）和 control0（F$_0k.\,e$）。每种算子都会捕获直到最近外围分隔符 reset（hei）为止的续延。例如，shift 会捕获如下所示的续延：

$$\langle(\mathcal{S}k.\,k\,(k\,2))+3\rangle+4 \;=\; \langle k\,(k\,2)\,[\lambda x.\langle x+3\rangle/k]\rangle+4 \;=\; 12$$

shift 捕获直到最近分隔符为止的外围上下文后，该上下文会绑定到变量 k，然后我们继续对 shift 的主体求值。

这四种算子在捕获当前上下文的方式以及处理捕获上下文的方式上各不相同。下面展示了每种算子的行为，其中 $E[\ldots]$ 是一个求值上下文，其洞不被 reset 包围。

$$\begin{array}{rcl}\langle E[\mathcal{S}k.\,e]\rangle &=& \langle e\,[\lambda x.\langle E[x]\rangle/k]\rangle\\ \langle E[\mathcal{F}k.\,e]\rangle &=& \langle e\,[\lambda x.\;E[x]\;/k]\rangle\\ \langle E[\mathcal{S}_0k.\,e]\rangle &=& e\,[\lambda x.\langle E[x]\rangle/k]\\ \langle E[\mathcal{F}_0k.\,e]\rangle &=& e\,[\lambda x.\;E[x]\;/k]\end{array}$$

与 shift 不同，shift0 捕获续延后，其周围的分隔符便会消失。另一方面，control 捕获的续延 k 并未被分隔符包围。这两种差异在 control0 中都会出现。

如果分隔控制算子嵌套使用，或算子包含在捕获的上下文中，这些差异就会变得至关重要。在下面的例子中，control 捕获的上下文范围并未被 reset 包围，这与 shift 和 shift0 不同。由于捕获的上下文 k_1 没有任何外围分隔符，k_1 中的 F$\,k_2.\ldots$ 捕获的不仅是其周围的原始上下文（即 $y+[\,]$），还包括 k_1的调用上下文 $2+[\,]$。

$$\begin{array}{rcl}&&\langle(\mathcal{F}k_1.\,(2+k_1\,1))+\mathcal{F}k_2.\,(4+k_2\,3)\rangle\\ &=&\langle(2+k_1\,1)\,[\lambda y.\,y+\mathcal{F}k_2.\,(4+k_2\,3)/k_1]\rangle\\ &=&\langle 2+(1+\mathcal{F}k_2.\,(4+k_2\,3))\rangle\\ &=&\langle(4+k_2\,3)\,[\lambda x.\,2+(1+x)/k_2]\rangle\\ &=&\langle 4+(2+(1+3))\rangle=10\end{array}$$

通过在捕获的延续中执行更多 control，contro
将捕获更多调用上下文。本文将这一系列调用上下文称为“轨迹”。为了定义 control 的类型规则，必须跟踪轨迹的类型。

同样，在下面的例子中，外层 shift0 运算符无法访问内层 reset 之外的上下文。然而，在第一个 shift0 求值后，第二个 shift0 可以访问外部上下文，因为第一个 shift0 移除了内部分隔符。

$$\begin{array}{rcl}&&\langle\langle(\mathcal{S}_0k_1.\,\mathcal{S}_0k_2.\,e)+2\rangle+3\rangle\\ &=&\langle(\mathcal{S}_0k_2.\,e\,[\lambda x.\langle x+2\rangle/k_1])+3\rangle\\ &=&e\,[\lambda x.\langle x+2\rangle/k_1,\lambda x.\langle x+3\rangle/k_2]\end{array}$$

通过嵌套更多的 shift0，我们可以访问更多外部上下文。我们将周围上下文的续延称为“元续延”。为了定义 shift0 的类型规则，有必要跟踪元续延的类型。

3 语法和语义

本节将介绍语言 λ_D 和 λ_C。语言 λ_D是一种按值调用、从左到右的 λ演算，扩展了数字和四种分隔控制算子。语言 λ_D 的语义由一个 CPS 解释器定义，该解释器将λ_D作为源语言，并将其翻译为 λ_C，这是一种标准的简单类型 λ演算，扩展了单位类型和序对，以支持元续延。该 CPS 解释器构成了第 4 节中类型系统的基础。

3.1 λ_D的语法

图 1 展示了 λ_D的语法。它是一种按值调用、从左到右的 λ演算，扩展了数字和带分隔符的 control 算子。可捕获续延的 control 算子有四种：`shift` ($\mathcal{S}$)、`control` ($\mathcal{F}$)、`shift0` ($\mathcal{S}_0$)，以及`control0` ($\mathcal{F}_0$)。这些算子分别与名为`reset`、`prompt`、`reset0`和`prompt0`的分隔符配对，它们的行为均相同 [15]。本文使用单个分隔符 ⟨⟩ 来表示上述四种分隔符。[2]

类型 λ_D （另见图 1）的形式相当复杂。续延的类型是理解 λ_D 中各类型的关键。在我们的类型系统中，它表示为 $\tau_1 \to \langle\mu,\;\sigma\rangle\,\tau_2$，其中它接收一个类型为 τ_1 的值、一个类型为 μ 的轨迹，以及一个类型为 σ 的元续延，并返回一个类型为 τ_2的值。牢记续延的类型后，函数的类型为 $\tau_1 \to \tau_2\langle\mu_\alpha,\;\sigma_\alpha\rangle\;\alpha\langle\mu_\beta,\;\sigma_\beta\rangle\;\beta$，这对应于标准函数类型 $\tau_1 \to \tau_2$，但求值时使用

- 一个类型为 $\tau_2 \to$ h$\mu_\alpha,\;\sigma_\alpha$i α 的续延，
- 一个类型为 μ_β 的轨迹，以及
- 一个类型为σ_β的元续延，

最后返回一个类型为 β 的值。

轨迹和元续延是带分隔的 control 操作符的内部表示，它们只会显式出现在第 3.2 节所示的 CPS 解释器中。然而，它们的类型对于定义第 4 节中操作符的类型至关重要。轨迹的类型要么是 $\bullet_\mu$，表示轨迹为空；要么是表示调用上下文组合的续延。元续延的类型要么是

[1]实现可从 https://github.com/chiaki-i/type4d获取。

[2]我们将 h i 称为 reset，但将 (TPrompt0) 用作该类型规则的名称。

$$
\begin{array}{rlcl}
\text{值} & v & := & n \mid x \mid \lambda x.\, e \\
\text{Term} & e & := & v \mid e_1\, e_2 \mid \mathrm{S}k.\, e \mid \mathrm{F}\, k.\, e \mid \mathrm{S}_0 k.\, e \mid \mathrm{F}_0 k.\, e \mid \mathrm{h}e\mathrm{i} \\
\text{值类型和项类型} & \tau,\ \alpha,\ \beta & := & \mathbb{N} \mid \tau_1 \to \tau_2 \mathrm{h}\mu_\alpha,\ \sigma_\alpha \mathrm{i}\ \alpha \mathrm{h}\mu_\beta,\ \sigma_\beta \mathrm{i}\ \beta \\
\text{轨迹类型} & \mu & := & \bullet_\mu \mid \tau_1 \to \mathrm{h}\mu,\quad \sigma \mathrm{i}\, \tau_2 \\
\text{元续延类型} & \sigma & := & \bullet_\sigma \mid ((\tau_1 \to \mathrm{h}\mu_1,\quad \sigma_1 \mathrm{i}\, \tau_2)\ \times \mu_2) :: \sigma_2
\end{array}
$$

图1. λD的语法 λ_D

$$
\begin{array}{l}
\text{值}\ v := \quad n \mid () \ \mid x \mid \lambda x.\, e \text{项}\ e: \\
= \quad v \mid e_1\, e_2 \mid (e_1,\quad e_2)\ \mid (\text{case}\, e_1\ \text{of}\ () \\
\Rightarrow e_2 \mid e_3 \Rightarrow e_4)\ \text{类型}\ \tau := \text{整数}\ \mid \text{单位类型} \\
\mid \tau_1 \to \tau_2 \mid \tau_1 \times \tau_2
\end{array}
$$

图2。语法 λ_C

$\bullet_\sigma$ 表示元续延为空，或者是由续延和外围上下文轨迹组成的序对列表。

"答案类型"是外围上下文的类型，或续延的返回类型。在图1中，箭头类型包含两个答案类型：α 是初始答案类型，而 β 是最终答案类型。初始答案类型指项预期的返回类型，而最终答案类型是项实际的返回类型。以表达式$\langle(\lambda x.\,\mathtt{isZero}\,(x-1))\,1\mathrm{i}$为例。假设 isZero 是一个从整数到布尔值的函数，那么该函数 $(\lambda x.\ldots)$ 的初始和最终答案类型都是布尔值，因为上下文会返回一个布尔值。另一方面，在表达式 $\mathrm{h}(\lambda x.\,\mathrm{isZero}\,(\mathrm{S}k.\,x-1)\,1\mathrm{i})$ 函数的初始答案类型（布尔值）变为最终答案类型（整数）：尽管上下文最初返回的值是布尔值（即 isZero 的返回值），但由于 shift，实际返回值是一个数字。这种类型变化称为答案类型修改（ATM），使用带定界控制运算符时可能会发生这种情况。

3.2 λ_C的语法

图2 展示了 λ_C的语法，它是第3.3节将介绍的 CPS 解释器的目标语言。基本上，它是一个标准的简单类型 λ演算，并扩展了单位类型、序对和分情况分析构造。我们将在第 λ_C 3.3节末尾详细说明的类型。

我们用类型为单位类型的 () 表示空轨迹，用续延（形式为 $\lambda v.\lambda t.\lambda m.\ldots$）表示非空轨迹。Shan [24]也使用相同的表示方式。将非空轨迹表示为函数，使我们能够拥有类型各异的调用上下文 [7]。

我们也用 () 表示空元续延。非空元续延则表示为一个（异构）列表，其中包含由一个续延和一条轨迹组成的序对；列表用 () 表示空列表，用序对表示 cons。

3.3 CPS 解释器

现在，我们定义 λ_D 的 CPS 解释器，记作 E [[ ]]。首先给出图 3 所示的 CPS 解释器定义。该解释器基于一个 2CPS[3] 解释器 [8]，接收一个续延 κ 和一个元续延 m。此外，它还接收一条轨迹 t，用于记录调用上下文。这与 Shan[24] 展示的解释器相同。一般而言，解释器 E [[e]] 接收一个环境ρ、一个续延 κ、一条轨迹 t 和一个元续延 m，并对项 e 求值。前四种情况中，元续延 m 和轨迹t 都不起作用；如果我们对它们进行 η-归约，解释器就会成为一个标准的 CPS 解释器。在该解释器中，我们用下划线标出轨迹的（重要部分），用波浪线标出元续延。对于分隔控制算子的情况，它会将捕获的续延绑定到变量 k，将 k 添加到环境 ρ 中，并使用其他参数递归地对 e 求值。

对于 shift 和 control，使用初始延续 k_{id}、空轨迹 () 和元续延 m 对主体 e 求值。也就是说，当前延续和轨迹会被清空，而元续延保持不变，这意味着主体 e 在空上下文中求值，并由 reset 限定。例如，$\mathrm{h}(\mathrm{F}\, k_1.\, e_1)\ +1\ \mathrm{i}$ 会变成 $\mathrm{h}e_1\mathrm{i}$，其中包含 $k_1 = \lambda x.\, x+1$；周围的上下文 $\lambda x.\, x+1$ 会被清空，而原始表达式周围的 reset 则会保留。

另一方面，在 shift0 和 control0 的情形中，e 会使用在元续延中找到的续延 κ_0 和轨迹 t_0 进行求值。也就是说，当前续延和轨迹取自当前分隔符之外的部分，这意味着主体 e 可以访问这些信息。例如，$\mathrm{h}E[\mathrm{h}(\mathrm{F}_0 k_1.\, e_1)\ +1\mathrm{i}]\ \mathrm{i}$ 会变成 $\mathrm{h}E[e_1]\mathrm{i}$，其中 $k_1 = \lambda x.\, x+1$，不仅外围上下文 $\lambda x.\, x+1$ 被清除，分隔符也被移除，从而 e_1 可以访问原始内部分隔符之外的上下文。

接下来，让我们关注每个分隔控制算子中所捕获续延的结构。对于 shift 和 shift0，当前续延和轨迹（κ'，t'）是

[3]当一个续延传递风格表达式再次被转换为续延传递风格时，它就称为 2CPS [9]。续延传递风格表达式携带一个续延，而 2CPS 表达式则同时携带一个续延和一个元续延。为了区分这两种风格，有时会将标准续延传递风格称为一重续延传递风格。

$$
\begin{array}{rcl}
k_{id} & = & \lambda v.\,\lambda t.\,\lambda m.\,\mathsf{case}\,(t,\,m)\,\mathsf{of}\,((),\,())\Rightarrow v \mid ((),\,((k_0,\,t_0),\,m_0))\Rightarrow k_0\,v\,t_0\,m_0 \mid (t,\,m)\Rightarrow t\,v\,()\,m \\
\_@\_ & = & \lambda t_1.\,\lambda t_2.\,\mathsf{case}\,(t_1,\,t_2)\,\mathsf{of}\,((),\,t_2)\Rightarrow t_2 \mid (k,\,t_2)\Rightarrow k :: t_2 \\
\_::\_ & = & \lambda k.\,\lambda t.\,\mathsf{case}\,(k,\,t)\,\mathsf{of}\,(k,\,())\Rightarrow k \mid (k,\,t)\Rightarrow \lambda v.\,\lambda t'.\,\lambda m'.\,k\,v\,(t :: t')\,m'
\end{array}
$$

$$
\begin{array}{l}
\mathrm{E}\,[\![x]\!]\,\rho\,\kappa \;=\; \lambda t.\,\lambda m.\,\kappa\,\rho(x)\;t\;m \qquad \mathrm{E}\,[\![n]\!]\,\rho\,\kappa \;=\; \lambda t.\,\lambda m.\,\kappa\,n\,t\,m \\
\mathrm{E}\,[\![\lambda x.\,e]\!]\,\rho\,\kappa \;=\; \lambda t.\,\lambda m.\,\kappa(\lambda v.\mathrm{E}\,[\![e]\!]\,\rho[v/x])\;t\;m \\
\mathrm{E}\,[\![e_1\,e_2]\!]\,\rho\,\kappa \;=\; \lambda t.\,\lambda m.\mathrm{E}\,[\![e_1]\!]\,\rho(\lambda v_1.\,\lambda t_1.\,\lambda m_1.\mathrm{E}\,[\![e_2]\!]\,\rho(\lambda v_2.\,\lambda t_2.\,\lambda m_2.\,v_1\,v_2\,\kappa\,t_2\,m_2)\;t_1\;m_1)\;t\;m \\
\mathrm{E}\,[\![\mathrm{S}k.\,e]\!]\,\rho\,\kappa \;=\; \lambda t.\,\lambda m.\mathrm{E}\,[\![e]\!]\,\rho[\lambda v.\,\lambda\kappa'.\,\lambda t'.\,\lambda m'.\,\kappa\,v\,t((\kappa',\,t')\;m')/k]\;k_{id}()\;m \\
\mathrm{E}\,[\![\mathrm{F}\;k.\,e]\!]\,\rho\,\kappa \;=\; \lambda t.\,\lambda m.\mathrm{E}\,[\![e]\!]\,\rho[\lambda v.\,\lambda\kappa'.\,\lambda t'.\,\lambda m'.\,\kappa\,v(t@\,(\kappa' :: t'))\;m'/k]\;k_{id}()\;m \\
\mathrm{E}\,[\![\mathrm{S}_0k.\,e]\!]\,\rho\,\kappa \;=\; \lambda t.\,\lambda((\kappa_0,\,t_0)\;m_0)。\;\mathrm{E}\,[\![e]\!]\,\rho[\lambda v.\,\lambda\kappa'.\,\lambda t'.\,\lambda m'.\,\kappa\,v\,t((\kappa',\,t')\;m')/k]\;\kappa_0\,t_0\,m_0 \\
\mathrm{E}\,[\![\mathrm{F}_0k.\,e]\!]\,\rho\,\kappa \;=\; \lambda t.\,\lambda((\kappa_0,\,t_0)\;m_0)\cdot\mathrm{E}\,[\![e]\!]\,\rho[\lambda v.\,\lambda\kappa'.\,\lambda t'.\,\lambda m'.\,\kappa\,v(t@\,(\kappa' :: t'))\;m'/k]\;\kappa_0\,t_0\,m_0 \\
\mathrm{E}\,[\![\mathrm{hei}]\!]\,\rho\,\kappa \;=\; \lambda t.\,\lambda m.\mathrm{E}\,[\![e]\!]\,\rho\;k_{id}()\,((\kappa,\,t)\quad m)
\end{array}
$$

图 3. λ_D 的 CPS 解释器

$$
\begin{array}{rcl}
\mathbb{N}^* & = & \mathtt{int} \\
(\tau_1 \to \tau_2\,\langle\mu_\alpha,\,\sigma_\alpha\rangle\,\alpha\,\langle\mu_\beta,\,\sigma_\beta\rangle\,\beta)^* & = & \tau_1^* \to (\tau_2^* \to \mu_\alpha^* \to \sigma_\alpha^* \to \alpha^*) \to \mu_\beta^* \to \sigma_\beta^* \to \beta^* \\
\bullet_\mu^* & = & \mathtt{unit} \\
(\tau_1 \to \langle\mu,\,\sigma\rangle\,\tau_2)^* & = & \tau_1^* \to \mu^* \to \sigma^* \to {\tau_2}^* \\
\bullet_\sigma^* & = & \mathtt{unit} \\
(((\tau_1 \to \langle\mu_1,\,\sigma_1\rangle\,\tau_2) \times \mu_2) :: \sigma_2)^* & = & ((\tau_1^* \to \mu_1^* \to \sigma_1^* \to {\tau_2}^*) \times {\mu_2}^*) \times {\sigma_2}^*
\end{array}
$$

图 4. λ_D 的类型级 CPS 转换

添加到当时的元续延 m'。由于控制算子捕获的是当前续延和轨迹，而不是元续延中的续延和轨迹，这意味着被捕获的续延 κ 无法直接访问那些续延和轨迹（除非使用 shift0 或 control0 来分解元续延）。也就是说，被捕获续延的主体保留了外围的 reset。

我们用对来表示元续延，以记录 κ' 和 t' 的每个快照，因为 `shift0` 和 `control0`需要访问元续延的每一层。如果控制算子只有 `shift` 和 `control`，那么元续延就可以表示为一个函数，正如 Danvy 和 Filinski 为 `shift/reset`[8]编写的 CPS 解释器中所示。

另一方面，对于 control 和 control0 捕获的续延，κ' 和 t' 会被追加到轨迹 t 中。这意味着，被捕获的续延 κ 确实可以直接访问续延和轨迹，并通过 shift 或 control 实现这一点（无需使用 shift0 或 control0 来分解元续延）。也就是说，被捕获续延的函数体并未被 reset 包围。

_@_ 运算符将两条轨迹连接起来，而 _:: _ 运算符则将一个续延 cons 到一条轨迹上。我们使用函数形式来将轨迹和续延都表示出来，如第3.1节所述。因此，_@_和_::_都通过函数复合来实现[4]（图3）。

现在我们介绍了 CPS 解释器及其辅助函数，接下来仔细考察这些函数的类型。要为三个辅助函数 k_{id}、_@_ 和 _::_ 中的分情况分析构造指定类型，需要检查给定参数的类型。例如，k_{id} 接受 $(t,\,m)$，并根据 t 和 m 是否都具有单位类型来分情况处理。与这些函数相对应，CPS 解释器也隐式地对 t 和 m[5] 的类型施加了额外约束。

最后，图 4 展示了类型的 CPS 翻译。轨迹 $\tau_1 \to \langle\mu,\,\sigma\rangle\,\tau_2$ in λ_D 被翻译为标准函数类型，其中所有参数都递归地进行翻译。元续延的类型是一个积，这意味着元续延表示为嵌套的序对。

$$\boxed{\Gamma \vdash e : \tau \,\langle \mu_\alpha,\, \sigma_\alpha\rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta\rangle\, \beta} \qquad \frac{x \,:\, \tau \in \Gamma}{\Gamma \vdash x : \tau \,\langle \mu_\alpha,\, \sigma_\alpha\rangle\, \alpha \,\langle \mu_\alpha,\, \sigma_\alpha\rangle\, \alpha}\ (\textsc{TVar}) \qquad \frac{}{\Gamma \vdash n : \mathbb{N} \,\langle \mu_\alpha,\, \sigma_\alpha\rangle\, \alpha \,\langle \mu_\alpha,\, \sigma_\alpha\rangle\, \alpha}\ (\textsc{TNum})$$

$$\frac{\Gamma,\, x \,:\, \tau_1 \vdash e : \tau_2 \,\langle \mu_\alpha,\, \sigma_\alpha\rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta\rangle\, \beta}{\Gamma \vdash (\lambda\, x.\, e) : (\tau_1 \to \tau_2 \,\langle \mu_\alpha,\, \sigma_\alpha\rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta\rangle\, \beta) \,\langle \mu_\gamma,\, \sigma_\gamma\rangle\, \gamma \,\langle \mu_\gamma,\, \sigma_\gamma\rangle\, \gamma}\ (\textsc{TLam})$$

$$\frac{\Gamma \vdash e_1 : (\tau_1 \to \tau_2 \,\langle \mu_\alpha,\, \sigma_\alpha\rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta\rangle\, \beta) \,\langle \mu_\gamma,\, \sigma_\gamma\rangle\, \gamma \,\langle \mu_\delta,\, \sigma_\delta\rangle\, \delta \quad \Gamma \vdash e_2 : \tau_1 \,\langle \mu_\beta,\, \sigma_\beta\rangle\, \beta \,\langle \mu_\gamma,\, \sigma_\gamma\rangle\, \gamma}{\Gamma \vdash (e_1\, e_2) : \tau_2 \,\langle \mu_\alpha,\, \sigma_\alpha\rangle\, \alpha \,\langle \mu_\delta,\, \sigma_\delta\rangle\, \delta}\ (\textsc{TApp})$$

$$\frac{\begin{array}{c}\text{id-cont-type}(\gamma,\, \mu_{id},\, \sigma_{id},\, \gamma') \\ \Gamma,\, k \,:\, \tau \to \tau_1 \,\langle \mu_1,\, \sigma_1\rangle\, \tau_2 \,\langle \mu_2,\, \sigma_2\rangle\, \alpha \vdash e : \gamma \,\langle \mu_{id},\, \sigma_{id}\rangle\, \gamma' \,\langle \bullet_\mu,\, \sigma_\beta\rangle\, \beta\end{array}}{\Gamma \vdash \mathcal{S}k.\, e : \tau \,\langle \mu_\beta,\, (((\tau_1 \to \langle \mu_1,\, \sigma_1\rangle\, \tau_2) \times \mu_2) :: \sigma_2)\rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta\rangle\, \beta}\ (\textsc{TShift})$$

$$\frac{\begin{array}{c}\text{id-cont-type}(\gamma,\, \mu_{id},\, \sigma_{id},\, \gamma') \\ \text{compatible}((\tau_1 \to \langle \mu_1,\, \sigma_1\rangle\, \tau_2),\, \mu_2,\, \mu_\gamma) \quad \text{compatible}(\mu_\beta,\, \mu_\gamma,\, \mu_\alpha) \\ \Gamma,\, k \,:\, \tau \to \tau_1 \,\langle \mu_1,\, \sigma_1\rangle\, \tau_2 \,\langle \mu_2,\, \sigma_\alpha\rangle\, \alpha \vdash e : \gamma \,\langle \mu_{id},\, \sigma_{id}\rangle\, \gamma' \,\langle \bullet_\mu,\, \sigma_\beta\rangle\, \beta\end{array}}{\Gamma \vdash \mathcal{F}k.\, e : \tau \,\langle \mu_\alpha,\, \sigma_\alpha\rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta\rangle\, \beta}\ (\textsc{TControl})$$

$$\frac{\Gamma,\, k \,:\, \tau \to \tau_1 \,\langle \mu_1,\, \sigma_1\rangle\, \tau_2 \,\langle \mu_2,\, \sigma_2\rangle\, \alpha \vdash e : \tau_0 \,\langle \mu_0,\, \sigma_0\rangle\, \tau_0' \,\langle \mu_0',\, \sigma_0'\rangle\, \beta}{\Gamma \vdash \mathcal{S}_0k.\, e : \tau \,\langle \mu_\beta,\, (((\tau_1 \to \langle \mu_1,\, \sigma_1\rangle\, \tau_2) \times \mu_2) :: \sigma_2)\rangle\, \alpha \,\langle \mu_\beta,\, (((\tau_0 \to \langle \mu_0,\, \sigma_0\rangle\, \tau_0') \times \mu_0') :: \sigma_0')\rangle\, \beta}\ (\textsc{TShift0})$$

$$\frac{\begin{array}{c}\text{compatible}((\tau_1 \to \langle \mu_1,\, \sigma_1\rangle\, \tau_2),\, \mu_2,\, \mu_\gamma) \quad \text{compatible}(\mu_\beta,\, \mu_\gamma,\, \mu_\alpha) \\ \Gamma,\, k \,:\, \tau \to \tau_1 \,\langle \mu_1,\, \sigma_1\rangle\, \tau_2 \,\langle \mu_2,\, \sigma_\alpha\rangle\, \alpha \vdash e : \tau_0 \,\langle \mu_0,\, \sigma_0\rangle\, \tau_0' \,\langle \mu_0',\, \sigma_0'\rangle\, \beta\end{array}}{\Gamma \vdash \mathcal{F}_0k.\, e : \tau \,\langle \mu_\alpha,\, \sigma_\alpha\rangle\, \alpha \,\langle \mu_\beta,\, (((\tau_0 \to \langle \mu_0,\, \sigma_0\rangle\, \tau_0') \times \mu_0') :: \sigma_0')\rangle\, \beta}\ (\textsc{TControl0})$$

$$\frac{\begin{array}{c}\text{id-cont-type}(\gamma,\, \mu_{id},\, \sigma_{id},\, \gamma') \\ \Gamma \vdash e : \gamma \,\langle \mu_{id},\, \sigma_{id}\rangle\, \gamma' \,\langle \bullet_\mu,\, (((\tau \to \langle \mu_\alpha,\, \sigma_\alpha\rangle\, \alpha) \times \mu_\beta) :: \sigma_\beta)\rangle\, \beta\end{array}}{\Gamma \vdash \langle e\rangle : \tau \,\langle \mu_\alpha,\, \sigma_\alpha\rangle\, \alpha \,\langle \mu_\beta,\, \sigma_\beta\rangle\, \beta}\ (\textsc{TPrompt0})$$

$$\begin{array}{lcl}
\text{id-cont-type}(\tau,\, \bullet_\mu,\, \bullet_\sigma,\, \tau') & = & \tau \equiv \tau' \\
\text{id-cont-type}(\tau,\, \bullet_\mu,\, (((\tau_1 \to \langle \mu_1,\, \sigma_1\rangle\, \tau_1') \times \mu_2) :: \sigma_2),\, \tau') & = & (\tau \equiv \tau_1) \wedge (\tau' \equiv \tau_1') \wedge (\mu_1 \equiv \mu_2) \wedge (\sigma_1 \equiv \sigma_2) \\
\text{id-cont-type}(\tau,\, (\tau_1 \to \langle \mu_1,\, \sigma_1\rangle\, \tau_1'),\, \sigma_2,\, \tau') & = & (\tau \equiv \tau_1) \wedge (\tau' \equiv \tau_1') \wedge (\mu_1 \equiv \bullet_\mu) \wedge (\sigma_1 \equiv \sigma_2)
\end{array}$$

$$\begin{array}{lcl}
\text{compatible}(\bullet_\mu,\, \mu_2,\, \mu_3) & = & \mu_2 \equiv \mu_3 \\
\text{compatible}((\tau_1 \to \langle \mu_1,\, \sigma_1\rangle\, \tau_1'),\, \bullet_\mu,\, \mu_3) & = & (\tau_1 \to \langle \mu_1,\, \sigma_1\rangle\, \tau_1') \equiv \mu_3 \\
\text{compatible}((\tau_1 \to \langle \mu_1,\, \sigma_1\rangle\, \tau_1'),\, (\tau_2 \to \langle \mu_2,\, \sigma_2\rangle\, \tau_2'),\, \bullet_\mu) & = & \bot \\
\text{compatible}((\tau_1 \to (\mu_1,\, \sigma_1)\, \tau_1'),\, (\tau_2 \to (\mu_2,\, \sigma_2)\, \tau_2'),\, (\tau_3 \to (\mu_3,\, \sigma_3)\, \tau_3')) & = & (\tau_1 \equiv \tau_3) \wedge (\tau_1' \equiv \tau_3') \wedge (\sigma_1 \equiv \sigma_3) \\
 & & \wedge (\text{compatible}((\tau_2 \to \langle \mu_2,\, \sigma_2\rangle\, \tau_2'),\, \mu_3,\, \mu_1))
\end{array}$$

图 5。 λ_D 的类型系统

4 λ_D 的类型系统

本节将介绍 λ_D（图 5）。按照 Danvy 和 Filinski [8]以及 Cong 等人 [7], , 我们直接从图 3中的 CPS 解释器推导出这一类型系统。

类型判断 $\Gamma \vdash e\colon\ \tau \mathrm{h} \mu_\alpha,\ \ \sigma_\alpha \mathrm{i}\ \alpha \mathrm{h} \mu_\beta,\ \ \sigma_\beta \mathrm{i}\ \beta$可读作："在类型环境 Γ 下，项 e具有类型 τ。当e在类型为 $\tau \to \mathrm{h} \mu_\alpha,\ \sigma_\alpha \mathrm{i}\ \alpha$ 的续延、类型为 μ_β 的轨迹以及类型为 σ_β 的元续延下求值时，它最终会规约为一个类型为 β 的值。"

图 5中的所有类型规则里，规则（TShift0）、（TControl0）和（TPrompt0）是全新的。规则（TShift）和（TControl）基于先前的工作[7, 8], , 但它们的创新之处在于同时考虑轨迹和元续延。

4.1 如何推导类型规则

在本节中，我们以（TControl0）为例，展示如何从 CPS 解释器推导出类型规则。下方是图 `control0`3 中的求值规则，图 3 中

[4]轨迹组合的实现对应于 Shan 的工作中的 `compose` 函数 [24]。 [5]这些约束随后在图 id-cont-type 和 compatible 5中实现。此外，Agda 中相应的实现严格遵循这些定义。

每个项右上角的类型注解。

$$\mathcal{E}[[\mathcal{F}_0 k.\, e]]\ \rho\ \kappa^{(\tau \to \langle \mu_\alpha,\ \sigma_\alpha \rangle\ \alpha)^*} =$$
$$\lambda t^{\mu_\beta^*}.\, \lambda((\kappa_0^{(\tau_0 \to \langle \mu_0,\ \sigma_0 \rangle\ \tau_0')^*},\ t_0^{\mu_0'^*}),\ m_0^{\sigma_0'^*}).$$
$$(\mathcal{E}[[e]]\ \rho[\lambda v^{\tau^*}.\, \lambda \kappa'^{(\tau_1 \to \langle \mu_1,\ \sigma_1 \rangle\ \tau_2)^*}.\, \lambda t'^{\mu_2^*}.\, \lambda m'^{\sigma_2^*}.$$
$$(\kappa^{(\tau \to \langle \mu_\alpha,\ \sigma_\alpha \rangle\ \alpha)^*}\ v^{\tau^*}$$
$$(t^{\mu_\beta^*}\ @\ (\kappa'^{(\tau_1 \to \langle \mu_1,\ \sigma_1 \rangle\ \tau_2)^*} :: t'^{\mu_2^*})^{\mu_\gamma^*})^{\mu_\alpha^*}\ m'^{\sigma_\alpha^*})^{\alpha^*}\ /\ k]$$
$$\kappa_0^{(\tau_0 \to \langle \mu_0,\ \sigma_0 \rangle\ \tau_0')^*}\ t_0^{\mu_0'^*}\ m_0^{\sigma_0'^*})^{\beta^*}$$

所有带注释的类型在展开类型的 CPS 翻译 * 后，都遵循目标语言 λ_C 的类型规则。根据这些类型，我们推导出 $\mathrm{F}_0 k.\, e$ 的类型规则。

求值 $\mathrm{F}_0 k.\, e$ 时，它会接收以下参数：

- 一个类型为 $(\tau \to \mathrm{h}\mu_\alpha,\ \sigma_\alpha \mathrm{i}\ \alpha)^*$ 的续延 κ，
- 一个类型为 μ_β^* 的轨迹 t，以及
- 一个类型为 $(((\tau_0 \to \mathrm{h}\mu_0,\ \sigma_0 \mathrm{i}\ \tau_0) \times \mu_0) :: \sigma_0)^*$ 的元续延 $((\kappa_0,\ t_0)\ m_0)$。

项的其余部分 $(\mathcal{E}[[e]]\ \ldots)$ 求值为一个类型为β^*的值。这些类型与（TControl0）的结论部分相匹配。注意，求值规则中的类型都根据图 4中的规则转换为 CPS。项 e 和 k的类型也是如此。我们保持带类型注释的 CPS 解释器与类型规则之间的精确对应关系。

剩下的唯一部分是轨迹组合（t@ κ':: t'）的类型约束，我们将在下一节中介绍这一点。

4.2 类型约束

在图 5中，有两个称为 兼容 和 id-cont-type的类型约束。前者用于轨迹组合，后者用于初始延续。与第 4.1节类似，这些约束直接源自图 3中的相应定义。

compatible 的规则分别对应轨迹组合 _@_ 和 _ ::_。如第 3.3 节所述，它们都实现为函数组合。当轨迹类型满足 compatible(μ_1, μ_2, μ_3) 时，这意味着我们有一个类型为 μ_1 的轨迹 t_1，以及一个类型为 μ_2 的轨迹 t_2，并将它们组合成一个类型为 μ_3 的轨迹 t_3。具体来说，compatible 的第一条规则对应于 () @t = t，第二条规则对应于 k:: () = k。第三条规则说明，组合非空轨迹的结果不应为空。最后一条规则似乎有些复杂，但这是由最后一条规则推导出来的：

$$k^{(\tau_1 \to \langle \mu_1,\ \sigma_1 \rangle\ \tau_1')^*} :: t^{(\tau_2 \to \langle \mu_2,\ \sigma_2 \rangle\ \tau_2')^*}$$
$$= \lambda v^{\tau_3^*}.\, \lambda t'^{\mu_3^*}.\, \lambda m'^{\sigma_3^*}.$$
$$(k^{(\tau_1 \to \langle \mu_1,\ \sigma_1 \rangle\ \tau_1')^*}\ v^{\tau_3^*}\ (t^{(\tau_2 \to \langle \mu_2,\ \sigma_2 \rangle\ \tau_2')^*} :: t'^{\mu_3^*})\ m'^{\sigma_3^*})^{\tau_3'^*}$$

在上面等式的最后一行，τ_1^* 应当等于 τ_3^*，这样 k 才能接收类型为 τ_3^*的值 v作为参数。此外，为了让 k 接收（t:: t'）作为第二个参数，还需要再次使用 compatible 来说明，将类型为（$\tau_2 \to \mathrm{h}\mu_2,\ \sigma_2 \mathrm{i}\ \tau_2'$）*和 μ_3^* 的两条轨迹组合后，应当得到一条类型为 μ_1^*的轨迹。这些约束对应于规则（TControl0）中的两个 compatible 约束。

接下来讨论初始延续的类型约束。在图 3 中，shift、control 和 prompt0 使用初始延续。因此，类型为 $\gamma \to \langle \mu_{id},\ \sigma_{id} \rangle\ \gamma'$ 的初始延续需要满足图 id-cont-type(γ、 μ_{id}、 σ_{id}、 γ') 中的约束 5。例如，下面展示了 k_{id} (图 3) 的第二种情况如何与类型相关联：

$$(k_{id}\ v^{\tau^*}\ ()^{\bullet_\mu^*}\ ((k_0,\ t_0),\ m_0)^{(((\tau_1 \to \langle \mu_1,\ \sigma_1 \rangle\ \tau_1') \times \mu_2) :: \sigma_2)^*})^{\tau'^*}$$
$$= k_0^{(\tau_1 \to \langle \mu_1,\ \sigma_1 \rangle\ \tau_1')^*}\ v^{\tau^*}\ t_0^{\mu_2^*}\ m_0^{\sigma_2^*}$$

The k_0 的箭头类型与其两个参数的类型相匹配，而 k_0 的返回类型与 k_{id} 的返回类型相匹配。

4.3 属性与观察

的 CPS 解释器和类型系统都 λ_D 以固有类型的方式在 Agda 中形式化 [1]，使用 PHOAS[5]。也就是说，所有项在构造时就已赋予类型，因此，成功地将 CPS 解释器实现为一个类型良好的 Agda 程序，意味着 CPS 翻译保留类型。

定理 4.1 (CPS翻译的类型保持).
若 $\Gamma \vdash e\colon \tau\langle \mu_\alpha,\ \sigma_\alpha \rangle\ \alpha\langle \mu_\beta,\ \sigma_\beta \rangle\ \beta$属于 λ_D，则对于任意尊重 Γ^* 的 ρ （即，对于任意 x， $\vdash \rho(x)\colon \Gamma^*(x)$），$\Gamma^* \vdash \mathcal{E}[[e]]\ \rho :(\tau^* \to \mu_\alpha^* \to \sigma_\alpha^* \to \alpha^*) \to \mu_\beta^* \to \sigma_\beta^* \to \beta^*$ 属于 λ_C。

此外，由于 CPS 解释器通过了 Agda 的终止性检查器，我们可以得出：给定初始延续、轨迹和元续延，对 λ_D项的求值总会终止。

定理 4.2 （终止性）。
如果$\vdash e\colon \tau\langle \bullet_\mu,\ \bullet_\sigma \rangle\ \tau\langle \bullet_\mu,\ \bullet_\sigma \rangle\ \tau$属于 λ_D，那么对 e的求值会终止。

5 与其他类型系统的比较

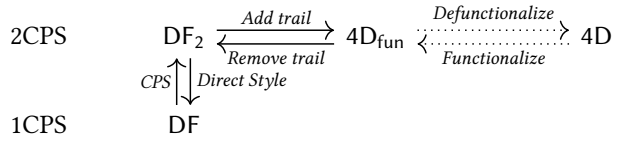
到目前为止，我们已经介绍了针对四种带分隔符的控制操作符的类型系统（下文简称 4D）。本节将它与先前研究中提出的其他类型系统进行比较。

5.1 shift/reset

在第 4 节中，我们遵循 Danvy 和 Filinski 的方法，从 CPS 解释器推导出了我们的类型系统 [8]。本节将他们针对 shift/reset(下文称为 DF) 的类型系统与 4D 进行比较。

以下是 DF 和 4D 底层解释器之间关系的概览图。图中的实线表示相应的转换是

已在 Agda 中形式化。



DF 和 4D 有三处不同：底层 CPS 解释器是用一重续延传递风格还是 2CPS 编写的，解释器是否将轨迹作为额外参数，以及元续延的数据结构是函数还是嵌套序对。如果我们将 DF 的 CPS 解释器再转换一次为 2CPS，就会得到 DF_2 的解释器。之后，如果给 DF_2 的解释器添加轨迹，就会得到 $4D_{fun}$ 的解释器。最后，如果对 $4D_{fun}$ 的元续延进行去函数化，就会得到 4D 的解释器。

在接下来的讨论中，我们将仔细考察以下类型系统之间的关系：A. DF 和 DF_2,B. DF_2与 $4D_{fun}$，以及 C. $4D_{fun}$ 与 4D。本节的目标是说明，项 e 在 DF 中可类型化，当且仅当 e 在 4D 中可类型化。请注意，所有这些解释器的源语言仅包含简单类型 λ 演算和 shift/reset。

***A. DF* 和 *DF_2*。** DF DF 的底层解释器采用一重续延传递风格，这意味着它不包含任何元续延。此外，该解释器不包含轨迹，因为除非源语言中含有 `control`或`control0`，否则没有必要使用轨迹。 DF 类型判断 $\Gamma \vdash_{\mathsf{DF}} e : \tau, \alpha, \beta$ 表示一个 DF 项 e 的类型为 τ，而对 e 求值会将答案类型从 α 变为 β。下面以 shift 为例，给出续延传递风格解释器及其类型规则 [8]。`shift`。

$$\mathcal{E}_{\mathsf{DF}}[\![\mathcal{S}k.\, e]\!]\, \rho\, \kappa = \mathcal{E}_{\mathsf{DF}}[\![e]\!]\, \rho\, [\lambda v.\, \lambda\kappa'.\, \kappa'\, (\kappa\, v)/k]\, (\lambda v'.\, v')$$

$$\frac{\Gamma,\, k : \tau \to \alpha,\, \delta,\, \delta \vdash_{\mathsf{DF}} e : \epsilon,\, \epsilon,\, \beta}{\Gamma \vdash_{\mathsf{DF}} \mathcal{S}k.\, e : \tau,\, \alpha,\, \beta}\ (\text{DF-Shift})$$

函数的类型（由 k 的类型所体现）形式为 $\tau_1 \to \tau_2,\ \alpha,\ \beta$ 它是一个从 τ_1 到 τ_2 的函数，并且函数体会将答案类型从 α 改变为 β。解释器和类型判断都比 4D 的更简单，因为与其他控制算子不同，shift 不会改变调用上下文或元上下文。

另一方面，DF_2 的类型判断与 4D; $\Gamma \vdash_{\text{DF-2}} e$: $\tau \mathrm{h} \sigma_\alpha \mathrm{i}\ \alpha \mathrm{h}^{\sigma_\beta} \mathrm{i}\ \beta$ 类似，表示 e 的类型为 τ; 如果在类型为 $\tau \to \mathrm{h}\sigma_\alpha\ \mathrm{i}\ \alpha$ 的续延和类型为 $^{\sigma_\beta}$ 的元续延下求值，e 会得到一个类型为 β 的值。下面给出 CPS 解释器（与 Danvy 和 Filinski [9]提出的解释器相同），以及从 DF_2 解释器推导出的 shift 类型规则。在 DF_2,中，元续延具有函数类型。

$$\begin{aligned}&\mathcal{E}_{\mathsf{DF2}}[\![\mathcal{S}k.\, e]\!]\, \rho\, \kappa = \lambda m.\\ &\quad \mathcal{E}_{\mathsf{DF2}}[\![e]\!]\, \rho\, [\lambda v.\, \lambda\kappa'.\, \lambda m'.\, \kappa\, v\, (\lambda v'.\, \kappa'\, v'\, m')/k]\\ &\qquad\qquad (\lambda v''.\, \lambda m''.\, m''\, v'')\, m\end{aligned}$$

$$\frac{\Gamma,\, k : \tau \to \tau_1\, \langle\sigma_1\rangle\, \tau_2\, \langle\sigma_1\rangle\, \alpha \vdash_{\mathsf{DF2}} e : \gamma\, \langle\gamma \to \gamma'\rangle\, \gamma'\, \langle\sigma_\beta\rangle\, \beta}{\Gamma \vdash_{\mathsf{DF2}} \mathcal{S}k.\, e : \tau\, \langle\tau_1 \to \tau_2\rangle\, \alpha\, \langle\sigma_\beta\rangle\, \beta}\ (\text{DF2-Shift})$$

现在我们有了这两个类型系统 DF 和 DF_2,，接下来证明一个项在 DF 中可定型，当且仅当它在 DF_2中可定型。图 6 描述了类型级翻译。第一个翻译（$(\tau, \gamma)^{\rightarrow^\tau_A}$）用于 DF 类型，它将 DF 类型 τ 递归地翻译成一个 DF_2 类型，并采用任意但固定的 DF_2 答案类型 γ。第二个翻译（$\Gamma^{\rightarrow^\Gamma_A}$）用于 DF 类型环境，其中 $\bullet_\Gamma$ 表示空类型环境。类似地，我们定义逆向翻译（$\tau^{\leftarrow^\tau_A}$ 和$\Gamma^{\leftarrow^\Gamma_A}$），用于去掉答案类型。直观来说，这些类型级翻译意味着元续延的答案类型不会改变，因为 shift 无法访问元续延。利用这些翻译，我们可以证明 DF 和 DF_2 的可定型性相同。

定理 5.1（DF 与 DF_2 之间的可定型性）。
(1) If $\Gamma \vdash_{DF} e : \tau, \alpha, \beta$, *then for any* DF_2 *type* γ,
$\Gamma^{\rightarrow^\Gamma_A} \vdash_{DF2} e : (\tau, \gamma)^{\rightarrow^\tau_A}\, \langle(\alpha, \gamma)^{\rightarrow^\tau_A} \to \gamma\rangle\, \gamma\, \langle(\beta, \gamma)^{\rightarrow^\tau_A} \to \gamma\rangle\, \gamma$.
(2) If $\Gamma \vdash_{DF} e : \tau\, \langle\alpha \to \gamma_1\rangle\, \gamma_2\, \langle\beta \to \gamma_3\rangle\, \gamma_4$, *then,*
$\Gamma^{\leftarrow^\Gamma_A} \vdash_{DF2} e : \tau^{\leftarrow^\tau_A}, \alpha^{\leftarrow^\tau_A}, \beta^{\leftarrow^\tau_A}$. □

证明采用对类型推导的归纳法，并已在 Agda 中形式化。

***B.DF_2* 与 *$4D_{fun}$*。** 接下来，我们转而讨论 DF_2 与 $4D_{fun}$ 之间的关系。与 DF_2 相比，$4D_{fun}$ 的解释器及其初始续延 k''_{id} 会额外接收一个轨迹作为参数。此外，类型判断的形式为$\Gamma \vdash_{\text{4D-fun}} e$: $\tau \mathrm{h} \mu_\alpha,\ \sigma_\alpha \mathrm{i}\ \alpha \mathrm{h}^{\mu_\beta},\ ^{\sigma_\beta} \mathrm{i}\ \beta$，其含义与 4D 中完全相同。请注意，(4Dfun-Shift) 中的所有轨迹都是空的，因为在求值过程中，shift 不会修改任何轨迹。

$$\begin{aligned}&\mathrm{E}_{\mathsf{4Dfun}}[\![\mathrm{S}k.\, e]\!]\, \rho\, \kappa = \lambda t.\, \lambda m.\mathrm{E}_{\mathsf{4Dfun}}[\![e]\!]\, \rho\, [\lambda v.\, \lambda\kappa'.\, \lambda t'\\ &.\, \lambda m'.\, \kappa\, v\, t(\lambda v'.\, \kappa'\, v'\, t'\, m')/k]k''_{id}\, ()\, m\text{其中 } k_{id} = \lambda v.\, \lambda t.\, \lambda m.\\ &\mathsf{case}\ (t,\ \ m)\ \mathsf{of}\ ((),\ ())\ \Rightarrow v|((),\ ((k_0,\ \ t_0)\ \ m_0)))\\ &\Rightarrow k_0\, v\, t_0\, m_0\, |\, (t,\ \ m)\ \Rightarrow t\, v\, ()\, m\end{aligned}$$

$$\frac{\begin{array}{l}\text{id-cont-type}(\gamma,\ \ \mu_{id},\ \ \sigma_{id},\ \ \gamma')\\ \Gamma,\, k\ :\ \ \tau \to \tau_1\langle^{\bullet_\mu},\ \ \sigma_1\rangle\, \tau_2\, \langle^{\bullet_\mu},\ \ \sigma_1\rangle\, \alpha\\ \quad \vdash_{\mathsf{4Dfun}} e\text{: } \gamma \mathrm{h}^{\bullet_\mu},\ \ \sigma_{id}\mathrm{i}\ \gamma'\, \mathrm{h}^{\bullet_\mu},\ \ ^{\sigma_\beta}\mathrm{i}\ \beta\end{array}}{\Gamma \vdash_{\mathsf{4Dfun}} \mathcal{S}k.\, e : \tau\, \langle\bullet_\mu,\, \tau_1 \to \tau_2\rangle\, \alpha\, \langle\bullet_\mu,\, \sigma_\beta\rangle\, \beta}\ (\text{4Dfun-Shift})$$

从 DF_2 到 $4D_{fun}$ ($\tau^{\rightarrow^\tau_B}, \sigma^{\rightarrow^\sigma_B}$，以及 $\Gamma^{\rightarrow^\Gamma_B}$) 的类型级翻译，以及从 $4D_{fun}$ 到 DF_2 ($\tau^{\leftarrow^\tau_B}, \sigma^{\leftarrow^\sigma_B}$，以及 $\Gamma^{\leftarrow^\Gamma_B}$) 的类型级翻译定义于图 7。我们只需为函数类型添加空轨迹或移除轨迹，因为对于 `shift`，轨迹并非必需，而且在 $4D_{fun}$中不会使用这些轨迹。借助这些类型级翻译，我们可以证明 DF_2 与 $4D_{fun}$ 的可定型性相同。

定理 5.2（DF_2 与 $4D_{fun}$ 之间的可定型性）。
(1) 若 $\Gamma \vdash_{DF2} e$: $\tau\langle\sigma_\alpha\rangle\ \alpha\langle\sigma_\beta\rangle\ \beta$，则
$\Gamma^{\rightarrow^\Gamma_B} \vdash_{4Dfun} e : \tau^{\rightarrow^\tau_B}\, \langle\bullet_\mu, \sigma_\alpha{}^{\rightarrow^\sigma_B}\rangle\, \alpha^{\rightarrow^\tau_B}\, \langle\bullet_\mu, \sigma_\beta{}^{\rightarrow^\sigma_B}\rangle\, \beta^{\rightarrow^\tau_B}$.

$$
\begin{array}{rcl}
(\mathbb{N},\ \gamma)^{\rightarrow^{\tau}_{\mathrm{A}}} & = & \mathbb{N} \\
(((\tau_1 \to \tau_2),\ \alpha,\ \beta),\ \gamma)^{\rightarrow^{\tau}_{\mathrm{A}}} & = & (\tau_1,\ \gamma)^{\rightarrow^{\tau}_{\mathrm{A}}} \to (\tau_2,\ \gamma)^{\rightarrow^{\tau}_{\mathrm{A}}} \langle (\alpha,\ \gamma)^{\rightarrow^{\tau}_{\mathrm{A}}} \to \gamma \rangle\ \gamma\ \langle (\beta,\ \gamma)^{\rightarrow^{\tau}_{\mathrm{A}}} \to \gamma \rangle\ \gamma \\
\bullet_{\Gamma}{}^{\rightarrow^{\Gamma}_{\mathrm{A}}} & = & \bullet_{\Gamma} \\
(x : \tau,\ \Gamma)^{\rightarrow^{\Gamma}_{\mathrm{A}}} & = & x : (\tau,\ \gamma)^{\rightarrow^{\tau}_{\mathrm{A}}},\ \Gamma^{\rightarrow^{\Gamma}_{\mathrm{A}}}
\end{array}
$$

$$
\begin{array}{rcl}
\mathbb{N}^{\leftarrow^{\tau}_{\mathrm{A}}} & = & \mathbb{N} \\
\tau_1 \to \tau_2\ \langle \alpha \to \gamma_1 \rangle\ \gamma_2\ \langle \beta \to \gamma_3 \rangle\ \gamma_4{}^{\leftarrow^{\tau}_{\mathrm{A}}} & = & (\tau_1{}^{\leftarrow^{\tau}_{\mathrm{A}}} \to \tau_2{}^{\leftarrow^{\tau}_{\mathrm{A}}}),\ \alpha^{\leftarrow^{\tau}_{\mathrm{A}}},\ \beta^{\leftarrow^{\tau}_{\mathrm{A}}} \\
\bullet_{\Gamma}{}^{\leftarrow^{\Gamma}_{\mathrm{A}}} & = & \bullet_{\Gamma} \\
(x : \tau,\ \Gamma)^{\leftarrow^{\Gamma}_{\mathrm{A}}} & = & x : \tau^{\leftarrow^{\tau}_{\mathrm{A}}},\ \Gamma^{\leftarrow^{\Gamma}_{\mathrm{A}}}
\end{array}
$$

图 6。DF 与 DF_2 之间的类型级翻译

$$
\begin{array}{rcl}
\mathbb{N}^{\rightarrow^{\tau}_{\mathrm{B}}} & = & \mathbb{N} \\
(\tau_1 \to \tau_2\ \langle \sigma_1 \rangle\ \alpha\ \langle \sigma_2 \rangle\ \beta)^{\rightarrow^{\tau}_{\mathrm{B}}} & = & \tau_1{}^{\rightarrow^{\tau}_{\mathrm{B}}} \to \tau_2{}^{\rightarrow^{\tau}_{\mathrm{B}}}\ \langle \bullet_{\mu},\ \sigma_1{}^{\rightarrow^{\sigma}_{\mathrm{B}}} \rangle\ \alpha^{\rightarrow^{\tau}_{\mathrm{B}}}\ \langle \bullet_{\mu},\ \sigma_2{}^{\rightarrow^{\sigma}_{\mathrm{B}}} \rangle\ \beta^{\rightarrow^{\tau}_{\mathrm{B}}} \\
(\tau_1 \to \tau_2)^{\rightarrow^{\sigma}_{\mathrm{B}}} & = & \tau_1{}^{\rightarrow^{\tau}_{\mathrm{B}}} \to \tau_2{}^{\rightarrow^{\tau}_{\mathrm{B}}} \\
\bullet_{\Gamma}{}^{\rightarrow^{\Gamma}_{\mathrm{B}}} & = & \bullet_{\Gamma} \\
(x : \tau,\ \Gamma)^{\rightarrow^{\Gamma}_{\mathrm{B}}} & = & x : \tau^{\rightarrow^{\tau}_{\mathrm{B}}},\ \Gamma^{\rightarrow^{\Gamma}_{\mathrm{B}}}
\end{array}
$$

$$
\begin{array}{rcl}
\mathbb{N}^{\leftarrow^{\tau}_{\mathrm{B}}} & = & \mathbb{N} \\
(\tau_1 \to \tau_2\ \langle \mu_1,\ \sigma_1 \rangle\ \alpha\ \langle \mu_2,\ \sigma_2 \rangle\ \beta)^{\leftarrow^{\tau}_{\mathrm{B}}} & = & \tau_1{}^{\leftarrow^{\tau}_{\mathrm{B}}} \to \tau_2{}^{\leftarrow^{\tau}_{\mathrm{B}}}\ \langle \sigma_1{}^{\leftarrow^{\sigma}_{\mathrm{B}}} \rangle\ \alpha^{\leftarrow^{\tau}_{\mathrm{B}}}\ \langle \sigma_2{}^{\leftarrow^{\sigma}_{\mathrm{B}}} \rangle\ \beta^{\leftarrow^{\tau}_{\mathrm{B}}} \\
(\tau_1 \to \tau_2)^{\leftarrow^{\sigma}_{\mathrm{B}}} & = & \tau_1{}^{\leftarrow^{\tau}_{\mathrm{B}}} \to \tau_2{}^{\leftarrow^{\tau}_{\mathrm{B}}} \\
\bullet_{\Gamma}{}^{\leftarrow^{\Gamma}_{\mathrm{B}}} & = & \bullet_{\Gamma} \\
(x : \tau,\ \Gamma)^{\leftarrow^{\Gamma}_{\mathrm{B}}} & = & x : \tau^{\leftarrow^{\tau}_{\mathrm{B}}},\ \Gamma^{\leftarrow^{\Gamma}_{\mathrm{B}}}
\end{array}
$$

图 7。 DF_2 与 $4\mathrm{D}_{\mathrm{fun}}$ 之间的类型级翻译

$$
\begin{array}{rcl}
(\mathbb{N},\ \gamma)^{\leftarrow^{\tau}_{\mathrm{C}}} & = & \mathbb{N} \\
(\tau_1 \to \tau_2\ \langle \mu_1,\ \sigma_1 \rangle\ \alpha\ \langle \mu_2,\ \sigma_2 \rangle\ \beta,\ \gamma)^{\leftarrow^{\tau}_{\mathrm{C}}} & = & (\tau_1,\ \gamma)^{\leftarrow^{\tau}_{\mathrm{C}}} \to (\tau_2,\ \gamma)^{\leftarrow^{\tau}_{\mathrm{C}}} \langle (\mu_1,\ \gamma)^{\leftarrow^{\mu}_{\mathrm{C}}},\ (\sigma_1,\ \gamma)^{\leftarrow^{\sigma}_{\mathrm{C}}} \rangle\ (\alpha,\ \gamma)^{\leftarrow^{\tau}_{\mathrm{C}}} \\
 & & \langle (\mu_2,\ \gamma)^{\leftarrow^{\mu}_{\mathrm{C}}},\ (\sigma_2,\ \gamma)^{\leftarrow^{\sigma}_{\mathrm{C}}} \rangle\ (\beta,\ \gamma)^{\leftarrow^{\tau}_{\mathrm{C}}} \\
(\bullet_{\mu},\ \gamma)^{\leftarrow^{\mu}_{\mathrm{C}}} & = & \bullet_{\mu} \\
(\tau_1 \to \langle \mu_1,\ \sigma_1 \rangle\ \tau_2,\ \gamma)^{\leftarrow^{\mu}_{\mathrm{C}}} & = & (\tau_1,\ \gamma)^{\leftarrow^{\tau}_{\mathrm{C}}} \to \langle (\mu_1,\ \gamma)^{\leftarrow^{\mu}_{\mathrm{C}}},\ (\sigma_1,\ \gamma)^{\leftarrow^{\sigma}_{\mathrm{C}}} \rangle\ (\tau_2,\ \gamma)^{\leftarrow^{\tau}_{\mathrm{C}}} \\
(\bullet_{\sigma},\ \gamma)^{\leftarrow^{\sigma}_{\mathrm{C}}} & = & \gamma \to \gamma \\
(((\tau_1 \to \langle \mu_1,\ \sigma_1 \rangle\ \tau_2) \times \mu_2) :: \sigma_2,\ \gamma)^{\leftarrow^{\sigma}_{\mathrm{C}}} & = & (\tau_1,\ \gamma)^{\leftarrow^{\sigma}_{\mathrm{C}}} \to (\tau_2,\ \gamma)^{\leftarrow^{\tau}_{\mathrm{C}}} \\
\bullet_{\Gamma}{}^{\leftarrow^{\Gamma}_{\mathrm{C}}} & = & \bullet_{\Gamma} \\
(x : \tau,\ \Gamma)^{\leftarrow^{\Gamma}_{\mathrm{C}}} & = & x : (\tau,\ \gamma)^{\leftarrow^{\tau}_{\mathrm{C}}},\ \Gamma^{\leftarrow^{\Gamma}_{\mathrm{C}}}
\end{array}
$$

图 8。 从 4D 到 $4\mathrm{D}_{\mathrm{fun}}$ 的类型级翻译（该翻译不起作用）

(2) 如果 $\Gamma \vdash_{4Dfun} e : \tau_1 \to \tau_2\ \langle \mu_1,\ \sigma_1 \rangle\ \alpha\ \langle \mu_2,\ \sigma_2 \rangle\ \beta$，那么 $\Gamma^{\leftarrow^{\Gamma}_{B}} \vdash_{DF2} e : \tau_1{}^{\leftarrow^{\tau}_{B}} \to \tau_2{}^{\leftarrow^{\tau}_{B}} \langle \sigma_1{}^{\leftarrow^{\sigma}_{B}} \rangle\ \alpha^{\leftarrow^{\tau}_{B}} \langle \sigma_2{}^{\leftarrow^{\sigma}_{B}} \rangle\ \beta^{\leftarrow^{\tau}_{B}}$。 □

证明再次通过对类型推导进行归纳法完成，并已在 Agda 中形式化。

C. $4D_{fun}$ 和 4D。 最后，我们探讨 $4\mathrm{D}_{\mathrm{fun}}$ 和 4D 之间的关系。4D 的底层解释器是 $4\mathrm{D}_{\mathrm{fun}}$ 的一个变体，其中 $4\mathrm{D}_{\mathrm{fun}}$的元续延经过了去函数化处理。

以下是 4D 的求值规则（摘自图 3），以及针对 `shift` 稍作修改的类型规则。

$$
\begin{array}{l}
\mathrm{E}\ [\![\mathrm{S}k.\, e]\!]\ \rho\ \kappa\ =\ \lambda t.\, \lambda m. \\
\quad \mathrm{E}\ [\![e]\!]\ \rho[\lambda v.\, \lambda \kappa'.\, \lambda t'.\, \lambda m'.\, \kappa\ v\ t((\kappa',\ \ t')\ m')/k]\ k_{id}()\ m\text{其中} \\
\quad k_{id} = \lambda v.\, \lambda t.\, \lambda m.\, \mathsf{case}\ (t,\ \ m)\ \mathsf{of}\ ((),\ \ ())\ \Rightarrow v\,|\,((),\ \ ((k_0,\ \ t_0)\ m_0)))\ \Rightarrow k_0\ v\ t_0\ m_0\,|\,(t,\ \ m)\ \Rightarrow t\ v()\ m
\end{array}
$$

$$
\begin{array}{rcl}
(\mathbb{N}, \gamma)^{\rightarrow^{\tau}_{A'}} & = & \mathbb{N} \\
((\tau \langle \mu_1 \rangle \alpha \langle \mu_2 \rangle \beta), \gamma)^{\rightarrow^{\tau}_{A'}} & = & (\tau, \gamma)^{\rightarrow^{\tau}_{A'}} \langle (\mu_1, \gamma)^{\rightarrow^{\tau}_{A'}}, ((\alpha, \gamma)^{\rightarrow^{\tau}_{A'}} \rightarrow \gamma) \rangle \gamma \langle (\mu_2, \gamma)^{\rightarrow^{\tau}_{A'}}, ((\beta, \gamma)^{\rightarrow^{\tau}_{A'}} \rightarrow \gamma) \rangle \gamma \\
(\bullet_\mu, \gamma)^{\rightarrow^{\mu}_{A'}} & = & \bullet_\mu \\
((\tau_1 \rightarrow \langle \mu \rangle \tau_2), \gamma)^{\rightarrow^{\mu}_{A'}} & = & (\tau_1, \gamma)^{\rightarrow^{\tau}_{A'}} \rightarrow \langle ((\mu, \gamma)^{\rightarrow^{\mu}_{A'}}, (\tau_1, \gamma)^{\rightarrow^{\tau}_{A'}} \rightarrow \gamma) \rangle \gamma \\
\bullet_\Gamma{}^{\rightarrow^{\Gamma}_{A'}} & = & \bullet_\Gamma \\
(x : \tau, \Gamma)^{\rightarrow^{\Gamma}_{A'}} & = & x : (\tau, \gamma)^{\rightarrow^{\tau}_{A'}}, \Gamma^{\rightarrow^{\Gamma}_{A'}}
\end{array}
$$

图 9. 从 CP 到 $4D_{fun}$ 的类型级翻译

$$
\frac{\begin{array}{l}\text{id-cont-type}(\gamma, \mu_{id}, \sigma_{id}, \gamma') \\ \Gamma, k : \tau \rightarrow \tau_1 \langle \bullet_\mu, \sigma_1 \rangle \tau_1 \langle \bullet_\mu, \sigma_1 \rangle \alpha \\ \quad \vdash e : \gamma \langle \mu_{id}, \sigma_{id} \rangle \gamma' \langle \bullet_\mu, \sigma_\beta \rangle \beta\end{array} \qquad (\text{TSHIFT}')}{\begin{array}{l}\Gamma \vdash \mathcal{S}k.e : \\ \quad \tau \langle \mu_\beta, (\tau_1 \rightarrow \langle \bullet_\mu, \sigma_1 \rangle \tau_2) \times \bullet_\mu :: \sigma_1 \rangle \alpha \langle \mu_\beta, \sigma_\beta \rangle \beta\end{array}}
$$

在 (TSHIFT′) 中，请注意，下划线部分的类型与波浪线部分的类型相同（与图 TSHIFT5 中原始的 5 中的情况不同）。这是因为本节中 4D 的源语言只包含 `shift` 这一分隔控制算子。求值 `shift` 不会改变元续延，而 `shift0` 或 `control0` 则会。

$4D_{fun}$ 和 4D 的解释器接收相同的轨迹和元续延，因此两个解释器中捕获的续延 k 也是如此。相应地，$4D_{fun}$ 和 4D 中的每条类型规则都具有相同的类型参数。唯一的区别在于，轨迹类型和元续延类型是否被显式存储在元续延中。具体而言，(4Dfun-Shift) 在 k 中包含类型 μ_1 和 σ_1，但在 $\mathcal{S}k.e$ 的元续延类型（$\tau_1 \rightarrow \tau_2$）中并未显式包含它们。另一方面，$\mathcal{S}k.e$ 在 (TShift′) 中的元续延类型包含类型 $\bullet_\mu$ 和 σ_1，如 $((\tau_1 \rightarrow \mathrm{h}^{\bullet_\mu}, \ \sigma_1 \mathrm{i}\, \tau_2) \times \bullet_\mu :: \sigma_1)$ 所示。

现在很清楚，我们可以将 $4D_{fun}$ 中的每条类型规则转换为 4D 中对应的规则（反之亦然）：在结论的元续延类型中添加（删除）前提中的轨迹和元续延类型。所有其他类型规则也同样如此（见附录）。因此，项 e 在 4D 中可类型化，当且仅当它在 $4D_{fun}$ 中可类型化，由此证明以下定理。

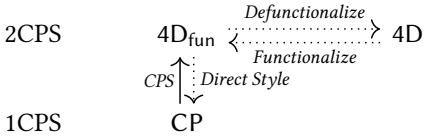
定理 5.3（$4D_{fun}$ 与 4D 之间的可定型性）。
(1) 若 $\Gamma \vdash_{4Dfun} e : \tau \, h\mu_1, \sigma_1 i \, \alpha \, h\mu_2, \sigma_2 i \ \beta$，则 $\Gamma \vdash_{4D} e : \tau' \, h\mu_1', \sigma' 1 i \ \alpha' \, h\mu_2', \sigma' 2 i \ \beta'$，其中 τ'、μ_1'、$\sigma' 1$、α'、μ_2'、σ_2' 和 β' 均存在。*(2)* 若 $\Gamma \vdash_{4D} e : \tau \, h\mu_1, \sigma_1 i \, \alpha \, h\mu_2, \sigma_2 i \ \beta$，则 $\Gamma \vdash_{4Dfun} e : \tau' h\mu_1', \sigma' 1 i \ \alpha' h\mu_2', \sigma' 2 i \ \beta'$，其中 τ'、μ_1'、$\sigma' 1$、α'、μ_2'、σ_2' 和 β' 均存在。□

我们尚未在 Agda 中形式化这个定理，因为对于第一部分，目前还不清楚如何定义类型级翻译。一个简单的翻译是在 $4D_{fun}$ 的非空元续延（$\tau_1 \rightarrow \tau_2$）中加入任意的 μ 和 σ，以创建 4D 的非空元续延 $(((\tau_1 \rightarrow \mathrm{h}\mu, \ \sigma \mathrm{i}\ \tau_2) \times \mu) :: \sigma)$ 行不通，因为这会导致 (TShift′) 中出现冲突。在 (TShift′) 的情形下，必须证明 $\mathcal{S}k.e$ 的类型为 $\tau \mathrm{h}^{\mu_\beta}, \ ((\tau_1 \rightarrow \mathrm{h}\mu, \ \sigma \mathrm{i}\ \tau_2) \times \mu) :: \sigma \mathrm{i}\ \alpha \mathrm{h}^{\mu_\beta}, \ \sigma_\beta \mathrm{i}\ \beta$，但我们只有一个类型为 $(\tau \rightarrow \tau_1 \mathrm{h}^{\bullet_\mu}, \ \sigma_1 \mathrm{i}\, \tau_2 \mathrm{h}^{\bullet_\mu}, \ \sigma_1 \mathrm{i}\ \alpha)$ 的变量 k，其中 k 中的类型 $\bullet_\mu$ 和 σ_1 与该情形目标中的类型 μ 和 σ 不匹配。

对于定理的第二部分，可以仿照图 8来定义类型级翻译，其中空元续延 $\bullet_\sigma$ 被翻译为 $\gamma \rightarrow \gamma$，对于给定的$\gamma$而言。然而，这种翻译并不可行，因为将 $\bullet_\sigma$ 统一翻译为相同的 $\gamma \rightarrow \gamma$ 最终会与约束 id-cont-type发生冲突。为了定义精确的对应关系，我们需要微调 $\bullet_\sigma$的翻译，这相当于比较 4D 和 $4D_{fun}$中的类型推导。

5.2 control/prompt

Cong 等人 [7] 从相应的一重续延传递风格解释器中推导出了 `control/prompt`（下文称为 CP）的类型系统。CP 与 4D 之间的关系如下。由于 CP 已经包含了用于处理调用上下文的轨迹，下面的图示比第 5.1节中的图示更为直接。CP 与 4D 之间的区别在于底层解释器是一重续延传递风格还是二重续延传递风格，以及元续延是表示为函数还是序对。



从 CP 到 $4D_{fun}$ 的翻译定义为 $\rightarrow^*{}_{A'}$，见图 9。翻译规则是图 6前半部分规则的一个变体。根据这些规则，我们证明了一个 CP项在 $4D_{fun}$中可类型化，反之亦然。

定理 5.4（CP 与 $4D_{fun}$ 之间的可定型性）。
(1) If $\Gamma \vdash_{CP} e : \tau \langle \mu_1 \rangle \alpha \langle \mu_2 \rangle \beta$, *then for any* $4D_{fun}$ *type* γ, $\Gamma \vdash_{4Dfun} e : (\tau, \gamma)^{\rightarrow^{\tau}_{A'}} \langle (\mu_1, \gamma)^{\rightarrow^{\tau}_{A'}}, ((\alpha, \gamma)^{\rightarrow^{\tau}_{A'}} \rightarrow \gamma) \rangle \gamma \langle (\mu_2, \gamma)^{\rightarrow^{\tau}_{A'}}, ((\beta, \gamma)^{\rightarrow^{\tau}_{A'}} \rightarrow \gamma) \rangle \gamma$.
(2) If $\Gamma \vdash_{4Dfun} e : (\tau, \gamma)^{\rightarrow^{\tau}_{A'}} \langle (\mu_1, \gamma)^{\rightarrow^{\tau}_{A'}}, ((\alpha, \gamma)^{\rightarrow^{\tau}_{A'}} \rightarrow \gamma) \rangle \gamma \langle (\mu_2, \gamma)^{\rightarrow^{\tau}_{A'}}, ((\beta, \gamma)^{\rightarrow^{\tau}_{A'}} \rightarrow \gamma) \rangle \gamma$ *for some* $4D_{fun}$ *type* γ, *then* $\Gamma \vdash_{CP} e : \tau \langle \mu_1 \rangle \alpha \langle \mu_2 \rangle \beta$. □

该定理的证明通过对类型推导进行归纳法得出。定理的第一部分已在 Agda 中形式化。第二部分要求项 e 具有某个类型，该类型是 $(\cdot\ \gamma)^{\rightarrow^{\tau}{}_{A'}}$ 对某个 γ 的像。我们不能简单地写成 $\Gamma \vdash_{4Dfun} e\colon\ \tau \mathrm{h} \mu_1,\ \alpha \to \gamma \mathrm{i}\ \gamma \mathrm{h} \mu_2,\ \beta \to \gamma \mathrm{i}\, \gamma$，因为出现在 τ、μ_1、α、μ_2 和 β 中的所有元续延都必须一致地使用 γ。目前，我们还不知道如何在 Agda 中强制执行这一性质。

至于 $4D_{fun}$ 与 4D 之间的关系，我们可以在此应用第 5.1 节 C 部分中的相同讨论。

定理5.5 ($4D_{fun}$ 与4D之间的可定型性)。*()*
1 If $\Gamma \vdash_{4Dfun} e : \tau\ \langle \mu_1, \sigma_1 \rangle\ \alpha\ \langle \mu_2, \sigma_2 \rangle\ \beta$, *th*
对于某个 τ'、μ_1', $\sigma'1$, α', $\mu_2'\sigma_2'$ 和 β'，有$\Gamma \vdash_{4D} e\colon\ \tau\ ' h\mu_1'$, $\sigma'1i\ \alpha' h\mu_2'$, $\sigma'2i\ \beta'$。*(2)* 如果 $\Gamma \vdash_{4D} e\colon\ \tau h\mu_1$, $\sigma_1 i\, \alpha\, h\mu_2$, $\sigma_2 i\ \beta$，那么对于某个 τ'、μ_1', $\sigma'1$, α',μ_2', σ_2' 和 β'，有$\Gamma \vdash_{4Dfun} e\colon\ \tau' h\mu_1'$, $\sigma'1i\ \ \alpha' h\mu_2'$, $\sigma'2i\ \beta'$。□

5.3 shift0/reset0

本节将 Materzok 和 Biernacki 针对 shift0/reset0 的单态类型系统（下文简称 MB）[18] 与 4D 进行比较。MB 与 4D 之间的关系如下图所示。MB 采用一种称为类型注解的不同记法来表示续延和元续延，其语法将在以下段落中介绍。为了关联 MB 和 4D，我们引入 4D′，它与 4D 相同，但不包含轨迹。

$$\mathrm{MB} \underset{\textit{Type annotation}}{\overset{\textit{Meta continuation}}{\rightleftarrows}} 4D' \underset{\textit{Remove trail}}{\overset{\textit{Add trail}}{\rightleftarrows}} 4D$$

我们首先介绍 MB 的演算 λ_{S_0}，然后说明项的可定型性在 MB 与 4D′ 之间以及在 4D′ 与 4D 之间得以保持。

下面是 λ_{S_0} 的语法：一种带有 shift0/reset0 的简单类型 λ 演算。

$$\begin{array}{rrcl} \text{Value} & v & := & n \mid x \mid \lambda x.\, e \\ \text{Term} & e & := & v \mid e_1\, e_2 \mid \mathcal{S}_0 k.\, e \mid \langle e \rangle \\ \text{Type} & \tau & := & \mathbb{N} \mid \tau \xrightarrow{\sigma} \tau' \\ \text{Type Annotation} & \sigma & := & \epsilon \mid [\tau\, \sigma]\, \tau'\, \sigma' \end{array}$$

这种语言的特点是其类型和类型注解。类型注解表明项的周围上下文是否为空 ϵ。类型判断写作 $\Gamma \vdash e$: $\tau_1'\, [\tau_1\, \sigma_1]\, \tau_2' \ldots [\tau_n\, \sigma_n]\, \tau$，其含义是："在类型环境 Γ下，MB 项 e 在类型为 $\tau_1' \xrightarrow{\sigma_1} \tau_1, \ldots, \tau_n' \xrightarrow{\sigma_n} \tau_n$的上下文中被求值为类型为 τ的值。"这种类型注解可以表示任意深度的上下文，即续延和元续延。例如，$\Gamma \vdash e$: $\tau_1\, [\tau_2\ [\tau_4\, \sigma_4]\, \tau_5\, \sigma_5]\, \tau_3\, \sigma_3$ 表示"e 在类型为 $\tau_1 \xrightarrow{[\tau_4\, \sigma_4]\, \tau_5\, \sigma_5} \tau_2$ 的续延中求值为值 τ_3。"如果注解 σ_4 和 σ_5 均为 ϵ，则相应的类型对应于类型为 $\tau_1 \to (\tau_2 \to \tau_4) \to \tau_5$的元续延。

MB 类型与 4D′ 类型之间的精确对应关系将在图 11 中展示。

图 10 展示了 λ_{S_0} 的类型系统。前五条规则（从 (MB-Var) 到 (MB-Reset0)）源自先前的研究 [18]。然而，我们没有使用原始类型系统，而是将 (MB-Abs) 和 (MB-Shift0) 分别替换为 (MB-Abs-Ext) 和 (MB-Shift0-Ext，这些规则也展示在图 10 中。规则 (MB-Abs-Ext) 是将 (σ) 中的 (MB-Abs) 替换为 $[\tau_3\, \sigma_3]\, \tau_4\, \sigma_4$ 得到的。类似地，规则 (MB-Shift0-Ext) 是通过在 (σ_1 和 σ_2) 中实例化 (MB-Shift0)，并使用非空注释得到的。

通过这种替换，项的可定型性不会改变：使用 (MB-Abs) 和 (MB-Shift0) 可以定型的项集合，与使用 (MB-Abs-Ext) 和 (MB-Shift0-Ext) 可以定型的项集合相同，因为图 10 中的原始规则都不要求注释为 ϵ[6]。

现在我们准备继续讨论图 11 中 MB 与 4D′ 之间的类型层面转换。转换 $\leftharpoondown^{\tau}{}_{D'}$ 和 $\leftharpoondown^{\sigma}{}_{D'}$ 将类型从 4D′ 转换为 MB。在这两个翻译中，$\leftharpoondown^{\sigma}{}_{D'}$ 接收一个 4D′ 元续延和 4D′ 答案类型，然后返回一个 MB 类型和一个 MB 注释。另一对转换 $\rightharpoonup^{\tau}{}_{D'}$ 和 $\rightharpoonup^{\sigma}{}_{D'}$分别是 $\leftharpoondown^{\tau}{}_{D'}$ 和 $\leftharpoondown^{\sigma}{}_{D'}$ 的逆变换，并满足以下性质。

引理 5.6。对于任意 λ_{S_0} 类型 τ_1，$(\tau_1{}^{\rightarrow^{\tau}{}_{D'}})^{\leftarrow^{\tau}{}_{D'}} \equiv \tau_1$。此外，对于任意 λ_{S_0} 注释 $[\tau_2\, \sigma_2]\, \tau_3\, \sigma_3$，
$(((\tau_2, \sigma_2)^{\rightarrow^{\sigma}_{D'}})^{\leftarrow^{\sigma}_{D'}}, ((\tau_3, \sigma_3)^{\rightarrow^{\sigma}_{D'}})^{\leftarrow^{\sigma}_{D'}}) \equiv ((\tau_2, \sigma_2), (\tau_3, \sigma_3))$.
□

引理 5.7。对于任意 λ_D 类型 τ，$(\tau^{\leftarrow^{\tau}{}_{D'}})^{\rightarrow^{\tau}{}_{D'}} \equiv \tau$。此外，对于任意$\lambda_D$ 元续延类型 σ_α 和答案类型 α，
$((\sigma_\alpha, \alpha)^{\leftarrow^{\sigma}_{D'}})^{\rightarrow^{\sigma}_{D'}} \equiv (\sigma_\alpha, \alpha)$. □

根据引理5.6 和 5.7，MB 表达式也可以在 4D′ 中进行类型推导，反之亦然（定理5.8）。这些性质的证明通过对类型推导进行归纳法得出，并已在 Agda 中形式化。

定理 5.8 MB 与 4D′ 之间的可类型化性。)
(1) Suppose that $\Gamma \vdash_{MB} e : \tau_1\, [\tau_2\, \sigma_2]\, \tau_3\, \sigma_3$. *If* $\tau_1{}^{\rightarrow^{\tau}_{D'}} = \tau$ *and* $((\tau_2, \sigma_2)^{\rightarrow^{\sigma}_{D'}}, (\tau_3, \sigma_3)^{\rightarrow^{\sigma}_{D'}}) = ((\sigma_\alpha, \alpha), (\sigma_\beta, \beta))$ *holds, then* $\Gamma^{\rightarrow^{\Gamma}_{D'}} \vdash_{4D'} e : \tau\, \langle \sigma_\alpha \rangle\, \alpha\, \langle \sigma_\beta \rangle\, \beta$.
(2) Suppose that $\Gamma' \vdash_{4D'} e' : \tau'\, \langle \sigma_\alpha' \rangle\, \alpha'\, \langle \sigma_\beta' \rangle\, \beta'$. *If* $\tau'^{\leftarrow^{\tau}_{D'}} = \tau_1'$ *and* $((\sigma_\alpha', \alpha')^{\leftarrow^{\sigma}_{D'}}, (\sigma_\beta', \beta')^{\leftarrow^{\sigma}_{D'}}) = ((\tau_2', \sigma_2'), (\tau_3', \sigma_3'))$ *holds, then* $\Gamma'^{\leftarrow^{\Gamma}_{D'}} \vdash_{MB} e' : \tau_1'\, [\tau_2'\, \sigma_2']\, \tau_3'\, \sigma_3'$. □

最后，我们展示 4D′ 与 4D之间的关系。4D′类似于 4D，但不含轨迹。图 4D′ 与4D 之间的类型层面关系如图 12所示。从 4D′ 到 4D以及从 4D 到 4D′ 的每种转换都定义为 $\rightharpoonup^{*}_{D}$

[6]我们认为，函数和 shift0 的主体的类型注解永远不会为空，但我们尚未证明这一点。

$$\frac{}{\Gamma,\, x : \tau_x \vdash x : \tau_x\ [\tau\ \sigma]\ \tau\ \sigma}\ \text{(MB-Var)} \qquad \frac{\Gamma,\, x : \tau_1 \vdash e : \tau_2\ \sigma}{\Gamma \vdash \lambda x.\, e : \tau_1 \xrightarrow{\sigma} \tau_2\ [\tau\ \sigma']\ \tau\ \sigma'}\ \text{(MB-Abs)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \xrightarrow{[\tau_1'\ \sigma_1]\ \tau_3'\ \sigma_3} \tau_2\ [\tau_4'\ \sigma_4]\ \tau_2'\ \sigma_2 \quad \Gamma \vdash e_2 : \tau_1\ [\tau_3'\ \sigma_3]\ \tau_4'\ \sigma_4}{\Gamma \vdash e_1\ e_2 : \tau_2\ [\tau_1'\ \sigma_1]\ \tau_2'\ \sigma_2}\ \text{(MB-App)}$$

$$\frac{\Gamma,\, k : \tau_1 \xrightarrow{\sigma_1} \tau_2 \vdash e : \tau_3\ \sigma_2}{\Gamma \vdash \mathcal{S}_0 k.\, e : \tau_1\ [\tau_2\ \sigma_1]\ \tau_3\ \sigma_2}\ \text{(MB-Shift0)} \qquad \frac{\Gamma \vdash e : \tau'\ [\tau'\ [\tau''\ \sigma'']\ \tau''\ \sigma'']\ \tau\ \sigma}{\Gamma \vdash \langle e \rangle : \tau\ \sigma}\ \text{(MB-Reset0)}$$

$$\frac{\Gamma,\, x : \tau_1 \vdash e : \tau_2\ [\tau_3\ \sigma_3]\ \tau_4\ \sigma_4}{\Gamma \vdash \lambda x.\, e : \tau_1 \xrightarrow{[\tau_3\ \sigma_3]\ \tau_4\ \sigma_4} \tau_2\ [\tau\ \sigma']\ \tau\ \sigma'}\ \text{(MB-Abs-Ext)}$$

$$\frac{\Gamma,\, k : \tau_1 \xrightarrow{[\tau_4\ \sigma_4]\ \tau_5\ \sigma_5} \tau_2 \vdash e : \tau_3\ [\tau_6\ \sigma_6]\ \tau_7\ \sigma_7}{\Gamma \vdash \mathcal{S}_0 k.\, e : \tau_1\ [\tau_2\ [\tau_4\ \sigma_4]\ \tau_5\ \sigma_5]\ \tau_3\ [\tau_6\ \sigma_6]\ \tau_7\ \sigma_7}\ \text{(MB-Shift0-Ext)}$$

图 10. MB类型系统 [18] 及扩展规则

$$\begin{aligned}
\mathbb{N}^{\rightarrow^{\tau}_{\mathrm{D}'}} &= \mathbb{N} \\
(\tau_1 \xrightarrow{[\tau_3\ \sigma_3]\ \tau_4\ \sigma_4} \tau_2)^{\rightarrow^{\tau}_{\mathrm{D}'}} &= (\tau_1^{\rightarrow^{\tau}_{\mathrm{D}'}}) \to (\tau_2^{\rightarrow^{\tau}_{\mathrm{D}'}})\ \langle\sigma_\alpha\rangle\ \alpha\ \langle\sigma_\beta\rangle\ \beta \\
&\quad \textit{where}\quad (\sigma_\alpha,\, \alpha) = (\tau_3,\, \sigma_3)^{\rightarrow^{\sigma}_{\mathrm{D}'}},\quad (\sigma_\beta,\, \beta) = (\tau_4,\, \sigma_4)^{\rightarrow^{\sigma}_{\mathrm{D}'}} \\
(\tau,\, \epsilon)^{\rightarrow^{\sigma}_{\mathrm{D}'}} &= (\bullet_\sigma,\, \tau^{\rightarrow^{\tau}_{\mathrm{D}'}}) \\
(\tau,\, (\ [\tau_1\ \sigma_1]\ \tau_2\ \sigma_2))^{\rightarrow^{\sigma}_{\mathrm{D}'}} &= (((\tau^{\rightarrow^{\tau}_{\mathrm{D}'}} \to \langle\sigma_3\rangle\ \tau_3) :: \sigma_4),\, \tau_4) \\
&\quad \textit{where}\quad (\sigma_3,\, \tau_3) = (\tau_1,\, \sigma_1)^{\rightarrow^{\sigma}_{\mathrm{D}'}},\quad (\sigma_4,\, \tau_4) = (\tau_2,\, \sigma_2)^{\rightarrow^{\sigma}_{\mathrm{D}'}} \\
\bullet_\Gamma{}^{\rightarrow^{\Gamma}_{\mathrm{D}'}} &= \bullet_\Gamma \\
(x\ :\ \tau,\, \Gamma)^{\rightarrow^{\Gamma}_{\mathrm{D}'}} &= x\ :\ \tau^{\rightarrow^{\tau}_{\mathrm{D}'}},\, \Gamma^{\rightarrow^{\Gamma}_{\mathrm{D}'}}
\end{aligned}$$

$$\begin{aligned}
\mathbb{N}^{\leftarrow^{\tau}_{\mathrm{D}'}} &= \mathbb{N} \\
(\tau_1 \to \tau_2\ \langle\sigma_\alpha\rangle\ \alpha\ \langle\sigma_\beta\rangle\ \beta)^{\leftarrow^{\tau}_{\mathrm{D}'}} &= \tau_1{}^{\leftarrow^{\tau}_{\mathrm{D}'}} \xrightarrow{[\tau_3\ \sigma_3]\ \tau_4\ \sigma_4} \tau_2{}^{\leftarrow^{\tau}_{\mathrm{D}'}} \\
&\quad \textit{where}\quad (\tau_3,\, \sigma_3) = (\sigma_\alpha,\, \alpha)^{\leftarrow^{\sigma}_{\mathrm{D}'}},\quad (\tau_4,\, \sigma_4) = (\sigma_\beta,\, \beta)^{\leftarrow^{\sigma}_{\mathrm{D}'}} \\
(\bullet_\sigma,\, \tau)^{\leftarrow^{\sigma}_{\mathrm{D}'}} &= (\tau^{\leftarrow^{\tau}_{\mathrm{D}'}},\, \epsilon) \\
(((\tau_1 \to \langle\sigma_1\rangle\ \tau_2) :: \sigma_2),\, \tau)^{\leftarrow^{\sigma}_{\mathrm{D}'}} &= (\tau_1{}^{\leftarrow^{\tau}_{\mathrm{D}'}},\ [\tau_3\ \sigma_3]\ \tau_4\ \sigma_4) \\
&\quad \textit{where}\quad (\tau_3,\, \sigma_3) = (\sigma_1,\, \tau_2)^{\leftarrow^{\sigma}_{\mathrm{D}'}},\quad (\tau_4,\, \sigma_4) = (\sigma_2,\, \tau)^{\leftarrow^{\sigma}_{\mathrm{D}'}} \\
\bullet_\Gamma{}^{\leftarrow^{\Gamma}_{\mathrm{D}'}} &= \bullet_\Gamma \\
(x\ :\ \tau,\, \Gamma)^{\leftarrow^{\Gamma}_{\mathrm{D}'}} &= x\ :\ \tau^{\leftarrow^{\tau}_{\mathrm{D}'}},\, \Gamma^{\leftarrow^{\Gamma}_{\mathrm{D}'}}
\end{aligned}$$

图 11. MB 与 4D′ 之间的类型级翻译

以及 $\leftarrow^{*}_{\mathrm{D}}$，其思路分别类似于第 5.1节 B 部分中的做法。我们只需为 4D′ 类型添加空轨迹，或从4D 类型中移除轨迹。基于这些转换，我们证明 4D′ 项在 4D中可类型化，反之亦然。下面的定理通过对类型推导进行归纳法证明，并已在 Agda 中形式化。

定理 5.9 （4D′ 与 4D 之间的可定型性）。
(1) 若 $\Gamma \vdash_{4D'} e :\ \tau h\sigma_\alpha i\ \alpha h^{\sigma}\beta i\ \beta$，则 $\Gamma^{\rightarrow^{\Gamma}_{D}} \vdash_{4D} e :\ \tau^{\rightarrow^{\tau}_{D}} h^{\bullet}\mu,\ \sigma_\alpha{}^{\rightarrow^{\sigma}_{D}} i\ \alpha^{\rightarrow^{\tau}_{D}} h^{\bullet}\mu,\ \sigma_\beta{}^{\rightarrow^{\tau}_{D}} i\ \beta^{\rightarrow^{\tau}_{D}}$。*(2)* 若 $\Gamma \vdash_{4D} e :\ \tau h\mu_\alpha,\ \sigma_\alpha i\ \alpha h\mu_\beta,\ \sigma\beta i\ \beta$，则 $\Gamma^{\leftarrow^{\Gamma}_{D}} \vdash_{4D'} e : \tau^{\leftarrow^{\tau}_{D}} h\sigma_\alpha{}^{\leftarrow^{\sigma}_{D}}\ i\ \alpha^{\leftarrow^{\tau}_{D}} h\sigma_\beta{}^{\leftarrow^{\sigma}_{D}} i\ \beta^{\leftarrow^{\tau}_{D}}$。 □

6 相关工作

分隔延续的类型判定。 已经有许多针对分隔控制算子的类型系统。作为 Danvy 和 Filinski 的 shift/reset[8], 类型系统的扩展，Asai 和 Kameyama [2] 提出了一种支持 let-多态的 shift/reset 类型系统。Materzok 和 Biernacki 针对 shift0/reset0 [18] 提出的类型系统支持子类型，并且

$$
\begin{array}{rcl}
\mathbb{N}^{\rightarrow^{\tau}_{\mathrm{D}}} & = & \mathbb{N} \\
(\tau_2 \to \tau_1 \langle \sigma_1 \rangle \, \alpha \, \langle \sigma_2 \rangle \, \beta)^{\rightarrow^{\tau}_{\mathrm{D}}} & = & {\tau_2}^{\rightarrow^{\tau}_{\mathrm{D}}} \to {\tau_1}^{\rightarrow^{\tau}_{\mathrm{D}}} \langle \bullet_{\mu}, {\sigma_1}^{\rightarrow^{\sigma}_{\mathrm{D}}} \rangle \, \alpha^{\rightarrow^{\tau}_{\mathrm{D}}} \langle \bullet_{\mu}, {\sigma_2}^{\rightarrow^{\sigma}_{\mathrm{D}}} \rangle \, \beta^{\rightarrow^{\tau}_{\mathrm{D}}} \\
{\bullet_{\sigma}}^{\rightarrow^{\sigma}_{\mathrm{D}}} & = & \bullet_{\sigma} \\
(\tau_1 \to \langle \sigma_1 \rangle \, \tau_2 :: \sigma_2)^{\rightarrow^{\sigma}_{\mathrm{D}}} & = & ((\tau_1^{\rightarrow^{\tau}_{\mathrm{D}}} \to \langle \bullet_{\mu}, {\sigma_1}^{\rightarrow^{\sigma}_{\mathrm{D}}} \rangle \, {\tau_2}^{\rightarrow^{\tau}_{\mathrm{D}}}) \times \bullet_{\mu}) :: {\sigma_2}^{\rightarrow^{\sigma}_{\mathrm{D}}} \\
{\bullet_{\Gamma}}^{\rightarrow^{\Gamma}_{\mathrm{D}}} & = & \bullet_{\Gamma} \\
x : \tau, \Gamma^{\rightarrow^{\Gamma}_{\mathrm{D}}} & = & x : \tau^{\rightarrow^{\tau}_{\mathrm{D}}}, \Gamma^{\rightarrow^{\Gamma}_{\mathrm{D}}} \\
 & & \\
\mathbb{N}^{\leftarrow^{\tau}_{\mathrm{D}}} & = & \mathbb{N} \\
(\tau_1 \to \tau_2 \langle \mu_1, \sigma_1 \rangle \, \alpha \, \langle \mu_2, \sigma_2 \rangle \, \beta)^{\leftarrow^{\tau}_{\mathrm{D}}} & = & {\tau_1}^{\leftarrow^{\tau}_{\mathrm{D}}} \to {\tau_2}^{\leftarrow^{\tau}_{\mathrm{D}}} \langle {\sigma_1}^{\leftarrow^{\sigma}_{\mathrm{D}}} \rangle \, \alpha^{\leftarrow^{\tau}_{\mathrm{D}}} \langle {\sigma_2}^{\leftarrow^{\sigma}_{\mathrm{D}}} \rangle \, \beta^{\leftarrow^{\tau}_{\mathrm{D}}} \\
{\bullet_{\sigma}}^{\leftarrow^{\sigma}_{\mathrm{D}}} & = & \bullet_{\sigma} \\
(((\tau_1 \to \langle \mu_1, \sigma_1 \rangle \, \tau_2) \times \mu_2) :: \sigma_2)^{\leftarrow^{\sigma}_{\mathrm{D}}} & = & {\tau_1}^{\leftarrow^{\tau}_{\mathrm{D}}} \to \langle {\sigma_1}^{\leftarrow^{\sigma}_{\mathrm{D}}} \rangle \, {\tau_2}^{\leftarrow^{\tau}_{\mathrm{D}}} :: {\sigma_2}^{\leftarrow^{\sigma}_{\mathrm{D}}} \\
{\bullet_{\Gamma}}^{\leftarrow^{\Gamma}_{\mathrm{D}}} & = & \bullet_{\Gamma} \\
x : \tau, \Gamma^{\leftarrow^{\Gamma}_{\mathrm{D}}} & = & x : \tau^{\leftarrow^{\tau}_{\mathrm{D}}}, \Gamma^{\leftarrow^{\Gamma}_{\mathrm{D}}}
\end{array}
$$

Figure 12.4D′ an 之间的类型级翻译 d 4D

使我们能够在不同上下文中使用捕获的延续。Cong 等人[7] 以及 Kameyama 和 Yonezawa [14] 展示了适用于 `control/prompt` 的不同类型系统。（有关他们的比较，请参见 Cong 等人 [7]。）与以往的工作不同，本文展示了一种适用于全部四种分隔控制算子的类型系统，尤其是 `control0`。另一方面，我们的类型系统是单态的，不支持任何形式的多态。

Dybvig、Peyton Jones 和 Sabry [10]构建了一个单子框架，用于为分隔延续指定类型。他们提出了一组新算子，作为构件来模拟现有算子，例如 shift（他们论文中的 ${}^{+}\mathrm{F}^{+}$）、control（${}^{+}\mathrm{F}^{-}$）、shift0（${}^{-}\mathrm{F}^{+}$）和 control0（${}^{-}\mathrm{F}^{-}$）。他们展示了一个 Haskell 类型系统，但它并非针对四种分隔控制算子本身，而是针对他们引入的基本算子集。基于这项工作，Kiselyov [16] 在 OCaml 中实现了四种分隔控制算子（通过这组基本算子实现），但未将答案类型纳入考虑。

分隔控制算子的续延传递风格语义。 第 3 节中四种控制算子的 CPS 解释器由 Shan [24]提出。他们展示了 `shift/reset` 如何在无类型环境中模拟其他控制算子。Biernacki、Danvy 和 Millikin [4] 提出了一种与 Shan 的 CPS 转换等价的 CPS 转换。该 CPS 转换源自定义机器的去函数化版本。

代数效应与处理器。 除了受限控制运算符之外，代数效应与处理器 [21, 22] 也是处理续延的另一种方式。shift0 与深层效应处理器之间的关系，已在无类型环境 [12]和有类型环境 [6,20]下得到确立。

7 结论与未来工作

我们从相应的 CPS 解释器中提出了一个针对四种分隔控制算子——shift、control、shift0 和 control0——的单态类型系统。据我们所知，我们首次提出了允许答案类型修改的 control0 类型规则，并以统一表示法形式化了所有这些算子的类型规则。此外，关于以往研究之间的关系，我们证明了我们的类型系统涵盖了现有的单态类型系统。

拓展这项工作的一个方向是扩大类型系统的覆盖范围，例如纳入多态和答案类型多态。另一种方法是在有类型环境下，理解使用分隔控制算子的代数效应处理器之间的关系。众所周知，shift0 和 control0 分别与深层和浅层效应处理器密切相关，因此本文将为探索它们之间的关系奠定坚实基础。

致谢

感谢 Youyou Cong 和匿名审稿人提出的宝贵意见与反馈。本研究部分获得了 JSPS KAKENHI 项目资助，项目编号为 JP22H03563。

$4D_{fun}$ 与 4D 之间的关系

本节将给出$4D_{fun}$ 与4D之间可定型性的完整证明，该内容已在第5.3节 C 部分的5中说明。

$4D_{fun}$ 和 4D 的源语言都由一个简单类型 λ 演算扩展而成，其中包含数字以及 shift /reset。唯一的区别在于元连续体的类型：在 $4D_{fun}$ 中，该类型要么为空（$\bullet_{\sigma}$），要么是函数类型（$\tau_1 \to \tau_2$）；而在 4D 中，该类型要么为空（$\bullet_{\sigma}$），要么是积类型（$((\tau_1 \to \mathrm{h}\mu_1, \sigma_1\mathrm{i}\, \tau_2) \times \mu_2) :: \sigma_2$）。类型规则

$$\boxed{\Gamma \vdash_{\text{4Dfun}} e : \tau \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\beta \rangle \beta}$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash_{\text{4Dfun}} x : \tau \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\alpha \rangle \alpha} \text{ (4Dfun-Var)} \qquad \frac{}{\Gamma \vdash_{\text{4Dfun}} n : \mathbb{N} \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\alpha \rangle \alpha} \text{ (4Dfun-Num)}$$

$$\frac{\Gamma, x : \tau_1 \vdash_{\text{4Dfun}} e : \tau_2 \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\beta \rangle \beta}{\Gamma \vdash_{\text{4Dfun}} (\lambda x.e) : (\tau_1 \to \tau_2 \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\beta \rangle \beta) \langle \bullet_\mu, \sigma_\gamma \rangle \gamma \langle \bullet_\mu, \sigma_\gamma \rangle \gamma} \text{ (4Dfun-Lam)}$$

$$\frac{\Gamma \vdash_{\text{4Dfun}} e_1 : (\tau_1 \to \tau_2 \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\beta \rangle \beta) \langle \bullet_\mu, \sigma_\gamma \rangle \gamma \langle \bullet_\mu, \sigma_\delta \rangle \delta \quad \Gamma \vdash_{\text{4Dfun}} e_2 : \tau_1 \langle \bullet_\mu, \sigma_\beta \rangle \beta \langle \bullet_\mu, \sigma_\gamma \rangle \gamma}{\Gamma \vdash_{\text{4Dfun}} (e_1\, e_2) : \tau_2 \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\delta \rangle \delta} \text{ (4Dfun-App)}$$

$$\frac{\begin{array}{c}\text{id-cont-type}(\gamma, \bullet_\mu, \sigma_{id}, \gamma') \\ \Gamma, k : \tau \to \tau_1 \langle \bullet_\mu, \sigma_1 \rangle \tau_2 \langle \bullet_\mu, \sigma_1 \rangle \alpha \vdash_{\text{4Dfun}} e : \gamma \langle \bullet_\mu, \sigma_{id} \rangle \gamma' \langle \bullet_\mu, \sigma_\beta \rangle \beta\end{array}}{\Gamma \vdash_{\text{4Dfun}} \mathcal{S}k.e : \tau \langle \bullet_\mu, (\tau_1 \to \tau_2) \rangle \alpha \langle \bullet_\mu, \sigma_\beta \rangle \beta} \text{ (4Dfun-Shift)}$$

$$\frac{\begin{array}{c}\text{id-cont-type}(\gamma, \bullet_\mu, \sigma_{id}, \gamma') \\ \Gamma \vdash_{\text{4Dfun}} e : \gamma \langle \bullet_\mu, \sigma_{id} \rangle \gamma' \langle \bullet_\mu, (\tau \to \alpha) \rangle \beta\end{array}}{\Gamma \vdash_{\text{4Dfun}} \langle e \rangle : \tau \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\alpha \rangle \beta} \text{ (4Dfun-Prompt0)}$$

图13. $4D_{fun}$ shift/reset 的类型系统

$$\boxed{\Gamma \vdash e : \tau \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\beta \rangle \beta}$$

$$\frac{x : \tau \in \Gamma}{\Gamma \vdash x : \tau \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\alpha \rangle \alpha} \text{ (TVar')} \qquad \frac{}{\Gamma \vdash n : \mathbb{N} \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\alpha \rangle \alpha} \text{ (TNum')}$$

$$\frac{\Gamma, x : \tau_1 \vdash e : \tau_2 \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\beta \rangle \beta}{\Gamma \vdash (\lambda x.e) : (\tau_1 \to \tau_2 \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\beta \rangle \beta) \langle \bullet_\mu, \sigma_\gamma \rangle \gamma \langle \bullet_\mu, \sigma_\gamma \rangle \gamma} \text{ (TLam')}$$

$$\frac{\Gamma \vdash e_1 : (\tau_1 \to \tau_2 \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\beta \rangle \beta) \langle \bullet_\mu, \sigma_\gamma \rangle \gamma \langle \bullet_\mu, \sigma_\delta \rangle \delta \quad \Gamma \vdash e_2 : \tau_1 \langle \bullet_\mu, \sigma_\beta \rangle \beta \langle \bullet_\mu, \sigma_\gamma \rangle \gamma}{\Gamma \vdash (e_1\, e_2) : \tau_2 \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\delta \rangle \delta} \text{ (TApp')}$$

$$\frac{\begin{array}{c}\text{id-cont-type}(\gamma, \bullet_\mu, \sigma_{id}, \gamma') \\ \Gamma, k : \tau \to \tau_1 \langle \bullet_\mu, \sigma_1 \rangle \tau_2 \langle \bullet_\mu, \sigma_1 \rangle \alpha \vdash e : \gamma \langle \bullet_\mu, \sigma_{id} \rangle \gamma' \langle \bullet_\mu, \sigma_\beta \rangle \beta\end{array}}{\Gamma \vdash \mathcal{S}k.e : \tau \langle \bullet_\mu, (((\tau_1 \to \langle \bullet_\mu, \sigma_1 \rangle \tau_2) \times \bullet_\mu) :: \sigma_1) \rangle \alpha \langle \bullet_\mu, \sigma_\beta \rangle \beta} \text{ (TShift')}$$

$$\frac{\begin{array}{c}\text{id-cont-type}(\gamma, \bullet_\mu, \sigma_{id}, \gamma') \\ \Gamma \vdash e : \gamma \langle \bullet_\mu, \sigma_{id} \rangle \gamma' \langle \bullet_\mu, (((\tau \to \langle \bullet_\mu, \sigma_\alpha \rangle \alpha) \times \bullet_\mu) :: \sigma_\alpha) \rangle \beta\end{array}}{\Gamma \vdash \langle e \rangle : \tau \langle \bullet_\mu, \sigma_\alpha \rangle \alpha \langle \bullet_\mu, \sigma_\alpha \rangle \beta} \text{ (TPrompt0')}$$

图14. 4D shift/reset 的类型系统

的 $4D_{fun}$ 和 4D分别见图 13 和 14。两个类型系统都在 id-cont-type规则中使用`shift`和 `prompt0`，其定义与图 5中的相同。

A.1 从 $4D_{fun}$ 到 4D 的转换

本节对应于定理5.3 (1)。我们的方法是对$4D_{fun}$的类型判断进行结构归纳，其描述如下：$\Gamma \vdash_{\text{4Dfun}} e : \tau \langle \bullet_\mu, \sigma_1 \rangle \alpha \langle \bullet_\mu, \sigma_2 \rangle \beta$。

如果 e最后使用的类型规则是 (4Dfun-Var)、(4Dfun-Num)、(4Dfun-Lam) 或 (4Dfun-App) 之一，那么显然，如果 e 在 $4D_{fun}$中可类型化，它在 4D 中也可类型化，因为 $4D_{fun}$ 的每条求值规则和类型规则都与 4D 中的相同。

接下来，如果最后使用的类型规则是 (4Dfun-Shift)，那么 e 变为 $\mathcal{S}k.e$，其求值规则如下。

$$\begin{array}{l}\mathcal{E}_{\text{4Dfun}}[[\mathcal{S}k.e]]\,\rho\,\kappa = \lambda t^{\mu^*_\beta}.\lambda m^{\sigma^*_\beta}. \\ \quad \mathcal{E}_{\text{4Dfun}}[[e]]\,\rho\,[\lambda v^{\tau^*}.\lambda\kappa'^{(\tau_1 \to \langle \bullet_\mu, \sigma_1 \rangle \tau_2)^*}.\lambda t'^{\bullet^*_\mu}.\lambda m'^{\sigma^*_1}. \\ \qquad \kappa^{(\tau \to \langle \mu_\beta, \tau_1 \to \tau_2 \rangle \alpha)^*}\, v^{\tau^*}\, t^{\mu^*_\beta} \\ \qquad\quad (\lambda v'^{\tau^*_1}.\kappa'^{(\tau_1 \to \langle \bullet_\mu, \sigma_1 \rangle \tau_2)^*}\, v'^{\tau^*_1}\, t'^{\bullet^*_\mu}\, m'^{\sigma^*_1})^{(\tau_1 \to \tau_2)^*}/k] \\ \qquad\qquad k''^{(\gamma \to \langle \mu_{id}, \sigma_{id} \rangle \gamma')^*}_{id}\, ()^{\bullet^*_\mu}\, m^{\sigma^*_\beta}\end{array}$$

在第一行，解释器接收类型为 $(\tau \to \text{h}^{\mu_\beta}, \tau_1 \to \tau_2\text{i}\ \alpha)^*$ 的 κ。此外，在第三行和第四行，κ 接收类型为 $(\tau_1 \to \tau_2)^*$ 的元续延 $(\lambda v, \kappa'\, v'\, t, m')$。由于元续延的形式是函数，规则（4Dfun-Shift）中并未明确显示 t' 和 m' 的类型。若将元续延转换为

去函数化形式，求值规则如下。

$$\begin{aligned}&\mathcal{E}[[\mathcal{S}k.\,e]]\,\rho\,\kappa \;=\; \lambda t^{\mu^*_\beta}.\,\lambda m^{\sigma^*_\beta}.\\&\quad\mathcal{E}[[e]]\,\rho[\lambda v^{\tau^*}.\,\lambda\kappa'^{(\tau_1\to\langle\bullet_\mu,\,\sigma_1\rangle\,\tau_2)^*}.\,\lambda t'^{\bullet^*_\mu}.\,\lambda m'^{\sigma^*_1}.\\&\qquad\kappa^{(\tau\to\langle\mu_\beta,\,((\tau_1\to\langle\bullet_\mu,\,\sigma_1\rangle\,\tau_2)\times\bullet_\mu)\,::\,\sigma_1\rangle\,\alpha)^*}\,v^{\tau^*}\;t^{\mu^*_\beta}\\&\qquad\quad((\kappa'^{(\tau_1\to\langle\bullet_\mu,\,\sigma_1\rangle\,\tau_2)^*},\,t'^{\bullet^*_\mu})\,::\,m'^{\sigma^*_1})/k]\\&\qquad\quad k_{id}^{(\gamma\to\langle\mu_{id},\,\sigma_{id}\rangle\,\gamma')^*}\;()^{\bullet^*_\mu}\;m^{\sigma^*_\beta}\end{aligned}$$

元续延由相同的参数 κ'、t'和 m'组成，并且它们各自的类型也与前一个求值规则相同。因此，如果 $\mathsf{S}e.k$ 使用 (4Dfun-Shift) 进行类型判定，那么它也可以使用 (TShift′) 进行类型判定。

最后，如果最后使用的类型规则是 (4Dfun-Prompt0)，e 就变为 $\mathsf{h}e\mathsf{i}$。下面给出来自 $\mathrm{4D}_{\mathrm{fun}}$ 和 4D 的两个解释器。

$$\begin{aligned}&\mathcal{E}_{\mathrm{4Dfun}}[[\langle e\rangle]]\,\rho\,\kappa^{(\tau\to\langle\bullet_\mu,\,\sigma_\alpha\rangle\,\alpha)^*} \;=\; \lambda t^{\bullet^*_\mu}.\,\lambda m^{\sigma^*_\alpha}.\\&\quad\mathcal{E}_{\mathrm{4Dfun}}[[e]]\,\rho\,k_{id}^{(\gamma\to\langle\mu_{id},\,\sigma_{id}\rangle\,\gamma')^*}\;()^{\bullet^*_\mu}\\&\qquad(\lambda v^{\tau^*}.\,\kappa^{(\tau\to\langle\bullet_\mu,\,\sigma_\alpha\rangle\,\alpha)^*}\,v^{\tau^*}\;t^{\bullet^*_\mu}\;m^{\sigma^*_\alpha})\\&\mathcal{E}[[\langle e\rangle]]\,\rho\,\kappa^{(\tau\to\langle\bullet_\mu,\,\sigma_\alpha\rangle\,\alpha)^*} \;=\; \lambda t^{\bullet^*_\mu}.\,\lambda m^{\sigma^*_\alpha}.\\&\quad\mathcal{E}[[e]]\,\rho\,k_{id}^{(\gamma\to\langle\mu_{id},\,\sigma_{id}\rangle\,\gamma')^*}\;()^{\bullet^*_\mu}\\&\qquad((\kappa^{(\tau\to\langle\bullet_\mu,\,\sigma_\alpha\rangle\,\alpha)^*},\,t^{\bullet^*_\mu}),\,m^{\sigma^*_\alpha})\end{aligned}$$

这两个解释器的区别在于第三行的元续延形式 ($\lambda v.\,\kappa\,v\,t\,m$)，以及第六行的 $((\kappa,\,t)\;\;m)$。两个元续延具有相同的参数 κ、t 和 m，并且它们各自的类型相同。因此，如果 $\mathsf{h}e\mathsf{i}$ 使用 (4Dfun-Prompt0) 进行类型判定，那么它也可以使用 (TPrompt0′) 进行类型判定。

A.2 从 4D 到 4D 的翻译$_{\mathrm{fun}}$

本节对应于定理 5.3（2），证明策略与上一节相同。我们对 4D 的类型判断进行结构归纳，其描述为 $\Gamma \vdash_{\mathrm{4D}} e : \tau\langle\bullet_\mu,\;\;\sigma_1\rangle\,\alpha\,\langle\bullet_\mu,\;\;\sigma_2\rangle\;\beta$。

从（TVar′）到（TApp′），这是显然的，因为它们的求值规则和类型规则是相同的。

对于（TShift′）和（TPrompt0′）的情况，上一节中的相同论证在此处同样适用。由于两个解释器具有相同的参数，并且每个参数都具有相同的类型，如果 e 在 4D 中具有类型，那么它在 $\mathrm{4D}_{\mathrm{fun}}$ 中也具有类型。

参考文献

[1] T. Altenkirch 和 B. Reus. 1999. 使用广义归纳类型表示 λ 项的单子表示。载于计算机科学逻辑国际研讨会。Springer，453–468。[2] K. Asai 和 Y. Kameyama. 2007. 多态分隔延续。第 *5* 届亚洲编程语言与系统会议论文集（*APLAS'07*）（2007），239–254。[3] A. Bauer 和 M. Pretnar. 2013. 代数效应与处理器的效应系统。载于计算机科学中的代数与余代数国际会议。Springer，1–16。[4] D. Biernacki、O. Danvy 和 K. Millikin. 2015. 动态分隔续延的动态续延传递风格。*ACM*编程语言与系统汇刊（*TOPLAS*）38，1（2015），1–25。

[5] A. Chlipala. 2008. 面向机械化语义的参数化高阶抽象语法。载于 ACM SIGPLAN 函数式编程国际会议论文集（ICFP 2008）。143–156。[6] Y. Cong 和 K. Asai. 2022. 通过分隔控制算子理解代数效应处理器。载于《函数式编程趋势》（TFP'22）。[7] Y. Cong、C. Ishio、K. Honda 和 K. Asai. 2021. 类型化调用上下文的函数式抽象。载于第 6 届计算与演绎形式结构国际会议（FSCD 2021）。12:1– 12 18 10 4230 2021 12 : 。[8] 1989 https://doi.org/ 。/LIPIcs.FSCD. 。O. Danvy 和 A. Filinski. 。类型化上下文的函数式抽象。BRICS 89/12（1989）。[9] O. Danvy 和 A. Filinski. 1990. 抽象 control。ACM LISP 与函数式编程会议论文集（LFP'90）（1990），151–160。[10] R. K. Dybvig、S. Peyton Jones 和 A. Sabry. 2007. 分隔延续的单子框架。《函数式编程杂志》17，6（2007），687–730。[11] M. Felleisen. 1988. 一等 prompt 的理论与实践。载于第 15 届 ACM 编程语言原理研讨会论文集（POPL ' *88* 180 190 10 1145 73560 73576 ）。– [12] 2017 。https://doi.org/ 。/ 。Y. Forster 、 O. Kammar 、 S. Lindley 和 M. Pretnar. 。用户定义效应的表达能力：效应处理器、单子反射与分隔 control。ACM 程序设计语言会议录 1，ICFP，文章 13 2017 29 10 1145 3110257 （8 月），[13] 1995 页码。https://doi.org/ 。/ C. A. Gunter、D. Rémy 和 J. G. Riecke. 。类 ML 语言中异常与 control 的一种推广。载于第 7 届函数式编程语言与计算机体系结构国际会议论文集。12–23。[14] Y. Kameyama 和 T. Yonezawa. 2008. 分隔延续的类型化动态控制算子。载于函数式与逻辑编程国际研讨会。Springer，239–254。[15] O. Kiselyov. 2005. 如何移除动态 prompt：静态与动态分隔控制算子。技术报告 611。印第安纳大学（2005）。[16] O. Kiselyov. 2010. OCaml 中的分隔 control，抽象与具体：系统描述。载于函数式与逻辑编程国际研讨会。Springer，304–320。[17] S. Krishnamurthi、P. W. Hopkins、J. McCarthy、P. T. Graunke、G. Pettyjohn 和 M. Felleisen. 2007. PLT Scheme Web 服务器的实现与使用。《高阶与符号计算》20，4（2007），431–460。[18] M. Materzok 和 D. Biernacki. 2011. 分隔延续的子类型。载于 ACM SIGPLAN 函数式编程国际会议论文集（ICFP'11）。81–93。[19] U. Norell. 2007. 迈向一种基于依赖类型理论的实用编程语言。博士论文。查尔姆斯理工大学，瑞典哥德堡 SE-412 96。[20] M. Piróg、P. Polesiuk 和 F. Sieczkowski. 2019. 效应处理器与分隔 control 的类型化等价性。载于第 4 届计算与演绎形式结构国际会议（FSCD 2019）。Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik。[21] G. Plotkin 和 J. Power. 2003. 代数运算与泛型效应。《应用范畴结构》11，1（2003），69–94。[22] G. Plotkin 和 M. Pretnar. 2009. 代数效应的处理器。载于欧洲编程研讨会。Springer，80–94。[23] John Reppy、Claudio Russo 和 Yingqi Xiao. 2009. 并行并发 ML。载于第 14 届 ACM SIGPLAN 函数式编程国际会议论文集（ICFP 2009）。ACM，纽约州纽约市， 257 268 10 1145 1596550 1596588 – 。[24] 2007 https://doi.org/ 。/ 。C. Shan. 。动态分隔 control 的一种静态模拟。《高阶与符号计算》20，4（2007），371–401。

收稿日期：2022年8月12日；录用日期：2022年10月10日