arXiv:2601.08071v1 [cs.LO] 12 Jan 2026

作为中间逻辑和中间语言的 S4 模态序列演算

JEAN CASPAR，巴黎高等师范学院—PSL、INRIA、法国GUILLAUME MUNCH-MACCAGNONI，INRIA、LS2N CNRS、法国

在这篇短文中，我们主张，基于延续的中间语言对应于中间逻辑。中间语言旨在作为编译器中间表示的基础，使其能够表示具有表达力的程序变换，以用于优化和编译，同时保留使程序从一开始就能够被高效编译的性质，例如延续的“可栈化性”。就可证性而言，中间逻辑是介于直觉主义逻辑和经典逻辑之间的逻辑。

基于CPS的中间语言中使用的二等延续对应于经典模态逻辑 S4，但增加了一项限制：蕴含只能返回模态类型。这确实对应于一种中间逻辑，这得益于 Gödel-McKinsey-Tarski 定理；该定理指出，S4 的模态片段具有直觉主义性质。

我们引入 $\mathbf{L}^{\Box}_{\mathrm{pol}}$，这是一种用于 S4 的三种类极化相继式演算，并配有将堆与栈分离的操作机器模型。借助这一模型，我们研究 S4 模态片段的栈化性质。

1 引言

作为中间逻辑的中间语言。通过控制算子与经典公理之间的公式即类型对应，从经典逻辑中的构造性问题出发，对延续进行证明论研究，带来了诸多启示。例如，这让我们对控制算子有了更深入的理解，也促成了更适合对延续计算进行建模的演算，其中包括相继式演算与抽象机器之间的 Curry–Howard 对应。

从历史上看，编译一直是延续概念的起源之一 [Steele Jr.1978]。许多研究将证明论研究的启示应用于编译器的构造 [Downen 等人 2016年；Schuster 等人 2025]。与此同时，关于延续对现代编译器的相关性，也存在更广泛的讨论 [Maurer 等人 2017年；Cong 等人 2019]。

编译器的中间表示中使用的延续概念，与逻辑学家研究的延续概念后来被发现并不相同。虽然（极化的、即求值顺序感知的相继式演算 LJ（直觉主义：结论中是线性的）和 LK（经典：结论中是非线性的）版本）分别对应于线性使用的延续和非线性延续，但中间表示中出现的延续限制，按照柯里–霍华德对应，似乎对应于介于直觉主义相继式演算和经典相继式演算之间的一种中间逻辑。Downen、Maurer、Ariola 和 Peyton Jones [2016]对此作了最清晰的阐述：

> [Sequent Core]所体现的逻辑介于 [LK 和 LJ]之间：有时只能有一个结论，有时则可以有多个结论。此外[……] *Sequent Core* 仍然体现了与 λ-演算相似的纯粹性概念。这表明，直觉主义逻辑还可以有更微妙的变体，介于 *LK* 的自由性和 *LJ* 的纯粹性之间。

然而，用于编译的延续所具有的这种中间特性，一直与证明论处理格格不入，直到 Cong、Osvald、Essertel 和 Rompf [2019],从一项……开始

2026年1月14日。论文被 PEPM 2026 接收，列入“短篇论文”类别。作者联系信息：Jean Caspar，jean.caspar@ens.psl.eu，巴黎高等师范学院—PSL 和 INRIA，法国巴黎；Guillaume Munch-Maccagnoni，guillaume.munch-maccagnoni@inria.fr，INRIA、LS2N CNRS，法国南特。

为复现 Maurer 等人 [2017], 的结果而进行的实验中，他们基于早期关于二等值的研究，提出用简单类型系统来刻画延续的“第二类”特性[Osvald、Essertel、Wu、González-Alayón 和 Rompf 2016]。第一作者在硕士论文 Cong 等人 [2019] 的语言和类型系统确实适合通过模态逻辑 S4 的相继式演算来建模 [Caspar 2025], 为二等值提供了逻辑刻画。

Cong 等人提出的演算 [2019] 因此区分了第二类（受限）值和第一类（不受限）值。对第二类值的限制确保它们不会逃逸出其定义作用域：它们不能作为函数的返回值，也不能被第一类值引用。特别是，引用第二类值的闭包本身也是第二类值。这确保了 [Osvald 等人 2016] 提出的第二类值可以分配在栈上，并且当函数接受一个第一类参数时，可以释放栈。编译器按照Cong 等人 [2019] 提出的方法引入的延续属于第二类，从而确保它们可以采用类似栈的方式分配。尽管如此，从柯里-霍华德视角来看，它们的 CPS 翻译仍呈现出经典特性：延续可以在一定程度上被复制或擦除，从而放宽了直觉逻辑 CPS 模型中对延续严格线性使用的要求。

近来，Schuster、Müller、Ostermann 和 Brachthäuser [2025]通过采用单侧极化经典序列演算作为高效编译的基础，挑战了我们的先入之见——这种演算最初由Munch-Maccagnoni [2009] 提出，作为 Girard 完全对称经典逻辑的一种演算 [Girard 1991；Danos、Joinet 和 Schellinx 1997], 则表达了不受限的*call/cc*运算符。由于它事先不对延续的使用施加限制，诀窍就在于在后端动态地利用线性，通过引用计数重用以线性方式使用的内存单元。正如他们所指出的，“由于续延帧通常以线性方式使用，因此即使我们不维护栈，也能为类似栈的用法生成快速代码。”

我们认为，让中间表示能够识别延续的实际线性特征仍有其优势，这样便能更好地保留这些特性（例如，在用于优化的代码转换过程中），并加以利用（例如，使用硬件栈支持），从而实现高效的编译策略。

作为中间表示的经典 *S4* 序列演算。在以下工作的基础上：Cong、Osvald、Essertel 和 Rompf [2019], 本文报告了第一作者硕士论文中的一些研究成果 [Caspar 2025] 该论文利用（经典）S4 模态序列演算的一种极化版本，研究二等延续；该演算$\mathbf{L}^{\Box}_{\mathrm{pol}}$。其主要成果之一，也是本文短文的研究动机（尽管本文不在此详述），是从模态逻辑的视角分析Cong 等人 [2019]:

> *The CPS translation in Cong et al. [2019] factors into an interpretation of their calculus into* $\mathbf{L}^{\Box}_{\mathrm{pol}}$*, followed by a CPS translation of* $\mathbf{L}^{\Box}_{\mathrm{pol}}$.

这将 $\lambda\mu$-演算的 CPS 翻译通过单侧经典序列演算进行类似分解的工作加以扩展，参见 [Danos 等人 1997；Ogata 2000年；Laurent2002；Munch-Maccagnoni 2013]。在这篇短文中，我们省略$\mathbf{L}^{\Box}_{\mathrm{pol}}$ 的 CPS 翻译，并直接建立其操作语义的性质。

直觉主义 S4 模态逻辑在编程语言领域有广泛的应用，包括分阶段编译 [Davies 和 Pfenning 2001], 上下文类型理论 [Nanevski、Pfenning 和 Pientka 2008],以及信息流 [Miyamoto 和 Igarashi 2004]。

经典 （经典）S4 模态逻辑可被视为将经典逻辑与直觉主义逻辑相融合，这归因于 Gödel-McKinsey-Tarski 定理 [Gödel 2001；McKinsey 和 Tarski 1948], 该定理指出，其模态片段对于直觉主义可证性是完备的。此外，如果我们考虑一个

通过限制到返回类型为模态类型的函数而得到的经典 S4 模态逻辑片段，我们便获得了介于经典逻辑与直觉主义逻辑之间的中间逻辑。实际上，这一限制改变了 S4 的经典特征——模态片段并未改变——但某些受限形式的经典公理仍然是可证的。其直觉是，正则类型属于二等公民（用编程术语来说，它们无法逃离定义它们的作用域），而模态类型则属于一等公民。

通过 *S4* 探讨二等延续的理论。本文介绍了用于模态逻辑 S4 的极化序列演算 $\mathbf{L}^{\Box}_{\text{pol}}$，具有三种极性（□、 +、 −）。我们还介绍了一种类机器语义，其特征与 Cong 等人 [2019]提出的语义相似：我们将内存划分为两部分，一部分是用于存放模态值的普通堆，其中的值只能引用模态变量；另一部分类似栈，每当我们求值一个模态余变量时就会释放。返回非模态类型的值时，栈不会被释放。

接着，如果我们将 S4 限定为返回类型为模态类型的函数，从而完整实现二等限制，就能恢复 Cong 等人 [2019]的栈化性质。结果表明，二等延续的元理论结果可以直接从 S4 的元理论结果中推导出来。

2 系统 $\mathbf{L}^{\Box}_{\text{pol}}$

在本节中，我们介绍一种用于经典 S4 逻辑的极化序列演算，其项语法称为 $\mathbf{L}^{\Box}_{\text{pol}}$，它扩展了 Munch-Maccagnoni [2009] 的极化经典 L 演算。

极化 *L* 演算简述。 该演算的语法由形如 $t \parallel e$ 的命令 c 构成，这些命令由环境 e 中的表达式 t 构成（环境是求值上下文的具体化）。环境可以绑定变量；例如，通常情况下，程序“let $x = t$ in c”可以表示为 $t \parallel \tilde{\mu}x.c$：它先对 t 求值，然后执行 c，其中 t 的值绑定到变量 x。

$\mathbf{L}^{\Box}_{\text{pol}}$中还提供了其他构造，例如对 (t, u)，以及使用环境 $\tilde{\mu}(x, y).c$对这类对进行模式匹配。因此，程序 let $(x, y) = t$in c可表示为 $t \parallel \tilde{\mu}(x, y).c$：它先将 t求值为一个值，然后执行 c，并将 t的值绑定到 (x, y)，前提是该值确实分解为一个对。我们还具有一个单元值 $()$，可以使用 $\tilde{\mu}().c$对其进行匹配；此外还有带有注入 $\iota_i\, t$的和类型，以及匹配构造 $\tilde{\mu}(\iota_1\, x.c \mid \iota_2\, y.c')$。

该类型系统对应于经典逻辑，并遵循针对 call/cc 风格控制算子的公式即类型解释 [Griffin 1989]。当前续延（或在此处的环境）通过构造 $\mu\alpha.c$ 捕获：该表达式捕获当前环境，将其绑定到一个“余变量” α，并继续执行 c。

遵循经典逻辑的线性分析 [Girard 1991；Danos 等人 1997], , 如果 α 和 x 是新鲜的，则表面上不可合流的临界对

$$c \leftarrow \langle \mu\alpha.c \parallel \tilde{\mu}x.c' \rangle \rightarrow c'$$

通过对“正”类型优先采用左侧归约（对应于按值调用求值），对“负”类型优先采用右侧归约（对应于按名调用求值）来解决。绑定子和命令均标注有极性 ε。

我们现在还必须区分哪些项可以被替换，哪些不可以。因此，我们还有另外两种句法类别：值 V 和余值 S，它们分别对应可以被替换的“纯”表达式和环境。$()$ 是一个值，并且我们限制对和注入，使得只有 (V, V) 和 $\iota_i\, V$可以作为值。 $\mu\alpha^{\varepsilon}.c$在 ε为负时是一个值，但在 ε 为正时则不是； $\tilde{\mu}x^{\varepsilon}.c$ 在 ε 为负时是一个余值，但在 ε 为负时则不是。除此之外，之前引入的每个表达式都是一个值，每个环境都是一个余值：变量是值，

而协变量和 $\tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c$是余值。我们有以下归约规则：

$$\langle \mu\alpha^{\varepsilon}.c \parallel^{\varepsilon} S\rangle \to c[S/\alpha]$$
$$\langle V \parallel^{\varepsilon} \tilde{\mu}x^{\varepsilon}.c\rangle \to c[V/x]$$

$\mu\alpha^{\varepsilon}.c \parallel^{\varepsilon} \tilde{\mu}x^{\varepsilon}.c'$ 的规约由极性 ε 唯一确定。由于极性由类型决定，因此如果命令类型正确，两侧的极性就会匹配。

该系统同时支持按值调用和按名调用，因此除了严格对（V，W）之外，我们还有惰性记录 $\mu(\pi_1\ \alpha^{\varepsilon}.c \mid \pi_2\ \beta^{\varepsilon'}.c')$。这表示一个包含两个字段的记录，其计算尚未完成：要访问某个字段，我们可以使用余值 $\pi_i S$，其中 S 表示我们希望在其中使用由字段 i 计算所得值的环境。规约 $\mu(\pi_1\ \alpha_1^{\varepsilon_1}.c_1 \mid \pi_2\ \alpha_2^{\varepsilon_2}.c_2) \parallel^{-} \pi_i S \to c_i[S/\alpha_i]$ 将此余值赋给 α_i 并计算该字段。我们还有表示余值对的余值（S，S'），与之对应的值为 $\mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c$，即一个检查当前余值并取出 S 和 S' 的表达式；此外还有由单个值 $[V]$ 组成的余值，以及与之相对的值 $\mu[x^{\varepsilon}].c$，它们实现了否定。

添加非极性 $\Box$。 目前介绍的系统对应于经典极化 L 演算的一个片段。我们不只是添加 S4 的 $\Box$ 模态：我们还考虑一种新的极性 $\Box$，它与 $+$ 类似，因为它遵循按值调用求值。相应的值由 $\Box V$引入，余值则由 $\tilde{\mu}\Box x^{\varepsilon}.c$引入。此外，由 $\Box$标注的模态变量具有特殊之处，因为模态值只能引用模态变量；类型判断会对此加以约束。

极性与模态的区别在于，极化类型系统反映了类型构造下的稳定性——典型例子是，两个模态值组成的正对本身也是模态的。类型系统通过极性表体现这一点；极性表描述了如何根据类型的组成部分推导出类型的极性。[1] 用范畴论术语来说，这相当于将关注点从余单子 $\Box$ 转向其分解为一个伴随，通常涉及 $\Box$-余代数范畴，并在由伴随联系起来的多个范畴之间解释演绎系统。这里我们不深入讨论范畴论细节，但我们的类型 A 具有极性 $\Box$，且都满足 $\Box A \to A$。

最终，我们有三种极性：通常的正极性和负极性 $+$，$-$以及一种额外的模态正极性 $\Box$。正极性 $+$，$\Box$ 记作 $\boxplus$，非模态极性 $+$，$-$ 记作 $\pm$。正如我们将看到的，极性 $\Box$ 的行为几乎与 $+$相同，并且与之共享许多构造。不过，在图 3 和图 4 所示的第二种语义中，它们的行为会有所不同。3 和 4。

ε $::=$ $+$ $\mid$ $-$ $\mid$ $\Box$ 极性$\boxplus$ $::=$ $+$ $\mid$ $\Box$ 正极性$\pm$ $::=$ $+$ $\mid$ $-$ 非模态极性$V, W$$::=$ x $\mid (V, W)$ $\mid$ $\Box V$ $\mid ()$ $\mid$ $\iota_1\ V$ $\mid$ $\iota_2\ V$ $\mid$ $\mu[x^{\varepsilon}].c$值$\mid$ $\mu(\pi_1\ \alpha^{\varepsilon_1}.c_1 \mid \pi_2\ \beta^{\varepsilon_2}.c_2)$ $\mid$ $\mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c$ $\mid$ $\mu\alpha^{-}.cS, S'$ $::$ $=$ α $\mid$ $\pi_1\ S$ $\mid$ $\pi_2\ S$ $\mid$ $\tilde{\mu}\Box x^{\varepsilon}.c$ $\mid$ $[V]$余值$\mid (S, S')$ $\mid$ $\tilde{\mu}().c$ $\mid$ $\tilde{\mu}x^{\boxplus}.c$ $\mid$ $\tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c$ $\mid$ $\tilde{\mu}(\iota_1\ x^{\varepsilon_1}.c_1 \mid \iota_2\ y^{\varepsilon_2}.c_2)t$ $::=$ V $\mid$ $\mu\alpha^{\boxplus}.c$ 表达式e $::=$ S $\mid$ $\tilde{\mu}x^{-}.c$ 环境c $::=$ $\langle t \parallel^{\boxplus} S\rangle$ $\mid \langle V \parallel^{-} e\rangle$ 命令 文法 *1*。*systemL* 的语法$_{\mathrm{pol}}^{\Box}$

[1]这个想法可以追溯到 Girard [1991, 1993], ；他通过线性-非线性项演算和模型间接影响了模态演算 [Benton 1994] （至少如此）。

我们为该系统定义了操作语义 $\triangleright$ （图1）。它只在顶层进行约简，并且具有确定性：若 $c \triangleright c_1$ 和 $c \triangleright c_2$，则 $c_1 = c_2$。

$$\begin{array}{rcc}
\langle \mu\alpha^{\varepsilon}.c \parallel^{\varepsilon} S\rangle & \triangleright & c[S/\alpha] \\
\langle V \parallel^{\varepsilon} \tilde{\mu}x^{\varepsilon}.c\rangle & \triangleright & c[V/x] \\
\langle () \parallel^{\square} \tilde{\mu}().c\rangle & \triangleright & c \\
\langle (V,W) \parallel^{\varepsilon} \tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c\rangle & \triangleright & c[V/x, W/y] \\
\langle \iota_i\ V \parallel^{\varepsilon} \tilde{\mu}(\iota_1\ x_1^{\varepsilon_1}.c_1 \mid \iota_2\ x_2^{\varepsilon_2}.c_2)\rangle & \triangleright & c_i[V/x_i] \\
\langle \square V \parallel^{\square} \tilde{\mu}\square x^{\varepsilon}.c\rangle & \triangleright & c[V/x] \\
\langle \mu[x^{\varepsilon}].c \parallel^{-} [V]\rangle & \triangleright & c[V/x] \\
\langle \mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c \parallel^{-} (S, S')\rangle & \triangleright & c[S/\alpha, S'/\beta] \\
\langle \mu(\pi_1\ \alpha_1^{\varepsilon_1}.c_1 \mid \pi_2\ \alpha_2^{\varepsilon_2}.c_2) \parallel^{-} \pi_i\ S\rangle & \triangleright & c_i[S/\alpha_i]
\end{array}$$

图 1. 系统$\mathbf{L}^{\square}_{\mathrm{pol}}$的操作语义

3 类型判断

类型及其对应的极性定义如下：

$$\begin{array}{cccccccccc}
A & , & B & ::= & \mathbb{1} & \mid & A \otimes B & \mid & A \oplus B & \mid \square A \mid \neg A \mid A \,\&\, B \mid A \quad B \\
\varpi(A), & & \varpi(B) & = & \square & & + & & + & \square \quad - \quad - \quad\quad -
\end{array}$$

其中 $x = \varpi(A) \odot \varpi(B)$，其定义将在下文给出。换言之，我们在不含指数算子的经典线性逻辑的联结词基础上，加入一个模态 $\square$，表示一个强余单子。然而，该系统并非线性的（因此余单子记作“$\square$”，而非“!”）。所以，尽管 $A \oplus B$和 $A \quad B$ 在逻辑上等价，且都表示析取，但它们并不同构，归约规则也不同。同样，$\otimes$ 和 & 都表示合取，而 $\mathbb{1}$ 是 $\otimes$ 的单位元。最后，$\neg$ 是（负）否定。

类型具有极性 $\varpi(A) = \varepsilon$：负类型为 $\varpi(A) = -$，非模态的正类型为 $\varpi(A) = +$，而模态类型为 $\varpi(A) = \square$；本质上，它们是 $\square$ 的余代数。正类型和模态类型对应按值调用类型，负类型则对应按名调用类型。极性上的运算 $\odot$ 定义如下：

$$\varepsilon \odot \varepsilon' = \begin{cases} \square & \text{whenever } (\varepsilon, \varepsilon') = (\square, \square) \\ + & \text{otherwise.} \end{cases}$$

我们注意到 A_{ε} 要断言类型 A 满足条件 $\varpi(A) = \varepsilon$。

可以看出，该系统中没有函数类型。类型 $A \to B$表示推值调用的箭头类型，可以宏定义为 $\neg A \quad B$，其中表示负析取；类型为 $A \to B$的值通过 $\mu(a \cdot \beta).c \triangleq \mu(\alpha,\ \beta)$。$\mu[a].c \parallel^{-} \alpha$ 引入——在按名调用中，$\lambda x.t$定义为 $\mu(x \cdot \beta)$。$t \parallel^{-} \beta$：其思路是，类型为 $A \to B$的值捕获一个类型为A 的值和一个类型为 B的延续：函数返回的值必须发送给这个延续。类型为 $A \to B$的余值通过 $V \cdot S \triangleq ([V],\ S)$引入。其思路是，类型为$A \to B$ 的余值由一个位于类型为 B的余值之上的类型为 V 的值构成，该余值表示函数结果将在其上运行的余值：例如，$t\ u$ 被翻译为 $\mu\beta^{-}.t \parallel^{-} u \cdot \beta$（$u$ 是一个值，因为在按名调用中，每个表达式都被翻译为负类型的值）。

类似地，可以定义极性转换，将类型转换为正类型或负类型：类型 $\Uparrow A \triangleq A \otimes \mathbb{1}$ 是正类型：其值为 $\Uparrow V \triangleq (V, ())$，其余值为 $\tilde{\mu}\Uparrow x^{\varepsilon}.c \triangleq \tilde{\mu}(x^{\varepsilon}, u^{\square}).c$；而类型 $\Downarrow A \triangleq \neg\mathbb{1} \quad A$ 是负类型，其值为 $\mu\Downarrow\alpha^{\varepsilon}.c \triangleq \mu(\beta^{-}, \alpha).c$，其余值为 $\Downarrow S \triangleq ([()], S)$。

模态逻辑 S4 是一种逻辑，带有满足 $\Box(A \to B) \to \Box A \to \Box B$、$\Box A \to A$ 和 $\Box A \to \Box\Box A$ 的模态 $\Box$。它还包含必然化规则：若 A 无需假设即可证明，则 $\Box A$ 也无需假设即可证明。这里，$\Box$ 遵循的规则几乎相同。然而，将模态与求值顺序（效应）结合起来，会暴露出与值限制相关的困难，这与 Curien、Fiore 和 Munch-Maccagnoni [2016] 的讨论类似：正如他们所解释的，在值范畴上既有一个余单子，又有一个效应伴随，就要求规则 $\vdash\Box$ 仅限于值。因此，在 $\mathrm{L}^{\Box}_{\mathrm{pol}}$ 中，仅凭 $A \to B$ 的证明通常无法推出 $\Box A \to \Box B$ 的证明，而只能推出 $\Box A \to \Box\Uparrow B$，这是值限制所致。Curien 等人 [2016] 通过将该模态实质上视为 $\Box\Uparrow$，以另一种方式处理了这一特殊性。我们对模态的处理与 Curien 等人 [2016] 的做法有两点不同：构造 $\Box V$ 不会引入隐式的延迟，而模态上下文则按照 Kavvos [2017] 的方式处理。直观而言，$\Box$ 保持积结构，这与 Kavvos [2017] 的语义一致；而 Curien 等人 [2016], 的模态 ! 则不具备这一性质，因此我们的方法不适用于它。

上下文（$\Gamma \mid \Theta \vdash \Delta$）由三部分组成，分别用（余）变量的有序列表表示：$\Gamma$、$\Theta$和 Δ。它们都不是线性的：都支持置换、收缩和弱化。Γ 是上下文中通常所说的“直觉主义”部分，包含变量及其类型。Θ 是模态部分，同样由变量及其类型组成：$x : A \in \Theta$ 应视为 $x : \Box A \in \Gamma$，但它允许提取模态背后的值。最后，Δ 是经典部分，由余变量$\alpha : A$组成，代表延续。空列表表示为 $\diamond$。

事实上，根据经典逻辑的 Curry–Howard 对应，我们知道 $\vdash$ 右侧的公式对应于延续变量。其语法本质上是双侧 L 演算的一个片段 Munch-Maccagnoni [2009, 2017], ，左侧带有一个对偶上下文，其设计受到以下语法的启发 Kavvos [2017]，专用于保积模态。最后，相继式也可以包含一个特殊区域（命令判断除外），该区域可以位于左侧或右侧，且最多包含一个公式。直观而言，该区域中的公式就是我们当前处理的公式。

我们考虑图 2 中给出的类型规则。共有五种类型判断：

- “$\Gamma \mid \Theta \vdash V : A; \Delta$”：$V$是上下文 $(\Gamma \mid \Theta \vdash \Delta)$ 中类型为 A 的值；
- “$\Gamma \mid \Theta \vdash t : A \mid \Delta$”：$t$是上下文 $(\Gamma \mid \Theta \vdash \Delta)$ 中类型为 A 的表达式；
- “$\Gamma \mid \Theta; S : A \vdash \Delta$”：$S$是上下文 $(\Gamma \mid \Theta \vdash \Delta)$ 中类型为 A 的余值；
- “$\Gamma \mid \Theta \mid e : A \vdash \Delta$”：$e$是上下文 $(\Gamma \mid \Theta \vdash \Delta)$ 中类型为 A 的环境；
- “$c : (\Gamma \mid \Theta \vdash \Delta)$”：$c$是上下文 $(\Gamma \mid \Theta \vdash \Delta)$ 中的命令。

书写 $\Gamma, \Gamma', \Theta, \Theta'$ 或 Δ, Δ'时，我们假设 Γ 与 Γ'、Θ 与 Θ' 以及 Δ 与 Δ' 互不相交。$\Gamma_{\Box}$ 表示一个上下文，其中所有类型 A 均满足 $\omega(A) = \Box$。我们定义重命名 $\theta \in \mathrm{R}(\Gamma \mid \Theta \vdash \Delta \Rightarrow \Gamma' \mid \Theta' \vdash \Delta')$: θ是一个变量与余变量之间的函数，使得若 $x : A \in \Gamma$，则$\theta(x) : A \in \Gamma'$；若 $x : A \in \Theta$，则 $\theta(x) : A \in \Theta'$；若 $\alpha : A \in \Delta$，则 $\theta(\alpha) : A \in \Delta'$。

我们注意到，某些规则要求将上下文分开，例如 R$\otimes$ 规则。不过这不是问题，因为我们可以使用重命名来合并变量。

$$\dfrac{\dfrac{}{x : A \mid \diamond \vdash x : A; \diamond}(\vdash \mathrm{Ax}) \qquad \dfrac{}{y : A \mid \diamond \vdash y : A; \diamond}(\vdash \mathrm{Ax})}{\dfrac{x : A, y : A \mid \diamond \vdash (x, y) : A \otimes A; \diamond}{z : A \mid \diamond \vdash (z, z) : A \otimes A; \diamond}(\vdash \mathfrak{R}(V))}(\vdash \otimes)$$

我们应用 了重命名 $\theta(x) = \theta(y) = z$。它保持类型不变：$x$和 y是类型 A，z也是。

$$\frac{}{x : A \mid \diamond \vdash x : A; \diamond}\,(\vdash \mathrm{Ax}) \qquad \frac{}{\diamond \mid x : A \vdash x : A; \diamond}\,(\vdash \Box \mathrm{Ax})$$

$$\frac{}{\diamond \mid \diamond \mid \alpha : A \vdash \alpha : A}\,(\mathrm{Ax}\vdash)$$

$$\frac{\Gamma \mid \Theta \vdash V : A_{\boxplus}; \Delta \qquad \Gamma' \mid \Theta' \mid e : A_{\boxplus} \vdash \Delta'}{\langle V \parallel^{\boxplus} e \rangle : (\Gamma, \Gamma' \mid \Theta, \Theta' \vdash \Delta, \Delta')}\,(\mathrm{cut}_{\boxplus})$$

$$\frac{\Gamma \mid \Theta \vdash t : A_{-} \mid \Delta \qquad \Gamma' \mid \Theta'; S : A_{-} \vdash \Delta'}{\langle t \parallel^{-} S \rangle : (\Gamma, \Gamma' \mid \Theta, \Theta' \vdash \Delta, \Delta')}\,(\mathrm{cut}_{-})$$

$$\frac{c : (\Gamma \mid \Theta \vdash \alpha : A_{\boxplus}, \Delta)}{\Gamma \mid \Theta \vdash \mu\alpha^{\boxplus}.c : A_{\boxplus} \mid \Delta}\,(\vdash \mu^{\boxplus}) \qquad \frac{c : (\Gamma \mid \Theta \vdash \alpha : A_{-}, \Delta)}{\Gamma \mid \Theta \vdash \mu\alpha^{-}.c : A_{-}; \Delta}\,(\vdash \mu^{-})$$

$$\frac{c : (\Gamma, x : A_{\boxplus} \mid \Theta \vdash \Delta)}{\Gamma \mid \Theta; \tilde{\mu}x^{\boxplus}.c : A_{\boxplus} \vdash \Delta}\,(\tilde{\mu}^{\boxplus}\vdash) \qquad \frac{c : (\Gamma, x : A_{-} \mid \Theta \vdash \Delta)}{\Gamma \mid \Theta; \tilde{\mu}x^{-}.c : A_{-} \vdash \Delta}\,(\tilde{\mu}^{-}\vdash)$$

$$\frac{}{\diamond \mid \diamond \vdash () : \mathbb{1}; \diamond}\,(\vdash \mathbb{1}) \qquad \frac{c : (\Gamma \mid \Theta \vdash \Delta)}{\Gamma \mid \Theta; \tilde{\mu}().c : \mathbb{1} \vdash \Delta}\,(\mathbb{1}\vdash)$$

$$\frac{\Gamma \mid \Theta \vdash V : A; \Delta \qquad \Gamma' \mid \Theta' \vdash W : B; \Delta'}{\Gamma, \Gamma' \mid \Theta, \Theta' \vdash (V, W) : A \otimes B; \Delta, \Delta'}\,(\vdash \otimes)$$

$$\frac{c : (\Gamma, x : A_{\varepsilon_1}, y : B_{\varepsilon_2} \mid \Theta \vdash \Delta)}{\Gamma \mid \Theta; \tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c : A_{\varepsilon_1} \otimes B_{\varepsilon_2} \vdash \Delta}\,(\otimes\vdash)$$

$$\frac{\Gamma \mid \Theta \vdash V : A_i; \Delta}{\Gamma \mid \Theta \vdash \iota_i\, V : A_1 \oplus A_2; \Delta}\,(\vdash \oplus)$$

$$\frac{c_1 : (\Gamma, x : A_{\varepsilon_1} \mid \Theta \vdash \Delta) \qquad c_2 : (\Gamma, y : B_{\varepsilon_2} \mid \Theta \vdash \Delta)}{\Gamma \mid \Theta; \tilde{\mu}(\iota_1\, x^{\varepsilon_1}.c_1 \mid \iota_2\, y^{\varepsilon_2}.c_2) : A_{\varepsilon_1} \oplus B_{\varepsilon_2} \vdash \Delta}\,(\oplus\vdash)$$

$$\frac{\Gamma \mid \Theta \vdash V : A_{\boxplus}; \Delta}{\Gamma \mid \Theta \vdash V : A_{\boxplus} \mid \Delta}\,(\vdash V) \qquad \frac{\Gamma \mid \Theta; S : A_{-} \vdash \Delta}{\Gamma \mid \Theta \mid S : A_{-} \vdash \Delta}\,(S\vdash)$$

$$\frac{c : (\Gamma \mid \Theta \vdash \alpha : A_{\varepsilon_1}, \beta : B_{\varepsilon_2}, \Delta)}{\Gamma \mid \Theta \vdash \mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c : A_{\varepsilon_2} \quad B_{\varepsilon_1}; \Delta}\,(\vdash \ )$$

$$\frac{\Gamma \mid \Theta \mid e : A \vdash \Delta \qquad \Gamma' \mid \Theta' \mid f : B \vdash \Delta'}{\Gamma, \Gamma' \mid \Theta, \Theta' \mid (e, f) : A \quad B \vdash \Delta, \Delta'}\,(\ \vdash)$$

$$\frac{c : (\Gamma, x : A_{\varepsilon} \mid \Theta \vdash \Delta)}{\Gamma \mid \Theta \vdash \mu[x^{\varepsilon}].c : \neg A_{\varepsilon}; \Delta}\,(\vdash \neg) \qquad \frac{\Gamma \mid \Theta \vdash V : A; \Delta}{\Gamma \mid \Theta; [V] : \neg A \vdash \Delta}\,(\neg\vdash)$$

$$\frac{c_1 : (\Gamma, \mid \Theta \vdash \Delta, \alpha : A_{\varepsilon_1}) \qquad c_2 : (\Gamma, \mid \Theta \vdash \Delta, \beta : B_{\varepsilon_2})}{\Gamma \mid \Theta \vdash \mu(\pi_1\ \alpha^{\varepsilon_1}.c_1 \mid \pi_2\ \beta^{\varepsilon_2}.c_2) : A_{\varepsilon_1} \,\&\, B_{\varepsilon_2}; \Delta}\,(\vdash \&)$$

$$\frac{\Gamma \mid \Theta; S : A_i^{\varepsilon_i} \vdash \Delta}{\Gamma \mid \Theta; \pi_i\ S : A_1^{\varepsilon_1} \,\&\, A_2^{\varepsilon_2} \vdash \Delta}\,(\&\vdash)$$

$$\frac{\Gamma_{\Box} \mid \Theta \vdash V : A; \diamond}{\Gamma_{\Box} \mid \Theta \vdash \Box V : \Box A; \diamond}\,(\vdash \Box) \qquad \frac{c : (\Gamma \mid x : A_{\varepsilon}, \Theta \vdash \Delta)}{\Gamma \mid \Theta; \tilde{\mu}\Box x^{\varepsilon}.c : \Box A_{\varepsilon} \vdash \Delta}\,(\Box\vdash)$$

$$\frac{\Gamma \mid \Theta \vdash V : A; \Delta}{\Gamma' \mid \Theta' \vdash \theta(V) : A; \Delta'}\,(\vdash \mathfrak{R}(V)) \qquad \frac{\Gamma \mid \Theta \vdash t : A \mid \Delta}{\Gamma' \mid \Theta' \vdash \theta(t) : A; \Delta'}\,(\vdash \mathfrak{R}(t))$$

$$\frac{\Gamma \mid \Theta; S : A \vdash \Delta}{\Gamma' \mid \Theta'; \theta(S) : A \vdash \Delta'}\,(\mathfrak{R}(S)\vdash) \qquad \frac{\Gamma \mid \Theta \mid e : A \vdash \Delta}{\Gamma' \mid \Theta' \mid \theta(e) : A \vdash \Delta'}\,(\mathfrak{R}(e)\vdash)$$

$$\frac{c : (\Gamma \mid \Theta \vdash \Delta)}{\theta(c) : (\Gamma' \mid \Theta' \vdash \Delta')}\,(\mathfrak{R}(c))$$

图 2。系统**L**的类型规则$^{\Box}_{\mathrm{pol}}$

同样，公理规则只允许上下文中出现一个变量/余变量。这同样不是问题：

$$\frac{\dfrac{}{\diamond \mid \diamond; \alpha : A \vdash \alpha : A}\,(\mathrm{Ax}\vdash)}{\Gamma \mid \Theta; \alpha : A \vdash \alpha : A, \Delta}\,(\mathfrak{R}(S)\vdash)$$

这里我们应用了重命名 $\theta(\alpha) = \alpha$。当然，它保留了 α 的类型。我们在上下文中添加的其他变量并未出现在表达式中，因此它们不会出现任何“问题”。

最后，我们可以使用重命名来交换变量，以便在正确的位置分隔上下文，例如这里的 $\theta(x) = x$ 和 $\theta(y) = y$：

$$\frac{\dfrac{\dfrac{}{\diamond \mid y : B \vdash y : B; \diamond}\,(\vdash \Box \mathrm{Ax}) \qquad \dfrac{}{\diamond \mid x : A \vdash x : A; \diamond}\,(\vdash \Box \mathrm{Ax})}{\diamond \mid y : B, x : A \vdash (y, x) : B \otimes A; \diamond}\,(\vdash \otimes)}{\diamond \mid x : A; y : B \vdash (y, x) : B \otimes A; \diamond}\,(\vdash \mathfrak{R}(V))$$

这个类型系统具有一些可靠性性质：

引理 3.1（模态限制）。若 V是一个值，且 $\Gamma \mid \Theta \vdash V\colon A_{\Box}; \Delta$ 则 $\Gamma_{\Box} \mid \Theta \vdash V\colon A_{\Box}; \diamond$.

这是该演算的关键性质。它确保模态值在其闭包中仅引用模态值；因此，如果我们将模态值分配在堆上，其余值分配在栈上，就能确保堆上的值不会引用栈。

引理 *3.2*（类型化替换）。如果 c（或 t、V、S 或 e）是在上下文（$\Gamma \mid \Theta \vdash \Delta$）中的良类型命令、表达式等；如果（$\Gamma' \mid \Theta' \vdash \Delta'$）是另一个上下文，且 σ 是一个函数，使得对所有$x : A \in \Gamma$，$\sigma(x)$都是一个值，满足 $\Gamma' \mid \Theta' \vdash \sigma(x) : A; \Delta'$；对每个 $x : A \in \Theta$，$\sigma(x)$都是一个值，满足 $\Gamma'_{\Box} \mid \Theta' \vdash \sigma(x) : A; \diamond$；并且对每个 $\alpha : A \in \Delta$，$\sigma(\alpha)$都是一个余值，满足 $\Gamma' \mid \Theta'; \sigma(\alpha) : A \vdash \Delta'$，那么 $\sigma(c)$，相应地 $\sigma(t)$等，在上下文（$\Gamma' \mid \Theta' \vdash \Delta'$）中都是类型正确的。此外，如果适用，其类型保持不变。

引理 3.3（主体归约）。若 $c \triangleright c'$ 且 $c : (\Gamma \mid \Theta \vdash \Delta)$ 则 $c' : (\Gamma \mid \Theta \vdash \Delta)$。

引理 3.4（正规化）。 $\triangleright$ 在良类型项上是正规化的。

证明。事实上，我们有一个更强的结果：$\triangleright$ 的上下文闭包是强正规化的。实际上，$\mathrm{L}^{\Box}_{\mathrm{pol}}$ 可以通过一种直接的方式翻译为 Curien 等人 [2016] 的 LJ^{η}_{p} 系统：将 $\Box A$ 翻译为 $\Uparrow A$，将 $(\Gamma \mid \Theta \vdash \Delta)$ 翻译为 $(\Gamma, \Uparrow\Theta \vdash \Delta)$，将 $\Box V$ 翻译为 $(V, ())$，将 $\tilde{\mu}\Box x.c$ 翻译为 $\tilde{\mu}\Uparrow x.c$。此外，极性 $\Box$ 翻译为 $+$。这种翻译保留类型判断，并且是一种模拟；他们的系统是强正规化的（例如参见 [Munch-Maccagnoni 2017]）；因此我们的系统也是强正规化的。□

这些引理共同使我们能够证明，每个“闭合”命令 $c : (\diamond \mid \diamond \vdash tp : R_{\Box})$都可以求值为一个值，其中 R 是一个模态类型，而 tp 是顶层延续。事实上，这样的命令会归约到正规形式 c'。通过分类讨论，我们可以看出，对于某个V，$c' = V \Vdash^{\Box} tp$。根据主体归约，这个 V 在上下文$(\diamond \mid \diamond \vdash tp : R)$中具有类型 R；但借助模态限制引理，我们实际上可以证明它不包含自由变量。这个 V 是唯一的，因为$\triangleright$ 具有确定性。

定理 3.5（求值）。若 $c : (\diamond \mid \diamond \vdash tp : R_{\Box})$，则存在一个类型为 R 的闭值 V，使得$c \triangleright^{*} \langle V \Vdash^{\Box} tp\rangle$。此外，这个 V 是唯一的。

这很有意思，因为归根结底，我们关注的是对返回纯正值（如布尔值或整数）的完整程序求值；这些值确实由模态数据类型给出（例如 $\mathbb{1} \oplus \mathbb{1}$ 它是一种模态类型）。与 $\lambda_{1/2}$ 中描述的类型系统类似，[Cong 等人 2019], 在这个系统中，我们可以用 $\Box$ 包装一种类型，以强制其值为纯值，即不包含延续；但我们可以在返回纯值的函数中局部使用延续，而类型系统会确保延续不会逸出。

4 类机器语义

本节介绍一种用于演算的类机器语义，它显式管理内存。对于他们的演算，Cong 等人 [2019] 设计了一种具有有趣性质的机器，即栈化性：调用一个接受一等参数的函数时，栈会被释放，并重置为该函数定义时的状态。这里，我们并不直接采用一等类型或二等类型的概念；不过，他们系统中的一等类型对应于我们系统中的模态类型，因此，在我们的系统中，接受一等参数的函数类型为 $A_{\Box} \to B_{\Box} = \Uparrow(\neg A_{\Box} \quad B_{\Box})$。请注意，在他们的设定中，$B$ 始终是一等类型，因此在我们的系统中是模态类型。我们提出的语义基于对栈化性质的推广。

我们首先在 gra. 2 中介绍机器的内存。它可以存储值和余值。类型规则将确保，只要变量 x 被存储在 $\mathcal{M}$ 中，它就必须是负的，即表示一个尚未求值的延迟计算。更一般地，如果一个正变量或模态变量

x 出现在存储于内存中的值 V 里，那么它的每次出现 x 都是某个命令 c的子项，而该命令本身又是 V的子项。与此对偶，出现在存储于 M 中的余值 S里的负协变量，只能作为某个命令的子项出现，而该命令又是 S的子项。因此，如果 x是正的，那么严格对 (x, x) 不能存储在 M 中；但 $\mu(\pi_1\ \alpha^+ x \Vdash^+ \alpha \mid \pi_2\ \beta^+ x \Vdash^+ \beta)$ 可以存储在 M 中，它表示一个惰性记录，其两个字段的值均为 x。与此对偶，存储的协变量 α必须是正的或模态的。为了概述这一区别，我们将用 $\mathbb{V}, \mathbb{W}$ 或 $\mathbb{S}$ 表示存储在 M 中的值和余值。

$$
\begin{array}{lcll}
\mathcal{H} & ::= & \diamond \mid \mathcal{H}, x^{\varepsilon} := \mathbb{V} & \textit{Heap} \\
\mathcal{E} & ::= & x^{\pm} := \mathbb{V} \mid \alpha^{\varepsilon} := \mathbb{S} & \textit{Equations} \\
\mathcal{F} & ::= & (\mathcal{E}) \mid (\mathcal{E}, \mathcal{E}) & \textit{Stack frame} \\
\mathcal{S} & ::= & \diamond \mid \mathcal{S}, \mathcal{F} & \textit{Stack} \\
\mathcal{M} & ::= & (\mathcal{H}; \mathcal{S}) & \textit{Memory}
\end{array}
$$

语法 2. 内存语法

M 表示内存；它由堆 H 和栈 S 组成。堆 H 用于存储模态类型的值，栈 S 用于存储非模态类型的值和余值。我们也可以在 H 中分配非模态类型的值，但前提是这些值不引用 Θ 之外的变量，模态类型的变量除外。这样可以确保堆上的值不会引用栈。栈由若干栈帧 F 组成。我们可以通过操作 $\mathrm{M}[x^{\varepsilon} := \mathbb{V}]$ 和 $\mathrm{M}[\alpha^{\varepsilon} := \mathbb{S}]$ 分配新值，这些操作会相应地在堆或栈上分配内存：每当在栈上分配新变量时，都会创建一个新的栈帧。栈帧有一个地址：我们用 $@x$ 或 $@\alpha$ 表示分配 x 或 α 的栈帧地址。我们可以限制内存 $\mathrm{M}|_{@\alpha}$，使其丢弃 α 所属的栈帧及之后分配的所有栈帧。我们用 $\mathrm{M}(x)$ 和 $\mathrm{M}(\alpha)$ 分别表示 M 中分配给 x 和 α 的值和栈；用 $\varpi(x)$ 或 $\varpi(\alpha)$ 表示其极性 ε（堆上的值全都是模态的）。内存在记法中将予以省略。

更形式化地，对于该分配：

$$
\begin{array}{rcl}
(\mathcal{H}; \mathcal{S})[x^{\square} := \mathbb{V}] & \stackrel{\text{def}}{=} & (\mathcal{H}, (x^{\square} := \mathbb{V}); \mathcal{S}) \\
(\mathcal{H}; \mathcal{S})[x^{\pm} := \mathbb{V}] & \stackrel{\text{def}}{=} & (\mathcal{H}; \mathcal{S}, (x^{\varepsilon} := \mathbb{V})) \\
(\mathcal{H}; \mathcal{S})[\alpha^{\varepsilon} := \mathbb{S}] & \stackrel{\text{def}}{=} & (\mathcal{H}; \mathcal{S}, (\alpha^{\varepsilon} := \mathbb{S})) \\
(\mathcal{H}; \mathcal{S})[x^{\square} := \mathbb{V}, y^{\square} := \mathbb{V}'] & \stackrel{\text{def}}{=} & (\mathcal{H}, (x := \mathbb{V}), (y := \mathbb{V}'); \mathcal{S}) \\
(\mathcal{H}; \mathcal{S})[x^{\square} := \mathbb{V}, \alpha^{\varepsilon} := \mathbb{S}] & \stackrel{\text{def}}{=} & (\mathcal{H}, (x := \mathbb{V}); \mathcal{S}, (\alpha^{\varepsilon} := \mathbb{S})) \\
(\mathcal{H}; \mathcal{S})[\alpha^{\varepsilon} := \mathbb{S}, \beta^{\varepsilon'} := \mathbb{S}'] & \stackrel{\text{def}}{=} & (\mathcal{H}; \mathcal{S}, (\alpha^{\varepsilon} := \mathbb{S}, \beta^{\varepsilon'} := \mathbb{S}')) \\
\multicolumn{3}{c}{\text{etc.}}
\end{array}
$$

对于限制条件：

$$
\begin{array}{rcl}
(\mathcal{H}; \diamond)|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H}; \diamond) \\
(\mathcal{H}; \mathcal{S}, (\alpha^{\varepsilon} := \mathbb{S}))|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H}; \mathcal{S}) \\
(\mathcal{H}; \mathcal{S}, (\beta^{\varepsilon} := \mathbb{S}))|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H}; \mathcal{S})|_{@\alpha} \\
(\mathcal{H}; \mathcal{S}, (x^{\pm} := \mathbb{V}))|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H}; \mathcal{S})|_{@\alpha} \\
(\mathcal{H}; \mathcal{S}, (\beta^{\varepsilon} := \mathbb{S}, \gamma^{\varepsilon'} := \mathbb{S}'))|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H}; \mathcal{S})|_{@\alpha} \\
(\mathcal{H}; \mathcal{S}, (\alpha^{\varepsilon} := \mathbb{S}, \beta^{\varepsilon'} := \mathbb{S}'))|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H}; \mathcal{S}) \\
(\mathcal{H}; \mathcal{S}, (\beta^{\varepsilon'} := \mathbb{S}, \alpha^{\varepsilon} := \mathbb{S}'))|_{@\alpha} & \stackrel{\text{def}}{=} & (\mathcal{H}; \mathcal{S}) \\
\multicolumn{3}{c}{\text{etc.}}
\end{array}
$$

现在，我们如何对值和余值求值。我们定义以下关系（图 3）：

$$\mathcal{M} \vDash V \Downarrow_v \mathbb{V} \qquad \mathcal{M} \vDash S \Downarrow_s \mathbb{S}$$

第一种表示 V 在环境 M 中求值为值 $\mathbb{V}$，第二种表示 S 在环境 M 中求值为余值 $\mathbb{S}$。它们不会强制求值延迟计算；因此，负变量不会被求值，正余变量或模态余变量也同样如此。

$$\frac{\mathcal{M}(x) = \mathbb{V} \qquad \varpi(x) \in \{+, \square\}}{\mathcal{M} \vDash x \Downarrow_v \mathbb{V}} \qquad \frac{\varpi(x) = -}{\mathcal{M} \vDash x \Downarrow_v x}$$

$$\frac{\varpi(\alpha) \in \{+, \square\}}{\mathcal{M} \vDash \alpha \Downarrow_s \alpha} \qquad \frac{\mathcal{M}(\alpha) = \mathbb{S} \qquad \varpi(\alpha) = -}{\mathcal{M} \vDash \alpha \Downarrow_s \mathbb{S}}$$

$$\frac{\mathcal{M} \vDash V \Downarrow_v \mathbb{V} \qquad \mathcal{M} \vDash V' \Downarrow_v \mathbb{W}}{\mathcal{M} \vDash (V, V') \Downarrow_v (\mathbb{V}, \mathbb{W})} \qquad \frac{\mathcal{M} \vDash V \Downarrow_v \mathbb{V}}{\mathcal{M} \vDash \iota_i\, V \Downarrow_v \iota_i\, \mathbb{V}}$$

$$\frac{}{\mathcal{M} \vDash \tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c \Downarrow_s \tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c} \qquad \frac{\mathcal{M} \vDash S \Downarrow_s \mathbb{S}}{\mathcal{M} \vDash \pi_i\, S \Downarrow_s \pi_i\, \mathbb{S}}$$

$$\frac{}{\mathcal{M} \vDash \tilde{\mu}(\iota_1\, x_1^{\varepsilon_1}.c_1 \mid \iota_2\, x_2^{\varepsilon_2}.c_2) \Downarrow_s \tilde{\mu}(\iota_1\, x_1^{\varepsilon_1}.c_1 \mid \iota_2\, x_2^{\varepsilon_2}.c_2)}$$

$$\frac{}{\mathcal{M} \vDash \mu(\pi_1\, \alpha_1^{\varepsilon_1}.c_1 \mid \pi_2\, \alpha_2^{\varepsilon_2}.c_2) \Downarrow_v \mu(\pi_1\, \alpha_1^{\varepsilon_1}.c_1 \mid \pi_2\, \alpha_2^{\varepsilon_2}.c_2)}$$

$$\frac{}{\mathcal{M} \vDash () \Downarrow_v ()} \qquad \frac{}{\mathcal{M} \vDash \tilde{\mu}().c \Downarrow_s \tilde{\mu}().c}$$

$$\frac{\mathcal{M} \vDash V \Downarrow_v \mathbb{V}}{\mathcal{M} \vDash [V] \Downarrow_s [\mathbb{V}]} \qquad \frac{}{\mathcal{M} \vDash \mu[x^{\varepsilon}].c \Downarrow_v \mu[x^{\varepsilon}].c}$$

$$\frac{\mathcal{M} \vDash S \Downarrow_s \mathbb{S} \qquad \mathcal{M} \vDash S' \Downarrow_s \mathbb{S}'}{\mathcal{M} \vDash (S, S') \Downarrow_s (\mathbb{S}, \mathbb{S}')} \qquad \frac{}{\mathcal{M} \vDash \mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c \Downarrow_v \mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c}$$

$$\frac{\mathcal{M} \vDash V \Downarrow_v \mathbb{V}}{\mathcal{M} \vDash \square V \Downarrow_v \square \mathbb{V}} \qquad \frac{}{\mathcal{M} \vDash \tilde{\mu}\square x^{\varepsilon}.c \Downarrow_s \tilde{\mu}\square x^{\varepsilon}.c}$$

图 3. 值与栈的求值

例如，我们有：

$$(x := (); (y^- := \mathbb{V})) \vDash (x, y) \Downarrow_v ((), y)$$

现在，我们在图 4 中定义关系 $(\mathcal{M} \vDash c) \rightsquigarrow (\mathcal{M}' \vDash c')$，其含义是，给定内存 $\mathcal{M}$，命令 c 规约为命令 c'，并将内存修改为 $\mathcal{M}'$。这种语义的关键在于规则 求值$\tilde{\mu}\square$，它会在堆上分配模态变量（属于 Θ）：

$$\frac{\mathcal{M} \vDash V \Downarrow_v \square\mathbb{V}}{\left((\mathcal{H}; \mathcal{S}) \vDash \langle V \parallel^{\square} \tilde{\mu}\square x^{\varepsilon}.c\rangle\right) \rightsquigarrow ((\mathcal{H}, (x^{\varepsilon} := \mathbb{V}); \mathcal{S}) \vDash c)}\ \text{Eval}\tilde{\mu}\square$$

以及规则 求值$\square$:

$$\frac{\mathcal{M}(\alpha) =^{\square} \mathbb{S}}{\left(\mathcal{M} \vDash \langle V \parallel^{\square} \alpha\rangle\right) \rightsquigarrow \left(\mathcal{M}|_{@\alpha} \vDash \langle V \parallel^{\square} \mathbb{S}\rangle\right)}\ \text{Eval}\square$$

$$\frac{\mathcal{M}(\alpha) =^{+} \mathbb{S}}{\left(\mathcal{M} \vDash \langle V \Vdash^{+} \alpha\rangle\right) \rightsquigarrow \left(\mathcal{M} \vDash \langle V \Vdash^{+} \mathbb{S}\rangle\right)}\ \mathrm{Eval}+ \qquad \frac{\mathcal{M}(x) =^{-} \mathbb{V}}{\left(\mathcal{M} \vDash \langle x \Vdash^{-} S\rangle\right) \rightsquigarrow \left(\mathcal{M} \vDash \langle \mathbb{V} \Vdash^{-} S\rangle\right)}\ \mathrm{Eval}-$$

$$\frac{\mathcal{M}(\alpha) =^{\square} \mathbb{S}}{\left(\mathcal{M} \vDash \langle V \Vdash^{\square} \alpha\rangle\right) \rightsquigarrow \left(\mathcal{M}|_{@\alpha} \vDash \langle V \Vdash^{\square} \mathbb{S}\rangle\right)}\ \mathrm{Eval}\square \qquad \frac{\mathcal{M} \vDash S \Downarrow_s [\mathbb{V}]}{\left(\mathcal{M} \vDash \langle \mu[x^{\varepsilon}].c \Vdash^{-} S\rangle\right) \rightsquigarrow \left(\mathcal{M}[x^{\varepsilon} := \mathbb{V}] \vDash c\right)}\ \mathrm{Eval}\mu\neg$$

$$\frac{\mathcal{M} \vDash S \Downarrow_s \mathbb{S}}{\left(\mathcal{M} \vDash \langle \mu\alpha^{\varepsilon}.c \Vdash^{\varepsilon} S\rangle\right) \rightsquigarrow \left(\mathcal{M}[\alpha^{\varepsilon} := \mathbb{S}] \vDash c\right)}\ \mathrm{Eval}\mu\alpha^{\varepsilon} \qquad \frac{\mathcal{M} \vDash V \Downarrow_v \mathbb{V}}{\left(\mathcal{M} \vDash \langle V \Vdash^{\varepsilon} \tilde{\mu}x^{\varepsilon}.c\rangle\right) \rightsquigarrow \left(\mathcal{M}[x^{\varepsilon} := \mathbb{V}] \vDash c\right)}\ \mathrm{Eval}\tilde{\mu}x^{\varepsilon}$$

$$\frac{\mathcal{M} \vDash V \Downarrow_v (\mathbb{V}, \mathbb{V}')}{\left(\mathcal{M} \vDash \langle V \Vdash^{\varepsilon} \tilde{\mu}(x^{\varepsilon_1}, y^{\varepsilon_2}).c\rangle\right) \rightsquigarrow \left(\mathcal{M}[x^{\varepsilon_1} := \mathbb{V}, y^{\varepsilon_2} := \mathbb{V}'] \vDash c\right)}\ \mathrm{Eval}\tilde{\mu}\otimes$$

$$\frac{\mathcal{M} \vDash S \Downarrow_s (\mathbb{S}, \mathbb{S}')}{\left(\mathcal{M} \vDash \langle \mu(\alpha^{\varepsilon_1}, \beta^{\varepsilon_2}).c \Vdash^{-} S\rangle\right) \rightsquigarrow \left(\mathcal{M}[\alpha^{\varepsilon_1} := \mathbb{S}, \beta^{\varepsilon_2} := \mathbb{S}'] \vDash c\right)}\ \mathrm{Eval}\mu$$

$$\frac{\mathcal{M} \vDash V \Downarrow_v \iota_i\ \mathbb{V}}{\left(\mathcal{M} \vDash \langle V \Vdash^{\varepsilon} \tilde{\mu}(\iota_1\ x_1^{\varepsilon_1}.c_1 \mid \iota_2\ x_2^{\varepsilon_2}.c_2)\rangle\right) \rightsquigarrow \left(\mathcal{M}[x_i^{\varepsilon_i} := \mathbb{V}] \vDash c_i\right)}\ \mathrm{Eval}\tilde{\mu}\oplus$$

$$\frac{\mathcal{M} \vDash S \Downarrow_s \pi_i\ \mathbb{S}}{\left(\mathcal{M} \vDash \langle \mu(\pi_1\ \alpha_1^{\varepsilon_1}.c_1 \mid \pi_2\ \alpha_2^{\varepsilon_2}.c_2) \Vdash^{-} S\rangle\right) \rightsquigarrow \left(\mathcal{M}[\alpha_i^{\varepsilon_i} := \mathbb{S}] \vDash c_i\right)}\ \mathrm{Eval}\mu\&$$

$$\frac{\mathcal{M} \vDash V \Downarrow_v ()}{\left(\mathcal{M} \vDash \langle V \Vdash^{\square} \tilde{\mu}().c\rangle\right) \rightsquigarrow \left(\mathcal{M} \vDash c\right)}\ \mathrm{Eval}\tilde{\mu}\mathbb{1} \qquad \frac{\mathcal{M} \vDash V \Downarrow_v \square\mathbb{V}}{\left((\mathcal{H};\mathcal{S}) \vDash \langle V \Vdash^{\square} \tilde{\mu}\square x^{\varepsilon}.c\rangle\right) \rightsquigarrow \left((\mathcal{H}, x^{\varepsilon} := \mathbb{V};\mathcal{S}) \vDash c\right)}\ \mathrm{Eval}\tilde{\mu}\square$$

图 4. 命令求值

对模态类型的余变量 α 求值后，我们可以缩减栈。事实上，如果我们将范围限制在带类型值， V 属于模态类型，因此不可能包含对 S 的引用；而 α 的值 $\mathbb{S}$必须在分配 α 之前的环境中具有类型。

让我们考虑一下在求值时会发生什么，例如，对某个表达式进行（按值调用）函数 t 求值 u。我们假设 t 的类型为 $\Uparrow(A_{\square} \to B_{\square})$，也就是说，其中 A 和 B 是模态的。请注意 t 并不是模态类型。这正是Cong 等人 [2019]所考虑的情况。现在考虑下面表达式的求值：

$$\mu\gamma^{\square}.\left\langle t \,\middle\|^{+}\, \tilde{\mu}f^{+}.\left\langle \mu\alpha^{\square}.\langle u \Vdash^{\square} \alpha\rangle \,\middle\|^{\square}\, \tilde{\mu}x^{\square}.\left\langle f^{+} \,\middle\|^{+}\, \tilde{\mu}\Uparrow g^{-}.\langle g \Vdash^{-} x \cdot \gamma\rangle\right\rangle\right\rangle$$

首先，我们必须求值 t。假设它求值得到 f。然后我们求值 u（注意 η-展开）。假设它返回一个值 V：这个值将针对 α求值，后者出现在 $V \Vdash^{\square} \alpha$ 中。因此，在这一点上，由于 α 是模态的，栈将首次重置到开始求值 u 之前的位置。这是有效的，因为 V不包含对栈的引用，因为它是模态的；而且 α指向 $\tilde{\mu}x^{\square}.\left\langle f^{+} \,\middle\|^{+}\, \tilde{\mu}\Uparrow g^{-}.g \Vdash^{-} x \cdot \gamma\right\rangle$，其自由变量在开始求值 u之前就已分配。接着，我们将 $x := V$存储在堆上，从其包装器 f中取出实际函数 g，并对 x求值。这会返回某个值 V'，并针对γ求值。此时，由于 γ 也是模态的，我们可以将栈恢复到调用 $t\ u$之前的状态：因此，栈恢复到了初始状态，也没有发生内存泄漏。即使 A 是非模态的，我们仍能确保函数返回时释放栈，不过在求值 u之后则无法释放。这正是 Cong 等人 [2019] 所介绍的 CPS 翻译中的栈纪律。

更多基因 总之，在这种语义中，余值 S和非模态值 V遵循栈纪律：

定理 4.1（可栈化性）。这种语义会将余值 S和非模态值 V分配到栈$\mathcal{S}$上，并在模态表达式求值返回时收缩栈，使栈恢复到模态表达式开始求值之前的状态；这种语义是可靠的。

特别是，这适用于返回类型为模态类型的函数；若函数的参数也是模态类型，则同样适用。这意味着，如果将演算限制为返回类型为模态类型的函数，那么我们分配值的栈就对应于调用栈。

然而，返回非模态类型的函数，其行为有些奇怪：例如，$(\lambda x^+.\text{let } y = (x,\ x) \text{ in } y)\ V$会被翻译为 $t_{\text{def}}= \mu\beta^+.\left\langle \mu(x\cdot\ \alpha).\left\langle (x,\ x) \parallel^+ \tilde{\mu}y^+.y \parallel^+ \alpha \right\rangle \middle\| \ V\cdot\ \beta \right\rangle$，对此我们有：

$$\left((\mathcal{H};\mathcal{S}) \vDash \langle t \parallel^+ S\rangle\right) \quad \rightsquigarrow^* \quad \left((\mathcal{H};\mathcal{S},(\beta^+ := \mathbb{S}),(x^+ := \mathbb{V},\alpha^+ := \beta),(y^+ := (x,x))) \vDash \langle y \parallel^+ \alpha\rangle\right)$$

其中 S是一个任意的正类型余值，并且我们有 $(\mathrm{H};\ \mathrm{S})\ \vDash\ S\ \Downarrow_s\ \mathbb{S}$和 $(\mathrm{H};\ \mathrm{S},(\beta^+\ :=\ \mathbb{S}))\ \vDash\ V\ \Downarrow_v\ \mathbb{V}$。也就是说，返回值 y取决于最后一个栈帧，尤其取决于绑定 $y^+ := (x,x)$；该绑定在求值开始时尚未分配。因此，在执行到此处时，栈无法缩减并返回到 S。

特别是，这意味着，如果没有完整的二等限制，当函数可以返回非模态值时，我们进行分配的栈就不对应于调用栈。一方面，栈化性质是否以及如何能帮助我们改进不加区分的堆分配，仍有待研究。另一方面，这并不影响我们的论点：二等延续的元理论源自 S4 序列演算的元理论——而后者或许允许对不逃逸值这一概念作出更宽松的刻画（正如二等值是借用的一种简化形式）。

5 正确性

现在，我们将证明这种类机器语义相对于操作语义 $\triangleright$的正确性。

我们首先在图 5 中定义环境的类型关系。首先固定一个模态类型 R，它在本文中始终不变，并将作为返回类型以及 tp的类型。类型环境的定义与环境几乎相同，只是将 $x^{\varepsilon} := \mathbb{V}$ 替换为 $x :\ A_{\varepsilon}$，等等。我们将这样的类型环境记作 Σ或 $(\Sigma_{\mathcal{H}};\Sigma_{\mathcal{S}})$。我们可以为类型环境关联一个类型上下文：将 $\mathcal{S}$ 中的变量放入 Γ，将余变量放入 Δ，并将 $\mathcal{H}$ 中的变量放入 Θ。我们还总是将$tp :\ R$添加到 Δ。例如，

$$[\![((x:A),(y:B);(\alpha:C,\beta:D),(z:E))]\!] = z : E \mid x : A, y : B \vdash tp : R, \alpha : C, \beta : B$$

请注意，我们的规则确保，在栈上分配的值或栈只包含指向堆和之前分配的变量的引用。引理 3.1 保证堆上的值只引用堆。

我们列举该类型系统的一些有用性质：

引理 5.1（进展性）。若 $\mathcal{M} :\ \Sigma$，且 c 在环境 $[\![\Sigma]\!]$ 中类型正确，则 $c = \langle \mathbb{V} \parallel^+ tp\rangle$，或者存在 c' 和 $\mathcal{M}'$，使得 $(\mathcal{M}\ \vDash c)\ \rightsquigarrow (\mathcal{M}' \vDash c')$。

引理 5.2（主体归约）。如果$\mathcal{M} :\ \Sigma\ and\ c$ 在环境 $[\![\Sigma]\!]$ 中类型正确，且$(\mathcal{M}\ \vDash c)\ \rightsquigarrow (\mathcal{M}' \vDash c')$，那么$\mathcal{M}'$ 可定型，类型环境可以是 Σ' 对 Σ 的扩展或限制，并且 c' 在环境 Σ' 中类型正确。

$$\frac{}{(\diamond;\diamond) : (\diamond;\diamond)}$$

$$\frac{(\mathcal{H};\mathcal{S}) : (\Sigma_{\mathcal{H}};\Sigma_{\mathcal{S}}) \qquad \Gamma_{\Box} \nmid \Theta \vdash \mathbb{V} : A_{\varepsilon};\diamond \qquad [\![(\mathcal{H};\mathcal{S})]\!] = (\Gamma \nmid \Theta \vdash \Delta)}{(\mathcal{H}, (x^{\varepsilon} \coloneqq \mathbb{V});\mathcal{S}) : (\Sigma_{\mathcal{H}}, (x : A_{\varepsilon});\Sigma_{\mathcal{S}})}$$

$$\frac{(\mathcal{H};\mathcal{S}) : (\Sigma_{\mathcal{H}};\Sigma_{\mathcal{S}}) \qquad \Gamma \nmid \Theta \vdash \mathbb{V} : A_{\pm};\Delta \qquad [\![(\mathcal{H};\mathcal{S})]\!] = (\Gamma \nmid \Theta \vdash \Delta)}{(\mathcal{H};\mathcal{S}, (x^{\pm} \coloneqq \mathbb{V})) : (\Sigma_{\mathcal{H}};\Sigma_{\mathcal{S}}, (x : A_{\pm}))}$$

$$\frac{(\mathcal{H};\mathcal{S}) : (\Sigma_{\mathcal{H}};\Sigma_{\mathcal{S}}) \qquad \begin{array}{c}\Gamma \nmid \Theta;\mathbb{S} : A_{\varepsilon} \vdash \Delta \\ \Gamma \nmid \Theta;\mathbb{S}' : B_{\varepsilon'} \vdash \Delta\end{array} \qquad [\![(\mathcal{H};\mathcal{S})]\!] = (\Gamma \nmid \Theta \vdash \Delta)}{(\mathcal{H};\mathcal{S}, (\alpha^{\varepsilon} \coloneqq \mathbb{S}, \beta^{\varepsilon'} \coloneqq \mathbb{S}')) : (\Sigma_{\mathcal{H}};\Sigma_{\mathcal{S}}, (\alpha : A_{\varepsilon}, \beta : B_{\varepsilon'}))}$$

etc.

图 5. 环境的类型判定

证明概要。对推导 $(\mathcal{M} \vDash c) \rightsquigarrow (\mathcal{M}' \vDash c')$进行分类讨论。对于规则*Eval*□对应的情形，我们使用引理 3.1 证明 V不包含对 $\mathcal{S}$ 的引用；然后，如果 $\mathcal{M}(\alpha) =^{\Box} \mathbb{S}$ and$\mathcal{M}$类型正确，那么 $\mathbb{S}$必须在 $[\![\mathcal{M}|_{@\alpha}]\!]$ 中具有类型。因此 $\langle V \Vdash^{\Box} \mathbb{S}\rangle$在 $[\![\mathcal{M}|_{@\alpha}]\!]$中类型正确。 □

引理 5.3. 若$\mathcal{M} : \Sigma$，则$\mathcal{M}$ 定义了一个从 $[\![\Sigma]\!]$ 到 $[\![(\diamond;\diamond)]\!]$的类型化替换，记作$[\mathcal{M}]$。

引理 5.4. 若$\mathcal{M} : \Sigma$，且 V在环境 $[\![\Sigma]\!]$ 中具有类型 A，并且$\mathcal{M} \vDash V \Downarrow_v \mathbb{V}$，则 $V[\mathcal{M}] =\mathbb{V}[\mathcal{M}]$； $\mathbb{S}$也同样如此。

引理 5.5. 若$\mathcal{M} : \Sigma$，且 c 在环境 $[\![\Sigma]\!]$ 中类型正确，并且$(\mathcal{M} \vDash c) \rightsquigarrow (\mathcal{M}' \vDash c')$，则$c[\mathcal{M}] \rhd^{\leq 1} c'[\mathcal{M}']$。此外，不存在能够导出一个等式的 $\rightsquigarrow$无限序列。

现在，若 c是闭合的，即在环境 $[\![(\diamond;\diamond)]\!] = (\diamond \nmid \diamond \vdash tp : R)$中定义的，依据定理 3.5，存在唯一的 V，其类型为 R，使得 $c \rhd^{*} \langle V \Vdash^{\dagger} tp\rangle$。我们将证明如下结论：

定理 5.6（求值）。存在 V'和 $\mathcal{H}$，使得 $((\diamond;\diamond) \vDash c) \rightsquigarrow^{*} ((\mathcal{H};\mathcal{S}) \vDash \langle V' \Vdash^{\Box} tp\rangle)$，$V'$均不包含对 $\mathcal{S}$的引用，且 $V'[\mathcal{H}] = V$。

这证明了我们的语义是可靠的。

证明概要。借助此前证明的引理，我们可以证明 $\rightsquigarrow$ 会正规化。此外，最终得到的命令必定具有 $\langle V' \Vdash^{\Box} tp\rangle$ 的形式，处于某个环境 $\mathcal{M}'$之中。不过，V' 是一个模态类型的值，因此根据引理 $\mathcal{S}'$，它不能包含对其中任何变量或余变量的引用，也不能包含对 tp的引用，参见引理 3.1。□

引用

Nick Benton。1994年。“混合线性与非线性逻辑：证明、项与模型。”载于：《*CSL*论文集》（《计算机科学讲义》）。第933卷。Springer-Verlag。Jean Caspar。2025年8月20日。“混合经典与直觉主义构造性：经典 S4 模态逻辑与二等延续的公式即类型解释。”硕士论文。INRIA，（2025年8月20日）。Youyou Cong、Leo Osvald、Grégory M. Essertel 和 Tiark Rompf。2019年7月26日。“使用延续编译，还是不使用？随便。”《*ACM* 编程语言汇刊》，3，（2019年7月26日），1–28，ICFP，（2019年7月26日）。DOI：10.1145/3341643。

Pierre-Louis Curien、Marcelo Fiore 和 Guillaume Munch-Maccagnoni。2016年1月11日。“效应与资源理论：伴随模型与极化演算。”载于：第43届ACM SIGPLAN-SIGACT编程语言原理年会论文集。POPL '16：第43届ACM SIGPLAN-SIGACT编程语言原理年会。ACM，美国佛罗里达州圣彼得堡，（2016年1月11日 2837 614 44 56 978 1 4503 3549 2 10 1145 2837652），–。ISBN：----。DOI：。/。。Vincent Danos、Jean-Baptiste Joinet 和 Harold Schellinx。1997年。“一种新的解构逻辑：线性逻辑。”《符号逻辑杂志》，62，3，755–807。DOI：10.2307/2275572。Rowan Davies 和 Frank Pfenning。2001年5月。“分阶段计算的模态分析。”《ACM计算机杂志》，48，3，（2001年5月），555–604。DOI：10.1145/382780.382785。Paul Downen、Luke Maurer、Zena M. Ariola 和 Simon Peyton Jones。2016年9月4日。“相继式演算作为编译器中间语言。”《SIGPLAN通知》，51，9，（2016年9月4日），74–88。DOI：10.1145/3022670.2951931。Jean-Yves Girard。1991年11月。“一种新的构造逻辑：经典逻辑。”《计算机科学中的数学结构》，1，3，（1991年11月），255–296。DOI：10.1017/s0960129500001328。Jean-Yves Girard。1993年2月16日。“论逻辑的统一性。”《纯粹与应用逻辑年刊》，59，3，（1993年2月16日），201–217。DOI：10.1016/0168-0072(93)90093-S。库尔特·哥德尔。2001年7月5日。“直觉主义命题演算的一种解释（1933f）。”载于：库尔特·哥德尔：文集，第一卷：1929—1936年著作。Kurt Gödel 等编。牛津大学出版社，牛津、纽约，（7月）。ISBN：----。Timothy G. Griffin。1989年12月1日。“一种公式即类型的控制概念。”载于：第17届ACM SIGPLAN-SIGACT编程语言原理研讨会论文集（POPL '90）。计算机协会，美国纽约州纽约，（1989年12月1日），47–58。ISBN：978-0-89791-343-0。DOI：10.1145/96709.96714。G. A. Kavvos。2017年6月。“模态逻辑的双上下文演算。”载于：2017年第32届ACM/IEEE计算机科学逻辑年会（LICS）。2017年第32届ACM/IEEE计算机科学逻辑年会（LICS）。IEEE，冰岛雷克雅未克，（2017年6月），1–12。ISBN：978-1-5090-3018-7。DOI：10.1109/LICS.2017.8005089。Olivier Laurent。2002年3月。“逻辑中的极化研究。”博士论文。艾克斯-马赛第二大学，（2002年3月）。Luke Maurer、Paul Downen、Zena M. Ariola 和 Simon Peyton Jones。2017年6月。“无延续的编译。”《SIGPLAN通知》，52，6，（2017年6月），482–494。DOI：10.1145/3140587.3062380。J. C. C. McKinsey 和 Alfred Tarski。1948年。“关于 Lewis 与 Heyting 的命题演算的一些定理。”《符号逻辑杂志》，13，1，1–15。DOI：10.2307/2268135。Kenji Miyamoto 和 Atsushi Igarashi。2004年。“安全信息流的模态基础。”载于：计算机安全基础研讨会，187–203。Guillaume Munch-Maccagnoni。2009年。“聚焦与经典可实现性（含附录版本）。”载于：第18届EACSL计算机科学逻辑年会论文集——CSL 09。第5771卷。Springer-Verlag，409。DOI：10.1007/978-3 -642-04027-6 30。_ Guillaume Munch-Maccagnoni。2017年。《关于线性调用-按需传值中Curry风格的注记》。（2017年）。https://inria.hal.science/hal -01528857v3。Guillaume Munch-Maccagnoni。2013年12月10日。“程序与证明的非结合组合之语法与模型。”博士论文。巴黎狄德罗大学（巴黎第七大学），（2013年12月10日）。https://theses.hal.science/tel-00918642。Aleksandar Nanevski、Frank Pfenning 和 Brigitte Pientka。2008年6月。“上下文模态类型理论。”《ACM计算逻辑汇刊》，9，3，第23篇文章，（2008年6月），49页。DOI：10.1145/1352582.1352591。Ichiro Ogata。2000年。“作为CPS演算的构造性经典逻辑。”《计算机科学基础国际期刊》，11，01，89–112。电子版：https://doi.org/10.1142/S0129054100000065。DOI：10.1142/S0129054100000065。Leo Osvald、Grégory Essertel、Xilun Wu、Lilliam I González-Alayón 和 Tiark Rompf。2016年。“绅士化是否过头？用于函数与（共）效应的实惠二等值。”载于：2016年ACM SIGPLAN面向对象编程、系统、语言与应用国际会议论文集。OOPSLA 2016。计算机协会，美国纽约州纽约，234–251。ISBN：978-1-4503-4444-9。DOI：10.1145/2983990.2984009。Philipp Schuster、Marius Müller、Klaus Ostermann 和 Jonathan Immanuel Brachthäuser。2025年4月9日。“将经典相继式演算编译到现成硬件：编译的对偶性。”《ACM 编程语言汇刊》，9，（2025年4月9日），1746–1773，OOPSLA1，（2025年4月9日）。DOI：10.1145/3720507。Guy L. Steele Jr.。1978年5月。“RABBIT：SCHEME编译器。”硕士论文。人工智能实验室，马萨诸塞州剑桥市麻省理工学院，（1978年5月）。技术报告 AI-TR-474。