arXiv:2301.08659v1 [cs.LO] 20 Jan 2023

带有上下文无关会话类型的 System F_{ω}^{μ} *

Diana Costa Andreia Mordido、Diogo Po.cas 和 Vasco T. Vasconcelos

LASIGE，里斯本大学科学学院，葡萄牙{
dfdcosta,afmordido,dmpocas,vmvasconcelos}@ciencias.ulisboa.pt

摘要。我们研究了表达能力逐步增强的类型系统，从 F^{μ}——带有等递归类型的多态 λ 演算扩展——到 $F_{\omega}^{\mu;}$——带有等递归类型和上下文无关会话类型的高阶多态 λ 演算。类型等价性由一种标准双模拟给出，该双模拟定义在一个新颖的类型标记迁移系统之上。我们的系统涵盖了文献中研究的 F_{ω}^{μ} 收缩片段。通过将类型语言中的类型转换为适当计算模型的对象，得出了各种类型语言的类型等价性判定结果：有限状态自动机、简单文法和确定性下推自动机。我们证明，类型语言的一个重要片段中的类型等价性是可判定的。此外，我们提出一种配备了该表达力强大的类型语言的消息传递并发函数式语言，并证明该语言对可类型化进程具有保持性且不会出现运行时错误。

关键词：System F、高阶种类、上下文无关会话类型

1 引言

对 λ-演算进行扩展以纳入日益复杂的类型结F [60]构的研究已十分广泛，并产生了一些重要性得到广泛认可的系统：System F^{μ} [30]、System F_{ω} [36]、SystemF_{ω}^{μ} [16]。理想情况下，我们希望将一份类型结构愿望清单 结合起来，从而构建一个功能极其强大的系统，实现广泛的表达能力。然而，类型的表达能力自然受到其所处宇宙的限制：编程语言。富有表达力的类型系统给编译器带来了其他（表达力较弱的）类型甚至不会暴露的挑战；其中一个例子就是类型等价性检查。

System F 可以通过添加不同的类型构造子来扩展，以描述通信协议。我们分析此类构造子的组合对类型等价问题的影响。为此，我们将会话类型 [42,43,67] 引入 System$F$。会话类型以类型的形式提供详细的协议规范。传统的递归会话类型仅限于

* 本研究得到了葡萄牙科学技术基金会通过 SafeSessions 项目（编号：PTDC/CCI-COM/6453/2020）的资助，以及 LASIGE 研究单位（编号：UIDB/00408/2020 和 UIDP/00408/2020）的资助。

尾递归，因此无法刻画所有轨迹不能用正则语言描述的协议。上下文无关会话类型通过为类型添加顺序组合的概念，克服了这一限制，$T; U$ [3,68]。类型集合与二元运算；构成一个幺半群，其中一种新的类型 Skip充当单位元，而End 充当吸收元。

正则递归类型 $\mu\alpha\colon \mathrm{S}.\&\{\text{Done: End, More: ?Int; } \alpha\}$ 描述了从消费者视角观察到的整数流。它提供 Done 和 More 两种选择：选择 Done 后必须关闭通道，这由类型 End 表示；选择 More 后必须接收一个整数值，随后接收流的剩余部分。类型按种类分类，因此我们知道递归变量 α 属于会话种类（记作 S），因而可以与分号搭配使用。相反，我们可能想要编写一种更具上下文无关风格的类型。类型 $\mu\alpha\colon \mathrm{S}.\&\{\text{Leaf : Skip, Node : } \alpha\text{; ?Int; } \alpha\}$;End 描述了通过通道以类型安全的方式流式传输整数树的协议。Leaf 选项的延续是 Skip，此时不会发生通信，但通道仍保持打开，以便继续组合。Node 选择的延续会接收一棵左子树、一个位于根节点处的整数，以及一棵右子树。无论哪种情况，接收完整棵树后都必须关闭通道，这由末尾的 End 表示。除了交换基本类型的一阶上下文无关会话类型之外，[3,68] 我们可能还对能够交换复杂类型值的高阶会话类型感兴趣 [21]。本文的一个目标是将高阶上下文无关会话类型集成到系统 F^{μ}_{ω} 中。我们希望能够抽象树通道上接收的类型，现在可以通过写作 $\lambda\alpha\colon \mathrm{T}.\mu\beta\colon \mathrm{S}.\&\{\text{Leaf : Skip, Node : } \beta\text{; ?}\alpha\text{; } \beta\}$;End 来实现，其中 T 是函数类型的种类。

Das 等人此前提出了一种对会话类型进行抽象的形式，即通过（嵌套）参数多态。[25,26] 按照 Das 等人的记法，我们可以写出类型方程 $\text{Stream}\langle\alpha\rangle \doteq \&\{\text{Done: End, More: ?}\alpha\text{; Stream}\langle\alpha\rangle\}$，它抽象了流通道上所接收的类型。要用抽象来表示相同的类型，我们可以把 Stream 看作其参数的函数 α，$\text{Stream} \doteq \lambda\alpha\colon \mathrm{T}.\&\{\text{Done: End, More: ?}\alpha\text{; Stream } \alpha\}$；随后，我们可以用 μ-运算符重写 Stream：$\text{Stream} = \lambda\alpha\colon \mathrm{T}.(\mu\beta\colon \mathrm{S}.\&\{\text{Done: End, More: ?}\alpha.\beta\})$。Das 等人证明，基于正则会话类型的参数化类型定义，其表达能力严格强于上下文无关会话类型。在某种程度上，这一类比为我们的方法提供了指引：如果通过参数多态为正则类型添加抽象会产生嵌套类型，那么通过类型级 λ-运算符为上下文无关类型添加抽象究竟意味着什么？本文将分析 System F^{μ} 的若干扩展，最终在上下文无关会话类型中加入λ-抽象。

我们的研究重点之一必然是对类型等价问题的分析。关于递归参数化类型上的这一问题是否可判定，争论可追溯到20世纪70年代 [18,63]。尽管参数化（嵌套）会话类型和上下文无关会话类型的类型等价问题是可判定的，但对于上下文无关类型上的抽象进行组合的情形，这一问题可能不再可判定。事实上，这项分析是一次有趣的探索，旨在更好地

理解在存在顺序组合时高阶多态递归所起的作用，以及将抽象与任意递归（而非尾递归）结合所带来的收益（和损失）。

归根结底，可判定性并不足以充分衡量类型系统的实用性。我们寻求能够集成到编译器中的类型系统。因此，我们关注类型等价性检查算法。仅 F_ω^μ 的等价性，其表达能力就至少不亚于确定性下推自动机。若将递归限制在单态情形（要求递归变量表示适当类型，即种类为S 或 T，统称为 $*$），则类型等价的复杂度会降低至有限状态自动机等价性的复杂度。扩展到上下文无关会话类型后，问题会稍微复杂一些。为了获得“良好”的算法，我们将递归限制在单态情形，从而得到类 $F_\omega^{\mu *}$、 $F^{\mu *;\,\omega}$。此时， $F^{\mu *;\,\omega}$ 的类型相等性问题可转化为简单文法的等价性问题，而后者仍然是可判定的 [6,33]。由于 $F^{\mu *;\,\omega}$ 包含 $F_\omega^{\mu *}$，我们关于类型等价可判定性的证明，可作为 Cai 等人 [16] （限于收缩类型）所给证明的另一种证明。

高阶多态性使我们能够定义类型算子，并将各种（会话类型）构造内化，否则这些构造只能作为内置构造子提供。这样，我们就能够内化基本的会话类型构造子，例如顺序组合；以及 对偶类型算子（它会反转参与方之间的通信方向）。对偶性通常被视为外部宏。Gay 等人 [34] 探讨了处理对偶算子的不同方式，且均处于单态设定中。在多态性存在的情况下，若不引入余变量，就无法完全消除对偶算子。内化提供了一种简洁得多的解决方案。

由于存在顺序组合，正则树不足以作为表示类型的模型（第 2 节中的类型 TreeC a 就是一个例子）。将 System F_ω^μ 与上下文无关会话类型结合起来时，主要的技术挑战在于确保所得模型仍可用简单文法表示，从而能够通过实用算法判定类型等价。难点在于重命名约束变量。对于无限类型，无论是用新鲜变量重命名，还是使用 de Bruijn 指数，都可能产生无限多个不同的变量，这使得构造简单文法根本不可能。例如，考虑类型 $\lambda\alpha\colon \mathrm{T}.\mu\,\gamma\colon \mathrm{T}.\,\lambda\beta\colon \mathrm{T}.\alpha \to \gamma$，它表示无限类型 $\lambda\alpha\colon \mathrm{T}.\lambda\beta\colon \mathrm{T}.\alpha \to \lambda\beta\colon \mathrm{T}.\alpha \to \lambda\beta\colon \mathrm{T}\ldots$。每一步都使用一个新鲜变量来重命名该类型，会得到形如 $\lambda\upsilon_1\colon \mathrm{T}.\lambda\upsilon_2\colon \mathrm{T}.\upsilon_1 \to \lambda\upsilon_3\colon \mathrm{T}.\upsilon_1 \to \lambda\upsilon_4\colon \mathrm{T}\ldots$ 的类型，需要无限多个变量。同样，de Bruijn 指数 [28] 会得到形如 $\lambda_{\mathrm{T}}\lambda_{\mathrm{T}}1 \to \lambda_{\mathrm{T}}2 \to \lambda_{\mathrm{T}}3 \to \ldots$ 的类型，需要无限多个自然数索引。因此，我们引入最小重命名，尽可能少用变量名（参见 Gauthier 和 Pottier [30]）。这样只需有限多个终结符，就能将类型转换为简单文法。

类型语言存在于项语言中，我们提出一种用于处理 F_ω^μ类型的项语言。基于 Almeida 等人 [2], ，我们引入一种消息传递

并发编程语言。如果类型等价可判定，那么类型检查也可判定；特别是，对于 $F^{\mu *;\,\omega}$，类型检查是可判定的。

本文的主要贡献如下。

– 将（高阶）上下文无关会话类型集成到系统 F^{μ}_{ω}中，并将其称为 $F^{\mu;}_{\omega}$。– 通过标记迁移系统对类型等价性进行语义定义。– 确定 System $F^{\mu;}_{\omega}$的一个合适片段，使得类型等价性可归约为简单文法的双模拟等价。– 证明完整 System $F^{\mu;}_{\omega}$上的类型等价性至少与确定性下推自动机的双模拟等价一样难。– 首次在类型语言中内化 对偶 类型算子。– 一种用于消费 $F^{\mu;}_{\omega}$ 类型的项语言及其配套的元理论。

本文介绍的类型系统结合了三种构造：会话类型的顺序组合、通过类型级抽象和应用实现的高阶种类，以及高阶递归。在我们的工作之前，Almeida 等人提出了一个系统 [6]，该系统包含顺序组合和（一阶）递归，但不包含高阶种类。Cai 等人也提出了一个系统 [16]，该系统包含高阶种类和高阶递归，但不包含顺序组合。我们的系统首次纳入了这三种构造。尽管其中一些结果是在已有成果基础上的渐进发展，并推广了文献中的结果，但主要的技术挑战在于理解这些结果不再成立的边界。例如，仅仅将高阶种类加入 Almeida 等人的系统并不可行，因为我们必须密切关注变量名，确保类型等价性在 α-转换（约束变量的重命名）下保持不变。为此，我们受 Gauthier 和 Pottier [30]的启发，定义了一种新颖的重命名概念。同样，仅仅将顺序组合加入 Cai 等人的系统也不可行，因为有限状态自动机（或正则树）不足以刻画新类型系统的表达能力，即使 将其限制为一阶递归也是如此。因此，我们转而研究表达能力更强的简单文法框架，并引入一种从类型到简单文法的词的转换。

本文其余部分安排如下。下一节阐述类型语言的动机，并通过一个示例介绍项语言。第 3节介绍 System $F^{\mu;}_{\omega}$，第 4 节讨论类型等价性，第 5 节证明，对于类型语言的一个片段，类型等价性是可判定的。第 6节介绍项语言及其元理论。第 7 节讨论相关工作，第 8 节总结全文，并指出未来工作的方向。

2 动机

我们的目标是研究结合了等递归、高阶多态性和高阶上下文无关会话类型的类型系统，以及将它们纳入编程语言的方式。

$$\langle\!|\,|\!\rangle ::= \{\} \mid \langle\rangle \qquad \sharp ::= ? \mid !\qquad \odot ::= \& \mid \oplus$$

$$\begin{array}{lll}
T ::= T \to T \mid \langle\!|\overline{l_i : T_i}|\!\rangle \mid \forall\alpha{:}\kappa.T \mid \mu\,\alpha{:}\kappa.T \mid \alpha & (F^{\mu}) \\
T ::= (F^{\mu}) \mid \sharp T.T \mid \odot\{\overline{l_i : T_i}\} \mid \mathsf{End} & (F^{\mu\cdot}) \\
T ::= (F^{\mu}) \mid \sharp T \mid \odot\{\overline{l_i : T_i}\} \mid \mathsf{End} \mid T;T \mid \mathsf{Skip} & (F^{\mu;}) \\
T ::= (F^{M}) \mid \lambda\alpha{:}\kappa.T \mid T\,T & (F^{M}_{\omega}),\quad M ::= \mu, \mu\cdot, \mu;
\end{array}$$

图 1：六种 F-系统。

对 F 的扩展。图 1 通过提出六种不同的类型语法，逐步引出 $F^{\mu;}_{\omega}$，以此说明这一构造。初始系统 F^{μ} 包含一些广为人知的基本类型算子[57]：函数 $T \to U$、记录 $\{l_i : T_i\}$ 以及变体 $\langle l_i : T_i\rangle$。类型 Unit 是 $\{\}$（即空记录）的简写；我们可以认为，Unit 代指任意标量类型，例如 Int 和 Bool。我们还引入了变量名 α、类型量化 $\forall\alpha{:}\kappa.T$以及递归 $\mu\,\alpha{:}\kappa.T$。为了控制类型形成，所有变量绑定都必须带有某种种类 κ；即便在初始系统 F^{μ} 中，我们也只使用函数种类 T。

接下来，我们以 F^{μ} 为基础，考虑（正则、尾递归的）会话类型；我们用 $F^{\mu\cdot}$ 表示由此得到的系统。例如，?Int.!Bool.End 是一种通道端点类型，表示接收一个整数、发送一个布尔值，然后终止。此时，我们引入会话类型的种类 S，以限制会话类型与函数类型的组合方式。例如，良构类型 $?T.U$ 属于种类 S，并要求 U 也属于种类 S（而 T 可以属于种类 $*$，即 S 或 T）。无限会话类型的一个例子是 $\mu\,\alpha{:}\ \mathrm{S}.!\mathsf{Int}.\alpha$，它会不断输出整数值。再来看一个更复杂的例子：类型 IntStream $= \mu\,\alpha{:}\ \mathrm{S}.\&\{\mathsf{Done}\colon \mathsf{End}, \mathsf{More}\colon ?\mathsf{Int}.\alpha\}$，它描述了一个用于接收（有限或无限）整数值流的通道端点。选择 Done 后，通信便结束。

我们构造的下一步将带我们进入上下文无关会话类型；由此得到的系统记为 $F^{\mu;}$。我们引入一个用于顺序组合的新构造 $T; U$，以及一个新类型 Skip，它充当顺序组合[68]的单位元。消息构造子现在是一元的（$?T$ 和 $!T$），而非二元的。在系统 $F^{\mu;}$ 中，我们区分传统的 End 类型和 Skip 类型。这两种类型的行为不同：End 会终止通道，而 Skip 则允许继续通信。由于顺序组合采用幺半群语义，上下文无关会话类型的类型等式更加微妙。它可由以下公理推导得出：

$$\begin{array}{rl}
\mathsf{Skip}; T \sim T & \text{Neutral element} \\
\mathsf{End}; T \sim \mathsf{End} & \text{Absorbing element} \\
(T;U);V \sim T;(U;V) & \text{Associativity} \\
\odot\{\overline{l_i : T_i}\};U \sim \odot\{\overline{l_i : T_i;U}\} & \text{Distributivity}
\end{array} \tag{1}$$

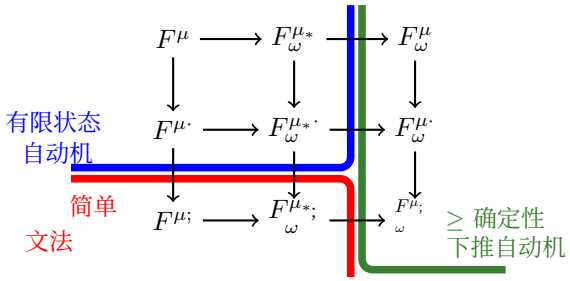

图 2：本文中主要类型类别之间的关系（箭头表示严格包含）。

尽管 $F^{\mu\cdot}$ 的语法并未正式包含在 F^{μ}的语法中，但我们可以通过将$\sharp T.U$映射到 $\sharp T$中；U，将递归会话类型嵌入上下文无关会话类型。众所周知，上下文无关会话类型具有更高的计算表达能力：虽然 F^{μ}和 $F^{\mu\cdot}$可以通过有限状态自动机表示，但 $F^{\mu;}$只能通过简单文法表示 [6,33]。

为了完成我们的构造，我们引入类型抽象 $\lambda\alpha\colon\ \kappa.T$ 以及类型应用 $T\ U$。同样，类型抽象会绑定一个必须具有种类的变量。现在，种类可以是高阶的$\kappa \Rightarrow \kappa'$。对于三个系统 F^{μ}, $F^{\mu\cdot}$, F^{μ} F^{μ} ；我们分别得到其高阶版本 F^{μ}_{ω}, $F^{\mu\cdot;\,\omega}_{\omega}$（均记作 F^{M}_{ω}）。例如，在 System $F^{\mu\cdot}_{\omega}$ 中，我们可以指定通道，用于接收任意（但固定）类型的值序列（有限或无限），

$$\mathsf{Stream} = \lambda\alpha\colon \mathrm{T}.(\mu\,\beta\colon \mathrm{S}.\,\&\{\mathsf{Done}\colon \mathsf{End}, \mathsf{More}\colon ?\alpha.\beta\})$$

其中 α 可以实例化为所需的类型；特别是，StreamInt 等价于前文提到的IntStream。

事实证明，一般高阶系统的表达能力 F^{M}_{ω}对于实际应用来说过于强大。即使是最简单的情形 F^{μ}_{ω}，其表达能力也至少与确定性下推自动机（或等价地，第一类文法）相当，而已知的等价性算法因难以实际应用而臭名昭著。所谓难以实际应用，是指尽管已有可判定性的证明（由 S´enizergues 提出，[61], 后由 Stirling 和 Jancar 改进），[46,65] 但其底层算法相当复杂。据我们所知，目前没有实用的算法实现可以判定确定性下推自动机的等价性。这主要是因为多态递归可以通过高阶 μ递归算子编码（第5节末尾给出了一个例子）。因此，限制递归算子的种类 κ 是有意义的$\mu\,\alpha\colon\ \kappa.T$。我们用记号 μ_{*}表示仅使用 $*$种类递归所写成的类型子类，即 $\mu\,\alpha\colon\ \mathrm{T}.T$或 $\mu\,\alpha\colon\ \mathrm{S}.T$。

图2 概述了本文中各类类型之间的主要关系。首先，我们得到一个格，其表达能力沿着向下的方向（从函数式到会话类型，再到上下文无关会话类型）和向右的方向（从简单多态到具有单态

递归的高阶多态，再到具有任意递归的高阶多态）逐渐增强。其中四类可以用有限状态自动机表示（至 $F^{\mu_{*}\cdot}_{\omega}$）。加入顺序组合（$F^{\mu;}$和 $F^{\mu_{*};\ \omega}$）后，我们仍能用简单文法表示类型。一旦允许任意递归，我们的模型就需要确定性下推自动机的计算能力。

使用 $F^{\mu;}_{\omega}$ 编程。 现在，我们将注意力转向项语言。这是一种消息传递式并发函数式语言，配备了上下文无关会话类型。先来看一个类型为 a 的值流。从读取方的角度看，这种流提供两种选择： Done 和 更多。前一种情况下，交互结束；后一种情况下，读取方读取一个类型为 a 的值，如 ?a 所示，然后递归进行。这就是我们之前见过的流类型，只不过它不会关闭信道端点（类型为 End），而是以类型 Skip 终止，因此可以与其他类型顺序组合。在这段对项语言的非正式介绍中，我们省略类型变量的种类。

```
类型 流 a  =&{Done : Skip, 更多 : ?a ; 流 a}
```

从折叠方的角度看，折叠信道是一种如下形式的类型。我们假设应用的绑定优先级高于分号，也就是说，类型流 a ; !b ; End 应解释为 (流 a) ; !b ; End。

```
类型 折叠 a b  = ?(b  →  a  →  b) ; ?b ; 流 a ; !b ; End
```

此类型的消费者首先接收折叠函数，然后接收起始元素，接着以流的形式接收要折叠的元素，最后输出折叠结果。该类型以 End 结束，因为我们不期望类型 折叠 继续组合。请将 折叠 与传统函数式左折叠的类型进行比较： (b →a → b) → b → 列表 a →b。

现在我们来开发一个函数，用于消费 折叠 通道。语法 x ▷f 表示低优先级的逆函数应用，也就是 x ▷f ▷g = g (f x)。回想一下，单位类型 是空记录类型的另一种记法，{}。

```
foldServer:  ∀a.∀b. 折叠 a b → 单位类型foldServer c  =  令 (f,
c)  =  接收 c 于令 (e, c)  =  接收 c 于 foldS f e c


foldS:  ∀a.∀b.(b →a →b) →b →流 a ;!b; End →单位类型foldS f e c = 匹配 c
with
  {Done c  →  c  ▷ 发送 e  ▷ 关闭, 更多 c  →let (x, c) =接收 c
  in foldS f (f e x) c}
```

函数 foldServer 会消耗通道的初始部分，并将通道的其余部分传递给递归函数 foldS ； foldS 会在累积折叠值的同时消耗整个流。最后，当选择 Done 分支时，折叠值会写入通道，然后关闭通道。一般来说，通道运算符——接收、发送、选择——会以新标识符的形式返回同一通道。通常会重复使用标识符名称——c 如示例所示，例如 let (f, c)= 接收 c——因为它指代同一通道。语法 c ▷...

会隐藏延续通道。外部选择——匹配——的情况也会返回延续（在每个分支中），以便信道端点上的交互可以继续。

现在，我们可以为 foldServer 编写不同的客户端。例如，一个客户端可以根据一对整数值（表示一个区间）生成流；另一个客户端可以根据值的列表生成流；还有一个客户端可以根据二叉树生成流。我们提出另一个客户端。考虑一个以序列化格式交换树的通道的类型 [68]。从读取方的角度看，其多态版本如下：

```
类型 TreeChannel a = TreeC a; End类型 TreeC a =&{ 叶节点: Skip, 节点: TreeC a; ? a; TreeC a}
```

我们在从树通道读取树时，将其转换为流。函数flatten 接收一个树通道和一个流通道（从写入方的角度看，因此使用对偶），并返回流通道中尚未使用的部分。

```
flatten: ∀a。∀c。TreeChannel a →(对偶 Stream a); c →c
```

现在，我们可以编写一个客户端，用来检查树通道中的所有值是否都为正数。

```
allPositive : 树通道 Int →对偶 (折叠 Int Bool) →Boolallpositive tc =
  令 c =发送 (λx:Bool .λy:Int. x&&y > 0) c inlet c = 发送 True c inlet c =展平 [Int] [?Bool ; End ] t c inlet (x, c) = 接收 c in 关闭 c;x
```

客户端在折叠信道上发送一个函数和起始值。然后，它展平给定的树 t，接收折叠后的值并关闭通道。语法 展平[Int] [?Bool;End] 表示项级类型应用。我们要在一个流通道上展平一棵由Int 值构成的树，该通道的延续类型为?Bool;End。延续通道绑定到 c，这样我们就可以进一步接收折叠值，然后关闭通道。语法 e1;e2 表示顺序组合，简写为(λx:单位类型.令 {} = x in e2)e1，前提是 {}（单位类型 值）是线性的，因此必须被消耗。

最后，一个简单的应用会创建一个新的 TreeC 通道，并将其中一端传递给一个线程，该线程会生成一个树通道。随后，它会创建一个 折叠 通道，将一端分配给线程 foldServer，另一端则交给函数 全部为正。原语 分叉接收一个挂起计算（一个惰性求值对象，形式为 λx: 单位类型.e），并创建一个新线程，使其与调用 分叉的线程并行运行。

```
系统: Bool系统 =令 (tr , tw) =新建 TreeC Int 并分叉 (λ_: 单位类型 。 produce tw );令 (fr , fw) =新建 折叠 Int Bool 并分叉 (λ_: 单位类型 。 foldServer fr );
```

$*$	::=		适当类型的种类
	S		会话
	T		函数式
κ	::=		Kind
	$*$		适当类型的种类
	$\kappa \Rightarrow \kappa$		类型算子的种类
T	::=		Type
	ι		类型常量
	α		类型变量
	$\lambda\alpha\colon \kappa.T$		类型级抽象
	$T\,T$		类型级应用

图 3：类型的语法。

ι	::=		类型常量
	$\rightarrow$	$* \Rightarrow * \Rightarrow \mathrm{T}$	箭头类型
	$(\overline{l_i})$	$\overline{* \Rightarrow \mathrm{T}}$	记录，变体类型
	μ_κ	$(\kappa \Rightarrow \kappa) \Rightarrow \kappa$	递归类型
	$\forall_\kappa$	$(\kappa \Rightarrow *) \Rightarrow \mathrm{T}$	全称类型
	Skip	S	skip
	End	S	end
	$\sharp$	$* \Rightarrow \mathrm{S}$	输入，输出
	;	$\mathrm{S} \Rightarrow \mathrm{S} \Rightarrow \mathrm{S}$	序列组合
	$\odot\{\overline{l_i}\}$	$\overline{\mathrm{S} \Rightarrow \mathrm{S}}$	选择运算符
	Dual	$\mathrm{S} \Rightarrow \mathrm{S}$	对偶算子

图 4：类型常量和种类。

全部为正 tr fw

3 种类与类型

本节将详细介绍系统 $F^{\mu;}_{\omega}$，它是系统 F^{μ}_{ω} 的扩展，纳入了高阶上下文无关会话类型。类型的语法见图 3。类型可以是常量 ι （如图 4 所示）、类型变量 α、抽象 $\lambda\alpha\colon \kappa.T$，或应用 $T\,U$。除了将标准会话类型构造器纳入为常量外，系统 $F^{\mu;}_{\omega}$ 还包含对偶（Dual），作为一种类型算子的常量，它将会话类型映射为其对偶。还需注意，$\forall\alpha\colon \kappa.T$ 是 $\forall_\kappa(\lambda\alpha\colon \kappa.T)$ 的语法糖。类似地，$\mu\,\alpha\colon \kappa.T$ 是 $\mu_\kappa(\lambda\alpha\colon \kappa.T)$ 的简写。这样可以简化我们的分析，因为 λ 抽象由此成为唯一的绑定运算符。

会话类型和函数类型之间的区别分别通过种类 S 和 T 来表示。这些是适当类型 $*$ 的种类；我们使用符号 κ 表示适当类型或类型算子的种类，其形式为 $\kappa \Rightarrow \kappa'$。种类上下文 Δ 为类型变量存储种类，使用形式为 $\alpha\colon \kappa$ 的绑定。记号 $\Delta + \alpha\colon \kappa$ 表示对种类上下文 Δ 的更新，定义为 $(\Delta, \alpha\colon \kappa) + \alpha\colon \kappa' = \Delta, \alpha\colon \kappa'$，以及 $\Delta + \alpha\colon \kappa = \Delta, \alpha\colon \kappa$，其中 $\alpha \notin \Delta$。

为了定义类型形成，我们需要一些概念。首先介绍重命名这一概念，该概念改编自 Gauthier 和 Pottier[30]，并展示于图 5。重命名本质上是将类型 T 替换为 T 的最小 α-转换。α-转换是指，$\mathrm{rename}_S(T)$ 对 T 中的约束变量进行重命名。所谓“最小”，是指每个约束变量都被重命名为其可能的最小值。我们假设可使用一个由类型变量构成的可数良序集 $\{\upsilon_1, \ldots, \upsilon_n, \ldots\}$。在 $\mathrm{rename}_S(T)$ 中，参数 S 是一个包含类型变量的集合

类型重命名 $\boxed{\text{重命名}_S(T)}$

$$\text{重命名}_S(\iota) = \iota \quad \text{重命名}_S(\alpha) = \alpha \quad \text{重命名}_S(\lambda\alpha\colon \kappa.T) = \lambda\upsilon\colon \kappa.\text{重命名}_S(T[\upsilon/\alpha]),\ \text{其中}\ \upsilon = \text{first}_S(\lambda\alpha\colon \kappa.T) \quad \text{重命名}_S(T\,U) = \text{重命名}_{S\cup \text{fv}(U)}(T)\ \text{重命名}_S(U)$$

图 5：类型重命名。

不可用于重命名；在重命名过程开始时，S 是空集，因为所有变量都可用。在这种情况下，下标 S 通常会省略。对于 λ 抽象的情况，通过将约束变量重命名为集合 $S \cup \text{fv}(\lambda\alpha\colon \kappa.T)$ 中不包含的最小变量来处理，我们将其记作 $\text{first}_S(\lambda\alpha\colon \kappa.T)$。

重命名使我们能够检查类型抽象 $\lambda\alpha\colon \kappa.T$,$\lambda\beta\colon \kappa.U$ 是否等价。如果两个约束变量 α 和β 都被重命名为同一个变量 υ_j，那么这些类型就是等价的。总之，重命名提供了一种由语法引导的 λ 抽象等价性判定方法，其中约束变量的名称不应产生影响。我们的类型等价概念在重命名下保留 α-转换：如果 T, U 彼此之间可通过 α-转换得到，那么 $\text{rename}(T) = \text{rename}(U)$，尤其是 $\text{rename}(T) \sim \text{rename}(U)$。在第 4 节定义类型等价之后，我们将回到这一点。

我们很容易看出，重命名使用了尽可能少的变量名；例如，$\text{rename}(\lambda\alpha\colon \text{T}.\lambda\beta\colon \text{S}.\beta) = \lambda\upsilon_1\colon \text{T}.\lambda\upsilon_1\colon \text{S}.\upsilon_1$。注意，两个约束变量 α 和 β 都被重命名为 υ_1，即第一个可用于替换的变量。此外，重命名明显违反了许多著作中采用的 Barendregt 变量约定 [11]；例如，$\text{rename}(\upsilon_1\,(\lambda\alpha\colon T.\alpha)) = \upsilon_1\,(\lambda\upsilon_1\colon T.\upsilon_1)$，其中变量 υ_1 在结果类型中既是自由变量又是约束变量。即使重命名违反了变量约定，仍然可以执行替换，而无须采用 Curry 和 Feys 的“即时”重命名方法 [22,40]。当 $\upsilon_1 \neq \upsilon_2$时，我们有

$$(\lambda\upsilon_1\colon \kappa.\lambda\upsilon_2\colon \kappa'.U)\,T \longrightarrow (\lambda\upsilon_2\colon \kappa'.U)[T/\upsilon_1] = \lambda\upsilon_2\colon \kappa'.(U[T/\upsilon_1])$$

因为应用的重命名规则保证 $\upsilon_2 \notin \text{fv}(T)$。否则，若 $\upsilon_1 = \upsilon_2$，则有 $(\lambda\upsilon_1\colon \kappa'.U)[T/\upsilon_1] = \lambda\upsilon_1\colon \kappa'.U$。这说明了为何在重命名过程中要包含集合 S。从现在起，我们假设所有类型都已完成重命名。

接下来介绍类型归约的概念（图6）。除了 β-归约（规则 R-β）之外，该定义还包含顺序组合、递归类型的展开以及对偶T 类型的归约。请注意，规则 R-β 中还进一步调用了重命名，因为 β-归约不保留重命名：考虑重命名后的类型 $(\lambda\upsilon_1\colon \text{T}.\lambda\upsilon_2\colon \text{T}.\upsilon_1 \to \upsilon_2)$ 单位类型。替换 $(\lambda\upsilon_2\colon \text{T}.\upsilon_1 \to \upsilon_2)[\text{单位类型}/\upsilon_1]$ 所得的类型为 $\lambda\upsilon_2\colon \text{T}.\text{单位类型} \to \upsilon_2$，它并未重命名，因此根据我们在第4节中的规则，它不等价于 $\lambda\upsilon_1\colon \text{T}.\text{单位类型} \to \upsilon_1$。

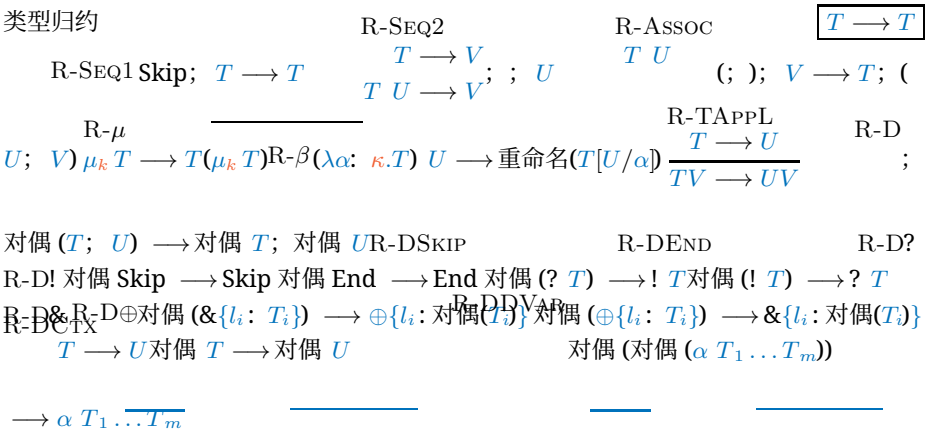

图 6：类型归约。

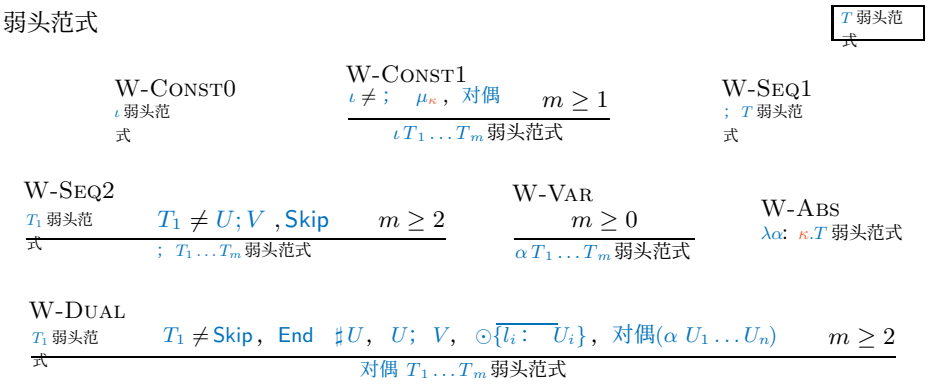

图 7：弱头范式。

得益于我们修改后的规则 R-β，我们在归约下保持重命名的保持性：如果 $T =$重命名(T) 且 $T \longrightarrow U$，那么 $U =$重命名(U)。

我们还需要弱头范式的概念弱头范式，T弱头范式，该概念借鉴自 λ 演算 [11,12]。图7 给出了基于规则的刻画（可用于编译器）。请注意，在规则 W-Seq1 和 W-Seq2 中，顺序组合算子; 采用前缀表示法。然而，对于类型形成（种类判定），事实证明，$;\ T_1 \ldots T_m$ 仅当 $m \leq 2$时才是一个（良构的）类型。

下面的结果表明，基于规则的刻画等价于不可约性。

引理 1. T 弱头范式 当且仅当 $T \not\longrightarrow$。

证明。（⇒）假设 $T \longrightarrow U$ 对于某个 U成立。我们证明其不满足 T 弱头范式。

– Skip； $T \longrightarrow T$ 根据弱头范式规则进行检查，我们得出 Skip； T 不是弱头范式。– T； $U \longrightarrow V$； U 当且仅当 $T \longrightarrow V$。然而，根据假设，鉴于 T会归约， T 不是弱头范式。因此，没有任何弱头范式规则可以应用。所以， T; U 不是弱头范式。–（T； U）； $V \longrightarrow T$； （U； V），根据弱头范式规则进行检查，我们得出（T； U）； V不是弱头范式。– $\mu_\kappa T \longrightarrow T$（$\mu_\kappa T$），根据弱头范式规则进行检查，我们得出 $\mu_k T$不是弱头范式。–（$\lambda\alpha$： $\kappa.T$） $U \longrightarrow$重命名（$T[U/\alpha]$），根据弱头范式规则进行检查，我们得出（$\lambda\alpha$: $\kappa.T$） U不是弱头范式。– $T\ V \longrightarrow U\ V$当且仅当 $T \longrightarrow U$。然而，根据假设，鉴于 T会归约，T 不是弱头范式。因此，根据弱头范式规则进行检查，我们得出 $T\,V$ 不是弱头范式。– T 的情形中，若某个 U 使得 Dual$T \longrightarrow U$ 成立，则会被规则 W-Dual自动排除，因此这些 Dual$T$ 类型都不是弱头范式。

(⇐) 我们必须考察所有满足 $T \not\longrightarrow$ 的类型 T 。下面通过几个例子来说明：

– 常量 ι 不会归约，根据 W-Const0，它们都处于弱头范式。对于 ; T, $\alpha\,T_1$, . . . , T_m($m \geq 0$) 和 $\lambda\alpha$，情况类似：分别使用规则 W-Seq1, W-Var 和 W-Abs。– 对于 T，若 T; U 不会归约，那么根据规则 R-Seq2， T 也不会归约。根据归纳假设， T 处于弱头范式。此外， T ≠Skip T_1; T_2，否则类型 T; U 将分别通过 R-Seq1 或 R-Assoc 归约。这种情况已在 W-Seq2 中涵盖，因此该类型处于弱头范式。– 对于 T，若 Dual(T) 不会归约，则它们均满足以下条件： $T \neq$ Skip, End, T_1; T_2, $\sharp T$, $\odot\{t_i\colon T_i\}$, Dual($\alpha\,T_1 \ldots T_m$) 和 T 不会归约。根据归纳假设，~~T 处~~于弱头范式。 T 的每种情况都由 W-Dual 涵盖。

我们称类型T规范化为类型 U，记作 $T \Downarrow U$，如果从 T经过有限个归约步骤可到达 U弱头范式和 U（注意，任何已经处于弱头范式的项都规范化为其自身）。我们用 T norm 表示存在某个 U使得 $T \Downarrow U$ 。

类型形成 $\boxed{\Delta \vdash T : \kappa}$

$$\text{K-Const}\ \ \Delta \vdash \iota : \kappa_{\iota} \qquad \text{K-Var}\ \ \frac{\alpha : \kappa \in \Delta}{\Delta \vdash \alpha : \kappa} \qquad \text{K-TAbs}\ \ \frac{\Delta, \alpha : \kappa \vdash T : \kappa'}{\Delta \vdash \lambda \alpha : \kappa . T : \kappa \Rightarrow \kappa'} \qquad \text{K-TApp}\ \ \frac{\Delta \vdash T : \kappa \Rightarrow \kappa' \quad \Delta \vdash U : \kappa \quad T\,U\ \mathsf{norm}}{\Delta \vdash T\,U : \kappa'}$$

图 8：类型形成。

例如，假设我们想要规范化类型 $\mu_{\mathrm{S}} T$，其中 T 是类型 $\lambda v_1 :\ _{\mathrm{S}}\!\oplus\{\mathsf{Done}: \mathsf{End},\ \mathsf{更多}: !\ \alpha\};\ \mathsf{对偶}\ v_1$。通过计算从$\mu_{\mathrm{S}} T$出发的所有归约，我们得到序列

$$\mu_{\mathrm{S}} T \longrightarrow T\,(\mu_{\mathrm{S}} T) \longrightarrow \oplus\{\mathsf{Done}: \mathsf{End}, \mathsf{More}: !\alpha\}; \mathsf{Dual}\,(\mu_{\mathrm{S}} T) \not\longrightarrow$$

最后，我们可以证明 $\oplus\{\mathsf{Done}: \mathsf{End},\ \mathsf{更多}: !\ \alpha\};\ \mathsf{对偶}\ (\mu_{\mathrm{S}} T)$ 弱头范式：

$$\dfrac{\dfrac{}{\oplus\{\mathsf{Done}: \mathsf{End}, \mathsf{More}: !\alpha\}\ \mathsf{whnf}}\ \text{W-Const1}}{\oplus\{\mathsf{Done}: \mathsf{End}, \mathsf{More}: !\alpha\}; \mathsf{Dual}\,(\mu_{\mathrm{S}} T)\ \mathsf{whnf}}\ \text{W-Seq2}$$

因此，我们得出结论：$\mu_{\mathrm{S}} T \Downarrow \oplus\{\mathsf{Done}: \mathsf{End}, \mathsf{More}: !\alpha\}; \mathsf{Dual}\,(\mu_{\mathrm{S}} T)$。类似地，我们可以推知，$\mu_{\mathrm{T}}\,(\lambda v_1 :\ _{\mathrm{T}}\!. v_1)$、$\mu_{\mathrm{S}}\,(\lambda v_1 :\ _{\mathrm{S}}\!\mathsf{Skip};\ v_1)$ 和 $\mu_{\mathrm{S}}\,(\lambda v_1 :\ _{\mathrm{S}}\!\mathsf{Dual}\ v_1)$ 都是非规范化表达式的例子。

有了正规化之后，我们便可以引入类型形成，其规则见图8。规则 K-Const 引入常量作为类型，其种类与图4中的种类相同。规则 K-Var 从上下文 Δ中读取类型变量的种类。抽象 $\lambda \alpha :\ \kappa . T$ 是一个种类为 $\kappa \Rightarrow \kappa'$ 的良构类型，当且仅当 T在上下文 Δ 中（该上下文已加入条目 $\alpha :\ \kappa$；规则 K-TAbs）是良构的。之所以需要更新上下文，是因为我们处理的是经过重命名的类型，而且同一个类型变量可能在嵌套抽象中以不同的种类出现。

直到规则 K-TApp，我们才会看到关于类型正规化的条件。这是标准做法，也类似于收缩性条件。其目的是排除那些无限规约、始终无法到达弱头范式的类型。

定理 1。设$\Delta \vdash T :\ \kappa$。

保持性。若 $T \longrightarrow U$，则 $\Delta \vdash U :\ \kappa$。合流性。若$T \longrightarrow U$ 且 $T \longrightarrow V$，则 $U \longrightarrow^{*} W$ 且 $V \longrightarrow^{*} W$。弱规范化。 $T \Downarrow U$ 对某个 U成立。此外，若 $T \Downarrow V$，则 $U = V$。

证明。保持性：对 $T \longrightarrow U$ 进行规则归纳。让我们检查一些会归约的（良构）类型 T，以作说明：

– $\mathsf{Skip};\ U \longrightarrow U$；根据假设 $\Delta \vdash T :\ \ \kappa$ 可推出 $\kappa = {}_{\mathrm{S}}$ 且 $\Delta \vdash U :\ \ {}_{\mathrm{S}}$。

– U_1; $U_2 \longrightarrow V$；仅当 $U_1 \longrightarrow V$；根据假设 $\Delta \vdash T:\ \kappa$可推出 $\kappa = \mathrm{S}$，$\Delta \vdash U_1:\ \mathrm{S}$且 $\Delta \vdash U_2:\ \mathrm{S}$。根据假设，$\Delta \vdash V:\ \mathrm{S}$。因此，$\Delta \vdash V;\ U_2:\ \mathrm{S}$。– $\mu_k U \longrightarrow U\ (\mu_k U)$；根据假设 $\Delta \vdash T:\ \kappa$，且由于 $\Delta \vdash \mu_\kappa:\ (\kappa \Rightarrow \kappa) \Rightarrow \kappa$，可推出 $\Delta \vdash U:\ \kappa \Rightarrow \kappa$。因此，$\Delta \vdash U\ (\mu_\kappa U):\ \kappa$。

合流性：对从 T 出发的各种可能归约进行分类讨论。我们简要说明一个简单的情形。(Skip; T); $U \longrightarrow T$; U(经由 R-Seq2) 和 (Skip; T); $U \longrightarrow$Skip; (T; U) (经由 R-Assoc)。不过，后者还可经由 R-Seq1 再次归约：Skip; (T; U) $\longrightarrow T$; U。

弱规范化：对 T 进行分类讨论。若 $T = \iota$、α 或 $\lambda\alpha:\ \kappa.U$，则根据规则 W-Const0、W-Var、W-Abs，T 分别处于弱头范式。因此根据 N-Whnf，$T \Downarrow T$。若 $T = U\,V$，则根据 K-TApp UV 规范化，因此根据定义，存在某个 T' 使得 $T \Downarrow T'$。第二部分由合流性得出。

最后，我们得出了本节的主要可判定性结果。在其证明中，我们利用了这样一个事实：递归仅限于种类 $*$，以限制 $\mu_* U$ 正规化时可能出现的形如 T 的子表达式。

定理 2（类型形成的可判定性）。$\Delta \vdash T:\ \kappa$ 对于 $F^{\mu_*;\omega}$ 中的类型是可判定的。

证明。见附录 A。

4 类型等价

本节介绍类型双模拟，作为类型等价性的概念。我们在所有类型构成的空间上定义一个标记迁移系统（LTS），并记作$T \xrightarrow{a} U$ 以表示 T 通过标签 a 迁移至 U。标签的文法和标记迁移系统规则见图9。

如果 T 不是弱头范式，那么我们必须将其规范化为某个类型 U，使得 T 与 U 具有相同的迁移（规则 L-Red）。否则，如果T 是弱头范式，则可通过查看 T 对应的规则，立即推导出其迁移，如下所示。如果 T 是变量，则使用规则 L-Var1 （其中$m = 0$）。如果 T 是常量（但不是Skip），则使用规则 L-Const。注意，如果 T 是单独的Skip，它没有任何迁移。如果 T 是抽象，则使用规则 L-Abs。

如果 T 是一个应用，那么我们需要检查头部内部。我们将 T 写作 $T_0\ T_1 \ldots T_m$，其中 $m \geq 1$，这里 T_0 不是一个应用，并检查 T_0。如果 T_0 是一个变量，则使用规则 L-Var1 和 L-Var2。如果 T_0 是常量 $\rightarrow$、$\forall_\kappa$、$\odot\{\overline{l_i}\}$ 或 $\langle\overline{l_i}\rangle$ 之一，则使用规则 L-ConstApp。请注意，如果 T_0 是一个抽象或 μ_κ，那么 T 不是弱头范式，必须对其进行规范化。如果 T_0 是 $\sharp$，则使用规则 L-Msg1 和 L-Msg2。如果 T_0 是对偶，那么要使 T 良构且处于弱头范式，唯一的情况是 $m = 1$ 且 T_1 是 α 或 $\alpha\ U_1 \ldots U_m$；此时我们使用规则 L-DualVar1 和 L-DualVar2。

如果 T_0 是 ; ，我们需要对 T_1进行额外的分类讨论。如果 $m = 1$，使用规则L-Seq1。否则，$m = 2$ 根据种类判定即可推出。如果 T_1 是变量，则使用规则 L-VarSeq1

$a ::= \alpha_i \mid \iota_i \mid \lambda\alpha\colon \kappa \qquad (i \geq 0, \iota \neq \mathsf{Skip})$ 迁移标签

标记迁移系统 $\boxed{T \xrightarrow{a} U}$

L-Red
$$\frac{T \longrightarrow U \quad U \xrightarrow{a} V}{T \xrightarrow{a} V}$$

L-Var1
$$\frac{m \geq 0}{\alpha\ T_1 \ldots T_m \xrightarrow{\alpha_0} \mathsf{Skip}}$$

L-Var2
$$\frac{1 \leq j \leq m}{\alpha\ T_1 \ldots T_m \xrightarrow{\alpha_j} T_j}$$

L-Const
$$\frac{\iota \neq \mathsf{Skip}}{\iota \xrightarrow{\iota_0} \mathsf{Skip}}$$

L-ConstApp
$$\frac{\iota = \rightarrow,\ \forall_\kappa,\ \odot\{\overline{\ell_i}\},\ (\overline{\ell_i}) \qquad 1 \leq j \leq m}{\iota\ T_1 \ldots T_m \xrightarrow{\iota_j} T_j}$$

L-Abs $\lambda\alpha\colon \kappa.T \xrightarrow{\lambda\alpha\colon\kappa} T$

L-Msg1 $\sharp T \xrightarrow{\sharp_1} T$

L-Msg2 $\sharp T \xrightarrow{\sharp_2} \mathsf{Skip}$

L-Seq1 $; T \xrightarrow{;_1} T$

L-VarSeq1
$$\frac{m \geq 0}{(\alpha\ T_1 \ldots T_m);\ U \xrightarrow{\alpha_0} U}$$

L-VarSeq2
$$\frac{1 \leq j \leq m}{(\alpha\ T_1 \ldots T_m);\ U \xrightarrow{\alpha_j} T_j}$$

L-MsgSeq1 $\sharp T;\ U \xrightarrow{\sharp_1} T$

L-MsgSeq2 $\sharp T;\ U \xrightarrow{\sharp_2} U$

L-ChoiceSeq
$$\frac{1 \leq j \leq m}{\odot\{\overline{\ell_i\colon T_i}\};U \xrightarrow{\odot\{\overline{\ell_i}\}_j} T_j;U}$$

L-EndSeq $\mathsf{End};\ U \xrightarrow{\mathsf{End}} \mathsf{Skip}$

L-DualVar1 对偶$(\alpha\ T_1 \ldots T_m)$ 对偶 $\longrightarrow^{1} \alpha\ T_1 \ldots T_m$

L-DualVar2 对偶 $(\alpha\ T_1 \ldots T_m)$ 对偶 $\longrightarrow^{2} \mathsf{Skip}$

L-DualSeq1 $(\text{对偶}(\alpha\ T_1 \ldots T_m));\ U$ 对偶$\longrightarrow^{1} \alpha\ T_1 \ldots T_m$

L-DualSeq2 $(\text{对偶}(\alpha\ T_1 \ldots T_m));\ U$ 对偶$\longrightarrow^{2} U$

图 9：类型的标记迁移系统 s.

（其中 $m = 0$）。如果 T_1 是常量，则其种类必须为 S，因此它必定是Skip 或 End。如果 T_1 是 Skip，则 T 不是弱正规形式，必须规范化。如果 T_1 是 End，则使用规则 L-EndSeq （End 是一个吸收元，因此End； U只会迁移到 Skip，而不会执行 U）。注意， T_1根据种类判定，不可能是抽象。

如果 T_1 是一个应用，那么同样地，我们将 T_1 写作 $U_0\ U_1 \ldots U_n$，其中 $n \geq 1$，头部 U_0 不是一个应用，并考察 U_0。如果 U_0 是一个变量，则使用规则 L-VarSeq1 和 L-VarSeq2。如果 U_0 是一个常量，由于种类判定，它必须是 ;、 μ_κ、 $\sharp$、$\odot\{\overline{l_i}\}$ 或 Dual 之一。如果 U_0 是 $\sharp$，则使用规则 L-MsgSeq1 和 L-MsgSeq2。如果U_0 是 $\odot\{\overline{l_i}\}$，则使用规则 L-ChoiceSeq。如果 U_0 是 Dual，那么要使 T 良构且处于弱头范式，唯一的可能是 $n = 1$，并且 U_1 是 α 或 $\alpha\ V_1 \ldots V_\ell$；在这种情况下，我们使用规则 L-DualSeq1 和 L-DualSeq2。最后，如果 U_0 是 ;、μ_κ 或一个抽象，则 T 不处于弱正规形式，必须对其进行规范化。

让我们通过一个例子来阐明标记迁移系统规则。考虑以下类型$\lambda v_1\colon$ T.$\mu\, v_2\colon$ S.$\oplus\{$Done： End，更多：！ $v_1\}$； 对偶 v_2，并将其称为 T。显然， T属于

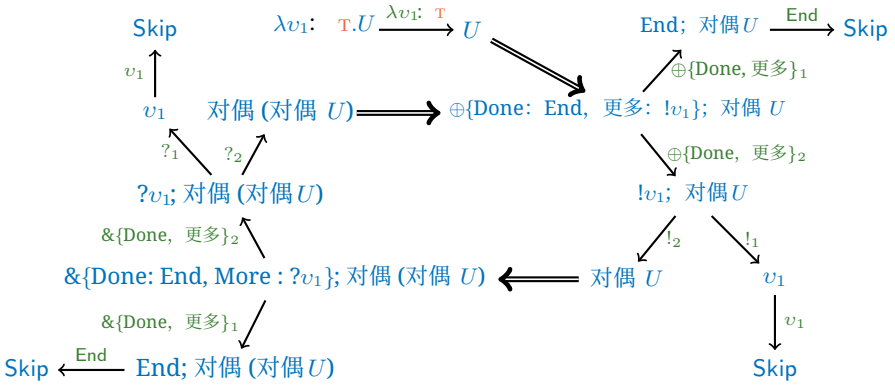

图 10: 类型 λv_1: T.U的标记迁移系统。正规化 $T_1 \Downarrow T_2$表示为 $T_1 \Rightarrow T_2$，而 U 是类型的简写$\mu_{\mathrm{S}}(\lambda v_2$: S.⊕{Done: End, 更多: ! v_1}; 对偶 v_2)。

种类 $\mathrm{T} \Rightarrow \mathrm{S}$。对于给定的函数类型 v_1，该类型指定了一个通道，它交替执行以下操作：提供一个选择并输出一个类型为 v_1 的值；或者选择一个选项并输入一个类型为 v_1 的值。由于对递归变量 v_2应用常量 Dual，极性得以互换。要构造由此类型生成的标记迁移系统（片段），我们先将 T 展开为 λv_1: T.U，其中 U 是类型 $\mu_{\mathrm{S}}(\lambda v_2$: S.⊕{Done: End, More: !$v_1$}; Dual v_2)。注意，U 规范化为 ⊕{Done: End, More: !v_1}; Dual U。该示例的标记迁移系统如图 10 所示。在本例中，只会出现有限多个类型。然而，涉及顺序组合或高阶递归的更复杂示例可能会产生无限迁移图。

根据标记迁移系统规则，我们可以用标准方式定义双模拟的概念。类型上的二元关系 R称为双模拟，如果对于每个 $(T, U) \in R$以及每个迁移标签 a:

1. 若 $T \xrightarrow{a} T'$，则存在 U'使得 $U \xrightarrow{a} U'$且 $(T', U') \in R$；2. 若 $U \xrightarrow{a} U'$，则存在 T'使得 $T \xrightarrow{a} T'$且 $(T', U') \in R$。

如果存在一个双模拟 R，使得 $(T,U) \in R$，我们就称类型 T 和 U 是双模拟的，记作 $T \sim U$。

直观而言，类型等价的概念必须保留并反映类型构造子的语法：例如，类型 $T \to U$ 等价于类型$T' \to U'$当且仅当 T、T'等价，且 U、U'等价。借助双模拟技术，我们通过考察类型上的标记迁移系统来实现这一点：$T \to U$ 有一个标记为 $\to_1$ 、迁移至 T 的迁移，以及一个标记为$\to_2$ 、迁移至 U的迁移。这样一来，$T \to U$ 只能与另一个具有相同两个标签的迁移的类型等价。对于每种类型构造子（$\to$、$\forall_\kappa$、!、?、$\odot\{\overline{\ell_i}\}$等等），我们都有相应的迁移规则。此外，有时需要先对类型进行归约，才能在语法树的

根节点处找到类型构造子。如果 T 归约为 U，那么我们期望 T 和 U 是双模拟的，这由规则 L-Red实现。该规则处理各种归约：由 λ 抽象和应用产生的 β-归约（例如，$(\lambda\alpha\colon \kappa.T)\ U$归约为 $[U/\alpha]T$）；由顺序组合的幺半群结构产生的归约（例如，Skip; T归约为 T）；由将对偶性内化为类型构造子而产生的归约（例如，Dual (! T) 归约为 ?T）；以及由递归产生的归约（例如，$\mu_\kappa\, T$归约为 $T(\mu_\kappa\, T)$）。

让我们来看一些双模拟等价的例子。

– 考虑类型 $T = \alpha$ (!Int) 和 $U = \alpha$ (!Int); Skip。它们是双模拟的，因为二者都存在一条标签为 α_1、指向 !Int 的迁移（分别由规则 L-Var2 和 L-VarSeq2 导出），以及另一条标签为 α_0、指向 Skip 的迁移（分别由规则 L-Var1 和 L-VarSeq1 导出）。一般而言，两个类型 $\alpha\ T$ 和 $\alpha\ U$; V 是双模拟的，当且仅当 $T \sim U$ 且 Skip $\sim V$。这证明了我们为变量设置这些规则的选择是合理的。– 对于 $\sharp T$ 与 $\sharp T$; Skip，以及 Dual $(\alpha\ T_1 \ldots T_m)$ 与 Dual $(\alpha\ T_1 \ldots T_m)$; Skip，也有类似的考虑。– 对于内部选择和外部选择，情况略有不同。类型 $\odot\{\text{Go}\colon T;\ V, \text{Quit}\colon U;\ V\}$ 和 $\odot\{\text{Go}\colon T, \text{Quit}\colon U\};\ V$ 是双模拟的。前者根据规则 L-ConstApp 导出迁移 $\odot\{\text{Go, Quit}\}_1$ 和 $\odot\{\text{Go, Quit}\}_2$，分别指向 T; V 和 U; V。后者导出相同的迁移，但依据的是规则 L-ChoiceSeq。这正合我们所需（顺序组合对选择满足分配律），也证明了我们为这些规则所作的选择是合理的。– 根据我们对规则 L-Const 和 L-EndSeq 的选择，End 与任意类型 End; U 都是双模拟的。– Skip 是弱头范式中唯一没有迁移的良构类型。因此，若 $\Delta \vdash T\colon \kappa$，则 $T \sim$ Skip 当且仅当 $T \Downarrow$ Skip。– 考虑类型 $\lambda\alpha\colon \kappa.T$ 和 $\lambda\beta\colon \kappa.U$。在生成标记迁移系统之前，我们必须先将它们重命名为 $\lambda\upsilon_i\colon \kappa.[\upsilon_i/\alpha]T$ 和 $\lambda\upsilon_j\colon \kappa.[\upsilon_j/\beta]U$。根据规则 L-Abs，重命名后的类型只有在 $\upsilon_i = \upsilon_j$ 时才是双模拟的；此时，我们检查 $[\upsilon_i/\alpha]T$ 和 $[\upsilon_j/\beta]U$ 的等价性。在这里，υ_i 的选择仅取决于 T（而非 U）。这使我们能够定义 T 的标记迁移系统（独立于 U），以及 U 的标记迁移系统（独立于 T），并确保这些类型等价，当且仅当对应的标记迁移系统是双模拟的。如果不使用重命名，我们就必须以某种方式“记住”变量 α、β 应当“关联”起来，这将使独立标记迁移系统的生成成为不可能。这是我们的重命名概念所解决的主要技术难题。

我们的类型等价概念具有自然的性质，并且相对于第 3 节中的归约、正规化和种类判定概念，其行为符合预期。我们可以推导出类型等价规则，并通过有效的语法制导规则来定义另一种余归纳等价概念。换句话说，可以证明 $\sim$ 是满足以下性质 1-15 的最大关系。

引理 2。

1. $T \sim \alpha$ 当且仅当 $T \Downarrow \alpha$。

2. $T \sim \lambda\alpha\colon\ \kappa.U$当且仅当 $T \Downarrow \lambda\alpha\colon\ \kappa.V$存在 V使得 $U \sim V$。
3. $T \sim$End 当且仅当– $T \Downarrow$End;
或– $T \Downarrow$ End; U对于某个 U。

4. 令 ι 为一个不等于End的常量。那么 $T \sim \iota$ 当且仅当 $T \Downarrow \iota$。
5. $T \sim \alpha\ U_1 \ldots U_m$ 当且仅当: – $T \Downarrow \alpha\ U'_1 \ldots U'_m$对于某个 $U'_1, \ldots, U'_m$ 使得 $U_j \sim U'_j$对于每个 j; 或– $T \Downarrow (\alpha\ U'_1 \ldots U'_m); W$对于某个 $U'_1, \ldots, U'_m, V$使得 $U_j \sim U'_j$且Skip $\sim V$。

6. $T \sim U \to V$当且仅当 $T \Downarrow U' \to V'$对于某个 U', V'使得 $U \sim U'$且 $V \sim V'$。
7. $T \sim \forall_\kappa U$当且仅当存在某个 U', 使得 $U \sim U'$。
8. $T \sim (\!|\ell\colon\ U_\ell|_{\ell\in L}$当且仅当 $T \Downarrow (\!|\ell\colon\ U'_\ell|_{\ell\in L}$存在某个 U'_ℓ使得 $U_\ell \sim U'_\ell$对于每个 ℓ。
9. $T \sim \sharp U$ 当且仅当: – $T \Downarrow \sharp U'$存在某个 U'使得 $U \sim U'$;
或– $T \Downarrow \sharp U'$; V存在某个 U', V使得 $U \sim U'$且 Skip $\sim V$。

10. $T \sim \odot\{\ell\colon\ U_\ell\}_{\ell\in L}$ 当且仅当: – $T \Downarrow \odot\{\ell\colon\ U'_\ell\}_{\ell\in L}$存在某个 U'_ℓ使得 $U_\ell \sim U'_\ell$对于每个 ℓ; 或–$T \Downarrow \odot\{\ell\colon\ U'_\ell\}_{\ell\in L}$; V存在某个 U'_ℓ, V使得 $U_\ell \sim U'_\ell$; V 对于每个 ℓ。
11. $T \sim$ End; U 当且仅当– $T \Downarrow$ End; 或– $T \Downarrow$End; U'对于某个 U'。

12. $T \sim \sharp U;\ V$当且仅当: – $T \Downarrow \sharp U'$对于某个 U'使得 $U \sim U'$且 $V \sim$Skip; 或– $T \Downarrow \sharp U';\ V'$对于某个 U', V'使得 $U \sim U'$且 $V \sim V'$。

13. $T \sim \odot\{\ell\colon\ U_\ell\}_{\ell\in L};\ V$当且仅当: – $T \Downarrow \odot\{\ell\colon\ U'_\ell\}_{\ell\in L}$对于某个 U'_ℓ使得 $U_\ell;\ V \sim U'_\ell$对于每个 ℓ; 或– $T \Downarrow \odot\{\ell\colon\ U'_\ell\}_{\ell\in L}; V'$对于某个 U', V'使得 $U_\ell;\ V \sim U'_\ell; V$对于每个 ℓ。
14. $T \sim$对偶 $(\alpha\ U_1 \ldots U_m)$ 当且仅当– $T \Downarrow$对偶 $(\alpha\ U'_1 \ldots U'_m)$对于某个 U'_j使得 $U_j \sim U'_j$ 对于每个 j; 或者 –$T \Downarrow$ 对偶 $(\alpha\ U'_1 \ldots U'_m)$; V 对于某个 U'_j, V 使得 $U_j \sim U'_j$对于每个 j并且Skip $\sim V$。

15. $T \sim$对偶 $(\alpha\ U_1 \ldots U_m)$; V当且仅当 – $T \Downarrow$对偶 $(\alpha\ U'_1 \ldots U'_m)$, 对于某个 U'_j 使得 $U_j \sim U'_j$对于每个 j并且 $V \sim$Skip; 或者 – $T \Downarrow$对偶 $(\alpha\ U'_1 \ldots U'_m)$; V' 对于某个 U'_j, V'使得 $U_j \sim U'_j$对于每个 j并且$V \sim V'$。

接下来, 我们证明类型等价在重命名、归约和规范化下保持不变。

引理 3。

1. 若 $T \sim U$, 则 rename(T) $\sim$rename(U)。2. 若 $T \longrightarrow U$, 则 $T \sim U$。若 $T \Downarrow U$, 则 $T \sim U$。3. 假设 $T \Downarrow T'$and $U \Downarrow U'$。则 $T \sim U$当且仅当 $T' \sim U'$.

证明。第 1 项可通过对 $T \sim U$ 进行余归纳得出，即我们证明集合{(rename(T), rename(U)) : $T \sim U$}是一个双模拟。唯一有趣的情形是 $T = \lambda\alpha{:}\,\kappa.T'$ and $U = \lambda\beta{:}\,\kappa'.U'$。由于 $T \sim U$，我们得到 $\alpha = \beta$, $\kappa = \kappa'$ and $T' \sim U'$。显然，$T \sim U$蕴含 fv(T) =fv(U)，因为在消耗 T, U的过程中，自由变量最终会被“看到”。因此，对于某个 k，first(T) =first(U) = υ_k，且 rename(T) = $\lambda\upsilon_k{:}\,\kappa$.rename($T'[\upsilon_k/\alpha]$)，rename($U$) = $\lambda\upsilon_k{:}\,\kappa$.rename($U'[\upsilon_k/\alpha]$)。最后，我们将 L-Abs 应用于这对(rename(T), rename(U))，得到这对 (rename($T'[\upsilon_k/\alpha]$), rename($U'[\upsilon_k/\alpha]$))。由于 $T' \sim U'$蕴含$T'[\upsilon_k/\alpha] \sim U'[\upsilon_k/\alpha]$，这对元素属于我们的集合。

条目 2 的第一部分很直接：如果 $T \longrightarrow U$，那么根据 L-Red，每个迁移 $T \xrightarrow{a} V$都由一个迁移 $U \xrightarrow{a} V$匹配，反之亦然。要证明条目 2的第二部分，假设 $T \Downarrow U$ 并考虑一个有限归约序列 $T = T_0 \longrightarrow T_1 \longrightarrow \ldots \longrightarrow T_n = U$。在该序列的每一步，我们都有 $T_k \sim T_{k+1}$，因此（由于 $\sim$ 是一个等价关系）可得 $T = T_0 \sim T_n = U$。

条目 3 是条目 2的直接推论。 $T \Downarrow T'$和 $U \Downarrow U'$蕴含 $T \sim T'$以及 $U \sim U'$。由于 $\sim$是一个等价关系，因此我们得到 $T \sim U$当且仅当 $T' \sim U'$。

最后，我们证明引言中关于顺序组合的公理（1）可以从我们的双模拟概念推导出来。

引理 4。

1. Skip; $T \sim T$（单位元）。2. End; $T \sim$ End （吸收元）。3. T (T; U) ; $V \sim T$; (U; V) （结合律）。
4. $\odot\{l_i{:}\ T_i\}$; $U \sim \odot\{l_i{:}\ T_i;\ U\}$（分配律）。

证明。第 1 项由以下观察以及引理 3 的第 2 项可得：Skip;$T \longrightarrow T$ (R-Seq1), to- gether with Item 2 of Lemma 3。第 2 项由以下观察可得：End;T End $\longrightarrow$ Skip (L-EndSeq) 和 End End $\longrightarrow$ Skip (L-Const)。第 3 项由以下观察以及引理 3 的第 2 项可得：(T; U); $V \longrightarrow T$; (U; V) (R-Assoc)。第 4 项由以下观察可得： $\odot\{l_i{:}\ T_i\}$; $U \xrightarrow{\odot\{l_i\}_j} T_j$; U(L-ChoiceSeq) 和 $\odot\{l_i{:}\ T_i;\ U\} \xrightarrow{\odot\{l_i\}_j} T_j$; U(L-ConstApp)。

5 类型等价的可判定性

本节介绍类型等价可判定性的相关结果。我们的方法是将类型转换为某种计算模型中的对象。我们将考察有限状态自动机（适用于 F^{μ}、 $F^{\mu_*}_{\omega}$、 $F^{\mu\cdot}$ 和 $F^{\mu_*\cdot}_{\omega}$ 类型）、简单文法（适用于 $F^{\mu;}$ 和 $F^{\mu_*;}$, ω 类型）以及确定性下推自动机（适用于 F^{μ}_{ω}、 $F^{\mu\cdot}_{\omega}$ 和 $F^{\mu;}_{\omega}$ 类型）。

我们称格雷巴赫范式文法是一个元组 $(\mathcal{T}, \mathcal{N}, \gamma, \mathcal{R})$，其中： $\mathcal{T}$是一个终结符集合，记作 a, b, c； $\mathcal{N}$是一个非终结符集合

符号，记作 X, Y, Z； $\gamma \in \mathcal{N}^*$ 是起始词；并且 $\mathcal{R} \subseteq \mathcal{N} \times \mathcal{T} \times \mathcal{N}^*$是一个产生式集合。如果对于每个非终结符$X$ 和每个终结符 a，至多存在一个产生式 $(X, a, \delta) \in \mathcal{R}$ [51]，则称该文法为简单的。

希腊字母γ 和 δ 表示（可能为空的）非终结符序列。产生式写作 $X \xrightarrow{a} \delta$。我们通过标记迁移系统定义文法的双模拟概念。该系统包含一组状态$\mathcal{N}^*$，这些状态对应于非终结符序列。对于每个产生式 $X \xrightarrow{a} \gamma$ 和每个非终结符序列 δ，都有一个标号转移 $X\delta \xrightarrow{a} \gamma\delta$。我们用 $\approx$ 表示文法的双模拟关系（定义与第4节中的定义类似）。

目前我们关注类 $F^{\mu *;\ \omega}$ 并说明如何将类型 T转换为简单文法（$\mathcal{T}_T$， $\mathcal{N}_T$，$\mathrm{word}(T)$ $\mathcal{R}_T$）。该转换基于函数 word(T)，它将每个类型 T 映射为一个非终结符序列，同时引入新的非终结符和产生式。在我们的构造中，遵循 Costa 等人[21], 的方法，我们使用一个没有产生式的非终结符号，记作 $\bot$，以分隔发送/接收操作（例如$!\ T;\ U$）的两个后继。非终结符序列 word(T) 定义如下。首先考虑 T whnf的情况。

– 对于任意 $m \geq 0$: word($\alpha\ T_1 \ldots T_m$) $= Y$，为 Y一个新的非终结符号及其产生式 $Y \xrightarrow{\alpha_0} \varepsilon$，以及 $Y \xrightarrow{\alpha_j}$ word(T_j)$\bot$，对每个 $1 \leq j \leq m$ 均如此。– word(Skip) $= \varepsilon$。– word(End) $= Y$，为 Y 一个带有单个产生式 YEnd $\longrightarrow \bot$ 的新符号。– 对于任意 $\iota \neq$ Skip、End: word(ι) $= Y$，为 Y一个带有单个产生式 $Y \xrightarrow{\iota} \varepsilon$ 的新非终结符号。– word($\lambda\alpha{:}\ \kappa.T$) $= Y$，为 Y一个带有产生式 $Y \xrightarrow{\lambda\alpha{:}\ \kappa}$word($T$) 的新符号。– 对于任意 $m \geq 1$，且对于 ι 中的某一个 $\rightarrow$、 $\forall_\kappa$、 $\odot\{\overline{l_i}\}$、 $\langle\!|\overline{l_i}|\!\rangle$: word($\iota\ T_1 \cdots T_m$) $= Y$，为每个 $1 \leq j \leq m$ 创建一个带有产生式 $Y \xrightarrow{\iota_j}$ word(T_j) 的新非终结符 Y。– word($\sharp T$) $= Y$，为 Y一个带有产生式 $Y \xrightarrow{\sharp_1}$word(T)$\bot$和$Y \xrightarrow{\sharp_2} \varepsilon$ 的新符号。– word(; T) $= Y$，为 Y一个带有产生式 $Y;\xrightarrow{1}$word(T) 的新符号。– word(T; U) $=$ word(T) word(U)。– word(Dual ($\alpha\ T_1 \ldots T_m$)) $= Y$，为 Y一个带有产生式 YDual$\longrightarrow^1$word($\alpha\ T_1 \ldots T_m$) 和 YDual$\longrightarrow^2 \varepsilon$ 的新符号。

最后，我们处理 T 不处于弱头范式的情况。

– 如果 $T \Downarrow$ Skip，那么词(T) $= \varepsilon$。– 否则，如果 $T \Downarrow U \neq$ Skip，那么词(T) $= Y$ 对于 Y 一个新的非终结符号。令 $Z\delta =$ 词(U)。那么 Y 对每个产生式 $Y \xrightarrow{a} \gamma\delta$ 都有一个产生式 $Z \xrightarrow{a} \gamma$。

在上述构造中，每当遇到一个除Skip以外的弱头范式时，我们都会创建新的符号。换句话说， $\mathcal{N}_T$ 是由 $\bot$ 以及在计算 word(Y) 的过程中创建的所有非终结符T组成的集合。另一个关键洞见是，类型的顺序组合会被翻译为

词的串接：$\mathrm{word}(T_1;T_2;\ldots;T_n) = \mathrm{word}(T_1)\,\mathrm{word}(T_2)\ldots\mathrm{word}(T_n)$。这使得我们的构造能够终止：即使迁移会导向无限多个类型，它们也会按顺序组合算子进行拆分，因此我们只需考虑有限多个子表达式。

为了使构造中的最后一种情况定义良好，即当 $T \Downarrow U \neq \mathsf{Skip}$ 时，我们要求 $\mathrm{word}(U)$ 非空。确实，如果 U 处于弱头范式，那么我们可以通过检查所有情况观察到，$\mathrm{word}(U) = \varepsilon$ 当且仅当 $U = \mathsf{Skip}$。

我们还需要说明 $\mathrm{word}(T)$ 的构造最终会终止。为此，我们跟踪构造过程中访问过的所有类型，并且只有当访问到的类型在语法上不同于此前访问过的所有类型时，才向文法中添加一个新的非终结符 Y。因此，每次重新访问某个类型时，我们都会复用同一个符号 Y 及其相同的产生式。结合以上观察，我们得到以下结果。

引理5。 假设 $T \in F^{\mu *;\,\omega}$。那么，$\mathrm{word}(T)$ 的构造会终止，并生成一个简单文法。

我们用第 2 节中的多态树交换示例来说明上述构造，

类型 `TreeC a` = &{ 叶节点: `Skip`, 节点: TreeC a; ?a ; TreeC a}

该类型写作 $F^{\mu *;\omega}$ 为 $T_0 = \lambda\upsilon_1\colon \mathrm{T}.\mu\,\upsilon_2\colon \mathrm{S}.\&\{\text{叶节点} : \mathsf{Skip},\ \text{节点}:\ \upsilon_2;\ ?\upsilon_1;\ \upsilon_2\}$. 由于 T_0 处于弱头范式，$\mathrm{word}(T_0)$ 会返回一个新符号，我们称之为 X_0。我们还有一个产生式 $X_0 \xrightarrow{\lambda\upsilon_1 : \kappa} \mathrm{word}(T_1)$，其中 T_1 是类型 $\mu\,\upsilon_2\colon \mathrm{S}.\&\{\text{叶节点} : \mathsf{Skip},\ \text{节点}:\ \upsilon_2;\ ?\upsilon_1;\ \upsilon_2\}$。由于 T_1 不处于弱头范式，我们必须对其进行规范化，得到 $T_2 = \&\{\text{叶节点} : \mathsf{Skip},\ \text{节点}:\ T_1;\ ?\upsilon_1;\ T_1\}$。因此，$\mathrm{word}(T_1)$ 会返回一个新符号，我们称之为 X_1。若要获取 X_1的转换，我们必须先计算 $\mathrm{word}(T_2)$，它是一个带有转换 $X_2 \,\&\xrightarrow{1}\, \mathrm{word}(\mathsf{Skip})$ 和 $X_2 \,\&\xrightarrow{2}\, \mathrm{word}(T_1;\ ?\upsilon_1;\ T_1)$ 的新符号 X_2。因此我们还得到 $X_1 \,\&\xrightarrow{1}\, \mathrm{word}(\mathsf{Skip})$ 和 $X_1 \,\&\xrightarrow{2}\, \mathrm{word}(T_1;\ ?\upsilon_1;\ T_1)$。

我们有 $\mathrm{word}(\mathsf{Skip}) = \varepsilon$，但仍需计算 $\mathrm{word}(T_1;\ ?\upsilon_1;\ T_1)$。该类型规范化为 $T_3 = T_2;\ ?\upsilon_1;\ T_1$，因为 $T_1 \Downarrow T_2$。因此，$\mathrm{word}(T_1;\ ?\upsilon_1;\ T_1)$ 是一个新符号 X_3。要得到 X_3 的产生式，我们必须计算 $\mathrm{word}(T_2;\ ?\upsilon_1;\ T_1) = \mathrm{word}(T_2)\,\mathrm{word}(?\upsilon_1)\,\mathrm{word}(T_1)$。此时，我们已经有 $\mathrm{word}(T_1) = X_1$，以及 $\mathrm{word}(T_2) = X_2$。

我们仍需计算 $?\upsilon_1$，它是一个新符号 X_4，其产生式为 $X_4 \xrightarrow{?\,1} \mathrm{word}(\upsilon_1)\bot$ 和 $X_4 \xrightarrow{?\,2} \varepsilon$。接着，$\mathrm{word}(\upsilon_1)$ 是一个新符号 X_5，其产生式为 $X_5 \xrightarrow{\upsilon_1} \varepsilon$。最后，我们得到 $\mathrm{word}(T_2;?\upsilon_1;\ T_1) = X_2X_4X_1$，这意味着我们可以写出 X_3 的产生式：$X_3 \,\&\xrightarrow{1}\, X_4X_1$ 和 $X_3 \,\&\xrightarrow{2}\, X_3X_4X_1$。

综上所述，我们终于可以得到这个简单文法：

$$X_0 \xrightarrow{\lambda\upsilon_1 : \mathrm{T}} X_1 \qquad X_1 \xrightarrow{\&1} \varepsilon \qquad X_1 \xrightarrow{\&2} X_3 \qquad X_2 \xrightarrow{\&1} \varepsilon \qquad X_2 \xrightarrow{\&2} X_3$$

$$X_3 \xrightarrow{\&1} X_4X_1 \qquad X_3 \xrightarrow{\&2} X_3X_4X_1 \qquad X_4 \xrightarrow{?1} X_5\bot \qquad X_4 \xrightarrow{?2} \varepsilon \qquad X_5 \xrightarrow{\upsilon_1} \varepsilon$$

接下来，我们将论证类型等价（即类型上的双模拟等价）对应于相应文法上的双模拟等价。以下引理给出了这一结论：类型的标记迁移系统与相应非终结符词的标记迁移系统恰好具有相同的迁移。

引理 6（完全抽象）。令 $T \in F^{\mu_*;\,\omega}$ 以及（$\mathcal{T}_T$，$\mathcal{N}_T$，词(T) $\mathcal{R}_T$）为对应的简单文法。另假设词$(T) \approx \gamma$。

1. 若 $T \xrightarrow{a} U$ 则存在 γ' 使得 $\gamma \xrightarrow{a} \gamma'$ 且词$(U) \approx \gamma'$。2. 若$\gamma \xrightarrow{a} \gamma'$ 则存在 U 使得 $T \xrightarrow{a} U$ 且词$(U) \approx \gamma'$。

证明。证明采用余归纳法，证明该关系

$$\mathcal{R} = \{(T, \gamma) \colon \mathrm{word}(T) \approx \gamma\}$$

对迁移关系向后封闭。这通过对T进行分类讨论来完成。首先，假设 T 弱头范式。下面展示几个情形。

– 若 T是 $\alpha\, T_1 \ldots T_m$，则根据规则 L-Var1 和 L-Var2，T处的标记迁移系统有针对 $1 \le j \le m$的迁移 $T \xrightarrow{\alpha_0}$ Skip和 $T \xrightarrow{\alpha_j} T_j$。类似地，对于具有产生式 $Y \xrightarrow{\alpha_0} \varepsilon$ 的 Y，词$(T) = Y$，此外，对于 $1 \le j \le m$，词$(T_j)\bot$。由于$Y \approx \gamma$，γ 处的标记迁移系统有迁移 $\gamma \xrightarrow{\alpha_0} \gamma_0$ 和 $\gamma \xrightarrow{\alpha_j} \gamma_j$，其中存在某个 γ_0 使得$\varepsilon \approx \gamma_0$，且存在某个 γ_j 使得词$(T_j)\bot \approx \gamma_j$，对于 $1 \le j \le m$。由于词(Skip) $= \varepsilon$，可得(Skip γ_0) $\in \mathcal{R}$。最后，由于词(T_j) $\approx$ 词$(T_j)\bot$，可得$(T_j, \gamma_j) \in \mathcal{R}$，对于 $1 \le j \le m$。

– 如果 T 是 Skip，那么 T 处的标记迁移系统没有转换。类似地，word$(T) = \varepsilon$也没有转换。由于 $\varepsilon \approx \gamma$，$\gamma$ 处的标记迁移系统也没有转换。– 如果 T是 U; V，那么根据规则 W-Seq2 Uwhnf，且 U既不是 Skip，也不是顺序组合。我们必须对 U进行第二次分类讨论。例如，若 U是 $\alpha\, U_1 \ldots U_m$，那么根据规则 L-VarSeq1 和 L-VarSeq2，T 处的标记迁移系统有转换 $T \xrightarrow{\alpha_0} V$ 和 $T \xrightarrow{\alpha_j} T_j$，对应于 $1 \le j \le m$。类似地，对于带有产生式 $Y \xrightarrow{\alpha_0} \varepsilon$的 Y，word(T) $=$word(U)word(V) $= Y$word(V)；此外，对于 $1 \le j \le m$，还有 $Y \xrightarrow{\alpha_j}$ word$(U_j)\bot$。由于 Yword(V) $\approx \gamma$，γ 处的标记迁移系统有转换 $\gamma \xrightarrow{\alpha_0} \gamma_0$ 和 $\gamma \xrightarrow{\alpha_j} \gamma_j$，分别对应某个满足 word$(V)$ $\approx \gamma_0$的 γ_0，以及某个满足 word$(U_j)\bot$word(V) $\approx \gamma_j$ 的 γ_j，其中 $1 \le j \le m$。我们立即得到 $(V, \gamma_0) \in \mathcal{R}$。最后，由于 word$(U_j)$ $\approx$word$(U_j)\bot$ word(V)，对于 $1 \le j \le m$，我们得到 $(U_j, \gamma_j) \in \mathcal{R}$。

接下来，假设 T 不处于弱头范式。有两种可能。

– 如果$T \Downarrow$Skip，那么根据规则 L-Red，T 处的标记迁移系统没有转换。类似地，word(T) $= \varepsilon$ 没有转换。由于 $\varepsilon \approx \gamma$，$\gamma$ 处的标记迁移系统也没有转换。– 如果 $T \Downarrow U \neq$ Skip，那么根据规则 L-Red，T处的标记迁移系统有迁移 $T \xrightarrow{a} U'$，当且仅当 U 处的标记迁移系统有相应的迁移 $U \xrightarrow{a} U'$（具有相同的U'）。类似地，word(T) $= Y$对于带有产生式 $Y \xrightarrow{a} \gamma'\delta$的 Y，其中 $Z\delta$ =word(U)，而 Z有产生式 $Z \xrightarrow{a} \gamma'$。因此，word$(T)$处的标记迁移系统

有迁移 word(T) $\xrightarrow{a}$ δ' 当且仅当 word(U) 处的标记迁移系统有迁移 word(U) $\xrightarrow{a}$ δ'（具有相同的 δ'）。因此，word(U) $\approx$word(T) $\approx \gamma$，从而 (U，γ) $\in \mathcal{R}$。最后，由于 U 弱头范式 前述分类讨论表明，U和 γ 具有匹配的迁移。由此得出，T 和 γ 也具有匹配的迁移。

根据上述结果，我们得到双模拟等价词(T) $\approx$词(U) 关于双模拟等价 $T \sim U$的可靠性与完备性。事实上，根据引理 6，任何从 T开始的迁移序列，都可以由从词(T) 开始的迁移序列匹配；对于 U，情况也类似。因此$T \sim U$当且仅当词(T) $\approx$词(U)。

定理 3. 对于 $F^{\mu_{*};\omega}$ 中的类型，类型等价问题是可判定的。

证明。 给定类型 T, U在 $F^{\mu_{*};\omega}$中，判定 $T \sim U$的算法如下。首先计算词(T) 和词(U)；根据引理 5，此过程会终止。然后，使用任何已知的简单文法双模拟等价算法（例如 Almeida 等人 [6] 或 Burkart 等人 [15], 提出的算法）判定词(T) $\approx$词(U)，其时间复杂度为双指数级。正确性可由本定理前紧接着的讨论得出。

在本节余下部分中，我们将考察图2 中的其他类型类别，并研究它们对应的计算模型。由于类别 $F^{\mu;}$包含于 $F^{\mu_{*};\omega}$，因此我们也可以用简单文法表示不含 λ-抽象的类型。这样，我们便重新得到文献 [6,21] 中的已有结果。

现在来看类别 $F^{\mu_{*}\cdot}_{\omega}$。在这个类别中，我们没有 Skip，也没有顺序组合，并且消息运算符是二元的（$\sharp T.U$），而非一元的。由于没有顺序组合，因此无需考虑由非终结符组成的词，只需将类型转换为单个符号，即自动机中的状态。此外，由于递归不会超出μ_{κ}，从给定的 T出发，只能到达有限多个类型。因此，我们可以针对 $F^{\mu_{*}\cdot}_{\omega}$按如下方式调整构造。

在标记迁移系统的定义中（图 9）：

– 删除所有涉及顺序组合的规则；– 删除规则 L-Var1 for $m > 0$ and L-DualVar2 （这些规则只是为了区分顺序组合中的类型）；– 删除规则 L-Const 中的 $\iota =$ End 情形（这样 End 就不再有迁移）；– 将规则 L-Var1 with $m = 0$and L-Const右侧的 Skip 替换为 End；– 删除规则 L-Msg1 和 L-Msg2，并在规则 L-ConstApp中像处理其他常量一样处理 $\iota = \sharp$。

我们还将 word(T) 的构造替换为 state(T) 的构造，将每个类型 T与有限状态自动机中的一个状态相关联。对于每个迁移$T \xrightarrow{a} U$，我们都有对应的迁移 state(T) $\xrightarrow{a}$state(U)。注意，由于原始 LTS 同样是确定性的，所得自动机也是确定性的（对于每个类型 T和标签 a，至多存在一个迁移 $T \xrightarrow{a} U$）。

由于确定性有限状态自动机的双模拟等价可以在多项式时间内判定 [44], ，我们得到以下结果。

定理 4。

1. 对每种类型 T 在 $F_\omega^{\mu_*\cdot}$中，我们都可以关联一个有限状态自动机，它对应于由 T生成的标记迁移系统（LTS）的（片段）。2. 对于 $F_\omega^{\mu_*\cdot}$中的类型，类型等价问题可在多项式时间内判定。

显然，定理 4 适用于 $F_\omega^{\mu_*\cdot}$的以下子类： F^μ、 $F^{\mu\cdot}$ 以及 $F_\omega^{\mu_*}$。这样，我们重新得到文献 [16,21,33]中的已有结果。

最后，我们考察 F_ω^μ、 $F_\omega^{\mu\cdot}$和 F_ω^μ这些涉及任意种类递归的类。我们将证明，这些类已足以模拟确定性下推自动机；因此，类型等价问题变得不切实际（即，目前尚无已知的算法实用实现）。我们只关注最简单的情形 F_ω^μ，因为另外两个类的表达能力更强。我们不考察确定性下推自动机，而是考察确定性一阶文法，这是一种等价的计算模型 [46]。考察确定性一阶文法的优势在于可以简化我们的构造。我们称一个 一阶文法 为一个元组（$\mathcal{X}$、 $\mathcal{T}$、 $\mathcal{N}$、 E、 $\mathcal{R}$），其中：

– $\mathcal{X}$是变量集合 $\alpha, \beta, \ldots$； $\mathcal{T}$是终结符集合 $a, b, \ldots$； $\mathcal{N}$是非终结符集合 X, $Y, \ldots$ – 每个非终结符 X 都有元数 $m = \text{arity}(X) \in \mathbb{N}$. – 表达式集合 $\mathcal{E}$ 上的表达式由以下两条规则归纳定义：任意变量 α 都是表达式；如果元数(X) $= m$，且 $E_1, \ldots, E_m$ 都是表达式，那么 $X\ E_1 \ldots E_m$也是表达式。每当 $m = 0$, X 称为常量。– E 是 $\mathcal{N}$上的表达式，称为初始表达式。– $\mathcal{R}$ 是产生式集合。每个产生式都是一个三元组 (X, a, E)，记作$X\ \alpha_1 \ldots \alpha_m \xrightarrow{a} E$，其中 $m = \text{arity}(X)$，且 E中的变量必须取自 $\alpha_1, \ldots, \alpha_m$。

一阶文法是确定性的，如果对于每个 X 和 a，至多存在一个产生式（X， a， E） $\in \mathcal{R}$。

正如简单文法定义了一个作用于非终结符词集上的标记迁移系统，一阶文法定义了一个作用于 $\mathcal{E}_0$ 闭表达式集合上的标记迁移系统。对于每个产生式 $X\ \alpha_1 \ldots \alpha_m \xrightarrow{a} E$，我们都有标号转移 $X\ E_1 \ldots E_m \xrightarrow{a} E[E_1/\alpha_1, \ldots, E_m/\alpha_m]$。

令 $\approx$ 表示根据一阶文法定义的闭表达式上的双模拟等价。现在我们给出一种将确定性一阶文法翻译为 F_ω^μ 中类型的完全抽象（即保持双模拟等价）方法。每个文法变量 α 都对应一个类型变量 α （种类为 T）。表达式 $X\ E_1 \ldots E_m$表示为类型应用 $X\ E_1 \ldots E_m$。若 X的元数为m，且产生式 $X\ \alpha_1 \ldots \alpha_m \xrightarrow{a_j} E_j$的左侧为某个 j，则我们将指定 X 的方程写成记录形式（由于一阶文法是确定性的，所有记录标签都互不相同，因此该方程的右侧

指定 X 的方程是良构的）。

$$X \doteq \lambda\alpha_1 : \mathrm{T}. \ldots \lambda\alpha_m : \mathrm{T}.\{a_1 : E_1, \ldots, a_m : E_m\}$$

由此得到一个方程组 $\{X_i \doteq T_i\}$，每个非终结符对应一个方程 X_i，其中非终结符可以出现在方程的右侧 T_i。最后，给定一个初始表达式 E，如何使用上述系统将其转换为 μ-类型是标准做法。

利用上述转换，我们能够将一阶文法的一次迁移$E \xrightarrow{a_j} F$模拟为相应类型上的一次迁移 $E \xrightarrow{\{a_i\}_j} F$。因此，该转换是完全抽象的，我们得到如下结果。

定理 5. 设 E 和 F 是某一阶文法上的闭表达式，且E， F是对应的类型。那么 $E \approx F$当且仅当 $E \sim F$。

让我们通过一个例子来更好地理解上述转换。考虑语言 $L_3 = \{\ell^n a r^n a \mid n \geq 0\} \cup \{\ell^n b r^n b \mid n \geq 0\}$，其字母表为 $\{a, b, \ell, r\}$。 L_3是一个典型的语言示例：它无法用简单文法描述，但可以由确定性下推自动机接受 [51]。考虑具有以下非终结符的一阶文法： X， R， A， B， $\bot$，初始表达式为 $X\ A\ B$，产生式为

$$X\ \alpha\ \beta \xrightarrow{\ell} X\ (R\ \alpha)\ (R\ \beta) \qquad X\ \alpha\ \beta \xrightarrow{a} \alpha \qquad X\ \alpha\ \beta \xrightarrow{b} \beta$$
$$R\ \alpha \xrightarrow{r} \alpha \qquad A \xrightarrow{a} \bot \qquad B \xrightarrow{b} \bot$$

请注意， $\bot$ 是一个没有产生式的常量。很容易看出，这个一阶文法的轨迹与 L_3 中的词完全对应。按照上述翻译步骤，我们得到以下方程组

$$X \doteq \lambda\alpha : \mathrm{T}.\lambda\beta : \mathrm{T}.\{\ell : X(R\alpha)(R\beta), a : \alpha, b : \beta\} \quad R \doteq \lambda\alpha : \mathrm{T}.\{r : \alpha\}$$
$$A \doteq \{a : \bot\} \qquad B \doteq \{b : \bot\} \qquad \bot \doteq \{\}$$

因此，初始表达式 $X\ A\ B$ 变为类型

$$(\mu\,\xi : \mathrm{T} \Rightarrow \mathrm{T} \Rightarrow \mathrm{T}.\ \lambda\alpha : \mathrm{T}.\lambda\beta : \mathrm{T}.\{\ell : \xi\{r : \alpha\}\{r : \beta\}, a : \alpha, b : \beta\})\{a : \{\}\}\{b : \{\}\},$$

其迁移模拟了一阶文法的迁移。

6 项语言及其元理论

本节简要介绍一种配备$F^{\mu_{*;\omega}}$ 类型的并发函数式语言及其元理论。相关结果大多源自文献，不过项层面的显式递归以及类型上下文中的无限制绑定，在会话类型中还较为新颖。

值、项、进程和按值调用求值上下文的语法由图 11中的文法定义。同一幅图还引入了以下类型

$$
\begin{array}{rcl}
v & ::= & c \mid x \mid \lambda x\colon T.t \mid \mathsf{rec}\, x\colon T.v \mid \Lambda\alpha\colon \kappa.v \mid \{\overline{l_i = v_i}\} \mid \langle l = v\rangle \,\mathsf{as}\, T \\
 & & \mathsf{receive}[T] \mid \mathsf{receive}[T][T] \mid \mathsf{send}[T] \mid \mathsf{send}[T]\, v \mid \mathsf{send}[T]\, v[T] \\
t & ::= & v \mid t\, t \mid t[T] \mid \{\overline{l_i = t_i}\} \mid \mathsf{let}\, \{\overline{l_i = x_i}\} = t\, \mathsf{in}\, t \\
 & & \langle l = t\rangle \,\mathsf{as}\, T \mid \mathsf{case}\, t\, \mathsf{of}\, t \mid \mathsf{match}\, t\, \mathsf{with}\, t \\
p & ::= & \langle t\rangle \mid p \mid p \mid (\nu xx)p \\
E & ::= & [] \mid E\, t \mid v\, E \mid E[T] \mid \{l_1 = v_1, \ldots, l_j = E, \ldots, l_n = e_n\} \\
 & & \mathsf{let}\, \{\overline{l_1 = x_i}\} = E\, \mathsf{in}\, t \mid \langle l = E\rangle \,\mathsf{as}\, T \mid \mathsf{case}\, E\, \mathsf{of}\, t \mid \mathsf{match}\, E\, \mathsf{with}\, t
\end{array}
$$

$c ::=$		项常量
接收	$\forall\alpha\colon \mathrm{T}.\forall\beta\colon \mathrm{S}.?\ \alpha.\beta \to \alpha \otimes \beta$	在通道上接收
send	$\forall\alpha\colon \mathrm{T}.\alpha \to \forall\beta\colon \mathrm{S}.!\ \alpha.\beta \to \beta$	在通道上发送
选择 l_j 作为 $\oplus\{\overline{l_i\colon T_i}\}$	$\oplus\{\overline{l_i\colon T_i}\} \to T_j$	内部选择
关闭	End →单位类型	通道关闭
fork	(单位类型 → 单位类型) → 单位类型	创建新线程
new	$\forall\alpha\colon \mathrm{S}.\alpha \otimes$ 对偶 α	通道创建

图 11：项，以及项常量的类型标注。

常量。项语言本质上是支持会话运算符的多态 λ 演算，其表述方式沿用 Almeida 等人和 Cai 等人的工作。[2,16]。它由 System F 扩展而来，包含项和类型抽象、记录和变体，以及各自的构造子和析构子。对会话操作和并发的支持包括通道创建（新建）、各种通道操作（接收、发送、匹配、选择和关闭）以及线程创建（分叉）。我们在项层面编程，仅在运行时使用进程。进程包括作为线程的项、并行组合和通道创建，这些都受到双重绑定子的 π 演算的启发 [73]。

项和进程的类型判定见图 12。形如 $\Delta \mid \Gamma \vdash t\colon T$的判定记录了这样一个事实：项 t 在上下文 Δ （记录类型变量的种类）和 Γ （记录项变量的类型）下具有类型 T 。进程的判定 $\Gamma \vdash p$表示， p 在上下文 Γ下是良类型的。进程的判定是项的判定的简化形式，因为进程没有自由类型变量，也不指定特定类型。这里，Unit 是空记录类型 $\{\}$的简写，而$T \otimes U$是记录类型 $\{\mathrm{Fst}\colon T, \mathrm{Snd}\colon U\}$的简写。再次说明，这些规则改编自上述两篇文献。与 Cai 等人 [16] 的不同之处在于，我们采用线性设定，因此公理（T-Const 和 T-Var）作用于空上下文，而大多数其他规则必须相应地拆分上下文。规则 T-TAbs 是对 Cai 等人 [16]所用规则的简化；我们可以轻松证明这两条规则可以互换。我们支持用于递归函数的指数类型 [37]，从而可以编写包含多次递归调用的函数（这对

Term typing $\boxed{\Delta \mid \Gamma \vdash t : T}$

$$\textsc{T-Const}\ \frac{\Delta \vdash T_c : *}{\Delta \mid \cdot \vdash c : T_c} \qquad \textsc{T-Var}\ \Delta \mid x : T \vdash x : T \qquad \textsc{T-TApp}\ \frac{\Delta \mid \Gamma_1 \vdash t_1 : U \to T \qquad \Delta \mid \Gamma_2 \vdash t_2 : U}{\Delta \mid \Gamma_1, \Gamma_2 \vdash t_1\, t_2 : T}$$

$$\textsc{T-Rec}\ \frac{\Delta \vdash T : * \qquad \Delta \mid \Gamma, x :^{\omega} T \to U \vdash v : T \to U}{\Delta \mid \Gamma \vdash \mathsf{rec}\, x : T \to U.v : T \to U} \qquad \textsc{T-TAbs}\ \frac{\Delta, \alpha : \kappa \mid \Gamma \vdash v : T\,\alpha}{\Delta \mid \Gamma \vdash (\Lambda\alpha : \kappa.v) : \forall_{\kappa} T}$$

$$\textsc{T-Record}\ \frac{\overline{\Delta \mid \Gamma_i \vdash t_i : T_i}}{\Delta \mid \overline{\Gamma} \vdash \{\overline{l_i = t_i}\} : \{\overline{l_i : T_i}\}} \qquad \textsc{T-Proj}\ \frac{\Delta \mid \Gamma_1 \vdash t_1 : \{\overline{l_i : T_i}\} \qquad \Delta \mid \Gamma_2, \overline{x_i : T_i} \vdash t_2 : T}{\Delta \mid \Gamma_1, \Gamma_2 \vdash \mathsf{let}\, \{\overline{l_i = x_i}\} = t_1\, \mathsf{in}\, t_2 : T}$$

$$\textsc{T-Match}\ \frac{\Delta \mid \Gamma_1 \vdash t_1 : \&\{\overline{l_i : T_i}\} \qquad \Delta \mid \Gamma_2 \vdash t_2 : \{\overline{l_i : T_i \to T}\}}{\Delta \mid \Gamma_1, \Gamma_2 \vdash \mathsf{match}\, t_1\, \mathsf{with}\, t_2 : T} \qquad \textsc{T-Eq}\ \frac{\Delta \mid \Gamma \vdash t : U \qquad \Delta \vdash U : * \qquad U \sim T}{\Delta \mid \Gamma \vdash t : T}$$

$$\textsc{T-Dereliction}\ \frac{\Delta \mid \Gamma, x : T \vdash t : U}{\Delta \mid \Gamma, x :^{\omega} T \vdash t : U} \qquad \textsc{T-Weakening}\ \frac{\Delta \mid \Gamma \vdash t : U}{\Delta \mid \Gamma, x :^{\omega} T \vdash t : U} \qquad \textsc{T-Contraction}\ \frac{\Delta \mid \Gamma, y :^{\omega} T, z :^{\omega} T \vdash t : U}{\Delta \mid \Gamma, x :^{\omega} T \vdash t[x/y][x/z] : U}$$

进程类型判定 $\boxed{\Gamma \vdash p}$

$$\frac{\varepsilon \mid \Gamma \vdash t : \mathsf{Unit}}{\Gamma \vdash \langle t \rangle} \qquad \frac{\Gamma_1 \vdash p_1 \qquad \Gamma_2 \vdash p_2}{\Gamma_1, \Gamma_2 \vdash p_1 \mid p_2} \qquad \frac{\Gamma, x : T, y : \mathsf{Dual}\, T \vdash p}{\Gamma \vdash (\nu xy)p}$$

(T-App、T-Abs：参见 Cai 等人 [16]；T-Case 改编自 T-Match)

图 12：类型判定。

例如，消耗二叉树）以及不使用递归函数的分支（适用于预期会终止的代码）。为此，我们在项变量上下文中添加一个无限制绑定 $x : {}^{\omega} T$，为 rec 添加一条显式规则（而不是像 Cai 等人那样将 rec 设为常量 [16]），并为无限制绑定添加结构规则（T-Dereliction、T-Weakening 和 T-Contraction）。得益于 System F的强大表达能力，大多数会话和并发运算符都可以表示为常量。例如，接收 接收一种会话类型 $!\alpha.\beta$，其中 α是消息的负载，类型任意； β是延续，其类型为会话类型；该操作返回一个由接收到的值和延续通道组成的对。与往常一样， $\forall\alpha :\ \kappa.T$是类型 $\forall_{\kappa}\ (\lambda\alpha :\ \kappa.T)$的缩写。例外是外部选择（T-Match），它无法由类型表示（与 T-Case 类似），因此需要专门的类型判定规则。

项约简和进程约简见图 13。项约简由 System F 中关于记录、变体类型和递归的标准公理构成。求值上下文大大简化了结构归约规则，并为进程约简铺平了道路。遵循 Milner [55]，我们通过以下方式将进程提取出来：

项约简 $\boxed{t \to t}$

$(\lambda x\colon T.t)\ v \to t[v/x]\ (\Lambda\alpha\colon \kappa.t)[T] \to t[T/\alpha]$ 令 $\{\overline{l_i = x_i}\} = \{\overline{l_i = v_i}\}$ 于 $t \to t\overline{[v_i/x_i]}$ 分情况 $(\langle l_j = v\rangle$ 作为 $T)$ 匹配 $\{\overline{l_i = t_i}\} \to t_j\ v$ (递归 $x\colon T.v)\ u \to (v$[递归 $x\colon T.v/x]) \ u$

$$\frac{t_1 \to t_2}{E[t_1] \to E[t_2]}$$

结构同余（略去同余规则 ed) $\boxed{p \equiv p}$

$$p_1 \mid p_2 \equiv p_2 \mid p_1 \qquad (p_1 \mid p_2) \mid p_3 \equiv p_1 \mid (p_2 \mid p_3)$$
$$(\nu xy)p_1 \mid p_2 \equiv (\nu xy)(p_1 \mid p_2) \qquad (\nu wx)(\nu yz)p \equiv (\nu yz)(\nu wx)p$$

进程约简 $\boxed{p \to p}$

$$\frac{t_1 \to t_2}{\langle t_1\rangle \to \langle t_2\rangle} \qquad \langle E[\text{分叉}\ v]\rangle \to \langle E[\{\}]\rangle \mid \langle v\ \{\}\rangle \qquad \langle E[\text{new}[T]]\rangle \to (\nu xy)\langle E[\{x, y\}]\rangle$$

$$(\nu xy)(\langle E_1[\text{接收}[T][U]\ y\ ]\rangle \mid \langle E_2[\text{发送}[V][W]\ v\ x]\rangle) \to (\nu xy)(\langle E_1[\{y, v\}]\rangle \mid \langle E_2[x]\rangle)$$

$$(\nu xy)(\langle E_1[\text{匹配}\ y\ \text{与}\ \{\overline{l_i = t_i}\}]\rangle \mid \langle E_2[(\text{选择}\ l_j\ \text{作为}\ T)\ x]\rangle) \to (\nu xy)\langle E_1[t_j\ y]\rangle \mid \langle E_2[x]\rangle$$

$$(\nu xy)(\langle E_1[\text{关闭}\ y]\rangle \mid \langle E_2[\text{关闭}\ x]\rangle) \to \langle E_1[\{\}]\rangle \mid \langle E_2[\{\}]\rangle \qquad \frac{p_1 \to p_2}{p_1 \mid q \to p_2 \mid q}$$

$$\frac{p_1 \to p_2}{(\nu xy)p_1 \to (\nu xy)p_2} \qquad \frac{p_1 \equiv p_2 \qquad p_2 \to p_3 \qquad p_3 \equiv p_4}{p_1 \to p_4}$$

图 13：项与进程归约。

结构同余关系考虑了并行组合的结合性和交换性、作用域外扩以及通道绑定顺序的交换。这些规则与 Almeida 等人以及 Gay 和 Vasconcelos 的工作密切相关[2,35]。其中包括新线程分叉、新通道创建以及通信的公理（接收/发送、匹配/选择 和 关闭/关闭），以及允许在并行组合、通道创建和结构同余下进行归约的结构规则。

现在我们来讨论该语言的元理论，首先介绍项和进程的保持性。

定理 6（保持性）。

1. 若 $\Delta \mid \Gamma \vdash t\colon T$ 且 $t \to t'$，则 $\Delta \mid \Gamma \vdash t'\colon T$。2. 若 $\Gamma \vdash p$ 且 $p \equiv p'$，则 $\Gamma \vdash p'$。3. 若 $\Gamma \vdash p$ 且 $p \to p'$，则 $\Gamma \vdash p'$。

项语言的进展性仅在类型上下文只包含信道端点时才能得到保证。当 Δ 可从上下文中理解时，我们记作Γ^{S}，表示 Γ 只包含种类为 S 的类型，也就是说 $\Delta \vdash T : \mathrm{S}$ 对所有类型T，在 Γ 中均成立。良类型项要么是值，要么可以归约，或者在进程层面已准备好归约。对于会话操作——接收、发送、匹配、选择、关闭——归约尚待匹配的对端操作。

定理 7（项语言的进展性）。如果 $\Delta \mid \Gamma^{\mathrm{S}} \vdash t : T$，那么 t 是一个值，t 可以规约，或者 t 处于以下某种停滞形式：$E[\,\text{分叉}\ v]$、$E[\,\text{新建}\ [T]]$、$E[\,\text{接收}\ [T][U]\,v\,]$、$E[\,\text{发送}\ [U]\,T[v]\,x]$、$E[\,\text{匹配}\ y\ \text{与}\ \{\overline{l_i = t_i}\}]$、$E[\,(\text{选择}\ l_j\ \text{作为}\ T)\ x]$，或 $E[\,\text{关闭}\ x]$。

为了陈述我们关于不存在运行时错误的结果，我们需要了解项和进程结构的一些概念；这里我们遵循 Almeida 等人[2]。该主题表达式 e，记作 subj(e)，在 x 以下几种情况下定义。

$$\text{接收}[T][U]\,x \quad \text{发送}[T]\,v[U]\,x \quad \text{匹配}\ x\ \text{与}\ t\ (\text{选择}\ l_j\ \text{作为}\ T)\ x \quad \text{关闭}\ x$$

两个项 e_1 和 e_2 在通道 xy 上一致，记作 $\mathsf{agree}^{xy}(e_1, e_2)$，具体情况如下（省略对称形式）。

$$\mathsf{agree}^{xy}(\mathsf{receive}[T][U]\,x, \mathsf{send}[V]\,v[W]\,y) \qquad \mathsf{agree}^{xy}(\mathsf{close}\,x, \mathsf{close}\,y)$$
$$\mathsf{agree}^{xy}(\mathsf{match}\,x\,\mathsf{with}\,\{\overline{l_i = t_i}\}_{i\in I}, (\mathsf{select}\,l_j\,\mathsf{as}\,T)\,y) \quad j \in I$$

若一个封闭进程与某个包含以下形式的子表达式或子进程的进程结构等价，则该封闭进程是运行时错误。

1. $v\,u$其中 v不是 λ或 rec，$v \neq$接收$[T][U]$、发送$[T]\,u$、发送$[T]\,w[U]$、选择 l_j为 T、关闭、分叉；2. $v[T]$其中 v不是 Λ、$v \neq$接收、接收$[U]$、发送、发送$[U]$、new；3. let $\{\overline{l_i = x_i}\} = v$ in t和 v不属于形式 $\{\overline{l_i = u_i}\}$；4. case v of t和 $v \neq \langle l_j = u\rangle$为$T$或 $t \neq \{\overline{l_i = u_i}\}_{i\in I}$带有 $j \notin I$；5. 接收$[T][U]\,v$或发送$[T]\,u[U]\,v$或匹配 v带有 t或（选择 l为 T）v或关闭 v，且 v 不是端点 x；6. $\langle E_1[e_1]\rangle \mid \langle E_2[e_2]\rangle$ 且 subj(e_1) = subj(e_2)；7. $(\nu xy)(\langle E_1[e_1]\rangle \mid \langle E_2[e_2]\rangle)$ 且 subj(e_1) = x 且 subj(e_2) = y 且$\neg\mathsf{agree}^{xy}(e_1, e_2)$。

前五种情况是带记录 F 和变体类型的系统中的标准情况。前两种情况（项应用和类型应用）对会话类型和并发的支持，源自此类运算符的值类型（图11）。第5 项针对应用于非端点的会话运算符。第6 项针对同一通道端上的两个并发会话运算符。最后，第7 项针对同一通道的两个端点上两个会话操作不匹配的情况。

定理 8（安全性）。如果 $\Gamma^{\mathrm{S}} \vdash p$，那么 p 不是运行时错误。

可以从图 12 中项的声明式系统中轻松提取出一种算法化类型系统，方法是采用双向类型系统。该系统按照 Almeida 等人 [2], 提出的思路构造，将非语法导向的判定$\Delta \mid \Gamma \vdash t\colon\ T$转换为两个函数式判定：类型合成 $\Delta \mid \Gamma_1 \vdash t \Rightarrow T \mid \Gamma_2$ 和类型检查 $\Delta \mid \Gamma_1 \vdash t \Leftarrow T \mid \Gamma_2$。在算法化类型系统中，上下文 Γ_2包含上下文 Γ_1 [73,74,75]中未使用的部分。

7 相关工作

我们简要讨论与本文工作较为接近的研究。
系统 F 中的等递归。早期关于等递归类型的研究通常以余归纳的方式定义类型等价这一概念 [7,13,20,29,38]。如果两个类型展开后得到相同的无限树，则它们等价。只要这种展开是唯一的类型级计算，生成的树就是正则的，从而可以高效地进行判定。部分作者还研究了等递归与其他类型级计算概念的结合。Solomon 研究了参数化类型定义，它们对应于高阶种类 [63]。由于类型可以调用其他类型，这些定义会隐式地对应于 λ-项，并发生归约。一些作者在系统 F_ω 中研究等递归，并采用较弱或较强的相等性概念 [1,14,16,41]。关于系统 F 中的等递归，Cai 等人 [16] 的模型与我们的工作最为接近，事实上，我们截至 $F^{\mu\star\cdot}_{\omega}$ 的结果可以视为对其工作的推广。不过，Cai 等人偏离了通常的设定，允许非收缩类型（大多数作者，包括本文，都禁止这类类型），因此需要某种无穷 λ 演算。此外，本文还通过纳入具有独特语义的会话类型（例如顺序组合和对偶性），进一步扩展了等价性质。

会话类型系统。会话类型由 Honda 等人于 90 年代引入 [42,43,67]。等递归是构造无限会话类型的首种方法，它通常允许根据双模拟的余归纳概念来解释类型相等性 [52]。沿着这一思路，Keizer 等人 [48] 利用余代数来表示会话类型。自会话类型问世以来，人们一直致力于将相关理论扩展到非正则协议 [58,59,66]。上下文无关会话类型应运而生，成为一种自然的扩展，因为它仍然支持实用的类型相等算法 [4,5,6,21,56,68]。超越正则会话类型的其他方法包括嵌套会话类型 [25]，以及单计数器、下推和双计数器会话类型 [33]。然而，表达能力更强的概念并不适用于实用的类型等价算法，我们系统中的高阶类型也是如此 F^{μ}_{ω}。无论是否带有递归，会话类型中的多态性也一直是一个研究课题 [17,23,24,31,39]。

Dual 类型算子。 据我们所知，本文首次将对偶性内化为类型构造子。其他方案，例如 Alms 语言 [72], 将会话类型的对偶性视为用户可定义的类型函数，而非内置功能。我们的 Dual 是类型算子，而非类型函数。两者的区别在于，类型函数涉及类型级计算，该计算会收敛为一个

不带对偶的类型。例如，在 Alms 中，我们会写作 dual(!Int.End) =?Int.End（作为类型级计算），两边是 相同的类型。在我们的方案中，Dual (!Int;End) 本身就是一种类型，它恰好等价于 ?Int; End。与此同时，我们的方案允许使用诸如 Dual α，或(Dual α); $T_1; T_2$这样的类型，它们不会化简。

类型等价算法。 判定类型等价性的算法必然与相应类型系统的计算能力相关。这一点已被隐式或显式地用于获得可判定性结果。如前所述，如果等递归是唯一的类型级计算，那么类型可以表示为有限状态自动机（或等价地，表示为无限正则树）。尽管最初有人提出了一些指数时间算法 [32], 但后来已证明，该问题可在二次时间内解决 [53],这也在预料之中，因为它与有限状态自动机的双模拟问题相对应 [44]；另见 Pierce [57]。

下一个“最简单”的计算模型是简单文法，直观上它对应于只有一个状态的确定性下推自动机 [33]
Almeida 等人 [6] 提出了一种实用算法，用于检验简单文法的双模拟等价性。去掉确定性假设后，我们便得到格雷巴赫范式文法，它们等价于基本进程代数 [8,9]。在这一领域，双模拟算法已得到广泛研究 [15,19,47,49]；目前已知该问题的复杂度介于指数时间和双指数时间之间，但这并未排除针对更简单的简单文法模型存在多项式时间算法的可能性。

本文提出了一种从一阶文法到 F^{μ}_{ω}-类型的归约，表明表达能力更强的类型系统（F^{μ}_{ω}，本文及 Cai 等人 [16], 中提出的类型系统及其扩展）至少与确定性下推自动机一样强。据我们所知，与我们的结果最接近的是 Solomon [63],的工作，该工作展示了“上下文无关类型”这一类型全集与确定性上下文无关语言之间的转换。Solomon 研究的类型全集不同于 F^{μ}_{ω}。经过一些工作，我们可以证明 Solomon 的类型能够嵌入 F^{μ}_{ω}，从而将我们的结果作为推论导出。不过，直接证明我们所做的归约更容易、更简单。

确定性下推自动机的等价性问题长期以来一直是一个著名的开放问题，直到 S´enizergues 证明它是可判定的 [61,62]。自他的证明问世以来，许多作者都试图进一步完善这一结果，以期得到一个可实现的算法 [46,64,65]。

并发项语言。类型系统的实用性与其在编程语言中的可用性直接相关。本文讨论的这类类型系统很容易应用于函数式项语言 [45]。对于会话类型，现有的项语言要么受到 π演算 [27,73,69] 的启发，要么受到 λ演算 [35,54,70]，的启发，也有一些同时受到二者的启发 [71]。本文介绍的系统是线性的，这意味着资源必须恰好使用一次 [50,74]。一些作者通过考虑无限制类型限定符 [48,73] 或显式共享 [10]，进一步超越了线性性。

8 结论与未来工作

本文介绍了系统 F 的扩展，其中包含等递归、λ 抽象和上下文无关会话类型。我们提出了类型等价算法、一种项语言及其元理论。尽管我们定义了一个相当通用的系统，但事实证明，出于实际考虑，必须将递归限制为 μ_*，也就是类型级单态递归。无论如何，主系统 $F^{\mu_*;\,\omega}$是对（由 Cai 等人 $F^{\mu_*}_{\omega}$研究的）（收缩片段的）[16]以及 $F^{\mu;}$（由 Almeida 等人 [21]研究的）的非平凡扩展。

我们仅考虑了具有函数式性质的多态类型：类型$\forall\alpha\colon\ \kappa.\,T$必须始终属于种类 T。值得研究会话类型上的多态，因为这将支持更多额外行为。例如，我们可能会关注异质性值的流式传输，如类型 $\mu\,\alpha\colon\ \mathrm{S}.\&\{\mathrm{Done}\colon\ \mathsf{Skip},\ \text{更多}\colon\ \forall\beta\colon\ \mathrm{T}.?\beta;\ \alpha\}$。不过，这种扩展是否仍能转换为简单文法，目前尚不明确。

我们证明，系统 F^{μ}_{ω}, $F^{\mu\cdot}_{\omega}$, $F^{\mu;\omega}$的类型等价问题至少与一个非高效可判定问题一样困难。我们猜想，这些系统的能力与确定性下推自动机相同（因此，类型等价是可判定的），但我们尚无证明这一结果的构造方法。无论如何，我们证明了类型等价问题至少与确定性下推自动机的双模拟等价一样困难，这足以说明应聚焦于具有受限递归的重要片段。

我们研究完全递归（用于理论结果），或将递归限制为种类 $*$（用于算法结果）。研究介于两者之间的递归种类将会很有意思；下一个自然的例子是 $\mu_{*\Rightarrow *}$。如果考虑使用这种递归算子编写的类型，我们会得到什么计算模型？我们猜想，类型 F^{μ}_{ω}和 $F^{\mu\cdot}_{\omega}$在将递归限制为种类 $*\Rightarrow *$时，仍可表示为简单文法；而在表达能力更强的 $F^{\mu;}_{\omega}$中施加这种限制，则会超出这一模型的范围，但或许还达不到确定性下推自动机的表达能力。

参考文献

1. Abel, A.：基于类型的终止性：一种具有带大小的高阶类型的多态 λ 演算。博士论文，慕尼黑路德维希-马克西米利安大学（2007），https://d-nb.info/984765581 2. Almeida, B., Mordido, A., Thiemann, P., Vasconcelos, V.T.：多态上下文无关会话类型。CoRR **abs/2106.06658**（2021）。https://doi.org/1006658 3485502106./arXiv... Almeida, B., Mordido, A., Thiemann, P., Vasconcelos, V.T.：具有上下文无关会话类型的多态 λ 演算。Information and Computation（2022）。https://doi.org/10104948 410162022./j.ic... Almeida, B., Mordido, A., Vasconcelos, V.T.：FreeST：一种具有上下文无关会话类型的编程语言。http://rss.di.fc.ul.pt/tools/freest/（2019）5. Almeida, B., Mordido, A., Vasconcelos, V.T.：FreeST：函数式语言中的上下文无关会话类型。载于：PLACES。EPTCS，第291卷，第12–23页（2019）。https://doi.org/10.4204/EPTCS.291.2

6. Almeida, B., Mordido, A., Vasconcelos, V.T.：判定上下文无关会话类型的双模拟等价性。载于：TACAS。计算机科学讲义，第12079卷，第39–56页。施普林格（2020）。https://doi.org/10 7 3 7 1007 978 3 030 45237 ./ - - - - . Amadio, R.M., Cardelli, L.：递归类型子类型化。载于：POPL。第104–118页。ACM出版社（1991）。https://doi.org/10.1145/99583.99600 8. Baeten, J.C.M., Bergstra, J.A., Klop, J.W.：生成上下文无关语言的进程的双模拟等价性的可判定性。载于：PARLE。计算机科学讲义，第259卷，第94–111页。施普林格（1987）。https://doi.org/10.1007/3-540-17945-3 5 9. Baeten, J.C.M., Bergstra, J.A., Klop, J.W.：生成上下文无关语言的进程的双模拟等价性的可判定性。《ACM杂志》40(3)，653–682（1993 174141 10 10 1145 174130）。https://doi.org/ . / . . Balzer, S., Pfenning, F.：会话类型中的显式共享。《ACM程序设计语言汇刊》1(ICFP)，37:1–37:29（2017）。https://doi.org/10.1145/3110281 11. Barendregt, H.P.：λ演算——其语法与语义，《逻辑与数学基础研究》，第103卷。北荷兰出版社（1985）12. Barendregt, H.P.：无类型λ演算。载于：《逻辑与数学基础研究》，第90卷，第1091–1132页。爱思唯尔（1977）13. Brandt, M., Henglein, F.：递归类型相等与子类型的余归纳公理化。《Fundamenta Informaticae》33(4)，309–338（1998）。https://doi.org/10.3233/FI-1998-33401 14. Bruce, K.B., Cardelli, L., Pierce, B.C.：比较对象编码。载于：TACS。计算机科学讲义，第1281卷，第415–438页。施普林格（1997）。https://doi.org/10 0014561 15 1007. /BFb . Burkart, O., Caucal, D., Steffen, B.：任意上下文无关进程的初等双模拟判定过程。载于：MFCS。计算机科学讲义，第969卷，第423–433页。施普林格（1995 1 148 16 10 1007 3 540 60246）。https://doi.org/ . / - - - . Cai, Y., Giarrusso, P.G., Ostermann, K.：用于数据类型泛型编程的带等递归类型的System F-omega。载于：POPL。第30–43页。ACM（2016）。https://doi.org/10 1145 2837614 2837660 . / . . Caires, L., Perez, J.A., Pfenning, F., Toninho, B.：基于会话的通信中的行为多态性与参数性。载于：ESOP。计算机科学讲义，第7792卷，第330 6 19 18 349页。施普林格（2013）。https://doi.org/ . / - - - - . Cardelli, L., Wegner, P.：论类型、数据抽象与多态性的理解。《ACM计算调查》17(4)，471–522（1985）。https://doi.org/10.1145/6041.6042 19. Christensen, S., H¨uttel, H., Stirling, C.：所有上下文无关进程的双模拟等价性均可判定。《信息与计算》121(2)，143–148（1995）。https://doi.org/10.1006/inco.1995.1129 20. Colazzo, D., Ghelli, G.：核心Fun中的递归类型子类型化。载于：LICS。第137–146页。IEEE计算机学会（1999）。https://doi.org/ . /LICS. . . Costa, D., Mordido, A., Po.cas, D., Vasconcelos, V.T.：System F中的高阶上下文无关会话类型。载于：PLACES。EPTCS，第356卷，第24–35页（2022）。https://doi.org/10 3 22 4204 356 . /EPTCS. . . Curry, H.H., Feys, R., Craig, W.（编）：《组合逻辑》，第一卷。北荷兰出版社（1958）23. Dardha, O.：重新审视递归会话类型。载于：BEAT。EPTCS，第162卷，第27 4 24 34页（2014）。https://doi.org/ . /EPTCS. . . Dardha, O., Giachino, E., Sangiorgi, D.：重新审视会话类型。《信息与计算》256，253–286（2017）。https://doi.org/ . /j.ic. . .

25. Das, A., DeYoung, H., Mordido, A., Pfenning, F.：嵌套会话类型。载于：ESOP。计算机科学讲义，第 12648 卷，第 178–206 页。施普林格（2021）。https://doi.org/10 3 7 26 1007 978 3 030 72019 . / - - - - . Das, A., DeYoung, H., Mordido, A., Pfenning, F.：嵌套会话类型。ACM《程序语言与系统汇刊》44(3)，19:1–19:45（2022）。https://doi.org/10 3539656 27 1145. / . Das, A., Pfenning, F.：Rast：一种资源感知会话类型语言。《逻辑方法与计算机科学》18 2022 28 1 2022 10 46298 18 1 9 ( ) ( )。https://doi.org/ . /lmcs- (:) . De Bruijn, N.G.：采用无名占位符的 λ演算记法：一种公式自动处理工具，并应用于 Church-Rosser 定理。载于：《数学探索》。第 75 卷，第 381–392 页。爱思唯尔（1972）。https://doi.org/10 0 29 1016 1385 7258 72 90034 . / - ( ) - . Gapeyev, V., Levin, M.Y., Pierce, B.C.：揭示递归子类型：函数式编程瑰宝。载于：ICFP。第 221 351261 30 231 2000 10 1145 351240 – 页。ACM（）。https://doi.org/ . / . . Gauthier, N., Pottier, F.：编号的重要性：二阶递归类型的一阶规范形式。载于：ICFP。第 150–161 页。ACM（2004）。https://doi.org/10 1016872 31 1145 1016850 . / . . Gay, S.J.：会话类型中的有界多态。MSCS 18(5)，895–930（2008）。https://doi.org/10 0960129508006944 32 1017. /S . Gay, S.J., Hole, M.：π演算中会话类型的子类型。《信息学学报》42(2-3)，191–225（2005）。https://doi.org/10.1007/s00236-005-0177-z 33. Gay, S.J., Po cas, D., Vasconcelos, V.T.：无限会话类型的不同面貌。载于：FoSSaCS。计算机科学讲义，第 13242 卷，第 347–367 页。施普林格（2022）。https://doi.org/10 8 18 34 1007 978 3 030 99253 . / - - - - . Gay, S.J., Thiemann, P., Vasconcelos, V.T.：会话类型的对偶性：最终剪切。载于：PLACES。EPTCS，第 314 卷，第 23–33 页（2020）。https://doi.org/10 3 35 4204 314 . /EPTCS. . . Gay, S.J., Vasconcelos, V.T.：异步会话类型的线性类型理论。《函数式程序设计杂志》 **20** 1 19 50 2010 10 1017 0956796809990268 ( )，– 36 ( )。https://doi.org/ . /S . Girard, J.Y.：高阶算术的函数式解释与切消除。博士论文， 未知出版商（1972）37. Girard, J.：线性逻辑。《理论计算机科学》50，1–102（1987）。https://doi.org/10 4 38 1016 0304 3975 87 90045 . / - ( ) - . Glew, N.：二阶树理论。载于：ESOP。计算机科学讲义，第 2305 卷，第 147–161 页。施普林格（2002 8 11 39 10 1007 3 540 45927）。https://doi.org/ . / - - - . Griffith, D.E.：极化次结构会话类型。博士论文，伊利诺伊大学厄巴纳-香槟分校（2016 1562827 40 10 2172）。https://doi.org/ . / . Hindley, J.R., Seldin, J.P.：《组合子与λ演算导论》。剑桥大学出版社（1986）41. Hinze, R.：多类型值具有多种类类型。《科学计算编程》43(2- 3 4 42 129 159 2002 10 1016 0167 6423 02 00025)，– ( )。https://doi.org/ . /S - ( ) - . Honda, K.：二元交互的类型。载于：CONCUR。计算机科学讲义，第 715 卷，第 509– 523 页 2 35 43 1993 10 1007 3 540 57208 . 施普林格（）。https://doi.org/ . / - - - . Honda, K., Vasconcelos, V.T., Kubo, M.：基于结构化通信的编程的语言原语与类型纪律。载于：ESOP。计算机科学讲义，第 1381 卷，第 122 0053567 44 138 1998 10 1007 – 页。施普林格（）。https://doi.org/ . /BFb . Hopcroft, J.E., Karp, R.M.：用于测试有限自动机等价性的线性算法。技术报告，康奈尔大学（1971）45. Im, H., Nakata, K., Park, S.：具有递归类型、类型参数和抽象类型的收缩签名。载于：ICALP。计算机科学讲义，第 7966 卷，第 299–311 页。施普林格（2013）。https://doi.org/10.1007/978-3-642-39212-2 28

46. Janˇcar, P.：通过一阶文法双模拟简短证明 DPDA 语言等价性的可判定性。CoRR abs/1010.4760（2010），http://arxiv.org/abs/1010 4760 47 . . Janˇcar, P.：基本进程代数上的双模拟等价属于双指数时间（一个明确的证明）。《逻辑方法与计算机科学》9(1)（2012）。https://doi.org/ 10 2168 9 1 10 2013 . /LMCS- (:) . 48 Keizer, A.C., Basold, H., Perez, J.A.：会话余代数：会话类型与通信协议的余代数视角。载于：ESOP。计算机科学讲义，第 12648 卷，第 375– 403 页 3 14 49 2021 10 1007 978 3 030 72019 . 施普林格（）。https://doi.org/ . / - - - - . Kiefer, S.：BPA双模拟等价是指数时间困难的。《信息处理快报》113(4)，101–106（2013 004 50 10 1016 2012 12）。https://doi.org/ . /j.ipl. . . . Kobayashi, N., Pierce, B.C., Turner, D.N.：线性性与π演算。《ACM程序语言与系统汇刊》21(5)，914–947（1999）。https://doi.org/10 330251 51 1145 330249. / . . Korenjak, A.J., Hopcroft, J.E.：简单确定性语言。载于：SWAT。第 36 22 52 46 页。IEEE计算机学会（1966）。https://doi.org/ . /SWAT. . . Kozen, D., Silva, A.：实用余归纳。《数学结构与计算机科学》（7），1132 0960129515000493 53 1152 2017 10 1017 –（）。https://doi.org/ . /S . Lange, J., Yoshida, N.：会话类型的特征公式。载于：TACAS。计算机科学讲义，第 9636 卷，第 833–850 页。施普林格（2016）。https://doi.org/10 9 52 54 1007 978 3 662 49674 . / - - - - . Lindley, S., Morris, J.G.：谈论香蕉：会话类型的结构递归。载于：ICFP。第 434 2951921 55 447 2016 10 1145 2951913 2 2 119 – 页。ACM（）。https://doi.org/ . / . . Milnēr, R.：作为进程的函数。《数学结构与计算机科学》（），–141（1992 0960129500001407 56 10 1017）。https://doi.org/ . /S . Padovani, L.：上下文无关会话类型推断。《ACM程序语言与系统汇刊》41 3229062 57 2 9 1 9 37 2019 10 1145（），:–:（）。https://doi.org/ . / . Pierce, B.C.：《类型与程序设计语言》。麻省理工学院出版社（2002）58. Puntigam, F.：非正则进程类型。载于：Euro-Par。计算机科学讲义，第 1685 卷，第 1334– 1343 页 189 59 1999 10 1007 3 540 48311 . 施普林格（）。https://doi.org/ . / - - -X . Ravara, A., Vasconcelos, V.T.：并发对象演算的行为类型。载于：Euro-Par。计算机科学讲义，第 1300 卷，第 554–561 页。施普林格（1997）。https://doi.org/10 0002782 60 1007. /BFb . Reynolds, J.C.：迈向类型结构理论。载于：程序设计研讨会。计算机科学讲义，第 19 卷，第 408–423 页。施普林格（1974）。https://doi.org/ 10 1007 3 540 06859 7_148 . / - - - . Senizergues, G.：确定性下推自动机的等价性问题是可判定的。载于：ICALP。计算机科学讲义，第 1256 卷，第 671–681 页。施普林格（ 1997 ）。https://doi.org/ 10 1007 3 540 63165 8_221 . / - - - . Senizergues, G.：L(A) L(B)? 完备形式系统中的可判定性结果。载于：ICALP。计算机科学讲义，第 2380 卷，第 37 页。施普林格（2002）。https://doi.org/10 9 4 63 1007 3 540 45465 . / - - - . Solomon, M.H.：带参数的类型定义。载于：POPL。第 31–38 页。ACM出版社（1978 512765 64 10 1145 512760）。https://doi.org/ . / . . Stirling, C.：DPDA等价性的可判定性。《理论计算机科学》255(1-2)，1 3 65 31（2001）。https://doi.org/ . /S - ( ) - . Stirling, C.：判定DPDA等价性是原始递归的。载于：ICALP。《计算机科学讲义》第 2380 卷，第 821–832 页。施普林格（2002）。https://doi.org/ 10 1007 3 540 45465 9_70 . / - - - . Sudholt, M.：具有非正则协议的组件模型。载于：SC。计算机科学讲义，第 3628 卷，第 99–113 页。施普林格（2005）。https://doi.org/10.1007/11550679 8

67. Takeuchi, K., Honda, K., Kubo, M：一种基于交互的语言及其类型系统。载于：PARLE。计算机科学讲义，第 817 卷，第 398–413 页。施普林格（1994）。 https://doi.org/107 118 681007354058184./---. Thiemann, P., Vasconcelos, V.T.：上下文无关会话类型。载于：ICFP。第 462–475 页2951926 6920161011452951913. ACM（）。https://doi.org/./.. Toninho, B., Caires, L., Pfenning, F.：通过直觉主义线性类型理论实现依赖会话类型。载于：PPDP。第 161–172 页。ACM（2011）。 https://doi.org/102003499 7011452003476./.. Toninho, B., Caires, L., Pfenning, F.：高阶进程、函数与会话：一种单子式集成。载于：ESOP。计算机科学讲义，第 7792 卷，第 350–369 页。施普林格（20136 20 711010079783642337036）。 https://doi.org/./----. Toninho, B., Yoshida, N.：论多态会话与函数：两个（完全抽象）编码的故事。ACM《程序语言与系统汇刊》**43**(2), 7:1–7:55 (20213457884 72101145）。 https://doi.org/./. Tov, J.A.：次结构类型的实用编程。博士论文，东北大学（2012）。 73. Vasconcelos, V.T.：会话类型基础。《信息与计算》**217**，52–70（2012）。 https://doi.org/10002 741016201205./j.ic.... Walker, D.：《类型与程序设计语言高级专题》，第 3 章“次结构类型系统”，第 3–44 页。麻省理工学院出版社（2005）。75. Zalakain, U., Dardha, O：π 带有剩余项：一种基于 Agda 的形式化。在：FORTE。计算机科学讲义，第 12719 卷，第 157–174 页。施普林格（2021）。 https://doi.org/10.1007/978-3-030-78089-0 9

定理 2 的证明

本节旨在证明定理 2，即种类判定是可判定的。由于种类判定需要正规化（这是因为规则 K-TApp），我们必须先研究这一问题。

即使不考虑顺序组合和递归，正规化也可能不终止，这一点早已为人所知（$(\lambda x.x\ x)\ (\lambda x.x\ x)$ 就是典型例子）；事实上，判断一个类型是否能够正规化是不可判定的。标准做法是考虑经过种类判定的类型，它们是强正规化的。然而，在我们的模型中，种类判定本身也需要正规化，由此陷入了“先有鸡还是先有蛋”的困境。解决办法是考虑一种预种类判定的概念。我们将用$\Delta \vdash_{\mathsf{pre}} T : \kappa$ 表示 T 具有种类 κ的预种类判定。预种类判定的规则与种类判定相同，唯一的例外是规则 K-TApp，该规则取消了正规化条件，成为规则 PK-TApp:

$$\frac{\Delta \vdash T : \kappa \Rightarrow \kappa' \quad \Delta \vdash U : \kappa \quad T\,U\ \mathsf{norm}}{\Delta \vdash T\,U : \kappa'}\ \text{K-TApp} \Rightarrow \frac{\Delta \vdash_{\mathsf{pre}} T : \kappa \Rightarrow \kappa' \quad \Delta \vdash_{\mathsf{pre}} U : \kappa}{\Delta \vdash_{\mathsf{pre}} T\,U : \kappa'}\ \text{PK-TApp}$$

引理 7。 $\Delta \vdash_{\mathsf{pre}} T : \kappa$ 可判定（在线性时间内）。

证明。 预种类判定使用规则 PK-TApp 而非 PK-TApp，因此不需要正规化。因此，给定 Δ 和 T，通过遍历定义 T的抽象语法树，并使用上下文 Δ 来推断

变量的种类。此过程只需单次遍历，因此可在线性时间内终止。因此 $\Delta \vdash_{\mathsf{pre}} T : \kappa$ 也可在线性时间内判定。

现在我们可以使用预种类判定来考察正规化。回想一下，根据图 6，类型 T 基本上有四种归约方式：对 λ-项的应用 (R-β) 进行通常的 β-归约；递归算子的归约 (R-μ)；顺序组合的归约 (R-Seq1、 R-Seq2 和 R-Assoc)；以及对偶的归约。让我们把由 R-μ 引起的归约与其他规则区分开来，也就是说，用 $\longrightarrow_{\mu}$ 表示递归下的归约，用 $\longrightarrow_{\beta;D}$ 表示通常的 β-归约，以及顺序组合和对偶的归约。我们还将这些归约扩展到应用中的函数，即当 $T \longrightarrow_{\mu} U$、$T \longrightarrow_{\beta;D} U$ 时，分别有 $T\ V \longrightarrow_{\mu} U\ V$、$T\ V \longrightarrow_{\beta;D} U\ V$。这样，我们便有 $\longrightarrow = \longrightarrow_{\beta;D} \cup \longrightarrow_{\mu}$。我们也将弱头范式和正规化的概念扩展到归约 $\longrightarrow_{\beta;D}$ 和 $\longrightarrow_{\mu}$，并分别记作 $T \Downarrow_{\beta;D} U$ 和 $T \Downarrow_{\mu} U$。

引理 8（正规化）。若 $\Delta \vdash_{\mathsf{pre}} T : \kappa$，则存在唯一的 U 使得$T \Downarrow_{\beta;D} U$。

证明。这是对简单类型 λ 演算正规化结果 [57, 第 12] 章的直接扩展。纳入顺序组合和对偶性下的归约这一事实并不会使标准证明失效，因为这些归约通过尝试将类型构造子移至前端来简化类型；特别是，定义该类型的抽象语法树的深度不会随着此类归约而增加。

注意，我们无法将上述引理扩展到完整的 β-正规化，因为递归算子的归约可能会增大所得类型的规模。最简单的例子是 $\mu_{\mathrm{S}}(\lambda\alpha : \mathrm{S}.\alpha)$，其预种类为 $\vdash_{\mathsf{pre}} \mu_{\mathrm{S}}(\lambda\alpha : \mathrm{S}.\alpha) : \mathrm{S}$，但无法规范化：

$$\mu_{\mathrm{S}}\ (\lambda\alpha : \mathrm{S}.\alpha) \longrightarrow_{\mu} (\lambda\alpha : \mathrm{S}.\alpha)\ (\mu_{\mathrm{S}}\ (\lambda\alpha : \mathrm{S}.\alpha)) \longrightarrow_{\beta;D} \mu_{\mathrm{S}}\ (\lambda\alpha : \mathrm{S}.\alpha) \longrightarrow_{\mu} \cdots$$

定理 2（类型形成的可判定性）。 $\Delta \vdash T : \kappa$ 对 $F^{\mu_*;\omega}$ 中的类型是可判定的。

证明。 给定 Δ， T， κ 。我们可以先判断 $\Delta \vdash_{\mathsf{pre}} T : \kappa$（引理 7）。如果 T 不是预种类的，那么它也不是种类。否则，我们需要判断 T 是否可规范化，或者等价地，是否存在一个无限归约序列 $T = T_0 \longrightarrow T_1 \longrightarrow T_2 \longrightarrow \cdots$。根据引理 8，这样的序列必定包含无限多次 μ-归约，且任意两次 μ-归约之间必须有有限次其他归约。换句话说，我们可以构造（该序列的任意有限前缀）：

$$T = T_0 \Downarrow_{\beta;D} T_0' \longrightarrow_{\mu} T_1 \Downarrow_{\beta;D} T_1' \longrightarrow_{\mu} T_2 \Downarrow_{\beta;D} T_2' \longrightarrow_{\mu} \cdots$$

其中，对于每个 i， T_i' 都必定能在有限步之后从 T_i 到达。

在上述序列中，唯一可以应用于T_i' 的归约是 μ-归约，因为就其他归约而言， T_i' 处于弱头范式。如果 T_i'

没有任何 μ-归约，序列便会终止，我们可以正确判定 T 可以规范化。否则，如果 T_i' （仅）有一个 μ-归约，那么它必定属于以下情况之一（这里我们用到了递归的种类限制 $*$）：

$$
\begin{aligned}
T_i' &= \mu_* \, U \longrightarrow_\mu U \, (\mu_* \, U) \\
T_i' &= (\mu_* \, U); V \longrightarrow_\mu (U \, (\mu_* \, U)); V \\
T_i' &= \mathsf{Dual} \, (\mu_* \, U) \longrightarrow_\mu \mathsf{Dual} \, (U \, (\mu_* \, U)) \\
T_i' &= (\mathsf{Dual} \, (\mu_* \, U)); V \longrightarrow_\mu (\mathsf{Dual} \, (U \, (\mu_* \, U))); V
\end{aligned}
$$

注意，在上述四种情况中的每一种情况下，表达式 $\mu_* \; U$ 都会在经过 μ-归约后再次出现（且没有变化）。因此，归约序列中可能出现的不同子表达式 $\mu_* \; U$ 的数量是有限的，我们可以通过给表达式 $\mu_* \; U$ “标记”并在$\mu_* \; U$ 再次出现时停止，来检测无限序列。因此，我们可以设计一种算法来判定 T是否可以规范化：沿着归约序列进行，直到没有可进行的归约，或再次遇到先前“标记”过的 $\mu_* \; U$时终止。