arXiv:2601.15180v1 [cs.PL] 21 Jan 2026

会话类型的上下文元编程

Pedro ˆAngelo[1], Atsushi Igarashi[2], Yuito Murase[2], 和 Vasco T. Vasconcelos[1]

[1] 里斯本大学理学院，LASIGE，葡萄牙{pjangelo,vmvasconcelos}@ciencias.ulisboa.pt[2] 日本京都京都大学{murase@fos.,igarashi@}kuis.kyoto-u.ac.jp

摘要。我们提出将分阶段元编程集成到会话类型消息传递函数式语言中。我们基于一种具有多层上下文的上下文模态类型理论模型，其中，将任意项闭合于一系列变量之上的上下文值可以装箱并在消息中传递。接收到这样的值后，可以将其拆箱，在本地应用，然后运行。为阐明这种集成的动机，我们介绍了若干适合采用本系统的现实世界用例，例如服务器通过会话类型消息按需准备并发送代码。我们提出一种类型系统，用于区分线性（恰好使用一次）资源与非受限（使用次数不受限制）资源，并进一步定义了适用于具体实现的类型检查器。我们证明了类型保持性、顺序计算的进展性结果，以及并发运行时环境中不会发生运行时错误，并证明了类型检查器的正确性。

1 引言

元编程通过操作代码来生成并在运行时求值代码，使编译器尤其能够探索函数的某些参数是否可用，从而节省计算开销。本文关注这样一类程序设计语言：其中生成的代码在构造时即带有类型，并且代码可以引用一个为自由变量提供类型的上下文，这通常称为上下文定型[21,27,28,29]。在另一个正交的维度上，会话类型被倡导作为规范并发计的手段，通过准确描述用于在进程之间交换消息的信道所遵循的协议 [2,15,17,18,46,47,51]。

将会话类型与元编程相结合，使我们能够建立代码生成服务器，让它们与程序的其余部分并行运行，并通过带类型信道按需提供代码。线性性是会话类型的核心，但当前的元编程模型不支持这一特性。我们扩展了一个简单的上下文模态类型理论模型，使其支持会话类型，从而得到一种带多层上下文的按值调用线性 λ 演算。

我们简要讨论了所采取的设计决策。将会话类型与元编程相结合时，我们面临着若干种可选的理论基础，每个组成部分都有各自的优缺点。

我们的研究基于 Davies 和 Pfenning [12]，其中方框模态用于区分生成代码。我们进一步允许代码引用上下文中的变量，这通过上下文类型来描述，其思路沿袭了 Nanevski 等人的工作。 [29]Mœbius 语言由 Jang 等人提出， [21], 它在模态上下文类型理论的基础上进一步提供了对代码进行模式匹配、生成多态代码以及生成依赖于其他代码片段的代码的能力。我们不采用前两个方向，而是基于最后一个方向展开研究。我们提出一种支持会话类型的多层上下文模态线性 λ 演算；尤其是，代码片段的组合不会因装箱和拆箱而产生额外的管理性可归约式。

另一种出发点本可以是 Fitch 风格或 Kripke 风格表述，该表述支持 Lisp 引用/反引用，并将类型上下文视为用于建模不同计算阶段的栈 [10,27,50]。我们认为，let-box 方法能够简化向线性设定和会话类型的扩展。或者，时态类型 [11] 也可以是另一种方法，MetaML 和许多分阶段编程语言都采用了这种方法 [23,44,45,55]。然而，众所周知，时态类型系统在存在计算效应时是不健全的 [22,38,45], ，而基于信道的通信本质上具有副作用。因此，我们更倾向于采用 Mœbius 等上下文模态类型系统，因为它们与会话类型兼容。

关于会话类型，我们的方案在很大程度上遵循 Gay 和 Vasconcelos 的方案 [15]，该方案优雅地将函数式编程与运行时使用的进程演算相结合，为基于消息传递的并发编程语言提供了恰当的形式化模型。程序采用传统且直观的程序设计语言编写，无需像 Sano 等人近期提出的方案那样，迫使程序员转而使用进程层级 [39]。另一种可行的出发点是采用受线性逻辑启发的语言，以保证无死锁性，正如 Caires、Pfenning 和 Wadler 所提出的方案 [7,52]。然而，按照上述工作的思路强制消除死锁，会带来显著的句法限制；我们更倾向于避免这些限制，转而采用更灵活的设计。

值得强调的是，我们的贡献并非只是将 Mœbius 与 Gay 和 Vasconcelos 的语言结合起来。为了整合二者，我们需要引入（消息传递）并发，以及能够处理（递归）线性资源和非受限资源的线性类型系统。这一扩展构成本工作的关键技术贡献之一。

本文其余部分结构安排如下。在第 2 节，我们通过示例说明所提出的方法。第 3 节介绍项、进程、项求值和进程归约。第 4 节介绍类型和类型赋值系统。第 6 节介绍类型检查。第 7 节回顾相关工作。第

8 节总结本文，并指出未来工作的方向。附录中收录了额外示例（第 A 节）以及所有结果的证明（第 B 节）。

2 动机

我们首先通过几个示例，非正式地介绍本文提出的语言。首先，我们展示一个小程序，其中进程通过信道发送和接收代码片段，并执行这些代码片段，以此说明该语言语法和类型系统的基本思想。然后，我们讨论该语言如何对涉及通过信道传输代码片段的各种现实世界用例进行建模。

我们先展示如何生成代码片段，以便在流上发送固定数量的整数值。从在流上写入的进程一侧来看，流的类型如下。

类型 流 = ⊕{更多 : ! **Int** . 流 , 完成 : **Close** }

写入者可以选择发送更多 个值，也可以选择完成。在前一种情况下，写入者发送一个整数值（!整数），然后“回到起点”（流）；在后一种情况下，写入者必须关闭信道（关闭）。下面的函数sendFives 接收一个整数值，并返回一个代码片段。该代码片段需要一个信道端点（类型为流），并在执行时生成一个单位类型值；其类型写作 [流 ⊢ 单位类型], ，这称为盒类型。接下来，我们对整数参数进行模式匹配。

发送五个: 整数 → [流 ⊢ 单位类型]发送五个0 = 装箱(y . 关闭(选择 完成 y)) 发送五个 n = let 装箱u = 发送五个 (n − 1)in 装箱(x . u [发送5 (选择更多 x)])

当流上的所有值都已发送完毕（即n 为0时），剩下的就是选择完成，然后关闭信道。装箱项会在变量环境（即求值上下文）下生成代码；在此例中，该环境仅包含变量y，它表示信道端点。包含在装箱中的代码片段y. 关闭(选择完成 y) 称为上下文值。

当还有待发送的值时（即 n 不等于 0），我们会递归地计算用于发送 n−1 个值的代码，将其拆箱，并把得到的上下文值（例如 z. close (select Done z)）存入 u，然后准备发送第 n 个值的代码。let-box 绑定变量 u 的用法是：u[e], ，其中 e 是一个 Stream 类型的项，u 的上下文类型为 Stream ⊢ Unit（不带方括号 [ ]）。这种用法始终会得到一个 Unit 类型的项，其方式是将 e（不经求值）替换 z。部分 [e] 称为显式替换。在这个例子中，项 u[send 5 (select More x)] 将 Stream 类型的 u 应用于 Stream 类型的项 send 5 (select More x)。如果 u 是上下文值 z. close (select Done z)，那么 box x.u[send 5 (select More x)] 会求值为 box x. close (select Done (send 5 (select More x)))，这段代码会通过信道 x 发送数字 5，然后关闭该信道。

诸如 Stream ⊢ **Unit** 这样的上下文类型与对应的盒类型（[Stream ⊢ **Unit**]）之间的主要区别在于，后一类类型表示一等代码值——可以将其传递给另一个函数或存入数据结构——而前一类是二等类型：类型为 Stream ⊢ **Unit** 的变量只能通过显式替换来组合另一段代码。有关更多详情，请参阅上下文模态类型理论的相关研究（例如， [29]）。

现在，我们可以计算并运行代码，以发送固定数量的整数值。

```
send4Fives:流 →单位类型send4Fivesc =let装箱u
 =sendFives4inu[c]
```

项 sendFives 4 是一个类型为 [流 ⊢ 单位类型] 的装箱代码片段（一个项）。接着，u 是一个未装箱的代码片段（一个上下文值），其上下文类型为流 ⊢ 单位类型。我们为这个上下文值提供显式替换[c]。整个 **let** 项相当于运行代码

```
close (select Done (send 5 (select More (...send 5 (select
    More c)...))))
```

而无需调用函数 sendFives，也无需以其他任何形式使用递归。

下一个示例通过信道传输代码。设想一台服务器代表客户端准备代码。服务器使用信道与客户端交互：它先接收一个数字 n，然后回复发送n 个五的代码，最后等待信道关闭。通信信道的类型如下。

```
类型 Builder =?Int.![流 ⊢单位类型 ].等待
```

服务器在给定的信道上接收n，并通过调用sendFives计算代码。然后等待信道关闭。

```
serveFives:Builder →单位类型serveFivesc =let(n,c) =接收c
inwait(发送(sendFivesn)c)
```

信道的另一端是客户端：它发送一个数字（此处为 4），接收代码（类型为 [流 ⊢ 单位类型]），关闭信道并执行收到的代码。

```
sendFives':DualBuilder →流 →单位类型sendFives'cd =
let(code,c) =接收(发送4c)inclosec;let装箱u =代码in
u[d]
```

会话类型上的**Dual** 运算符提供了一种查看信道另一端的方式。在本例中，**Dual**Builder是类型!**Int**.?[Stream ⊢**Unit**].**Close**，其中! 会变为? ，而**Wait** 会变为 **Close** （反过来也同样成立）。注意，code 是一个类型为 [Stream ⊢ **Unit**], 的装箱代码片段，因此u 是相应的代码片段（类型为Stream ⊢ **Unit**），而u[d] 会在信道d上运行这段代码。

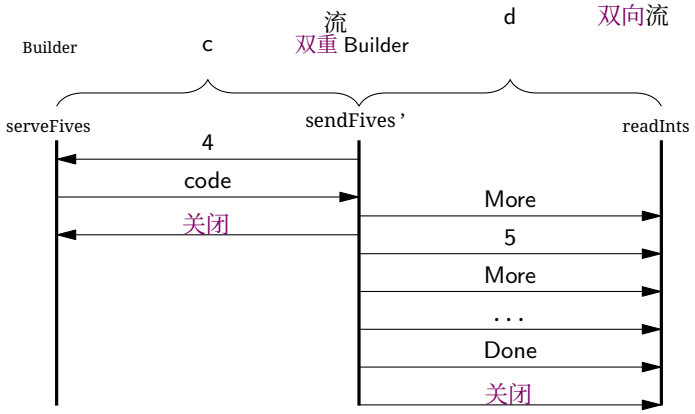

图1. sendFives 示例的消息序列图

要完成这个示例，我们需要一个用于读取流的函数，也就是一个类型为 **Dual**Stream →**Unit**的消费者。函数readInts 会读取并丢弃流中的所有整数值，然后等待流关闭。这里我们通过对信道上接收到的标签进行模式匹配来处理。

```
读取整数: 对偶流 → 单位类型读取整数（完成 c） = 等待读取整数（更多
c） = 令（, c） = 接收c于读取整数 c
```

最后，主线程分叉出两个线程——一个运行 serveFives，另一个收集整数值（读取整数）——然后继续执行 sendFives'。我们利用一个原语函数 forkWith：它接收一个挂起计算（thunk），新建一个信道，在该信道的一端分叉运行这个 thunk，并返回另一端供后续交互。

```
main: 单位类型
main =
  令c =通过分叉λ. serveFives）于 —c: 对偶 B u i l d e r 令d =通过分
  叉λ. 读取整数）于—d: 流发送五个 ’ c d
```

三个进程之间的交互如图 1所示，其中由函数 serveFives 生成并传输给 sendFives’ 的代码 片段为

```
装箱 (c.关 闭 (选 择 完成 ( 发送 5 (选 择 更多 ( . . . ( 选 择 更多 c ) . . . ) ) ) ) )
```

会话类型的上下文元编程建立在线性类型系统之上：每个资源都被归类为线性（恰好使用一次）或非受限（使用次数不限，包括零次）。信道端点始终是线性的，因此协议不会遇到意料之外的交互。函数可以是线性的，也可以是非受限的。我们目前看到的所有函数

都是非受限的，因为它们不会捕获自由线性值。递归函数（例如 sendFives ）必定是非受限的，因为它们会在递归结束时被丢弃。如果需要，也可以将其他一些值归类为线性值。一个例子是函数 serveFives ，它在函数 main中恰好使用一次。我们用 ω 标注非受限箭头，用 1 标注线性箭头。在示例中，我们常常省略 ω 标签。函数 serveFives 也可以具有类型 Builder $\to_1$单位类型。

装箱代码片段始终是非受限的，因此可以根据需要重复使用。下面的示例展示了如何复制（第一个示例）或丢弃（第二个示例）一个在给定信道上发送 5 的代码片段（记作 u，类型为 流 ⊢ 流）。

```
sendTwice : 流 → 单位类型 sendTwice = let 装箱 u = 装箱 (y . 发送
5 (select 更多 y)) in λx.close (select 完成 (u [u [x ]])
sendNone : 流 → 单位类型 sendNone = let 装箱 u = 装箱 (y . 发送
5 (select 更多 y)) in λx.close (select 完成 x)
```

在第一种情况下，两个 5 条消息通过信道 x 发送，随后关闭该信道；在第二种情况下，与该信道的唯一交互就是关闭它。第 A节包含一个多层级编程的示例。

sendFives 示例有些人为，主要用于说明将会话类型与元编程相结合的基本思想。然而，这种组合自然地以各种形式出现在现实世界的程序中，下面我们将对此进行讨论。

分布式计算。在分布式系统中，多台联网机器的计算能力被用于共同完成一个目标；这些机器不必位于同一地理位置或网络中。为了在节点之间分配任务，包含待执行代码的消息会通过网络发送出去。Apache Spark[56] 和 Hadoop [54], 等商业实现遵循受函数式编程启发的 MapReduce 编程模型 [13], 。一个相关概念是志愿计算，即志愿者（例如个人笔记本电脑）捐出闲置的处理能力来解决大规模问题。面向计算需求密集型科学项目的 Boinc [6], 就是一个例子：志愿者订阅后以代码形式接收任务，执行计算，然后返回结果。

考虑以下协议，它在素数搜索计算任务（例如 PrimeGrid[36]）的上下文中，规定了中央服务器与志愿者之间的通信。

```
类型 PrimeTask = &{ 任务: ! [Int⊢ 布尔值 ]. PrimeTask, 测试: !
Int .? 布尔值 . PrimeTask, 完成: 等待 }
```

服务器向已订阅的志愿者提供三种选择。第一种选择是任务，它会为志愿者提供一项待执行的任务。任务本身指定为盒类型 [整数 ⊢ 布尔值]。随后，任务会发送给

志愿者（![整数 ⊢ 布尔值]），然后协议从开头（PrimeTask）继续。之后，志愿者可以通过选择测试来捐出闲置的计算资源。服务器会发送一个要进行测试的数字（!整数），我们称之为素数候选数，然后等待志愿者返回结果，即真或假（?布尔值），并由志愿者保存该结果。交互再次回到开头。服务器还提供第三种选择完成，允许志愿者停止贡献，随后等待志愿者关闭连接（等待）。

假设有一个类型为 [整数 ⊢ 布尔值], 的 testPrime 函数，也就是说，它接受一个被拼接进来的数字，并返回**True** 或**False**，我们可以将服务器写成如下形式：

```
服务器: [整数 ] → [布尔值 ] →PrimeTask → [整数 ]
服务器csres (任务 s) =令 s =发送 testPrime于 服务器
csres s 服务器 (c : cs) res (测试 s) =令 s =发送 c于 let (re
sult,s) =接收 s于服务器cs (res ++ [result ]) s 服务器
res (完成 s) =等待 s;res
```

-

志愿者连接到服务器后，可以在以下三个选项中进行选择。选择任务后，志愿者会接收代码并保存起来以备后用。选择测试后，志愿者会接收素数候选数，并将候选数插入代码中运行测试。随后，测试结果会发送给服务器。志愿者可以随心所欲地重复这一操作，从而捐出更多处理时间。最后，当志愿者不再想要贡献时，只需选择完成并关闭连接即可。

```
志愿者: 双重 PrimeTask →单位类型志愿者 v =

  letv =select任务 vinlet(primeTest,v) =接收
  vinletv =select测试 vinlet(cand,v) =接收
  vinlet装箱 t =素数测试 in send(t [cand ]) vinl
  etv =select测试 vinlet(cand,v) =接收 vinl
  et装箱 t =素数测试 in send(t [cand ]) vin...let
  v =select完成 vin关闭 v
```

运行 main 程序时，交互过程如图 2所示。

```
main: [布尔值 ]main =lets =forkWith(λ.志愿者) in —s:
DualPrimeTaskserver[7..]s -
```

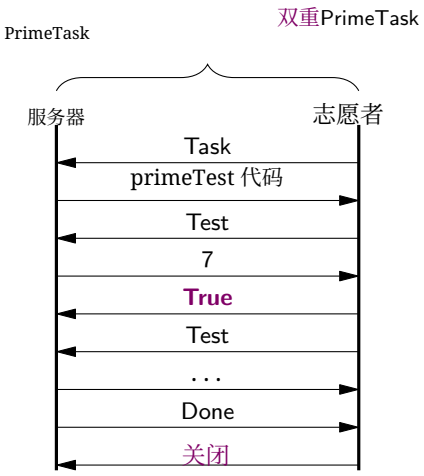

图2. PrimeTask 示例的消息序列图

然而，在交互过程中，素性测试算法可能会得到改进。因此，志愿者可以定期轮询以获取更新后的代码，通过再次选择任务 。

```
志 愿 者: 对偶PrimeTask  →单位类型志 愿 者 v=

  . . .令v  =选择任务 v于令( primeTest , v )  =
  接收v于. . .
```

在此示例中，会话类型用于管理任务分配的通信方面，而分阶段元编程则对通过消息发送代码进行建模。

计算卸载是指将计算密集型任务从资源受限的设备（无人机、物联网设备、智能手机）转移到性能更强大的远程服务器上。这种做法的主要优势在于能够克服这些设备的不足，尤其是存储容量小、处理能力弱和电池续航时间短等问题，从而提高效率并降低设备的能耗，Clonecloud 就体现了这一点。[9]这一概念在高吞吐量网络环境中尤为重要，因为通信方面的开销有所降低。我们的工作与计算卸载的概念密切相关：会话类型通过盒类型将以代码形式描述的任务卸载出去，如下方的类型所示。

```
类型 计算服务器  =&{ 计算: ？ [ ⊢整数 ] .!整数 . 计算服务器，完成: 等待  }
```

代码生成服务。 Sheard 和 Peyton Jones 提出的模板元编程 [42] 适用于 Haskell，可让人编写能够生成其他程序的元程序，例如能够操作任意大小数据结构的程序。例如，在 Haskell 中，可以编写如下元程序：fstN，它可以生成用于提取任意大小元组首个元素的函数；或 zipN，它可以生成 n 元 zip 函数。请看提供代码生成服务的服务器协议：

```
类型 代码生成器 =&{
  Fst3:![⊢(a,b,c) →a].代码生成器,Fst4:![⊢(a,b,c,d) →a].代
  码生成器,...Zip3:![⊢ [a] → [b] → [c] → [(a,b,c)]].代码生
  成器...完成:关闭 }
```

服务器向客户端提供用于操作任意大小数据结构的函数。为了满足请求，服务器会评估指定的模板元程序，将数据结构的大小嵌入其中，并将结果发送给客户端。服务器只存储函数模板并派生其实例（即运行元程序），因此随着其提供的函数实例越来越多，服务规模不断扩大时，服务器的存储需求仍然很低。虽然这个示例目前仍不可行，但它说明了我们这项工作的优势。

其他用例。 多层编程语言，例如 Hop.js [41],Ocsigen/Eliom [4] , Meteor.js [24] 和 Unison [48], 允许使用单一程序描述应用不同层级（例如客户端、服务器和数据库层）的功能，并将其统一在一种语言中。会话类型能够构建并验证客户端-服务器交互，同时提供与多层编程兼容的类型安全性。ScalaLoci [40] 和 Ur/Web [49] 是与多阶段编程相关的语言，但缺乏显式的通信模型。这些语言可以从会话类型中受益，例如确保跨计算阶段的正确性。

HTMX [19], Next.js [30] 和 React 服务器组件 [37] 是采用预渲染策略的 Web 框架：服务器预先渲染用户界面的部分内容并将其发送到客户端，同时还支持服务器组件，即仅在服务器上运行的代码。会话类型能够验证用户界面更新流程的正确性，而分阶段元编程有助于为预渲染策略奠定理论基础。

3 项、进程与操作语义

我们基于变量、 x、 y、 z 以及标签 l 的语法类别构建语言。我们用 $\overline{X}$ 表示由 $X_1 \cdots X_n$ 个对象组成的序列，其中 $n \geq 0$。空序列（当 $n = 0$ 时）记作 ε。项的语法见图 3。其中包括与信道相关的原语： 关闭 信道，等待 信道关闭，发送 给定值到信道，接收 从信道读取值，选择 l 在给定信道上发送标签 l ，以及

常量 $c ::=$ 关闭 | 等待 | 发送 | 接收 | 选择 l | 新建 | 分叉 | fix | 单位类型 上下文值 ρ, $\sigma ::= \overline{x}.M$值 $v, u ::= c \mid x[\varepsilon] \mid \lambda x.M \mid (v, v)$ |装箱 σ | 发送 v项 $M, N ::= v \mid x[\overline{\sigma}] \mid M\ M \mid (M, M)$ |let $(x, x) = M$in M |let box $x = M$in M |匹配 M with $\{l \to M_l\}^{l \in L}$进程 $P, Q ::= \langle M \rangle \mid P \mid Q \mid (\nu x x)P$求值上下文 $E ::= [] \mid E\ M \mid v\ E \mid (E, M)$ |(v, E) |let $(x, x) = E$in M |匹配 $E$with $\{l \to M_l\}^{l \in L}$ | let box $x = E$in M

图 3. 项与进程

new 用于创建信道。fork 会创建一个新线程，fix 是按值调用的定点组合子。单位类型只有一个值，可以将其视为其他基本值的占位符，例如示例中出现的整数或布尔值。

其他项还包括常量、模态变量（同时扮演项变量和信道端点的双重角色）、λ 抽象与应用、对的引入与消去、装箱的引入与消去，以及一个 匹配项，用于根据标签 l 进行分支。第 2 节的示例forkWith使用了一个方便的函数，它将信道创建与进程生成组合起来：

```
forkWith f = let (x,y) = new () in fork (λ_.f y) ; x
```

我们将形如 $\overline{x}.M$ 的对象称为上下文值。它们表示由序列 $\overline{x}$中的变量参数化的代码片段 M 。上下文项变量 $x[\overline{\sigma}]$ 将 x 所描述的代码片段应用于上下文值 $\overline{\sigma}$。装箱项 σ 将上下文值 σ 所表示的代码片段转换为一个项；let-box $x = M$ in N 项会消去 M 中的装箱，并将其绑定到 x ，以供 N 使用。let-box 项允许通过消去 M 中的装箱并绑定其结果，将代码拼接到另一个代码片段（N）中。当 $\overline{x}$ 为空序列时，有时会用 $\varepsilon.M$代替上下文值 M 。类似地，当 $\overline{\sigma}$ 为空序列时，有时会用 $x[\varepsilon]$代替上下文项变量 x 。信道端点始终表示为项$x[\varepsilon]$，它们属于值。

该语言中的绑定如下。变量 x 在 M 中被绑定，出现在项 $\lambda x.M$ 和let box $x = N$ in M中；变量 x 和 y 在 M 中被绑定，出现在项let $(x, y) = N$in M中；$\overline{x}$中的变量在 M中被绑定，出现在上下文值 $\overline{x}.M$中。项中约束变量和自由变量的集合（自由变量 M）据此定义。我们遵循变量约定：在所有上下文 [5,33]中，所有约束变量都与自由变量不同。

替换根据语言中的绑定来定义，同时利用变量约定。替换的定义相当常规

$$\text{R-Beta}\quad (\lambda x.M)\, v \to \{\varepsilon.v/x\}M \qquad \text{R-Split}\quad \mathsf{let}\ (x,y) = (u,v)\ \mathsf{in}\ M \to \{\varepsilon.u/x\}\{\varepsilon.v/y\}M$$

$$\text{R-LetBox}\quad \mathsf{let\ box}\ x = \mathsf{box}\,\sigma\ \mathsf{in}\ M \to \{\sigma/x\}M \qquad \text{R-Fix}\quad \mathsf{fix}\, v \to v\,(\lambda x.(\mathsf{fix}\, v)\, x) \qquad \text{R-Ctx}\quad \frac{M \to N}{E[M] \to E[N]}$$

图 4. 项求值 $M \to M$

，唯一不同之处在于，项语言包含应用模态变量 $x[\overline{\sigma}]$，而非普通变量 x。我们用 $\{\sigma/x\}M$ 表示将项M中变量 x 的自由出现替换为上下文值 σ 后得到的项，对于 $\{\sigma/x\}\rho$ 也采用类似的记法。下面详细说明这些新内容。

$$\{\overline{z}.M/x\}(x[\overline{\rho}]) = \{\{\overline{z}.M/x\}\overline{\rho}/\overline{z}\}M \qquad \{\sigma/x\}(y[\overline{\rho}]) = y[\{\sigma/x\}\overline{\rho}] \quad \text{if } x \neq y$$

$$\{\sigma/x\}(\overline{y}.M) = \overline{y}.\{\sigma/x\}M$$

多重替换 $\{\overline{\rho}/\overline{z}\}M$ 仅在 $\overline{\rho}$ 和 $\overline{z}$ 是长度相同的序列时才有定义。变量约定确保 $\overline{z}$ 中的变量两两不同，并且在 M之外没有自由出现，因此可以依次替换 $\overline{z}$ 中的各个变量。对应用变量 $\{\overline{z}.M/x\}(x[\overline{\rho}])$进行替换时，会进一步触发对 M中 $\overline{z}$ 的替换。例如，$\{z_1, z_2.\,$发送 $z_1\, z_2/x\}(x[y, 42])$ $=$ $\{y/z_1\}\{42/z_2\}($ 发送 $z_1\, z_2)$ $=$ 发送 $y\, 42$。对其余项构造子的替换是同态的；例如，$\{\sigma/x\}($ 装箱 $\rho)$ $=$ 装箱 $(\{\sigma/x\}\rho)$，以及 $\{\sigma/x\}($let 装箱 $y = M$in $N)$ $=$let 装箱 $y =\{\sigma/x\}M$ i n $\{\sigma/x\}N$。由于对应用变量的替换会触发另一次替换，良定义性并非显而易见。我们将在 B 节证明，对可定型项的替换是良定义的（前提是满足一些额外的侧条件）。

如果说项构成了语言的顺序部分，那么进程则处理并发，且仅用于运行时。程序员编写项 M，这些项会在初始线程 $\langle M\rangle$ 上运行，该线程最终可能分叉出新线程并创建新的通信信道。进程 $P \mid Q$ 表示两个进程的并行组合。进程 (νxy) P在进程 P中引入一个通信信道，该信道由两个端点 x 和 y描述。我们的做法在文献中相当常见 [15]。

项上的求值由关系 $M \to N$ 给出，其规则定义于图 4 中。该关系包括函数应用（R-Beta）、对值对的消去（u, v），以及将它们绑定到变量 x 和 y，并用于 M (R-Split)。请注意，替换仅针对上下文值定义，因此我们将值 u 提升为 $\varepsilon.u$，v 也同样如此。规则 R-LetBox 对装箱代码片段 σ 进行拆箱，并将其绑定到 x，以便在 M 中使用。规则 R-Fix 展开按值调用的定点构造子 fix [26]。最后，规则 R-Ctx 在求值上下文中对项进行求值，其语法见图 3。这些是标准的按值调用求值上下文。

$$P \equiv \langle \text{单位类型} \rangle \mid P \qquad P \mid Q \equiv Q \mid P \qquad (P_1 \mid P_2) \mid P_3 \equiv P_1 \mid (P_2 \mid P_3)$$

$$(\nu xy)P \mid Q \equiv (\nu xy)(P \mid Q) \qquad (\nu xy)P \equiv (\nu yx)P$$

$$(\nu xy)(\nu zw)P \equiv (\nu zw)(\nu xy)P$$

图 5. 结构同余 $P \equiv Q$

$$\text{R-Exp}\ \frac{M \to N}{\langle M \rangle \to \langle N \rangle} \qquad \text{R-Fork}\ \langle E[\mathsf{fork}\, v] \rangle \to \langle E[\mathsf{unit}] \rangle \mid \langle v\, \mathsf{unit} \rangle \qquad \text{R-New}\ \langle E[\mathsf{new}] \rangle \to (\nu xy)\langle E[(x, y)] \rangle$$

$$\text{R-Close}\ (\nu xy)(\langle E[\mathsf{close}\, x] \rangle \mid \langle F[\mathsf{wait}\, y] \rangle) \to \langle E[\mathsf{unit}] \rangle \mid \langle F[\mathsf{unit}] \rangle$$

$$\text{R-Com}\ (\nu xy)(\langle E[\mathsf{send}\, v\, x] \rangle \mid \langle F[\mathsf{receive}\, y] \rangle) \to (\nu xy)\langle E[x] \rangle \mid \langle F[(v, y)] \rangle$$

$$\text{R-Branch}(\nu xy)(\langle E[\text{选择 } l'\, x] \rangle \mid \langle F[\text{匹配 } y \text{ 与 } \{l \to M_l\}^{l \in L}] \rangle) \to (\nu xy)(\langle E[x] \rangle \mid \langle F[M_{l'}\, y] \rangle)$$

$$\text{R-Par}\ \frac{P \to P'}{P \mid Q \to P' \mid Q} \qquad \text{R-Res}\ \frac{P \to Q}{(\nu xy)P \to (\nu xy)Q} \qquad \text{R-Struct}\ \frac{P \equiv P' \quad P' \to Q' \quad Q' \equiv Q}{P \to Q}$$

图 6. 进程归约 $P \to P$

注意，我们不允许在装箱内部进行归约，就像我们不允许在 λ 下进行归约一样。

进程归约 由以下关系给出 $P \to Q$，其定义规则见图 6。按照 π-演算中的惯例，归约建立在另一个关系——结构同余 $P \equiv Q$——之上，该关系规定了进程的语法重排，同时为其进行归约做好准备 [25]。

结构同余是由图中公理生成的最小同余关系。5。第一条公理规定，线程 ⟨单位类型⟩ 是并行组合的单位元。接下来的两条公理说明并行组合满足交换律和结合律。定律（νxy）$P \mid Q \equiv$ （νxy）（$P \mid Q$）称为作用域外延，它允许根据需要将 ν-绑定子的作用域扩展到一个新进程 Q ，或从该进程中收回作用域。其他公理允许交换信道两个端点的顺序，以及交换 ν-绑定子的顺序。在作用域外延中，由于变量约定，条件“x， y 在 Q中不是自由变量”是多余的： x 和 y 在（νxy）P 中以受约束变量的形式出现，因此不可能在 Q中自由出现。

重数 $m ::= 1 \mid \omega$类型 $T, U ::=$ 单位类型 $\mid T \to_m T \mid T \times T \mid \square\tau^{n+1} \mid S$会话类型 $R, S ::=$ 关闭 $\mid$等待 $\mid !T.S \mid ?T.S \mid \oplus\{l\colon S_l\}^{l\in L} \mid \&\{l\colon S_l\}^{l\in L} \mid a \mid \mu a.S$上下文类型 $\tau^n ::= \overline{\tau} \vdash^n T$ 类型上下文 $\Gamma, \Delta ::= \varepsilon \mid \Gamma, x\colon \tau$

图 7. 类型

现在我们描述归约规则。规则 R-Exp 将项求值提升为进程归约。规则 R-Fork 新建一个线程。值 v 应当是一个挂起计算（一个 thunk），这样 v单位类型就会运行该计算。原线程中上下文里的空位由单位类型填入：两个线程之间的所有通信都必须通过在 v中空闲的信道上进行消息传递来完成。规则 R-New 新建一个信道，以其两个端点 x和 y表示。线程中的空位由一对（x, y）填入，以便线程可以操作该信道。

接下来的三条规则通过端点 x、y操作信道。为便于阅读，我们将上下文项值 $x[\varepsilon]$ 简写为 x，对 y也作同样处理。规则 R-Close 关闭一个信道：一个线程必须关闭其中一个端点，另一个线程则等待信道关闭。由于该信道不能再使用，因此移除 ν-绑定符。规则 R-Com 在发送线程和接收线程之间交换一个值。发送的结果是端点本身，即 x；接收的结果是由交换的值和端点 y组成的一对。随后，两个线程可以继续通过这些端点交换值。规则 R-Branch 执行匹配进程中分支的选择。一个线程上选定的标签（l'）决定另一个线程上的分支 $M_{l'}$ 。选择的结果是端点本身，与发送相同；匹配的结果是 $M_{l'}$，它应当是一个函数，并应用于端点 y。与 R-Com类似，两个线程可以继续在信道 xy上交互。最后三条规则允许在并行组合和 ν-绑定符下进行归约，并将结构同余纳入归约。

4 类型与类型赋值

本节介绍类型的概念，以及针对项和进程的类型赋值系统。

我们在类型引用,a, b的句法范畴上构建类型语言。我们还使用字母 k, n 表示自然数，并可称它们为层级。 类型的语法见图 7。重数 m包括 1 ，表示线性资源——必须恰好使用一次的资源——以及 ω ，表示非受限资源——可使用任意多次（包括零次）的资源。

$$\text{关闭} \perp \text{等待} \qquad \text{等待} \perp \text{关闭} \qquad \frac{S \perp R}{!T.S \perp ?T.R} \qquad \frac{S \perp R}{?T.S \perp !T.R}$$

$$\frac{R \perp S\{\mu a.S/a\}}{R \perp \mu a.S} \qquad \frac{R\{\mu a.R/a\} \perp S}{\mu a.R \perp S} \qquad \frac{S_l \perp R_l \quad \forall l \in L}{\oplus\{l : S_l\}^{l \in L} \perp \&\{l : R_l\}^{l \in L}}$$

$$\frac{S_l \perp R_l \quad \forall l \in L}{\&\{l : S_l\}^{l \in L} \perp \oplus\{l : R_l\}^{l \in L}}$$

图 8. 类型对偶（余归纳） $S \perp S$ □

类型 Unit（表示常量单位）可以看作其他基本类型的占位符，例如表示整数或布尔值的类型。其他类型还包括线性函数和非受限函数 $T \to_1 U$ 和 $T \to_\omega U$、线性对 $T \times U$（为简单起见，我们不考虑非受限对），以及盒类型 $\Box\tau$，即由上下文类型 τ刻画的代码片段。上下文类型的语法按层级索引；当层级无关紧要时，我们会省略层级，并将上下文类型简写为 τ。

会话类型描述通信信道端点。类型关闭和等待分别表示已准备关闭或等待关闭的信道。类型 $!T.S$和 $?T.S$ 分别表示发送或接收类型为 T 的值，并继续为 S 的信道。类型 $\oplus\{l\colon\ S_l\}^{l\in L}$ 表示信道可以选择一个分支 $k \in L$ 并继续为 S_k（内部选择）。其对偶类型 $\&\{l\colon\ S_l\}^{l\in L}$表示信道提供一组选项标签（外部选择）。递归会话类型由类型引用 a 和递归 $\mu a.S$构成。我们对类型采用等递归处理方式，不区分递归类型与其展开形式 [1,33]。

会话类型中的对偶概念由关系 $R \perp S$ 表示，见图 9。类型 等待和关闭；输入与输出；以及内部选择与外部选择互为对偶。递归类型会被展开。该定义是余归纳的，因此我们接受可能无限的推导。详情请参阅 Gay 等人 [14]。

盒类型的形式为 $\overline{\tau} \vdash^n T$，其中 $\overline{\tau}$ 表示一个可能为空的上下文类型序列。此类类型表示一个类型为 T 的代码片段 M，它由一系列上下文类型为 $\overline{\tau}$ 的变量 $\overline{x}$ 参数化，这些变量合称为上下文 [21,27]。普通（模态）变量的层级为 0。普通变量的上下文始终为空，因此可以将上下文类型 $\varepsilon \vdash^0 T$ 写作 T。此外，对于类型 $\overline{\tau} \vdash^{n+1} T$，我们要求$\overline{\tau}$ 中所有上下文类型的层级都小于或等于 n。$\overline{\tau}$ 中的类型刻画了上下文值 $\overline{x}.M$ 的参数 $\overline{x}$，因此其层级必须低于值本身的层级。我们通过一族类型形成谓词来表达这些直观认识：T ctype^n，其定义规则如下。

$$(\varepsilon \vdash^0 T)\ \text{ctype}^0 \qquad \frac{\overline{\tau}\ \text{ctype}^n}{(\overline{\tau} \vdash^{n+1} T)\ \text{ctype}^{n+1}} \qquad \frac{\tau\ \text{ctype}^n}{\tau\ \text{ctype}^{n+1}}$$

$$\frac{T\ \omega}{(\varepsilon \vdash^0 T)\ \omega} \qquad (\overline{\tau} \vdash^{n+1} T)\ \omega \qquad \underset{\omega}{\text{单位类型}} \qquad (T \to_\omega U)\ \omega \qquad (\Box\tau)\ \omega$$

图 9. 无限制类型 $T\ \omega$ $\tau\ \omega$

此后我们假设所有上下文类型 τ 都是良构的，也就是说，$\tau\ \mathrm{ctype}^n$ 对某个 n成立。例如，对象 (等待 $\vdash^1 U$) $\vdash^1 T$ 不能被视为良构类型。这种表述遵循 Mœbius 的发展及其种类系统 [21]。

最后，盒类型 $\Box\tau^{n+1}$表示上下文类型为 τ 的代码片段。装箱代码片段的类型位于从 1 开始的层级上。

类型表示只能使用恰好一次，或可无限次使用的资源。谓词 ω 对值可无限次使用的类型和上下文类型为真。其定义见图 9。层级为 1 或更高的上下文类型都是非受限的，而层级为 0 的上下文类型只有在代码的类型 T 为非受限时才是非受限的。层级 0 表示普通代码（类型为 T），因此 $\varepsilon \vdash^0 T$ 在 T 为非受限时也是非受限的。另一方面，高于 0 的层级表示可随意复制或丢弃的装箱代码片段，因此它们全都是非受限的。对于类型而言，单位类型、非受限箭头 $T \to_\omega U$ 和盒类型 $\Box\tau$ 是仅有的非受限类型。图 9 中的规则是算法式的。因此，谓词 ω 是可判定的，这使我们可以讨论其否定。非非受限的上下文类型 τ 称为线性的。相应的谓词记作 τ 1。另外两个谓词用于描述上下文类型层级的上界和下界。谓词 $< k$ 对满足 $\overline{\tau} \vdash^n T$ 的上下文类型 $n < k$为真。谓词 $\geq k$ 对满足 $\overline{\tau} \vdash^n T$ 的上下文类型 $n \geq k$为真。

类型上下文 Γ、Δ 将变量 x 绑定到上下文类型 τ。我们假设同一上下文中不会有变量被重复绑定，并将类型上下文视为在交换各个条目的顺序下等价。其余两个子结构规则——收缩和弱化——通过上下文拆分（如下所述）以及在公理中允许包含无限制类型的上下文来处理。然后，按照标准方式将这四个谓词提升到类型上下文。比如，$\Gamma\ \omega$ 对于满足以下条件的上下文 $x_1 :\ \tau_1$，……，$x_k :\ \tau_k$为真，其中 $\tau_1\ \omega$，……，$\tau_k\ \omega$。直观地说，$\Gamma < n$ 对于适合层级为 n 的代码片段中的局部变量的上下文 Γ为真，而$\Gamma \geq n$ 对于适合外部变量的上下文为真。注意，除了空上下文 ε之外，不存在满足 $\Gamma < 0$ 的上下文 Γ。另一方面，$\Gamma^{\geq 0}$ 是一个重言式。

我们使用记号 τ^ω 表示满足 $\tau\ \omega$ 的上下文类型 τ，对于 τ^1也同样如此。对于上下文，我们使用记号 Γ^ω表示满足 $\Gamma\ \omega$ 的上下文 Γ，对于 $\Gamma^{<n}$ 和 $\Gamma^{\geq n}$也同样如此。

这些常量的类型由图中的类型方案描述。10。会话类型的类型大多取自 Gay 和 Vasconcelos [15]。常量关闭和等待会关闭信道的不同端，并返回单位类型。常量发送

$$
\begin{aligned}
\mathrm{typeof}(\mathsf{close}) &= \mathsf{Close} \to_\omega \mathsf{Unit} \\
\mathrm{typeof}(\mathsf{wait}) &= \mathsf{Wait} \to_\omega \mathsf{Unit} \\
\mathrm{typeof}(\mathsf{send}) &= T \to_\omega !T.S \to_1 S \\
\mathrm{typeof}(\mathsf{receive}) &= ?T.S \to_\omega T \times S \\
\mathrm{typeof}(\mathsf{select}\ k) &= \oplus\{l : S_l\}^{l \in L} \to_\omega S_k, \text{with } k \in L \\
\mathrm{typeof}(\mathsf{new}) &= \mathsf{Unit} \to_\omega R \times S, \text{with } R \perp S \\
\mathrm{typeof}(\mathsf{fork}) &= (\mathsf{Unit} \to_1 \mathsf{Unit}) \to_\omega \mathsf{Unit} \\
\mathrm{typeof}(\mathsf{fix}) &= ((T \to_\omega T) \to_\omega (T \to_\omega T)) \to_\omega (T \to_\omega T) \\
\mathrm{typeof}(\mathsf{unit}) &= \mathsf{Unit}
\end{aligned}
$$

图10. 常量的类型方案 $\mathrm{typeof}(c) = T$

$$
\varepsilon = \varepsilon \circ \varepsilon \qquad \frac{\Gamma = \Gamma_1 \circ \Gamma_2}{\Gamma, x\colon \tau^1 = (\Gamma_1, x\colon \tau^1) \circ \Gamma_2} \qquad \frac{\Gamma = \Gamma_1 \circ \Gamma_2}{\Gamma, x\colon \tau^1 = \Gamma_1 \circ (\Gamma_2, x\colon \tau^1)}
$$

$$
\frac{\Gamma = \Gamma_1 \circ \Gamma_2}{\Gamma, x\colon \tau^\omega = (\Gamma_1, x\colon \tau^\omega) \circ (\Gamma_2, x\colon \tau^\omega)}
$$

图 11. 上下文拆分 $\Gamma = \Gamma \circ \Gamma$

接受一个值和一个用于发送该值的信道，并返回续延通道；第二个箭头是线性的，因为 T 可能表示线性值。常量接收接受一个信道，并返回由读取的值和续延通道组成的对。常量选择 k 接受一个外部选择类型，选择分支 k，并返回相应的续延通道。常量新建表示一个挂起计算；调用它（传入单位值）时，会返回一对具有对偶类型的信道端点。常量分叉接收一个挂起计算，生成一个新线程来运行该计算，并返回单位类型。最后，fix 的类型是按值调用 λ 演算中的不动点组合子的类型；所有箭头均为非受限，因为参数函数可以使用任意多次（在每次递归调用中），包括零次（在基本情形中），而 fix 本身也可以使用任意多次。

有时，类型系统必须将来自不同子推导的类型上下文合并为单个上下文。上下文拆分（从下往上读）可以完成这项工作。规则见图11，并逐字取自 Walker [53]：线性类型向左或向右移动（但不能同时向两个方向移动）；无限制类型则被复制到两个上下文中。

类型判断的形式为 $\Gamma \vdash M : T$，表示项 M 在类型上下文 Γ下具有类型 T。这些判断的规则将在下文介绍，见图 12。

$$\textsc{T-Const}\quad \Gamma^{\omega} \vdash c : \mathsf{typeof}(c) \qquad \textsc{T-Var}\quad \frac{\overline{\Gamma} \vdash \overline{\sigma} : \overline{\tau}}{(\Delta^{\omega}, x\colon (\overline{\tau} \vdash^{n} T)) \circ \overline{\Gamma} \vdash x[\overline{\sigma}] : T}$$

$$\textsc{T-LinFunI}\quad \frac{\Gamma, x\colon (\varepsilon \vdash^{0} T) \vdash M : U}{\Gamma \vdash \lambda x.M : T \to_{1} U} \qquad \textsc{T-UnFunI}\quad \frac{\Gamma^{\omega},\ x\colon (\varepsilon \vdash^{0} T) \vdash M : U}{\Gamma^{\omega} \vdash \lambda x.M : T \to_{\omega} U} \qquad \textsc{T-FunE}\quad \frac{\Gamma \vdash M : T \to_{m} U \quad \Delta \vdash N : T}{\Gamma \circ \Delta \vdash M\,N : U}$$

$$\textsc{T-PairI}\quad \frac{\Gamma \vdash M : T \qquad \Delta \vdash N : U}{\Gamma \circ \Delta \vdash (M, N) : T \times U} \qquad \textsc{T-PairE}\quad \frac{\Gamma \vdash M : T \times U \quad \Delta, x\colon (\varepsilon \vdash^{0} T)\ y\colon (\varepsilon \vdash^{0} U) \vdash N : V}{\Gamma \circ \Delta \vdash \text{令 } (x, y) = M \text{ 于 } N : V}$$

$$\textsc{T-Ctx}\quad \frac{\Gamma^{\geq n}, \overline{x\colon \tau}^{<n} \vdash M : T}{\Gamma^{\geq n} \vdash \overline{x}.M : (\overline{\tau} \vdash^{n} T)} \qquad \textsc{T-BoxI}\quad \frac{\Gamma^{\omega} \vdash \sigma : \tau}{\Gamma^{\omega}, \Delta^{\omega} \vdash \text{装箱 } \sigma : \Box\tau}$$

$$\textsc{T-BoxE}\quad \frac{\Gamma \vdash M : \Box\tau \qquad \Delta, x\colon \tau \vdash N : T}{\Gamma \circ \Delta \vdash \mathsf{let}\ \text{装箱 } x = M\ \mathsf{in}\ N : T} \qquad \textsc{T-Match}\quad \frac{\Gamma \vdash M : \&\{l : S_l\}^{l \in L} \quad \Delta \vdash N_l : S_l \to_{1} T \quad (\forall l \in L)}{\Gamma \circ \Delta \vdash \text{匹配 } M \text{ 时 } \{l : N_l\}^{l \in L} : T}$$

图1 2. 项构造 $\Gamma \vdash M : T$

上下文项是形如 $\overline{x}.M$ 的闭包，它将项 M 对变量序列 $\overline{x}$ 进行闭合。当假设 $\overline{x}$ 中的变量具有 $\overline{\tau}$ 中的类型时，若 T 的类型为 M，则上下文项具有上下文类型 $\overline{\tau} \vdash^{n} T$（规则 T-Ctx）。直观地说， M 的自由变量分为两组：局部变量和外部变量。局部变量 $\overline{x}$ 在上下文项 $\overline{x}.M$ 中被明确区分，并在上下文 $x\colon \tau^{<n}$ 下进行类型判断，其中 n 是分配给每个局部变量 x_i （位于 $\overline{x}$ 中）的层级 k_i 的上界。每个局部变量的层级可以任意指定，只要小于 n 即可。外部变量在上下文 $\Gamma^{\geq n}$ 下进行类型判断。因此，自然数 n 表示代码片段 $\overline{x}.M$ 进行类型判断时所处的层级。通常，一个上下文项可以在不同层级进行类型判断。结论中的上下文内所有资源（变量），以及资源 $\overline{x}$ ，都会在 M 的推导过程中被消耗。

模态变量的形式为 $x[\sigma_1 \cdots \sigma_k]$，表示一个应用于上下文值 $\sigma_1 \cdots \sigma_k$ 的上下文项。要对每个 σ_i 进行类型判断，我们需要一个单独的上下文Γ_i。 x 的类型必须是形式为 $\tau_1 \cdots \tau_k \vdash^{n} T$ 的上下文类型，并且每个上下文项 σ_i 的类型都必须为 τ_i。由于 x 可能在任意一个 σ_i 中自由出现，我们使用上下文拆分，允许条目 $x\colon (\tau_1 \cdots \tau_k \vdash^{n} T)$ 出现在每个 Γ_i 中。在这种情况下，上下文类型必须是非受限的。

在规则 T-Var 中，所有序列——$\overline{\Gamma}$、 $\overline{\sigma}$ 和 $\overline{\tau}$——的长度必须相同。当这些序列为空时，该规则就成为变量的常规公理

。事实上，规则 T-Var 是 Walker 的线性 λ-演算中公理的自然推广 [53], ，它通过将上下文中的类型 $\varepsilon \vdash^n T$ 写作 T，并将模态变量 $x[\varepsilon]$ 写作 x，从而得到 Δ^ω， $x\colon(\varepsilon \vdash^n T) \vdash x[\varepsilon]\colon T$ 可缩写为 Δ^ω， $x\colon T \vdash x\colon T$。该公理说明了为何结论 Δ中会出现非受限上下文 T-Var。

我们暂且中断类型系统的介绍，来讨论替换。在线性类型系统中，替换原理颇为棘手，因为替换可能会复制或丢弃项 [15,53]。此外，我们还必须考虑类型层级。由此得到的替换原理可以表述如下。

$$\frac{\Gamma \vdash \sigma : \tau^n \qquad \Delta, x\colon \tau^n \vdash M : T \qquad (\tau\ 1 \text{ or } \Gamma\ \omega) \qquad (\Gamma \geq n)}{\Gamma \circ \Delta \vdash \{\sigma/x\}M : T}\ \text{SUBS}$$

我们用 $\Gamma \circ \Delta$ 表示上下文 Θ，使得 $\Theta = \Gamma \circ \Delta$。这样做时，我们假定上下文拆分操作有定义。第一个附加条件 $\tau^n\ 1$ 或 $\Gamma\ \omega$ 可防止通过这种方式进行替换 $\{\sigma/x\}M$ 时， σ 中线性类型的自由变量被复制或丢弃：如果 τ^n 是线性的，就不会发生复制或丢弃；如果 $\Gamma\ \omega$，那么自由变量可以安全地被复制或丢弃。第二个附加条件 $\Gamma \geq n$ 意味着， σ 中自由变量的层级必须等于或高于 σ 的层级。

如果 $n = 0$，也就是说，当 x 是一个普通变量时，条件 $\Gamma \geq n$ 显然成立。尽管该条件中的第二个公式通常并不成立，但我们可以证明它对值成立（也就是说，如果 $\sigma = \varepsilon.v$ 对于某个 v成立）。因此，我们可以得到传统的（按值调用）替换原理:

$$\frac{\Gamma \vdash \varepsilon.v : (\varepsilon \vdash^0 U) \qquad \Delta, x\colon (\varepsilon \vdash^0 U) \vdash M : T}{\Gamma \circ \Delta \vdash \{\varepsilon.v/x\}M : T} \quad \text{abbreviated to} \quad \frac{\Gamma \vdash v : U \qquad \Delta, x\colon U \vdash M : T}{\Gamma \circ \Delta \vdash \{v/x\}M : T}$$

回到类型系统，我们讨论线性 λ 演算中针对λ-抽象和应用的传统类型规则。这些规则来自 Walker [53] ，但必须对引入规则作必要调整，以处理包含上下文类型的上下文。这里有两个引入规则，一个针对线性箭头，另一个针对非受限箭头。在这两种情况下， λ-约束变量 x 都局部于 M，因此其层级为 0。我们通过为 x 赋予形如 $\varepsilon \vdash^0 T$的类型来记录这一事实。对于非受限箭头，我们要求使用非受限上下文，因为该上下文包含项$\lambda x.M$ 中自由变量的条目，而该项是非受限的，因此可以被复制或丢弃。

盒引入规则和盒消去规则如下。在规则 T-BoxI 中，如果上下文项 σ 具有上下文类型 τ，那么装箱的类型就是类型 $\Box\tau$ 。盒消去规则的上下文 T-BoxE 分为两部分：一部分（Γ）用于为 M 定型，另一部分（Δ）用于为 N 定型。项 M 必须表示一个装箱代码片段，因此 M 的类型必须是 $\Box\tau$。项 N 在上下文 Δ 中定型，该上下文扩展了一个关于 x 的条目。

带有上下文拆分算子的线性类型系统会在推导的叶节点处执行所有必需的弱化。因此，所有公理都要求 Γ^ω 上下文能够

丢弃非限制变量。这适用于规则 T-Const 和 T-Var （其中$\overline{\sigma}$ 为空），在大多数情况下这就足够了。不过，在对装箱进行类型检查时，我们需要弱化的原因不同：为了丢弃层级较低的非受限资源（即层级低于装箱内代码片段进行类型检查所需层级的资源）。上下文 Δ^{ω} 承担这一作用；我们期望此上下文包含所有层级低于 n 的非受限条目。无论如何，线性资源都不能被丢弃。

规则 T-BoxI 与先前的工作完全一致[12,21,28], ，其中层级控制在盒引入时进行检查。由 T-Ctx fol-lowedby T-BoxI 组成的一个导出规则，与 Mœbius 的盒引入规则相同 [21]:

$$\frac{\Gamma^{\omega}, \overline{x\colon \tau}^{<n} \vdash M : T \qquad (\Gamma^{\omega} \geq n)}{\Gamma^{\omega}, \Delta^{\omega} \vdash \mathsf{box}\,(\overline{x}.M) : \Box(\overline{\tau} \vdash^{n} T)}$$

5 主要结果

本节讨论不同类型构造器的局部可靠性和局部完备性，以及进程语言的类型安全性结果。

为确保引入规则和消去规则构成有效的类型构造器，必须证明该类型构造器 [12,32] 满足局部可靠性和局部完备性条件。局部可靠性指出，直接先应用引入规则、再应用消去规则，不会获得任何额外信息。因此，完成推导不应需要这类迂回步骤，因为不经过它也能得到相同的判断。与之对偶的局部完备性指出，只需应用引入规则，就能恢复应用消去规则之前的所有信息。局部可靠性具体体现为归约规则，而局部完备性具体体现为扩展规则；局部可靠性的证明蕴含归约规则具有类型保持性。

我们首先研究箭头类型的引入规则和消去规则。对于局部可靠性，我们有:

$$\frac{\dfrac{\Gamma, x\colon (\varepsilon \vdash^{0} T) \vdash M : U}{\Gamma \vdash \lambda x.M : T \to_{\omega} U}\ \text{T-LinFunI} \qquad \Delta \vdash v : T}{\Gamma \circ \Delta \vdash (\lambda x.M)\, v : U}\ \text{T-FunE} \quad \Rightarrow$$

$$\frac{\dfrac{\Delta \vdash v : T}{\Delta \vdash \varepsilon.v : (\varepsilon \vdash^{0} T)}\ \text{T-Ctx} \qquad \Gamma, x\colon (\varepsilon \vdash^{0} T) \vdash M : U}{\Gamma \circ \Delta \vdash \{\varepsilon.v/x\}M : U}\ \text{SUBS}$$

$\to_{\omega}$ 的情况，也就是规则 T-UnFunI against T-FunE 的情况，类似地可以通过将 ω 限制添加到上下文 Γ 中得到。

对于局部完备性，假设 x 不是 M 的自由变量，并令 τ 为上下文类型 $\varepsilon \vdash^{0} T$。我们必须根据 Γ和/或 T 是非受限还是线性，区分四种情况。首先考虑 Γ 和 T都是线性的情况。此时， τ是线性的，并且 Γ、$x\colon \tau = \Gamma \circ x\colon \tau$，这为以下规则 x 的实例提供了依据

见下方规则 T-FunE 。

$$\Gamma \vdash M : T \to_1 U \quad \Rightarrow$$

$$\dfrac{\dfrac{\Gamma \vdash M : T \to_1 U \qquad \dfrac{}{x\colon \tau \vdash x[\varepsilon] : T}\,\textsc{T-Var}}{\Gamma, x\colon \tau \vdash M\,x[\varepsilon] : U}\,\textsc{T-FunE}}{\Gamma \vdash \lambda x.(M\,x[\varepsilon]) : T \to_1 U}\,\textsc{T-LinFunI}$$

当 T 是非受限的，那么 τ 也是非受限的，因此我们有 $\Gamma, x\colon \tau = (\Gamma, x\colon \tau) \circ x\colon \tau$。我们使用弱化（见附录）将 $x\colon \tau$ 添加到 M 的上下文中。我们展示 Γ 也是非受限时的情况；其余两种情况类似。

$$\Gamma^\omega \vdash M : T \to_\omega U \quad \Rightarrow$$

$$\dfrac{\dfrac{\dfrac{\Gamma^\omega \vdash M : T \to_\omega U}{\Gamma^\omega, x\colon \tau \vdash M : T \to_\omega U}\,\textsc{weak} \qquad \dfrac{}{x\colon \tau \vdash x[\varepsilon] : T}\,\textsc{T-Var}}{\Gamma^\omega, x\colon \tau \vdash M\,x[\varepsilon] : U}\,\textsc{T-FunE}}{\Gamma^\omega \vdash \lambda x.(M\,x[\varepsilon]) : T \to_\omega U}\,\textsc{T-UnFunI}$$

现在我们讨论装箱引入和消去规则。对于局部可靠性，我们有以下推导，其中规则 WEAK 将非受限项引入上下文。

$$\dfrac{\dfrac{\dfrac{\cdots \qquad \Gamma^\omega \geq n}{\Gamma^\omega \vdash \sigma : \tau^n}\,\textsc{T-Ctx}}{\Gamma^\omega, \Delta^\omega \vdash \mathsf{box}\,\sigma : \Box\tau^n}\,\textsc{T-BoxI} \qquad \Theta, x\colon \tau^n \vdash N : T}{(\Gamma^\omega, \Delta^\omega) \circ \Theta \vdash \mathsf{let\ box}\ x = \mathsf{box}\,\sigma\ \mathsf{in}\ N : T}\,\textsc{T-BoxE} \quad \Rightarrow$$

$$\dfrac{\dfrac{\Gamma^\omega \vdash \sigma : \tau^n}{\Gamma^\omega, \Delta^\omega \vdash \sigma : \tau^n}\,\textsc{weak} \qquad \Theta, x\colon \tau^n \vdash M : T \qquad (\Gamma^\omega \geq n)}{(\Gamma^\omega, \Delta^\omega) \circ \Theta \vdash \{\sigma/x\}N : T}\,\textsc{subs}$$

对于局部完备性，我们首先为上下文类型定义 η 展开，记作 $\eta(x, \tau)$，其规则如下，其中 $\overline{y}$ 中的变量均取为新鲜变量。

$$\eta(x, \varepsilon \vdash^0 T) = \varepsilon.x[\epsilon] \qquad \eta(x, \overline{\tau} \vdash^{n+1} T) = \overline{y}.x[\overline{\eta(y, \tau)}]$$

η 展开 $\eta(x, \tau)$ 定义于任意上下文类型 τ，因为 τ 的层级在每一步都严格递减。我们可以确认以下引理成立；其证明通过对 τ 的层级进行归纳法得出，可在附录中找到。

引理 1. $x\colon \tau \vdash \eta(x, \tau) : \tau$。

现在我们证明局部完备性。令 Δ 为 Γ 的无限制部分。那么 $\Gamma = \Gamma \circ \Delta$。令 $\overline{p}$ 为层级低于 n 的上下文类型序列，并令 τ 为上下文类型 $\overline{p} \vdash^n \underline{T}$。我们有 $(x\colon \rho)^{<n}$ 和 $(u\colon \tau)^{\geq n}$。

$$\text{R-Exp}\ \frac{\Gamma \vdash M :\ \text{单位类型}}{\Gamma \vdash \langle M \rangle} \qquad \text{P-Par}\ \frac{\Gamma \vdash P \qquad \Delta \vdash Q}{\Gamma \circ \Delta \vdash P \mid Q} \qquad \text{P-Res}\ \frac{\Gamma, x\colon (\varepsilon \vdash^0 R)\ y\colon (\varepsilon \vdash^0 S) \vdash P \qquad R \perp S}{\Gamma \vdash (\nu xy)P}$$

图13。进程构造 $\Gamma \vdash P$ □

此外，我们可以看到上下文 $(u\colon \tau)^{\geq n}$ 是非受限的，因为 $n > 0$ 是 $\square\tau$ 良构性条件所要求的。因此我们有：

$$\Gamma \vdash M : \square\tau \quad \Rightarrow$$

$$\dfrac{\Gamma \vdash M : \square\tau \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{\overline{x\colon \rho} \vdash \overline{\eta(x,\rho)} : \overline{\rho}}\ \text{Lemma 1}}{\overline{x\colon \rho} \circ u\colon \tau \vdash u[\overline{\eta(x,\rho)}] : T}\ \text{T-Var}}{u\colon \tau \vdash \overline{x}.u[\overline{\eta(x,\rho)}] : \tau}\ \text{T-Ctx}}{\Delta, u\colon \tau \vdash \mathsf{box}\,(\overline{x}.u[\overline{\eta(x,\rho)}]) : \square\tau}\ \text{T-BoxI}}{\Gamma \circ \Delta \vdash \mathsf{let\ box}\ u = M\ \mathsf{in\ box}\,(\overline{x}.u[\overline{\eta(x,\rho)}]) : \square\tau}\ \text{T-BoxE}$$

本节的第二部分讨论类型安全性。即使程序员不应直接编写进程，我们仍然需要为它们赋予类型，以便陈述并证明进程语言的类型安全性结果。图13中的规则十分直接，取自 Vasconcelos [51] 和 Thiemann 等人 [2,47]。在线程 $\langle M \rangle$中，项 M 必须具有单位类型。鉴于求值结果会被丢弃，使用任何非受限类型都可以。并行组合会在结论处分割上下文，并为每个进程使用其中一部分。作用域限制会为两个信道端点分别引入条目。它们必须具有对偶类型。

我们基于保持性 和 进展性——定理 1 和2——遵循 Pierce 的做法 [33]。讨论进展性时，我们会考虑那些可能（暂时）因等待信道上的通信而陷入停滞的项。

定理 1（保持性）。

1. 若 $M \to N$ 且 $\Gamma \vdash M\colon T$，则 $\Gamma \vdash N\colon T$。
2. 若 $\Gamma \vdash P$ 且 $P \to Q$，则 $\Gamma \vdash Q$。

进展性通常针对闭项陈述。由于信道端点也是变量，我们必须考虑含有自由端点的项，也就是含有会话类型变量的项。我们称上下文 Γ 仅包含会话类型，并在 Γ的形式为 $x_1\colon \varepsilon \vdash^0 S_1$，……，$x_n\colon \varepsilon \vdash^0 S_n$时，记作 Γ^S。

定理 2（求值的进展性）。设 $\Gamma^S \vdash M\colon T$。则，

1. M 是一个值，或者

2. $M \to N$，对于某个 N，或者 3. M 的形式为 $E[N]$，对于某个 E，其中 N 具有以下形式之一：关闭 x、等待 x、发送 $v\ x$、接收 x、选择 $l\ x$、匹配 x 与 $\{l \to N_l\}^{l \in L}$、新建 v 或分叉 v。

进程语言由于可能出现死锁而不具备进展性。进程语言的安全性基于保持性和无运行时错误——定理1 和3——遵循 Honda 等人 [18]。为了处理进程语言不存在运行时错误这一性质，我们先给出几个定义。项的主体 M 是变量 x（表示一个信道端点），在以下情形中有定义，在其他所有情形中均未定义。

关闭 x　　等待 x　　发送 $v\,x$　　接收 x　　选择 $k\,x$

匹配 x 与 $\{l \to N_l\}^{l \in L}$

如果进程 $(\nu xy)(\langle E[M]\rangle \mid \langle F[N]\rangle)$ 是可约表达式，则

– M 是关闭 x 且 N 是等待 y，– M 是发送 $v\,x$ 且 N 是接收 y，– M 是选择 $l'\,x$ 且 N 是匹配 y 与 $\{l \to N_l\}^{l \in L}$ 且 $l' \in L$。

或者反过来，对于进程（νxy）（$\langle F[N]\rangle \mid \langle E[M]\rangle$）。此定义对应于操作语义（图 6）中的公理 R-Close、R-Com 和 R-Branch。称 P 是可约表达式，等价于说 P 按其中一条规则归约一步。

最后，如果进程 P 在结构上等价于形如 $(\nu x_1 y_1)\cdots(\nu x_n y_n)(P_1 \mid \cdots \mid P_m)$ 的进程，并且存在 i, j, k 使得 P_i 的主体是 x_k，P_j 的主体是 y_k，但 $(\nu x_k y_k)(P_i \mid P_j)$ 不是可约表达式，则该进程是运行时错误。

定理 3（不存在即时错误）。如果 $\vdash P$，那么 P 不是运行时错误。

6 算法化类型检查

图 12 中的类型规则并非算法化的。这里存在两个问题：猜测正确的上下文拆分（这是一个非确定性过程），以及猜测 λ-抽象和上下文值的约束变量的类型（以及层级）。对于前一个问题，我们重构类型规则，使其返回输入上下文中未使用的部分 [53]。对于后一个问题，我们使用显式类型化项语言：现在我们写 c^{τ} 来解决常量的多态性，写 $\lambda_m x{:}\ T.M$ 来引入约束变量 x 的类型，并使用（$x{:}\ \tau$）$^n.M$ 来引入上下文值的上下文类型和层级。

我们引入一种新的类型合成判断 $\Gamma_{\text{in}} \vdash M_{\text{in}} \Rightarrow T_{\text{out}} \mid \Delta_{\text{out}}$，其中上下文 Γ 和项 M 被视为输入，而类型 T 和上下文 Δ 被视为输出。相关规则见图 14。部分规则使用了上下文差 $\Gamma \div x = \Delta$，该操作确保 x 的绑定被移除

$$\text{A-Const}\quad \Gamma \vdash c^{\tau} \Rightarrow \tau \mid \Gamma$$

$$\text{A-UnVar}\quad \frac{\Gamma_1, x\colon (\overline{\tau} \vdash^{n} T) \vdash \sigma_1 \Rightarrow \tau_1 \mid \Gamma_2 \qquad \Gamma_2 \vdash \sigma_2 \Rightarrow \tau_2 \mid \Gamma_3 \ \ldots \ \Gamma_k \vdash \sigma_n \Rightarrow \tau_n \mid \Gamma_{k+1}}{\Gamma_1, x\colon (\overline{\tau} \vdash^{n} T)^{\omega} \vdash x[\sigma_1, \ldots, \sigma_k] \Rightarrow T \mid \Gamma_{k+1}}$$

$$\text{A-LinVar}\quad \frac{\Gamma_1 \vdash \sigma_1 \Rightarrow \tau_1 \mid \Gamma_2 \ \ldots \ \Gamma_k \vdash \sigma_k \Rightarrow \tau_k \mid \Gamma_{k+1}}{\Gamma_1, x\colon (\overline{\tau} \vdash^{n} T) \vdash x[\sigma_1, \ldots, \sigma_k] \Rightarrow T \mid \Gamma_{k+1}}$$

$$\text{A-LinFunI}\quad \frac{\Gamma, x\colon (\varepsilon \vdash^{0} T) \vdash M \Rightarrow U \mid \Delta}{\Gamma \vdash \lambda_1 x : T.M \Rightarrow T \to_1 U \mid \Delta \div x}$$

$$\text{A-UnFunI}\quad \frac{\Gamma, x\colon (\varepsilon \vdash^{0} T) \vdash M \Rightarrow U \mid \Delta \qquad \Delta \div x = \Gamma}{\Gamma \vdash \lambda_{\omega} x : T.M \Rightarrow T \to_{\omega} U \mid \Gamma}$$

$$\text{A-FunE}\quad \frac{\Gamma_1 \vdash M \Rightarrow T \to_m U \mid \Gamma_2 \qquad \Gamma_2 \vdash N \Rightarrow T \mid \Gamma_3}{\Gamma_1 \vdash M\, N \Rightarrow U \mid \Gamma_3}$$

$$\text{A-PairI}\quad \frac{\Gamma_1 \vdash M \Rightarrow T \mid \Gamma_2 \qquad \Gamma_2 \vdash N \Rightarrow U \mid \Gamma_3}{\Gamma_1 \vdash (M, N) \Rightarrow T \times U \mid \Gamma_3}$$

$$\text{A-PairE}\quad \frac{\Gamma_1 \vdash M \Rightarrow T \times U \mid \Gamma_2 \qquad \Gamma_2, x\colon (\varepsilon \vdash^{0} T)\ y\colon (\varepsilon \vdash^{0} U) \vdash N \Rightarrow V \mid \Gamma_3}{\Gamma_1 \vdash \text{令}\ (x, y) = M \text{于}\ \ N \Rightarrow V \mid \Gamma_3 \div x \div y}$$

$$\text{A-BoxI}\quad \frac{\Gamma \vdash \sigma \Rightarrow \tau \mid \Gamma}{\Gamma \vdash \text{装箱}\, \sigma \Rightarrow \Box\tau \mid \Gamma}$$

$$\text{A-BoxE}\quad \frac{\Gamma_1 \vdash M \Rightarrow \Box\tau \mid \Gamma_2 \qquad \Gamma_2,\ x\colon\ \tau \vdash N \Rightarrow T \mid \Gamma_3}{\Gamma_1 \vdash \mathsf{let}\ \text{装箱}\ \ x = M\, \mathsf{in}\ \ N \Rightarrow T \mid \Gamma_3 \div x}$$

$$\text{A-Match}\quad \frac{\Gamma_1 \vdash M \Rightarrow \&\{l\colon\ S_l\}^{l \in L} \mid \Gamma_2 \qquad \Gamma_2 \vdash N_l \Rightarrow S_l \to_1 T \mid \Gamma_3 \qquad (\forall l \in L)}{\Gamma_1 \vdash \text{匹配}\, M\, \mathsf{with}\ \{l\colon\ N_l\}^{l \in L} \Rightarrow T \mid \Gamma_3}$$

$$\text{A-Ctx}\quad \frac{\Gamma_1 \overset{n}{\backsim} \Gamma_3^{\geq n},\ \ \Gamma_4^{<n} \qquad \Gamma_2 < n \qquad \Gamma_2,\ \ \Gamma_3 \vdash M \Rightarrow T \mid \Delta}{\Gamma_1 \vdash\ (\Gamma_2^{n}.M)\ \Rightarrow\ (\overline{\tau} \vdash^{n} T)\ \mid \Delta \div \overline{x},\ \ \Gamma_4}$$

图14. 算法化类型检查 $\Gamma \vdash M \Rightarrow T \boxed{\mid \Delta}$

当类型界是非受限的时，它来自 Γ；而当类型是线性的时，它甚至不会出现。我们规定运算符 $\div$ 为左结合。

$$\frac{x\colon \tau \notin \Gamma}{\Gamma \div x = \Gamma} \qquad\qquad \frac{}{(\Gamma, x\colon \tau^{\omega}, \Delta) \div x = \Gamma, \Delta}$$

类型规则受到 Almeida 等人启发。 [2]在规则 A-Const、 A-UnFunI 和 A-BoxI中，输入和输出上下文相同，这意味着不会消耗任何线性变量。规则 A-LinFunI、 A-UnFunI、 A-PairE、 A-BoxE 和 A-Ctx 确保约束变量不会逃逸出其作用域。在规则 A-Ctx中，我们必须根据层级 $\Gamma \overset{n}{\backsim} \Delta$ 分割上下文（ n），使用

仅使用外部片段为子表达式确定类型，以模仿规则 T-Ctx。其余规则很容易理解。

如果我们将 $M^\bullet$ 视为带注释的项（如图 14中的项），就可以轻松地通过对 $M^\bullet$的结构进行归纳，定义擦除函数 erase($M^\bullet$)。由此可得以下结果。

定理 4（算法正确性）。

1. 若 $\Gamma \vdash M^\bullet \Rightarrow T \mid \Delta$且 Δ^ω，则 $\Gamma \vdash \mathsf{erase}(M^\bullet) : T$。2. 若 $\Gamma \vdash M : T$，则存在一个 $M^\bullet$使得 $M = \mathsf{erase}(M^\bullet)$，且 $\Gamma \vdash M^\bullet \Rightarrow T \mid \Delta$和 Δ^ω。

7 相关工作

我们关于分阶段元编程的方法基于上下文模态类型。这种方法的逻辑基础可追溯到模态逻辑 S4 [35,12], ，其中类型 $\Box T$ 表示一个封闭代码片段，该片段求值后得到类型为T的值。随后，通过引入上下文对盒类型进行推广，例如 2($\overline{T_1} \vdash T_2$)，使代码片段能够包含自由变量，方法是在盒类型中列出这些变量的类型。这些类型称为上下文模态类型 [28,29]。为避免与上下文类型混淆，本文将其称为盒类型。Mœbius [21] 进一步将上下文模态类型推广为可容纳具有不同层级的上下文，从而支持更多样的元编程。形式化上下文模态类型似乎有两种风格：一种是双上下文风格 [12,28,29]，其中盒值通过 let-box 语法和元变量进行解构；另一种是 Kripke 风格或 Fitch 风格 [10,12,27,50] ，其中盒值通过 unquote 项进行解构。按照这种分类，Mœbius [21] 可视为一种双上下文风格的演算，但进一步推广后允许多层上下文。正如引言中所述，我们选择 Mœbius 作为基础，因为它的语法比 Kripke/Fitch 风格表述更易处理。

内涵分析 [20,31], ，即对代码片段进行模式匹配的能力，是一项值得加入的功能，有助于改进我们的工作。尽管 Mœbius 的原始表述也支持内涵分析，但要扩展它以支持线性类型可能并非易事。支持多态是另一种可能的扩展。Mœbius 支持生成依赖于多态类型的代码，这在我们的类型系统中应该是可行的。Murase 等人提出了多态上下文 [27] ，它们可以通过上下文变量抽象上下文的多个部分。这种扩展对我们的类型系统也可能有用，但鉴于 Murase 等人定义的多态上下文 [27] 是为 Fitch 风格表述设计的，我们不确定能否扩展基于 Mœbius 的类型系统。

会话类型由 Honda 等人 [17,18,46] 于 1990 年代提出。我们采用的类型——输入/输出和内部/外部选择——源自这些早期工作。可关闭通道的类型来自 Caires 等人 [7,8]。早期关于会话类型的工作建立在 π-演算之上；

函数式编程中的会话类型后来由 Gay 和 Vas- concelos 提出 [15]。使用不同标识符表示同一通信信道两端的想法来自 Vasconcelos [51]。

据我们所知，只有 Georges 等人 [16], 、Sano 等人 [39], ˆ和 Angelo 等人 [3]提出了整合线性类型系统与多阶段编程的工作。本文则支持并发、递归类型、线性类型与无限制类型、会话类型，以及一种涵盖单一（函数式）编程语言的编程模型。

Sano 等人 [39], 提出的 Fuses 以不同于我们的方法，将上下文模态类型与会话类型相结合。其系统利用上下文模态类型，将函数层与具有会话类型的进程层分离，提供了一种替代方案：只在进程层处理线性性。尽管 Fuses 的灵感源自线性逻辑，但与我们的系统一样，它不保证无死锁性。此外，尽管进程层可以在消息中交换递归函数式代码，却无法在不借助函数层的情况下将结果拼接进代码。另一方面，我们的方案提供了更简洁流畅的编程体验，所有编程都在函数层进行。不过，Fuses 支持代码分析和模式匹配，这些主题我们决定留待未来研究。

8 结论与未来工作

我们展示了通过引入分阶段元编程来增强会话类型系统的优势，使其能够支持交换代码的协议。这种扩展要求将分阶段元编程集成到线性类型系统中（会话类型正是基于该系统定义的），这构成了主要的技术挑战。我们还展示了如何以实用的方式实现该系统，并给出了类型检查规则和正确性结果。我们还计划提升系统的表达能力，包括协议的定义方式，以及代码的创建和使用方式。可以考虑两个彼此独立的扩展方向。第一个是多态：既针对会话类型 [2]，也针对分阶段元编程 [21]。第二个是纳入上下文无关会话类型，以支持使用类型的顺序组合 [34,43] 进行编程。这些扩展将有助于构建更通用、可复用的代码和协议。

致谢 我们感谢 Chuta Sano 提出的富有洞见的意见。本研究部分得到 JSPS 邀请短期研究奖学金（赴日本研究）的支持。此外，本研究还得到 FCT 通过项目“通过高级会话类型实现可靠且富有表现力的并发系统”（项目编号：2023.13752.PEX，https://doi.org/10.54499/2023.13752.PEX）以及 LASIGE 研究单位（项目编号：UID/00408/2025，https://doi.org/10.54499/UID/00408/2025）的支持，并得到 LIACC 研究单位（10.54499/UIDB/00027/2020和10.54499/UIDP/00027/2020）的支持。

参考文献

1. Abadi, M.、Fiore, M.P.：《递归类型的句法考量》。载于：LICS。第 242–252 页。IEEE计算机学会（1996年）。https://doi.org/10 561324 2 1109 1996 . /LICS. . . 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》。Inf. Comput. 289（Part），104948（2022年）。https://doi.org/10 104948 3 1016 2022 . /J.IC. . . ˆAngelo, P.、Igarashi, A.、Vasconcelos, V.T.：《线性上下文元编程与会话类型》。载于：PLACES。EPTCS，第 401 卷，第 1–10 页（2024年）。https://doi.org/10 1 4 4204 401 . /EPTCS. . . Balat, V.：《Ocsigen：使用 Objective Caml 对网页交互进行类型检查》。载于：ML研讨会。第 84 1159889 5 94 2006 10 1145 1159876 – 页。ACM（）。https://doi.org/ . / . . Barendregt, H.P.：《λ演算——其语法与语义》，《逻辑与数学基础研究》，第 103 卷。North-Holland（1985年）6 09 7 2025 04 . Boinc。https://boinc.berkeley.edu/，访问日期：- -。Caires, L.、Pfenning, F.：《作为直觉主义线性命题的会话类型》。载于：CONCUR论文集。《计算机科学讲义》，第 6269 卷，第 222–236 页。施普林格（2010年）。https://doi.org/10 4 16 8 1007 978 3 642 15375 . / - - - - . Caires, L. 、 Pfenning, F. 、 Toninho, B.：《作为会话类型的线性逻辑命题》。Math. Struct. Comput. Sci. 26(3)，367–423（2016年）。 https://doi.org/10 0960129514000218 9 1017. /S . Chun, B.、Ihm, S.、Maniatis, P.、Naik, M.、Patti, A.：《Clonecloud：移动设备与云之间的弹性执行》。载于：Kirsch, C.M.、Heiser, G.（编）：《欧洲计算机系统会议》，第六届欧洲计算机系统会议论文集，EuroSys 2011，奥地利萨尔茨堡，2011年4月10日至13日。第 301–314 页。ACM（2011年）。https://doi.org/10.1145/1966445.1966473, https://doi.org/10 1966473 10 1145 1966445 . / . . Clouston, R.：《Fitch风格模态λ演算》。载于：FOSSACS。《计算机科学讲义》，第 10803 卷，第 258–275 页。施普林格（2018年）。https://doi.org/10 2 14 11 1007 978 3 319 89366 . / - - - - . Davies, R.：《一种用于绑定时间分析的时序逻辑方法》。J. ACM 64(1)，1:1– 1 3011069 12 45 2017 10 1145 3011069, 10 1145 48： （）。https://doi.org/ . / https://doi.org/ . / . Davies, R.、Pfenning, F.：《分阶段计算的模态分析》。J. ACM (3)，555 382785 13 604 2001 10 1145 382780 –（）。https://doi.org/ . / . . Dean, J.、Ghemawat, S.：《MapReduce：大型集群中的简化数据处理》。Commun. ACM 51(1)，107–113（2008年）。https://doi.org/10.1145/1327452.1327492，https://doi.org/10 1327492 14 1145 1327452 . / . . Gay, S.J. 、 Thiemann, P. 、 Vasconcelos, V.T.：《会话类型的对偶性：最终切分》。载于：PLACES。EPTCS，第 314 卷，第 23–33 页（2020）。https://doi.org/10 3 15 4204 314 . /EPTCS. . . Gay, S.J.、Vasconcelos, V.T.：《异步会话类型的线性类型理论》。J. Funct. Program. 20 0956796809990268 16 1 19 50 2010 10 1017（），–（）。https://doi.org/ . /S . Georges, A.L.、Murawska, A.、Otis, S.、Pientka, B.：《LINCX：具有一等上下文的线性逻辑框架》。载于：ESOP。《计算机科学讲义》，第 10201 卷，第 530–555 页。施普林格（2017年）。https://doi.org/10 1 20 17 1007 978 3 662 54434 . / - - - - . Honda, K.：《二元交互类型》。载于：CONCUR。LNCS，第 715 卷，第 509– 523 页。施普林格（1993年）。https://doi.org/10.1007/3-540-57208-2 35

18. Honda, K., Vasconcelos, V.T., Kubo, M.：面向基于结构化通信的编程的语言原语和类型纪律。载于：ESOP。《计算机科学讲义》，第 1381 卷，第 122 页 0053567 19 138 1998 10 1007 2025 02 22 – 。施普林格（ ）。https://doi.org/ 。 /BFb 。HTMX。https://htmx.org/，访问日期：- - 20. Hu, J.Z.S., Pientka, B.：分层模态类型理论——元编程与内涵分析相遇之处。载于：ESOP。《计算机科学讲义》，第 14576 卷，第 52–82 页。施普林格（2024年）。https://doi.org/10.1007/978-3-031-57262-3 3, https://doi.org/10.1007/978-3-031-57262-3 3 21. Jang, J., G´elineau, S., Monnier, S., Pientka, B.：Mœbius：使用上下文类型进行元编程：system F 能对自身进行模式匹配的阶段。《ACM 编程语言会议论文集》6 3498700 22 1 27 2022 10 1145（POPL），–（）。https://doi.org/ 。 / 。Kameyama, Y., Kiselyov, O., Shan, C.：转换阶段——使用定界控制进行阶段化。《函数式编程杂志》21(6)，617–662（2011年）。https://doi.org/10.1017/S0956796811000256, https://doi.org/10.1017/S0956796811000256 23. Kiselyov, O.：Metaocaml 理论与实现。CoRR abs/2309.08207（2023年）。https://doi.org/10.48550/ARXIV.2309.08207, https://doi.org/10.48550/arXiv.2309.08207 24. Meteor.js。https://www.meteor.com/，访问日期：2025-04-09 25. Milner, R.：函数即进程。《数学结构与计算机科学》2(2)，119–141（1992 0960129500001407 26 10 1017 ）。https://doi.org/ 。 /S 。Mitchell, J.C.：程序设计语言基础。计算基础丛书，麻省理工学院出版社（1996年）26. Murase, Y., Nishiwaki, Y., Igarashi, A.：具有多态上下文的上下文模态类型理论。载于：ESOP。《计算机科学讲义》，第 13990 卷，第 281–308 页。施普林格（2023年）。https://doi.org/10.1007/978-3-031-30044-8 11 28. Nanevski, A., Pfenning, F.：使用名称和必然性的分阶段计算。《函数式编程杂志》15(5)，893–939（2005年）。https://doi.org/10.1017/S095679680500568X 29. Nanevski, A., Pfenning, F., Pientka, B.：上下文模态类型理论。《ACM 计算逻辑汇刊》9(3)，23:1–23:49（2008年）。https://doi.org/10.1145/1352582.1352591，https://doi.org/10 1352591 30 1145 1352582 。 / 。Next.js。https://nextjs.org/，访问日期：2025-04-09 31. Parreaux, L., Voizard, A., Shaikhha, A., Koch, C.E.：统一分析式与静态类型准引号。《ACM 编程语言会议论文集》2(POPL)，13:1–13:33（2018 3158101 32 10 1145 3158101，10 1145 ）。https://doi.org/ 。 / https://doi.org/ 。 / 。Pfenning, F., Davies, R.：模态逻辑的判断式重构。《数学结构与计算机科学》11(4)，511–540（2001年）。https://doi.org/10.1017/S0960129501003322，https://doi.org/10 0960129501003322 33 1017。 /S 。Pierce, B.C.：类型与程序设计语言。麻省理工学院出版社（2002年）34. Po。cas, D., Costa, D., Mordido, A., Vasconcelos, V.T.：带上下文无关会话类型的系统 f^{μ} ømega。载于：ESOP。《计算机科学讲义》，第 13990 卷，第 392–420 页。施普林格（2023年）。https://doi.org/10.1007/978-3-031-30044-8 15 35. Prawitz, D.：自然演绎。Almquist & Wiksell，斯德哥尔摩，瑞典（1965年）36 09 37 2025 04 。Primegrid。https://www.primegrid.com/，访问日期：- - 。React 服务器组件。https://react.dev/reference/rsc/server-components，访问日期：2025-04-09

38. Rhiger, M.：具有分阶段词法作用域的分阶段计算。载于：Seidl, H.（编）《编程语言与系统——第 21 届欧洲编程语言研讨会，ESOP 2012》，作为欧洲软件理论与实践联合会议 ETAPS 2012 的一部分举行，爱沙尼亚塔林，2012年3月24日至4月1日。会议论文集。《计算机科学讲义》，第 7211 卷，第 559–578 页。施普林格（2012年）。 https://doi.org/10.1007/978-3-642-28869-2 28，https://doi.org/10 2 28 39 1007 978 3 642 28869 。 / - - - - 。Sano, C., Garg, D., Kavanagh, R., Pientka, B., Toninho, B.：将会话类型并发编程融入函数式编程。《ACM 编程语言会议论文集》9(ICFP)（2025年8月）。 https://doi.org/10.1145/3747519，https://doi.org/10 3747519 40 1145。 / 。Scala-loci。 https://scala-loci.github.io/，访问日期：2025 09 41 04- - 。Serrano, M., Prunet, V.： hopjs 概览。载于：ICPF。第 180–192 页。美国计算机协会（2016年）。 https://doi.org/10 2951916 42 1145 2951913 。 / 。Sheard, T., Jones, S.P.：Haskell 的模板元编程。载于：Haskell 研讨会。第 1 581691 43 16 2002 10 1145 581690 – 页。美国计算机协会（2002年）。https://doi.org/ 。 / 。Silva, G., Mordido, A., Vasconcelos, V.T.：上下文无关会话类型的子类型化。载于：CONCUR。LIPIcs，第 279 卷，第 11:1–11:19 页。 Schloss Dagstuhl - Leibniz-Zentrum für Informatik（）。https://doi.org/ 。 / LIPICS.CONCUR. 。 。Stucki, N., Brachthauser, J.I., Odersky, M.：使用生成式宏和分析式宏进行多阶段编程。载于：Tilevich, E., Roover, C.D.（编）《GPCE '21：概念与经验》，美国伊利诺伊州芝加哥，2021年10月17日至18日。第 110–122 页。美国计算机协会（2021年）。 https://doi.org/10.1145/3486609.3487203，https://doi.org/10 3487203 45 1145 3486609 。 / 。Taha, W., Nielsen, M.F.：环境分类器。载于：POPL。第 26–37 页。美国计算机协会（2003年）。https://doi.org/10.1145/604131.604134，https://doi.org/10 604134 46 1145 604131。 / 。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年）。 https://doi.org/10 7 118 47 1007 3 540 58184 。 / - - - 。Thiemann, P., Vasconcelos, V.T.： 上下文无关会话类型。载于：ICFP。第 462– 475 2951926 48 2016 10 1145 2951913 2025 04 页。美国计算机协会（2016年）。https://doi.org/ 。 / 。 。Unison。 https://www.unison-lang.org/，访问日期：- -09 49 2025 。Ur/Web。 https://github.com/urweb/urweb，访问日期：-04 09 50 - 。Valliappan, N., Ruch, F., Tom'e Corti nas, C.：Fitch 风格模态演算的规范化。《ACM 编程语言会议论文集》 6(ICFP)，772–798（2022年）。https://doi.org/10 3547649 51 1145。 / 。Vasconcelos, V.T.：会话类型基础。《信息与计算》217，52–70（2012年）。https://doi.org/10 002 52 1016 2012 05 。 /J.IC. 。 。Wadler, P.：作为会话的命题。《函数式编程杂志》24(2-3)， 384–418（2014年）53. Walker, D.：《类型与编程语言高级主题》，第"子结构类型系统"章，第 3–44 页。麻省理工学院出版社（2005年）54. White, T.：Hadoop——权威指南：互联网规模的存储与分析（第4版，修订与更新版）。O'Reilly（2015年）55. Xie, N., White, L., Nicole, O., Yallop, J.：Macocaml：阶段化可组合且可编译的宏。《ACM 编程语言会议论文集》7(ICFP)，604–648（2023年）。https://doi.org/10 3607851 56 1145 3607851，10 1145 。 / https://doi.org/ 。 / 。Zaharia, M., Chowdhury, M., Franklin, M.J., Shenker, S., Stoica, I.：Spark：带工作集的集群计算。载于：HotCloud'10。USENIX 协会（2010年）

一个多层级编程

上下文类型由 层级索引。我们一直将 流 ⊢ 单位类型 作为 流 ⊢1 单位类型的缩写，这是一种第 1 层上下文类型。可以区分两种代码片段：具有盒类型（例如 [流 ⊢ 流]）并存储在第 0 层变量中的代码片段，以及具有上下文类型（例如 流 ⊢ 流）并存储在第 1 层或更高层级变量中的代码片段。第 2 节的开头示例说明，第 1 层上下文值可以被复制和丢弃。值得注意的是，具有盒类型的第 0 层资源同样可以是非受限的。下面的 sendTwice 变体展示了盒类型的非受限性质，其中具有盒类型 [流 ⊢1 流] 的第 0 层变量 z 被使用了两次。

```
sendTwice': 流 → 单位类型sendTwice' = let z = 装箱 (y. 发送
5 (select 更多 y)) let 装箱 u = z in let 装箱 v = z in λx. close (s
elect 完成 u [v [x ]])
```

在我们的类型系统中，代码片段是非受限资源，因为装箱的设计初衷是使其独立于线性资源。层级-n 框会抽象所有低于层级 n的变量，这意味着层级 0 的变量总是会被抽象。我们的类型系统只允许层级 0 的变量是线性的。因此，框不会依赖线性资源，即使代码片段会计算出线性资源，复制或丢弃这些代码片段也是安全的。

到目前为止，我们的所有示例都展示了最高为层级 1 的上下文类型。下面是一个包含层级 2 类型的示例，其中，发送任意数量整数值的代码被夹在发送整数 100 的代码之间。

```
sandwich: [流,(流 ⊢ 1流) ⊢ 2单位类型 ]sandwich = 装箱 (x,u.

    let x1 = 发送 100 (select 更多 x) in let x2 = u [
    x1 ] in let x3 = 发送 100 (select 更多 x2) incl
    ose x3)
```

通信模板 sandwich 是以两个参数为参数化的代码。第一个参数是类型为 流 的信道端点，它是一个 0 层类型。第二个参数是以一个信道端点为参数化的代码，其计算结果是另一个信道端点，两者的类型都是 流。因此，第二个参数属于 1 层。由此，表达式sandwich 的类型是一个 2 层盒类型。

最后，我们将 sandwich 的代码拼接到函数 sandwichMe 中，使信道端点和以盒形式提供的值都能拼接到模板代码中，从而填入要发送的值。代码 sandwichMe 装箱(z. 发送 5 (选择 更多 (发送 5 (选择 更多 z)))) 会在关闭信道之前发送四个整数值（100、5、5 和100）。

```
sandwichMe: [流 ⊢ 1 流] → [流 ⊢ 1 单位类型 ]sandwichMe y =
let 装箱 u = y in let 装箱 v = sandwich in 装箱 (x.v [x,u ])
```

B 主要结果的证明

B.1 基本结果

证明（引理 *1*）。根据上下文类型 n 的层级 τ进行归纳法。

情形 $n = 0$。我们有 $\tau = (\varepsilon \vdash^0 T)$，因为假设 τ 是良构的（即满足 $\tau\mathsf{ctype}^0$），且 $\eta(x, \varepsilon \vdash^0 T) = \varepsilon.x[\epsilon]$。根据规则 T-Var，$x\colon \tau \vdash x[\epsilon]\colon T$。根据规则 T-Ctx，$x\colon \tau \vdash \varepsilon.x[\epsilon]\colon \tau$

情形 $n = k + 1$。我们有 $\tau = (\overline{\tau} \vdash^{k+1} T)$，并且 $\eta(x, \tau) = \overline{y}.x[\eta(y, \overline{\tau})]$。根据归纳法，我们有 $y\colon \tau \vdash \eta(y, \tau)\colon \tau$。根据 T-Var，我们有 $x\colon \tau, y\colon \overline{\tau \vdash x}[\eta(y, \tau)]\colon T$。由于 τ 是良构的，我们有 $\overline{\tau} < n$，因此可以应用规则 T-Ctx 得到 $x\colon \tau \vdash \overline{y}.x[\eta(y, \tau)]\colon \tau$。

令 $\mathcal{U}(\Gamma)$ 表示 Γ 的无限制部分，即恰好包含 Γ 中满足 $x\colon \tau^\omega$ 的那些绑定的类型上下文。我们以相应的方式定义 $\mathcal{L}(\Gamma)$。

引理 2（上下文拆分 [51]的基本性质）。令 $\Gamma = \Gamma_1 \circ \Gamma_2$。

1. $\mathcal{U}(\Gamma) = \mathcal{U}(\Gamma_1) = \mathcal{U}(\Gamma_2)$。2. 若 $x\colon \tau^1 \in \Gamma$，则要么 $x\colon \tau^1 \in \Gamma_1$ 且 $x\colon \tau' \notin \Gamma_2$；要么 $x\colon \tau^1 \in \Gamma_2$ 且$x\colon \tau' \notin \Gamma_1$，对任意 τ'。3. $\Gamma = \Gamma_2 \circ \Gamma_2$。4. 若 $\Gamma_1 = \Delta_1 \circ \Delta_2$，则 $\Delta = \Delta_2 \circ \Gamma_2$ 且 $\Gamma = \Delta_1 \circ \Delta$。

B.2 保持性

引理 3（无限制弱化）。

$$\frac{\Gamma \vdash M : T}{\Gamma, x\colon \tau^\omega \vdash M : T}\ \textsc{weak}$$

证明。对 $\Gamma \vdash M\colon T$ 进行规则归纳。

引理 4（无限制强化）。如果 $\Gamma, x\colon \tau \vdash M\colon T$，且 $x \notin$ 在 M 中自由出现，则 $\tau\,\omega$，且 $\Gamma \vdash M\colon T$。

证明。对 $\Gamma, x\colon \tau \vdash M\colon T$ 进行规则归纳。

引理 5（上下文类型判断）。 $\Gamma \vdash E[M]\colon T$ 当且仅当 $\Gamma = \Gamma_1 \circ \Gamma_2$ 且 $\Gamma_1 \vdash M\colon U$，并且 $\Gamma_2, x\colon U \vdash E[x]\colon T$。

证明。对 E 进行结构归纳。

引理 6（替换）。设 $\Gamma \vdash \sigma\colon \tau$，且 $\Gamma \geq n$，其中 n 是 τ 的层级，并假设 $\tau\,1$ 或 $\Gamma\,\omega$。

1. 若 $\Delta, x\colon \tau \vdash M\colon T$和 $\Gamma \circ \Delta$定义良好，则 $\Gamma \circ \Delta \vdash \{\sigma/x\}M\colon T$。2. 若 Δ, $x\colon \tau \vdash \rho\colon \tau'$和 $\Gamma \circ \Delta$定义良好，则 $\Gamma \circ \Delta \vdash \{\sigma/x\}\rho\colon \tau'$。

证明。我们通过对 τ的层级 n以及 $\Delta, x\colon \tau \vdash M\colon T$或 $\Delta, x\colon \tau \vdash \rho\colon \tau'$ 的推导大小进行字典序归纳，并针对所用的最后一个类型规则进行分类讨论，从而同时证明这两个命题。在此证明中，我们用$\Gamma \setminus x$ 表示从 Γ中移除 x 的绑定后得到的类型环境（如果有的话）。

我们概述最有趣的情况。

情况规则 T-Var：我们有 $M = y[\overline{\rho}]$和 $\Delta, x\colon \tau = \Gamma_1 \circ \cdots \circ \Gamma_m \circ (\Delta_0^\omega, y\colon (\overline{\tau} \vdash^{n'} T))$，以及 $\overline{\Gamma} \vdash \overline{\rho}\colon \overline{\tau}$，其中 $y, \overline{\rho}, \Delta_0, \overline{\Gamma}, \overline{\tau}, n'$。有两种可能的子情况需要考虑：

– 子情况 $x \neq y$：我们将证明 $\Gamma \circ \Delta \vdash y[\{\sigma/x\}\overline{\rho}] : T$ 因为 $\{\sigma/x\}(y[\overline{\rho}]) = y[\{\sigma/x\}\overline{\rho}]$。

- 子子情况 τ 1：必定存在 i，使得 $\Gamma_i = \Gamma_i', x\colon \tau$ 对于某个 Γ_i'，且对于 $j \neq i$, $x \notin \mathsf{dom}(\Gamma_j)$。容易证明$\Gamma \circ \Gamma_j'$定义良好，因为我们假设 $\Gamma \circ \Delta$定义良好。根据归纳假设，$\Gamma \circ \Gamma_i' \vdash \{\sigma/x\}\rho_i\colon \tau_i$。对于 $j \neq i$, $\{\sigma/x\}\rho_j = \rho_j$，由于 $x \notin \mathsf{dom}(\Gamma_j)$，因此 x 不是 ρ_j的自由变量。根据 T-Var，我们有 $(\Gamma_1 \circ \cdots \circ (\Gamma \circ \Gamma_i') \circ \cdots \circ \Gamma_m) \circ (\Delta_0^\omega, y\colon (\overline{\tau} \vdash^{n'} T)) \vdash y[\{\sigma/x\}\overline{\rho}]\colon T$。容易证明 $(\Gamma_1 \circ \cdots \circ (\Gamma \circ \Gamma_j') \circ \cdots \circ \Gamma_m) \circ \Gamma \circ (\Delta_0^\omega, y\colon (\overline{\tau} \vdash^{n'} T)) = \Gamma \circ (\Delta, x\colon \tau) \setminus x = \Delta$。

- 子子情况 $\tau\omega$。必然存在这样的情况：对于任意 $1 \leq i \leq m$、$\Gamma_i = \Gamma_i'$、$x\colon \tau$存在某个 Γ_i'。根据假设，我们还有 $\Gamma\omega$。与上述情况类似，根据归纳假设，对于任意 i、$\Gamma \circ \Gamma_i' \vdash \{\sigma/x\}\rho_i\colon \tau_i$。根据 T-Var，我们有 $((\Gamma \circ \Gamma_1') \circ \cdots \circ (\Gamma_m \circ \Gamma_m')) \circ (\Delta_0^\omega, y\colon (\overline{\tau} \vdash^{n'} T)) \vdash y[\{\sigma/x\}\overline{\rho}]\colon T$。容易证明 $((\Gamma \circ \Gamma_1') \Gamma \circ \circ \overline{\tau} \vdash^{n'} T = \Gamma \circ \Delta\ x\ \tau (\Gamma \circ \Gamma_m')) (\Delta_0^\omega, : (\overline{\tau} \vdash^{n'} T)) (\Gamma_1' \circ \cdots \circ \Gamma_m') (\Delta_0^\omega, y\colon ( )) (, :) \setminus x = \Gamma \circ \Delta$。

– 子情况 $x = y$：对于某些 $\overline{z}$、N 和 $\{\sigma/x\}M = \{\{\sigma/x\}\overline{\rho}/\overline{z}\}N$，我们有 $\tau = \overline{\tau} \vdash^{n'} T$ 和 $\sigma = \overline{z}.N$。我们将证明 $\Gamma \circ \Delta \vdash \{\{\sigma/x\}\overline{\rho}/\overline{z}\}N\colon T$。根据 T-Ctx，我们有 Γ、$z\colon \tau \vdash N\colon T$。注意，$\Gamma^{\geq n}$ 且 $(z\colon \tau)^{<n}$。~~我们~~进一步分类讨论 $n = 0$ 是否成立，以及~~τ~~的重数。

- 子子情况 $n = 0$。在这种情况下，$\overline{z}$ 必须为空。只需证明 $\Gamma \circ \Delta \vdash N\colon T$，而这可由弱化得出。（容易证明 $\Delta\,\omega$，因为 $\overline{\Gamma}$ 也为空且 $\Delta = \Delta_0$。）

- 子子情况 $n > 0$ 和 τ 1。与 $x \neq y$ 和 τ 1的情况类似，存在 i 使得 $\Gamma_i = \Gamma_i'$、$x\colon \tau$ 且 $\Gamma \circ \Gamma_i' \vdash \{\sigma/x\}\rho_i\colon \tau_i$，并且对于 $j \neq i$，有 $\Gamma_j' \vdash \{\sigma/x\}\rho_j\colon \tau_j$。根据归纳假设（注意，$\tau_j$ 的层级严格低于 n），$\Gamma_1 \circ \cdots \circ (\Gamma \circ \Gamma_i') \circ \cdots \circ \Gamma_m \vdash \{\{\sigma/x\}\overline{\rho}/\overline{z}\}N\colon T$。容易证明 $\Delta = \Gamma_1 \circ \cdots \circ \Gamma_i' \circ \cdots \circ \Gamma_m \circ \Delta_0^\omega$。弱化完成此情况的证明。
- 子子情况 $n > 0$ 和 $\tau\,\omega$ 相似。

情形规则 T-UnFunI。设 $M = \lambda y.M_0$和 $T = T' \to_\omega U'$。于是有 $\Gamma \vdash \sigma\colon \tau$和 $\Delta, x\colon \tau \vdash \lambda y.M_0\colon T' \to_\omega U'$。我们要证明$\Gamma \circ \Delta \vdash \{\sigma/x\}(\lambda y.M_0)\colon T' \to_\omega U'$。

我们有 $\{\sigma/x\}(\lambda y.M_0) = \lambda y.\{\sigma/x\}M_0$。根据 T-UnFunI，我们有 $\Delta\,\omega$

并且 $\tau\,\omega$且 $\Delta, x\colon \tau, y\colon (\varepsilon \vdash^0 T') \vdash M_0\colon U'$。不失一般性，我们可以假设 $y \notin \mathrm{dom}(\Gamma)$。现在，根据 y 的类型的重数，我们分为两种子情况。

– 子情况（$\varepsilon \vdash^0 T'$）1： $\Gamma \circ (\Delta, y\colon (\varepsilon \vdash^0 T'))$ 定义良好。根据归纳假设，$\Gamma \circ (\Delta, y\colon (\varepsilon \vdash^0 T')) \vdash \{\sigma/x\}M_0\colon U'$，这等价于（$\Gamma \circ \Delta$） $y\colon (\varepsilon \vdash^0 T') \vdash \{\sigma/x\}M_0\colon U'$。– 子情况（$\varepsilon \vdash^0 T'$） $\omega\colon (\Gamma, y\colon (\varepsilon \vdash^0 T'))$ 。（$\Delta, y\colon (\varepsilon \vdash^0 T')$）定义良好，且等于（$\Gamma \circ \Delta$） $y\colon (\varepsilon \vdash^0 T')$。根据弱化和归纳假设，（$\Gamma \circ \Delta$） $y\colon (\varepsilon \vdash^0 T') \vdash \{\sigma/x\}M_0\colon U'$。

由于 $\tau\,\omega$，根据假设有 $\Gamma\,\omega$。因此，$(\Gamma \circ \Delta)\ \omega$。由 T-UnFunI， $\Gamma \circ \Delta \vdash \{\sigma/x\}(\lambda y.M_0)\colon T' \to_\omega U'$。
情形规则 T-BoxI。我们有 Δ， $x\colon \tau = \Delta_1$， Δ_2 以及 $\Delta_1\,\omega$和 $\Delta_2\,\omega$（因此 $\tau\,\omega$和 $\Gamma\,\omega$），以及 $M =$装箱 ρ和 $T = \Box\ (\overline{\tau'} \vdash^n U)$）以及 $\Delta_1 \vdash \rho\colon (\overline{\tau'} \vdash^n U)$。由于根据假设 $\Gamma \circ \Delta$ 定义良好，并且 $\Gamma\,\omega$ 和 $\Delta\,\omega$，因此必有 $\Gamma = \Delta$和 $\Gamma \circ \Delta = \Gamma$。我们希望证明 $\Delta \vdash \{\sigma/x\}$装箱 $\rho\colon \Box(\overline{\tau'} \vdash^k U)$。

由弱化， $\Delta, x\colon \tau \vdash \rho\colon (\overline{\tau'} \vdash^n U)$。由归纳假设， $\Delta \vdash \{\sigma/x\}\rho\colon (\overline{\tau'} \vdash^n U)$。规则 T-BoxI完成此情形。
情形：规则 T-BoxE。我们有 $M =$let box $y = M'$in N'，以及 $\Delta, x\colon \tau = \Delta_1 \circ \Delta_2$ 和 $\Delta_1 \vdash M'\colon \Box\tau'$，还有 $\Delta_2, y\colon \tau' \vdash N'\colon T$，其中 $y, M', N', \Delta_1, \Delta_2$ 为某些值。我们要证明 $\Gamma \circ \Delta \vdash \{\sigma/x\}$(let box $y = M'$in N') $: T$。我们有 $\{\sigma/x\}$(let box $y =M'$in N') $=$let box $y = \{\sigma/x\}M'$in $\{\sigma/x\}N'$。需要考虑两种可能的子情形：

– 子情况 $\tau\,\omega$：根据 T-BoxE、 $\Delta_i = \Delta_i'$、 $x\colon$ 对于某些 Δ_1'、 Δ_2'和 Δ_1'、 x，有 τ（对于 $i = 1, 2$）： $\tau \vdash M'\colon \Box\tau'$ 和 Δ_2'、 $x\colon \tau^n$、 $y\colon \tau' \vdash N'\colon T$。根据归纳假设， $\Gamma \circ \Delta_1' \vdash \{\sigma/x\}M'\colon \Box\tau'$ 和 $\Gamma \circ (\Delta_2'$、 $y\colon \tau')$ $\vdash \{\sigma/x\}N'\colon T$。由于 $\tau\,\omega$，我们有 $\Gamma\,\omega$。根据 T-BoxE 以及引理 2， $\Gamma \circ (\Delta_1' \circ \Delta_2')$ $\vdash$ 令 box $y =\{\sigma/x\}M'$，则 $\{\sigma/x\}N'\colon T$。显然 $\Delta = \Delta_1' \circ \Delta_2'$。– 子情况 τ1，且 x 在 M' 中自由出现：根据 T-BoxE、 $\Delta_1 = \Delta_1'$、 $x\colon$ 对于某些 Δ_1' 和 Δ_1'、 $x\colon \tau \vdash M'\colon \Box\tau'$ 以及 Δ_2、 $y\colon \tau' \vdash N'\colon T$。根据归纳假设，$\Gamma \circ \Delta_1' \vdash \{\sigma/x\}M'\colon \Box\tau'$。由于 Δ_2、 $y\colon \tau' \vdash N'\colon T$ 以及 $x\colon \tau \notin \Delta_2$、 $y\colon \tau'$，因此 x 不在 N' 中自由出现，从而 $\{\sigma/x\}N' = N'$。因此，Δ_2、 $y\colon \tau' \vdash \{\sigma/x\}N'\colon T$。根据 T-BoxE 以及引理 2， $\Gamma \circ (\Delta_1' \circ \Delta_2) \vdash$ 令 box $y = \{\sigma/x\}M'$，则 $\{\sigma/x\}N'\colon T$。显然 $\Delta = \Delta_1' \circ \Delta_2$。如果 x 在 N' 中自由出现，证明类似。

情形规则 T-Ctx：我们有 $\rho = \overline{y}.M$和 $\tau' = \overline{\tau'} \vdash^k T'$，其中某些 $\overline{y}$、 M 和 $(\Delta, x\colon \tau) \geq k$ 以及 $\Delta, x\colon \tau, y\colon \overline{\tau' \vdash M\colon T'}$。我们要证明 $\Gamma \circ \Delta \vdash \{\sigma/x\}\overline{y}.M\colon \overline{\tau'} \vdash^k T'$。

令 $\overline{z\colon \tau''}$be $y\colon \tau'$的一个子序列，$\overline{\text{其中包含所有 } \tau_i''}\omega$。于是，$(\Gamma, \overline{z\colon \tau''}) \circ (\Delta, \overline{y\colon \tau'}) = (\Gamma \circ \Delta)\ \overline{y\colon \tau'}$。根据弱化，我们有 $\Gamma, z\colon \tau'' \vdash \overline{\sigma\colon \tau}$。根据归纳假设，$(\Gamma, z\colon \tau'') \circ (\Delta, y\colon \tau') \vdash \{\sigma/x\}M\colon T'$，这等价于 $(\Gamma \circ \Delta)\ y\colon \tau' \vdash \{\sigma/x\}M\colon T'$。由于 $(\Delta, x\colon \tau) \geq k$，特别是 $\tau \geq k$，因此我们有 $\Gamma \geq k$。由T-Ctx, $\Gamma \circ \Delta \vdash \overline{y}.\{\sigma/x\}M\colon \overline{\tau'} \vdash^k T'$。

证明（定理 *1*的证明）。对 $M \to N$进行规则归纳。关于 R-Beta 和 R-LetBox 的情形已在前文讨论过。关于 R-Split 的情形与 R-Beta类似，利用上下文拆分具有交换性和结合性这一事实（引理 2）。关于 R-Fix 的情形显而易见。关于 R-Ctx 的情形由引理 5可得。

引理 7（无限制值在无限制类型环境下具有类型）。如果 $\Gamma \vdash v : T$ 且 $T\,\omega$，那么 $\Gamma\,\omega$。

证明。通过对 $\Gamma \vdash v : T$ 所使用的最后一个类型规则进行分类讨论。

引理 8（值替换）。如果 $\Gamma \vdash v : U$ 且 $\Delta,\ x : U \vdash M : T$ 且 $\Gamma \circ \Delta$ 已定义，那么 $\Gamma \circ \Delta \vdash \{v/x\}M : T$。

证明。 根据引理 6 和7。

引理 9（结构同余的可靠性）。设 $P \equiv Q$。当且仅当 $\Gamma \vdash Q$ 时，$\Gamma \vdash P$。

证明。通过对 $\Gamma \vdash P$ 以及 $\Gamma \vdash Q$ 进行规则归纳。

B.3 函数式语言的进展

证明（引理 *10* 的证明）。证明对 $\Gamma^S \vdash v$ 的推导中所使用的最后一条规则进行分类讨论：T。

1. 唯一适用的规则是 T-Const。于是，我们有 $v = c$，并且 Unit =typeof(c)。根据图 10，满足 typeof(c) =Unit 的唯一常量 c 是 unit。因此，v =unit。
2. 唯一适用的规则是 T-Var。于是我们有 $v = x[\overline{\sigma}]$，并且 $(\Delta^{\omega}, \overline{\Gamma} \circ x : (\overline{\tau} \vdash^n T))^S \vdash x[\overline{\sigma}] : T$。由于上下文限制，我们有$\overline{\tau} \vdash^n T = \varepsilon \vdash^0 T$，因此 $\overline{\sigma} = \varepsilon$。所以，$v = x$，即 $x[\varepsilon]$ 的缩写形式。 3. 适用的规则只有：– T-Const。因此，$v =$ 可以是 close、wait、send、send v、receive、select l、new、fork 或 fix。– T-LinFunI 或 T-UnFunI。因此，$v = \lambda x.M$。 4. 唯一适用的规则是 T-PairI。因此，$v = (u, w)$。 5. 唯一适用的规则是 T-BoxI。因此，$v =$ box σ。

证明（定理 *2*的证明）。根据假设进行规则归纳，使用规范形式（引理 10）。规则 T-Const、 T-Var、 T-LinFunI、 T-UnFunI、T-PairI 和 T-BoxI 的情形是平凡的。

情形规则 T-FunE。那么 $M = M_1\,M_2$和 ${\Gamma_1}^S \circ {\Gamma_2}^S \vdash M_1\,M_2 : T$，且 $\Gamma^S = {\Gamma_1}^S \circ {\Gamma_2}^S$。根据 T-FunE，${\Gamma_1}^S \vdash M_1 : U \to_m T$且 ${\Gamma_2}^S \vdash M_2 : U$。根据归纳假设，对于 M_1 和 M_2，1.、2. 或 3. 之一成立。有以下几种情形：

– M_1 和 M_2 都是值。那么，根据引理 10， M_1 是 $\lambda x.M_1'$、关闭、等待、发送、发送 v、接收、选择 l、新建、分叉或 fix。若 $M_1 = \lambda x.M_1'$，则根据 R-Beta，$M \to N$，其中某个 N。若 M_1是关闭、等待、发送 v、接收或选择 l，则根据 T-Const， U 分别是 Close、Wait、$!T.S$、$?T.S$ 或 $\oplus\{l : S_l\}^{l\in L}$。根据引理 10，$M_2 = x$，且 3. 成立。若 M_1 是发送，则 M 是一个值。若M_1 是新建，则 3. 成立。若 M_1 是分叉，则 3. 成立。若 M_1 是 fix，则根据R-Fix， $M \to N$，其中某个 N。– M_1 是一个值，且 $M_2 \to N_2$，其中某个 N_2。那么，根据 R-Ctx，$M \to N$，其中某个 N。– M_1 是一个值，且 M_2 满足 3. 那么 3. 成立。– $M_1 \to N_1$，其中某个 N_1。那么，根据 R-Ctx， $M \to N$，其中某个 N。– M_1 满足 3. 那么 3. 成立。

情形规则 T-PairE。于是 $M =$let $(x, y) = M_1$ in M_2 and $\Gamma_1^S \circ \Gamma_2^S \vdash M :\ T$。根据 T-PairE，我们有 $\Gamma_1^S \vdash M_1 :\ U_1 \times U_2$ and $\Delta^S, x :\ (\varepsilon \vdash^0 S_1)\ y :\ (\varepsilon \vdash^0 S_2) \vdash M_2 :\ T$。根据归纳假设，对于 M_1 and M_2，1.、2. 或 3. 中的某一项成立。分以下几种情形：

– M_1 是一个值。于是，根据引理 10， $M_1 = (u, w)$ 和 $M \to N$ 根据 R-Split。– $M_1 \to N_1$，对于某个 N_1。于是，根据 R-Ctx， $M \to N$，对于某个 N。– M_1 满足 3. 因此 3. 成立。

规则 T-BoxE 情形。证明与前一种情形类似。规则 T-Match 情形。证明与前一种情形类似。

B.4 进程语言中不存在运行时错误

引理 10（典范形式）。设 $\Gamma^S \vdash v :\ T$。

1. 若 $T =$Unit，则 $v =$ 为 unit。2. 若 $T =$ Close、Wait、$!T.S$、$?T.S$、$\oplus\{l :\ S_l\}^{l\in L}$ 或 $\&\{l :\ S_l\}^{l\in L}$，则 $v = x[\varepsilon]$。3. 若 $T = U \to_m V$，则 v 是 $\lambda x.M$、关闭、等待、发送、发送 v、接收、选择 l、新建、分叉或 fix。4. 若 $T = U \times V$，则 $v = (u, w)$。5. 若 $T = \Box\tau$，则 $v =$ 装箱 σ。

证明。对 $\Gamma^S \vdash v :\ T$ 的推导中使用的最后一条规则进行分类讨论。

证明（定理的证明 *2*）。根据假设进行规则归纳法，并使用规范形式（引理 10）。

证明（定理 3 的证明）。取 $P \equiv (\nu x_1 y_1) \cdots (\nu x_n y_n)(P_1 \mid \cdots \mid P_m)$。结构同余的可靠性（引理 9）保证后一个进程在空上下文下也可类型化。现在取 i, j, k，使得 P_i的主体为 x_k， P_j 的主体为 y_k。我们知道存在一个 Γ^S，使得 $\Gamma^S \vdash P_i \mid P_j$，因为 Γ 中的所有条目都是由规则 P-Res引入的。同一规则还确保 Γ^S 包含形如 $x_k :\ R, y_k :\ S$ 的条目，其中 $R \perp S$。对偶保证 $(\nu x_k y_k)(P_i \mid P_j)$ 是一个可约表达式，因此 P 不是运行时错误。

B.5 算法类型检查的证明（第 6节）

引理 11（上下文差的性质）。设 $\Gamma \div x = \Delta$。

1. $\Delta = \Gamma \setminus \{x\colon\ \tau\}$2. $\mathcal{L}(\Gamma) = \mathcal{L}(\Delta)$ 3. 若 $x\colon \tau \in \Gamma$，则 τ^ω和 $x\colon\ \tau' \notin \Delta$，其中 τ'。

证明。通过分类讨论即可得证。

引理 12（算法单调性）。若 $\Gamma \vdash M^\bullet \Rightarrow T \mid \Delta$，则 $\mathcal{U}(\Gamma) = \mathcal{U}(\Delta)$，且 $\mathcal{L}(\Delta) \subseteq \mathcal{L}(\Gamma)$。

证明。对 $\Gamma \vdash M^\bullet \Rightarrow T \mid \Delta$ 的规则进行归纳即可得证，使用引理 11。

引理 13（算法线性强化）。假设 τ 1。若 $\Gamma, x\colon\ \tau \vdash M^\bullet \Rightarrow T \mid \Delta, x\colon\ \tau$，则 $\Gamma \vdash M^\bullet \Rightarrow T \mid \Delta$。

证明。 根据 $\Gamma \vdash M^\bullet \Rightarrow T \mid \Delta$ 的规则进行归纳法，并使用引理 11 和引理 12。

引理 14（算法弱化）。若 $\Gamma \vdash M^\bullet \Rightarrow T \mid \Delta$，则 $\Gamma, x\colon\ \tau \vdash M^\bullet \Rightarrow T \mid \Delta, x\colon\ \tau$。

证明。 根据 $\Gamma \vdash M^\bullet \Rightarrow T \mid \Delta$ 的规则进行归纳法，使用引理 11。

证明（定理 4 的证明，条目 1）。根据 $\Gamma \vdash M^\bullet \Rightarrow T \mid \Delta$ 的规则进行归纳法，使用引理 11、引理 12 和引理 13。规则 A-Const 的证明显然。规则 A-UnVar、A-LinVar、 A-FunE、A-PairI、 A-PairE 和 A-Match 的证明采用与规则 A-BoxE相同的策略。下面详述以下情形：

– 规则 A-LinFunI。假设 $(\Delta \div x)^\omega$ 和 $\Gamma \vdash \lambda_1 x\colon\ T.M^\bullet \Rightarrow T \to_1 U \mid \Delta \div x$。根据引理 11，我们有 $\mathcal{L}(\Delta) = \mathcal{L}(\Delta \div x)$，因此 Δ^ω。根据规则 A-LinFunI,$\Gamma, x\colon (\varepsilon \vdash^0 T) \vdash M^\bullet \Rightarrow U \mid \Delta$。根据归纳假设，我们有$\Gamma, x\colon (\varepsilon \vdash^0 T) \vdash \mathsf{erase}(M^\bullet)\colon U$。根据规则 T-LinFunI, $\Gamma \vdash \lambda x.\mathsf{erase}(M^\bullet)\colon T \to_1 U$。根据擦除， $\Gamma \vdash \mathsf{erase}(\lambda_1 x\colon\ T.M^\bullet)\colon T \to_1 U$。– 规则 A-UnFunI。假设 Γ^ω和 $\Gamma \vdash \lambda_\omega x\colon\ T.M^\bullet \Rightarrow T \to_\omega U \mid \Gamma$。根据规则A-UnFunI, $\Gamma, x\colon (\varepsilon \vdash^0 T) \vdash M^\bullet \Rightarrow U \mid \Delta$和 $\Delta \div x = \Gamma$。根据引理 11，我们有 $\mathcal{L}(\Delta) = \mathcal{L}(\Gamma)$，因此 Δ^ω。根据归纳假设，我们有$\Gamma^\omega, x\colon (\varepsilon \vdash^0 T) \vdash \mathsf{erase}(M^\bullet)\colon U$。根据规则 T-UnFunI, $\Gamma^\omega \vdash \lambda x.\mathsf{erase}(M^\bullet)\colon T \to_\omega U$。根据擦除， $\Gamma \vdash \mathsf{erase}(\lambda_\omega x\colon\ T.M^\bullet)\colon T \to_\omega U$。– 规则 A-BoxI。假设 Γ^ω和 $\Gamma \vdash \mathsf{box}\,(x\colon\ \tau.M^\bullet) \Rightarrow \square(\overline{\tau} \vdash^n T) \mid \Gamma$。根据A-BoxI, $\Gamma \vdash x\colon\ \tau.M^\bullet \Rightarrow (\overline{\tau} \vdash^n T) \mid \Gamma$。唯一适用的规则是 A-Ctx，因此 $\Gamma_1, x\colon\ \tau \vdash M^\bullet \Rightarrow \overline{T} \mid \Delta$，其中 $\Gamma \overset{n}{\rightsquigarrow} \Gamma_1^{\geq n}, \Gamma_2^{<n}$和 $\Gamma = \Delta \div \overline{x}, \Gamma_2$。由此自然可得 $(\Delta \div \overline{x})^\omega$ 和 $(\overline{\Gamma_2})^\omega$。此外，根据引理 11，我们有 $\mathcal{L}(\Delta) = \mathcal{L}(\Delta \div x)$，因此 Δ^ω。根据归纳假设，$\Gamma_1^{\geq n}, x\colon\ \tau^{<n} \vdash \mathsf{erase}(M^\bullet)\colon T$。根据规则 T-Ctx, $\Gamma_1^{\geq n} \vdash \overline{x}.\mathsf{erase}(M^\bullet)\colon (\overline{\tau} \vdash^n T)$。根据规则 T-BoxI, $\Gamma_1, \Gamma_2 \vdash \mathsf{box}\,(\overline{x}.\mathsf{erase}(M^\bullet))\colon \square(\overline{\tau} \vdash^n T)$。根据擦除，$\Gamma_1, \Gamma_2 \vdash \mathsf{erase}(\mathsf{box}\,(x\colon\ \tau.M^\bullet))\colon \square(\overline{\tau} \vdash^n T)$。

– 规则 A-BoxE。假设 $(\Gamma_3 \div x)^\omega$和 $\Gamma_1 \vdash$let box $x = M^\bullet$in $N^\bullet \Rightarrow T \mid \Gamma_3 \div x$。根据引理 11，我们有 $\mathcal{L}(\Gamma_3) = \mathcal{L}(\Gamma_3 \div x)$，因此 $(\Gamma_3)^\omega$。根据 A-BoxE，我们有 $\Gamma_1 \vdash M^\bullet \Rightarrow \square\tau \mid \Gamma_2$和 $\Gamma_2, x\colon \tau \vdash N^\bullet \Rightarrow T \mid \Gamma_3$。根据引理 13，我们有 $\Gamma_1 \setminus \mathcal{L}(\Gamma_2) \vdash M^\bullet \Rightarrow \square\tau \mid \Gamma_2 \setminus \mathcal{L}(\Gamma_2)$。根据归纳假设，$\Gamma_1 \setminus \mathcal{L}(\Gamma_2) \vdash$erase$(M^\bullet)\colon \square\tau$和 $\Gamma_2, x\colon \tau \vdash$ erase$(N^\bullet)\colon T$。根据 T-BoxE,$\Gamma_1 \setminus \mathcal{L}(\Gamma_2) \circ \Gamma_2 \vdash$let box $x =$erase$(M^\bullet)$ in erase$(N^\bullet)\colon T$，并且根据擦除，let box $x =$erase$(M^\bullet)$ in erase$(N^\bullet) =$erase(let box $x = M^\bullet$in $N^\bullet$)。根据引理 12，$\mathcal{U}(\Gamma_1) = \mathcal{U}(\Gamma_2)$ 和 $\mathcal{L}(\Gamma_2) \subseteq \mathcal{L}(\Gamma_1)$。根据引理 2，我们有$\Gamma_1 \setminus \mathcal{L}(\Gamma_2) \circ \Gamma_2 = \Gamma_1$。

证明（定理 4的证明，引理 2）。通过对以下规则进行归纳即可得证：$\Gamma \vdash M\colon T$，T，利用引理 12 和引理 14。规则 T-Const，T-LinFunI 和 T-UnFunI 的证明是显然的。规则 T-Var，T-FunE，T-PairI，T-PairE 和 T-Match 的证明与规则 T-BoxE的证明采用相同策略。对于规则 T-Var，我们必须将归纳假设得到的判断从左向右弱化。下面详述几种情况：

– 规则 A-BoxI。假设 $\Gamma^\omega, \Gamma'^\omega \vdash$装箱 $(\overline{x}.M)\colon \square(\overline{\tau} \vdash^n T)$。根据 T-BoxI 和 T-Ctx，我们有 $\Gamma^{\geq n}, x\colon \tau^{<n} \overline{\vdash M}\colon T$。根据归纳假设，存在一个 $M^\bullet$，使得 erase$(M^\bullet) = M$，且 $\Gamma^{\geq n}, x\colon \tau^{<n} \vdash M^\bullet \Rightarrow T \mid \Delta$和 Δ^ω。将未使用上下文 Γ'按如下方式拆分：$\Gamma' \overset{n}{\rightsquigarrow} (\Gamma'_1)^{\geq n}, (\Gamma'_2)^{<n}$。然后我们可以先应用 A-Ctx，再应用 A-BoxI，得到$\Gamma^{\geq n}, \Gamma_2'^{<n} \vdash$装箱 $(x\colon \tau.M^\bullet) \Rightarrow \square(\overline{\tau} \vdash^n T) \mid \Delta \div \overline{x}, (\Gamma'_2)^{<n}$。根据引理 14，我们用 $(\Gamma'_1)^{\geq n}$弱化算法类型判断。因此，我们得到 $\Gamma, \Gamma' \vdash$装箱 $(\overline{x}.M^\bullet) \Rightarrow \square(\overline{\tau} \vdash^n T) \mid (\Delta \div \overline{x})\ \Gamma'$。容易验证 $((\Delta \div \overline{x})\ \Gamma')^\omega$。

– 规则 T-BoxE。假设 $\Gamma \circ \Delta \vdash$ let box $x = M$ in $N\colon T$。对前提进行归纳可得 $\Gamma \vdash M^\bullet \Rightarrow \square\tau \mid \Gamma'$with $(\Gamma')^\omega$and erase$(M^\bullet) = M$，以及$\Delta, x\colon \tau \vdash N^\bullet \Rightarrow T \mid \Delta'$ with $(\Delta')^\omega$and erase$(N^\bullet) = N$。由于 $\Gamma \cup \mathcal{L}(\Delta) = \Gamma \circ \Delta$，我们根据引理 14 弱化 $\Gamma \vdash M^\bullet \Rightarrow \square\tau \mid \Gamma'$，得到 $\Gamma \circ \Delta \vdash M^\bullet \Rightarrow \square\tau \mid \Gamma' \cup \mathcal{L}(\Delta)$。我们需要证明 $\Gamma' \cup \mathcal{L}(\Delta) = \Delta$。由于 $(\Gamma')^\omega$，因此 $\mathcal{L}(\Gamma' \cup \mathcal{L}(\Delta)) = \mathcal{L}(\Delta)$。根据引理 12，我们有 $\mathcal{U}(\Gamma) = \mathcal{U}(\Gamma')$。因此，$\mathcal{U}(\Gamma' \cup \mathcal{L}(\Delta)) = \mathcal{U}(\Gamma') = \mathcal{U}(\Gamma) = \mathcal{U}(\Gamma \circ \Delta) = \mathcal{U}(\Delta)$。因此，$\Gamma' \cup \mathcal{L}(\Delta) = \Delta$。根据 A-BoxE，我们有 $\Gamma \circ \Delta \vdash$let box $x = M^\bullet$in $N^\bullet \Rightarrow T \mid \Delta' \div x$。由于 $(\Delta')^\omega$，根据引理 11，$(\Delta' \div x)^\omega$。