arXiv:2611.32234v1 [cs.LO] 24 Nov 2026

并发可见性：采用一阶存储的高阶并发

Iwan Quémerais里昂高等师范学院与博洛尼亚大学
GuilhèmeJaber 南特大学
Ken Sakayori
东京大学
Davide Sangiorgi博洛尼亚大学与法国国家信息与自动化研究院

摘要

我们为一种带有一阶存储的（按值调用）并发高阶语言提出了一种操作博弈语义（引用可以包含其他引用或整数、布尔值等一阶值，但不能存储函数等高阶值）。我们将博弈语义中的可见性概念扩展到并发环境。该概念在语义上刻画了高阶引用的缺失，最初是为顺序高阶语言提出的。由此，我们定义了一种完备迹 预序，并通过引入语义配置的基于同步的组合、建立观测充分性结果，证明它对于上下文预序是可靠的。我们还证明了语言中函数返回一阶值的子集具有完备性。

与顺序可见性的情形不同，在标记迁移语义中，我们必须考虑多个具有不同可见性的活动线程，以及用于管理由此创建的线程间依赖关系的树形结构。此外，我们必须针对迹族而非单个迹进行推理，因为在并发环境中，某些动作之间的顺序无法强制规定。

2012年ACM主题分类 计算理论 → 程序语义

关键词和短语 操作博弈语义、高阶效应程序、可变存储

数字对象标识符 10.4230/LIPIcs.CONCUR.2026.15

资助 *Ke n Sakayori*: JSPS KAKENHI 项目编号 JP24K 20731

1 引言

编程语言语义中的一个核心问题是，某个给定程序构造的存在或缺失（例如可变存储、控制算子、并行性或非确定性选择）如何反映在语义中。从上下文等价的角度来看，可以通过询问添加某个构造是否赋予上下文更强的能力来区分程序，从而探讨这一问题。例如，在顺序高阶语言中，局部存储可以让上下文通过计算 f 被调用的次数，区分 $\lambda f.\, f0$ 与 $\lambda f.\, f0;\ f0$。

在交互式计算模型中，程序通过其可能的
与环境的交互，例如上文示例中对 f 的调用。因此，缺少某种程序构造往往可以在语义上刻画为对允许交互的限制。博弈语义 [12, 1], 诞生并发展于顺序高阶语言领域，它提供了若干引人注目的例子：缺少非局部控制算子对应于良括号性 [21], ，缺少高阶存储对应于可见性 [2], ，而对状态访问施加更严格的限制则会产生一些条件

© Iwan Quémerais、Guilhème Jaber、Ken Sakayori 和 Davide Sangiorgi；依据知识共享署名 4.0 国际许可协议授权。
第37届并发理论国际会议（CONCUR 2026）。编辑：Ana Sokolova 和 Patrick Totzke；文章编号：15；页码：15:1–15:20。莱布尼茨国际信息学会议论文集，Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl出版社，德国
LIPICS

例如无罪性 [12]。在基于标记迁移系统的博弈语义操作性表述中，这类条件通常通过扩充状态来强制实现，新增的组成部分用于记录关于过去交互的适当信息。

当上下文可以访问并行性时，它们能够区分出比在顺序设置中更多的程序。请看下面的例子：

▶**例 1**。

$\mathtt{M}_1 \triangleq \mathtt{let}\ x = \mathtt{ref\ true\ in}\ f\,(\lambda_.x := \mathtt{false}); \mathtt{false}$
$\mathtt{M}_2 \triangleq \mathtt{let}\ x = \mathtt{ref\ true\ in}\ f\,(\lambda_.x := \mathtt{false}); !x\ \&\&\ \mathtt{not}\,!x$

两个项首先都创建一个存储在 x 中的布尔引用，然后调用高阶函数 f，并将设置器函数作为参数传入。在顺序设置中，这两个项在上下文上是等价的，因为表达式 $!x$ && not$!x$ 始终求值为 false：在两次读取 x 期间，存储在引用 x 中的值不可能发生改变。相比之下，在一种将并行性与高阶引用结合的语言中，下面的上下文 E 和 f 的实例可以区分 $\mathtt{M}_1$ 和 $\mathtt{M}_2$：它将传递给 f 的设置器函数 $\lambda_.x := \mathtt{false}$ 存储在高阶引用 ℓ 中，并在独立线程中调用它；该调用可以在程序执行的两次读取之间进行。

$\mathtt{E} \triangleq \mathsf{par}\ (\mathsf{let}\, x = \bullet\, \mathsf{in}\, \mathtt{if}\ x\ \mathtt{then}\ ()\ \mathtt{else}\ \Omega)\ \mathsf{and}\ !\ell() \qquad f \triangleq \lambda x.\ell := x$

因此， $\mathtt{E}[\mathtt{M}_1]$ 总是发散，而 $\mathtt{E}[\mathtt{M}_2]$ 可以终止。高阶存储在此至关重要：对 f 的调用返回后，setter 就不再处于作用域内，若非如此便无法使用。这表明，即使存在并发，只要无法使用高阶引用，项 $\mathtt{M}_1$ 和 $\mathtt{M}_2$就应当保持等价。接下来的问题是如何从语义上刻画这一点。

作为语言，我们采用一种按值调用的高阶语言，支持并行组合和完整基类型引用（完整基类型指引用可以包含其他引用或整数、布尔值等一阶值，但不能包含函数等高阶值）。接着，我们将博弈语义中的可见性概念推广到并发环境。为此，我们首先定义一个交互式标记迁移系统（LTS），用于表达项与其环境之间可能发生的交互。遵循操作博弈语义和高阶并发的传统，该 LTS 的动作不能包含高阶值；高阶值会被抽象值（具体而言，即名称）所取代，这些名称提供了对实际值的访问途径。在这一环境中，线程的可见性直观上是指该线程获准调用的函数名称集合（即处于其作用域内的函数）。随后，我们将这个交互式 LTS 与另外两个组件结合起来：一个类型化*LTS*，用于控制新名称及其类型的引入；以及一个可见性标号转换系统，用于追踪当前活动线程可能各不相同的可见性。借助可见性 LTS，我们得以在模型中考虑不存在高阶引用这一事实。此类 LTS 还记录了语言项之间交互所遵循的调用—返回良括号纪律，这反映出该语言不包含非局部控制算子。在顺序情形下，良括号性通过栈来管理；而在我们的并发情形下，则必须采用树形结构，以记录所创建线程之间的依赖关系。

在复合标记迁移系统的基础上，我们定义了一种完全轨迹预序（直观而言，完整迹表示项与其环境之间一段已“完成”的交互序列）。我们的第一个主要结果是完全轨迹预序相对于上下文预序的可靠性。

这一结果通过引入语义配置的基于同步的组合，并确立观测充分性结果而得出。我们的第二个主要结果是，对于函数返回一阶值的语言子集，证明了完备性。完备性证明的常见策略是：考虑上下文中的两个项，以及其中一个项的一条完整迹。接着构造一个上下文，使其产生补轨迹，随后发出成功信号。然后，根据这两个项之间的上下文关系，可以得出第二个项也具有该轨迹。我们所处的并发环境中的难点在于，无法强制迹（由上下文或项产生）中某些动作的顺序。事实上，轨迹中某些相邻动作总是可以交换。因此，我们必须对迹族而非单条轨迹进行推理。这使证明大为复杂，也引出了论文中讨论的要求：函数必须返回一阶值。

本文不考虑程序与其环境共享存储的可能性。虽然博弈语义提供了一种通过丰富 LTS 的动作来表示这种共享存储的方法 [23, 28], ，但这会使模型变得复杂，而其方向与可见性的处理大多无关。

2 目标语言与操作语义

我们采用一种扩展了引用和并行组合的按值调用 λ-演算。一个引用可以包含对其他引用的引用——从而允许程序员编写循环数据结构——但不能存储函数；这类引用称为全地基 [29]。类型、值、项和（顺序）求值上下文由以下文法定义：

$$
\begin{aligned}
&\text{地基类型 } \iota ::= \mathtt{Unit} \mid \mathtt{Bool} \mid \mathtt{Int} \quad \text{引用类型 } \rho ::= \mathsf{ref}_\iota \mid \mathsf{ref}\,\rho \quad \text{类型 } \sigma,\tau ::\\
&= \iota \mid \rho \mid \sigma \times \tau \mid \sigma \to \tau \quad \text{值 } \mathtt{V},\mathtt{W} ::= x \mid \ell \mid () \mid \mathtt{b} \mid \mathtt{k} \mid \mathtt{V},\mathtt{W} \mid \lambda x.\mathtt{M} \quad \text{项 } \mathtt{M},\mathtt{N} ::\\
&= \mathtt{V} \mid \mathtt{V}\,\mathtt{M} \mid \mathsf{proj}_i\,\mathtt{V} \mid \mathtt{if\ V\ then\ M\ else\ N} \mid \mathtt{rec}\ f(x).\mathtt{M} \mid \mathtt{ref\ V} \mid !\mathtt{V} \mid \mathtt{V} := \mathtt{W} \mid \text{并行}\\
&\ \mathtt{M} \text{ 与 } \mathtt{N} \mid \mathtt{CAS\ V\ V\ V} \quad \text{顺序上下文 } \mathtt{E} ::= \bullet \mid \mathtt{V\,E} \quad \text{求值上下文 } \mathtt{D} ::= \bullet \mid \mathtt{V\,D} \mid \text{并行 } \mathtt{D}\\
&\text{与 } \mathtt{M} \mid \text{并行 } \mathtt{M} \text{ 与 } \mathtt{D}
\end{aligned}
$$

我们令 x 遍历变量，ℓ 遍历位置，$\mathtt{b} \in \{\mathtt{true}, \mathtt{false}\}$，并令 $\mathtt{k}$ 遍历整数。项 $\mathtt{par\ M}$ 和 $\mathtt{N}$ 是一个“并行对”，会并行执行第一个元素和第二个元素。 [4], 中可以找到这种并行对构造，它为我们的语言提供了灵感。比较并交换操作 $\mathtt{CAS}\ \ell\ \mathtt{V}_1\mathtt{V}_2$ 会以原子方式 比较位置 ℓ 上存储的值与 $\mathtt{V}_1$，如果二者相同，则将其替换为 $\mathtt{V}_2$ 。来自带存储的 λ-演算的其他构造都是标准的。通过在语法中限制可以出现项而非值的位置，我们强制求值顺序完全显式化，正如管理正规形 [30]中所做的那样。这样一来，用于定义操作语义和 OGS 的顺序求值上下文 $\mathtt{E}$ 和并行求值上下文 $\mathtt{D}$就有了特别简单的定义。作为语法糖，我们有时写作：$\mathsf{let}\ x = \mathtt{M}\ \mathsf{in}\ \mathtt{N}$ 表示 $(\lambda x.\mathtt{N})\,\mathtt{M}$；而 $\mathtt{M\,N}$（分别是 $\langle \mathtt{M},\mathtt{N}\rangle$）表示 $\mathsf{let}\ x = \mathtt{M}\ \mathsf{in}\ x\,\mathtt{N}$（分别是 $\mathsf{let}\ x = \mathtt{M}\ \mathsf{in}\ \mathsf{let}\ y = \mathtt{N}\ \mathsf{in}\ \langle x, y\rangle$）。

项的类型判断形式为 $\Sigma;\Gamma \vdash \mathtt{M} : \sigma$，其中 Γ （分别是 Σ）为类型上下文（分别是存储类型上下文），它是一个从变量（分别是位置）到

$$((\lambda x.M)\ \mathtt{V};\ \mathtt{S}) \mapsto_{\mathsf{red}} (M\{x := \mathtt{V}\};\ \mathtt{S})\ (\mathsf{proj}_i\ \mathtt{V}_1, \mathtt{V}_2;\ \mathtt{S}) \mapsto_{\mathsf{red}} (\mathtt{V}_i;\ \mathtt{S})\ (\mathtt{if\ true\ then\ M\ else\ N};\mathtt{S})$$
$$\mapsto_{\mathsf{red}} (\mathtt{M};\ \mathtt{S})\ (\mathtt{CAS}\ \ell\ \mathtt{V}_1\ \mathtt{V}_2;\ \mathtt{S}\cdot[\ell \mapsto \mathtt{V}_1]) \mapsto_{\mathsf{red}} (\mathtt{true};\ \mathtt{S}\cdot[\ell \mapsto \mathtt{V}_2])\ (\mathtt{if\ false\ then\ M\ else\ N};\mathtt{S})$$
$$\mapsto_{\mathsf{red}} (\mathtt{N};\ \mathtt{S})(\mathtt{CAS}\ \ell\ \mathtt{V}_1\ \mathtt{V}_2;\ \mathtt{S}\cdot[\ell \mapsto \mathtt{V}_3]) \mapsto_{\mathsf{red}} (\mathtt{false};\ \mathtt{S}\cdot[\ell \mapsto \mathtt{V}_3])\ (\mathtt{ref\ V};\mathtt{S}) \mapsto_{\mathsf{red}} (\ell;\ \mathtt{S}\cdot$$
$$[\ell \mapsto \mathtt{V}])\ (\text{若}\ \ \mathtt{V}_1 \neq \mathtt{V}_3)\ (\text{若}\ \ \ell \notin \mathsf{dom}(\mathtt{S}))\ (\text{并行}\ \ \mathtt{V}\ \text{且}\ \ \mathtt{W};\ \mathtt{S}) \mapsto_{\mathsf{red}} (\mathtt{V},\mathtt{W};\mathtt{S})\ (!\ell;\mathtt{S}) \mapsto_{\mathsf{red}} (\mathtt{S}(\ell);\ \mathtt{S})$$
$$(\text{若}\ \ \ell \in \mathsf{dom}(\mathtt{S}))\ (\mathtt{M};\mathtt{S}) \mapsto_{\mathsf{red}} (\mathtt{M}';\ \mathtt{T})\ (\ell := \mathtt{V};\mathtt{S}) \mapsto_{\mathsf{red}} (();\ \mathtt{S}\cdot[\ell \mapsto \mathtt{V}])\ (\mathtt{D}[\mathtt{M}];\ \mathtt{S}) \mapsto_{\mathsf{red}} (\mathtt{D}[\mathtt{M}'];\ \mathtt{T})$$

图1 归约语义。

类型。同样，求值上下文的类型判断形式为 $\Sigma;\Gamma \vdash \mathtt{D} : \tau \rightsquigarrow \sigma$，其中 τ 表示空位的类型，σ 表示结果的类型。类型规则是标准的，见附录 B.1

现在，我们定义主要的操作语义和行为等价，即归约语义和上下文等价。对于带引用的语言的归约关系，标准做法是使用存储（或堆），将其表示为从位置到值的映射。我们用 $[\ell \mapsto \mathtt{V}]$ 表示将 ℓ 映射到 V的单例存储；用 $\mathtt{S}\cdot\mathtt{T}$ 表示两个存储的不相交连接；用 $\mathtt{S}[\ell \mapsto \mathtt{V}]$ 表示将 ℓ 映射到 V 、将 $\ell' \neq \ell$ 映射到 $S(\ell')$ 的存储。判断 $\vdash \mathtt{S} : \Sigma$表示，对于每个 $\ell \in$dom(S)，都有 $\Sigma \vdash \mathtt{S}(\ell) : \Sigma(\ell)$。一个（归约）配置是由一个项和一个存储组成的对，记作 $(\mathtt{M};\ \mathtt{S})$。[1] 若 $\vdash \Sigma :\ \mathtt{S}$ 且 $\Sigma;\Gamma \vdash \mathtt{M} : \sigma$，我们记作 $\Sigma;\Gamma \vdash (\mathtt{M};\ \mathtt{S}) : \sigma$。归约规则见图1。左栏展示了带存储的顺序编程语言的标准操作规则。右栏的规则与并发有关。当并行对的两个分量都是值时，它会被视为普通对；否则，这两个分量可以按任意顺序归约。对于某个 $\mathtt{S}'$，若 $(\mathtt{M},\mathtt{S}) \mapsto^*_{\mathsf{red}} (\mathtt{V},\mathtt{S}')$，我们记作 $(\mathtt{M},\mathtt{S}) \Downarrow \mathtt{V}$ 。

通常，上下文等价关联的是在所有闭合上下文中具有相同终止性质的项。(Γ/τ)-测试上下文是一个带有空位的项，它（1）接受一个项$\varnothing$；$\Gamma \vdash \mathtt{M} :\ \tau$，且（2）在空位被填入后成为类型为 Unit 的闭项。

▶ 定义 *2*（上下文等价）。设 $\Sigma;\ \Gamma \vdash \mathtt{M}_1,\mathtt{M}_2 :\ \tau$。若对于所有（$\Gamma/\tau$）-测试上下文 C 和 $\mathtt{S} :\ \Sigma$，*(*$\mathtt{C}[\mathtt{M}_1];\ \mathtt{S}$*)* $\Downarrow$*()* 蕴含 *(*$\mathtt{C}[\mathtt{M}_2];\ \mathtt{S}$*)* $\Downarrow$*()*，则上下文预序 $\Sigma;\ \Gamma \vdash \mathtt{M}_1 \preceq_{ctx} \mathtt{M}_2 :\ \tau$成立。若 $\mathtt{M}_1$ 低于 $\mathtt{M}_2$，反之亦然，我们记作 $\Sigma;\ \Gamma \vdash \mathtt{M}_1 \simeq_{ctx} \mathtt{M}_2 :\ \tau$。

上下文等价的一种标准简化是用途的闭实例（CIU）等价 [27], ，它使用求值上下文，而非一般上下文。对于一个从变量到值的赋值 γ ，若对任意 $x \in \mathsf{dom}(\Gamma)$，都有$\Sigma;\emptyset \vdash \gamma(x) : \Gamma(x)$，我们记作 $\Sigma \vdash \gamma :\ \Gamma$ 。我们用 $\mathtt{M}\{\gamma\}$ 表示通过在 M 中并行替换，将 $x \in$ dom(γ)中的所有变量替换为 $\gamma(x)$后得到的项。

▶ 定义 *3.* 对于两个项 $\Sigma;\Gamma \vdash \mathtt{M}_1,\mathtt{M}_2 : \tau$我们称 $\mathtt{M}_1$ 是$\mathtt{M}_2$的 *CIU* 近似，记作 $\Sigma;\Gamma \vdash \mathtt{M}_1 \preceq_{ciu} \mathtt{M}_2 : \tau$ （或简称 $\mathtt{M}_1 \preceq_{ciu} \mathtt{M}_2$），如果对于所有满足 $\Sigma \subseteq \Sigma'$ 和 $\vdash \mathtt{S}$ 的 Σ', S, γ 和 $\mathtt{D}:\ \Sigma';$ $\Sigma' \vdash \gamma : \Gamma$ 以及 $\Sigma';\emptyset \vdash \mathtt{D} : \tau \rightsquigarrow \mathtt{Unit}$;

[1] 本文将介绍几种配置概念，但在含义可根据上下文明确判断时，通常将它们统称为配置。

$(\mathtt{D}[\mathtt{M}_1\{\gamma\}];\mathtt{S}) \Downarrow ()$ 蕴含 $(\mathtt{D}[\mathtt{M}_2\{\gamma\}];\mathtt{S}) \Downarrow ()$。如果 $\mathtt{M}_1 \preceq_{ciu} \mathtt{M}_2$ 和 $\mathtt{M}_2 \preceq_{ciu} \mathtt{M}_1$ 均成立，则称 $\mathtt{M}_1$ 和 $\mathtt{M}_2$ 是CIU 等价，记作 $\Sigma;\Gamma \vdash \mathtt{M}_1 \simeq_{ciu} \mathtt{M}_2 : \tau$ （或简称 $\mathtt{M}_1 \simeq_{ciu} \mathtt{M}_2$）。

CIU 前序与上下文预序一致。证明通过展示 CIU 前序是一个同余来完成，这遵循了 [?]中确立的一种标准方法。

▶ **命题 4. </style}我们有；** $\Gamma \vdash \mathtt{M}_1 \preceq_{ctx} \mathtt{M}_2 : \tau$ 当且仅当 $\varnothing;\Gamma \vdash \mathtt{M}_1 \preceq_{ciu} \mathtt{M}_2 : \tau$。

下面我们只研究程序与上下文之间不共享任何位置的交互。为了在归约过程中保持这一性质，我们要求它们接口处的类型不包含任何 **引用** 类型构造子。因此，如果 Γ 和 τ中均未出现引用类型（即使是作为子表达式出现），我们就称项$\Sigma;\Gamma \vdash \mathtt{M} : \tau$是无引用 的。

3 操作博弈语义

3.1 标记转换系统

上下文等价和 CIU 等价都需要对上下文进行大量的全称量化。我们的目标是遵循操作博弈语义的思路，推导出一个标记转换系统（LTS），并在此基础上定义迹语义，用于推理上下文等价。我们将通过几个步骤实现这一目标。

1. 首先，我们定义一个 LTS $\mathcal{L}_{\mathsf{env}}$，描述项如何与环境交互。该 LTS 不考虑项内部的归约；对于这些归约，我们仍采用基本归约语义（在操作博弈语义中，通常会将内部归约与和环境的交互分开处理）。

2. 在 LTS$\mathcal{L}_{\mathsf{env}}$中，动作可能携带高阶值。众所周知，这类动作会使行为关系的定义变得复杂（例如参见 [31, 32]）。在 OGS 中，此类高阶值会被转换为抽象值，直观而言，就是指向实际值的指针。这在第二步中完成，即图4中的 LTS$\mathcal{L}_{\mathsf{I}}$。在 $\mathcal{L}_{\mathsf{I}}$ 中，我们还将 LTS 的状态组织为一个线程池；每个线程，比如 $[\mathtt{q}]\mathtt{M}$，都是一个正在运行的项 $\mathtt{M}$，它将在 $\mathtt{q}$ 时计算出一个要返回给环境的值。

3. 仍需考虑的两个方面是类型判断和可见性；也就是说，我们必须确保只执行类型正确的动作，并确保动作遵守可见性所产生的约束（环境无法存储高阶值）。为考虑这两个方面，我们分别引入标记转换系统$\mathcal{L}_{\mathsf{Ty}}$（图5），以确保类型正确性；以及标记转换系统$\mathcal{L}_{\mathsf{vis}}$（图6），用于跟踪环境中线程的可见性视图如何演变。

4. 最终的标记转换系统由上述三个标记转换系统组合而成：$\mathcal{L}_{\mathsf{I}}$、$\mathcal{L}_{\mathsf{Ty}}$、$\mathcal{L}_{\mathsf{vis}}$；也就是说，合法动作是由$\mathcal{L}_{\mathsf{I}}$ 产生，并且受到类型化LTS $\mathcal{L}_{\mathsf{Ty}}$ 和可见性标号转换系统 $\mathcal{L}_{\mathsf{vis}}$允许的动作。

部分挂起项与标记转换系统 $\mathcal{L}_{\mathsf{env}}$

标记转换系统$\mathcal{L}_{\mathsf{env}}$ 描述项如何与环境交互。该标记转换系统的状态是项的扩展形式，称为部分挂起项，因为其中可能包含一些项：它们在询问环境后，正等待答复：

$$\text{Partially suspended terms} \quad \underline{\mathtt{M}}, \underline{\mathtt{N}} ::= \mathtt{M} \mid \{\mathtt{E}\}_{\mathsf{p}} \mid \mathtt{E}[\mathsf{par}\ \underline{\mathtt{M}}\ \mathsf{and}\ \underline{\mathtt{N}}]$$

我们将 $\mathtt{p}$ 这类名称称为延续名称。项$\{\mathtt{E}\}_{\mathsf{p}}$ 处于挂起状态，等待环境在 $\mathtt{p}$ 处提供一个值。我们用 $\mathtt{M}[(\mathtt{N},\mathtt{p})]$ 表示（部分挂起的）项 _

$$\frac{\mathtt{p}\text{ is fresh}}{\mathtt{E}[f\ \mathtt{V}] \overset{f(\mathtt{V},\mathtt{p})}{\longmapsto}_{\mathtt{env}} \{\mathtt{E}\}_{\mathtt{p}}} \qquad \frac{}{\{\mathtt{E}\}_{\mathtt{p}} \overset{\mathtt{p}(\mathtt{V})}{\longmapsto}_{\mathtt{env}} \mathtt{E}[\mathtt{V}]} \qquad \frac{\underline{\mathtt{M}} \overset{\mu}{\mapsto}_{\mathtt{env}} \underline{\mathtt{M}}'}{\mathsf{par}\ \underline{\mathtt{M}}\ \mathsf{and}\ \underline{\mathtt{N}} \overset{\mu}{\mapsto}_{\mathtt{env}} \mathsf{par}\ \underline{\mathtt{M}}'\ \mathsf{and}\ \underline{\mathtt{N}}}$$

$$\frac{\underline{\mathtt{N}} \overset{\mu}{\mapsto}_{\mathtt{env}} \underline{\mathtt{N}}'}{\mathsf{par}\ \underline{\mathtt{M}}\ \mathsf{and}\ \underline{\mathtt{N}} \overset{\mu}{\mapsto}_{\mathtt{env}} \mathsf{par}\ \underline{\mathtt{M}}\ \mathsf{and}\ \underline{\mathtt{N}}'} \qquad \frac{\mathsf{par}\ \underline{\mathtt{M}}_1\ \mathsf{and}\ \underline{\mathtt{M}}_2 \overset{\mu}{\mapsto}_{\mathtt{env}} \underline{\mathtt{N}}}{\mathtt{E}[\mathsf{par}\ \underline{\mathtt{M}}_1\ \mathsf{and}\ \underline{\mathtt{M}}_2] \overset{\mu}{\mapsto}_{\mathtt{env}} \mathtt{E}[\underline{\mathtt{N}}]}$$

图2 $\mathcal{L}_{\mathtt{env}}$的定义。

$$\frac{}{\mathtt{f}\colon \sigma \to \tau \Vdash \mathtt{f} : \sigma \to \tau} \qquad \frac{\mathtt{B}\text{ 一个基类型的基本值 } \iota}{\varnothing \Vdash \mathtt{B}\colon \iota}$$

$$\frac{\Delta_1 \Vdash \mathtt{A}_1 : \sigma_1 \qquad \Delta_2 \Vdash \mathtt{A}_2 : \sigma_2}{\Delta_1 \cdot \Delta_2 \Vdash \langle \mathtt{A}_1;\ \mathtt{A}_2 \rangle :\ \ \sigma_1 \times \sigma_2}$$

图3 抽象值的类型判断定义

通过将 $\mathtt{M}$ 在 $\mathtt{p}$ 上暂停的子项（例如$\{\mathtt{E}\}_{\mathtt{p}}$）替换为 $\mathtt{E}[\mathtt{N}]$ 得到。转换规则如图2所示。转换的形式为 $\mathtt{M} \overset{\mu}{\mapsto}_{\mathtt{env}} \mathtt{N}$，其中动作 μ 的形式为 $\mathtt{f}(\mathtt{V},\ \mathtt{p})$ 或 $\mathtt{p}(\mathtt{V})$。我们省略了将归约关系$\mapsto_{\mathsf{red}}$ 直接扩展到部分暂停项的过程，后续的标记转换系统将会用到它。

抽象值与标记转换系统 $\mathcal{L}_{\mathsf{l}}$

在 OGS 中，项（主张者）与环境（对手）之间交换的值会被抽象化，转换为抽象值（又称终极模式 [25]） $\mathtt{A}$，定义如下：

$$\mathtt{A} ::= \mathtt{B} \mid \mathtt{f} \mid \langle \mathtt{A}, \mathtt{A} \rangle$$

其中，$\mathtt{f} \in$FNames 是一个函数名，而 $\mathtt{B}$ 是一个基本值。每个语法值 $\mathtt{V}$ 都可以分解为一个抽象值 $\mathtt{A}$（其正骨架）以及一个相应的名称赋值映射 γ（其负填充），使得 $\gamma(\mathtt{A}) = \mathtt{V}$。抽象值的类型规则见图3。主张者与对手通过以下形式的步$\mathbf{m}$ 交换抽象值：

P-提问	P-回答	O-提问	O-回答
$\overline{\mathtt{f}(\mathtt{A},\ \mathtt{p})}^{\ \mathtt{q}}$	$\overline{\mathtt{p}}(\mathtt{A})$	$\mathtt{f}(\mathtt{A},\ \mathtt{p})^{\ \mathtt{q}}$	$\mathtt{p}(\mathtt{A})$

这些步是图4中交互式标号转移系统 $\mathcal{L}_{\mathsf{l}}$的标签。提问动作以续延名称作为索引，这在 OGS 中并不常见。这些索引类似于标号转移语义 [18, 19]，中有时使用的线程标识符，后者使我们能够识别动作源自哪个线程。引入这些索引是为了在技术上便于定义良括号性，但在考察轨迹时，它们被视为不可观测并予以移除。

标记转换系统的状态是交互式配置 $\mathbb{I}$，$\mathbb{J}$ 其形式为$\langle \mathtt{P};\ \mathtt{S}; \gamma \rangle$，其中包含 $\mathtt{P}$ 一个线程池 $\mathtt{S}$ 、一个存储，以及 γ 一个从名称映射到变量或 λ-抽象的赋值。线程池是线程的并行组合，形式为

$$\frac{(\underline{M};S) \mapsto_{\mathsf{red}} (\underline{N};T)}{\langle P|[p]\underline{M}|Q;S;\gamma\rangle \to_{\mathsf{I}} \langle P|[p]\underline{N}|Q;T;\gamma\rangle}$$

$$\mathrm{PQ}\ \frac{V = A\{\gamma'\} \qquad \underline{M} \xmapsto{f(V,p)}_{\mathsf{env}} \underline{N}}{\langle P|[q]\underline{M}|Q;S;\gamma\rangle \xrightarrow{\bar{f}(A,p)_q}_{\mathsf{I}} \langle P|[q]\underline{N}|Q;S;\gamma\cdot\gamma'\rangle} \qquad \frac{V = A\{\gamma'\}}{\langle P|[p]V|Q;S;\gamma\rangle \xrightarrow{\bar{p}(A)}_{\mathsf{I}} \langle P|Q;S;\gamma\cdot\gamma'\rangle}\ \mathrm{PA}$$

$$\mathrm{OQ}\ \frac{}{\langle P;S;\gamma\rangle \xrightarrow{f(A,p)_q}_{\mathsf{I}} \langle P|[p](\gamma(f)\,A);S;\gamma\rangle} \qquad \frac{\underline{M} \xmapsto{p(A)}_{\mathsf{env}} \underline{N}}{\langle P|[q]\underline{M};S;\gamma\rangle \xrightarrow{p(A)}_{\mathsf{I}} \langle P|[q]\underline{N};S;\gamma\rangle}\ \mathrm{OA}$$

图4 $\mathcal{L}$的定义$_{\mathsf{I}}$

$$\mathrm{PQ}\ \frac{\Delta_O(f) = \sigma \to \tau \quad \Delta \Vdash A : \sigma}{\langle \Delta_O \mid \Delta_P\rangle \xrightarrow{\bar{f}(A,p)_q}_{\mathsf{Ty}} \langle \Delta_O \mid \Delta_P, \Delta, p : \neg\tau\rangle} \qquad \frac{\Delta_O(p) = \neg\sigma \qquad \Delta \Vdash A : \sigma}{\langle \Delta_O \mid \Delta_P\rangle \xrightarrow{\bar{p}(A)}_{\mathsf{Ty}} \langle \Delta_O \mid \Delta_P, \Delta\rangle}\ \mathrm{PA}$$

$$\mathrm{OQ}\ \frac{\Delta_P(f) = \sigma \to \tau \quad \Delta \Vdash A : \sigma}{\langle \Delta_O \mid \Delta_P\rangle \xrightarrow{f(A,p)_q}_{\mathsf{Ty}} \langle \Delta_O, \Delta, p : \neg\tau \mid \Delta_P\rangle} \qquad \frac{\Delta_P(p) = \neg\sigma \qquad \Delta \Vdash A : \sigma}{\langle \Delta_O \mid \Delta_P\rangle \xrightarrow{p(A)}_{\mathsf{Ty}} \langle \Delta_O, \Delta \mid \Delta_P\rangle}\ \mathrm{OA}$$

图5 $\mathcal{L}_{\mathsf{Ty}}$的定义

$[p_1]\underline{M}_1|\cdots|[p_n]\underline{M}_n$。因此，并行性分为两个层次：线程池并行运行多个线程，而每个线程都可能因并行对构造而包含并行性。执行 OQ-迁移时会创建一个新线程。直观而言，这相当于并行调用一个闭包，同时运行其他线程。

类型化LTS $\mathcal{L}_{\mathsf{Ty}}$ 和可见LTS $\mathcal{L}_{\mathsf{vis}}$

类型化LTS $\mathcal{L}_{\mathsf{Ty}}$ 会跟踪新引入名称的类型，并确保所有内容始终符合类型要求。类型上下文配置记作 $\mathbb{S},\mathbb{T}$，其形式为$\langle\Delta_O \mid \Delta_P\rangle$，其中 Δ_O,Δ_P 是两个互不相交的类型上下文，分别将函数名映射到函数类型 $\sigma \to \tau$，将延续名称映射到上下文类型 $\neg\sigma$。状态为类型上下文配置的类型化LTS $\mathcal{L}_{\mathsf{Ty}}$见图5。 Δ_O中的名称称为对手名称，即由对手引入的名称。与之对称，Δ_P 中的名称则由主张者引入。

图 6 中的 可见标记转换系统 $\mathcal{L}_{\mathsf{vis}}$ 使我们能够管理对手名称的不同可见性。状态是配置 $\langle\mathcal{F},\ \mathcal{G},\ \mathcal{T}\rangle$，以 $\mathbb{V}$，$\mathbb{W}$ 表示，其中：

- $\mathcal{F}$：名称 $\to \mathcal{P}_{\mathrm{fin}}$(名称) 是一个从对手名称到支持者名称集合的函数，它用于说明每个对手名称的可见性（直观地说，即引入该对手名称时它已知晓哪些支持者名称；换言之，在调用对手名称时可以使用哪些由支持者引入的函数）；

- $\mathcal{G}$：CNames $\to \mathcal{P}_{\mathrm{fin}}$(Names) 是一个从主张者续延名称（被视为线程标识符）到主张者名称集合的函数，用于告知我们每个线程的可见性。规则 OQ 的前提 $\mathsf{f} \in \mathcal{G}(\mathsf{q})$ 是强制环境

$$\text{PQ}\ \frac{}{\langle \mathcal{F};\mathcal{G};\mathcal{T}\rangle \xrightarrow{\overline{\mathsf{f}(\mathtt{A},\mathsf{p})_{\mathsf{q}}}}_{\mathsf{vis}} \langle \mathcal{F};\mathcal{G}\cdot[\mathsf{p}\mapsto \mathcal{F}(\mathsf{f})\cup \mathsf{supp}(\mathtt{A})];\mathtt{addLeaf}(\mathcal{T},\mathsf{p},\mathsf{q})\rangle}$$

$$\text{PA}\ \frac{\mathtt{removeLeaf}(\mathcal{T},\mathsf{p}) = \mathtt{Some}(\mathcal{T}',\mathsf{q}^{?})}{\langle \mathcal{F};\mathcal{G};\mathcal{T}\rangle \xrightarrow{\overline{\mathsf{p}(\mathtt{A})}}_{\mathsf{vis}} \langle \mathcal{F};\mathcal{G}[\mathsf{q}^{?}\mapsto \mathcal{G}(\mathsf{q})\cup \mathsf{supp}(\mathtt{A})];\mathcal{T}'\rangle}$$

$$\text{OQ}\ \frac{\mathsf{f}\in\mathcal{G}(\mathsf{q})}{\langle \mathcal{F};\mathcal{G};\mathcal{T}\rangle \xrightarrow{\mathsf{f}(\mathtt{A},\mathsf{p})_{\mathsf{q}}}_{\mathsf{vis}} \langle \mathcal{F}\cdot\left(\prod_{\mathsf{g}\in\mathsf{supp}(\mathtt{A})}[\mathsf{g}\mapsto\mathcal{G}(\mathsf{q})]\right);\mathcal{G};\mathtt{addLeaf}(\mathcal{T},\mathsf{p},\mathsf{q})\rangle}$$

$$\text{OA}\ \frac{\mathtt{removeLeaf}(\mathcal{T},\mathsf{p}) = \mathtt{Some}(\mathcal{T}',\mathsf{q}^{?})}{\langle \mathcal{F};\mathcal{G}\cdot[\mathsf{p}\mapsto\mathcal{V}];\mathcal{T}\rangle \xrightarrow{\mathsf{p}(\mathtt{A})}_{\mathsf{vis}} \langle \mathcal{F}\cdot\prod_{\mathsf{f}\in\mathsf{supp}(\mathtt{A})}[\mathsf{f}\mapsto\mathcal{V}];\mathcal{G};\mathcal{T}'\rangle}$$

图 6 $\mathcal{L}_{\mathsf{vis}}$ 的定义

遵守可见性的关键条件。

- $\mathcal{T}$： `Tree` 是一棵树，用来表示延续名称的“多线程栈”，并记录各项之间返回调用交互的“良括号”结构。在不含控制算子的高阶语言中，良括号规范很常见。在顺序语言中，该规范通过延续名称栈及其操作来形式化。由于我们的语言是并发的，我们需要树形栈，以及用于处理这类树的操作。树的文法如下：

$$\mathtt{Tree} \triangleq \mathtt{Tree_aux} \mid \varepsilon \qquad \mathtt{Tree_aux} \triangleq \mathrm{CNames} \times (\mathtt{list}\ \mathtt{Tree_aux})$$

我们使用两个函数来操作树：

- `addLeaf`($\mathcal{T}$，p，q) 将 p 添加到树 $\mathcal{T}$中，作为 q 的叶节点；`removeLeaf`($\mathcal{T}$，p) 返回一个
- 选项类型；直观地说，该函数用于从$\mathcal{T}$中移除叶节点 p，并获得 p的父节点。形式化地说：
  - 如果 p 是$\mathcal{T}$中的叶节点，该函数返回 `Some`($\mathcal{T}$′，q?)，其中$\mathcal{T}$′ 是移除了 p 的树；如果 p 是$\mathcal{T}$的根节点，则 q? = `None`；否则 q? = `Some`(q)，其中 q 是 p 的父节点；
  - 如果 p 不是叶节点，该函数将返回 `None`。

OGS 标记转换系统$\mathcal{L}_{\mathsf{OGS}}$

操作博弈语义 (OGS) 标记转换系统 $\mathcal{L}_{\mathsf{OGS}} \triangleq$ (配置$_{\mathsf{ogs}}$，动作，$\mathtt{a}\rightarrow_{\mathsf{ogs}}$) 定义在形如 ($\mathbb{I}$，$\mathbb{S}$，$\mathbb{V}$) 的配置 $\mathbb{G}$，$\mathbb{H}\in$配置$_{\mathsf{ogs}}$之上，其转换关系是前面三个标记转换系统的转换关系的合取，并由以下规则定义：

$$\frac{\mathbb{I}\xrightarrow{\mathbf{a}}_{\mathsf{I}}\mathbb{J} \qquad \mathbb{S}\xrightarrow{\mathbf{a}}_{\mathsf{Ty}}\mathbb{T} \qquad \mathbb{V}\xrightarrow{\mathbf{a}}_{\mathsf{vis}}\mathbb{W}}{(\mathbb{I},\mathbb{S},\mathbb{V})\xrightarrow{\mathbf{a}}_{\mathsf{ogs}}(\mathbb{J},\mathbb{T},\mathbb{W})} \qquad \frac{\mathbb{I}\rightarrow_{\mathsf{I}}\mathbb{J}}{(\mathbb{I},\mathbb{S},\mathbb{V})\rightarrow_{\mathsf{ogs}}(\mathbb{J},\mathbb{S},\mathbb{V})}$$

当 OGS 配置的所有组件彼此一致时，该配置是有效的，这包括续延名称树栈相对于线程池的类型和结构；参见附录C，其中我们还展示了有效性在转换下得以保持。

$$\text{PI}\ \frac{\Gamma = \overrightarrow{(x_i : \tau_i)} \qquad \Delta_i \Vdash \mathtt{A}_i : \tau_i}{\langle \Gamma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle \xrightarrow{?(\overrightarrow{\mathtt{A}_i}, \mathtt{p})}_{\mathsf{ogs}} \left(\begin{array}{l} \langle [\mathtt{p}]\mathtt{M}\{\overrightarrow{\mathtt{A}_i/x_i}\}; \mathtt{S}; \varepsilon \rangle; \\ \langle \overrightarrow{\Delta_i}, \mathtt{p} : \neg\tau | \varepsilon \rangle; \\ \langle \prod\limits_{\mathtt{f} \in \mathsf{supp}(\overrightarrow{\mathtt{A}_i})} [\mathtt{f} \mapsto \emptyset]; \varepsilon; (\mathtt{p}, []) \rangle \end{array} \right)}$$

$$\text{OI}\ \frac{\Gamma = \overrightarrow{(x_i : \tau_i)} \qquad \Delta_i \Vdash \mathtt{A}_i : \tau_i \qquad \mathtt{A}_i\{\gamma_i\} = \delta(x_i) \qquad \mathtt{D} \xrightarrow{?(\mathtt{p})}_{\mathsf{env}} \underline{\mathtt{M}}}{\langle \vdash (\mathtt{D}, \delta, \mathtt{S}) : (\tau \rightsquigarrow \mathtt{Unit}; \Gamma) \rangle \xrightarrow{!(\overrightarrow{\mathtt{A}_i}, \mathtt{p})}_{\mathsf{ogs}} \left(\begin{array}{l} \langle [\mathtt{p}_f]\underline{\mathtt{M}}; \mathtt{S}; \overrightarrow{\gamma_i} \rangle; \\ \langle \mathtt{p}_f : \neg\mathtt{Unit} | \overrightarrow{\Delta_i}, p : \neg\tau \rangle; \\ \langle \varepsilon; [\mathtt{p} \mapsto \mathsf{supp}(\overrightarrow{\mathtt{A}_i})]; (\mathtt{p}_f, [(\mathtt{p}, [])]) \rangle \end{array} \right)}$$

图 7 $\mathcal{L}_{\mathsf{OGS}}$ 的初始步

▶ **例 5** （转换示例）。 从 $\mathtt{V} \triangleq \lambda\mathtt{f}.\ \mathtt{g}(\lambda_.\mathtt{f}())$出发，我们得到以下执行过程，其中省略了交互式配置的存储组件和类型配置：

$$\begin{array}{lll} \langle [\mathtt{p}_0]\mathtt{V}; \varepsilon \rangle; \\ \langle \mathcal{F}_0; \mathcal{G}_0; (\mathtt{q}_0, [(\mathtt{p}_0, [])]) \rangle & \xrightarrow{\overline{\mathtt{p}_0}(\mathtt{h})}_{\mathsf{ogs}} & \langle \varepsilon; \overbrace{[\mathtt{h} \mapsto \mathtt{V}]}^{\gamma_1} \rangle; \\ & & \langle \mathcal{F}_0; \underbrace{[\mathtt{q}_0 \mapsto \{\mathtt{h}\}]}_{\mathcal{G}_1}; (\mathtt{q}_0, []) \rangle \\ & & \langle [\mathtt{p}_1]\mathtt{V}\ \mathtt{f}; \gamma_1 \rangle \\ & \xrightarrow{\mathtt{h}(\mathtt{f};\mathtt{p}_1)_{\mathtt{q}_0}}_{\mathsf{ogs}} & \langle \underbrace{\mathcal{F}_0 \cdot [\mathtt{f} \mapsto \{\mathtt{h}\}]}_{\mathcal{F}_1}; \mathcal{G}_1; (\mathtt{q}_0, [(\mathtt{p}_1, [])]) \rangle \text{ since } \mathtt{h} \in \mathcal{G}_1(\mathtt{q}_0) \\ & & \langle [\mathtt{p}_1]\mathtt{g}\ (\lambda_.\mathtt{f}()); \gamma_1 \rangle \\ & \rightarrow_{\mathsf{ogs}} & \langle \mathcal{F}_1; \mathcal{G}_1; (\mathtt{q}_0, [(\mathtt{p}_1, [])]) \rangle \\ & & \langle [\mathtt{p}_1]\{\bullet\}_{\mathtt{q}_1}; \overbrace{\gamma_1 \cdot [v \mapsto \lambda_.\mathtt{f}()]}^{\gamma_2} \rangle \\ & \xrightarrow{\overline{\mathtt{g}}(v;\mathtt{q}_1)_{\mathtt{p}_1}}_{\mathsf{ogs}} & \langle \mathcal{F}_1; \mathcal{G}_1 \cdot [\mathtt{q}_1 \mapsto \{v\}]; (\mathtt{q}_0, [(\mathtt{p}_1, [(\mathtt{q}_1, [])])]) \rangle \end{array}$$

其中 $\mathcal{F}_0 \triangleq [\mathtt{g},\ \mathtt{p}_0 \mapsto \emptyset]$ 和 $\mathcal{G}_0 \triangleq [\mathtt{q}_0 \mapsto \emptyset]$。

3.2 完全轨迹预序

为了定义项的 OGS 轨迹，我们还必须添加初始步，从而将带类型项转换为$\mathcal{L}_{\mathsf{OGS}}$ 标记转换系统中适当的配置。这在图 7 的规则中完成。在 PI 规则中， $\mathtt{A}_i$ 是替换初始项自由变量的抽象值；$\mathtt{p}$ 是最终返回值将沿其传递的名称；树的初始化仅以 $\mathtt{p}$ 为根节点；可见性组件最初为空。OI 规则与之类似，适用于 CIU 前序中使用的环境，并将在可靠性证明中发挥作用。为了将并行求值上下文 $\mathtt{D}$ 转换为挂起项，我们向$\mathcal{L}_{\mathsf{env}}$ 标记转换系统添加规则 $\mathtt{E} \xrightarrow{?(\mathtt{p})}_{\mathsf{env}} \{\mathtt{E}\}_{\mathtt{P}}$，这样便可以挂起出现在并行上下文空位周围的最大顺序求值上下文 $\mathtt{E}$。此规则还使用一个专用的续延名称 $\mathtt{p}_f$ ，供对手执行最终应答。

在定义轨迹时，我们会擦除线程标签，因为这些标签不应被观察到；例如，$\mathtt{f}(\mathtt{A}, \mathtt{p})_{\mathtt{q}}$ 会变成 $\mathtt{f}(\mathtt{A},\mathtt{p})$，而 $\mathtt{f}(\mathtt{A}, \mathtt{p})_{\mathtt{q}}$ 会变成 $\mathtt{f}(\mathtt{A}, \mathtt{p})$。弱转移按预期方式定义。因此，$\bar{\ }$ $\Longrightarrow_{\mathsf{ogs}}$ 是.....的自反传递闭包。

$\rightarrow_{\mathsf{ogs}}$；以及$\mathbf{a} \Longrightarrow_{\mathsf{ogs}} \triangleq \Longrightarrow_{\mathsf{ogs}} \mathbf{a} \rightarrow_{\mathsf{ogs}} \Longrightarrow_{\mathsf{ogs}}$；最后，对于一个迹$\mathbf{t} = \mathbf{a}_1 \ldots \mathbf{a}_n$，我们写作$\mathbf{t} \Longrightarrow_{\mathsf{ogs}}$ for$\mathbf{a} \overset{1}{\Longrightarrow}_{\mathsf{ogs}} \cdots \mathbf{a} \overset{n}{\Longrightarrow}_{\mathsf{ogs}}$。完整迹 是指“所有询问都已得到回答”的迹；形式上，在迹的末尾，树栈应为空。

▶ 定义 6。$\langle \Gamma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle$ 的一个完整迹 t 是 $(\mathtt{M}, \mathtt{S})$ 的一个迹，且存在 $\mathbb{I}, \mathbb{S}, F, G$，使得 $\langle \Gamma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle \; t \Longrightarrow_{\mathsf{ogs}} (\mathbb{I}; \mathbb{S}; \langle F, G \; \varepsilon \rangle)$。我们用 $CTr_{\mathsf{OGS}}(\langle \Gamma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle)$ 表示 $\langle \Gamma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle$ 的完整迹集合。

▶ **定义 7**。给定 $\Sigma, \Gamma \vdash \mathtt{M}_1, \mathtt{M}_2 : \tau$，若对于所有满足 $\vdash \mathtt{S} : \Sigma$ 的 $\mathtt{S}$，都有 $\mathbf{CTr}_{\mathsf{OGS}}(\langle \Gamma \vdash (\mathtt{M}_1, \mathtt{S}) : \tau \rangle) \subseteq \mathbf{CTr}_{\mathsf{OGS}}(\langle \Gamma \vdash (\mathtt{M}_2, \mathtt{S}) : \tau \rangle)$，则项 $\mathtt{M}_1$ 是 $\mathtt{M}_2$ 的迹逼近，记作$\Sigma, \Gamma \vdash \mathtt{M}_1 \preceq_{\mathsf{tr}} \mathtt{M}_2 : \tau$。

示例

示例 8 表明，可见性约束可用于执行一些垃圾回收。这利用了存储不是高阶的这一性质，因此某些函数名虽然是自由的，却无法被环境使用。

▶ 示例 8. 考虑交互式配置 $\mathbb{I} \triangleq [\mathtt{p}]\mathtt{let}\ x = \mathtt{f}\ (\lambda y.\mathtt{M})\ \mathtt{in}\ \mathtt{N}; \mathtt{S}; \gamma$：对于任意类型配置 $\mathbb{S}$和可见配置 $\mathbb{V}$，如果 $(\mathbb{I}, \mathbb{S}, \mathbb{V})\ \mathtt{f}(\mathtt{g} \xrightarrow{\mathtt{p}}_{\mathsf{ogs}} \mathtt{q})\ \mathtt{q}(n) \longrightarrow_{\mathsf{ogs}} (\mathtt{P}; \mathtt{S}'; \gamma' \; \mathbb{S}', \mathbb{V}')$，那么 $\mathrm{Tr}_{\mathsf{OGS}}(\mathtt{P}; \mathtt{S}; \gamma \; \mathbb{S}, \mathbb{V}) = \mathrm{Tr}_{\mathsf{OGS}}(\mathtt{P}; \mathtt{S}'; \gamma' \setminus \{\mathtt{g}\} \; \mathbb{S}' \setminus \{\mathtt{g}\} \; \mathbb{V}')$。这是因为可见性保证了在收到答案 $\mathtt{q}(n)$ 后，$\mathtt{g}$ 会被遗忘。

更一般地，考虑一个配置 $(\mathtt{P}; \mathtt{S}; \gamma \; \mathbb{S}, \mathtt{F}; \mathtt{G}; \mathtt{T})$，以及一个满足 $\mathtt{g} \in \mathrm{dom}(\gamma)$ 的支持者名称 $\mathtt{g}$，如果 $\mathtt{g} \notin \mathrm{supp}(\mathtt{F}, \mathtt{G})$，那么 $\mathrm{Tr}_{\mathsf{OGS}}(\mathtt{P}; \mathtt{S}; \gamma \; \mathbb{S}, \mathtt{F}; \mathtt{G}; \mathtt{T}) = \mathrm{Tr}_{\mathsf{OGS}}(\mathtt{P}; \mathtt{S}; \gamma \setminus \{\mathtt{g}\} \; \mathbb{S} \setminus \{\mathtt{g}\}, \mathtt{F}; \mathtt{G}; \mathtt{T})$

▶ **例 9.** 回到引言中例 1 的项，其中 $!x$ && not$!x$ 定义为 if $!x$ then (if $!x$ then false else true) else false：在我们的并发语言中，$\mathtt{M}_1$和 $\mathtt{M}_2$ 迹等价。如果我们有高阶存储，那么一个区分迹可以这样开始：?()，r)。f(g，p)。p()。g(q)。q()。然而，如示例 8 所示，在回答 p() 之后，我们可以对 g 进行垃圾回收，这意味着标记转换系统不允许问题 g(q)。更一般地，在对 f(g，p) 作出回答之后，名称 g 就不能再使用，因此对手无法访问 x；所以 $!x$ && not$!x$ 总是返回 false。

4 OGS 的可靠性

本节旨在概述以下定理的证明，该定理刻画了迹语义预序与 ciu 预序之间的关系。

▶ **定理 10** (可靠性)。若 $\varnothing, \Gamma \vdash \mathtt{M}_1 \preceq_{\mathsf{tr}} \mathtt{M}_2 : \tau$，且 Γ, τ无引用，则 $\varnothing, \Gamma \vdash \mathtt{M}_1 \preceq_{ciu} \mathtt{M}_2 : \tau$。

通过同步进行组合

证明该定理的第一步，是引入一种组合方式，将项$\mathtt{M}$ 作为主动 OGS 配置的嵌入，与求值上下文 $\mathtt{D}$ 作为被动配置的嵌入组合起来。该组合通过$\mathcal{L}_{\mathsf{OGS}}$ 转换之间的一种同步来定义，这些转换分别由被动配置和主动配置触发。按照进程代数中的惯例，这种同步通过一种并行组合来定义。由于我们的目标是组合两个彼此“闭合”的 OGS 配置（就像组合

$\mathtt{D}[\mathtt{M}\{\gamma\}]$ 从而得到一个闭合项），我们不允许组合后的配置与外部世界交互。因此，并行组合被定义为一种归约关系。

我们通过引入可组合配置对（$\mathbb{G}_P$，$\mathbb{G}_O$）的概念，体现了两个 OGS 配置相互闭合这一思想。除其他要求外，它规定，一个配置中存在的每个暂停上下文都对应于另一个配置中的一个线程，并且它们各自的类型配置可以写成$\langle \Delta_O \mid \Delta_P \rangle$ 和$\langle \Delta_P,\ \mathrm{p}f :\ \neg\mathtt{Unit} \mid \Delta_O \rangle$。对手类型配置中的额外续延名称 $\mathrm{p}f$ 用于执行最终的对手回答。

▶ **定义 11**（并行组合）。我们将归约关系 $\mapsto_{\mathsf{so}}$ 定义在 *OGS*可组合配置对 $(\mathbb{G}_1 \parallel \mathbb{G}_2)$上，记作 $\mapsto_{\mathsf{so}}^{\mathsf{red}} \cup \mapsto_{\mathsf{so}}^{\mathsf{sync}}$，其中：

$$\frac{\mathbb{G}_P \to_{\mathsf{ogs}} \mathbb{H}_P \qquad \mathsf{supp}(\mathbb{H}_P) \cap \mathsf{supp}(\mathbb{G}_O) \subseteq \mathsf{supp}(\mathbb{G}_P) \cap \mathsf{supp}(\mathbb{G}_O)}{(\mathbb{G}_P \parallel \mathbb{G}_O) \mapsto_{\mathsf{so}}^{\mathsf{red}} (\mathbb{H}_P \parallel \mathbb{G}_O)}$$

$$\frac{\mathbb{G}_O \to_{\mathsf{ogs}} \mathbb{H}_O \qquad \mathsf{supp}(\mathbb{H}_O) \cap \mathsf{supp}(\mathbb{G}_P) \subseteq \mathsf{supp}(\mathbb{G}_O) \cap \mathsf{supp}(\mathbb{G}_P)}{(\mathbb{G}_P \parallel \mathbb{G}_O) \mapsto_{\mathsf{so}}^{\mathsf{red}} (\mathbb{G}_P \parallel \mathbb{H}_O)} \qquad \frac{\mathbb{G}_P \xrightarrow{\mathbf{m}}_{\mathsf{ogs}} \mathbb{H}_P \qquad \mathbb{G}_O \xrightarrow{\overline{\mathbf{m}}}_{\mathsf{ogs}} \mathbb{H}_O}{(\mathbb{G}_P \parallel \mathbb{G}_O) \mapsto_{\mathsf{so}}^{\mathsf{sync}} (\mathbb{H}_P \parallel \mathbb{H}_O)}$$

根据 [24]，，$\mapsto_{\mathsf{so}}^{\mathsf{red}}$的名义侧条件确保一个配置中新分配的位置不会出现在另一个配置中。因此，可组合性在 $\mapsto_{\mathsf{so}}^{\mathsf{sync}}$下得以保持。我们使用并行组合来观察两个 OGS 配置之间的同步过程。因此，当存在两个 OGS 配置$\mathbb{H}_P$，$\mathbb{H}_O$ 使得 $(\mathbb{G}_P \parallel \mathbb{G}_O) \mapsto_{\mathsf{so}}^{*} (\mathbb{H}_P \parallel \mathbb{H}_O)$，且 $\mathbb{H}_P$ 的交互式配置具有$\langle \mathtt{S}_P;\ \gamma_P \rangle$的形式，而 $\mathbb{H}_O$ 的交互式配置具有$\langle [\mathrm{p}]();\ \mathtt{S}_O;\ \gamma_O \rangle$的形式时，我们记作 $(\mathbb{G}_P \parallel \mathbb{G}_O) \Downarrow \mathrm{p}$。

观测充分性

由于并行组合 $(\mathbb{G}_P \parallel \mathbb{G}_O)$ 由两个彼此闭合且可组合的配置构成，我们引入一种将它们合并为一个操作配置的方法，记作 $(\mathbb{G}_P \between \mathbb{G}_O)$，其形式为 $(\mathtt{M}\{\gamma\}\ \mathtt{S})$，其中：

- $\mathtt{M}$ 的定义是将 $\mathbb{G}_P$ 和 $\mathbb{G}_O$ 的线程池交错合并为一个线程池，并填入各自对应的空位。它使用可见配置中的树 $\mathcal{T}_P$ 和 $\mathcal{T}_O$，将线程代入各自对应的暂停上下文。γ 的定义是将 $\mathbb{G}_P$ 和 $\mathbb{G}_O$ 的值赋值交错合并为一个，并遵循其定义域中函数名的引入顺序。因此，需要确保这种顺序确实存在；这在可组合性的定义中得以保证，并由并行组合的归约语义维持。
- $\mathtt{S}$ 则只是将 $\mathbb{G}_P$ 和 $\mathbb{G}_O$各自的存储串接起来，即 $\mathtt{S}_P \cdot \mathtt{S}_O$；由于我们不允许共享存储，因此它们彼此不相交，而且只包含一阶值，所以无需对它们使用 γ。

虽然 γ_P 和 γ_O 的交错方式与顺序 OGS 中采用的构造相同（例如 [14, 5]），但线程池的交错方式是新的。事实上，在顺序良括号设置中，我们只有两个长度相同的栈，而不是两棵树结构。

因此，我们的目标是将通过并行组合的归约语义定义的、关于 $(\mathbb{G}_P \parallel \mathbb{G}_O)$ 的观察，与通过操作归约定义的、关于 $(\mathbb{G}_P \between \mathbb{G}_O)$的终止观察联系起来。

▶ **定理 12**。给定 $\mathbb{G}_P$、$\mathbb{G}_O$ 两个可组合的 *OGS* 配置，则 $(\mathbb{G}_P \parallel \mathbb{G}_O) \Downarrow \mathrm{p}f$当且仅当 $(\mathbb{G}_P \between \mathbb{G}_O) \Downarrow ()$。

从左到右的方向遵循 OGS 的标准证明，即如果 $(\mathbb{G}_P \mid\mid \mathbb{G}_O) \mapsto_{so}^{sync} (\mathbb{H}_P \mid\mid \mathbb{H}_O)$ 那么 $(\mathbb{G}_P \between \mathbb{G}_O) = (\mathbb{H}_P \between \mathbb{H}_O)$ 。在编程语言中存在并行性时，从右到左的方向则更具挑战性。事实上，当 $(\mathbb{G}_P \between \mathbb{G}_O)$可进行操作归约时，$(\mathbb{G}_P \mid\mid \mathbb{G}_O)$ 可能需要执行一些同步步骤，才能进行相同的归约。随后我们证明，可以始终积极地执行同步步骤，而不会失去在归约可用时执行归约步骤的可能性。最后，我们证明，两个归约步骤之间始终只有有限个同步步骤。

由 $(\langle \Sigma \vdash (M, S) : \tau \rangle \mid\mid \langle \vdash (D, \gamma, S') : (\tau \rightsquigarrow \mathtt{Unit};\ \Gamma) \rangle) \mapsto_{so}^{sync} (\mathbb{G}_P \mid\mid \mathbb{G}_O)$，我们推导出 $(\mathbb{G}_P \between \mathbb{G}_O) = (D[M\{\gamma\}], S \cdot S')$；接着，我们推导出定理 12 关于初始配置的以下推论。

▶ **推论 13**。$(D[M\{\gamma\}], S \cdot S') \Downarrow ()$ 当且仅当 $(\langle \Sigma \vdash (M, S) : \tau \rangle \mid\mid \langle \vdash (D, \gamma, S') : (\tau \rightsquigarrow \mathtt{Unit};\ \Gamma) \rangle) \Downarrow p^f$。

可靠性证明

证明可靠性陈述的最后一个要素，是将并行组合的归约关系所定义的观测与完整迹的存在联系起来。

▶ **定理 14.** 取 $\mathbb{G}_P, \mathbb{G}_O$ 两个可组合的 *OGS* 配置，则 $(\mathbb{G}_P \mid\mid \mathbb{G}_O) \Downarrow p^f$ 当且仅当存在一个完整迹 t 使得 $t \in \mathbf{CTr}_{OGS}(\mathbb{G}_P)$ 且 $t \cdot p^f() \in \mathbf{CTr}_{OGS}(\mathbb{G}_O)$。 ———

▶ 推论 *15.* $(\langle \Sigma \vdash (M, S) : \tau \rangle \mid\mid \langle \vdash (D, \gamma, S') : (\tau \rightsquigarrow \mathtt{Unit};\ \Gamma) \rangle) \Downarrow p^f$ 当且仅当存在一个迹 t，使得 $t \in CTr_{OGS}(\langle \Sigma \vdash (M, S) : \tau \rangle)$ 且 $t \cdot p^f() \in CTr_{OGS}(\langle \vdash (D, \gamma, S') : (\tau \rightsquigarrow \mathtt{Unit};\ \Gamma) \rangle)$

现在我们已经具备证明定理 10 所需的一切。

证明。取 $\Sigma, \Gamma, \tau, M_1, M_2$ 使得 $\Sigma, \Gamma \vdash M_1 \preceq_{tr} M_2 : \tau$。令 $\Sigma', \gamma, D, S, S'$ 如 ciu 预序的定义 3 中所述，且 $(D[M_1\{\gamma\}]; S \cdot S') \Downarrow ()$。我们证明 $(D[M_2\{\gamma\}]; S \cdot S') \Downarrow ()$。根据推论 13，我们有 $(\Sigma \vdash (M_1, S) : \tau \mid\mid \vdash (D, \gamma, S') : (\tau \rightsquigarrow \mathtt{Unit};\ \Gamma)) \Downarrow p^f$。由推论 15，我们推断存在一个迹 t，使得 $t \in \mathrm{CTr}_{OGS}(\Sigma \vdash (M_1, S) : \tau)$，且 $t \cdot p^f() \in \mathrm{CTr}_{OGS}(\vdash (D, \gamma, S') : (\tau \rightsquigarrow \mathtt{Unit};\ \Gamma))$。

我们有 $\Sigma, \Gamma \vdash M_1 \preceq_{tr} M_2 : \tau$，因此 $t \in \mathrm{CTr}_{OGS}(\Sigma \vdash (M_2, S) : \tau)$。根据推论 15，$(\Sigma \vdash (M_2, S) : \tau \mid\mid \vdash (D, \gamma, S') : (\tau \rightsquigarrow \mathtt{Unit};\ \Gamma)) \Downarrow p^f$；最后，根据推论 13，$(D[M_2\{\gamma\}]; S \cdot S') \Downarrow ()$。◀

5 完备性

我们证明了语言的一个子集具有完备性。在深入探讨该结果之前，我们先讨论整个语言的完备性问题。在 OGS 中，证明完备性的通常策略是，取两个上下文等价项，以及其中一个项的一个迹 t 。然后构造一个生成补轨迹的上下文；因此，根据这两个项之间的上下文等价性，并结合推论 13 和 15，可以推断另一个项也能够产生迹 t。现在，假设 t 如下所示，其中使用“V”表示一个抽象值，使得对“V”中名称的询问会触发对 V 中名称的询问（对于不相关的子项，则为 _ 或...）：

$$f(_, p) \cdot f(_, q) \cdot \overline{p}(g) \cdot f(\text{“g”}, r) \cdot \overline{q}(h) \cdot f(\text{“}\langle g, h \rangle\text{”}, s) \cdot \overline{r}() \cdot \overline{s}()$$

迹 $\mathbf{t}$ 确实可以由一个配置生成。下面解释为什么其对偶迹 $\mathbf{t}$ 无法由上下文生成。$\mathbf{t}$ 的前两个动作必然分别来自并行组件，因为在$\mathbf{t}$中，第二个询问先于第一个询问得到回答。也就是说，上下文应当具有如下形式：$\mathsf{par\ let\ g\ = f\ _ in \ldots and\ let\ h\ = f\ _\ in \ldots}$。

此外，在$\mathbf{t}$中，问题 f(“g”,r) 出现在 q 处的回答之前，并且使用了 p 处回答中引入的一个名称；这迫使上下文具有 $\mathsf{par\ let\ g\ = f\ _\ in\ f\ g\ and\ let\ h\ = f\ _\ in \ldots}$ 的形式。相应地，$\mathbf{t}$ 应当展示动作 f(“$\langle \mathsf{g}, \mathsf{h}\rangle$”,$\mathsf{s}$)；这是不可能的，因为上下文中没有任何子项同时知道 g 和 h。

本节余下部分我们聚焦于受限语言：它要求边界处的函数只返回一阶值，我们称之为 *fo-returning* 片段。尽管有此限制，并发仍带来了重大挑战。首先，迹中两个连续动作发生的顺序并非总能得到强制。也就是说，产生给定迹的项或上下文也会产生其他迹，而这些迹是通过交换某些动作得到的。直观而言，当两个动作相互独立时，便可以进行交换；也就是说，它们在语法上彼此无关，并且它们之间不存在内部因果性（即通过引用产生的因果关系）。具体来说，在以下情形中可能出现这种情况：两个动作都是主张者步；两个动作都是对手步；第一个是主张者步$\mathbf{p}$，第二个是$\mathbf{o}$对手步，且该步并非由主张者步*justified*（记作$\mathbf{p} \not\curvearrowright \mathbf{o}$），也就是说，对手步没有使用主张者步中引入的名称[2]。若$\mathbf{t}' \preceq \mathbf{t}$ 是$\mathbf{t}'$ 的合法排列，我们就记作$\mathbf{t}$，其中 $\preceq$ 是由以下连续动作之间的关系所诱导的、迹上的最小前同余：

$$\frac{}{\mathbf{o}\cdot\mathbf{o}' \preceq \mathbf{o}'\cdot\mathbf{o}} \qquad \frac{}{\mathbf{p}\cdot\mathbf{p}' \preceq \mathbf{p}'\cdot\mathbf{p}} \qquad \textsc{Crt}\ \frac{\mathbf{p} \not\curvearrowright \mathbf{o}}{\mathbf{o}\cdot\mathbf{p} \preceq \mathbf{p}\cdot\mathbf{o}}$$

▶ 注 16. 礼貌规则 `Crt` 的对偶并未出现，这意味着对手步后接支持者步不能交换。原因在于，在这些情况下，可以利用引用 [9]产生内部因果性。举例来说，考虑一个线程池 $[\mathsf{p}]\{\mathtt{E}\}_{\mathsf{r}}|[\mathsf{q}]\mathsf{f}\,\mathtt{V}$：对 r 的回答和对 f 的提问看起来可能彼此独立；然而，我们可以在它们之间强制建立一个顺序，如下所示：

$$[\mathsf{p}]\{(\lambda x.\ \ell := 1; x)\mathtt{E}\}_{\mathsf{r}}|[\mathsf{q}]\mathtt{if}\ !\ell = 1\ \mathtt{then}\ \mathsf{f}\,\mathtt{V}\ \mathtt{else}\ \Omega$$

如果在对 r 作出回答之前求值 `if then else`，那么进程就会发散；若对 f 的提问发生在对 r 作出回答之后，则可以避免这种情况。

▶ 备注17（发散）。与上一条备注一样，利用发散舍弃某些项的不期望行为是一种惯常做法。这利用了完整迹而非一般迹的优势：当项的任何部分发生发散时，整个项都无法终止，执行也无法完成为一条完整迹，因此无需考虑该执行。

然而，我们还必须考虑到，我们的语言不允许高阶存储，因此只有可见迹才相关，也就是说，直观而言，只有由可见LTS在某个给定的可见配置上生成的轨迹才相关。引理18表明，对于可见迹，上文定义的置换确实是可能的。

▶**引理18。 </style}若$\mathbf{t} \in \mathbf{CTr}_{\mathsf{OGS}}(\mathbb{G})$，则对于所有满足$\mathbf{t}' \preceq \mathbf{t}$ 且$\mathbf{t}'$ 从 $\mathbb{V}_{\mathbb{G}}$ 可见的情况，$\mathbf{t}' \in \mathbf{CTr}_{\mathsf{OGS}}(\mathbb{G})$*</style}*。**

在引理的证明中，我们对$\mathbf{t}' \preceq \mathbf{t}$的证明进行归纳。困难之处在于
case 是礼貌规则 `Crt`，有时，置换可能意味着对

[2] 这在并发博弈语义中称为“礼貌性” [7]

所涉及的对手动作进行修改，例如更改 f(A，p)q 中的线程标签 q。最后也是至关重要的一步是证明：对于任意给定的完全可见轨迹，都存在一个上下文，能够生成其补轨迹及其所有合法排列。

▶ 引理 *19.* 对于 *fo-returning* 片段中的所有 $\mathtt{M}, \mathtt{S}, \Gamma, \tau$，若 Γ, τ 无引用，则对于每条完整迹 $t \in CTr_{\mathsf{OGS}}(\langle \Gamma \vdash (\mathtt{M}, \mathtt{S}) : \tau \rangle)$，都存在一个上下文 $\mathtt{E}$、一个值赋值 γ 和一个与 $\mathtt{S}$ 不相交的存储 $\mathtt{S}'$，使得 $CTr_{\mathsf{OGS}}(\langle \vdash (\mathtt{E}, \gamma, \mathtt{S}') : (\tau \to \mathtt{Unit}; \Gamma) \rangle) = \{t' | t' \preceq t \cdot p^f$，且 t' 可见$\}$。

这个可定义性结果中构造的上下文是一个顺序上下文，在与项首次交互之前不会使用任何并行组合。我们从所构造的环境 $\mathtt{E}$，γ 的角度简述推理。以下问题需要谨慎处理：

- 如注 16 所示，对手步不能与其后紧接的主张者步交换顺序。可以按照该注中提出的思路来确保这一点。对于每个对手函数名 f，若以 f 为主体的询问有 n 个，则 f 应恰好被
- 调用 n 次。

我们使用锁来处理这些问题，并通过引用对锁进行建模。可以检查锁以了解其状态（打开/关闭）；还可以利用 CAS 构造以原子方式更改其状态。如果在某个分支中遇到意外情况，我们就强制发散，从而排除执行该分支的可能性。

- 锁通过引用建模：设为 0 表示锁已关闭，设为 1 表示锁已打开。对于每个对手步 $\mathbf{o}$，我们创建一个锁 $\ell_{\mathbf{o}}$，初始状态为关闭。执行对手步 $\mathbf{o}$ 后，我们打开该锁。对于每个支持者步 $\mathbf{p}$，我们考虑在$\mathbf{p}$ 之前应当发生的对手步集合，以及 $L_{\mathbf{p}}$ 它们对应的锁集合。执行任何支持者步 $\mathbf{p}$之前，我们检查 $L_{\mathbf{p}}$ 中的引用是否都设为 1；如果不是，就令程序发散：$\mathtt{check_lock}_{\mathbf{p}} \triangleq$ 并行$_{\ell \in L_{\mathbf{p}}}$(if $!\ell = 1$ then () else Ω) 使用相同的锁，我们确保每个对
- 手询问恰好发生一次。在一次对手询问 $\mathbf{o}$之后，以及在由 $\mathbf{o}$ 触发的任何支持者动作之前，我们检查锁 $\ell_{\mathbf{o}}$ 是否设为 0：如果是，就将其设为 1 并继续；否则，令程序发散。为确保交错无法绕过检查，我们使用原子构造 CAS $\ell_{\mathbf{o}}$ 0 1。

还需要处理一个技术细节，即同一函数名对应多个对手询问。为此，我们在这些函数的定义中，为每个询问分别设置一个分支；选择分支时，首先检查收到的抽象值的结构；若某些抽象值无法区分，则通过在不同选项之间引入竞争来设置非确定性分支。

利用上述思路，我们构造一个求值上下文 $\mathtt{E}_{\mathbf{t}}$、一个值赋值 $\delta_{\mathbf{t}}$ 和一个存储 $\mathtt{S}_{\mathbf{t}}$，使其能够生成预期的迹。接下来，我们需要证明三项结果：该上下文具有预期类型；该上下文会生成这条迹；该上下文不会生成意外的完整迹（即，迹 $\mathbf{t}'$ 满足 $\neg(\mathbf{t}' \preceq \mathbf{t})$）。对于最后一项结果，直观地说，使用锁可以确保上下文一旦生成不需要的前缀迹，就会发散。

▶ **定理 20**（关于 fo-returning 片段的完备性）。若 $\varnothing$、$\Gamma \vdash \mathtt{M}_1 \preceq_{ciu} \mathtt{M}_2 : \tau$，其中 $\mathtt{M}_1$、$\mathtt{M}_2$ 是 *fo-returning* 片段中的项，且 Γ、τ 无引用，则 $\varnothing$、$\Gamma \vdash \mathtt{M}_1 \preceq_{\mathsf{tr}} \mathtt{M}_2 : \tau$。

证明。取一个由 $\mathtt{M}_1$生成的迹 $(\mathtt{E}, \gamma, \mathtt{S}')$ 生成$\{\mathbf{t}' | \mathbf{t}' \preceq \mathbf{t} \cdot \mathrm{p}^f$，且 $\mathbf{t}'$ 可见 $\}$，利用上下文存在这一事实，沿着 CIU 近似、推论 13 和 15 的两个方向，

我们有 $(\Gamma \vdash (\mathtt{M}_2, \mathtt{S}) : \tau \ ||\vdash (\mathtt{E}, \gamma, \mathtt{S}') : (\tau \rightsquigarrow \mathtt{Unit}; \Gamma)) \Downarrow \mathrm{p}_f$。因此，存在某个可见的 t′，它由 $\mathtt{M}_2$ 生成，使得 $\mathrm{t}' \cdot \mathrm{p}_f$ 由该上下文生成。这意味着 $\mathrm{t}' \cdot \mathrm{p}_f \preceq \mathrm{t} \cdot \mathrm{p}_f$；由此可得 $\mathrm{t} \preceq \mathrm{t}'$，因此 t 是由 $\mathtt{M}_2$生成的完整迹。◀

6 结论、相关工作与未来工作

我们研究了如何在并发环境中纳入可见性这一操作博弈语义（OGS）中的核心概念。

目前我们尚不清楚完备性是否适用于完整语言。我们的 OGS 可以被视为 [14]，的并发扩展；后者系统地发展了顺序情形下 OGS 的可见性和良括号性。据我们所知，第一个考虑（O-）可见性的 OGS 是 [13]。[13] 与我们的工作（以及 [14]）之间的一个区别是，我们不允许程序与环境之间交换位置。要处理位置交换及其产生的共享存储，需要将步推广为带存储的动作 [23, 28], ，其中附带的存储包含公开位置的值。在我们的并发环境中，我们将这些问题留待未来研究。尽管并发对象的迹语义 [19]并非 OGS 的一个实例，但它也采用了博弈语义中的一些概念，例如良括号性和礼貌性。其良括号性的定义充分利用了线程标识符，并要求每个具有相同标识符的子轨迹都符合标准意义上的良括号性。

与我们的 OGS 最为接近的指称博弈模型之一，是 Ghica 和 Murawski 提出的模型 [9]。他们为理想化并行 Algol（本质上是我们目标语言的按名调用变体）开发了一种博弈语义，该语义相对于可能测试等价是完全抽象的。尽管在某些情况下，操作式博弈语义与指称博弈语义之间的关系已被形式化（例如，[13] 将 [23] 与 [28]联系起来），但目前还没有通用方法来关联操作式方法与指称方法。未来一项有趣的工作，是将我们的 OGS 与 Ghica 和 Murawski 的指称模型正式联系起来。Ghica-Murawski 模型还被扩展为一种基于事件结构的因果模型 [7, 6]。无论是在顺序情形还是因果情形中，都没有明确提出可见性的概念；我们指出，因果模型的可见性条件 [6] 更接近无罪性这一概念，而非缺少高阶引用。另一方面，Laird 通过引入并发指针 [22]，（附加到对局上的内部线程信息），在并发环境中提出了可见性的概念。探究并发指针与我们以延续名称进行标记的关系，则留待未来工作。

可见性的概念已被纳入 π-演算，作为一种类型系统 [10]。然而，计算是纯粹顺序进行的，如何将并发可见性表示为一种类型系统仍有待解决。这使得我们难以将 OGS 与进程编码联系起来，不同于纯 λ-演算的情况；对于后者，已有形式化联系 [16]。

用于推理带引用或异步通道和/或并发的高阶语言程序等价性的其他操作技术包括 Kripke 逻辑关系 [4, 8] 以及各种形式的互模拟——例如上下文互模拟、范式 [31, 18, 26]，范式 [3] 互模拟和环境 [32, 20] 互模拟——或它们的混合形式 [11, 17, 15]。在这些研究中，如果考虑引用，通常讨论的是完整的高阶引用，而非基类型引用。一个例外是 [8] ：该研究将高阶顺序语言中缺少高阶引用这一点，刻画为能够“回溯”Kripke框架。[15]，通过与可见性概念建立形式化联系，将这一技术引入 Kripke范式双模拟。将这种“回溯”证明技术拓展到并发环境，并将其与我们的并发可见性概念联系起来，是一个令人振奋的研究方向。

引用

1 Samson Abramsky、Radha Jagadeesan 和 Pasquale Malacaria。PCF 的完全抽象性。《信息与计算》，163(2):409–470，2000年。网址：https://doi.org/10.1006/inco.2000.2930, doi:10.1006/INCO.2000.2930。2 Samson Abramsky 和 Guy McCusker。线性性、共享与状态：带主动表达式的理想化 Algol 的完全抽象博弈语义。载于 Jean-Yves Girard、Mitsuhiro Okada 和 Andre Scedrov 编，《1996年东京线性逻辑会议》，庆应义塾大学三田校区，日本东京，1996年3月29日至4月2日，《理论计算机科学电子札记》，第2–14页。爱思唯尔，1996年。网址：https://doi.org/10.1016/S1571-0661(05)80398-6, doi:10.1016/S1571-0661(05)80398-6。3 Dariusz Biernacki、Sergueï Lenglet 和 Piotr Polesiuk。状态的完备范式互模拟。载于 Mikolaj Bojanczyk 和 Alex Simpson 编，《软件科学与计算结构基础——第22届国际会议，FOSSACS 2019》，作为欧洲软件理论与实践联合会议 ETAPS 2019 的一部分举行，捷克共和国布拉格，2019年4月6日至11日，论文集，《计算机科学讲义》，第98–114页。施普林格，2019年。网址：https://doi.org/10.1007/978-3-030-17127-8_6, doi:10.1007/978-3-030-17127-8_6。4 Lars Birkedal、Filip Sieczkowski 和 Jacob Thamsborg。并发逻辑关系。载于 Patrick Cégielski 和 Arnaud Durand 编，《计算机科学逻辑——第26届国际研讨会／欧洲计算机科学逻辑协会第21届年会，CSL 2012》，法国枫丹白露，2012年9月3日至6日，《LIPIcs》第16卷，第107–121页。Schloss Dagstuhl - Leibniz 信息学中心，2012年。网址：https://doi.org/10.4230/LIPIcs.CSL.2012.107,doi:10.4230/LIPICS.CSL.2012.107。5 Peio Borthelle、Tom Hirschowitz、Guilhem Jaber 和 Yannick Zakowski。操作博弈语义的抽象化、经认证的论述。载于 Viktor Vafeiadis 编，《编程语言与系统——第34届欧洲编程研讨会，ESOP 2025》，作为软件理论与实践国际联合会议 ETAPS 2025 的一部分举行，加拿大安大略省汉密尔顿，2025年5月3日至8日，论文集，第一部分，《计算机科学讲义》，第172–199页。施普林格，2025年。网址：https://doi.org/10.1007/978-3-031-91118-7_7, doi:10.1007/978-3-031-91118-7_7。6 Simon Castellan 和 Pierre Clairambault。厘清博弈语义中的并行性与干扰。《逻辑方法与计算机科学》，20(3)，2024年。网址：https://doi.org/10.46298/lmcs-20(3:24)2024, doi:10.46298/LMCS-20(3:24)2024。7 Simon Castellan、Pierre Clairambault 和 Glynn Winskel。具有对称性的薄博弈与并发 Hyland-Ong 博弈。《逻辑方法与计算机科学》，15(1)，2019年。网址：https://doi.org/10.23638/LMCS-15(1:18)2019, doi:10.23638/LMCS-15(1:18)2019。8 Derek Dreyer、Georg Neis 和 Lars Birkedal。高阶状态和控制效应对局部关系推理的影响。《函数式编程期刊》，22(4-5):477–528，2012年。网址：https://doi.org/10.1017/S095679681200024X, doi:10.1017/S095679681200024X。9 Dan R. Ghica 和 Andrzej S. Murawski。细粒度并发的天使语义。《纯粹与应用逻辑年刊》，151(2-3):89–114，2008年。网址：https://doi.org/10.1016/j.apal.2007.10.005, doi:10.1016/J.APAL.2007.10.005。10 Daniel Hirschkoff、Iwan Quémerais 和 Davide Sangiorgi。名称传递演算中的一阶存储与可见性。载于 Patricia Bouyer 和 Jaco van de Pol 编，《第36届并发理论国际会议，CONCUR 2025》，丹麦奥胡斯，2025年8月26日至29日，《LIPIcs》，第23:1–23:21页。Schloss Dagstuhl - Leibniz 信息学中心，2025年。网址：https://doi.org/10.4230/LIPIcs.CONCUR.2025.23, doi:10.4230/LIPICS.CONCUR.2025.23。11 Chung-Kil Hur、Derek Dreyer、Georg Neis 和 Viktor Vafeiadis。互模拟与 Kripke 逻辑关系的结合。载于 John Field 和 Michael Hicks 编，《第39届美国计算机协会 SIGPLAN-SIGACT 编程语言原理研讨会论文集，POPL

2012年，费城，宾夕法尼亚州，美国，2012年1月22日至28日，页码 59–72。美国计算机协会，2012年。URL:https://doi.org/10.1145/2103656.2103666, doi:10.1145/2103656.2103666. 12 J. M. E. Hyland 和 C.-H. Luke Ong。《PCF 的完全抽象性：i、ii 和 III》。《信息与计算》，163(2):285–408，2000年。URL: https://doi.org/10.1006/inco.2000.2917, doi:10.1006/INCO.2000.2917. 13 Guilhem Jaber。《操作名义博弈语义》。载于 Andrew M. Pitts（编），《软件科学与计算结构基础——第18届国际会议 FoSSaCS 2015》，作为欧洲软件理论与实践联合会议 ETAPS 2015 的一部分举行，英国伦敦，2015年4月11日至18日。会议论文集，《计算机科学讲义》，页码 264–278。施普林格，2015年。URL: https://doi.org/10.1007/978-3-662-46678-0_17, doi:10.1007/978-3-662-46678-0_17. 14 Guilhem Jaber 和 Andrzej S. Murawski。《状态与控制的完全迹模型》。载于 Nobuko Yoshida（编），《编程语言与系统——第30届欧洲编程研讨会 ESOP 2021》，作为欧洲软件理论与实践联合会议 ETAPS 2021 的一部分举行，卢森堡卢森堡市，2021年3月27日至4月1日。会议论文集，《计算机科学讲义》第12648卷，页码 348–374。施普林格，2021年。URL: https://doi.org/10.1007/978-3-030-72019-3_13, doi:10.1007/978-3-030-72019-3_13. 15 Guilhem Jaber 和 Andrzej S. Murawski。《通过操作博弈语义进行组合式关系推理》。载于第36届 ACM/IEEE 计算机科学逻辑年度研讨会 LICS 2021，意大利罗马，2021年6月29日至7月2日，页码 1–13。电气电子工程师学会，2021年。URL: https://doi.org/10.1109/LICS52264.2021.9470524, doi:10.1109/LICS52264.2021.9470524. 16 Guilhem Jaber 和 Davide Sangiorgi。《博弈、移动进程与函数》。载于 Florin Manea 和 Alex Simpson（编），第30届 EACSL 计算机科学逻辑年会 CSL 2022，德国哥廷根（线上会议），2022年2月14日至19日，LIPIcs，页码 25:1–25:18。达格斯图尔城堡—莱布尼茨信息学中心，2022年。URL: https://doi.org/10.4230/LIPIcs.CSL.2022.25, doi:10.4230/LIPICS.CSL.2022.25. 17 Guilhem Jaber 和 Nicolas Tabareau。《Kripke 开放双模拟——博弈语义与操作技术的结合》。载于 Xinyu Feng 和 Sungwoo Park（编），《编程语言与系统——第13届亚洲研讨会 APLAS 2015》，韩国浦项，2015年11月30日至12月2日。会议论文集，《计算机科学讲义》，页码 271–291。施普林格，2015年。URL: https://doi.org/10.1007/978-3-319-26529-2_15, doi:10.1007/978-3-319-26529-2_15. 18 Alan Jeffrey 和 Julian Rathke。《具有局部名称的并发 ML 片段的双模拟理论》。《理论计算机科学》，323(1-3):1–48，2004年。URL: https://doi.org/10.1016/j.tcs.2004.03.005, doi:10.1016/J.TCS.2004.03.005. 19 Alan Jeffrey 和 Julian Rathke。《并发对象的完全抽象可能测试语义》。《理论计算机科学》，338(1-3):17–63，2005年。URL: https://doi.org/10.1016/j.tcs.2004.10.012, doi:10.1016/J.TCS.2004.10.012. 20 Vasileios Koutavas 和 Mitchell Wand。《用于推理高阶命令式程序的小型双模拟》。载于 J. Gregory Morrisett 和 Simon L. Peyton Jones（编），第33届 ACM SIGPLAN-SIGACT 编程语言原理研讨会 POPL 2006 论文集，美国南卡罗来纳州查尔斯顿，2006年1月11日至13日，页码 141–152。美国计算机协会，2006年。URL: https://doi.org/10.1145/1111037.1111050, doi:10.1145/1111037.1111050. 21 James Laird。《带控制的函数式语言的完全抽象性》。载于第12届 IEEE 计算机科学逻辑年会论文集，波兰华沙，1997年6月29日至7月2日，页码 58–67。IEEE 计算机学会，1997年。URL: https://doi.org/10.1109/LICS.1997.614931,doi:10.1109/LICS.1997.614931. 22 James Laird。《理想化 CSP 的博弈语义》。载于 Stephen D. Brookes 和 Mi- chael W. Mislove（编），第17届编程语义数学基础会议 MFPS 2001，丹麦奥胡斯，2001年5月23日至26日，《理论计算机科学电子札记》，

《理论计算机科学札记》，页码 232–257。爱思唯尔，2001年。URL: https://doi.org/10.1016/S1571-0661(04)80965-4, doi:10.1016/S1571-0661(04)80965-4. 23 James Laird。《一般引用的完全抽象迹语义》。载于 Lars Arge、Christian Cachin、Tomasz Jurdzinski 和 Andrzej Tarlecki（编），《自动机、语言与编程：第34届国际研讨会 ICALP 2007》，波兰弗罗茨瓦夫，2007年7月9日至13日。会议论文集，《计算机科学讲义》，页码 667–679。施普林格，2007年。URL: https://doi.org/10.1007/978-3-540-73420-8_58, doi:10.1007/978-3-540-73420-8_58. 24 James Laird。《交互的柯里风格语义：从无类型到二阶惰性 $\lambda\mu$-演算》。载于 Jean Goubault-Larrecq 和 Barbara König（编），《软件科学与计算结构基础——第23届国际会议 FOSSACS 2020》，作为欧洲软件理论与实践联合会议 ETAPS 2020 的一部分举行，爱尔兰都柏林，2020年4月25日至30日。会议论文集，《计算机科学讲义》，页码 422–441。施普林格，2020年。URL: https://doi.org/10.1007/978-3-030-45231-5_22, doi:10.1007/978-3-030-45231-5_22. 25 Søren B. Lassen 和 Paul Blain Levy。《参数多态的类型化范式双模拟》。载于第二十三届 IEEE 计算机科学逻辑年度研讨会 LICS 2008 论文集，美国宾夕法尼亚州匹兹堡，2008年6月24日至27日，页码 341–352。IEEE 计算机学会，2008年。URL: https://doi.org/10.1109/LICS.2008.26, doi:10.1109/LICS.2008.26. 26 Sergueï Lenglet、Alan Schmitt 和 Jean-Bernard Stefani。《刻画带钝化演算中的上下文等价》。《信息与计算》，209(11):1390–1433，2011年。URL: https://doi.org/10.1016/j.ic.2011.08.002, doi:10.1016/J.IC.2011.08.002. 27 Ian Mason 和 Carolyn Talcott。《带效应的函数式语言中的等价性》。《函数式编程杂志》，1(3):287–327，1991年。 doi:10.1017/S0956796800000125. 28 Andrzej S. Murawski 和 Nikos Tzevelekos。《良好的一般引用的博弈语义》。载于第26届 IEEE 计算机科学逻辑年度研讨会 LICS 2011 论文集，加拿大安大略省多伦多，2011年6月21日至24日，页码 75–84。IEEE 计算机学会，2011年。URL: https://doi.org/10.1109/LICS.2011.31, doi:10.1109/LICS.2011.31. 29 Andrzej S. Murawski 和 Nikos Tzevelekos。《完整基类型引用的算法博弈》。载于 Artur Czumaj、Kurt Mehlhorn、Andrew M. Pitts 和 Roger Wattenhofer（编），《自动机、语言与编程——第39届国际研讨会 ICALP 2012》，英国华威，2012年7月9日至13日。会议论文集，第二部分，《计算机科学讲义》第7392卷，页码 312–324。施普林格，2012年。URL: https://doi.org/10.1007/978-3-642-31585-5_30,doi:10.1007/978-3-642-31585-5_30. 30 Amr Sabry 和 Matthias Felleisen。《以延续传递风格推理程序》。载于 Jon L. White（编），Lisp与函数式编程会议 LFP 1992 论文集，美国加利福尼亚州旧金山，1992年6月22日至24日，页码 288–298。美国计算机协会，1992年。URL: https://doi.org/10.1145/141471.141563, doi:10.1145/141471.141563. 31 D. Sangiorgi。《进程代数中的移动性表达：一阶与高阶范式》。博士论文 CST–99–93，爱丁堡大学计算机科学系，1992年。32 Davide Sangiorgi、Naoki Kobayashi 和 Eijiro Sumii。《高阶语言的环境双模拟》。《ACM 编程语言与系统汇刊》，33(1):5:1–5:69，2011年。URL:https://doi.org/10.1145/1889997.1890002, doi:10.1145/1889997.1890002.

主要记号摘要

记号	含义	定义
$\mathtt{M}, \mathtt{N}$	项	第 2 节
σ, τ	类型	第 2 节
Γ, Σ	类型上下文和存储类型上下文	第 2 节
$\mathtt{S}, \mathtt{S}[\ell \mapsto \mathtt{V}], \mathtt{S} \cdot \mathtt{T}$	存储、存储更新与存储的不相交拼接	第 2 节
$\mapsto_{\mathsf{red}}$	带存储项的归约语义	图1
$\preceq_{ciu}, \simeq_{ciu}$	CIU 近似与 CIU 等价	定义 3
$\{\mathtt{E}\}_{\mathtt{p}}, \mathtt{M}$	暂停上下文与部分暂停项	第 3.1 节
$\overset{\mu}{\mapsto}_{\mathsf{env}}, L_{\mathsf{env}}$	与环境交互的归约和标记转换系统	图2
$\mathtt{A}$	抽象值	第 3.1 节
$\mathtt{m}, \mathtt{f}_{(\mathtt{A})}(\mathtt{A}, \mathtt{p}), \mathtt{p}_{(\mathtt{A})}(\mathtt{A}), \mathtt{f}(\mathtt{A}, \mathtt{p}), \mathtt{p}$	步	第 3.1 节
γ	替换	第 3.1 节
$\mathtt{P}, \mathtt{Q}, [\mathtt{p}]\mathtt{M}$	线程池	第 3.1 节
$\mathbb{I}, \mathbb{J}, \langle \mathtt{P}; \mathtt{S}; \gamma \rangle$	交互式配置	第 3.1 节
$\xrightarrow{\mathtt{m}}_{\mathsf{I}}, L_{\mathsf{I}}$	交互归约和标记转换系统	图4
$\mathbb{S}, \mathbb{T}, \langle \Delta_{\mathsf{O}} \mid \Delta_{\mathsf{P}} \rangle$	类型配置	第 3.1 节
$\rightarrow_{\mathsf{Ty}} L_{\mathsf{Ty}}$	类型归约和标记转换系统	图5
$\mathbb{V}, \mathbb{W}, \langle F;\ G;\ T \rangle$	可见配置	第 3.1 节
$\rightarrow_{\mathsf{vis}} L_{\mathsf{vis}}$	可见归约和标记转换系统	图 6
$\mathbb{G}, \mathbb{H}, (\mathbb{I}, \mathbb{S}, \mathbb{V})$	OGS 配置	第 3.1 节
$\rightarrow_{\mathsf{ogs}}, L_{\mathsf{OGS}}$	OGS 归约和标记转换系统	第 3.1 节
$?(\overrightarrow{\mathtt{A}_i}, \mathtt{p}), !(\overrightarrow{\mathtt{A}_i}, \mathtt{p})$	初始 OGS 步	图 7
$\preceq_{ctx}, \simeq_{ctx}$	上下文预序和上下文等价	第 2 节
$\preceq_{\mathsf{tr}}, \simeq_{\mathsf{tr}}$	迹逼近与迹等价	第 3.2 节
$\|, \mapsto^{\mathsf{red}}_{\mathsf{so}}, \mapsto^{\mathsf{sync}}_{\mathsf{so}}$	并行组合、操作归约与同步	定义 11
$\between$	通过配置合并进行组合	第 4 节
$\preceq$	迹交换的预序	第 5 节

B 目标语言和操作语义

B.1 类型判断

类型规则如图 8 所示

C 操作博弈语义

▶ 定义 *21. OGS* 配置 $(\mathbb{I}, \mathbb{S}, \mathbb{V})$，其中：$\mathbb{I} = \langle [p_1]\mathtt{M}_1 | \cdots | [p_k]\mathtt{M}_k; \mathtt{S}; \gamma \rangle$; $\mathbb{S} = \langle \Delta_{\mathsf{O}} \vdash \Delta_{\mathsf{P}} \rangle$; $\mathbb{V} = \langle F; G; T \rangle$; 当满足以下条件时称其为有效：存在存储类型上下文 Σ，使得：$\Sigma; \Delta_{\mathsf{O}} \vdash \gamma : \Delta_{\mathsf{P}}$; $\vdash \mathtt{S} : \Sigma$; 对所有 $i \in \{1, \ldots, k\}$，$\Sigma; \Delta_{\mathsf{O}} \vdash \mathtt{M}_i : \sigma_i$，且 $\Delta_{\mathsf{O}}(p_i) = \neg\sigma_i$。$dom(F) = dom(\Delta_{\mathsf{O}})$ 且 $im(F) \subseteq P_{\mathrm{fin}}(dom(\Delta_{\mathsf{P}}))$;

$\boxed{\Sigma;\Gamma \vdash \mathtt{M} : \sigma}$

$$\textsc{Var}\ \frac{\Gamma(x) = \sigma}{\Sigma;\Gamma \vdash x : \sigma} \qquad \textsc{Loc}\ \frac{\Sigma(\ell) = \sigma}{\Sigma;\Gamma \vdash \ell : \textbf{引用}_\sigma} \qquad \textsc{Unit}\ \frac{}{\Sigma;\Gamma \vdash () : \mathtt{Unit}} \qquad \textsc{Bool}\ \frac{}{\Sigma;\Gamma \vdash \mathtt{b} : \mathtt{Bool}} \qquad \textsc{Int}\ \frac{}{\Sigma;\Gamma \vdash \mathtt{k} : \mathtt{Int}}$$

$$\textsc{Pair}\ \frac{\Sigma;\Gamma \vdash \mathtt{V} : \sigma \qquad \Sigma;\Gamma \vdash \mathtt{W} : \tau}{\Sigma;\Gamma \vdash \langle \mathtt{V}, \mathtt{W} \rangle : \sigma \times \tau} \qquad \textsc{Proj}\ \frac{\Sigma;\Gamma \vdash \mathtt{V} : \sigma_1 \times \sigma_2 \qquad i \in \{1,2\}}{\Sigma;\Gamma \vdash \mathrm{proj}_i\, \mathtt{V} : \sigma_i}$$

$$\textsc{Abs}\ \frac{\Sigma;\ \Gamma,\ x : \sigma \vdash \mathtt{M} : \tau}{\Sigma;\Gamma \vdash \lambda x.\mathtt{M} : \sigma \to \tau} \qquad \textsc{App}\ \frac{\Sigma;\Gamma \vdash \mathtt{V} : \sigma \to \tau \qquad \Sigma;\Gamma \vdash \mathtt{M} : \sigma}{\Sigma;\Gamma \vdash \mathtt{V}\,\mathtt{M} : \tau} \qquad \textsc{Rec}\ \frac{\Sigma;\ \Gamma,\ f : \sigma \to \tau,\ x : \sigma \vdash \mathtt{M} : \tau}{\Sigma;\ \Gamma \vdash \mathtt{rec}\, f(x).\mathtt{M} : \tau}$$

$$\textsc{Ite}\ \frac{\Sigma;\Gamma \vdash \mathtt{V} : \mathtt{Bool} \qquad \Sigma;\Gamma \vdash \mathtt{M} : \sigma \qquad \Sigma;\Gamma \vdash \mathtt{N} : \sigma}{\Sigma;\Gamma \vdash \mathtt{if\ V\ then\ M\ else\ N} : \sigma}$$

$$\textsc{New}\ \frac{\Sigma;\Gamma \vdash \mathtt{V} : \sigma}{\Sigma;\ \Gamma \vdash \mathtt{ref\ V} : \textbf{引用}_\sigma} \qquad \text{解引用}\ \frac{\Sigma;\ \Gamma \vdash \mathtt{V} : \textbf{引用}_\sigma}{\Sigma;\Gamma \vdash !\mathtt{V} : \sigma} \qquad \text{赋值}\ \frac{\Sigma;\ \Gamma \vdash \mathtt{V} : \textbf{引用}_\sigma \qquad \Sigma;\Gamma \vdash \mathtt{W} : \sigma}{\Sigma;\Gamma \vdash \mathtt{V} := \mathtt{W} : \mathtt{Unit}}$$

$$\textsc{Par}\ \frac{\Sigma;\Gamma \vdash \mathtt{M} : \sigma \qquad \Sigma;\ \Gamma \vdash \mathtt{N} : \tau}{\Sigma;\ \Gamma \vdash \text{并行 } \mathtt{M} \text{ 和 } \mathtt{N} : \sigma \times \tau} \qquad \textsc{Cas}\ \frac{\Sigma;\ \Gamma \vdash \mathtt{W} : \textbf{引用}_\sigma \qquad \Sigma;\ \Gamma \vdash \mathtt{V}_1 : \sigma \qquad \Sigma;\ \Gamma \vdash \mathtt{V}_2 : \sigma}{\Sigma;\ \Gamma \vdash \mathtt{CAS\ W\ V}_1\ \mathtt{V}_2 : \mathtt{Bool}}$$

$\boxed{\Sigma;\Gamma \vdash \mathtt{D} : \tau \rightsquigarrow \sigma}$

$$\textsc{Hole}\ \frac{}{\Sigma;\Gamma \vdash \bullet : \sigma \rightsquigarrow \sigma} \qquad \frac{\Sigma;\Gamma \vdash \mathtt{D} : \tau \rightsquigarrow \sigma \qquad \Sigma;\Gamma \vdash \mathtt{V} : \sigma \to \sigma'}{\Sigma;\Gamma \vdash \mathtt{V\,D} : \tau \rightsquigarrow \sigma'}$$

$$\frac{\Sigma;\Gamma \vdash \mathtt{D} : \tau \rightsquigarrow \sigma_1 \qquad \Sigma;\ \Gamma \vdash \mathtt{M} : \sigma_2}{\Sigma;\ \Gamma \vdash \text{并行 } \mathtt{D} \text{ 和 } \mathtt{M} : \tau \rightsquigarrow \sigma_1 \times \sigma_2} \qquad \frac{\Sigma;\ \Gamma \vdash \mathtt{M} : \sigma_1 \qquad \Sigma;\ \Gamma \vdash \mathtt{D} : \tau \rightsquigarrow \sigma_2}{\Sigma;\ \Gamma \vdash \text{并行 } \mathtt{M} \text{ 和 } \mathtt{D} : \tau \rightsquigarrow \sigma_1 \times \sigma_2}$$

图 8 类型规则。

- $dom(G) \subseteq dom(\Delta_P)$ 且 $im(G) \subseteq P_{\text{fin}}(dom(\Delta_P))$；树 T 是交替的，
- 也就是说，对于所有 $p \in T$：要么 p 是根，且 $p \in dom(\Delta_O)$；要
- 么 $p \in dom(\Delta_O)$ 且 $q \in dom(\Delta_P)$，其中 q 是 p 的父节点；要么
- $p \in dom(\Delta_P)$ 且 $q \in dom(\Delta_O)$，其中 q 是 p 的父节点；T 中属
- 于 Δ_O 的节点集合等于 $\{p1, \ldots, p_k\}$；$\{\mathtt{E}\}^{p_j}$ 出现在 $\mathtt{M}_i$ 中，当
- 且仅当 p_j 是 p_i 的子节点。

▶ **引理 22**。 取一个有效的 OGS 配置，使得 $\mathbb{G} \xrightarrow{\mathtt{a}}_{\text{ogs}} \mathbb{H}$，则 $\mathbb{G}$ 有效。