arXiv:2602.21630v1 [cs.PL] 25 Feb 2026

编舞式编程中非干扰的基于类型强制执行

Marco Bertoni[1], Saverio Giallorenzo[2,3], 和 Marco Peressotti[4]

[1] Inria Paris，法国[2] 博洛尼亚大学，意大利[3] Inria Sophia Antipolis，法国[4] FORM，南丹麦大学，丹麦

摘要 编排从全局视角描述分布式协议，从而能够构造即正确地综合局部行为。我们开发了一种策略参数化类型系统，通过程序计数器规约同时处理显式流与隐式流，防止高安全级别数据向 low安全级别观察者发生信息泄漏。该系统借助我们通过约束生成重构的过程上下文来支持递归过程。我们相对于标准小步语义证明了终止不敏感非干扰。

关键词： 信息流 · 非干扰 · 安全性。

1 引言

分布式软件通过通信组件实现协议——用户身份验证、支付对账等。分布式软件中有两个至关重要的问题：参与者之间的正确协调 以及所处理数据的机密性 。编舞式编程语言 [32] 从全局视角描述协议来解决协调问题，由此可以推导出合规分布式软件。本工作解决第二个问题：确保编排不会向公共观察者泄露秘密信息。

确保机密性的挑战出现在控制流依赖于敏感数据的现实协议中。我们用一个简单的密码恢复场景来说明这一概念。请求者 r 向服务 s 提交电子邮件地址，s 随后可能指派邮件发送服务 m 发送重置链接。尽管该协议只涉及几个简单步骤，但即使看似无害的设计选择也可能无意中泄露敏感信息。该电子邮件是否对应已注册账户属于敏感信息：泄露它会使用户名枚举攻击成为可能。因此，我们必须将 s 对 exists(email) 检查的结果视为秘密，而 r 可观察到的行为则被视为公开的。

让我们通过该场景中的一个不安全编舞来直观地看一下这个问题。在编舞中，a.e -> b.x 表示 a 将表达式 e 的求值结果发送给 b，后者将收到的值存储到其局部变量 x 中。

不安全编舞示例

r.email -> s.email
如果 s.exists(email)，则
  s.email ->m.email; s."邮件已发送" ->r.msg
否则 s."unknown user" -> r.msg

尽管 exists(email) 返回的布尔值从未直接发送给 r，请求者仍能通过观察不同的响应消息来区分不同的执行。因此，r 获知了敏感信息位：该账户是否存在。信息泄露并非源于秘密数据的显式传输，而是源于会产生可观察差异的控制流依赖。

我们可以通过确保对 exists(email) 的分支只会影响与邮件发送器的内部通信，同时让 r 处可观察到的行为保持一致，从而消除信息泄露。

安全的 `r.email->s.email`编排 `if s.exists(email)then s.email -> m.email else skip`变体 `s."check yourinbox" -> r.msg`

在安全变体中，仅秘密谓词不同的执行在 r 处产生完全相同的观察结果。此示例说明了信息的隐式信息流，即秘密会影响发生哪些操作；与之相对的是显式信息流，其中秘密值会被直接赋值或传输到公共位置。

为了系统地检测和防止此类泄露，我们需要对信息如何在编排中流动（包括显式和隐式流动）作出形式化描述。为此，遵循 Denning [21], 的方法，我们为每个数据分配一个取自格的安全级别，其序关系 $\sqsubseteq$ 表示相对敏感度。结合级别分配，格结构定义了信息流策略，规定信息何时可以在敏感度不同的位置之间流动：标记为 ℓ_1 的数据只有在 $\ell_1 \sqsubseteq \ell_2$ 时才能流向标记为 ℓ_2 的上下文。

在这项工作中，我们关注非干扰 [23], ，它要求高安全级输入的变化不得影响低安全级输出 [41]。直观而言，安全数据不应影响外部观察者从程序执行中能够获知的信息，也就是说，不应干扰观察到的行为——而这正是我们首次尝试实现密码恢复时所出现的违规情况。

我们认为，为编排式编程解决非干扰问题具有多项优势。首先，进程之间的所有通信都会在编排中显式呈现，因此潜在的信息流通道在语法层面清晰可见——在我们的示例中，从 s 到 r 的消息立即揭示了可能的信息泄露。其次，我们通过一次全局判断验证整个协议的安全性，而不是组合针对各个参与者的独立保证。第三，编排既描述局部控制流，也描述进程间协调，因此我们能够检测由某个参与者的选择影响其他参与者可观察行为而产生的隐式流。最后，一旦证明某个编排满足非干扰，端点投影便能通过构造保留这一性质 [18]。由此，我们从经过验证的全局规范到生成的分布式实现，获得端到端保证。这种全局推理、显式通信结构与正确性保持编译的结合，

使编排式编程成为在分布式系统中强制实现非干扰的天然基础。

我们的方法。 我们针对过程式编排编程的参考语言，开发了一种非干扰的基于类型的强制执行 [32]。我们的类型系统是策略参数化的：给定用户指定的、定义在格 $\mathscr{L}$ 上的信息流策略以及一个观察级别 low，该类型系统会拒绝那些高安全级别数据可能影响 low 可观察行为的编排。

我们的类型系统以如下形式的判断为核心： $\Delta; \Gamma; pc \vdash C$，其中 Γ 为变量赋予安全级别， $pc \in \mathscr{L}$ 跟踪当前控制级别（为捕获隐式信息流提供上下文），而 Δ 则规定过程如何在不同控制级别下安全调用。此定义将标准的信息流类型系统推广到了编排场景；在这一场景中，信息既通过进程间的消息传递流动，也通过全局协议中的控制依赖流动。

我们依靠两个基本的技术手段来获得这些结果，分别由 pc 和 Δ 体现。首先，我们采用标准的程序计数器规约：控制级别 pc 记录秘密数据对当前执行点的影响。当程序根据秘密信息进行分支判断（例如我们示例中的 exists(email)）时， pc 会升高，后续操作也会在这一较高的上下文中进行检查。这一机制可防止 low 可观察行为依赖于高安全级别的控制流，从而排除我们第一个密码恢复示例中的隐式信息流。其次，为了支持递归场景下的模块化验证，我们引入过程上下文 Δ，为每个过程和控制级别指定一组可接受的信息流策略。直观而言，Δ 充当一份契约： $\Delta; \Gamma; pc \vdash X(\bar{p})$ 确保在 Γ 下、控制级别为 pc 时调用过程 $X(\bar{p})$ 能够保持非干扰。我们开发了一种基于约束的重建过程，它可以根据过程定义本身计算出 Δ，从而自动完成这一步验证。

我们的执法目标是针对编排的标准小步语义。遵循基于语言的信息流安全领域的既有做法 [41]，我们采用对终止不敏感 的攻击者模型，其中发散不被视为可观察行为。尽管存在更强的概念，这一选择在表达能力（允许程序的运行时依赖于秘密信息）和安全保证之间取得了务实的平衡。

我们的主要结果（定理1）确立了类型系统对于对终止不敏感的非干扰的可靠性。

论文结构。 第2节介绍参考编舞语言及其小步操作语义。第 3 节介绍适用于参考编舞语言的策略参数化信息流类型系统，该系统追踪显式和隐式信息流，并支持在给定有效过程上下文 Δ 的情况下定义递归过程。第4节定义并证明一个可靠性定理，确立类型正确的编排满足终止

-

相对于语言语义的不敏感非干扰。在第 5 节中，我们定义并证明一种基于约束的重构算法的正确性。该算法给定一组过程定义，计算一个上下文 Δ，在该上下文下

这些过程均为类型良构——这正是预期的正确性标准。我们分别在第 6 节和第 7 节讨论相关工作并作总结。

2 编排基础知识

我们回顾编排编程语言的基本概念 [32] ，并介绍一种供用户编写分布式程序的语言，称为递归编排。编排以形式化方式描述并发与分布式系统中进程预期的协作行为，充当协议，规定组件如何交互以实现共同目标。尽管编排的书写方式让人联想到传统的本地程序，但它们通过表示分布式通信与计算步骤的转换来执行。编排表示所有参与者的全局状态，并涵盖共同实现协议的各项操作。

2.1 语法

进程是编排中的参与者，执行本地计算并与其他参与者通信。我们用 p、q、r、$s \in$ 表示进程名称。我们通过以下上下文无关文法定义递归编排：

$\mathscr{C} ::= \{X_i(\tilde{p}_i) = C_i\}_{i \in I}$ 编排过程定义 $C ::= I$； C 指令顺序执行 $\mid \mathbf{0}$ 以终止的编排 $I ::= p.e \to q.x$ p 将 e 的结果发送给 q，并将其赋给 x $\mid p \to q[L]$ p 将常量 L 发送给 $q \mid p.x$: 将 e 的结果赋给 $x \mid$ 如果 $p.e$，则执行 C_1，否则执行 C_2 p e 求值并选择继续执行 C_1 或 $C_2 \mid X(\tilde{p})$ 调用过程 $X \mid r : X(\tilde{p})$（运行时项，r 尚未进入调用）$e ::= v \mid x \mid f(\tilde{e})$ 本地值、变量和函数调用

此处，$\mathscr{C}$ 表示过程定义的上下文；C 表示编排——可以是已终止的 $\mathbf{0}$，也可以是由指令 I 和后续部分 C 顺序组合而成的编排。I 表示指令；从上到下依次为通信、选择、局部赋值、条件语句和过程调用。e 表示局部表达式（常量 $v \in$ Val、变量 $x \in$ Var、函数调用 $f(\tilde{e})$）。

2.2 语义

我们使用形成标记转移系统的小步操作语义 [35], 来定义递归编排的语义。配置的形式为 $\langle C, \Sigma, \mathscr{C} \rangle$，其中 C 是编排，$\mathscr{C}$ 是过程上下文，Σ 是编排存储。

一个进程存储 σ : Var $\to$ Val 对一个进程的内存进行建模。我们将所有进程存储的集合称为 PStore。一个编排存储 Σ : PName $\to$ PStore 对整个

$$\textsc{local}\quad \frac{\Sigma(p) \vdash e \downarrow v}{\langle p.x := e; C, \Sigma, \mathscr{C}\rangle \xrightarrow{\tau@p} \langle C, \Sigma[p.x \mapsto v], \mathscr{C}\rangle}$$

$$\textsc{com}\quad \frac{\Sigma(p) \vdash e \downarrow v}{\langle p.e \to q.x; C, \Sigma, \mathscr{C}\rangle \xrightarrow{p.v \to q} \langle C, \Sigma[q.x \mapsto v], \mathscr{C}\rangle} \qquad \textsc{sel}\quad \frac{}{\langle p \to q[L]; C, \Sigma, \mathscr{C}\rangle \xrightarrow{p \to q[L]} \langle C, \Sigma, \mathscr{C}\rangle}$$

$$\textsc{cond-then}\quad \frac{\Sigma(p) \vdash e \downarrow \mathit{true}}{\langle \text{if } p.e \text{ then } C_1 \text{ else } C_2; C, \Sigma, \mathscr{C}\rangle \xrightarrow{p.\text{then}} \langle C_1 \fatsemi C, \Sigma, \mathscr{C}\rangle}$$

$$\textsc{cond-else}\quad \frac{\Sigma(p) \vdash e \downarrow v \qquad v \neq \mathit{true}}{\langle \text{if } p.e \text{ then } C_1 \text{ else } C_2; C, \Sigma, \mathscr{C}\rangle \xrightarrow{p.\text{else}} \langle C_2 \fatsemi C, \Sigma, \mathscr{C}\rangle}$$

$$\textsc{call-first}\quad \frac{X(\bar{q}) = C \in \mathscr{C} \qquad \{p_0, \ldots, p_n\} = \{r_0, r_1, \ldots, r_n\}}{\langle X(\bar{p}); C', \Sigma, \mathscr{C}\rangle \xrightarrow{\tau@r_0} \langle r_1 : X(\bar{p}).C'; \ldots; r_n : X(\bar{p}).C'; C[\bar{q}/\bar{p}] \fatsemi C', \Sigma, \mathscr{C}\rangle}$$

$$\textsc{call-enter}\quad \frac{}{\langle q : X(\bar{p}).C'; C, \Sigma, \mathscr{C}\rangle \xrightarrow{\tau@q} \langle C, \Sigma, \mathscr{C}\rangle}$$

$$\textsc{delay}\quad \frac{\langle C, \Sigma, \mathscr{C}\rangle \xrightarrow{\mu} \langle C', \Sigma', \mathscr{C}\rangle \qquad \mathrm{pn}(I) \cap \mathrm{pn}(\mu) = \emptyset}{\langle I; C, \Sigma, \mathscr{C}\rangle \xrightarrow{\mu} \langle I; C', \Sigma', \mathscr{C}\rangle}$$

$$\textsc{delay-cond}\quad \frac{\langle C_1, \Sigma, \mathscr{C}\rangle \xrightarrow{\mu} \langle C_1', \Sigma', \mathscr{C}\rangle \qquad \langle C_2, \Sigma, \mathscr{C}\rangle \xrightarrow{\mu} \langle C_2', \Sigma', \mathscr{C}\rangle \qquad p \notin \mathrm{pn}(\mu)}{\langle \text{if } p.e \text{ then } C_1 \text{ else } C_2; C, \Sigma, \mathscr{C}\rangle \xrightarrow{\mu} \langle \text{if } p.e \text{ then } C_1' \text{ else } C_2'; C, \Sigma', \mathscr{C}\rangle}$$

图1. 操作语义。

系统内存进行建模。我们将 $\Sigma[p.x \mapsto v]$ 写作用于更新存储 Σ，使进程 p 的局部变量 x 映射到 v。我们将所有编排存储的集合称为 CStore。

给定进程存储 σ、表达式 e 和值 v，当 e 在 σ 下求值为 v 时，我们记作 $\sigma \vdash e \downarrow v$。我们并不规定用于推导形如 $\vdash f(\bar{v}) \downarrow v$ 这类命题的系统，因为这对我们的论述并不重要：该系统将取决于函数如何定义，而我们选择对此加以抽象。相反，我们仅假设存在这样的系统，并且对于任意 f 和 $\bar{v}$，总可以针对某个 $\bar{v}$ 推导出 $\vdash f(\bar{v})$。

转换标签记录所执行步骤的类型——在 p ($\tau@p$) 处的内部步骤、通信 $p.v \to q$、选择 ($p \to q[L]$)，或在 p (p.then/p.else) 处的条件语句结果。

我们在图 1 中给出了递归编排的语义规则。com规则描述运行时计算出的值的通信：

p 根据其本地存储对表达式 e 求值，并将结果 v 发送给q ；后者将该结果以变量 x 为名存入其本地存储。类似地，sel规则描述常量 L（称为选择）的通信。[5] cond-then 和 cond-else 规则描述条件分支：进程 p 在本地对守卫条件 e 求值，编排随后继续执行相应的分支；算子 $\S$通过将后续部分 C 接入 C_1 和 C_2中任意的 **0** 位置，实现顺序组合（$C; C'$ 不属于文法）。call-first 和 call-enter 规则描述对编排过程的去中心化调用，即进入该过程的各进程之间无需同步。这些规则借助运行时项 $r : X(\bar{p}).C'$ 进行跟踪，表示进程 r 尚未进入调用，因此无法通过延迟规则执行其后续部分中的任何动作。call-first 规则从上下文 $\mathscr{C}$ 展开过程定义，并在过程定义体 C中用实际参数 $\bar{p}$ 实例化形式过程参数 $\bar{q}$ 。由于前提条件 $\{p_0, \ldots, p_n\} = \{r_0, r_1, \ldots, r_n\}$ 允许任意地重新索引 $\bar{p}$，因此哪个进程先进入调用、哪个进程稍后通过 call-enter 进入调用是非确定性的。delay 和 delay-cond 规则描述独立进程上的动作如何通过延迟指令的执行或分支选择而并发执行。

3 类型系统

我们开发了一种受非干扰标准技术启发、并针对编排进行了调整的类型系统 [33, 42], 。安全标签是一个完备格 $(\mathscr{L}, \sqsubseteq)$ 的元素，其底元素为 $\bot$，并满足 $\bot \sqsubseteq l$，对每个 $l \in \mathscr{L}$ 均成立。标签用于表示 Denning 安全级别；我们为每个变量分配一个安全标签。进程安全标记 $\gamma : \text{Var} \to \mathscr{L}$ 为进程所访问的变量分配安全标签。编排安全标记 $\Gamma : \text{PName} \to (\text{Var} \to \mathscr{L})$ 则将进程名称映射到各自的标记。信息流策略禁止信息从安全标签较高的对象流向关联标签较低的对象，从而定义了机密性。

类型判断定义在递归编排的语法上，是满足图 2 所示推理模式的最小关系。规则 t-const、t-var、t-lfun 对表达式 $\Gamma\, p \vdash e : \ell$ 进行类型判定，其中 $\ell \in \mathscr{L}$ 表征 e 的敏感度。函数取其参数标签的并，因此假定局部函数能够保留标签且不会引入额外的信息泄露。规则 t-local 和 t-com 分别对向存储位置写入的两种结构进行类型判定，即局部赋值和通信。这些是唯一会实际强制执行安全条件的规则：直观而言，目标的机密性必须至少与被写入的数据以及写入发生时的控制上下文一样高。请注意，规则 t-com 将通信视为进程间的赋值，因此假定通信信道是私有的。规则 t-sel 对选择进行类型判定：由于它只发送固定标签 L ，因此不会引入任何

[5] 选择结构在端点投影中处理分支构造时发挥着关键作用，但这超出了本文的讨论范围。感兴趣的读者可参阅 [32, 第 6]章。

$$\textsc{T-CONST}\ \frac{}{\gamma \vdash v : \bot} \qquad \textsc{T-VAR}\ \frac{}{\gamma \vdash x : \gamma\, x} \qquad \textsc{T-LFUN}\ \frac{\gamma \vdash e_1 : \ell_1 \ \cdots\ \gamma \vdash e_n : \ell_n \qquad \ell' = \bigsqcup_{i=1}^{n} \ell_i}{\gamma \vdash f(e_1, \ldots, e_n) : \ell'}$$

$$\textsc{T-LOCAL}\ \frac{\Gamma\, p \vdash e : \ell' \quad \ell' \sqcup pc \sqsubseteq \Gamma\, p.x}{\Delta; \Gamma; pc \vdash\ p.x := e} \qquad \textsc{T-COM}\ \frac{\Gamma\, p \vdash e : \ell' \quad \ell' \sqcup pc \sqsubseteq \Gamma\, q.x}{\Delta; \Gamma; pc \vdash\ p.e \to q.x}$$

$$\textsc{T-SEL}\ \frac{}{\Delta; \Gamma; pc \vdash\ p \to q[L]} \qquad \textsc{T-COND}\ \frac{\Gamma\, p \vdash e : \ell' \quad \Delta; \Gamma;\, \ell' \sqcup pc \vdash C_1 \quad \Delta; \Gamma;\, \ell' \sqcup pc \vdash C_2}{\Delta; \Gamma; pc \vdash\ \textbf{if}\ p.e\ \textbf{then}\ C_1\ \textbf{else}\ C_2}$$

$$\textsc{T-PROC}\ \frac{\Gamma' \in \Delta\, X\, pc \quad \Gamma[\bar{q} \mapsto \bar{p}] \equiv_{\{\bar{q}\}} \Gamma'}{\Delta; \Gamma; pc \vdash\ X(\bar{p})} \qquad \textsc{T-SEQ}\ \frac{\Delta; \Gamma; pc \vdash I \quad \Delta; \Gamma; pc \vdash C}{\Delta; \Gamma; pc \vdash\ I\ ;\ C} \qquad \textsc{T-NIL}\ \frac{}{\Delta; \Gamma; pc \vdash\ \mathbf{0}}$$

图 2. 类型判断。

显式信息流以及任何隐式信息流都已由当前的 pc捕获。规则 t-cond 处理隐式流：它先对守卫进行类型检查，再在更新后的程序计数器标签下对两个分支进行类型检查，该标签会追踪守卫的敏感度。换言之，t-cond 是唯一能够提升 pc的规则；其他规则都会保持其不变。规则 t-proc 通过查询分析上下文 Δ对过程调用进行类型检查。 Δ 的作用是概括每个过程的可接受调用上下文，并将其表示为 pc的函数：只有当调用者的类型环境在形式参数重命名后与 Δ中记录的某个上下文相匹配时，调用才会被接受。这是支持递归而无需反复重新检查过程体的关键机制，同时也确保过程只会在为其确立的安全假设下被调用。为支持这一机制，我们引入了新的记法： $\bar{q}$ 表示过程 X的形式参数， $\bar{p}$ 表示调用点的实参。上下文重命名 $\Gamma[\bar{q} \mapsto \bar{p}]$会更新 Γ ，使每个 q_i 都指向相应p_i的进程安全标记。受限相等 $\Gamma \equiv_S \Gamma'$表示 Γ和 Γ'在 S中所有进程的所有变量上都一致。规则 t-nil 和 t-seq 纯粹是结构性的：$\mathbf{0}$ 始终可类型化，而顺序组合只需将可类型化的编排组合起来。

4 可靠性

现在，我们将证明我们的类型系统对于终止性不敏感非干扰是可靠的。为陈述该定理，我们先引入一些记号。

自然语义。我们引入一种自然语义 [27]。我们定义关系

$$\langle C, \Sigma, \mathscr{C} \rangle \Downarrow^{M} \Sigma'$$

其中 M 是一个转换标签序列，定义为最小关系：

$$\langle \mathbf{0}, \Sigma, \mathscr{C} \rangle \Downarrow^{[]} \Sigma \qquad \frac{\langle C, \Sigma, \mathscr{C} \rangle \xrightarrow{\mu} \langle C', \Sigma', \mathscr{C} \rangle \qquad \langle C', \Sigma', \mathscr{C} \rangle \Downarrow^{M} \Sigma''}{\langle C, \Sigma, \mathscr{C} \rangle \Downarrow^{\mu :: M} \Sigma''}$$

攻击者观察模型。我们通过定义low等价关系 $\Sigma_1 \equiv^{\Gamma}_{low} \Sigma_2$来形式化 low级观察；当两个存储在 Γ 中标签为 $\sqsubseteq low$的所有变量上一致时，该关系成立。唯一的 low可观察输出是此类 low安全变量的值。

然后，我们陈述可靠性所需的关键假设：

良构过程上下文。对于每个定义 $X(\bar{p}) = C \in \mathscr{C}$：

$$\mathrm{pn}(C) \subseteq \{\bar{p}\} \tag{1}$$

良类型安全过程上下文。对于每个 $X(\bar{p}) = C \in \mathscr{C}$ 与 $pc \in \mathscr{L}$：

$$\Delta\, X\, pc = \mathcal{G} \implies \forall \Gamma \in \mathcal{G}.\, \Delta; \Gamma; pc \vdash C \tag{2}$$

其中 $\mathcal{G}$ 是程序计数器 pc 处 C 的可接受环境集合。

确定性表达式。给定进程存储 σ、表达式 e 和取值 v_1, v_2：

$$\sigma \vdash e \downarrow v_1 \Rightarrow \sigma \vdash e \downarrow v_2 \Rightarrow v_1 = v_2$$

4.1 带注释的编排

我们通过对计算长度进行归纳来证明可靠性。由于配置的非干扰性可能取决于其执行历史，因此我们必须在执行步骤之间传递信息。为此，我们引入带注释的编排，作为递归编排的扩展。在这一变体中，显式的方括号用于界定代码片段，这些片段允许在不同的执行过程中有所差异。除了方括号之外，它与递归编排还有两点不同：

– 指令与编排的扁平化：方括号可以括起任意长度的指令序列（包括空序列），因此诸如 $[\mathbf{0}]$ 和$\mathbf{0}$ 这样的项彼此不同。– 移除运行时项和标签选择指令：尽管并非必需，这种简化让可靠性证明更为容易。其理由是，这两条指令都不会影响最终计算所得的存储，并且仅在语义中用于支持此处省略的投影 [32], 。

扩展语法支持一种弱于语法相等的等价概念。我们将编排之间的 $C_1 \approx_{low} C_2$ low-等价记作：该关系在忽略方括号子项内容的同时，以结构化方式比较编排，无论其内部结构如何，都将它们视为相等。

插桩存储。我们将 [CStore], 定义为CStore的扩展，其中包含可能带方括号的值 [Val]。形式化地：

$$[\mathsf{Val}] ::= v \mid [v] \quad \text{with } v \in \mathsf{Val}$$

为了关联插桩存储和参考存储，我们引入两个概念：良构性和正确性。

若对于所有 p, x, v： [CStore] $[\Sigma]$ 相对于 Γ 良构，则称其为

$$[\Sigma]\, p.x = [v] \iff \Gamma\, p.x \not\sqsubseteq low$$

其中， $\not\sqsubseteq$ 表示可观测性： $a \not\sqsubseteq low$ 意味着 low 无法看到 a 的值。

我们定义一个降级函数 $\lfloor [\Sigma] \rfloor$ ，它会移除值中的方括号，同时保持未加方括号的值不变。若插桩存储 $[\Sigma]$ 与参考存储 Σ 满足 $\lfloor [\Sigma] \rfloor = \Sigma$，则称 $[\Sigma]$ 正确。

配置的低等价性。 low-等价的概念可自然地扩展到插桩存储：忽略加方括号的值后，若两个存储在所有位置上逐点相同，则称它们是 low-等价的。我们将这一概念扩展到配置：若 $\langle [C_1], [\Sigma_1], \mathscr{C} \rangle$ 与 $\langle [C_2], [\Sigma_2], \mathscr{C} \rangle$ 满足 $[C_1] \approx_{low} [C_2]$ 和 $[\Sigma_1] \approx_{low} [\Sigma_2]$，则称它们是 low-等价的。

插桩语义。 插桩语义扩展了参考语义，显式处理带方括号的值和编排，并以信息流策略 Γ 和观察级别 low为参数。其目的有二：在整个执行过程中维持插桩存储的良构性，并在必要时通过在分支续延外加上方括号，记录高安全级别信息对控制流的依赖。

类型系统扩展。 我们为类型系统添加一条规则：只要带方括号的编排主体在严格较高的安全级别下执行，就可以在较低的程序计数器下为其赋型，从而将高安全级别效应隔离在方括号内。

关联参考执行与插桩执行。 要关联参考语义与插桩语义，可以将后者视为普通执行的装饰式重放。因此，参考执行的提升是指一种插桩执行：它遵循相同的计算选择，同时通过在相应的续延和值外加上方括号，额外记录高安全级别数据或控制可能影响执行的位置。

4.2 主要定理

定理1（终止不敏感非干扰）。设 C 是在判断式 $\Delta;\ \Gamma;\ \bot \vdash C$下类型良好的编排。给定两个满足 low-等价关系 $\Sigma_1 \equiv^{\Gamma}_{low} \Sigma_2$的编排存储 Σ_1、 Σ_2，假设存在以下终止执行：

$$\langle C, \Sigma_1, \mathscr{C} \rangle \Downarrow^{M_1} \Sigma_1' \quad and \quad \langle C, \Sigma_2, \mathscr{C} \rangle \Downarrow^{M_2} \Sigma_2'$$

那么，所得存储满足关系 $\Sigma_1' \equiv^{\Gamma}_{low} \Sigma_2'$。

证明概要。从概念上说，证明通过在插桩语义中构造相应的执行来展开，然后应用标准的进展性与保持性论证 [34], ，其中证明 low-等价性在插桩归约过程中保持不变。现在给出完成该论证所需的主要引理：

– 完备性：该引理确立了参考语义与插桩语义之间的对应关系。特别是，每个终止执行都可以提升为一次插桩执行，其所得存储相对于 Σ_1', Σ_2' 是正确的，且相对于 Γ 是**良构的**。– 保持性：类型判断在语义下保持不变，即从一个良类型配置出发，每个归约步骤都会产生一个良类型配置。这里，良类型配置由一个良类型编排和一个良构存储组成。处理相互递归的过程，关键在于假设存在良类型安全过程上下文。– 展开：在适当的类型与良构性假设下，插桩语义在每个归约步骤中都会保持配置的 low-等价性。该引理说明了引入插桩语法和语义的动机：尽管逐步执行时严格的语法相等可能不成立，插桩编排的 low-等价性仍得以保持。

证明通过对插桩执行中的归约步骤数进行归纳来进行；这些执行由 完备性 引理构造。零长度情形显然可由给定的自然语义定义得出。 $n+1$ 情形则结合保持性 引理（使我们能够使用归纳假设）和展开 引理（其本质上逐步证明非干扰）得出。

请注意，我们只考虑终止执行的配对；与终止不敏感非干扰的设定一致，我们有意忽略发散。

完整定义和证明见 [5, 第 4]章。

5 上下文重构

我们提出一种算法，通过约束生成和求解，从过程定义 Δ $\mathscr{C}$中重构。

约束生成阶段。 一个约束 的形式为 $\Psi \sqsubseteq p.x$，其中 Ψ 是一个符号化界限，由带位置变量、一个特殊的新鲜变量η以及格运算构成。对于在环境 Γ 和程序计数器标签 pc下对 Ψ 求值的结果，我们记作 $[\![\Psi]\!]\,\Gamma\,pc$ 。约束的求值结果为一个真值。

约束生成算法 $\delta,\ \eta \vdash C \rhd E$ 通过对编排进行递归定义：给定一个约束

上下文 δ （将过程名称映射到约束集）和一个新变量η ，用于表示 pc的界。对于每个过程 $X(\bar{p}) = C$ ，我们遍历C ，并严格按照第 3 节类型规则所要求的内容生成排序约束，同时将表达式级别保持为符号形式。例如， $p.x := e$生成约束 $\Psi_e \sqcup \eta \sqsubseteq p.x$ ，其中 Ψ_e 以符号形式表示 e的安全级别，而 η 表示 pc 的安全级别。该算法与类型规则的主要区别在于对过程调用的处理：调用会从 δ 中导入与被调用过程关联的约束集，并将形式参数重命名为实际参数。这种约束转发也是理解该算法在 δ 中单调性的关键直觉。

迭代。 单次重构遍会在给定的 δ 下为一个过程体生成约束。我们定义一个算子 $\phi_{\mathscr{C}}$ ，它通过针对 $\mathscr{C}$ 中的每个过程体运行一次约束生成并收集所得的约束集，将 δ 映射为一个新的上下文 δ' 。由于 $\phi_{\mathscr{C}}$ 是单调的，Knaster-Tarski 定理 [40] 保证存在一个最小不动点，记作 $\mu\phi_{\mathscr{C}}$。

生成 Δ。我们定义 $\Delta \triangleq gen\ (\mu\phi_{\mathscr{C}})$，其中 $gen\ \delta$ 会创建一个从过程和 pc 到满足 δ 中约束的环境集合的映射，形式化地：

$$\Delta\ (X, pc) \triangleq \{\Gamma \mid \forall (\Psi \sqsubseteq p.x) \in E_X,\ [\![\Psi]\!]\ \Gamma\ pc \sqsubseteq \Gamma\ p.x\}\,,$$

此处， E_X 是与 $\mu\phi_{\mathscr{C}}$ 中的 X 相关联的约束集。

5.1 重构的正确性

现在我们证明，生成的上下文相对于类型关系是正确的。

定理 2（生成的过程上下文的类型良构性）。设$\Delta \triangleq gen\ (\mu\phi_{\mathscr{C}})$ 。那么，对于每个过程标识符 X、程序计数器标签pc和环境 Γ，若 $\Gamma \in \Delta\ X\ pc$，则有 $\Delta, \Gamma, pc \vdash C_X$。

证明概要。关键引理是单步可靠性：若约束重建为编排 C 在某个 δ 下生成约束 E ，且环境 Γ 满足 E，那么 C 在 $\Delta = gen\ \delta$ 和 Γ 下是可类型化的。直观而言，约束重建只是将类型规则反向执行：对于每种结构，它都会恰好生成作为规则前提所需的流不等式，因此，任何满足生成约束的 Γ 都能立即提供重建 C的类型推导所需的假设。最后，设 $\delta^\star = \mu\phi_{\mathscr{C}}$；由于 $\delta^\star$ 是不动点，对于每个过程 X ，存储在 $\delta^\star\ X$ 处的约束集恰好是在 $\delta^\star$下从 C_X 重建得到的约束集，因此任何$\Delta^\star = gen\ \delta^\star, \Gamma \in \Delta^\star\ X\ pc$ 都满足单步可靠性的假设，进而有 $\Delta^\star, \Gamma, pc \vdash C_X$。

终止性。由于本节中的所有函数都定义在有限结构上，因此总能找到一个可终止的算法来计算它们。

完整的定义和证明见 [5,第 5]章。

6 相关工作

Lluch-Lafuente 等人的相关研究聚焦于编排程序的信息流控制。 [29], 他们针对的是自由裁量型信息流策略，这些策略以访问控制约束的形式表达，用于规定哪些主体可以读取或更新特定数据项。他们在操作语义中加入了丰富的信息流注解，并提出了一种可靠的类型系统，用于静态检查编排程序是否符合给定策略。相比之下，我们确立了更强的语义性质：终止不敏感非干扰。

更广泛地说，信息流安全有着悠久的历史，始于 Denning 格模型 [21]以及 Goguen 和 Meseguer 对非干扰的开创性阐述 [23]。Volpano 等人 [41] 提出了最早的一批用于命令式环境下安全信息流分析的类型系统，之后大量研究进一步改进并推广了适用于顺序语言的基于类型的强制执行机制 [33,38]。这些系统通常通过与我们采用的规约密切相关的程序计数器规约，为显式流和隐式流提供静态、组合式保证。我们的工作直接建立在这条研究脉络之上。关键区别在于，我们将这些基础思想从顺序的局部程序拓展到分布式环境，在其中通信和控制流均以编排的形式进行全局描述。

信息流安全的动态与混合技术也得到了广泛研究。例如，基于语言的运行时强制机制 [38] 可以监控执行过程，并在运行时阻止不安全行为。近期关于宽松的运行时信息流控制的研究 [7]提高了在存在异常等语言特性时的精确度。然而，纯动态机制本质上只能针对单次具体执行进行推理，而非干扰是关于多次执行的关系属性。因此，仅靠动态强制通常无法确立语义上的非干扰；要将此类保证扩展到全局指定的分布式协议，也会带来额外挑战，尤其是针对某些类别的隐式流。相比之下，我们的方法为编排程序提供了静态、组合式的终止不敏感非干扰证明。

我们的工作也与并发演算的类型系统，以及带有安全注解的会话形式体系相关。Bhargavan 等人 [6]和 Capecchi 等人 [10] 率先将访问控制和信息流保证融入会话类型化进程演算，将协议结构与安全级别结合起来。后续关于安全会话类型和多方协议的研究，在本地端点进程层面确立了访问控制或信息流属性 [9, 14]。这些方法与我们的方法之间的区别，反映了多方会话类型与编舞式编程之间更广泛的差异。基于会话的方法通常会根据全局类型规范验证端点实现，并推导出用于类型检查进程的本地类型，而进程的内部计算则由类型抽象

了。在此相反，我们验证的是以编排程序编写的完整分布式实现；该程序不仅描述通信结构，还指定具体的数据操作和控制流。随后通过投影得到端点实现。因此，非干扰只需在全局程序层面确立一次，投影端点便能通过构造继承这一性质。

ProVerif [8]和 Tamarin [30]等安全协议验证工具，为在符号模型中检查密码协议提供了强大的自动化支持。这些方法擅长对密码原语和攻击者能力建模，并能够验证复杂的认证和保密性质。然而，它们基于专门的协议形式系统，通过对特定协议实例进行模型检测，来验证其是否满足用户指定的安全目标。它们并不旨在为分布式应用的编程模型提供一种以类型为基础、可组合的非干扰刻画。我们的方法与之互为补充。我们为参考编排编程语言配备了静态类型系统，支持可组合的静态类型检查与重构，并证明了一个一般性的句法非干扰定理。我们并非逐个分析协议实例，而是建立了一项语言级保证，统一适用于所有类型正确的编舞程序。

编舞式编程已被广泛发展为构造即正确的分布式编程基础 [32]。其核心成果是端点投影：编排程序可以编译为局部端点实现，并且这些实现由构造保证正确，从而确保无死锁性和通信安全等性质 [3, 4, 16, 24, 26, 31, 36]。基础性研究确立了投影的语义正确性及其与多方会话类型和全局类型的联系 [11, 13, 20]，并推动了进一步发展，例如基于模型的综合 [2], 、用于关于编舞程序进行推理的逻辑 [12, 15], ，以及经过机械认证、可编译为端点代码的编译过程 [17, 18, 25, 37]。这些理论基础已通过多种编舞式编程语言的实现得到落实，包括 Chor [11], 、AIOCJ [19] ，以及较新的 Choral [22], 和 HasChor [39],。这些语言将编舞式编程与主流的面向对象和函数式编程语言相结合，并探索高阶编排、位置多态和库级端点投影等高级特性。在这项研究中，主要关注点一直是功能正确性和通信安全，而我们的工作则首次探讨了非干扰。

最后，我们的证明方法建立在用于确立类型可靠性与安全性质的标准句法技术之上。我们遵循Pierce [34]以及Wright与Felleisen [42], 的进展与保持风格，并采用Kahn [27]风格的自然语义来表述终止不敏感非干扰。带括号的插桩语义如今在基于语言的信息流安全中已属标准做法 [33, 38]。

7 结论

我们提出了一种面向编舞程序中非干扰的基于类型的技术。我们的类型系统将策略参数化安全格与程序计数器规约相结合，以防止通过显式数据流和隐式控制依赖造成的信息泄漏。良类型的编排满足终止不敏感非干扰，从而确保高安全级别数据不能影响低安全级观察。我们的类型系统通过面向可容许调用环境的过程上下文来支持过程式编排。我们形式化了一种基于约束的重构算法，用以计算该上下文，从而无需手工安全注解。

我们看到未来有几个研究方向。首先，我们计划扩展类型系统，以处理表达能力更强的安全策略，包括允许在指定条件下受控释放信息的降密机制。其次，我们打算探索其他安全性质，不再局限于终止不敏感非干扰。事实上，侧信道 [28]（例如时序、终止性、资源使用、缓存效应、消息大小、依赖于调度器的行为）不在这一视角的范围内，除非语义和攻击者观察模型明确将其设为可观察量。我们的终止不敏感方法有意抽象掉时序和发散，这足以应对许多实际场景，但并未涵盖所有潜在的隐蔽信道。当侧信道至关重要时，可以通过为语义加入成本或时序可观察量，并采用时序敏感／步数敏感定义 [1]，将其纳入考量。最后，我们计划使用 Lean 或 Rocq 等证明助手，将整个开发过程形式化。纸笔证明虽然增强了我们对结果的信心，但完整的形式化将提供机器验证保证，并消除冗长的分类讨论和归纳论证中出现细微错误的可能性。除增强我们对结果的信心外，成功完成这一形式化工作还将为开发经认证的类型检查器和面向安全编舞式编程的经验证编译工具链奠定基础。我们设想，这些检查器能够搭建桥梁，将理论成果带入实际实现，例如将其集成到现有的编舞式编程框架中，如 Choral [22]。

参考文献

1. Almeida, J.B., Barbosa, M., Barthe, G., Dupressoir, F., Emmi, M.：验证恒定时间实现。载于：第25届USENIX安全研讨会（USENIX Security 16）。第53–70页。USENIX协会，美国德克萨斯州奥斯汀（2016年8月），
https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/almeida
2. Autili, M., Ruscio, D.D., Salle, A.D., Inverardi, P., Tivoli, M.：用于强制实施编舞可实现性的基于模型的综合过程。载于：Cortellessa, V., Varró, D.（编）《软件工程基础方法——第16届国际会议FASE 2013》，作为欧洲软件理论与实践联合会议ETAPS 2013的一部分举行，意大利罗马，2013年3月16–24日，

论文集。计算机科学讲义，第7793卷，第37–52页。施普林格（ 2013 10 1007 978 3 642 37057 1_4 ）。https://doi.org/ 。 / 3 - - - - 。Barbanera, F., Lanese, I., Tuosto, E.：编舞自动机。载于：Bliudze, S., Bocchi, L.（编）《协调模型与语言——第22届IFIP WG 6.1国际会议COORDINATION 2020》，作为第15届分布式计算技术国际联合会议DisCoTec 2020的一部分举行，马耳他瓦莱塔，2020年6月15–19日，论文集。计算机科学讲义，第12134卷，第86–106页。施普林格（2020）。https://doi.org/10.1007/978-3-030-50029-0_64. Barbanera, F., Lanese, I., Tuosto, E.：形式编舞语言理论。《逻辑方法与计算机科学》**19**(3)（2023）。https://doi.org/10.46298/LMCS-19(3:9)2023 5. Bertoni, M.：编舞语言中非干扰的基于类型的机械化强制实施，https://amslaurea.unibo.it/id/eprint/36727/ 6. Bhargavan, K., Corin, R., Deniélou, P., Fournet, C., Leifer, J.J.：多方会话的密码协议合成与验证。载于：第22届IEEE计算机安全基础研讨会论文集，CSF 2009，美国纽约杰斐逊港，2009年7月8–10日。第124–140页。IEEE计算机学会（2009）。https://doi.org/10 26 7 1109 2009 . /CSF. . . Bichhawat, A., Rajani, V., Garg, D., Hammer, C.：存在异常时的宽松运行时信息流控制。《计算机安全杂志》**29**(4)，361–401（2021）。https://doi.org/10 211385 8 3233. /JCS- . Blanchet, B.：使用应用π演算和ProVerif对安全协议进行建模与验证。《隐私与安全基础与趋势》**1**(1-2)，1–135（2016）。https://doi.org/10.1561/3300000004 9. Capecchi, S., Castellani, I., Dezani-Ciancaglini, M.：多方会话中的信息流安全性。《数学结构与计算机科学》**26**(8)，1352–1394（2016）。https://doi.org/10 0960129514000619 10 1017. /S . Capecchi, S., Castellani, I., Dezani-Ciancaglini, M., Rezk, T.：用于访问控制和信息流控制的会话类型。载于：Gastin, P., Laroussinie, F.（编）《CONCUR 2010——并发理论，第21届国际会议》，法国巴黎，2010年8月31日–9月3日，论文集。计算机科学讲义，第6269卷，第237–252页。施普林格（2010）。https://doi.org/10.1007/978-3-642-15375-4_1711. Carbone, M., Montesi, F.：设计即无死锁：多方异步全局编程。载于：Giacobazzi, R., Cousot, R.（编）《第40届ACM SIGPLAN-SIGACT编程语言原理研讨会》，POPL ’13，意大利罗马，2013年1月23–25日。第263–274页。ACM（2013）。https://doi.org/10 2429101 12 1145 2429069 . / . . Carbone, M., Montesi, F., Schürmann, C.：逻辑视角下的编舞。《分布式计算》**31**(1)，51–67（2018）。https://doi.org/10.1007/S00446-017-0295-1，https://doi.org/10 1 13 1007 00446 017 0295 . /s - - - . Castagna, G., Dezani-Ciancaglini, M., Padovani, L.：论全局类型与多方会话。《逻辑方法与计算机科学》**8**(1)（2012）。https://doi.org/10.2168/LMCS-8(1:24)2012 14. Castellani, I., Dezani-Ciancaglini, M., Pérez, J.A.：多方通信中的自适应与安全信息流。《计算机形式化方法》**28**(4)，669 3 15 696 2016 10 1007 00165 016 0381 –（）。https://doi.org/ . /S - - - . Cruz-Filipe, L., Graversen, E., Montesi, F., Peressotti, M.：关于编舞程序的推理。载于：Jongmans, S., Lopes, A.（编）《协调模型与

语言。计算机科学讲义，第13908卷，第144–162页。施普林格（
2023 10 1007 978 3 031 35361 1_8 ）。https://doi.org/ 。/ - - - -
16 。Cruz-Filipe, L., Montesi, F.：编舞式编程的核心模型。《理论计算机科学》 **802**，38 005 17 66 2020 10 1016 2019 07 – ( )。https://doi.org/ 。/J.TCS. . . . Cruz-Filipe, L., Montesi, F., Peressotti, M.：编舞编译认证。载于：Cerone, A., Ölveczky, P.C.（编），计算的理论方面 - ICTAC 2021 - 第18届国际研讨会，线上活动，哈萨克斯坦努尔苏丹，2021年9月8日至10日，论文集。计算机科学讲义，第12819卷，第
115 133 2021 10 1007 978 3 030 85315 0_8 –页。
18 施普林格（）。https://doi.org/ 。/ - - - - 。Cruz-Filipe, L., Montesi, F., Peressotti, M.：编舞式编程形式理论。《自动推理杂志》**67**(2)，21（2023）。https://doi.org/10.1007/ S10817-023-09665-3 19. Dalla Preda, M., Gabbrielli, M., Giallorenzo, S., Lanese, I., Mauro, J.：动态编舞：理论与实现。逻辑方法与计算机科学**13**(2)（2017 2017 20 10 23638 13 2 1 ）。https://doi.org/ 。/LMCS- (:) 。Deniélou, P., Yoshida, N.：多方会话类型与通信自动机相遇。载于：Seidl, H.（编），编程语言与系统 - 第21届欧洲编程研讨会，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，论文集。计算机科学讲义，第7211卷，第
194 213 2012 10 1007 978 3 642 28869 2_10 –页。施
21 普林格（）。https://doi.org/ 。/ - - - - 。Denning, D.E.：安全信息流的格模型。美国计算机协会通讯**19**(5)，236–243（1976）22. Giallorenzo, S., Montesi, F., Peressotti, M.：Choral：面向对象的编舞式编程。《ACM程序设计语言汇刊》**46**(1)，1:1–1:59（2024）。https: //doi.org/10 3632398 23 1145. / 。Goguen, J.A., Meseguer, J.：安全策略与安全模型。载于：1982年IEEE安全与隐私研讨会。第11–11页。IEEE（1982）24. Graversen, E., Montesi, F., Peressotti, M.：编舞式编程中的遗漏故障。载于：Moscardelli, L.（编），第26届意大利理论计算机科学会议论文集，意大利佩斯卡拉，2025年9月10日至12日。CEUR研讨会论文集，第4039卷，第169–184页。CEUR-WS.org（2025），https://ceur-ws.org/Vol-4039/paper02.pdf 25. Hirsch, A.K., Garg, D.：Pirouette：高阶类型化函数式编舞。《ACM程序设计语言会议录》**6**(POPL)，1–27（2022）。https://doi.org/10.1145/ 3498684, 3498684 26 10 1145 https://doi.org/ 。/ 。Jongmans, S., van den Bos, P.：编舞的谓词变换器——计算编舞式编程中的前置条件。载于：Sergey, I.（编），编程语言与系统 - 第31届欧洲编程研讨会，ESOP 2022，作为欧洲软件理论与实践联合会议ETAPS 2022的一部分举行，德国慕尼黑，2022年4月2日至7日，论文集。计算机科学讲义，第13240卷，第520–547页。施普林格（2022）。https://doi.org/
10 1007 978 3 030 99336 8_19 . / - - - - 。Kahn, G.：自然语义。载
27 于：计算机科学理论方面年度研讨会。第22–39页。施普林格（1987）28. Kelsey, J., Schneier, B., Wagner, D., Hall, C.：乘积密码的侧信道密码分析。载于：欧洲计算机安全研究研讨会。第97–110页。施普林格（1998）29. Lluch-Lafuente, A., Nielson, F., Nielson, H.R.：面向交互的规格说明的自主信息流控制。载于：Martí-Oliet, N., Ölveczky, P.C.,

Talcott, C.L.（编），逻辑、重写与并发——献给José Meseguer的65岁生日论文集。计算机科学讲义，第9200卷，第427–450页。施普林格（2015）。https://doi.org/10.1007/ 978-3-319-23165-5_2030. Meier, S., Schmidt, B., Cremers, C., Basin, D.A.：用于安全协议符号分析的TAMARIN 证明器。载于：Sharygina, N., Veith, H.（编），计算机辅助验证 - 第25届国际会议，CAV 2013，俄罗斯圣彼得堡，2013年7月13日至19日，论文集。计算机科学讲义，第8044卷，第696–701页。施普林格（2013）。https://doi.org/10.1007/ 978-3-642-39799-8_4831. Montesi, F.：编舞式编程。博士论文，哥本哈根信息技术大学（2013），https://www.fabriziomontesi.com/files/ choreographic-programming.pdf 32. Montesi, F.：编舞导论。剑桥大学出版社（2023）33. Myers, A.：使用小步操作语义证明 while 语言的非干扰性（2011）34. Pierce, B.C.：类型与程序设计语言。麻省理工学院出版社（2002）35. Plotkin, G.D.：结构操作语义的起源。《逻辑与代数编程期刊》**60**，3–15（2004）36. Plyukhin, D., Peressotti, M., Montesi, F.：Ozone：完全无序的编舞。载于：Aldrich, J., Salvaneschi, G.（编），第38届欧洲面向对象编程会议，ECOOP 2024，2024年9月16日至20日，奥地利维也纳。LIPIcs，第313卷，第31:1–31:28页。达格斯图尔城堡—莱布尼茨信息学中心（2024 31 37 10 4230 2024 ）。https://doi.org/ 。/LIPICS.ECOOP. . . Pohjola, J.Å., Gómez-Londoño, A., Shaker, J., Norrish, M.：Kalas：经过验证的端到端编舞语言编译器。载于：Andronick, J., de Moura, L.（编），第13届国际交互式定理证明会议，ITP 2022，以色列海法，2022年8月7日至10日。LIPIcs，第237卷，第27:1–27:18页。达格斯图尔城堡—莱布尼茨信息学中心（2022）。https://doi.org/10.4230/LIPICS.ITP. 2022 27 38 27, 10 4230 2022 。https://doi.org/ 。/LIPIcs.ITP. . . Sabelfeld, A., Myers, A.C.：基于语言的信息流安全。IEEE《通信选定领域期刊》**21**(1)，5–19（2003）39. Shen, G., Kashiwa, S., Kuper, L.：HasChor：人人皆可使用的函数式编舞编程（函数式珍珠）。《ACM程序设计语言会议录》**7**(ICFP)，541–565（2023）。https://doi.org/10 3607849 40 1145. / 。Tarski, A.：格论不动点定理及其应用。（1955）41. Volpano, D., Irvine, C., Smith, G.：用于安全流分析的可靠类型系统。《计算机安全期刊》**4**(2-3)，167–187（1996）42. Wright, A.K., Felleisen, M.：类型可靠性的句法方法。《信息与计算》**115**(1)，38–94（1994）