arXiv:2301.06439v1 [cs.PL] 16 Jan 2023

基于局部轨迹的聚类关系线程模块化抽象解释

Michael Schwarz[1], Simmo Saan[2], Helmut Seidl[1],Julian Erhard[1], 和Vesal Vojdani[2]

[1] 慕尼黑工业大学，德国加兴 {m.schwarz, helmut.seidl, julian.erhard}@tum.de[2] 塔尔图大学，爱沙尼亚塔尔图 {simmo.saan, vesal.vojdani}@ut.ee

摘要。 我们构建了新颖的线程模块化分析，用于跟踪可能重叠的全局变量簇之间的关系信息，前提是这些变量簇受到公共互斥锁的保护。我们提出了一个框架，通过基于局部轨迹的抽象划分控制位置，系统地提高聚类关系分析的精度。作为一个实例，我们实现了对动态线程创建和汇合的分析。有趣的是，跟踪较少的关系全局变量信息可能会带来更高的精度。我们考虑了 2-可分解域这一类域，它涵盖了许多弱关系域（例如，八边形域）。对于这些域，我们证明，只需使用大小至多为 2 的全局变量簇，即可达到最高精度。

关键词： 线程模块化关系抽象解释，收集局部轨迹语义，簇，动态线程创建，并发

1 引言

追踪程序变量之间的关系，对于证明程序的性质或验证是否不存在某些编程错误[14, 16, 33]至关重要。对于多线程程序而言，推断关系性质尤其具有挑战性，因为必须考虑其他线程可能并行产生的所有干扰。在这样的环境中，只有由公共互斥锁保护的全局变量之间的关系，才可能在程序执行过程中保持不变。一般来说，由较少变量组成的簇上的关系，比变量较多的簇上的关系更稳定。此外，采用多面体抽象域等方法的整体式关系分析，众所周知代价极高 [36, 52]。追踪较小的 簇甚至可能比追踪较大的簇更精确 [19]。

例 *1.* 考虑以下程序。对全局变量 g、 h 和 i的所有访问都由互斥锁 a保护。

```
m    ain : x = 创建(t1); y =创建(t2);
lock(a); g = ?; h = ?; i = ?; 解锁(a); r =
汇合(y); lock(a); z = ?; g = z; h = z;i =
z; 解锁(a); lock(a); // 断言(h==i); (1) 断言(g
==h); (2) 解锁(a);
```

```
t1: t2: 锁（a）; 锁（a）;
x = h; g = ?; h = ?; i =
x; 解锁（a）; 解锁（a）; 返
回0; 返回 1;
```

在该程序中，主线程分别从 t_1 和 t_2开始创建两个新线程。随后，它锁定互斥锁 a，将所有全局变量以非确定方式设置为某个值，然后再次解锁 a。在汇合线程 t_2之后，它再次锁定a，将所有全局变量设置为相同的未知量，然后再次解锁 a。线程 t_1 将 i 设置为 h的值。线程 t_2 将 g 和 h 设置为（可能不同的）未知量。假设我们关注全局变量之间的等式。为了成功证明断言（1），必须检测出主线程是唯一的，因此无法读取其过去的写操作，因为这些写操作已被覆盖。此外，分析还需要证明线程 t_2 也是唯一的、已在断言之前被汇合，并且其写操作也必定已被覆盖。

为了证明断言 (2)，传播受 a 保护的所有全局变量值的联合抽象并不足够：在 t_1 中解锁 a 时，$g{=}h$ 不一定成立。如果将这一整体关系传播到 main 中最后一次对 a 的加锁处，就无法证明 (2)——尽管 t_1 既不修改 g，也不修改 h。□

这里我们说明，可以通过将互斥锁保护的所有全局变量的整体抽象替换为恰当选取的子簇，来弥补示例中所体现的精度损失。在该示例中，我们建议分别考虑子簇 $\{g, h\}$ 和 $\{h, i\}$。由于 t_1 不会向簇 $\{g, h\}$ 写入任何值，因此不会将不精确的关系 $\top$ 传播到主线程，从而可以证明断言 (2)。

为了对分析进行精细调整，我们依赖弱关系域。文献中提出了多种弱关系域，例如每个不等式两个变量 [51], 八边形域 [36, 37], 及其简化形式 [33, 35]。这些域共同具备一项我们感兴趣的技术性质：每个抽象关系都可以根据其在大小至多为 2 的变量子簇上的投影重构。我们将这类域称为 2-可分解域。除了数值型二可分解域之外，还可以构造非数值型二可分解域，例如关联字符串名称与函数指针的域。

基于二可分解域，我们设计了针对全局变量的线程模块化关系分析，通过考虑线程的局部知识，可以获得更高的精度。因此，我们不依赖全局轨迹语义，而是采用局部轨迹语义，为每个线程形式化描述它能够观察到的那部分计算历史 [48]。程序点的抽象值描述所有到达该程序点的局部轨迹集合。同样，为可观察动作记录的值，是以相应动作结束的所有局部轨迹的抽象。这类可观察动作例如互斥锁的解锁操作。抽象

值随后通过考虑局部轨迹的有限抽象而得到细化。为此，我们提出一个通用框架，将任意基础分析的组件作为黑盒重复利用。我们的贡献可概括如下：

– 我们提出一种新的全局变量关系分析，将其视为基于重叠变量簇的收集式局部轨迹语义的抽象（第3、4和8节）。– 我们的分析处理动态创建和汇合的线程；线程id可以通过变量传递给其他线程，也可以通过互斥锁进行同步（第3节）。– 我们提出一种通用方案，通过考虑局部轨迹的有限抽象，将基于历史的论证纳入分析（第 5 节）。– 我们将动态创建的线程id分析作为通用方案的一个实例，并将其用于排除自影响，以及来自不可能并行运行的线程的读操作（第6和第7节）。– 我们证明，通过跟踪变量的所有子簇，可以避免关系分析中的某些精度损失。对于二可分解关系域，我们证明，跟踪大小大于 2 的变量簇不会造成精度损失，因此可以省略（第8节）。

本文中的所有分析均已实现，第 9 节报告了实际评估结果，而第 10 节则详细介绍了相关工作。

2 关系域

首先，我们定义分析描述中所使用的关系域概念。令 $\mathcal{V}$ars 为一个变量集合，这些变量的类型可能各不相同。我们假设所有配置和赋值都是类型良好的，即（抽象）值的类型与为变量指定的类型相匹配。对于每种 τ 值类型，我们假设存在一个完备格 $\mathcal{V}_\tau^\sharp$，用于对来自 $\mathcal{V}_\tau$的相应具体值进行抽象。令 $\mathcal{V}^\sharp$ 表示这些格的集合，并令 $\mathcal{V}\text{ars} \to_\bot \mathcal{V}^\sharp$ 表示所有类型一致的赋值 σ 的集合，这些赋值将变量映射到非-$\bot$ 抽象值，并扩展一个专用的最小元（也记作 $\bot$），同时配备由此诱导的序关系。一个 关系域 $\mathcal{R}$ 则是一个完备格，并提供以下操作

$[\![x \leftarrow e]\!]^\sharp_{\mathcal{R}}$： $\mathcal{R} \to \mathcal{R}$ （表达式 e 的赋值）$r|_Y$： $\mathcal{R} \to \mathcal{R}$ （对 $Y \subseteq \mathcal{V}$ars 的限制）lift：$(\mathcal{V}\text{ars} \to_\bot \mathcal{V}^\sharp) \to \mathcal{R}$ unlift： $\mathcal{R} \to (\mathcal{V}\text{ars} \to_\bot \mathcal{V}^\sharp)[\![?e]\!]^\sharp_{\mathcal{R}}$： $\mathcal{R} \to \mathcal{R}$ （条件 e 的守卫）

左侧的操作为程序的基础操作提供了抽象状态变换器（将非确定性赋值表示为限制），而 lift 和 unlift 则允许将抽象变量赋值转换到关系域中，以及提取单个变量的信息。我们假设 lift $\bot = \bot$ and unlift $\bot = \bot$，并要求 unlift $\circ$lift $\sqsupseteq$ id，其中 $\sqsupseteq$指的是 $(\mathcal{V}\text{ars} \to_\bot \mathcal{V}^\sharp)$ 的排序。此外，我们要求 交操作 $\sqcap$ of $\mathcal{V}^\sharp$ and $\mathcal{R}$ 能安全地近似相应参数的具体化之交集

。将关系 r 限制到变量子集 Y，相当于忘记所有关于不属于 Y 的变量的信息。因此，我们要求$r|_{\mathcal{V}ars} = r$, $r|_\emptyset = \top$, $r|_{Y_1} \sqsupseteq r|_{Y_2}$ 在 $Y_1 \subseteq Y_2$ 时成立，$(r|_{Y_1})|_{Y_2} = r|_{Y_1 \cap Y_2}$，并且

$$\mathsf{unlift}\,(r|_Y)\,x = \top \quad (x \notin Y) \qquad\qquad \mathsf{unlift}\,(r|_Y)\,x = (\mathsf{unlift}\,r)\,x \quad (x \in Y) \quad (1)$$

因此，限制是幂等的。为方便起见，我们还定义了一个表示抽象值赋值的简写[3]: $[\![x \leftarrow^\sharp v]\!]^\sharp_{\mathcal{R}}\, r = \left(r|_{\mathcal{V}ars \setminus \{x\}}\right) \sqcap (\mathsf{lift}\,(\top \oplus \{x \mapsto v\}))$。为了构造抽象解释，我们还要求单调的具体化函数 $\gamma_{\mathcal{V}^\sharp}$： $\mathcal{V}^\sharp \to 2^{\mathcal{V}}$ 和 $\gamma_{\mathcal{R}}$： $\mathcal{R} \to 2^{\mathcal{V}ars \to \mathcal{V}}$，它们满足图 1 中给出的要求。

例 *2*。 作为值域 $\mathcal{V}^\sharp_\tau$，考虑适当类型的值集合 τ上的平坦格。关系域 $\mathcal{R}_1$ 由可满足的合取式构成，这些合取式表示变量之间或变量与常量之间的等式，其序关系为逻辑蕴含，并扩展为以 False 为最小元。在这个完备格中，最大元为 True。对于非$\bot$ 参数，操作 lift 和 unlift 可定义为

$$\mathsf{lift}\,\sigma = \bigwedge\{x = \sigma\,x \mid x \in \mathcal{V}ars, \sigma\,x \neq \top\} \qquad \mathsf{unlift}\,r\,x = \begin{cases} c & \text{if } r \implies (x = c) \\ \top & \text{otherwise} \end{cases}$$

r 到变量子集 Y的限制，由 r 所蕴含的所有等式的合取给出，这些等式仅包含来自 Y的变量或常量。□

按照例 2，也可以为非数值构造关系域。

变量聚类 $\mathcal{S} \subseteq 2^{\mathcal{V}ars}$ 是变量子集（簇）的集合。对于任意簇 $Y \subseteq \mathcal{V}$ars，令 $\mathcal{R}^Y = \{r \mid r \in \mathcal{R}, r|_Y = r\}$；这个集合收集了来自 $\mathcal{R}$ 的所有抽象值，其中仅包含关于 Y 中变量的信息。给定任意聚类 $\mathcal{S} \subseteq 2^{\mathcal{V}ars}$，任意关系 $r \in \mathcal{R}$ 都可以用来自 $\mathcal{R}^Y$ $(Y \in \mathcal{S})$ 的关系的交来近似，因为对于每个 $r \in \mathcal{R}$，都有 $r \sqsubseteq \bigsqcap\{r|_Y \mid Y \in \mathcal{S}\}$ 成立。

然而，一些关系域可以通过将其限制到特定簇子集而完全恢复。对于 $k \geq 1$，我们考虑由所有基数至多为 k的非空子集 $Y \subseteq \mathcal{V}$ars组成的集合 $\mathcal{S}_k$ 。如果关系域 $\mathcal{R}$ k-可分解，则其中每个抽象值都可以精确表示

[3] 我们用 $\sigma \oplus \{x_i \mapsto v_i \mid i = 1, \ldots, m\}$ 表示这样得到的变量赋值：将 σ 中 x_i 对应的值替换为 $v_i (i = 1, \ldots, m)$。

$$\forall a, b\colon\ a \sqsubseteq b \implies \gamma_{\mathcal{V}^\sharp}\,a \subseteq \gamma_{\mathcal{V}^\sharp}\,b \qquad \gamma_{\mathcal{R}}\,\bot = \emptyset \qquad \forall r, s\colon\ r \sqsubseteq s \implies \gamma_{\mathcal{R}}\,r \subseteq \gamma_{\mathcal{R}}\,s$$
$$\gamma_{\mathcal{R}}\ (\![x \leftarrow e]\!]^\sharp_{\mathcal{R}}\ r)\ \supseteq \{\sigma \oplus \{x \mapsto [\![e]\!]\sigma\} \mid \sigma \in \gamma_{\mathcal{R}} r\}$$
$$\gamma_{\mathcal{R}}\ (r|_Y)\ \supseteq \{\sigma \oplus \{x_1 \mapsto v_1, \ldots, x_m \mapsto v_m\} \mid v_i \in \mathcal{V}, x_i \in \mathcal{V}ars \setminus Y, \sigma \in \gamma_{\mathcal{R}} r\}$$
$$\gamma_{\mathcal{R}}\ (\text{提升}\ \sigma^\sharp)\ \supseteq \{\sigma \mid \forall x\colon\ \sigma\,x \in \gamma_{\mathcal{V}^\sharp}\ (\sigma^\sharp\,x)\ \} \qquad \gamma_{\mathcal{V}^\sharp}\ (\text{反提升}\ r)\ \ x \supseteq \{\sigma\,x \mid \sigma \in \gamma_{\mathcal{R}}\,r\}$$

图 1： $\gamma_{\mathcal{V}^\sharp}$ 所需的属性： $\mathcal{V}^\sharp \to 2^{\mathcal{V}}$和 $\gamma_{\mathcal{R}}$： $\mathcal{R} \to 2^{\mathcal{V}ars \to \mathcal{V}}$。

即取其在 $\mathcal{S}_k$ 的各个簇上的限制的交；并且，只需通过计算 $\mathcal{S}_k$ 的各个簇，就能恢复所有最小上界；也就是说，

$$r = \bigsqcap \left\{ r|_Q \mid Q \in \mathcal{S}_k \right\} \qquad (\bigsqcup R)\,|_Q = \bigsqcup \left\{ r|_Q \mid r \in R \right\} \quad (Q \in \mathcal{S}_k) \qquad (2)$$

对于每个抽象关系 $r \in \mathcal{R}$ 和每个抽象关系集合 $R \subseteq \mathcal{R}$，均成立。

示例 *3*。 上一个示例中的域 $\mathcal{R}_1$ 是 2-可分解的。这对于八边形 域 [36] 以及许多其他弱关系数值域（Pentagons [33], 加权六边形 [21], Logahedra [28], TVPI [51],dDBM [46], 和 AVO [11]）也成立。然而，仿射等式 或仿射不等式域[16, 30], 并非如此。Arceri 等人 [6] 最近在第 5.1 节至第 5.3 节提出的关系字符串域，是非数值型 2-可分解域的例子。

3 局部轨迹语义

我们基于 Schwarz 等人 [48]提出的局部轨迹语义框架展开研究。局部轨迹记录了所有影响特定线程当前配置的过去事件，该线程称为自我 线程。在 [48], 中已证明，局部轨迹语义等价于全局轨迹语义，而后者又等价于全局交错语义。具体而言，任何相对于局部轨迹语义而言可靠的分析，相对于交错语义而言也同样可靠。

虽然 Schwarz 等人 [48] 的框架允许对轨迹进行不同的形式化，但线程同步仅通过加锁/解锁和线程创建实现。我们对其语义进行推广，将某些动作识别为可被其他线程通过执行相应观察动作观察 到的动作（见表 1）。当自身线程执行观察动作时，会纳入一条以相应可观察动作为结尾的局部轨迹。这里，我们将互斥锁的加锁/解锁以及线程的创建/汇合作为可观察/观察动作。

例如，考虑图 2a 中的程序及其对应的局部轨迹（图 2b）。该轨迹为每个线程都包含一条泳道，表示该线程执行的步骤序列，其中图中的每个节点都表示该线程到达的一个配置。此外，轨迹还记录了创建和汇合的顺序，以及

表 1：可观察动作和观察动作，以及它们关联的并发原语。本文重点讨论的原语以粗体显示。

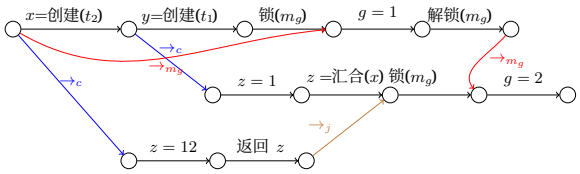
可观察动作	观察动作	编程概念
unlock(a)	锁(a)	**互斥锁**，监视器，...
返回 x	x'=汇合(x'')	**线程返回／汇合**
$g = x$	$x = g$	**读写全局变量**
signal(c)	wait(c)	条件变量
send(chan,v)	x = receive(chan)	基于通道的并发、套接字、...
集合_value	`get`	期货 / 承诺

```
main:
x  =创建(t2);
y = 创建(t1);
锁(m g);
g = 1;
解锁(m g);
z = 28;
```

```
t1:
z = 1;
z = 汇合(x);
锁(m g);
g = 2;
解锁(m g);
x = 创建(t2);
```

```
t2:
z = 12;
返回 z;
```

(a) 源代码



(b) 局部轨迹；对于此程序，执行开始于在程序点 main 处，x、y、z是局部变量，而 g 是全局变量。为确保原子性，对全局变量 g 的每次访问都由互斥锁 m_g保护，后续示例中将省略该锁。

图 2：示例程序及其对应的局部轨迹。

每个互斥锁 a 的加锁顺序，分别对应 a 、$\rightarrow_c$ 和 $\rightarrow_j$。这些顺序在线程配置之间引入了额外的关系。每条局部轨迹唯一的起始节点都是$main$ 线程的初始配置。

我们区分局部变量集合 $\mathcal{X}$ 和全局变量集合 $\mathcal{G}$ 。我们假设 $\mathcal{X}$ 包含一个特殊变量 self，用于保存从集合 $\mathcal{I}$中取值的当前线程*ID*。一个（局部）线程配置是一个二元组（u， σ），其中 u 是一个程序点，而类型一致映射 $\sigma: \mathcal{X} \to \mathcal{V}$ 为局部变量提供取值。全局变量的值不会在线程配置中显式表示，但可以通过查阅局部轨迹中对该全局变量的（唯一）最后一次写入来恢复。为建模弱内存效应，可以考虑较弱的最后一次写入概念。与[48], 中一样，我们考虑动作集合 $\mathcal{A}$ct，其中包括对集合 M 中的（不可重入）互斥锁进行加锁和解锁、将全局变量的值复制到局部变量中，反之亦然、创建新线程，以及对局部变量进行赋值和分支。我们为 $\mathcal{A}$ct 扩展了从线程返回和与线程汇合的动作。我们假设对全局变量的写入和读取是原子的（更准确地说，我们假设复制整数类型的值是原子的）。对每个全局变量 g，通过专用互斥锁 m_g 来保证这一点：访问 g之前立即加锁，访问后立即解锁。为简单起见，我们将对 g 的写入所对应的轨迹关联到该专用互斥锁 m_g，因此无需将全局变量的写入和读取视为可观察/观察动作。在示例中，我们省略这些互斥锁的显式加锁和解锁。按照约定，程序启动时所有全局变量的值均为 0，而局部变量最初可以取任意值。

每个线程均以一个控制流图表示，其边 $e \in \mathcal{E}$ 具有如下形式： $e = (u,$ 动作 $u')$，其中 act $\in \mathcal{A}$ct 是某个动作， u 和 u' 是程序点，且 $main$ 线程的起始点为 u_0。令 $\mathcal{T}$ 表示给定程序的所有局部轨迹的集合。对于控制流图中的每条边 e ，局部轨迹的形式体系都必须提供一个变换 $[\![e]\!]: \mathcal{T}^k \to 2^{\mathcal{T}}$ ，使得 $[\![e]\!](t_0, \ldots, t_{k-1})$ 扩展局部轨迹 t_0，并可能纳入其他

局部轨迹。对于操作 lock(a) $a \in \mathsf{M}$ 或 x=join(x') $x, x' \in \mathcal{X}$，$[\![e]\!]$ 的元数为二；也就是说，还会纳入另一条局部轨迹，该轨迹的最后一个操作分别为 unlock(a) 或 return x''。其余边变换的元数均为一。在所有情况下，如果操作不适用于其参数，所得局部轨迹集合可能为空。我们将 $[\![e]\!](T_0, \ldots, T_{k-1})$ 记作集合 $\bigcup_{t_0 \in T_0, \ldots, t_{k-1} \in T_{k-1}} [\![e]\!](t_0, \ldots, t_{k-1})$。

给定 $[\![e]\!]$ 的定义，集合 $\mathcal{T}$ 可以通过从集合 init 归纳定义，其中 init 由 ***main*** 线程的初始配置组成，包含 初始局部轨迹。为了开发高效的线程模块化抽象，我们关注以某个程序点 u 结尾、以对互斥锁 a 执行 解锁 操作（或来自 init）结尾，或以线程 i 的 返回 语句结尾的局部轨迹子集 $\mathcal{T}[u]$、$\mathcal{T}[a]$、$\mathcal{T}[i]$。Schwarz 等人 [48] 证明，这些子集可以描述为带副作用的约束系统 [5] 的最小解。在该系统中，除了对左侧未知量的贡献外，每个右侧还可以向其他未知量提供贡献（即 副作用）。这使得我们能够表达这样的分析：在对具有动态控制流的局部状态进行流敏感分析时，累积有关全局变量的流不敏感信息 [49]。在存在动态线程创建的情况下，我们在此使用副作用来表达：可观察动作、解锁或返回也应当 同样贡献到集合 $\mathcal{T}[a]$ 或 $\mathcal{T}[i]$，以便能在相应的观察动作处将其纳入。我们具体语义的带副作用表述如下：

$$(\eta, \eta\,[u_0]) \sqsupseteq (\{[a] \mapsto \mathsf{init} \mid a \in \mathsf{M}\}, \mathsf{init}) \quad (\eta, \eta\,[u']) \sqsupseteq [\![u, \mathsf{act}]\!]\eta \;\; (u, \mathsf{act}, u') \in \mathcal{E} \quad (3)$$

其中，$\sqsupseteq$ 由超集序诱导，右端项定义见图3。右端项接受系统未知量的一个赋值 η，并返回一对值(η', T)，其中 T是对出现在左侧的未知量的贡献（与普通约束系统中的情况相同）。第一个分量收集副作用，形式为赋值 η'。如果右端项是单调的，方程（3）有唯一的最小解。

我们仅详细说明线程创建的右侧表达式，以及新的汇合和返回动作；其余部分与 Schwarz 等人 [48]的定义保持一致。对于**线程创建**，他们提供了动作 x=create(u_1)。这里，u_1 是被创建线程应当开始执行的程序点。我们假设

$[\![u,$ 锁定(a)$]\!]\,\eta = (\emptyset, [\![e]\!](\eta\,[u], \eta\,[a]))$	$[\![u, x$=创建(u_1)$]\!]\,\eta =$ **令** $T = [\![e]\!](\eta\,[u])$ **在** $(\{[u_1] \mapsto$ 新建 $u\,u_1\,(\eta\,[u])\}, T)$
$[\![u,$ 解锁(a)$]\!]\,\eta =$ **令** $T = [\![e]\!]\,(\eta\,[u])$ **其中** $(\{[a] \mapsto T\}, T)$	$[\![u,\ x$=汇合(x')$]\!]\,\eta =$ **令** $T = \eta\,[u]$ **在** $(\emptyset, [\![e]\!](\eta\,[u], \bigcup\{\eta\,[t(x')] \mid t \in \eta\,[u]\}))$
$[\![u, x = g]\!]\,\eta = (\emptyset, [\![e]\!](\eta\,[u]))$	$[\![u,\ $ 返回 $x]\!]\,\eta =$ **令** $T = \eta\,[u]$ **在** $(\{[i] \mapsto [\![e]\!](\{t \in T \mid t(\mathsf{self}) = i\}) \mid i \in \mathcal{I}\}, [\![e]\!]T)$
$[\![u, g = x]\!]\,\eta = (\emptyset, [\![e]\!](\eta\,[u]))$	

图 3：具体语义的副作用式表述中的右端项；$t(y)$ 从轨迹 t 的终止配置中提取局部变量 y 的值。

创建线程的所有局部变量都会传递给被创建线程，self 变量除外。被创建线程中的 self 变量和创建线程中的 x 会获得一个新的线程标识。这里，new $u\, u_1\, t$ 根据创建线程的局部轨迹 t，计算起始点 u_1 处的局部轨迹。为了处理线程的**返回**和**汇合**，我们引入以下两个动作：

– 返回 x；– 终止一个线程，并将局部变量 x 的值返回给等待指定线程终止的线程。– x=join(x')；其中 x' 是保存线程 id 的局部变量——阻塞自身线程，直至具有指定线程 id 的线程终止。与*pthreads* 中一样，针对给定线程 id，最多只能有一个线程调用 join。被汇合线程通过 return 提供的值会赋给局部变量 x。

为返回结果并实现 join，我们使用未知量 $[i]$ 表示返回线程的线程 id i，如图3所示。

4 将关系分析作为局部轨迹的抽象

接下来，我们基于局部轨迹语义，对全局变量的值进行关系分析。这些分析适用于关系域 $\mathcal{R}$；其中，2-可分解域尤其适用，因为簇的概念*clusters*是这些分析的核心。我们重点关注由某个互斥锁共同*write*保护的全局变量之间的关系。假设对于每个全局变量 g，我们都给定一个由（写入）保护互斥锁组成的集合 $\mathcal{M}[g]$，即在写入 g 时*always* 始终持有的互斥锁。令 $\mathcal{G}[a] = \{g \in \mathcal{G} \mid a \in \mathcal{M}[g]\}$ 表示由互斥锁 a 保护的全局变量集合。令 $\emptyset \neq \mathcal{Q}_a \subseteq 2^{\mathcal{G}[a]}$ 表示我们为 a 关联的这些全局变量的簇集合。出于技术原因，我们要求每个互斥锁 a 至少对应一个簇；该簇可以是空簇 $\emptyset$，因此不与 a 关联任何信息。

我们的基础思路是在未知量 $[a, Q]$（针对每个互斥锁 a 和簇 $Q \in \mathcal{Q}_a$）中存储仅描述 Q 中全局变量之间关系的抽象。根据构造，Q 中的所有全局变量都受 a 保护。每当它被锁定时，存储在所有 $[a, Q]$ 中的关系信息都会通过格运算交并入局部状态，即此时局部状态不仅维护局部变量之间的关系也维护全局变量之间的关系，而在这个程序点，其他线程都无法访问这些全局变量。每当 a 解锁时，所有相应簇 $Q \in \mathcal{Q}_a$ 中全局变量之间的新关系都会以副作用更新写入相应的未知量 $[a, Q]$。与此同时，不再受保护的全局变量的所有信息都会被遗忘，以得到新的局部状态。这样一来，分析在局部状态中完全采用关系分析，同时只保留由某个互斥锁共同保护的全局变量簇内的关系。

为清晰起见，在到达程序点时，我们对持有的互斥锁集合执行控制点拆分。除此之外，分析所用的约束系统和右端项与具体语义（第3节）中的基本一致，唯一例外是：现在未知量取值自$\mathcal{R}$，并且针对 $Q \in \mathcal{Q}_a$，未知量 $[a]$ 被未知量 $[a, Q]$ 替代。

所有右端项均在图 4 中详细给出。对于程序的**起始点**和空锁集，右端项 init$^\sharp$ 返回 $\top$ 关系

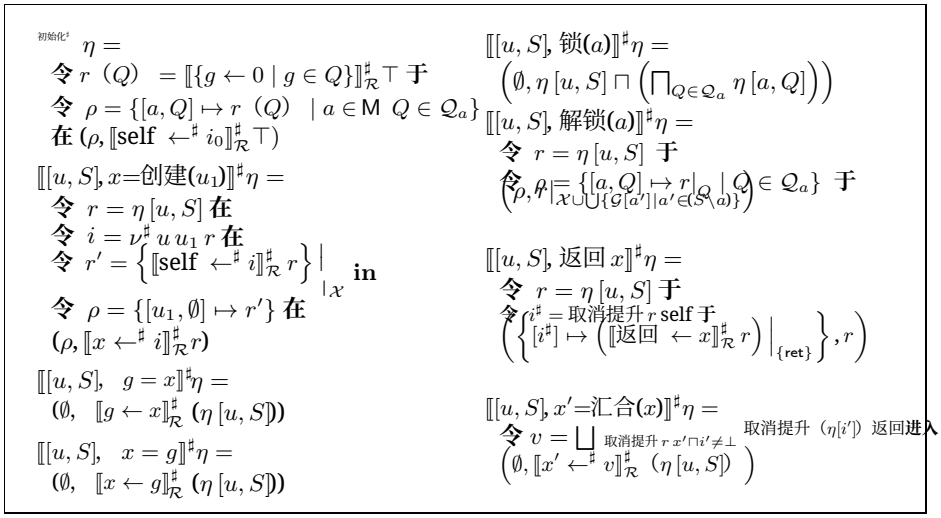


图4：基本分析的右端项。所有函数在 $\bot$ （描述空局部轨迹集合）上都是严格的；此处我们仅显示非-$\bot$抽象值的定义。 $[\![\{g \leftarrow 0 \mid g \in Q\}]\!]^\sharp_{\mathcal{R}}$ 是逐个将 Q 中的所有变量赋值为0所对应的抽象转换器的简写。

已更新，使变量 self 持有 main 线程的抽象线程*ID* i_0 。此外，init$^\sharp$ 会为每个互斥锁 a 和簇 Q产生副作用，将簇中的所有全局变量初始化为值 0。

对于一个从程序点 u 、锁集为 S的**创建线程**边开始的线程，右端项 $[\![u, S]$, x=create(u_1)$]\!]^\sharp$ 首先生成一个新的抽象线程id，我们假设可以使用函数 $\nu^\sharp$计算该线程。新的id 被赋给当前线程局部状态中的变量 x 。此外，还会构造新创建线程的初始状态 r' ，并将其以空锁集 $[u_1, \emptyset]$副作用更新到该线程的起始点。由于线程以空锁集启动，因此状态 r' 是通过从创建线程的局部状态中移除所有关于全局变量的信息，并将新的抽象线程id 赋给变量self而获得的。

当**对互斥锁** a**加锁**时，存储在未知量 $[a, Q]$ 中的状态会与 $Q \in \mathcal{Q}_a$通过交运算同局部状态合并。这是可靠的，因为任意 $[a, Q]$ 中存储的值只维护由 a写保护的变量之间的关系，而这些值能够可靠地反映每次解锁(a)时以及程序启动时的程序状态。另一方面，当**解锁** a时，将限制在相应簇 $Q \in \mathcal{Q}_a$ 上的局部状态副作用更新到相应的未知量 $[a, Q]$中，使簇内变量所做的更改对其他线程可见。此外，局部状态会限制为局部变量，以及至少仍有一个保护互斥锁处于加锁状态的全局变量。

由于使用紧邻 g 访问前后的特殊互斥锁 m_g 来确保原子性，并且与 g 相关的信息会与这些互斥锁关联，因此所有 **读操作**

和 **写操作** 都指向 g 的本地副本。**守卫** 和 **赋值** （其中可能只涉及局部变量）的定义与此类似。

对于**返回**边，要返回的抽象值会从局部状态中查找，然后作为当前线程的专用变量 ret 的值，副作用更新到当前线程的抽象标识中。对于**汇合**边，所有可能被汇合的线程的返回值的最小上界会赋给局部状态中汇合语句的左侧。

示例 *4*。考虑程序[4]，其中 $\mathcal{M}[g] = \{a, b, m_g\}$、 $\mathcal{M}[h] = \{a, b, m_h\}$、$\mathcal{Q}_a = \{\{g, h\}\}$、 $\mathcal{Q}_b = \{\{g, h\}\}$。

main：t1：t2：x = 创建(t1)；y =?；锁定(b)；锁定(b)；锁定(a)；锁定(b)；解锁(b)；锁定(a)；g = y；h = y+9；锁定(a)；// 断言(g==h)；(4) 解锁(b)；锁定(b)；锁定(b)；解锁(a)；h = y；// 断言(g==h)；(3) 解锁(b)；// 断言(g==y)；(1) y = ?；g = y；h = y；// 断言(h==y)；(2) 解锁(b)；解锁(b)；解锁(a)；解锁(a)；x = 创建(t2)；

我们的分析成功证明了此处的所有断言。线程 t_2 尤其值得关注：加锁 b 时，只能确定 $g \leq h$ 成立；而再加锁互斥锁 a 后，就能获得更精确的信息 $g = h$。另一方面，Mukherjee 等人 [42] 的分析只能成功证明断言 (2)——即使将所有全局变量都放入同一个区域也不例外。它无法证明 (1)，因为在主线程的第二次加锁(b)处应用混合 操作时，局部变量与全局变量之间的所有相关性都会丢失。无法证明 (3)，因为在 t_1中的加锁(b)处，混合操作还会合并主线程第一次解锁(b)后的状态，而在该状态下 $g = h$ 并不成立。(4)也同样如此。断言 (3)以及使用 Miné [39] 提出的锁不变式 的分析也存在同样的问题。该分析也无法证明 (1)，因为在主线程的加锁(b)处，与 b 相关联的锁不变式会被汇合到局部状态中。(4)同样无法证明。为这些分析添加线程编号后， [39, 42, 48] 也同样适用上述推理。□

定理 1. 约束系统的任何解相对于局部轨迹语义都是可靠的。

证明. 证明采用不动点归纳，详情见附录 B。

我们指出，与其依赖预先计算好的 $\mathcal{M}[g]$，分析也可以在运行过程中推断这项信息 [56]。例如，Goblint 能够处理由此产生的副作用非单调性。

[4] 在所有示例中， g、 h和 i 是全局变量，而 x、 y和 z 是局部变量；特殊互斥锁 m_g 处的簇恰好包含全局变量 g：$\mathcal{Q}_{m_g} = \{\{g\}\}$。除非另有明确说明，否则使用例 2 中的域 $\mathcal{R}_1$，并以变量间不等式对其进行扩展。

然而，这里描述的分析仍存在一些不足。所有对全局变量的写操作都会被累积，而不考虑执行写操作的线程。因此，例如，线程读取的不仅是它最近的本地写操作，还包括所有更早的本地写操作，即使这些写操作已经确定 被覆盖。排除某些线程的写操作，是排除不可能成为最后一次写入的写操作这一更一般思路的一个实例。下一节中，我们不再讨论解决这一具体缺陷的临时方法，而是提出一种通用机制，以提高任何线程模块化分析的精度，之后再将其应用于此处强调的问题。

5 通过局部轨迹的有限抽象进行细化

为了提高线程模块化分析的精度，我们将局部轨迹的额外抽象纳入考量。我们的方法具有通用性：它以基础分析的右端项为基础，并将其视为黑盒。稍后，我们将在第4节中实例化这一框架，以排除基于线程id的写操作。也可以设想其他实例化方式。为了使该框架具有广泛适用性，它允许基础分析已在程序点对未知量进行某种拆分（例如第4节中的锁集）。对于程序点 u，我们用 $\lceil u \rceil$ 表示这类（可能经过扩展的）未知量。

（基础）分析由其右端项以及一组不同的域定义：（1） $\mathcal{D}_S$，用于存储程序点未知量的抽象值；（2）$\mathcal{D}_{\mathsf{act}}$，用于存储可观察动作 act 的抽象值（例如在第4节中， $\mathcal{D}_M$ 用于解锁， $\mathcal{D}_T$ 用于线程返回）。

设 $\mathcal{A}$ 为一组有限信息，这些信息可由函数 $\alpha_{\mathcal{A}} \colon \mathcal{T} \to \mathcal{A}$从局部轨迹中提取。我们称 $\alpha_{\mathcal{A}}\, t \in \mathcal{A}$ 为某个局部轨迹 t的摘要。设 $[\![u,\ \mathsf{act}]\!]^\sharp_{\mathcal{A}} \colon \mathcal{A}^k \to 2^{\mathcal{A}}$ 表示对源自 u的控制流边执行 k元动作 $\mathsf{act} \in \mathcal{A}\mathsf{ct}$ 时对摘要产生的影响。我们要求对于 $e = (u, \mathsf{act}\ v) \in \mathcal{E}$,

$$\begin{array}{l} \forall A_0, \ldots, A_{k-1} \in \mathcal{A} \colon |[\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0, \ldots, A_{k-1})| \leq 1 \\ \forall t_0, \ldots, t_{k-1} \in \mathcal{T} \quad : \alpha_{\mathcal{A}}([\![e]\!](t_0, \ldots, t_{k-1})) \subseteq [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(\alpha_{\mathcal{A}}\, t_0, \ldots, \alpha_{\mathcal{A}}\, t_{k-1}) \end{array} \tag{4}$$

其中 $\alpha_{\mathcal{A}}$ 按元素方式提升到集合上。虽然第一个限制确保了确定性，但第二个限制直观地确保了 $[\![u,\ \mathsf{act}]\!]^\sharp_{\mathcal{A}}$ 能够对 $[\![e]\!]$进行可靠的抽象。

对于线程创建，我们还要求一个辅助函数 $\mathsf{new}^\sharp_{\mathcal{A}} : \mathcal{N} \to \mathcal{N} \to \mathcal{A} \to \mathcal{A}$，对于在源自 u 的边上创建、并从程序点 u_1 开始执行的线程，该函数返回新的摘要。对于边 $(u, x{=}\mathsf{create}(u_1)\ v) \in \mathcal{E}$，同样施加这些要求，

$$\forall A_0 \in \mathcal{A} : |\mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, A_0| \leq 1 \quad \forall t_0 \in \mathcal{T} : \alpha_{\mathcal{A}}(\mathsf{new}\, u\, u_1\, t) \subseteq \mathsf{new}^\sharp_{\mathcal{A}}\, u\, u_1\, (\alpha_{\mathcal{A}}\, t_0) \tag{5}$$

此外，我们为程序开始时的初始摘要定义

$$\mathsf{init}^\sharp_{\mathcal{A}} = \{\alpha_A\, t \mid t \in \mathsf{init}\} \tag{6}$$

在这些假设下，我们可以根据$\mathcal{A}$进行控制点拆分 。这意味着，程序点 u 对应的未知量 $\lceil u \rceil$ 将被替换为新的

$[\![[\hat{u}, A_0]$, 执行 $A_1]\!]^\sharp\,\eta =$
  令 $\eta'\,[x] =$ **如果** $[x] = [\hat{u}]$ **则**
    $\eta\,[\hat{u}, A_0]$
  否则 $\eta\,[x, A_1]$
  in
  $[\![\hat{u}]$, 执行$]\!]^\sharp\,\eta'$

$[\![[\hat{u}, A_0]$, 动作″$]\!]^\sharp\,\eta =$
  令 $\eta'\,[x] = \eta\,[x, A_0]$ 为.....
  $[\![\hat{u}]$, 动作″$]\!]^\sharp\,\eta'$

$[\![[\hat{u}, A_0]$, 动作′, $A']\!]^\sharp\,\eta =$
  令 $\eta'\,[x] = \eta\,[x, A_0]$ 为.....
  令 (ρ, v) $= [\![[\hat{u}]$, 动作′$]\!]^\sharp\,\eta'$.....
  令 $\rho' = \{[x, A'] \mapsto v' \mid ([x] \mapsto v') \in \rho\}$ 于
  (ρ', v)

$[\![[\hat{u}, A_0]$, x=创建$(u_1)]\!]^\sharp\,\eta =$
  令 $\eta'\,[x] = \eta\,[x, A_0]$ 于
  令$(\{[\hat{}u_1] \mapsto v'\}, v)$ $= [\![[\hat{u}]$, x=创建$(u_1)]\!]^\sharp\,\eta'$ 于
  $(\{[\hat{}u_1, A'] \mapsto v' \mid A' \in$ 新建$^\sharp_{\mathcal{A}}\,u\,u_1\,A_0\}$, $v)$

图5：细化分析中观察动作 act、可观察动作 act′、创建动作，以及既非观察动作也非可观察动作的动作 act″ 的右端项；这些分析被定义为基础分析右端项的包装器。

未知量 $[\hat{}u, A]$, $A \in \mathcal{A}$。类似地，可观察动作 [act] 对应的未知量将被替换为 $A \in \mathcal{A}$对应的未知量 [act A] 。考虑上一节抽象约束系统中的一个约束，它对程序的收集式局部轨迹语义进行了可靠的抽象。

$$(\eta, \eta\,[\hat{v}]) \sqsupseteq [\![[\hat{u}], \mathsf{act}]\!]^\sharp\,\eta$$

带有控制点拆分的细化系统的相应约束，取决于动作 act 是观察动作、可观察动作还是两者都不是。

– 当 act 是观察动作时，新的右端项还会获得与待纳入的局部轨迹相关联的摘要 A_1：

$$(\eta, \eta\,[\hat{v}, A']) \sqsupseteq [\![[\hat{u}, A_0], \mathsf{act}, A_1]\!]^\sharp\,\eta \qquad \text{for } A_0, A_1 \in \mathcal{A}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}\,(A_0, A_1)$$

– 当 act 是可观察动作时，所得局部轨迹的摘要 A' 会传递给新的右端项，从而使副作用能够重定向到适当的未知量：

$$(\eta, \eta\,[\hat{v}, A']) \sqsupseteq [\![[\hat{u}, A_0], \mathsf{act}, A']\!]^\sharp\,\eta \qquad \text{for } A_0 \in \mathcal{A}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}\,(A_0)$$

– 当 act 为 neither 时，不会传递额外的摘要：

$$(\eta, \eta\,[\hat{v}, A']) \sqsupseteq [\![[\hat{u}, A_0], \mathsf{act}]\!]^\sharp\,\eta \qquad \text{for } A_0 \in \mathcal{A}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}\,(A_0)$$

新的右端项是根据基础分析的右端项定义的，这些右端项被用作黑盒（图 5）。它们充当包装器，将原始分析中任何被查询或副作用更新的未知量映射到细化系统中相应的未知量。因此，精化分析会自动受益于摘要提供的额外信息。例如，它可以利用 $[\![u$、act$]\!]^\sharp_{\mathcal{A}}(A_0, A_1) = \emptyset$，这意味着，摘要为 A_0、A_1 的局部轨迹无法组合成一条以动作 act 结束的有效局部轨迹。

针对本文所考察的动作以及附加了锁集的程序点未知量实例化后的精化约束系统的完整定义，可在补充材料中找到（附录 C 中的图 14）。

使用锁集丰富程序点，实际上可以视为该框架的首次应用。右端项见图 6。

例 *5.* 作为另一个实例，考虑跟踪局部轨迹中哪些互斥锁至少被加锁过一次。在 lock(a) 处，线程执行过 lock(a) 的轨迹，不能与不包含对 a 执行任何加锁操作的轨迹合并。相应的右端项见图 7。

相应地对第4节的分析进行细化（假设 a 保护 g 和 h）后，该分析能够证明此程序中的断言，因为可以排除 g 和 h 的初始值为 0 的情况。

main:
```
加锁(a);
h = 9; g = 10;
解锁(a);
x =创建(t1);
```

t1:
```
x = 创建(t2);
锁(a);
h = 11; g = 12;
解锁(a);
```

t2:
```
锁(a);
// 断言(h<=g);
解锁(a);
```

这可以很自然地推广，以构造用于统计某个动作（例如，对全局变量 g的写入）发生频率的实例，并在达到某个常数时停止精确记账（本例中为 1）。□

为了证明第 4 节中基于局部轨迹的细化对我们的分析具有可靠性，我们首先构造相应的精化收集式局部轨迹语义。随后，我们验证精化分析相对于该精化语义是可靠的；而该精化语义相对于原始收集式局部轨迹语义的可靠性也已得到证明。

定理 2. 假设 $\alpha_{\mathcal{A}}$、新建$^{\sharp}_{\mathcal{A}}$ 和 $[\![u,$ 操作$]\!]^{\sharp}_{\mathcal{A}}$ 满足要求 (4)、(5) 和 (6)。那么，任何精化约束系统的解相对于收集式局部轨迹语义都是可靠的。

证明。 附录 D 给出了一个证明概要，其中实例化了本文所考虑的动作，以及通过锁集增强的程序点上的未知量。

6 线程标识符分析与唯一性

$^{\text{初始化}^{\sharp}}_{\mathcal{A}} = \{\emptyset\}$	$[\![u,$ 锁$(a)]\!]^{\sharp}_{\mathcal{A}}\,(S, S') = \{S \cup \{a\}\}$
$\text{new}^{\sharp}_{\mathcal{A}}\, u\, u_1\, S = \{\emptyset\}$	$[\![u,$ 解锁$(a)]\!]^{\sharp}_{\mathcal{A}}\, S = \{S \setminus \{a\}\}$
$[\![u, a]\!]^{\sharp}_{\mathcal{A}}\, S = \{S\}$ (其他非观察)	$[\![u, a]\!]^{\sharp}_{\mathcal{A}}\,(S, S') = \{S\}$ (其他观察)

图 6：将锁集表示为细化的右端项。

我们实例化上一节的方案，以计算抽象线程标识及其唯一性。这种对基础分析的细化通过排除来自尚未启动的线程的读操作等情况，提高了分析的精度。为此，我们根据线程创建历史来识别线程，即通过创建 边的序列来识别。由于这些序列可能无限增长，我们将首次重复之后发生的所有创建操作收集到一个集合 中，以获得有限抽象。

示例 *6.* 在图 8 的程序中，由 $main$ 创建的第一个线程会收到抽象线程 id(main $\cdot\langle u_1, t_1\rangle, \emptyset$)。它会创建一个抽象线程$id$(main $\cdot\langle u_1, t_1\rangle \cdot \langle u_3, t_1\rangle, \emptyset$)的线程。在程序点 u_3，后者会创建一个从 t_1 开始并收到抽象线程 id(main $\cdot\langle u_1, t_1\rangle$, $\{\langle u_3, t_1\rangle\}$) 的线程——此边上之后创建的所有线程也都如此。□

然而，在创建线程中，创建边也可能在循环内被反复遇到。为处理这种情况，我们会针对每个线程跟踪集合 C，其中记录可能已经遇到过的 创建 边。一旦再次遇到 创建 边，被创建线程就会收到一个非唯一线程 id。

示例 *7.* 主线程在图 8 的程序中首次到达程序点 u_2 时，被创建线程会被赋予唯一的抽象线程id(main $\cdot\langle u_2, t_1\rangle, \emptyset$)。在后续的循环迭代中，不再将创建的线程分别区分，因此它们会收到非唯一的 id(main $\{\langle u_2, t_1\rangle\}$)。□

形式化地，令 $\mathcal{N}_C, \mathcal{N}_S$ 分别表示这样一些程序点的子集：其出边标记为 $x=$ create(...)；以及线程的起始点。令 $\mathcal{P} \subseteq \mathcal{N}_C \times \mathcal{N}_S$ 表示一些数对的集合，这些数对将线程创建节点与被创建线程的起始点关联起来。抽象线程id的集合 $\mathcal{I}^\sharp$ 由所有数对 $(i, s) \in$ (main $\cdot \mathcal{P}^*) \times 2^{\mathcal{P}}$ 组成，其中每个数对 $\langle u, f\rangle$ 至多出现一次。给定集合 $\mathcal{I}^\sharp$，我们要求存在一个具体化 $\gamma\colon \mathcal{I}^\sharp \to 2^{\mathcal{I}}$ 以及一个函数 single : $\mathcal{I}^\sharp \to \mathcal{V}_{\mathcal{I}}^\sharp$，满足 $\gamma\, i^\sharp \subseteq \gamma_{\mathcal{V}^\sharp}$(single $i^\sharp$)。$main$线程的抽象线程id 由 (main $\emptyset$) 给出。其中，(main $\cdot \mathcal{P}^*) \times \{\emptyset\}$ 中的元素表示至多对应一个具体线程id的唯一 线程id，而 (i, s)，$s \neq \emptyset$ 中的元素则具有歧义，即可能对应多个具体线程id。此外，我们认为 $\mathcal{I}^\sharp$ all 中不同抽象线程id的具体化彼此不相交。

作为精化信息，我们不仅考虑抽象线程id，还会跟踪当前线程中已执行的线程创建集合。因此，我们设定 $\mathcal{A} = \mathcal{I}^\sharp \times 2^P$，并定义右端项，如图 9 所示，¯ 其中 i 表示序列 i中出现的数对集合。

初始化$^\sharp_{\mathcal{A}} = \{\emptyset\}$

新建$^\sharp_{\mathcal{A}}\, u\, u_1\, L = \{L\}$

$[\![u, a]\!]^\sharp_{\mathcal{A}}\, S = \{L\}$ (其他非观察)

$$[\![u, \text{锁}(a)]\!]^\sharp_{\mathcal{A}}(L, L') = \begin{cases} \emptyset & \text{如果 } a \in L \wedge a \notin L' \\ \{L \cup L' \cup \{a\}\} & \text{否则} \end{cases}$$

$[\![u,a]\!]^\sharp_{\mathcal{A}}(L, L') = \{L \cup L'\}$ (其他观察)

图 7：根据遇到的加锁操作进行细化的右端项。

示例 *8.* 再次考虑图 8 中的程序，其右端项如图 9 所示，并假设 join 缺失的右端项返回其第一个参数。初始线程的抽象线程id $i_0 =$ (main $\emptyset$)。在其起始点，摘要因此为 $(i_0, \emptyset)$。在从 u_1出发的创建边上，会创建一个具有id(main $\cdot \langle u_1, t_1 \rangle$, $\emptyset$)的新线程。该线程的摘要于是为 ((main $\langle u_1, t_1 \rangle, \emptyset$) $\emptyset$)。对于主线程，遇到的创建边 $\langle u_1, t_1 \rangle$ 会被加入摘要的第二个分量，使其变为 $(i_0, \{\langle u_1, t_1 \rangle\})$。

当以（i_0, $\{\langle u_1, t_1 \rangle\}$）到达 u_2时，会创建一个具有 id（main $\langle u_2$, $t_1 \rangle$, $\emptyset$）的唯一线程。随后，创建线程的新摘要为（i_0, $\{\langle u_1, t_1 \rangle, \langle u_2$, $t_1 \rangle\}$）。在后续循环迭代中，当以（i_0, $\{\langle u_1, t_1 \rangle, \langle u_2, t_1 \rangle\}$）到达 u_2 时，会创建一个具有 id（main $\{\langle u_2, t_1 \rangle\}$）的非唯一线程。

当以 id（main $\{\langle u_2, t_1 \rangle\}$）到达 u_3时，会创建一个具有 id（main $\{\langle u_2, t_1 \rangle$, $\langle u_3$, $t_1 \rangle\}$）的线程，因为创建线程的 id已经不是唯一的。当以 id（main $\cdot \langle u_1$, $t_1 \rangle$, $\emptyset$）到达该程序点时，会创建一个具有 id（main $\cdot \langle u_1$, $t_1 \rangle \langle u_3$, $t_1 \rangle$, $\emptyset$）的新线程。当新创建的线程到达该程序点时，在那里创建的线程具有 非唯一 *id*（main $\cdot \langle u_1$, $t_1 \rangle$, $\{\langle u_3, t_1 \rangle\}$），因为$\langle u_3$, $t_1 \rangle$ 已经出现在创建线程的 id 中。□

抽象线程id应为我们提供以下函数

– unique : $\mathcal{I}^\sharp \to$bool 用于判断线程id 是否唯一。– lcu_anc : $\mathcal{I}^\sharp \to \mathcal{I}^\sharp \to \mathcal{I}^\sharp$ 返回两个线程的最近共同唯一 祖先。– may_create : $\mathcal{I}^\sharp \to \mathcal{I}^\sharp \to$bool 用于检查一个线程是否可能（传递地）创建另一个线程。

对于我们的域 $\mathcal{I}^\sharp$，这些函数可以定义为 unique $(i, s) = (s = \emptyset)$ 和

$$\begin{aligned} &\text{lcu_anc}\,(i,s)\,(i',s') &&= (\text{longest common prefix}\ i\,i', \emptyset) \\ &\text{may_create}\,(i,s)\,(i',s') &&= (\bar{i} \cup s) \subseteq (\bar{i'} \cup s') \end{aligned}$$

我们利用这些额外信息来完善 $[\![u,\text{lock}(a)]\!]^\sharp_{\mathcal{A}}$ and$[\![u, x'{=}\text{join}(x)]\!]^\sharp_{\mathcal{A}}$ 的定义，考虑到自身线程无法获取另一个线程的互斥锁，也无法汇合一个确定尚未创建的线程。对于以下线程，情况便是如此 t'

(1) 由唯一 自身线程直接创建，但自身线程尚未到达创建 t' 的程序点；(2) 其线程id 表明，根据 (1)，尚未创建的某个线程属于 t' 的创建历史。

```
main: t1:x  = g; // PP u1 g  = 42; // PP u3y  = 创建
(t1); y  =创建(t1);循环(i  = 0; i  < 5; i++) { // PP
u2z = 创建(t1); }
```

图 8：包含多个线程创建的程序。

因此，我们引入谓词 may_run (i, C) (i', C')，定义如下

$$(\mathsf{lcu_anc}\, i\, i' = i) \implies \exists \langle u, u' \rangle \in C : (i \circ \langle u, u' \rangle = i' \vee \mathsf{may_create}\, (i \circ \langle u, u' \rangle)\, i')$$

当线程 i' 确定尚未启动时，该条件为假。然后我们设置

$$\begin{aligned} [\![u, \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}\, (i, C)\, (i', C') &= [\![u, x{=}\mathsf{join}(x)]\!]^\sharp_{\mathcal{A}}\, (i, C)\, (i', C') \\ &= \begin{cases} \{(i, C)\} & \text{if } \mathsf{may_run}\, (i, C)\, (i', C') \\ \emptyset & \text{otherwise} \end{cases} \end{aligned}$$

这种对线程标识符及唯一性的分析可视为一种可能并行发生（或者更准确地说，必定不并行发生）分析。MHP 信息在多种场景中都很有用：例如，尚未考虑线程标识符和线程汇合的数据竞争或死锁线程模块化分析，可以借助给定的线程标识符分析进行改进，从而排除原始分析无法排除的一些数据竞争或死锁。随后，我们概述第4节中的分析如何受益于 MHP 信息。

7 利用线程标识符改进关系分析

随后，我们利用抽象线程标识符及其唯一性，限制第4节分析所执行的读操作数量。

I1 来自尚未创建的其他线程。**I2** 自身线程过去的写操作，前提是其线程标识符唯一。**I3** 来自已汇合线程的过去写操作。

例如，在一种设置中，改进**I1** 和**I3** 已经实现：线程id以及在哪些位置汇合了哪个线程，都可以从控制流图中读出 [31]。然而，这里是在分析过程中 计算这些信息的。在我们的框架中，只需根据第 6 节细化基础分析，即可实现**I1** 。

示例 *9*。考虑下面的程序，其中 $\mathcal{M}[g] = \{a, b, m_g\}$、$\mathcal{M}[h] = \{a, b, m_h\}$、$\mathcal{M}[i] = \{m_i\}$，并假设 $\mathcal{Q}_a = \{\{g, h\}\}$。

$\overset{\text{初始化}^\sharp}{\mathcal{A}} = \{\ (\text{main}\ \emptyset)\ \ \emptyset)\ \}$

$[\![u, x{=}\text{创建}(u_1)]\!]^\sharp_{\ \mathcal{A}}\ (i, C) = \{\ (i, C \cup \{\langle u, u_1 \rangle\})\ \}$

$[\![u, a]\!]^\sharp_{\mathcal{A}}\, (i, C) = \{(i, C)\}$　　（其他动作 a）

新建$^\sharp_{\mathcal{A}}\, u\, u_1\, ((d, s)\ C) =$
　令 $(d', s') = (d, s) \circ \langle u, u_1 \rangle$ **于**
　如果 $s' = \emptyset \wedge \langle u, u_1 \rangle \in C$ **则** $((d, \{\langle u, u_1 \rangle\})\ \emptyset)$
　否则 $((d', s')\ \emptyset)$

$(d, s) \circ \langle u, u_1 \rangle =$
　如果 $d = (d_0 \cdot \langle u, u_1 \rangle) \cdot d_1$ **则**
　　$(d_0, s \cup d_1 \cup \{\langle u, u_1 \rangle\})$
　否则如果 $s = \emptyset$ **则** $(d \cdot \langle u, u_1 \rangle, \emptyset)$
　否则 $(d, s \cup \{\langle u, u_1 \rangle\})$

图 9：线程id的右端项。

```
main: x =创建(t1); 锁(a); //断言(g==h); (1)解锁(a); y =创建(t2); 锁(a); //断言(g==h); (2)g = 42; h = 42; 解锁(a); z =创建(t3); i = 3; i = 2; // 断言(i==2); (3)i = 8;

t1:锁(a);r = ?; g = r;h = r;解锁(a);t2:锁(a); v = g; 解锁(a);t3:锁(a); g = 19; 解锁(a);
```

分析成功证明了断言 (1)，因为在此程序点，尚未启动线程（从） t_3 会破坏不变式 $g{=}h$ 。如果不进行细化，第4节中的分析就无法证明 (1)。然而，仅凭这一点还不足以证明 (2)。在此程序点，线程t_2 可能已经启动。在 t_2 中的锁(a)处，线程 t_3 也可能已经启动；因此， t_3 对不变式 $g{=}h$ 的破坏会被纳入 t_2 在该锁处的局部状态。在解锁(a)时，尽管 t_2 只读操作 g，但由于 $g{=}h$ 不成立，精度不足的抽象关系会被副作用更新到 $[a, \{g, h\}, t_2]$，并在主线程的第二个锁(a)处纳入其中。最后一个不足之处在于，每个线程都会读取自身过去（甚至未来！）的所有写操作——即使已知这些写操作是唯一的。这意味着无法证明 (3)。□

为了实现**I2**，需要付出一些努力，因为当全局变量不再受保护时，我们的分析会忘记它们的值。这与 [39, 42] 等方法形成对比。因此，我们仅在自身线程自获取互斥锁以来可能写入过受保护的全局变量时，才将副作用归于该互斥锁。这与第4节形成对比：在那里，每次解锁都会产生副作用，也就是说，线程读到的所有内容都被视为由该线程写入。

具体而言，我们会在本地跟踪一个映射 $L : (\mathbb{M} \times \mathcal{Q}) \to \mathcal{R}$，其中 $L(a, Q)$ 为互斥锁 a 维护簇 $Q \in \mathcal{Q}_a$ 中全局变量之间的抽象关系。更具体地说， Q 中全局变量的抽象关系记录在 $L(a, Q)$ 中，该关系是在 a 首次汇合局部 解锁之后、且此前已对 $\mathcal{G}[a]$ 中某个全局变量进行最后一次汇合局部 写入时成立的关系。如果不存在这样的解锁(a)，则记录程序启动时的关系。若局部轨迹中的某项操作满足以下条件之一，我们就称其汇合局部 于自身线程：(a) 该操作是线程局部的，即由自身线程执行；(b) 该操作由（传递地）汇合到自身线程中的线程执行；或者 (c) 该操作在创建自身线程的节点处汇合局部 于父线程。这一概念对于实现**I3**也至关重要。图 10 展示了汇合局部性，其中局部轨迹中汇合局部 的部分以高亮显示。

对于汇合局部贡献，按照构造，只需查阅 $L\,a$ ，而无需查阅未知量 $[a, Q, i]$。这类贡献称为已计入贡献。为了检查来自特定线程id 的贡献是否已计入，我们引入函数 $\mathsf{acc} : (\mathcal{A} \times \mathcal{D}_S) \to \mathcal{A} \to \mathsf{bool}$（见下方定义（7））。除了来自 $\mathcal{R}$ 的抽象值外，局部状态 $\mathcal{D}_S$ 现在还包含两个额外组成部分：

– 映射 L： $(\mathbb{M} \times \mathcal{Q}) \to \mathcal{R}$ 其汇合按分量给出；

– 集合 $W : 2^{\mathcal{G}}$ （按 $\subseteq$ 排序）的全局变量：自其某个保护互斥锁被锁定后可能已被写入，且此后并非所有保护互斥锁都已解锁。

与 r、L 和 W 一样，它们也是可达局部轨迹的抽象。$\mathcal{D}_T$ 还增加了一个 L 组件，而 $\mathcal{D}_M$ 保持不变。我们在此概述右端项，定义见图11。对于**程序启动** $\text{init}^\sharp$，与第4节中的分析不同，互斥锁的未知量不会产生初始副作用。全局变量的初始值会汇合局部，因此也会纳入传递给后续创建的任何线程的 L 组件中。

线程创建 和**返回** 的右端项与第4节中增加线程id的分析相比，仅在处理额外数据结构 L和 W时有所不同。由于线程id在 $\mathcal{A}$组件中被精确跟踪，因此会直接利用这些信息来确定对哪个未知量施加副作用，以及未知量 $[(i, C)]$ 替换未知量 $[i', (i, C)]$。

对于**汇合** 边，如果未考虑线程的返回值，则会将其赋给左端的变量，并将自身线程和已汇合线程的 L 信息合并。另一方面，如果已考虑该返回值，则该线程已经汇合，不能在此再次汇合。每个 (i', C') 都有一个单独的约束，从而考虑所有可能在此汇合的线程。

对于互斥锁的**加锁**，加锁时，如果未记录（i', C'），则将其关于受 a 保护的全局变量的信息，与在 $L(a, Q)$、$Q \in \mathcal{Q}_a$ 中为 a 维护的汇合局部信息进行汇合。随后，通过 $\sqcap$ 将这些关于受 a 保护的全局变量的信息纳入局部状态。对于互斥锁的**解锁**，如果根据 W，自互斥锁加锁以来可能对某个受保护的全局变量执行过写操作，则更新汇合局部信息，并将局部状态限制为 Q，使其产生副作用并更新至适当的未知量 $[a, Q, (i, C)]$，用于 $Q \in \mathcal{Q}_a$。与第4节一样，随后将 r 限制为仅维护局部变量与那些至少仍有一个保护互斥锁被持有的全局变量之间的关系。再次从全局变量中**读取**和向全局变量**写入**都是纯粹的局部操作。为排除自身写入，我们设置

$$\text{acc}\left((i, C), _\right)(i', C') = \text{unique}\, i \wedge i = i' \tag{7}$$

因此，最终的分析会将 **I1** （通过 $[\![...]\!]^\sharp_{\mathcal{A}}$ 第 6 节中定义的内容）以及**I2** （通过 acc）纳入考量。在示例 9 中，现在能够证明所有断言。

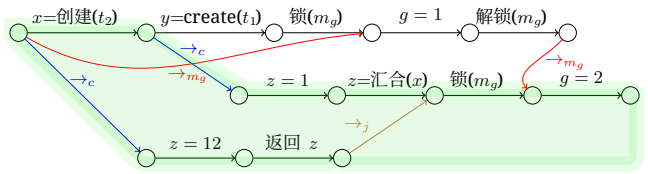


图 10：图示突出显示了图 2a 中程序局部轨迹的汇合局部 部分，以及因此由 L 计入的写操作。

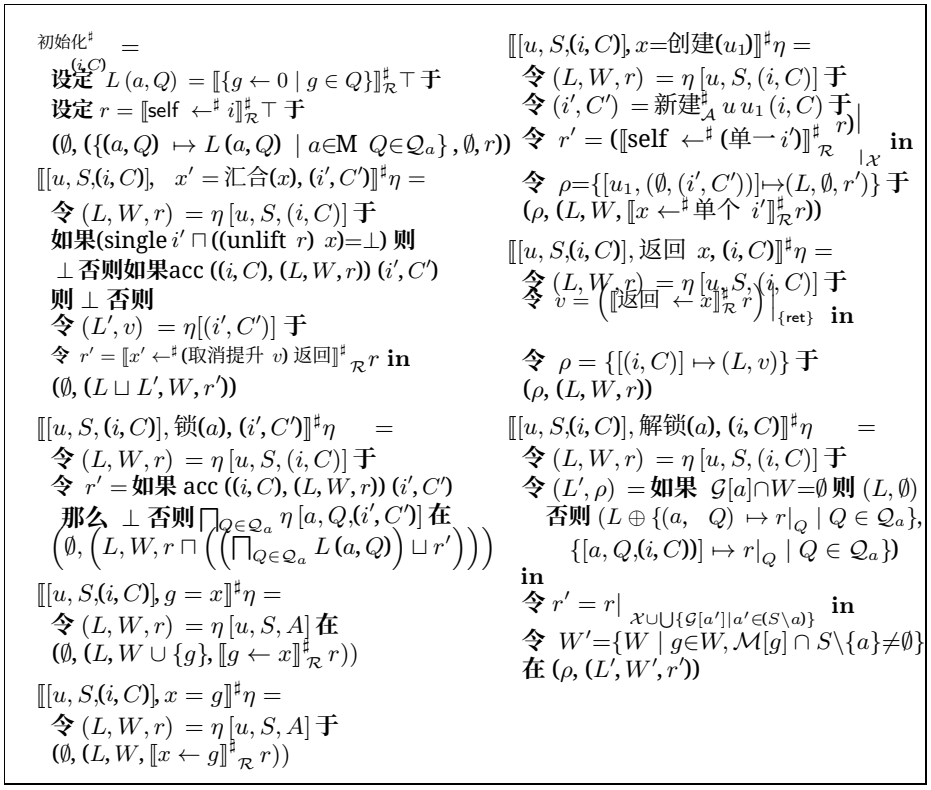


图11：利用线程id改进（**I1**、**I2**）分析的右端项。

定理 3. 相对于局部轨迹语义，此分析是可靠的。

证明。 证明依赖于以下观察：

– 当 $\mathcal{G}[a] \cap W = \emptyset$ 时，无需副作用。– 基于 acc 的排除是可靠的，即它只会排除汇合局部 写操作。

详细证明是对附录 F 中增强分析的正确性证明的简化，我们在附录 G 中概述了该证明。□

该分析未使用未知量 $[a, Q, (i, C)]$ 和$[i, C]$处的组件 C。我们在附录 E 中详细说明了如何利用这些信息排除另一类写操作——即由自身线程的祖先在线程自身创建之前执行的写操作。或者，实现也可以放弃根据互斥锁和线程id处的 C 进行控制点拆分，分别用 $[a, Q, i]$ 和 $[i]$ 替换未知量 $[a, Q, (i, C)]$、$[i, C]$。

转向改进**I3**时，我们观察到，在汇合线程 t及其唯一线程标识后， t 无法再执行写操作。由于汇合线程的所有写操作对于自身线程而言都是汇合局部的，因此无需从

相应的全局变量未知量中读取。因此，我们进一步增强分析，使其也在局部状态中跟踪集合 J：对于这些线程标识，在局部轨迹的汇合局部部分中，肯定已调用过汇合操作；并据此细化 acc：

$$\mathsf{acc}\left((i,C),(J,L,W,r)\right)(i',C') = \mathsf{unique}\, i' \wedge (i = i' \vee i' \in J)$$

关于这项改进的详情，请参见附录 F。

8 利用簇化关系域

直观地看，人们可能会认为，跟踪较大集合的全局变量之间的关系，必然比跟踪较小集合之间的关系更精确。有趣的是，对于我们的分析而言，情况不再如此，例如存在线程id时。在使用数据流图对干扰进行建模的分析中，也观察到了类似现象：关联更多全局变量会降低精度 [19]。

例 *10.* 再来看引言中的例 1，其中包含 $\mathcal{Q}_a = \{\{g,h,i\}\}$。对于该程序，分析的约束系统有唯一的最小解。它验证断言 (1) 成立。它确保对于 $[a,\{g,h,i\},t_1]$，$h = i$ 成立；而对于$main$ 线程以及每个断言之前的程序点，$L\,(a,\{g,h,i\}) = \{g = h, h = i\}$ 成立；对于 $[a,\{g,h,i\},\mathsf{main}]$ 和 $[a,\{g,h,i\},t_2]$，则只记录了 $\top$，对于与 m_g、m_h 或 m_i 相关的任何关系也是如此。然而，断言 (2) 不会通过，因为来自 t_1 的副作用会导致主线程中首次写入的旧值也传播到这些断言，因此，虽然 $h = i$ 得到了证明，$g = h$ 却没有。□

直观地说，分析失去精度的原因在于，解锁互斥锁a时，所有受 a 保护的簇之间当前的关系都会受到副作用更新。一旦某个全局变量被写入，分析就会表现得仿佛所有受保护的全局变量都已被写入。因此，如果发布仅限于至少有一个全局变量被写入的簇，那么其他簇中的信息便可能保持更高的精度，从而有望得到更好的结果。

因此，在改进后的分析中，解锁互斥锁 a时，只会对那些包含至少一个自上次执行 lock(a) 操作以来被写入的全局变量的簇 $Q \in \mathcal{Q}_a$ 产生副作用。加锁和解锁的定义见图 12。

另一方面，对于**对互斥锁 加锁**，要并入局部状态的抽象值由不同线程对各个簇的贡献汇集而成。为此，第 5 节中针对每个允许的摘要所设的独立约束，会合并为一个约束，适用于集合 $\mathrm{I} = \{(i',C') \mid (i,C) \in [\![\mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}((i,C),(i',C'))\}$ 中的所有允许的摘要。这是必要的，因为在 unlock(a) 时，对未受影响簇的副作用已被舍弃，因此一次仅与一个线程的簇值求交是不可靠的。对于每个簇 Q，join 局部信息 $L\,(a,Q)$ 会与尚未计入、但根据摘要获准对 Q 作出贡献的线程的所有贡献进行汇合。这里，不写入 Q 的线程不会产生影响，其值为 $\bot$，因此

$[\![[u,S,(i,C)],\text{解锁}(a),\ (i,C)]\!]^\sharp\eta\ =$
令$(L,W,r)=\eta[u,S,(i,C)]$**其中**
令 $\mathcal{Q}'=\{Q\mid Q\in\mathcal{Q}_a, Q\cap W\neq\emptyset\}$ **其中**
令 $L'=L\oplus\{(a,Q)\mapsto r|_Q\mid Q\in\mathcal{Q}'\}$ **其中**
让 $\rho=\{[a,Q,(i,C)]\mapsto r|_Q\mid Q\in\mathcal{Q}'\}$**在**
令$r'=r|_{\mathcal{X}\cup\bigcup\{\mathcal{G}[a']\mid a'\in(S\setminus a)\}}$ **在**
令 $W'=\{W\mid g\in W,\mathcal{M}[g]\cap S\setminus\{a\}\neq\emptyset\}$ **在**
$(\rho,(L',W',r''))$

$[\![[u,S,(i,C)],\text{锁}(a),\ \mathrm{I}]\!]^\sharp\eta\ =$
令 $(L,W,r)=\eta[u,S,(i,C)]$ **在**
令 $l=((i,C),(L,W,r))$ **于**
令 $J(Q)=\bigsqcup\{\eta[a,Q,(i',C')]\mid (i',C')\in\mathrm{I}\ \neg\text{acc}\ l(i',C')\}$**中**
让 $r'=\bigsqcap_{Q\in\mathcal{Q}_a}(J(Q)\sqcup L(a,Q))$
in
$(\emptyset,(L,W,r\sqcap r'))$

图12：当将副作用限制为可能被写入的簇时，用于解锁和加锁的右端项。

不会影响 Q 的值。最后，使用所得的值通过交运算来改进局部状态。因此，lock(a) 右侧利用了摘要和谓词 acc 提供的细粒度、按簇划分的 MHP 信息。根据构造，我们有：

定理4。给定满足图 *1* 要求的域 $\mathcal{R}$ 和 $\mathcal{V}^\sharp$，约束系统的任意解相对于局部轨迹语义都是可靠的。使用 $\mathcal{Q}_a=2^{\mathcal{G}[a]}$ 可获得最大精度。□

对于例 1，使用 $\mathcal{Q}_a=2^{\mathcal{G}[a]}$ 时，两个断言都得到验证。当集合 $\mathcal{G}[a]$ 较大时，同时使用所有可能的子簇进行分析可能会相当昂贵。因此，选择最合适的子聚类通常需要在精度和运行时间之间进行权衡。如果关系域是 k 可分解的，情况则有所不同：

定理5。如果关系域是 k 可分解的（式 (2)），则仅使用大小至多为 k 的所有子簇的簇化分析，与在互斥锁 a 处使用所有大小为 $\mathcal{Q}_a=2^{\mathcal{G}[a]}$ 的子簇的簇化分析具有同等精度。

证明。考虑满足 $\mathcal{Q}_a=2^{\mathcal{G}[a]}$ 的约束系统解 η。那么，对于未知量 $[a,Q,(i,C)]$ 和 $[a,Q',(i,C)]$，其中 $Q\subseteq Q'$ 和 $|Q|\leq k$，以及取值 $r{=}\eta[a,Q,(i,C)]$、$r'{=}\eta[a,Q',(i,C)]$，我们有 $r\sqsubseteq r'|_Q$（每当较小的簇接收到副作用时，较大的簇也会接收到副作用）。因此，根据 k 可分解性，额外的较大簇 Q' 不会改善单个线程 id 上大小至多为 k 的簇之间的 交，也不会改善所有线程 id 上其 汇合的 交。存储在 L 中的簇化信息同样适用。□

例*11.* 再次考虑例 1。如果使用簇$\mathcal{Q}_a=\{\{h,i\},\{g,h\},\{g,i\},\{g\},\{i\},\{h\}\}$ 进行分析，则可以证明两个断言。□

另一方面，单元素簇不能被舍弃——附加材料中附录H的示例对此有所说明。

9 实验评估

我们在扩展上下文敏感静态分析器 Goblint 的基础上实现了这些分析。它支持动态创建互斥锁，并为每个全局变量提供保护互斥锁的集合。该实现使用 Apron [29] 中的 区间或 八边形域来跟踪有关整数变量的信息。由于细节请参见附录 I，因此很难与其他工具进行比较：

– Duet [19] — 论文中的基准测试套件仅以二进制 goto 程序的形式提供，当前版本的 Duet 和此处考虑的其他工具均无法读取。由于 Duet 不支持函数调用，且内联支持有限，因此只能在部分基准测试上运行。– AstréeA [39] — 有公开版本，但未获准用于评估。– Watts [31] — 由于我们无法在任何程序上运行该工具，因此将其与作者报告的数据进行了比较。– NR-Goblint [48] — 使用 [48] 中的非关系分析的 Goblint。

我们针对这些分析考虑了四种不同的设置，即：区间： 第4节中仅使用区间的分析；八边形： 第4节中使用八边形域的分析；线程标识符： 第 7 节中结合附录 F 的增强方法并使用八边形域的分析；簇：线程标识符 仅使用大小不超过 2 的簇。所有基准测试均在一台运行 Ubuntu 20.04 的虚拟机上进行，所用处理器为 AMD EPYC 7742 64 核处理器[5] 。我们的评估结果汇总于表 2。

我们的基准测试。 为了捕捉多线程关系分析中的特定挑战，我们收集了一组小型基准测试（包括本文中的示例），并添加了断言。在这些基准测试上，我们评估了我们的分析器、NR-Goblint 和 Duet。我们在簇 配置下进行的分析能够验证所有程序。其他工具只能证明少数关系断言。

Goblint 基准测试 *[48]*。 这些基准测试不包含断言。为了仍能将我们的分析器的精度与非关系分析器 NR-Goblint 和 Duet 进行比较，我们使用工具的簇 设置，在每个加锁操作处自动推导不变式。或许令人惊讶的是，尽管 NR-Goblint 是非关系分析器且不使用线程标识 符，但它仍能验证 95% 的不变式。

Watts 基准测试 *[31]*。 这些来自不同来源的基准测试程序都添加了断言，并由作者进行了大幅修改。我们的分析能够验证 1000 多个断言中的全部断言，仅有 7 个例外。由于必须修复基准测试程序、我们无法运行他们的工具，以及论文中的断言数量不一致，这些数字无法直接比较。不过，在可扩展性测试中，报告的 Watts 运行时间比我们的结果最多慢两个数量级。更详细的讨论请参见附录 I。

[5] 该分析器是单线程的，因此每个分析任务只使用一个（虚拟）核心。

表 2：评估结果汇总，各个程序按组归类。每组列出了程序数量和断言总数。✓ 表示所有断言均已证明， 表示没有断言得到证明；否则列出已证明的断言数量。（—）表示产生了无效结果。

				我们的分析器					
Set	组	断言数量		区间（第 4 节）	八边形（第 4 节）	TIDs（第 7 节）	簇（第 8 节）	NR-Goblint 带区间	DUET
Our	基础	3	4	✓	✓	✓	✓	✓	3
	关系 10		35	✗	✓	✓	✓	4	2
	TID	12	19	✗	✗	✓	✓	✗	2
	簇	2	3	✗	✗	1	✓	✗	1
Goblint POSIX		5	1679	1146	1490	✓	✓	1582	—
	SV-COMP 7 360			✓	✓	✓	✓	✓	—
Watts	创建	3	3	2	2	2	2	2	✗
	SV-COMP 5		5	1	1	1	1	1	✗
	LKMPG	1	2	✗	✗	✗	✗	✗	✗
	DDVerify 28 1071			1043	1043	✓	✓	1043	—
	可扩展性	5	740	735	735	✓	✓	735	—
Ratcop		19	34	4	14	18	18	6	4

Ratcop 基准测试 *[42]*。这些是 Java 程序。手动翻译成 C 后，我们的分析器使用 八边形域，成功证明了 Ratcop 在任何配置下能够处理的所有断言；而 Ratcop 有一个案例需要使用多面体。

内部比较 我们在 Goblint 基准测试集 [48] 上对分析进行了更详细的评估。图 13a 展示了基准测试程序的规模（以逻辑代码行数计）以及第 6 节的分析所发现的线程 id 数量。被识别为唯一 的线程数量很高，这令人鼓舞。为了详细评估精度，我们比较了每个程序点处的抽象值（在所有上下文上汇合）。图 13a 展示了跟踪线程 id 后精度有所提高的程序点比例。没有程序点出现精度下降或抽象值变得不可比较的情况，而部分程序的精度提升超过了 50%。

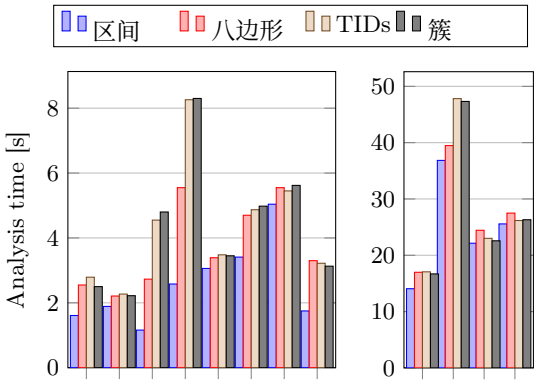
图 13b 展示了这些分析的运行时间。在 12 种情况中的 9 种情况下，我们的关系分析之间的性能差异可以忽略不计。在所有情况下，使用簇都不会产生额外开销。因此，对于线程模块化关系抽象解释，使用大小至多为 2 的簇的精度更高的分析似乎是首选方法。

10 相关工作

自 Miné 引入以来，弱关系数值域 八边形域已广泛应用于程序的分析与验证

Name	LLoC	#线程标识符 (唯一)	线程标识符 ⊏ 八边形
pfscan	550	3 (2)	19.0%
aget	581	6 (4)	0.0%
ctrace	651	3 (3)	0.0%
knot	973	9 (5)	0.0%
smtprc	3013	2 (2)	0.8%
iowarrior	1358	4 (4)	17.1%
adutux	1509	4 (4)	0.0%
w83977af	1515	6 (4)	12.1%
Tegra 20	1560	7 (5)	0.0%
nsc	2394	11 (7)	32.2%
Marvell 1	2476	6 (5)	59.5%
marvell2	2476	6 (5)	58.4%

(a) 已发现的线程数量 id 个以及 程序的比例 分析的程序点比例 id更 p精确。



(b) 分析时间。

图 13：Goblint 基准测试集上的精度与性能评估。

。由于跟踪所有变量之间的关系可能代价高昂，因此有人提出通过预分析来识别其关系可能值得关注的数值变量簇 [8, 14, 26, 45]。Singh 等人提出了一种基于学习的动态方法，将关系域分解为不重叠的簇 [53]。虽然这些方法以牺牲（不必要的）精度为代价换取效率，但其他方法则尝试在不损失精度的情况下将变量划分为簇 [15, 23, 24, 44, 52, 54]。这类聚类与我们的方法正交，或许可以与之结合。

Miné [39] 首创了将关系域整合到线程模块化抽象解释中的方法。他的分析基于锁不变式，这些不变式为每个互斥锁确定一个关系，并且该关系旨在于互斥锁未被持有时成立。随后使用弱干扰来处理对变量的异步访问。对于实际分析，提出仅对锁不变式采用关系抽象，而对弱干扰采用粗粒度非关系抽象。该框架紧随非关系分析 [38]，的相应框架，同时舍弃了背景锁集。另一方面，我们的关系分析在每个互斥锁处仅维护变量之间的关系，其中这些变量受 a 写保护。对于这些关系，可以获得更精确的结果，因为通过交将它们纳入锁处的局部状态（而 [39] 使用汇合）。

Monat 和 Miné [40] 提出了一种分析框架，我们认为它与我们的方法正交。该框架专为验证不依赖通过互斥锁进行显式同步的算法而设计，例如 *Bakery* 算法。Suzanne 和 Miné [55] 扩展了先前的工作，将内存缓冲区纳入线程局部语义，以处理弱内存效应（PSO、TSO）。

干扰的概念也被 Sharma and Sharma [50]用于分析 C11 释放/获取内存模型下的程序，他们还跟踪全局变量的修改序列抽象。他们只考虑固定的有限线程集合，不处理线程创建或线程汇合。

早期关于线程模块化关系分析的研究依靠 Datalog 规则，按照 Miné 的方式建模干扰，并结合应用于数据流图 [19] 或控制流图 [31]的抽象解释（后来扩展到弱内存 [32]）。Botbol 等人 [10] 通过将程序语义编码为符号转导器，对采用消息传递并发的多线程程序进行了非线程模块化分析。

所有这些方法的共同点是，变量簇（如果存在）都是预先定义的，分析不会对其进行特殊处理。Mukherjee 等人 [42]提出的线程模块化关系分析则有所不同。它会将信息从解锁操作传播到加锁操作。该分析针对每个线程的局部变量，以及称为区域的全局变量互不相交子集进行关系分析。这些区域必须预先确定，并满足区域无竞争这一特定要求。相比之下，我们的分析所需的唯一额外先验信息，是全局变量的（写）保护互斥锁集合——这些集合可以在分析过程中计算得出。在我们的方法中，与区域 最接近的概念，是由互斥锁共同保护的全局变量集合。这些集合可能重叠，而分析会明确利用这一点。和我们的方法一样，他们的正确性证明依据线程局部语义。与我们不同的是，该证明基于交错执行，因此过于精细。我们的分析所依据的具体语义，是一种收集式局部轨迹语义，它扩展了 Schwarz 等人 [48]的语义，额外考虑了线程终止和汇合。然而， [48], 中的分析是非关系的。该研究没有提供通过进一步有限抽象（例如线程编号）对局部轨迹进行细化的方法。

与我们最为密切相关的线程 id 分析由 Feret 提出 [20]，他将 π-演算中主体遇到的创建边序列抽象为 ids。并发程序分析的另一条研究路线则致力于确定哪些关键事件可能并行发生（MHP） [1–4, 7, 17, 43, 57] ，以检测潜在的编程错误，例如数据竞争，或找出优化机会。MHP 分析大多是通过对并发程序的全局变量 轨迹语义进行抽象而得到的 [18]。在这里，我们采用相关技术来改进线程模块化分析，但以局部 轨迹语义为基础。与 MHP 分析类似，我们将线程创建与汇合历史以及当前持有的互斥锁集合纳入考量。此外，我们还考虑全局变量修改历史的关键方面，并提供一个通用框架，以实现进一步的细化。

在顺序设置中，根据可达轨迹的某种抽象来拆分控制位置，是提高数据流分析 [9, 27] 或抽象解释精度的常用技术 [25, 34, 41, 47]。控制点拆分可视为缩减基数幂域的一个实例 [12, 13, 22]。对于多线程程序的分析，Miné [39] 采用

Mauborgne和 Rival [34] 的方法来分析单个线程，即独立于所有其他线程的动作。另一方面，我们的方法可以考虑局部轨迹的任意性质，因此更具一般性。

11 结论与未来工作

我们展示了针对可分解域定制的全局变量线程模块化关系分析。在某些情况下，通过考虑更小的簇可以获得更精确的结果。然而，对于 k-可分解域，我们证明了仅考虑大小至多为 k 的簇，就已经可以获得 最优结果。我们提供了一个框架，将局部轨迹的有限抽象纳入分析。在此基础上，我们将线程的创建和汇合都纳入考虑，但相信该框架为无缝提升线程模块化抽象解释的精度铺平了道路。根据文献中提出的基准测试对我们的分析进行评估表明，我们的实现无论在精度还是效率方面都具有竞争力。未来，我们希望尝试对局部轨迹进行进一步抽象，也许针对特定的编程习惯用法进行定制；同时还将探索非数值型二可分解域的潜力。

参考文献

[1] Agarwal, S.、Barik, R.、Sarkar, V.、Shyamasundar, R.K.：X10 程序的可能并行发生分析。载于：PPoPP '07，第 183–193 页，ACM（2007），doi：10.1145/1229428.1229471[2]Albert, E.、Flores-Montoya, A.、Genaim, S.：并发对象中的可能并行发生分析。载于：Giese, H.、Rosu, G.（编）：分布式系统形式化技术——第 14 届 IFIP WG 6.1 国际会议 FMOODS 2012 与第 32 届 IFIP WG 6.1 国际会议 FORTE 2012 联合会议，瑞典斯德哥尔摩，2012 年 6 月 13–16 日。论文集，《计算机科学讲义》，第 7273 卷，第 35–51 页，施普林格（2012），doi：10.1007/978-3-642-30793-5_3，URL https://doi.org/10.1007/978-3-642-30793-5_3[3]Albert, E.、Genaim, S.、Gordillo, P.：带过程间同步的异步程序的可能并行发生分析。载于：Blazy, S.、Jensen, T.P.（编）：静态分析——第 22 届国际研讨会，SAS 2015，法国圣马洛，2015 年 9 月 9–11 日。论文集，《计算机科学讲义》，第 9291 卷，第 72–89 页，施普林格（2015），doi：10.1007/978-3-662-48288-9_5，URL https://doi.org/10.1007/978-3-662-48288-9_5[4]Albert, E.、Genaim, S.、Gordillo, P.：带返回的未来对象的可能并行发生分析。载于：D'Souza, D.、Kumar, K.N.（编）：验证与分析自动化技术——第 15 届国际研讨会，ATVA 2017，印度浦那，2017 年 10 月 3–6 日。论文集，《计算机科学讲义》，第 10482 卷，第 42–58 页，施普林格（2017），doi：10.1007/978-3-319-68167-2_3，URL https://doi.org/10.1007/978-3-319-68167-2_3[5] Apinis, K.、Seidl, H.、Vojdani, V.：带副作用的约束系统：程序分析的瑞士军刀。载于：APLAS '12，第 157–172 页，施普林格（2012），doi：10.1007/978-3-642-35182-2_12[6]Arceri, V.、Olliaro, M.、Cortesi, A.、Ferrara, P.：关系字符串抽象域。载于：Finkbeiner, B.、Wies, T.（编）：验证、模型检验与抽象解释——第 23 届国际会议，VMCAI 2022，美国费城，2022 年 1 月 16 日至 18 日。论文集，《计算机科学讲义》，第 13182 卷，第 20–42 页，施普林格（2022），doi：10.1007/978-3-030-94583-1_2，URL https://doi.org/10.1007/978-3-030-94583-1_2[7] Barik, R.：并发 Java 程序中可能并行信息的高效计算。载于：LCPC '06，第 4339 卷 LNCS，第 152–169 页，施普林格（2006），doi：10.1007/978-3-540-69330-7_11[8]Blanchet, B.、Cousot, P.、Cousot, R.、Feret, J.、Mauborgne, L.、Miné, A.、Monniaux, D.、Rival, X.：大型安全关键软件的静态分析器。载于：ACM SIGPLAN 2003 编程语言设计与实现会议论文集，第

196–207，PLDI '03，计算机协会，纽约州纽约市，美国（2003），ISBN 1581136625，doi: 10.1145/781131.781153，URL https://doi.org/10.1145/781131.781153[9]Bodík, R., Gupta, R., Soffa, M.L.：使用不可行路径细化数据流信息。SIGSOFT 软件工程简报 **22**(6), 361–377（1997年11月），ISSN 0163-5948，doi: 10.1145/267896.267921，URL https://doi.org/10.1145/267896.267921[10]Botbol, V., Chailloux, E., Gall, T.L.：使用符号转导器对通信进程进行静态分析。载于：Bouajjani, A., Monniaux, D.（编）《验证、模型检验与抽象解释——第18届国际会议 VMCAI 2017》，法国巴黎，2017年1月15–17日，论文集，《计算机科学讲义》第10145卷，第73–90页，施普林格（2017），doi: 10.1007/978-3-319-52234-0_5，URL https://doi.org/10.1007/978-3-319-52234-0_5[11]Chen, L., Liu, J., Miné, A., Kapur, D., Wang, J.：一种用于推断带绝对值八边形约束的抽象域。载于：Müller-Olm, M., Seidl, H.（编）《静态分析——第21届国际研讨会 SAS 2014》，德国慕尼黑，2014年9月11–13日，论文集，《计算机科学讲义》第8723卷，第101–117页，施普林格（2014），doi: 10.1007/978-3-319-10936-7_7，URL https://doi.org/10.1007/978-3-319-10936-7_7[12] Cortesi, A., Costantini, G., Ferrara, P.：抽象解释中乘积算子的综述。载于：Banerjee, A., Danvy, O., Doh, K., Hat- cliff, J.（编）《语义、抽象解释与程序推理：献给 David A. Schmidt 六十寿辰的论文集》，美国堪萨斯州曼哈顿，2013年9月19–20日，EPTCS，第129卷，第325–336页（2013），doi: 10.4204/EPTCS.129.19，URL https://doi.org/10.4204/EPTCS.129.19[13]Cousot, P., Cousot, R.：程序分析框架的系统化设计。载于：Aho, A.V., Zilles, S.N., Rosen, B.K.（编）《第六届 ACM 编程语言原理年度研讨会会议记录》，美国得克萨斯州圣安东尼奥，1979年1月，第269–282页，ACM出版社（1979），doi: 10.1145/567752.567778，URL https://doi.org/10.1145/567752.567778[14] Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Rival, X.：Astrée 为何能够扩展？Form. Methods Syst. Des. **35**(3), 229–264（2009年12月），ISSN 0925-9856，doi: 10.1007/s10703-009-0089-6，URL https://doi.org/10.1007/s10703-009-0089-6[15] Cousot, P., Giacobazzi,R., Ranzato, F.：a^2i：抽象^2 解释。ACM 程序设计语言会议录 **3**(POPL)（2019年1月），doi: 10.1145/3290355，URL https://doi.org/10.1145/3290355[16] Cousot, P., Halbwachs, N.：自动发现程序变量之间的线性约束。载于：Aho, A.V., Zilles, S.N., Szymanski, T.G.（编）《第五届 ACM 编程语言原理年度研讨会会议记录》，美国亚利桑那州图森，1月

1978年，第84–96页，ACM出版社（1978），doi: 10.1145/512760.512770，URL https://doi.org/10.1145/512760.512770[17] Di, P., Sui, Y., Ye, D., Xue, J.：面向 C程序的基于区域的可能并行发生分析。载于：ICPP，第889–898页，IEEE（2015），ISBN 978-1-4673-7587-0，doi: 10.1109/ICPP.2015.98[18] Dwyer, M.B., Clarke, L.A.：用于验证并发程序属性的数据流分析。ACM SIGSOFT《软件工程札记》**19**(5), 62–75（1994年12月），doi: 10.1145/195274.195295[19] Farzan, A., Kincaid, Z.：通过对数据和控制进行模块化推理，验证参数化并发程序。载于：第39届 ACM SIGPLAN-SIGACT编程语言原理研讨会论文集，第297–308页，POPL '12，计算机协会，纽约州纽约市，美国（2012），ISBN 9781450310833，doi: 10.1145/2103656.2103693，URL https://doi.org/10.1145/2103656.2103693[20] Feret, J.：移动系统的抽象解释。J. Log. Algebraic Meth- ods Program. **63**(1), 59–130（2005），doi: 10.1016/j.jlap.2004.01.005，URL https://doi.org/10.1016/j.jlap.2004.01.005[21] Fulara, J., Durnoga, K., Jakubczyk, K., Schubert, A.：加权六边形的关系抽象域。Electron. Notes Theor. Comput. Sci. **267**(1), 59–72（2010），doi: 10.1016/j.entcs.2010.09.006，URL https://doi.org/10.1016/j.entcs.2010.09.006[22]Giacobazzi, R., Ranzato, F.：抽象域上的约化相对幂运算。Theor. Comput. Sci. **216**(1-2), 159–211（1999），doi: 10.1016/S0304-3975(98)00194-7，URL https://doi.org/10.1016/S0304-3975(98)00194-7[23]Halbwachs, N., Merchat, D., Gonnord, L.：降低多面体计算空间维数的一些方法。Formal Methods in System Design **29**(1), 79–95（2006年7月），ISSN 1572-8102，doi: 10.1007/s10703-006-0013-2，URL https://doi.org/10.1007/s10703-006-0013-2[24] Halbwachs, N., Merchat, D., Parent-Vigouroux, C.：线性关系分析中的多面体笛卡尔分解。载于：Cousot, R.（编）《静态分析》，第355–365页，Springer Berlin Heidelberg，德国柏林，德国海德堡（2003），ISBN 978-3-540-44898-3[25] Handjieva, M., Tzolovski, S.：使用基于轨迹的控制流划分细化静态分析。载于：Levi, G.（编）《静态分析》，第200–214页，Springer Berlin Heidelberg，德国柏林，德国海德堡（1998），ISBN 978-3-540-49727-1[26] Heo, K., Oh, H., Yang, H.：利用静态分析生成的带标签数据，学习八边形的变量聚类策略。载于：Rival, X.（编）《静态分析》，第237–256页，Springer Berlin Heidelberg，德国柏林，德国海德堡（2016），ISBN 978-3-662-53413-7[27]Holley, L.H., Rosen, B.K.：限定数据流问题。载于：第7届 ACM SIGPLAN-SIGACT编程语言原理研讨会论文集，第68–82页，POPL '80，

计算机协会，纽约，纽约州，美国（1980），ISBN 0897910117，doi: 10.1145/567446.567454，URL https://doi-org.eaccess.ub.tum.de/10.1145/567446.567454[28] Howe, J. M., King, A.: Logahedra：一种新的弱关系域。载于：Liu, Z., Ravn, A.P.（编）《自动化验证与分析技术》，第七届国际研讨会，ATVA 2009，中国澳门，2009年10月14日至16日。论文集，《计算机科学讲义》，第5799卷，第306–320页，施普林格（2009），doi: 10.1007/978-3-642-04761-9_23，URL https://doi.org/10.1007/978-3-642-04761-9_23[29]Jeannet, B., Miné, A.: Apron：一个用于静态分析的数值抽象域库。载于：Bouajjani, A., Maler, O.（编）《计算机辅助验证》，第21届国际会议，CAV 2009，法国格勒诺布尔，2009年6月26日至7月2日。论文集，LNCS，第5643卷，第661–667页，施普林格（2009），doi: 10.1007/978-3-642-02658-4_52，URL https://doi.org/10.1007/978-3-642-02658-4_52[30]Karr, M.: 程序变量之间的仿射关系。《计算机学报》**6**，133–151（1976），doi: 10.1007/BF00268497，URL https://doi.org/10.1007/BF00268497[31] Kusano, M., Wang, C.: 线程模块化抽象解释的流敏感组合。载于：2016年第24届ACM SIGSOFT软件工程基础国际研讨会论文集，第799–809页，FSE 2016，计算机协会，纽约，纽约州，美国（2016），ISBN 9781450342186，doi: 10.1145/2950290.2950291，URL https://doi.org/10.1145/2950290.2950291[32]Kusano, M., Wang, C.: 宽松内存模型的线程模块化静态分析。载于：2017年第11届软件工程基础联合会议论文集，第337–348页，ESEC/FSE 2017，计算机协会，纽约，纽约州，美国（2017），ISBN 9781450351058，doi: 10.1145/3106237.3106243，URL https://doi.org/10.1145/3106237.3106243[33]Logozzo, F., Fähndrich, M.: Pentagons：一种用于高效验证数组访问的弱关系抽象域。载于：2008年ACM应用计算研讨会论文集，第184–188页，SAC '08，计算机协会，纽约，纽约州，美国（2008），ISBN 9781595937537，doi: 10.1145/1363686.1363736，URL https://doi.org/10.1145/1363686.1363736[34]Mauborgne, L., Rival, X.: 基于抽象解释静态分析器的轨迹划分。载于：Sagiv, M.（编）《编程语言与系统》，第5–20页，Springer Berlin Heidelberg，柏林，海德堡（2005），ISBN 978-3-540-31987-0[35]Miné, A.: 一种基于差分界矩阵的新数值抽象域。载于：Danvy, O., Filinski, A.（编）《程序作为数据对象》，第二届研讨会，PADO 2001，奥胡斯，丹麦，2001年5月21日至23日。论文集，LNCS，第2053卷，第155–172页，施普林格（2001），doi: 10.1007/3-540-44978-7_10，URL https://doi.org/10.1007/3-540-44978-7_10

[36] Miné, A.: 八边形抽象域。载于：WCRE' 01，第310页，IEEE计算机学会（2001），doi: 10.1109/WCRE.2001.957836[37]Miné, A.: 八边形抽象域。《高阶符号计算》**19**(1)，31–100（2006年3月），ISSN 1388-3690，doi: 10.1007/s10990-006-8609-1，URL https://doi.org/10.1007/s10990-006-8609-1[38] Miné, A.: 嵌入式实时并行C程序的运行时错误静态分析。《计算机科学逻辑方法》**8**(1)，1–63（2012年3月），doi: 10.2168/LMCS-8(1:26)2012[39] Miné, A.: 通过抽象解释进行关系线程模块化静态值分析。载于：VMCAI '14，第8318卷LNCS，第39–58页，施普林格（2014），doi: 10.1007/978-3-642-54013-4_3[40]Monat, R., Miné, A.: 使用关系干扰抽象对并发程序进行精确线程模块化抽象解释。载于：VMCAI '17，第10145卷LNCS，第386–404页，施普林格（2017），doi: 10.1007/978-3-319-52234-0_21[41]Montagu, B., Jensen, T.: 基于轨迹的控制流分析。载于：PLDI '21，第482–496页，ACM（2021），doi: 10.1145/3453483.3454057，URL https://doi.org/10.1145/3453483.3454057[42] Mukherjee, S., Padon, O., Shoham, S., D'Souza, D., Rinetzky, N.: 线程局部语义及其对无数据竞争程序的高效顺序抽象。载于：SAS '17，第10422卷LNCS，第253–276页，施普林格（2017），doi: 10.1007/978-3-319-66706-5_13[43]Naumovich, G., Avrunin, G.S., Clarke, L.A.: 一种用于计算并发Java程序MHP信息的高效算法。载于：ESEC/FSE '99，第1687卷LNCS，第338–354页，施普林格（1999），doi: 10.1007/3-540-48166-4_21[44]Oh, H., Heo, K., Lee, W., Lee, W., Park, D., Kang, J., Yi, K.: 全局稀疏分析框架。《ACM程序设计语言与系统汇刊》**36**(3)（2014年9月），ISSN 0164-0925，doi: 10.1145/2590811，URL https://doi.org/10.1145/2590811[45]Oh, H., Lee, W., Heo, K., Yang, H., Yi, K.: 由影响预分析引导的选择性x敏感分析。《ACM程序设计语言与系统汇刊》**38**(2)（2015年12月），ISSN 0164-0925，doi: 10.1145/2821504，URL https://doi.org/10.1145/2821504[46]Péron, M., Halbwachs, N.: 一种将不等约束融入差分界矩阵的抽象域。载于：Cook, B., Podelski, A.（编）《验证、模型检验与抽象解释》，第八届国际会议，VMCAI 2007，法国尼斯，2007年1月14日至16日。论文集，《计算机科学讲义》，第4349卷，第268–282页，施普林格（2007），doi: 10.1007/978-3-540-69738-1_20，URL https://doi.org/10.1007/978-3-540-69738-1_20[47]Rival, X., Mauborgne, L.: 轨迹划分抽象域。《ACM程序设计语言与系统汇刊》**29**(5)，26–es（2007年8月），ISSN 0164-0925，doi: 10.1145/1275497.1275501，URL https://doi.org/10.1145/1275497.1275501

[48] Schwarz, M., Saan, S., Seidl, H., Apinis, K., Erhard, J., Vojdani, V.: 改进线程模块化抽象解释。载于：SAS '21，第 12913 卷（LNCS），第 359–383 页，施普林格（2021），doi: 10.1007/978-3-030-88806-0_18[49]Seidl, H., Vogler, R.: 对自顶向下求解器的三项改进。数学结构与计算机科学 **31**(9), 1090–1134 (2021), doi: 10.1017/S0960129521000499, URL https://doi.org/10.1017/S0960129521000499[50]Sharma, D., Sharma, S.: 释放-获取并发的线程模块化分析。载于：Dragoi, C., Mukherjee, S., Namjoshi, K.S.（编），静态分析——第 28 届国际研讨会，SAS 2021，芝加哥，伊利诺伊州，美国，2021 年 10 月 17–19 日，论文集，LNCS，第 12913 卷，第 384–404 页，施普林格（2021），doi: 10.1007/978-3-030-88806-0_19, URL https://doi.org/10.1007/978-3-030-88806-0_19[51] Simon, A., King, A., Howe, J.M.: 以每个线性不等式两个变量构成的抽象域。载于：Leuschel, M.（编），基于逻辑的程序综合与变换，第 12 届国际研讨会 LOPSTR 2002，马德里，西班牙，2002 年 9 月 17–20 日，修订版精选论文，LNCS，第 2664 卷，第 71–89 页，施普林格（2002），doi: 10.1007/3-540-45013-0_7, URL https://doi.org/10.1007/3-540-45013-0_7[52]Singh, G., Püschel, M., Vechev, M.: 快速多面体抽象域。载于：第 44 届 ACM SIGPLAN 编程语言原理研讨会论文集，第 46–59 页，POPL 2017，计算机协会，纽约，美国（2017），ISBN 9781450346603, doi: 10.1145/3009837.3009885, URL https://doi.org/10.1145/3009837.3009885[53] Singh, G., Püschel, M., Vechev, M.: 利用强化学习进行快速数值程序分析。载于：Chockler, H., Weissenbacher, G.（编），计算机辅助验证，第 211–229 页，施普林格国际出版，Cham（2018），ISBN 978-3-319-96145-3[54] Singh, G., Püschel, M., Vechev, M.: 分解数值抽象域的实用构造。ACM 程序设计语言会议录 **2**(POPL)（2018 年 12 月），doi: 10.1145/3158143, URL https://doi.org/10.1145/3158143[55] Suzanne, T., Miné, A.: 宽松内存模型下的关系线程模块化抽象解释。载于：APLAS '18，LNCS 第 11275 卷，第 109–128 页，施普林格（2018 年 12 月），doi: 10.1007/978-3-030-02768-1_6[56] Vojdani, V., Apinis, K., Rõtov, V., Seidl, H., Vene, V., Vogler, R.: 设备驱动程序静态竞争检测：Goblint 方法。载于：ASE '16，第 391–402 页，ACM（2016），doi: 10.1145/2970276.2970337[57] Zhou, Q., Li, L., Wang, L., Xue, J., Feng, X.: 利用静态向量时钟进行可能并行发生分析。载于：CGO '18，第 228–240 页，ACM（2018），doi: 10.1145/3168813

A 局部轨迹语义详述

我们首先详细介绍局部轨迹语义中局部变量和守卫的处理方式；这些内容在正文中略去。然后，我们给出一种局部轨迹的形式化示例，它扩展了 [48] 中提出的方法，以处理线程汇合和返回。

A.1 局部变量与守卫

在我们的设定中，所有作为守卫或赋值的非变量右端项出现的表达式 r 只能包含局部 变量，不能包含全局变量。于是，可以为守卫以及对局部变量的赋值定义相应的右端项函数:

$$[\![u, A]\!]\,\eta = (\emptyset, [\![e]\!](\eta\,[u]))$$

也就是说，计算完全在局部进行，不会触发任何副作用。

A.2 局部轨迹的形式化

具体并发语义对二元动作的定义施加了限制 。具体而言，只有当局部轨迹 t_0 和 t_1 的某些部分表示相同的计算时，才可以为这对轨迹定义二元操作 $[\![e]\!]$ 。为明确这些限制，我们回顾 [48] 中介绍的局部轨迹具体表示，并对其进行扩展，以涵盖线程返回和线程汇合。

单个线程 $i \in \mathcal{I}$的原始 （有限）轨迹是一系列状态 $\bar{u}_j = (u_j, \sigma_j)$ 和动作 act $_j \in \mathcal{A}$ct，即 $\lambda = \bar{u}_0 \mathsf{act}_1 \ldots \bar{u}_{n-1} \mathsf{act}_n \bar{u}_n$，其中包含 $\sigma_j$self $= i$；这些动作对应于线程 i 的局部状态转换，该线程从配置 $\bar{u}_0$ 开始并执行动作 act$_j$。在这个序列中，假定每个动作 lock(m) 都成功；访问全局变量 g时，可以读入任意值。动作 x= join(x') 的返回值也同样如此。

可以将 λ视为一个有向无环图，其节点是 3 元组 (j, u_j, σ_j) j =0, . . . , n，边则是 $((j-1,$ $u_{j-1}, \sigma_{j-1})$, act$_j$, $(j, u_j, \sigma_j))$ $j = 1, \ldots, n$。分别用 $V(\lambda)$ 和 $E(\lambda)$ 表示该图的节点集合和边集合。

用 $\Lambda(i)$ 表示线程 i的所有单个原始轨迹组成的集合，而 Λ 则表示这些集合的并集。

原始全局轨迹是一个有向无环图 $\tau = (\mathrm{V}\ \mathcal{E})$，其中 V $= \bigcup\{V(\lambda_i) \mid i \in I\}$，且 $\mathcal{E} = \bigcup\{E(\lambda_i) \mid i \in I\}$，对应一个线程 标识符集合 I 和原始局部轨迹 $\lambda_i \in \Lambda(i)$。在集合 V 上，我们将程序顺序定义为所有满足以下条件的数对 $\bar{u} \rightarrow_p \bar{u}'$ 的集合: 存在一条边 $(\bar{u}$,act $\bar{u}')$ 属于 $\mathcal{E}$。为了形式化局部轨迹的概念，我们将程序顺序扩展为因果顺序，进一步考虑线程创建和汇合的顺序，以及互斥锁获取和释放的顺序。

对于 $a \in$ M，令 $a^+ \subseteq$V 表示满足以下条件的节点集合：存在一条标记为 lock(a) 的入边，即 $\exists x(x,$ lock(a) $\bar{u}) \in \mathcal{E}$；对于 unlock(a)，定义类似。类似地，令 J 和 R 分别表示 V 中具有一条入边且该边标记为

join(x') 和 return x 的节点集合，其中 x、x' 为任意局部变量。另一方面，令 C 表示具有一条出边且该边标记为 x=create(u_1) 的节点集合（其中 x 为任意局部变量，u_1 为任意程序点）。令 S 表示相对于 $\rightarrow_p$ 的极小节点集合，即线程开始的程序点；并令 0 表示节点 $(0, u_0, \sigma_0)$，其中 σ_0 为 self $= 0$。

一个全局轨迹 t then表示为一个元组（τ、$\rightarrow_c$、$\rightarrow_j$、$(\rightarrow_a)_{a\in\mathsf{M}}$、$(\rightarrow_s)_{s\in\mathcal{S}}$），其中 τ 是原始全局轨迹，关系 $\rightarrow_c$、$\rightarrow_j$、$\rightarrow_a$（$a \in \mathrm{M}$）分别是第3节中介绍的创建顺序、汇合顺序和锁定顺序。t then的因果顺序$\leq$ 定义为该并集的自反传递闭包

$$\rightarrow_p \cup \rightarrow_c \cup \rightarrow_j \cup \bigcup_{a\in\mathsf{M}} \rightarrow_a$$

这些顺序需要满足以下性质：

– 因果顺序 $\leq$ 是一个具有唯一最小元（0、u_0、σ_0）的偏序，其中 $\sigma_0$self $= 0$；– 创建顺序：$\rightarrow_c \subseteq C \times (S \setminus \{\mathbf{0}\})$：$\forall s \in (S \setminus \{\mathbf{0}\})$：$|\{z \mid z \rightarrow_c s\}| = 1$，也就是说，除初始线程外，每个线程都恰好由一个create(...)动作创建，并且 $\forall x$：$|\{z \mid x \rightarrow_c z\}| \leq 1$，即每个create(...)动作至多创建一个线程。此外，对于（（$j-1$、u_{j-1}、σ_{j-1}）x=create(v)，（j、u_j、σ_j））$\in \mathcal{E}$以及（$j-1$、u_{j-1}、σ_{j-1}）$\rightarrow_c$（0、v、σ'_0）$\sigma'_0 = \sigma_{j-1} \oplus \{\text{self} \mapsto i'\}$，某个线程标识符 i' 满足 $\sigma_j\, x = i'$，即创建线程和被创建线程在被创建线程的线程标识符上保持一致；– 汇合顺序：$\rightarrow_j \subseteq R \times J$：$\forall j' \in J$：$|\{z \mid z \rightarrow_j j'\}| = 1$ 且 $\forall r' \in R$：$|\{z | r' \rightarrow_j z\}| \leq 1$，也就是说，轨迹中的每个汇合操作恰好汇合一个同样出现在轨迹中的线程，并且每个线程至多被汇合一次。此外，对于（（$j-1$、u_{j-1}、σ_{j-1}）x=join(x')，（j、u_j、σ_j））$\in \mathcal{E}$以及（（$j'-1$、$u_{j'-1}$、$\sigma'_{j'-1}$），return y，（j'、$u_{j'}$、$\sigma'_{j'}$））$\in \mathcal{E}$，并且（j'、$u_{j'}$、$\sigma'_{j'}$）$\rightarrow_j$（j、u_j、σ_j）：$\sigma_{j-1}\, x' = \sigma'_{j'-1}$ self，且 $\sigma_j\, x = \sigma'_{j'-1}\, y$，即被汇合线程的线程标识符存储在 x' 中，而在汇合之后，返回值被赋给 x；– 锁定顺序：$\forall a \in \mathrm{M}$：$\rightarrow_a \subseteq (a^- \cup \mathbf{0}) \times a^+$：$\forall x \in (a^- \cup \mathbf{0})$：$|\{z \mid x \rightarrow_a z\}| \leq 1$ 且 $\forall y \in a^+$：$|\{z \mid z \rightarrow_a y\}| = 1$，也就是说，对于互斥锁 a，每次加锁之前恰好有一次对 a的解锁（或者这是第一次加锁），并且每次解锁之后至多直接跟随一次加锁

此外，我们要求从全局变量读取的值满足一致性条件。对于 $((j-1, u_{j-1}, \sigma_{j-1})\ x = g, (j, u_j, \sigma_j)) \in \mathcal{E}$，相对于 $\leq$，存在唯一的极大节点 $(j', u_{j'}, \sigma_{j'})$，使得 $((j'-1, u_{j'-1}, \sigma_{j'-1})\ g = y, (j', u_{j'}, \sigma_{j'})) \in \mathcal{E}$，且 $(j', u_{j'}, \sigma_{j'}) \leq (j-1, u_{j-1}, \sigma_{j-1})$。由于 $\rightarrow_{m_g}$，此类极大节点的唯一性得以保证：对全局变量 g 的每次写入操作后，都会立即执行一次 unlock(m_g) 操作；对全局变量 g 的每次读取操作前，都会立即执行一次 lock(m_g) 操作。于是 $\sigma_j\, x = \sigma_{j'-1}\, y$，也就是说，对全局变量的读取所得值是最后写入该变量的值。

若全局轨迹 t 具有唯一的极大元素 $\bar{u} = (j, u, \sigma)$（相对于 $\leq$），则称其为局部轨迹。对于局部轨迹 t，我们定义函数 sink t，以提取元组 (u, σ)；并定义函数 loc t，以获取汇点节点的程序点 u。函数

另一方面，last 会提取自身线程的最后一个动作 act（如果存在），否则返回 $\perp$。对于程序点u 和 u_1，以及控制流边 e，偏函数 new $u\,u_1$ 和 $[\![e]\!]$ 定义为对给定局部轨迹进行适当扩展。

B 第4节分析的可靠性证明

将第4节分析的约束系统称为 $\mathcal{C}^\sharp$，将第3节具体语义的约束系统方程（3）称为 $\mathcal{C}$。我们首先修改 $\mathcal{C}$，得到一个新的约束系统 $\mathcal{C}'$，它与 $\mathcal{C}^\sharp$具有相同的未知量，但线程 id除外。这意味着，程序点 u 处的每个未知量 $[u]$ 都替换为未知量集合 $[u,S]$, $S\subseteq \mathsf{M}$，而互斥锁 a 对应的每个未知量 $[a]$ 都替换为未知量集合 $[u,Q]$, $Q\in\mathcal{Q}_a$。我们指出，每个互斥锁 a至少有一个这样的未知量，因为我们要求 $\mathcal{Q}_a\neq\emptyset$。线程 id和互斥锁对应的未知量保持不变。因此，约束系统 $\mathcal{C}'$ 由以下约束组成：

$$\begin{aligned}&(\eta,\eta\,[u_0,\emptyset])\sqsupseteq(\{[a,Q]\mapsto \text{初始化}\mid a\in\mathsf{M}\ Q\in\mathcal{Q}_a\},\ \text{初始化})\\&(\eta,\eta\,[u',S\cup\{a\}])\sqsupseteq[\![u,S]\!],\ \text{加锁}(a)]\!]\,\eta\,(\text{对于 }(u,\ \text{加锁}(a)\ u')\in\mathcal{E},S\subseteq\mathsf{M})\\&(\eta,\eta\,[u',S\backslash\{a\}])\sqsupseteq[\![u,S]\!],\ \text{解锁}(a)]\!]\,\eta\,(\text{对于 }(u,\ \text{解锁}(a)\ u')\in\mathcal{E},S\subseteq\mathsf{M})\\&(\eta,\eta\,[u',S])\sqsupseteq[\![u,S]\!],\ \text{动作}]\!]\,\eta\,(\text{对于 }(u,\ \text{动作 }u')\in\mathcal{E},S\subseteq\mathsf{M},\ \text{以及其他动作 act})\end{aligned}\tag{8}$$

其中，右端项经过修改，改为查询相应的未知量$[u,S]$，而非 $[u]$。此外，锁定和解锁互斥锁的右端项也经过修改，按如下方式查询相应的 $[a,Q]$，并产生副作用 $Q\in\mathcal{Q}_a$：

$$\begin{aligned}[\![[u,S],\mathsf{lock}(a)]\!]\,\eta&=\Big(\emptyset,[\![e]\!]\Big(\eta\,[u],\textstyle\bigcap_{Q\in\mathcal{Q}_a}(\eta\,[a,Q])\Big)\Big)\\[\![u,\mathsf{unlock}(a)]\!]\,\eta&=\mathbf{let}\ T=[\![e]\!](\eta\,[u])\ \mathbf{in}\\&\qquad(\{[a,Q]\mapsto T\mid Q\in\mathcal{Q}_a\},T)\end{aligned}$$

首先，我们将 $\mathcal{C}$ 和 $\mathcal{C}'$ 的唯一最小解联系起来。设 η 为 $\mathcal{C}$的唯一最小解。我们据此构造一个映射 η'，将 $\mathcal{C}'$ 的未知量映射到 $2^{\mathcal{T}}$。

$$\begin{aligned}\eta'\,[u,S]&=\eta\,[u]\cap\mathcal{T}_S&&(\text{for } u\in\mathcal{N},S\subseteq\mathsf{M})\\\eta'\,[i]&=\eta\,[i]&&(\text{for } i\in\mathcal{I})\\\eta'\,[a,Q]&=\eta\,[a]&&(\text{for } a\in\mathsf{M},Q\in\mathcal{Q}_a)\end{aligned}$$

其中，$\mathcal{T}_S$ 表示自身线程在汇点节点持有锁集 S时的所有轨迹的集合。

命题 1。 η' 是 $\mathcal{C}'$ 的最小解，当且仅当 η 是 $\mathcal{C}$的最小解。

证明。 命题 1 的证明采用不动点归纳法。

在继续进行可靠性证明之前，我们需要先了解第4节中的分析所得出的以下结论。

命题 2。 设 $\eta^\sharp$ 是由第*4*节中的分析计算得到的 $\mathcal{C}^\sharp$ 的一个解。那么，

$$\begin{array}{ll}\eta^\sharp\,[u,S] = \eta^\sharp\,[u,S]\big|_{\mathcal{X}\cup\{g\mid g\in\mathcal{G},\mathcal{M}\,[g]\cap S\neq\emptyset\}} & (for\ u\in\mathcal{N}, S\subseteq M)\\ \eta^\sharp\,[i] \quad = \eta^\sharp\,[i]\big|_{\{ret\}} & (for\ i\in\mathcal{V}_\mathcal{I}^\sharp)\\ \eta^\sharp\,[a,Q] = \eta^\sharp\,[a,Q]\big|_{Q} & (for\ a\in M, Q\in\mathcal{Q}_a)\end{array}$$

证明。 命题 2 的证明再次通过不动点归纳，利用操作 $restrict$ 的幂等性完成。

为了关联 $\mathcal{C}'$ 和 $\mathcal{C}^\sharp$ 的解，我们扩展函数 $x\mapsto t(x)$：它从局部轨迹 t 中提取汇点节点处局部变量 x 的值，现在还会提取 t中某个全局变量的最后 一个值。令 Σ 为所有映射 $\sigma:\mathcal{X}\to\mathcal{V}$ 的集合，这些映射为局部变量赋值。回顾一下，在局部轨迹 t 中，对于曾在 t中写入过的每个全局变量 g，都存在唯一一个最后 （按照因果顺序）的写操作 $((j-1,u_{j-1},\sigma_{j-1})\ g=x,(j,u_j,\sigma_j))$。令 $\mathsf{last_write}_g:\mathcal{T}\to((\mathbb{N}_0\times\mathcal{N}\times\Sigma)\times\mathcal{A}\mathrm{ct}\times(\mathbb{N}_0\times\mathcal{N}\times\Sigma))\cup\{\bot\}$ 为一个函数，用于从局部轨迹中提取对 g 的这次最后写入，其中 $\bot$ 表示 g 尚未被写入。令 $\mathsf{last_value}_g:\mathcal{T}\to\mathcal{V}$ 为一个函数，用于提取局部轨迹中某个全局变量的最后一个值。如果局部轨迹中出现对该全局变量的写入，那么该值就是最后 一次写入的值。否则，最后一个值就是该全局变量的初始值，即 0。

$$\mathsf{last_value}_g\,t=\begin{cases}\sigma_{j-1}\,x & \text{if } \mathsf{last_write}_g\,t=((j-1,u_{j-1},\sigma_{j-1}),g=x,_)\\ 0 & \text{if } \mathsf{last_write}_g\,t=\bot\end{cases}$$

接下来，我们为局部轨迹 $t\in\mathcal{T}$and $x\in(\mathcal{G}\cup\mathcal{X})$ 定义

$$t(x)=\begin{cases}\sigma\,x & \text{if } x\in\mathcal{X},\mathsf{sink}\,t=(_,\sigma)\\ \mathsf{last_value}_g\,t & \text{if } x\in\mathcal{G}\end{cases}$$

现在，让我们将 $\mathcal{C}'$ and $\mathcal{C}^\sharp$ 的解联系起来。为此，对于局部轨迹 t，我们定义函数 $\beta:\mathcal{T}\to(\mathcal{V}\mathrm{ars}\to\mathcal{V})$ 如下

$$\beta\,t=\{x\mapsto t(x)\mid x\in(\mathcal{X}\cup\mathcal{G})\}$$

基于此，我们为 $\mathcal{C}^\sharp$ 中的未知量提供专门的具体化函数。对于局部状态 $r\in\mathcal{R}$，我们定义

$\gamma_{u,\ S}(r)=\{t\mid t\in\mathcal{T}_S,\mathrm{loc}\,t=u,\exists\,v:(\beta\,t\cup\{\mathrm{ret}\mapsto v\})\in\gamma_\mathcal{R}\,r\}\gamma_i(r)=\{t\mid t\in\mathcal{T},\mathrm{last}\,t=$(返回 x) $\sigma=\beta\,t,(\sigma\cup\{\mathrm{ret}\mapsto\sigma\,x\})\in\gamma_\mathcal{R}\,r\}\gamma_a(r)=\{t\mid t\in\mathcal{T},($last $t=$解锁$(a)\ \vee\ t\in$ init$)\ \exists\,v:(\beta\,t\cup\{\mathrm{ret}\mapsto v\})\in\gamma_\mathcal{R}\,r\}$

并指出这些具体化函数是单调的。此外，我们要求操作 $\nu^\sharp:\mathcal{N}\to\mathcal{N}\to\mathcal{R}\to\mathcal{V}_\mathcal{I}^\sharp$ 对于具体状态中被创建线程的id是可靠的。

$$\gamma_{\mathcal{V}_\mathcal{I}^\sharp}(\nu^\sharp\,u\,u_1\,r)\supseteq\{t'(\mathsf{self})\mid t'=\mathsf{new}\,u\,u_1\,t,t\in\gamma_{u,S}(r)\}$$

对于 $\mathcal{C}^\sharp$ 的一个解 $\eta^\sharp$，我们随后通过以下方式构造映射 η'：

$$\begin{array}{ll}\eta'[u,S] = \gamma_{u,S}(\eta^\sharp\,[u,S]) & (\text{for } u \in \mathcal{N}, S \subseteq \mathsf{M}) \\ \eta'[i] \quad = \bigcup\{\gamma_i(\eta^\sharp\,[i^\sharp]) \mid i \in \gamma_{\mathcal{V}^\sharp}\, i^\sharp\} & (\text{for } i \in \mathcal{I}) \\ \eta'[a,Q] = \gamma_a(\eta^\sharp\,[a,Q]) & (\text{for } a \in \mathsf{M}, Q \in \mathcal{Q}_a)\end{array}$$

总之，约束系统 $\mathcal{C}^\sharp$ 的可靠性可由以下定理得出。

定理 6. *$\mathcal{C}^\sharp$ 的每个解相对于局部轨迹语义都是可靠的。*

证明。根据命题 1，$\mathcal{C}'$ 的最小解相对于约束系统 $\mathcal{C}$ 所指定的局部轨迹语义是可靠的。因此，只需证明：由上述 $\mathcal{C}^\sharp$ 的一个解 $\eta^\sharp$ 构造的映射 η' 是约束系统 $\mathcal{C}'$ 的一个解。我们通过不动点归纳验证，对于 $\mathcal{C}'$ 最小解的第 j 次近似 η^j，$\eta^j \subseteq \eta'$ 成立。为此，我们验证对于起始点 u_0 和空锁集，

$$(\{[a,Q] \mapsto \mathsf{init} \mid a \in \mathsf{M}, Q \in \mathcal{Q}_a\}, \mathsf{init}) \subseteq (\eta', \eta'\,[u_0, \emptyset])$$

成立；并且对于控制流图中的每条边 $(u, \mathsf{act}\ v)$ 以及每个可能的锁集 S，

$$[\![[u,S], \mathsf{act}]\!]'\,\eta^{j-1} \subseteq (\eta', \eta'\,[v,S'])$$

成立。

我们以对应于程序起始点、互斥锁加锁以及线程返回的约束为例进行说明。其他所有约束的证明过程类似。

首先，对于起始点 u_0 和空锁集：

$$(\{[a,Q] \mapsto \mathsf{init} \mid a \in \mathsf{M}, Q \in \mathcal{Q}_a\}, \mathsf{init}) \subseteq (\eta', \eta'\,[u_0, \emptyset])$$

首先，对于未知量 $[u_0, \emptyset]$ 的值：$\mathsf{init} \subseteq \eta'\,[u_0, \emptyset]$:

$$\begin{array}{l}\mathsf{init}^\sharp\,\eta = \mathbf{let}\ \rho = \left\{[a,Q] \mapsto [\![\{g \leftarrow 0 \mid g \in Q\}]\!]^\sharp_{\mathcal{R}}\,\top \mid a \in \mathsf{M}, Q \in \mathcal{Q}_a\right\}\ \mathbf{in} \\ \qquad (\rho, [\![\mathsf{self} \leftarrow^\sharp i_0]\!]^\sharp_{\mathcal{R}}\,\top)\end{array}$$

令 $\eta^\sharp\,[u_0, \emptyset] = r^\sharp$ 为 $\mathcal{C}^\sharp$ 在起始点和空锁集处提供的值。由于 $\eta^\sharp$ 是 $\mathcal{C}^\sharp$ 的一个解，$\left([\![\ \mathsf{self}\ \leftarrow i_0]\!]^\sharp_{\mathcal{R}}\,\top\right) \sqsubseteq r^\sharp$。根据定义，

$$\eta'[u_0, \emptyset] = \gamma_{u_0,\emptyset}(r^\sharp) = \{t \mid t \in \mathcal{T}_\emptyset, \mathsf{loc}\,t = u_0, \exists v : (\beta\,t \cup \{ret \mapsto v\}) \in \gamma_{\mathcal{R}}\,r^\sharp\}$$

对于每条轨迹 $t \in \mathsf{init}$，令 $r = \beta\,t \cup \{\mathsf{ret} \mapsto 0\}$。那么，

$$r \in \gamma_{\mathcal{R}}\left([\![\mathsf{self} \leftarrow^\sharp i_0]\!]^\sharp_{\mathcal{R}}\,\top\right) \subseteq \gamma_{\mathcal{R}}(r^\sharp)$$

因此，对于所有 $t \in \mathsf{init}$，都有 $t \in \gamma_{u_0,\emptyset}(r^\sharp) = \eta'\,[u_0, \emptyset]$。

接下来，对于副作用，我们需要验证：对于所有 $a \in \mathrm{M}$， $Q \in \mathcal{Q}_a$: init $\subseteq$ $\gamma_a\left(\left[\!\left[\{g \leftarrow 0 \mid g \in Q\}\right]\!\right]^\sharp_{\mathcal{R}} \top\right)$，即抽象中的副作用涵盖具体中的所有副作用。根据定义，

$$\begin{aligned}\gamma_a\left(\left[\!\left[\{g \leftarrow 0 \mid g \in Q\}\right]\!\right]^\sharp_{\mathcal{R}} \top\right) = \{t \mid t \in \mathcal{T},\\ (\mathsf{last}\, t = \mathsf{unlock}(a) \vee t \in \mathsf{init}),\\ \exists v : (\beta\, t \cup \{\mathsf{ret} \mapsto v\}) \in \gamma_{\mathcal{R}}\left(\left[\!\left[\{g \leftarrow 0 \mid g \in Q\}\right]\!\right]^\sharp_{\mathcal{R}} \top\right)\\ \}\end{aligned}$$

考虑 $t \in$ init，并设 $\sigma = \beta\, t \cup \{\mathsf{ret} \mapsto 0\}$。那么 $\forall g \in \mathcal{G}$： $\sigma\, g = 0$，因为所有全局变量的初始值均为 0。于是 $\sigma \in \gamma_{\mathcal{R}}\left(\left[\!\left[\{g \leftarrow 0 \mid g \in Q\}\right]\!\right]^\sharp_{\mathcal{R}} \top\right)$，因此对于所有 $t \in$ init $t \in \gamma_a\left(\left[\!\left[\{g \leftarrow 0 \mid g \in Q\}\right]\!\right]^\sharp_{\mathcal{R}} \top\right)$，即所有副作用都已计入。

接下来，考虑对互斥锁 a 加锁：

$$\begin{aligned}\left[\!\left[[u, S], \mathsf{lock}(a)\right]\!\right]^\sharp \eta^\sharp = \mathbf{let}\, r^\sharp = \eta^\sharp\,[u, S]\ \mathbf{in}\\ \mathbf{let}\, r^{\sharp\prime\prime} = r^\sharp \sqcap \left(\textstyle\bigsqcap_{Q \in \mathcal{Q}_a} \eta^\sharp\,[a, Q]\right)\ \mathbf{in}\\ (\emptyset, r^{\sharp\prime\prime})\end{aligned}$$

设 $r^\sharp = \eta^\sharp[u, S]$ 和 $r^{\sharp\prime} = \eta^\sharp[v, S \cup \{a\}]$ 分别为 $\eta^\sharp$ 为边的起点和终点提供的值。由于 $\eta^\sharp$ 是 $\mathcal{C}^\sharp$ 的解， $r^{\sharp\prime\prime} \sqsubseteq r^{\sharp\prime}$。那么，根据定义：

$$\eta'\,[v, S \cup \{a\}] = \{t \mid t \in \mathcal{T}_{S \cup \{a\}}, \mathsf{loc}\, t = v, \exists\, v' : (\beta\, t \cup \{\mathsf{ret} \mapsto v'\}) \in \gamma_{\mathcal{R}}\, r\}$$

对于任意轨迹 $t_0 \in \eta^{i-1}\,[u, S]$，令 $\beta\, t_0 = \sigma_0$。根据归纳假设， $\exists v'$:$\sigma_0 \cup \{\mathsf{ret} \mapsto v'\} \in \gamma_{\mathcal{R}}(r^\sharp)$。对于任意轨迹 $t_1 \in \bigcap_{Q \in \mathcal{Q}_a}\left(\eta^{i-1}\,[a, Q]\right)$，令 $\beta\, t_1 = \sigma_1$。根据归纳假设，$\exists v''$： $\sigma_1 \cup \{\mathsf{ret} \mapsto v''\} \in \bigcap_{Q \in \mathcal{Q}_a}\left(\gamma_{\mathcal{R}}\left(\eta^\sharp\,[a, Q]\right)\right)$。

对于任意 $t' \in [\![e]\!](\{t_0\}, \{t_1\})$， $\mathsf{loc}(t') = v$ and $t' \in \mathcal{T}_{S \cup \{a\}}$。令 $\sigma' = \beta\, t' = \{x \mapsto t'(x) \mid x \in (\mathcal{X} \cup \mathcal{G})\}$。于是 $\forall x \in \mathcal{X}$： $\sigma'\, x = \sigma_0\, x$。另一方面，对于所有全局变量 $g \in \mathcal{G}\,[a]$，以下情况之一成立：

(a) $S \cap \mathcal{M}[g] \neq \emptyset \wedge t'(g) = t_0(g) = t_1(g)$。

(b) $S \cap \mathcal{M}[g] = \emptyset \wedge t'(g) = t_1(g)$】【。

由于 $\mathcal{C}^\sharp$ 保持了这样的不变式：程序点处的 r 不包含有关未受保护的全局变量的信息；而且，针对互斥锁和簇的 r 不会保留局部变量的信息，也不会保留未受该互斥锁保护或不属于该簇的全局变量的信息（命题 2）；并且 $\mathcal{R}$ 中的 $\sqcap$ operation 操作相对于具体化的交集是可靠的，因此有：

$$\begin{aligned}\sigma' \cup \{\mathsf{ret} \mapsto v'\} \in \gamma_{\mathcal{R}}\left(r^\sharp\right) \cap \textstyle\bigcap_{Q \in \mathcal{Q}_a}\left(\gamma_{\mathcal{R}}\left(\eta^\sharp\,[a, Q]\right)\right)\\ \subseteq \gamma_{\mathcal{R}}\left(r^\sharp \sqcap \left(\textstyle\bigsqcap_{Q \in \mathcal{Q}_a} \eta^\sharp\,[a, Q]\right)\right) = \gamma_{\mathcal{R}}\left(r^{\sharp\prime}\right)\end{aligned}$$

因此，$t' \in \gamma_{v,\ S\cup\{a\}} r^{\sharp'}$，从而 $t' \in \eta'\,[v, S \cup \{a\}]$。我们得出结论：$[\![[u, S]$ 的返回值 lock(a)$]\!]\,\eta^{i-1}$ 被 $\eta'\,[v, S \cup \{a\}]$ 包含。由于该约束不会产生副作用，因此结论成立。

接下来，考虑一条对应于线程返回的边。

$$\begin{aligned}[\![[u, S], \mathsf{return}\ x]\!]^\sharp \eta^\sharp = \ & \mathbf{let}\ r^\sharp = \eta^\sharp\,[u, S]\ \mathbf{in} \\ & \mathbf{let}\ r^{\sharp''} = \left([\![\mathsf{ret} \leftarrow x]\!]^\sharp_{\mathcal{R}}\ r^\sharp\right)\Big|_{\{\mathsf{ret}\}}\ \mathbf{in} \\ & \mathbf{let}\ \rho = \left\{[\mathsf{unlift}\ r^\sharp\ \mathsf{self}] \mapsto r^{\sharp''}\right\}\ \mathbf{in} \\ & (\rho, r^\sharp)\end{aligned}$$

令 $r^\sharp = \eta^\sharp[u, S]$ 和 $r^{\sharp'} = \eta^\sharp[v, S]$ 分别为 $\eta^\sharp$ 为该边的起点和终点提供的值。由于 $\eta^\sharp$ 是 $\mathcal{C}^\sharp$ 的一个解，$r^\sharp \sqsubseteq r^{\sharp'}$。对于任意轨迹 $t \in \eta^{i-1}\,[u, S]$，令 $\beta\, t_0 = \sigma_0$。根据归纳假设，$\exists v'$：$\sigma_0 \cup \{$返回 $\mapsto v'\} \in \gamma_{\mathcal{R}}(r^\sharp)$。令 $t' = [\![e]\!]\,\{t\}$。则 $\beta\, t' = \sigma_1 = \sigma_0$。因此，

$$\sigma_1 \cup \{\mathsf{ret} \mapsto v'\} \in \gamma_{\mathcal{R}}(r^\sharp) \subseteq \gamma_{\mathcal{R}}(r^{\sharp'})$$

并且 $t' \in \gamma_{v,\ S} r^{\sharp'}$，因此 $t' \in \eta'\,[v, S]$。

接下来，考虑单条轨迹 t 所引起的副作用：

$$\{[t(\mathsf{self})] \mapsto [\![e]\!](t)\} \qquad\qquad \left\{[\mathsf{unlift}\ r^\sharp\ \mathsf{self}] \mapsto \left([\![\mathsf{ret} \leftarrow x]\!]^\sharp_{\mathcal{R}}\ r^\sharp\right)\Big|_{\{\mathsf{ret}\}}\right\}$$

和之前一样，考虑 $t' = [\![e]\!](t)$ 和 $\beta\, t' = \sigma_1 = \beta\, t = \sigma_0$。我们有 $\exists v'$：$\sigma_0 \cup \{\mathsf{ret} \mapsto v'\} \in \gamma_{\mathcal{R}}(r^\sharp)$。现在令 $\sigma' = \sigma_0 \cup \{\mathsf{ret} \mapsto \sigma_0\, x\}$。然后

$$\sigma' \in \gamma_{\mathcal{R}}\left([\![\mathsf{ret} \leftarrow x]\!]^\sharp_{\mathcal{R}} r^\sharp\right) \subseteq \gamma_{\mathcal{R}}\left(\left([\![\mathsf{ret} \leftarrow x]\!]^\sharp_{\mathcal{R}}\ r^\sharp\right)\Big|_{\{\mathsf{ret}\}}\right) = r^{\sharp''}$$

令 $i = t(\mathsf{self})$。于是 $t' \in \gamma_i\left(r^{\sharp''}\right)$，也就是说，具体语义中经过副作用更新的值由抽象语义中经过副作用更新的抽象值来抽象。接下来要证明的是，具体语义中的所有副作用都在抽象语义中得到了考虑。根据归纳假设以及对 $\mathcal{R}$ 和 $\mathcal{V}^\sharp$ (图 1) 的要求，我们有 $i \in \gamma_{\mathcal{V}^\sharp}(\mathsf{unlift}\ r^\sharp\ \mathsf{self})$。根据 η' 的构造，并且由于 $\eta^\sharp$ 是 $\mathcal{C}^\sharp$ 的一个解，

$$t' \in \gamma_i\left(r^{\sharp''}\right) \subseteq \gamma_i\left(\eta^\sharp[\mathsf{unlift}\ r^\sharp\ \mathsf{self}]\right) \subseteq \eta'\,[i]$$

因此，副作用已纳入考虑，该结论对于此约束成立。

C 精化分析的完整约束系统

考虑第4节中的抽象约束系统。本节所考虑动作的精化约束系统的完整定义

$(\eta, \eta\,[u_0, \emptyset, A'])$ ⊒ $\mathsf{init}^\sharp_{A'}$, η对于起始点 u_0, $A' \in \mathsf{init}^\sharp_{\mathcal{A}}(\eta, \eta\ [v, S \cup \{a\}, A'])$ ⊒ $[\![[u, S, A_0]$, 锁(a) $A_1]\!]^\sharp\,\eta$对于 (u, 锁(a) v) $\in \mathcal{E}, S \subseteq \mathbf{M}$ $A_0 \in \mathcal{A}, A_1 \in \mathcal{A}, A' \in [\![u$, 锁($a$)$]\!]^\sharp_{\mathcal{A}}\,(A_0, A_1)\,(\eta, \eta\ [v, S \setminus \{a\}, A'])$ ⊒ $[\![[u, S, A_0]$, 解锁(a) $A']\!]^\sharp\,\eta$对于 (u, 解锁(a) v) $\in \mathcal{E}, S \subseteq \mathbf{M}$ $A_0 \in \mathcal{A}, A' \in [\![u$, 解锁 ($a$)$]\!]^\sharp_{\mathcal{A}}\,(A_0)\,(\eta, \eta\ [v, S, A'])$ ⊒ $[\![[u, S, A_0]$, x'=汇合(x) $A_1]\!]^\sharp\,\eta$对于 (u, x'=汇合(x) v) $\in \mathcal{E}, S \subseteq \mathbf{M}$ $A_0 \in \mathcal{A}, A_1 \in \mathcal{A}, A' \in [\![u, x'$=汇合($x$)$]\!]^\sharp_{\mathcal{A}}\,(A_0, A_1)\,(\eta, \eta\ [v, S, A'])$ ⊒ $[\![[u, S, A_0]$, 返回 x, $A']\!]^\sharp\,\eta$对于 (u, 返回 x, v) $\in \mathcal{E}, S \subseteq \mathbf{M}$ $A_0 \in \mathcal{A}, A' \in [\![u$, 返回 $x]\!]^\sharp_{\mathcal{A}}\,(A_0)\,(\eta, \eta\ [v, S, A'])$ ⊒ $[\![[u, S, A_0]$, 动作$]\!]^\sharp\,\eta$对于 (u, 动作 v) $\in \mathcal{E}, S \subseteq \mathbf{M}$, 其他动作 act $A_0 \in \mathcal{A}, A' \in [\![u$, 动作$]\!]^\sharp_{\mathcal{A}}\,(A_0)$,

图 14：带细化的约束系统。

论文中添加了锁集的程序点未知量，如图 14 所示；通过控制点拆分细化后的系统右端项由原始右端项按如下方式确定

$[\![[u, S, A_0]$, **锁定(**a**)** $A_1]\!]^\sharp\,\eta$ = **令** $\eta'\,[x]$ = **若** $[x] = [u, S]$ **则** $\eta\,[u, S, A_0]$**否则** $\eta\,[x, A_1]$ **于 令 (**$\emptyset$**,** v**)** = $[\![[u, S]$, **锁定(**a**)**$]\!]^\sharp\,\eta'$ **于 (**$\emptyset$**,** v**)**$[\![[u, S, A_0]$, **解锁(**a**)** $A']\!]^\sharp\,\eta$ = **令** $\eta'\,[x] = \eta\,[x, A_0]$ **于 令 (**ρ**,** v**)** = $[\![[u, S]$, **解锁(**a**)**$]\!]^\sharp\,\eta'$ **于 令** $\rho' = \{[a, Q, A'] \mapsto v' \mid$ **(** $[a, Q] \mapsto v'$**)** $\in \rho\}$**(**ρ'**,** v**)**$[\![[u, S, A_0]$, x'=**汇合(**x''**)** $A_1]\!]^\sharp\,\eta$ = **令** $\eta'\,[x''] = $ **若** $[x] = [u, S]$ **则** $\eta\,[u, S, A_0]$**否则** $\eta\,[x, A_1]$ **于 令 (**$\emptyset$**,** v**)** = $[\![[u, S]$, x''=**汇合(**x'**)**$]\!]^\sharp\,\eta'$ **于 (**$\emptyset$**,** v**)**$[\![[u, S, A_0]$, **返回** x', $A']\!]^\sharp\,\eta$ = **令** $\eta'\,[x] = \eta\,[x, A_0]$ **于 令 (**$\{[i] \mapsto v'\}$**,** v**)** = $[\![[u, S]$, **返回** $x]\!]^\sharp\,\eta'$ **于 (**$\{[i, A'] \mapsto v'\}$**,** v**)**$[\![[u, S, A_0]$, x=**创建(**u_1**)**$]\!]^\sharp\,\eta$ = **令** $\eta'\,[x] = \eta\,[x, A_0]$ **于 令 (** $\{[u_1, \emptyset] \mapsto v'\}$**,** v**)** = $[\![[u, S]$, x=**创建(**u_1**)**$]\!]^\sharp\,\eta'$ **于 (**$\{[u_1, \emptyset, A'] \mapsto v' \mid A' \in$ **新建** $^\sharp_{\mathcal{A}}\,u\,u_1\,A_0\}$**,** v**)**

$$\begin{aligned}\mathsf{init}^\sharp_{A'}\,\eta = {}& \mathbf{let}\,\eta'\,_ = \bot\,\mathbf{in}\\ & \mathbf{let}\,(\rho, v) = \mathsf{init}^\sharp\,\eta'\,\mathbf{in}\\ & \mathbf{let}\,\rho' = \{[a, Q, A'] \mapsto v' \mid ([a, Q] \mapsto v') \in \rho\}\\ & (\rho', v)\end{aligned}$$

并且，对于所有其他动作 act，按照

$$[\![[u, S, A_0], \mathsf{act}]\!]^\sharp\,\eta = \mathbf{let}\,\eta'\,[x] = \eta\,[x, A_0]\,\mathbf{in}\\ [\![[u, S], \mathsf{act}]\!]^\sharp\,\eta'$$

我们将这个使用第4节的分析实例化后的约束系统称为$\mathcal{C}^{\sharp\prime}$。

D 第5节细化的可靠性证明

考虑一个修改后的具体约束系统，其中我们还将 $\mathcal{A}$中的信息与每个未知量相关联:

$$\begin{aligned}&(\eta, \eta\,[u_0, \emptyset, A]) && \sqsupseteq (\{[a, Q, A] \mapsto \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\} \mid a \in \mathsf{M}, Q \in \mathcal{Q}_a\},\\ & && \quad \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\})\\ &(\eta, \eta\,[u', S \cup \{a\}, A']) && \sqsupseteq (\![[u, S, A_0], \text{锁}(a)\ A_1]\!]\,\eta)\\ &\quad \text{对于}\ (u,\ \text{锁}\ (a)\ \ u')\ \in \mathcal{E}, S \subseteq \mathsf{M}\ A' \in [\![u,\ \text{锁}\ (a)\ ]\!]^\sharp_{\mathcal{A}}\ (A_0, A_1)\\ &(\eta, \eta\,[u', S \setminus \{a\}, A']) && \sqsupseteq (\![[u, S, A_0], \text{解锁}(a)\ A']\!]\,\eta)\\ &\quad \text{for}\ (u, \mathsf{unlock}(a), u') \in \mathcal{E}, S \subseteq \mathsf{M}, A' \in [\![u, \mathsf{unlock}(a)]\!]^\sharp_{\mathcal{A}}(A_0)\\ &(\eta, \eta\,[u', S \cup \{a\}, A']) && \sqsupseteq (\![[u, S, A_0], x{=}\text{汇合}(x')\ A_1]\!]\,\eta)\\ &\quad \text{对于}\ (u, x{=}\text{汇合}(x')\ u')\ \in \mathcal{E}, S \subseteq \mathsf{M}\ A' \in [\![u, x{=}\text{汇合}(x')]\!]^\sharp_{\mathcal{A}}(A_0, A_1)\\ &(\eta, \eta\,[u', S, A']) && \sqsupseteq ([\![[u, S, A_0], \mathsf{act}]\!]\,\eta)\\ &\quad \text{for}\ (u, \mathsf{act}, u') \in \mathcal{E}, S \subseteq \mathsf{M}, A' \in [\![u, \mathsf{act}]\!]^\sharp_{\mathcal{A}}(A_0), \text{other actions act}\end{aligned}$$

右端项经过修改，以使用适当的摘要访问相应的未知量，并以类似于附录 C 中对抽象部分所做的方式，相应地重定向副作用。下面以互斥锁 a 的加锁为例:

$$\begin{aligned}[\![[u, S, A_0], \mathsf{lock}(a), A_1]\!]\,\eta'' = {}& \mathbf{let}\,\eta'\,[x] = \mathbf{if}\,[x] = [u, S]\,\mathbf{then}\,\eta''\,[u, S, A_0]\\ & \quad \mathbf{else}\ \ \eta''\,[x, A_1]\,\mathbf{in}\\ & \mathbf{let}\,(\emptyset, v) = [\![[u, S], \mathsf{lock}(a)]\!]\,\eta'\,\mathbf{in}\\ & (\emptyset, v)\end{aligned}$$

将该约束系统称为 $\mathcal{C}''$。

令 η' 为式 (8) 中约束系统 $\mathcal{C}'$ 的唯一最小解。我们据此构造一个映射 η''，将 $\mathcal{C}''$ 的未知量映射到 $2^{\mathcal{T}}$。

$$\begin{aligned}\eta''\,[u, S, A] &= \eta'\,[u, S] \cap \mathcal{T}_A && (u \in \mathcal{N}, S \subseteq \mathsf{M}, A \in \mathcal{A})\\ \eta''\,[i, A] &= \eta'\,[i] \cap \mathcal{T}_A && (i \in \mathcal{I}, A \in \mathcal{A})\\ \eta''\,[a, Q, A] &= \eta'\,[a, Q] \cap \mathcal{T}_A && (a \in \mathsf{M}, Q \in \mathcal{Q}_a A \in \mathcal{A})\end{aligned}$$

其中 $\mathcal{T}_A$ 表示满足 $\{t \mid t \in \mathcal{T}, \alpha_{\mathcal{A}}(t) = A\}$ 的局部轨迹子集。

命题 3. 若 $\alpha_{\mathcal{A}}$、新$^{\sharp}_{\mathcal{A}}$ 和 $[\![u,$ 作用$]\!]^{\sharp}_{\mathcal{A}}$ 满足第 *5* 节(4)、(5)和(6) 中的要求， η'' 是 $\mathcal{C}''$ 的最小解，当且仅当 η' 是 $\mathcal{C}'$的最小解。

证明。 命题 3 的证明采用不动点归纳。

现在让我们建立精化抽象约束系统 $\mathcal{C}^{\sharp'}$ 的解与精化具体约束系统 $\mathcal{C}''$的解之间的联系。设 $\eta^{\sharp'}$ 是 $\mathcal{C}^{\sharp'}$的一个解。我们沿用附录 B 中具体化 γ 的定义，据此构造映射 η''：

$$\begin{array}{lll}
\eta''[u,S,A] & = \gamma_{u,S}(\eta^{\sharp'}\,[u,S,A]) \cap \mathcal{T}_A & (u \in \mathcal{N}, S \subseteq \mathsf{M}, A \in \mathcal{A}) \\
\eta''[i,A] & = \bigcup \left\{ \gamma_i((\eta^{\sharp'}\,[i^{\sharp},A])) \mid i \in \gamma_{\mathcal{V}^{\sharp}}\, i^{\sharp} \right\} \cap \mathcal{T}_A & (i \in \mathcal{I}, A \in \mathcal{A}) \\
\eta''[a,Q,A] & = \gamma_a((\eta^{\sharp'}\,[a,Q])) \cap \mathcal{T}_A & (a \in \mathsf{M}, Q \in \mathcal{Q}_a, A \in \mathcal{A})
\end{array}$$

定理 7。 假设 $\alpha_{\mathcal{A}}$、*new*$^{\sharp}_{\mathcal{A}}$ 和 $[\![u,$ *act*$]\!]^{\sharp}_{\mathcal{A}}$ 满足要求(4)、(5)和(6)。那么，精化约束系统的任意解相对于收集式局部轨迹语义都是可靠的。

证明。 根据命题 3， $\mathcal{C}''$ 的任意解相对于 $\mathcal{C}'$都是可靠的；根据命题 1， $\mathcal{C}'$ 的任意解相对于 $\mathcal{C}$所指定的收集式局部轨迹语义也是可靠的。因此，只需证明，从 $\mathcal{C}^{\sharp'}$ 的一个解构造出的映射 η'' 是 $\mathcal{C}''$的一个解。

我们通过不动点归纳验证，对于 $\mathcal{C}''$ 的最小解的第 j 次近似 η^j ，$\eta^j \subseteq \eta''$ 成立。为此，我们验证起始点 u_0处的空锁集，以及所有 $A \in \mathcal{A}$

$$\begin{array}{l}
(\{[a,Q,A] \mapsto \{t \mid t \in \mathsf{init}, Q \in \mathcal{Q}_a, A = \alpha_{\mathcal{A}}(t)\}\}, \{t \mid t \in \mathsf{init}, A = \alpha_{\mathcal{A}}(t)\}) \\
\quad \subseteq (\eta'', \eta''\,[u_0, \emptyset, A])
\end{array}$$

并且对于控制流图中每条边 $(u,$ act $v)$、每个可能的锁集 S和摘要A，都有

$$[\![[u,S,A_0], \mathsf{act}]\!]\,\eta^{j-1} \subseteq (\eta'', \eta''\,[v,S',A'])$$

成立（其中，按照约束系统的要求，还会将额外参数 A_1 和 A' 传递给右端项）。我们以对应于某个互斥锁 a的锁操作的一条边为例。我们有

$$\begin{array}{lll}
[\![[u,S,A_0], \mathsf{lock}(a), A_1]\!]\,\eta'' & = \mathbf{let}\ \eta_c\,[x] = \mathbf{if}\ [x] = [u,S]\ \mathbf{then}\ \eta''\,[u,S,A_0] & \\
 & \quad \mathbf{else}\ \eta''\,[x,A_1]\ \mathbf{in} & \\
 & \mathbf{let}\ (\emptyset, v) = [\![[u,S], \mathsf{lock}(a)]\!]\,\eta_c\ \mathbf{in} & \\
 & (\emptyset, v) & \\
[\![[u,S,A_0], \mathsf{lock}(a), A_1]\!]^{\sharp}\,\eta^{\sharp} & = \mathbf{let}\ \eta^{\sharp}_a\,[x] = \mathbf{if}\ [x] = [u,S]\ \mathbf{then}\ \eta^{\sharp}\,[u,S,A_0] & \\
 & \quad \mathbf{else}\ \eta^{\sharp}\,[x,A_1]\ \mathbf{in} & \\
 & \mathbf{let}\ (\emptyset, v^{\sharp}) = [\![[u,S], \mathsf{lock}(a)]\!]^{\sharp}\,\eta^{\sharp}_a\ \mathbf{in} & \\
 & (\emptyset, v^{\sharp}) &
\end{array}$$

根据归纳假设，$\eta^{j-1} \subseteq \eta''$。然后，根据针对 η_c 和 $\eta_a^\sharp$ 的构造，对于未细化约束系统中的每个未知量 x，

$$\eta_c\, x \subseteq \gamma_x(\eta_a^\sharp\, x)$$

因此，根据定理 6

$$v \subseteq \gamma_{v,S\cup\{a\}}(v^\sharp)$$

由于 $\eta^\sharp$ 是 $C^{\sharp'}$ 的解，因此 $v^\sharp \sqsubseteq \eta^\sharp\,[v, S\cup\{a\}, A']$。此外，根据式 (4)，$\left|[\![u,\ \mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}(A_0, A_1)\right| \leq 1$ 成立。综上，

$$v = v \cap T_{A'} \subseteq \gamma_{v,S\cup\{a\}}(v^\sharp) \cap T_{A'} \subseteq \gamma_{v,S\cup\{a\}}(\eta^\sharp\,[v, S\cup\{a\}, A']) \cap T_{A'} = \eta''\,[v, S\cup\{a\}, A']$$

由于该约束不会产生副作用，因此该论断成立。对于所有其他约束，证明过程类似。

E 进一步提高第 7 节分析的精度

第 7 节所述的分析并未利用构成未知量 $[a, Q, (i, C)]$、$[i, C]$ (a 一个互斥锁，$Q \in \mathcal{Q}_a$, i 一个抽象线程 id) 的组件 C。利用这些信息，可以排除另一类写操作——即自身线程或其祖先线程被创建之前，由创建线程执行的写操作。被创建线程在自身创建之前可能从创建线程读取的任何写操作，已经计入被创建线程的初始状态中，因此只需考虑当前线程创建之后发生的写操作。为此，令

$$\begin{aligned}&\mathsf{acc}\,((i, C), (L, W, r))\,(i', C') = (\mathsf{unique}\, i \wedge i = i') \vee \\ &\quad (\mathsf{lcu_anc}\, i'\, i = i' \wedge \nexists \langle u, u'\rangle \in C' : ((i' \circ \langle u, u'\rangle) = i) \vee \mathsf{may_create}\,(i' \circ \langle u, u'\rangle)\, i)\end{aligned}$$

例 *12*。考虑以下程序，其中 $\mathcal{M}[g] = \{a, m_g\}$ 和 $\mathcal{M}[h] = \{a, m_h\}$，并假设 $\mathcal{Q}_a = \{\{g, h\}\}$。

```
main:
  锁(a);
  g = 5; h = 8;
  解锁(a);
  锁(a);
  g = 10; h = 10;
  解锁(a);
  x = 创建(t1);
  锁(a);
  克= 20; 时 = 20;
  解锁(a);
  y = 汇合(x);
  锁(a);
  // 断言(g==20); (2)
  解锁(a);
```

```
t1:锁(a);// 断言(g==
h); (1)解锁(a);返回
0;
```

在这里，如果利用有关已遇到哪些创建边的额外信息，分析就能确定线程 $t1$ 中的断言 (1) 成立。□

当线程过去的写操作传播到它创建的线程，并在汇合时再传播回创建线程时，分析仍可能遭受不必要的精度损失。如果被创建线程没有写入某些全局变量，而原线程在此期间已在本地覆盖这些变量，就可能发生这种情况。

示例 *13.* 再看例 12。这里无法证明断言 (2)，因为 t_1 中 $L(a, \{g, h\})$ 的过时信息（其中包含 g 可能为 10 这一事实）会在汇合时并入主线程。可以通过跟踪每个线程可能已写入的全局变量集合 W 来避免这种情况；该集合包括此线程或它所汇合的任何线程可能写入的变量。随后，只有在至少一个受保护的全局变量已被写入时，才将已汇合线程的信息并入。

这里不详细介绍这项改进，但我们在实现中采用了它。

例如，自身线程中可维护其他有用的抽象信息：针对每个被创建线程 t'，跟踪自其创建 $W_C\, t'$以来，自身线程的汇合局部 部分可能写入过的全局变量集合。这样，在汇合t'时，对于满足 $Q \cap W_C\, T' = \emptyset$的互斥锁 a和簇 $Q \in \mathcal{Q}_a$，已汇合线程的 $L(a, Q)$ 信息必然包含最新信息，而自身线程的 $L(a, Q)$ 信息则可以丢弃。

F 线程汇合的分析与利用

这里详细介绍增强后的分析方法，该方法还会在局部状态中跟踪线程id集合J，其中这些线程的汇合操作已确定在局部轨迹的汇合局部 部分中被调用。回顾一下考虑到 J 的 acc 精化定义：

$$\mathsf{acc}\,((i, C), (J, L, W, r))\,(i', C') = \mathsf{unique}\, i' \wedge (i = i' \vee i' \in J)$$

接下来给出有趣的 右端项；其他项只需在局部传播此值，并将其作为附加参数传递给 acc。

$$\begin{aligned}[[u, S, (i, C)], \mathsf{return}\, x, (i, C)]]^\sharp \eta = \;& \mathbf{let}\ (J, L, W, r) = \eta\,[u, S, (i, C)]\ \mathbf{in} \\ & \mathbf{let}\ v = \left([\![\mathsf{ret} \leftarrow x]\!]^\sharp_{\mathcal{R}}\, r \right)\Big|_{\{\mathsf{ret}\}}\ \mathbf{in} \\ & \mathbf{let}\ \rho = \{[(i, C)] \mapsto (J, L, v)\}\ \mathbf{in} \\ & (\rho, (L, W, r))\end{aligned}$$

$[\![[u, S, (i, C)], x'$=合并$(x), (i', C')]\!]^\sharp \eta =$ 令 $(J, L, W, r) = \eta\,[u, S, (i, C)]$ 在 if $(i' \sqcap (($unlift $r)\, x) = \bot) \vee$acc $((i, C), (J, L, W, r))\, (i', C')$ then $\bot$else 令 $(J', L', v) = \eta[(i', C')]$ 在 令 $r' = [\![x' \leftarrow^\sharp ($unlift $v)$ ret$]\!]^\sharp_{\mathcal{R}} r$ 在 $(\emptyset, (J \cup J' \cup \{i'\}, L \sqcup L', W, r'))$

我们指出，在分析过程中执行线程合并时，如果可能调用合并操作的线程标识有多个，则每个线程都会对应一个约束，合并边之后的值将汇合起来，以得到合并后的新值。此处， J 对应的格汇合 是交集。这处理了多个线程被不同线程合并的情况，而这些线程的标识可能存储在 x中。

回顾一下，函数 acc 现在定义如下：

$$\mathsf{acc}\left((i,C),(J,L,W,r)\right)\left(i',C'\right) = \quad \mathsf{unique}\, i' \wedge (i = i' \vee i' \in J)$$

示例 *14.* 考虑以下程序，其中 $\mathcal{M}[g] = \{a, m_g\}$ and $\mathcal{M}[h] = \{a, m_h\}$，并假设 $\mathcal{Q}_a = \{\{g, h\}\}$。

```
main: t1 : x = create(t1); lock(a); lock(a); g
 = 4; h = 8; g = 20; h = 20; unlock(a);
unlock(a); x = ?; y = 汇合(x); lock(a);
lock(a); g = x; h = x; // 断言(g==h); (1)
unlock(a); g = 5; h = 5; 返回 0; unlock(a);
lock(a); // 断言(g==5); (1) unlock(a);
```

这里，两个断言都可以得到证明。在 (1) 处，线程 t_1 必须已合并。其最后一次写入已计入 $L\,(a, \{g, h\})$，因此不会查询抽象关系 $g = h$ 不成立的未知量 $[a, \{g, h\}, t_1]$。由于 L 中的更新具有破坏性，在 $main$ 线程向 g 写入 5 后，它读取 g 时也只会读到这个值，这意味着 (2) 也得到了证明。□

为了提高精度，可以在解锁时将 J 必须已合并的线程集合与受保护的全局变量一同发布。这样，如果某个线程的所有写操作都发生在第一个线程已经必须 合并之后，前者就不必从该线程读取。通过将 J 作为 $\mathcal{A}$中的一个额外组成部分进行跟踪，还可以进一步提高精度。

G 使用汇合的分析的可靠性证明

证明按以下方式进行。我们给出一个修改后的$\mathcal{A}$ 实例，除了跟踪第 6 节中的信息外，还会针对每个互斥锁跟踪以下信息：哪个抽象线程标识（按照与第 6 节相同的方式计算）执行了最后一次解锁，且该解锁紧接着一次对受该互斥锁保护的全局变量进行的线程局部写入；以及自程序开始以来发生了多少次此类解锁。为此，我们选择 $\mathcal{A} = (I^\sharp \times 2^P) \times (\mathrm{M} \to (\mathbb{N}_0 \times (I^\sharp \times 2^P)))$. 为方便起见，这里我们暂时也允许对控制点进行拆分

，即使对于一个无限的集合也是如此。例如，互斥锁a的抽象函数 $[\![u,\mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}$ 可表示为

$$[\![u,\mathsf{lock}(a)]\!]^\sharp_{\mathcal{A}}\,((i,C),H)\,((i',C'),H') =$$

$$\begin{cases} \emptyset & \neg\mathsf{can_be_started}\,(i,C)\,(M'\,a)_2 \\ \left\{\left(\begin{array}{l}(i,C),\\ \{a \mapsto H\,a \mid a \in \mathsf{M}, (M\,a)_1 \geq (H'\,a)_1\}\\ \cup\{a \mapsto H'\,a \mid a \in \mathsf{M}, (M\,a)_1 < (H'\,a)_1\}\end{array}\right)\right\} & \text{else}\end{cases}$$

其中，$(\cdot)_k$ 是访问元组第 k 个分量的简写。互斥锁 $[a,Q,((i,C)\ H)]$ 处未知量的抽象值的具体化定义为：它包含所有以 unlock(a) 结束的局部轨迹，其中对全局变量 $\mathcal{G}\,[a]$ 的最后一次线程局部写入由线程 $id(H\,a)_2$ 执行。

对于局部轨迹，我们再次引入函数 $\beta\colon \mathcal{T} \to (2^{\mathcal{I}} \times (\mathsf{M} \to (\mathcal{V}\mathrm{ars} \to \mathcal{V}) \times 2^{\mathcal{G}} \times (\mathcal{V}\mathrm{ars} \to \mathcal{V}))$，随后将用它来定义适当的具体化函数。令 $\mathsf{last_tl_write}_g\colon \mathcal{T} \to ((\mathbb{N}_0 \times \mathcal{N} \times \Sigma) \times \mathcal{A} \times (\mathbb{N}_0 \times \mathcal{N} \times \Sigma)) \cup \{\bot\}$为一个函数，用于提取对 g 的最后一次线程局部写入（如果存在），否则返回 $\bot$ 。

如果互斥锁 a 存在线程局部加锁，则也存在 a 的 最后一次线程局部加锁。令 $\mathsf{last_tl_lock}_a\colon \mathcal{T} \to ((\mathbb{N}_0 \times \mathcal{N} \times \Sigma) \times \mathcal{A} \times (\mathbb{N}_0 \times \mathcal{N} \times \Sigma)) \cup \{\bot\}$为一个函数，用于提取 a 的这次最后线程局部加锁（如果存在），否则返回 $\bot$。$\mathsf{last_tl_unlock}_a$ 也同理。

设 $\mathsf{jl_joins}\colon \mathcal{T} \to 2^{((\mathbb{N}_0 \times \mathcal{N} \times \Sigma) \times \mathcal{A} \times (\mathbb{N}_0 \times \mathcal{N} \times \Sigma))}$ 为一个函数，用于从局部轨迹中提取所有与该轨迹汇合局部的汇合调用所构成的集合。

设 $\mathsf{jl_unlock_after_write}_a\colon \mathcal{T} \to ((\mathbb{N}_0 \times \mathcal{N} \times \Sigma) \times \mathcal{A} \times (\mathbb{N}_0 \times \mathcal{N} \times \Sigma)) \cup \{\bot\}$ 为一个函数，用于针对互斥锁 a，提取在 $\mathcal{G}\,[a]$ 中对全局变量的最后一次写入之后紧接着发生的第一次与该写入汇合局部的 unlock(a) 动作。如果互斥锁 a 当前处于持有状态，$\mathsf{jl_unlock_after_write}_a$ 只考虑以自线程获取 a 为结尾的子轨迹。

设 $\mathsf{unlock_after_write}_a\colon \mathcal{T} \to ((\mathbb{N}_0 \times \mathcal{N} \times \Sigma) \times \mathcal{A} \times (\mathbb{N}_0 \times \mathcal{N} \times \Sigma)) \cup \{\bot\}$ 为一个函数，用于针对互斥锁 a，提取在 $\mathcal{G}\,[a]$ 中对全局变量的最后一次写入之后紧接着发生的第一次 unlock(a) 动作。

此外，对于出现在局部轨迹 t 中的节点 $\bar{u}'$，令 $(\bar{u}')\downarrow_t$ 表示以该节点结束的局部子轨迹。

$$\begin{array}{lll}
\beta\,t = (J,L,W,r) \text{ where} & & \\
J & = \{\sigma_i\,x' \mid ((i,u_i,\sigma_i), x{=}\mathsf{join}(x'), _) \in \mathsf{jl_joins}\,t\} & \\
L & = \{(a,Q) \mapsto \{x \mapsto t'(x) \mid g \in (\mathcal{G} \cup \mathcal{X})\} \mid a \in \mathsf{M}, Q \in \mathcal{Q}_a, & \\
 & \qquad \mathsf{jl_unlock_after_write}_a\,t = \bot, t' = (u_0,0,\sigma_0)\downarrow_t\} & \\
 & \quad \cup\{a \mapsto \{x \mapsto t'(x) \mid g \in (\mathcal{G} \cup \mathcal{X})\} \mid a \in \mathsf{M}, & \\
 & \quad \mathsf{jl_unlock_after_write}_a\,t = (\bar{u}_i,_,_), t' = (\bar{u}_i)\downarrow_t\} & \\
W & = \bigcup_{a\in\mathsf{M}}\{g \mid g \in \mathcal{M}[a], \mathsf{last_tl_write}_g\,t = (\bar{u}_i,_,_), & \\
 & \qquad ((\mathsf{last_tl_unlock}_a\,t = (\bar{u}_j,_,_) \wedge \bar{u}_j \leq \bar{u}_i) \vee \mathsf{last_tl_unlock}_a\,t = \bot)\} & \\
r & = \{x \mapsto t(x) \mid x \in (\mathcal{X} \cup \mathcal{G})\} &
\end{array}$$

基于此，我们为约束系统中的未知量提供了专用的具体化函数：

$$
\begin{aligned}
&\gamma_{u,S,((i,C),H)}(J^\sharp, L^\sharp, W^\sharp, r^\sharp) = \{t \mid t \in \mathcal{T}_S \cap \mathcal{T}_{(i,C)}, \text{位置 } t = u, (J, L, W, r) \\
&= \beta\, t, \\
&\bigcup\{\gamma_{\mathcal{I}^\sharp} i \mid i \in J^\sharp\} \subseteq J \qquad , \\
&\forall a \in \mathbf{M}\ Q \in \mathcal{Q}_a \colon \exists v \colon (L\,(a, Q) \cup \{\text{返回 } \mapsto v\}) \in \gamma_{\mathcal{R}}\,(L^\sharp\,(a, Q)), W^\sharp \supseteq W, \exists v \colon \\
&(r' \cup \{\text{返回 } \mapsto v\}) \in \gamma_{\mathcal{R}}\, r^\sharp\}\gamma_{((i,C),H)}(J^\sharp, L^\sharp, r^\sharp) = \{t \mid t \in \mathcal{T}_{(i,C)}, \text{最后 } t = (\\
&\text{返回 } x), (J, L, W, r) = \beta\, t, \\
&\bigcup\{\gamma_{\mathcal{I}^\sharp} i \mid i \in J^\sharp\} \subseteq J \qquad ,\forall a \in \\
&\mathbf{M} \colon \exists v \colon (L\, a \cup \{\text{返回 } \mapsto v\}) \in \gamma_{\mathcal{R}}\,(L^\sharp\, a), \exists v \colon (r' \cup \{\text{返回 } \mapsto v\}) \in \gamma_{\mathcal{R}}\, r^\sharp\}\gamma_{a,Q,} \\
&{}_{((i,C),H)}(r) = \{t \mid t \in \mathcal{T}, (\text{最后} t = \text{解锁}(a), \text{解锁_之后_写入}_a\, t = (\bar{u}_i, _, _), t' = \\
&(\bar{u}_i) \downarrow_t, t' \in \mathcal{T}_{(H\, a)_2}, (J, L, W, r) = \beta\, t', \exists v \colon (\beta\, t \cup \{\text{返回 } \mapsto v\}) \in \gamma_{\mathcal{R}}\, r \\
&\} \cup \{t \in \mathcal{T}_{(H\, a)_2} \cap \text{初始化}, (J, L, W, r) = \beta\, t, \exists v \colon (\beta\, t \cup \{\text{返回 } \mapsto v\}) \in \gamma_{\mathcal{R}}\, r\}
\end{aligned}
$$

并指出这些具体化函数是单调的。

因此，只要自线程自获取 a 以来确定没有向 $\mathcal{G}\,[a]$ 中的任何全局变量写入，就可以舍弃抽象约束系统中的副作用。只要在给定的 unlock(a) 处满足 $W^\sharp \cap \mathcal{G}\,[a] = \emptyset$，后一条件就成立。根据同样的论证，只要自线程自获取 a 以来确实向 $\mathcal{G}\,[a]$ 中的某个全局变量写入，那么其线程 id 就与在线程向 $\mathcal{G}[a]$ 中的任何全局变量写入之后执行最后一次 unlock(a) 的线程的线程 id 一致。这一构造的正确性证明与定理 1 和定理 2 的证明遵循相同思路。

最后一步，将互斥锁处的控制点拆分简化为只考虑自线程在紧接着对受 a 保护的全局变量进行最后一次写入之后的最后一次 unlock(a) 处的线程 id，即 $H\, a$。同样，程序点和线程 id 处的控制点拆分也简化回第 6 节提出的原始 $\mathcal{A}$ 信息。我们指出，这样一来，无限的控制点拆分再次消失。□

H 单元素簇

以下程序说明，不能舍弃单元素簇。假设 a 保护 g 和 h。

```
main: t1: x = 创建(t1); 加锁(a); y = 创建(t2);
g =- 1; 加锁(a); //断言(g<=h); (3) h = 31; 解锁(
a); 解锁(a); 返回 0; 加锁(a); h = 12; t2: 解锁
(a); 加锁(a); 加锁(a); h = ?; // 断言(g<=h); (1)
h = 12; // 断言(h==12); (2) 解锁(a); 解锁(a); 返回
0;
```

当仅使用簇 $\mathcal{Q}_a = \{\{g, h\}\}$ 运行簇化分析时， t_1 中 unlock(a) 处的副作用保留了关系 $g \leq h$，这意味着断言 (1) 和 (3) 成立。在未知量 $[a, \{g, h\}, t_1]$ 处，不会保留关于 h 值的精确信息。因此，当 $main$线程第三次执行 lock(a) 时，无法验证断言 (2)。不过，如果簇化分析还额外跟踪簇 $\{h\}$，则会在 $[a, \{h\}, t_2]$ 处记录 $h = 12$，而在 $[a, \{h\}, t_1]$ 处不记录任何信息。因此，可以验证断言 (2)。

I 基准测试的扩展描述

这里，我们将进一步介绍第 9 节中所呈现的一些基准测试。

I.1 Goblint 基准测试套件

我们考察了 [48] 中的基准测试。我们使用簇配置，通过工具生成断言并将其添加到基准测试中。我们在实验中排除了 ypbind，因为它会通过未知的函数指针生成一个线程，而我们的线程标识符分析无法处理这种情况。我们在生成的基准测试上运行了自己的工具，以及 NR-Goblint 和 Duet。结果见表 3。对于九个基准测试，NR-Goblint 和我们分析器的所有配置都能证明全部断言，而我们运行 Duet 得出的结果均无效。表中将这些基准测试的结果汇总在一行。对于其余三个基准测试，使用八边形配置而非区间配置，能够证明更多断言。有趣的是，NR-Goblint 能够证明的断言比我们的区间配置更多。将线程标识符分析添加到我们的工具后，在 pfscan 上证明的断言更多。事实上，对于这组基准测试，使用*TID*配置可以证明所有断言。

表 3：在 Schwarz 等人 [48], 的基准测试集上运行我们的分析器（使用所介绍的分析）、NR-Goblint [48]（使用区间）和 Duet [19] 的结果；我们使用第8节中的分析自动生成了这些基准测试的不变式。勾号（√）表示所有不变式均已证明，否则给出已证明的不变式数量。为简洁起见，我们汇总了除 Duet 之外所有分析器均已完全证明的基准测试结果；Duet 报告的不变式数量少于实际数量，或运行超时，因此我们将这些结果视为无效（—）。

		我们的分析器					
基准测试	不变式	区间（第 4 节）	八边形（第 4 节）	TIDs（第 7 节）	簇（第 8 节）	NR-Goblint 带区间	Duet
pfscan	221	14	32	✓	✓	165	—
aget	10	2	✓	✓	✓	✓	—
ctrace	1448	1130	✓	✓	✓	1407	—
（其他）	0–200	✓	✓	✓	✓	✓	—

I.2 WattsBenchmark 套件

虽然我们无法运行 Watts，但我们针对 [31] 中报告的基准测试运行了分析。我们采用了 GitHub代码仓库[6] 中提供的基准测试。本文提出的基准测试由两组包含多线程代码和断言的 C源文件构成。与所考察的其他基准测试集不同，这些基准测试文件中有相当数量的 assert(0)语句，而这些语句所在的代码本应被证明不可达。因此，对于这些基准测试，我们将可证明不可达的断言数计入所报告的已验证断言数中。

第一组基准测试包含 37 个来自不同来源、为 [31] 改编的 C文件。我们原样采用了这组基准测试，但移除了 wdt977_02benchmark 中一个明显放错位置的分号；该分号会导致具体执行中一个 assert(0) 可达。分析设置 *TIDs* 成功验证了这 31 个基准测试中的所有断言均不会失败。其余六个基准测试文件中共有七个断言无法证明；这些文件包含依赖于数据的线程同步，而我们的工具无法处理这种情况。

第二组基准测试包含基准测试集第一组中的 i8xx_tco_03 的五个版本，这些版本经过插桩，可创建不同数量的线程。在 [31], 中，基准测试创建的线程数从 30 到 70 不等。仓库中有两个文件会创建 40 个线程，却没有文件会创建 30 个线程，因此有一个测试用例是多余的。于是，我们从 i8xx_tco_03_thr01 中移除了十个线程创建操作。我们还将该文件中的函数存根数量调整为与其他基准测试文件相同。

我们的工具的运行时间见表4。在该基准测试中，耗时最长的分析约需两秒才能完成。虽然 [31], 中没有报告确切的运行时间，但该论文中的图 11 显示，

[6] https://github.com/markus-kusano/watts

表4：我们的分析器在 [31] 中五个可扩展性基准测试上的运行时间（秒）。第二列表示基准测试程序中的具体线程数（包括主线程）。运行时间明显短于 Watts 报告的结果，但测试所用的硬件不同。

		我们的分析器			
基准测试	#线程	区间（第 4 节）	八边形（第 4 节）	TIDs（第 7 节）	簇（第 8 节）
i8xx_tco_03_thr01	31	0.15	0.16	0.43	0.51
i8xx_tco_03_thr02	41	0.17	0.17	0.66	0.77
i8xx_tco_03_thr03	51	0.19	0.19	1.01	1.11
i8xx_tco_03_thr04	61	0.20	0.20	1.35	1.45
i8xx_tco_03_thr05	71	0.23	0.22	1.83	1.99

他们耗时最长的分析运行时间接近 400 秒，而在创建 70 个线程的基准测试中，耗时最短的配置也需要 200 秒以上。需要指出的是，虽然 [31]中报告的 Watts 运行时间是在另一台机器上测得的，因此这些数字不能直接比较，但由于量级相差很大，这种比较仍然有意义。

I.3 与 Duet 的比较

我们曾考虑在 [19], 提出的基准测试集上与 Duet 进行比较，但遇到了一些问题。基准测试档案取自 Duet 网站[7]。该档案包含：（1）一组 Linux 设备驱动程序的 C 源文件，但不含测试框架函数；（2）这些驱动程序的一个版本，以二进制 goto 文件的形式提供，并经过 DDVerify 插桩，且使用未知版本的 Cbmc 编译。然而，当前版本的 Duet 不接受 goto 文件作为输入，其他纳入考察的工具也都不接受。此外，我们也无法使用当前版本的 Cbmc 反编译这些 goto 文件。因此，本文的实验未使用这些基准测试。

虽然我们成功地在一些基准测试上运行了 Duet，但该工具的配置在另一些基准测试上无法得出有效结果：为处理包含函数调用的代码，Duet 依赖内联。撰写本文时可用的最新版本 Duet 中实现的内联功能无法处理一些示例，因此我们联系了作者，作者向我们发送了一个修复版的内联模块。在实验中，我们使用这一内联实现（并启用了八边形域）运行该工具；然而，在某些情况下，我们的工具配置仍报告可达断言数量过少，这表明该工具未考虑某些可达代码。因此，对于这些基准测试，我们不报告 Duet 的结果。此外，如果运行未能在 15 分钟内完成，我们也不报告 Duet 的结果。

[7] http://duet.cs.toronto.edu/