arXiv:2602.17142v1 [cs.LO] 19 Feb 2026

# 使用条件写入域生成依赖-保证条件

James Tobler[0000−0002−1205−3455] 和 Graeme Smith[0000−0003−1019−4761]

澳大利亚国防科学与技术集团；昆士兰大学电气工程与计算机科学学院，澳大利亚布里斯班
james.tobler@uq.edu.au

**摘要。** 抽象解释已被证明是用于并发程序线程模块化验证的一种颇具前景的技术。其核心是生成符合用户选择结构的干扰，形式为依赖-保证条件。在这项工作中，我们介绍一种称为条件写入域的结构，专为只需确定每个线程写入特定变量的条件的程序而设计。我们在一个新颖的抽象解释框架中形式化了这一分析。该框架具有高度模块化的特点，且易于扩展，以涵盖其他依赖-保证条件结构。我们形式化了该方法的两个版本，并在一种简单的编程语言上评估了它们的实现。

**关键词：** 抽象解释 · 并发 · 依赖-保证。

## 1 引言

抽象解释已被证明是一种可扩展的技术，可用于自动验证顺序程序 [2, 12]。它的独特优势在于提供了一个称为 抽象域的简单参数，用于约束程序状态上所生成中间断言的结构。只要为目标程序提供合适的抽象域，就能大规模生成证明。遗憾的是，用户并不总能轻松推断出合适的抽象域。

对于并发程序，由依赖-保证条件所刻画的线程之间的干扰也必须加以生成。我们的工作基于如下猜想：与顺序程序所需的中间断言不同，许多足以验证并发程序的干扰往往在各种并发程序中呈现相似的结构。在本文中，我们用一个称为条件写入域的抽象域来刻画其中一种结构；该域适用于只需依据变量可能被更新时所处的条件来推理线程干扰、而不必考虑它们被更新为何值的情形。例如，只需确立如下事实即可：“线程 $A$ 可在 $z = 0$ 时写入 $x$ ，并可在 $z = 1$ 时写入 $y$，但从不写入 $z$。”

为了在抽象域中精确刻画这一结构，我们提出了一个将抽象解释应用于并发程序的新颖框架。在诸如 [13], 等现有框架中，线程干扰被抽象到关于程序变量及其带撇对应物的数值抽象域中，其中带撇变量表示该变量在干扰后状态中的值。这种方法的局限性在于，带撇变量与不带撇变量被一视同仁——也就是说，变量是否带撇并不影响它在抽象域元素结构中可能所处的位置。因此，它们无法表示对前状态与后状态施加不同约束的结构，例如我们的条件写入域。这促使我们引入一类新的抽象域，称为干扰域，条件写入域即是其中一例。

在抽象解释框架中，传统抽象域将状态集合上的一个格 $\mathcal{D}$ 与一个传递函数 相结合，该函数推导出一个元素 $\mathcal{D}$，作为程序指令的保守前置条件或后置条件。相反，干扰域将状态对集合上的一个格 $\mathcal{I}$（表示依赖条件和保证条件）与两个函数相结合，这两个函数都以具有格 $\mathcal{D}$ 的抽象域为参数。第一个函数称为转换，它将某个 $d \in \mathcal{D}$ 和一项赋值映射为元素 $\mathcal{I}$，该元素对在任何由 $d$ 所涵盖的状态下执行给定赋值可能引发的状态转换集合进行过近似。第二个函数称为稳定化，它将某个 $d \in \mathcal{D}$ 和 $i \in \mathcal{I}$ 映射为元素 $\mathcal{D}$，该元素对从 $d$ 中任一初始状态出发、在 $i$ 内一步可达的状态集合进行过近似。在一起使用时，这两个函数可分别用于在线程模块化分析期间推导和应用线程干扰（分别对应保证条件和依赖条件）。

除了这两个函数之外，我们还为条件写入域配备了一个函数闭合，用于过近似线程干扰的传递闭包。通过使用此函数使某些 $i \in \mathcal{I}$ 具有传递性，我们只需调用一次稳定化，就能考虑 $i$ 中一个或多个转换的影响，从而简化应用干扰的过程，代价是需要推导这个传递闭包。我们通过实现并评估分析的两个版本来判断这种权衡是否值得：一个强制传递性，另一个则不强制。

我们在第2节介绍一种用于评估分析的简单编程语言。在第3节中，我们定义一个抽象状态域，作为分析的参数；在第4节中，我们定义干扰域，其中包括转换、稳定化和闭合函数。第5节将形式化有传递性和无传递性的分析，第6节则在该简单编程语言上评估这些分析。我们在第7节讨论相关工作，并在第8节作结。

## 2 编程语言

我们定义 $\mathcal{V}$ 为程序变量集合， $\mathcal{T}$ 为组成该程序的线程标识符集合， $\mathcal{S}$ 为所有状态的集合，其中每个状态都将每个

变量映射到 $\mathcal{V}$ 中的一个常量。每个线程 $t \in \mathcal{T}$都表示语言$inst$中的一个顺序程序：

指令 ::= 指令；指令 | $x_1, ..., x_n := e_1, ..., e_n$ | 如果条件 则指令 否则指令|当条件 执行指令 |跳过

其中，条件是所有布尔表达式的集合，对于每个 $i \in \{1, ..., n\}$， $x_i \in \mathcal{V}$ 和 $e_i$ 都是一个表达式。指令 指令的具体语义记为：

$$exec : inst \times \mathcal{S} \to \mathcal{S}$$

其中，$exec(c, s)$ 表示在状态 $s$中执行指令 $c$ 后到达的程序状态。在我们的简单语言中，所有赋值语句都是原子的，所有分支条件和循环守卫的求值也是原子的。

对于每个线程 $t \in \mathcal{T}$，我们还令 $\mathcal{V}_t \subseteq \mathcal{V}$ 表示依赖条件 $t$ 应在其上定义的变量集合。使用 $\mathcal{V}_t$，我们可以避免在考虑依赖所指定的干扰时，使为 $t$ 生成的中间断言变得复杂。通常， $\mathcal{V}_t$ 设为全局变量与 $t$的局部变量的并集。然而，在某些情况下，将其他线程的局部变量纳入 $\mathcal{V}_t$[1]可以提高精度；反过来，将 $\mathcal{V}_t$ 限制为仅包含与我们的特定验证目标相关的变量，则可以提高效率。大多数现有的并发程序抽象解释框架不区分局部变量和全局变量 [13, 11, 9]，因此实际上对所有 $t \in \mathcal{T}$都有 $\mathcal{V}_t = \mathcal{V}$。

## 3 抽象状态域

我们的分析以一个带有格 $\mathcal{D}$ 的抽象域为参数，该格表示状态集合。为将其与干扰域区分开来，我们称之为状态域，并将 $\mathcal{D}$ 称为状态格。对于 $d_1$， $d_2 \in \mathcal{D}$，我们用 $d_1 \sqsubseteq^\sharp d_2$表示 $d_1$ 在格上小于或等于 $d_2$ ，也就是说，它表示相同或更少的状态。（从现在起，我们可能会说 $d_1$ 比 $d_2$“更强”，或说 $d_2$ 比 $d_1$“更弱”。）格的最大元 $\top^\sharp$ 表示所有状态的集合，底元 $\bot^\sharp$ 表示空集。除了上述内容以及常见的并 $\sqcup^\sharp$ （最小上界）和交 $\sqcap^\sharp$ （最大下界）算子外，我们的状态域还配备了以下基本函数：

$\gamma^\sharp : \mathcal{D} \to \mathcal{P}(\mathcal{S})$，其中 $\gamma^\sharp(d)$ 是 $d$ 所表示的状态集合。$\mathbb{S}^\sharp$： 指令 $\times \mathcal{D} \to \mathcal{D}$，其中 $\mathbb{S}^\sharp(c, d)$ 是相对于抽象前状态 $d$，指令 $c$ 的抽象后状态。$\mathbb{B}^\sharp$： 条件 $\times \mathcal{D} \to \mathcal{D}$ ，其中 $\mathbb{B}^\sharp(b, d)$ 是抽象状态，表示 $d$ 中所有满足条件 $b$ 的状态。

[1] 在标准的依赖-保证方法中，这类似于将局部变量转换为“辅助变量” [15, 17]。

$\lightning$ : $\mathcal{D} \times \mathcal{P}(\mathcal{V}) \to \mathcal{D}$，其中 $d \lightning V$ 是一个弱于 $d$ 的抽象状态，它至少包含这样一组状态：当 $V$ 中所有变量的值均不受约束时，从 $d$ 获得的状态。形式化地说：

$$\gamma^\sharp(d \lightning V) \supseteq \left\{ s \in \mathcal{S} \mid \exists s_0 \in \gamma^\sharp(d) : \forall v \in \mathcal{V} \setminus V : s(v) = s_0(v) \right\}.$$

与上述函数类似，$\lightning$ 的精确定义由分析所采用的特定抽象状态域给出。不过，为保证分析的可靠性，该定义除了满足上述不等式外，还必须满足以下基本性质：

1. 对 $\bot^\sharp$ 中的任意变量集进行任意赋值，结果为 $\bot^\sharp$:

$$\forall V \in \mathcal{P}(\mathcal{V}) : \bot^\sharp \lightning V = \bot^\sharp.$$

2. 对未受约束的其他变量进行任意赋值不会产生任何影响：

$$\forall V_1, V_2 \in \mathcal{P}(\mathcal{V}), d \in \mathcal{D} : d \lightning V_1 = d \implies d \lightning V_1 \cup V_2 = d \lightning V_2.$$

3. $\lightning$ 算子具有单调性，即：

$$\forall V_1, V_2 \in \mathcal{P}(\mathcal{V}), d_1, d_2 \in \mathcal{D} : V_1 \subseteq V_2 \wedge d_1 \sqsubseteq^\sharp d_2 \implies d_1 \lightning V_1 \sqsubseteq^\sharp d_2 \lightning V_2.$$

我们赋予 $\lightning$ 最高的优先级，高于所有状态格算子。

## 4 条件写入域

为描述线程间的干扰，我们引入干扰域的概念。干扰域由一个格 $\mathcal{I}$ 定义，其元素表示状态对的集合。它定义了该格上的两个函数，这两个函数都以兼容的状态域为参数，该状态域的格为 $\mathcal{D}$:

$$\mathit{stabilise} : \mathcal{I} \times \mathcal{D} \to \mathcal{D} \qquad \mathit{transitions} : \mathcal{D} \times \mathit{Assign} \to \mathcal{I}$$

其中，赋值是$inst$中对应于赋值语句的元素。如果一个状态域在 $\mathcal{D}$ 上定义了稳定化和转换定义中使用的所有函数，我们就称该状态域为兼容的。

在本文中，我们定义了一种称为条件写入域的干扰域。它以第3节所述的任意状态域为参数，并将 $\mathcal{I}$ 定义为一个格，其元素是从程序变量到状态格元素的映射。形式化地说：

$$\mathcal{I} = \mathcal{V} \to \mathcal{D}.$$

直观而言，该域中的一个元素 $i \in \mathcal{I}$ 会将每个变量映射到一个状态集合的过近似，在这些状态下该变量可能会被写入，我们称之为该变量的写入条件。例如，如果 $i$ 表示线程 $t$的一个有效保证条件，且形式为 $i = [v_1 \mapsto P_1,\ v_2 \mapsto P_2]$，那么 $t$ 只有在程序处于 $P_1$所涵盖的状态时才会更新 $v_1$ ，并且只有在程序处于

$v_2$ $P_2$所涵盖的状态时才会更新。如果程序处于 $P_1$ 和 $P_2$共同涵盖的状态， $t$ 可能会更改任一变量，也可能在一个步骤中同时更改两个变量。这可由具体化函数 $\gamma^\times$ 表示: $\mathcal{I} \to \mathcal{P}(\mathcal{S} \times \mathcal{S})$:

$$\gamma^\times(i) \;\widehat{=}\; \left\{(s_1, s_2) \in \mathcal{S} \times \mathcal{S} \mid \forall v \in \mathcal{V} : s_2(v) \neq s_1(v) \implies s_1 \in \gamma^\sharp(i(v))\right\}.$$

也就是说， $\gamma^\times(i)$ 中任何会改变变量 $v$ 的转换，其前状态都必须是一个满足 $i$ 允许更新 $v$ 的状态。

与我们的抽象状态域类似，对于 $i_1$， $i_2 \in \mathcal{I}$，我们用 $i_1 \sqsubseteq^\times i_2$ 表示 $i_1$ 在格上小于或等于 $i_2$ ，这意味着它表示相同或更少的转换。同样，我们也可以说 $i_1$ 比 $i_2$“更强”，或者说 $i_2$ 比 $i_1$“更弱”。格的最大元 $\top^\times$ 表示所有状态转换的集合，而底元 $\bot^\times$ 表示恒等关系:

$$\top^\times \;\widehat{=}\; \lambda v \in \mathcal{V} : \top^\sharp \qquad\qquad \bot^\times \;\widehat{=}\; \lambda v \in \mathcal{V} : \bot^\sharp.$$

并与交算子 $\sqcup^\times$ 和 $\sqcap^\times$ 按分量定义:

$$i_1 \sqcup^\times i_2 \;\widehat{=}\; \lambda v \in \mathcal{V} : i_1(v) \sqcup^\sharp i_2(v) \qquad i_1 \sqcap^\times i_2 \;\widehat{=}\; \lambda v \in \mathcal{V} : i_1(v) \sqcap^\sharp i_2(v).$$

### 4.1 小例子

本节通过一个小例子来展示我们分析的运作机制。下面的程序由线程 $T_0$ 和 $T_1$、全局变量 $x$ 和 $z$，以及一个属于 $T_0$ 的局部变量 $r$ 组成。我们的目标是证明，程序在前置条件 $true$ 下终止时，$r = 0$ 成立。

```
T0:                 T1:
1: r := 0           1: if z == 1:
2: if z == 0:       2:      x := 1
3:      x := 0
4:      r := x
```

快速检查一下程序，我们可以推断，只要在 $T_0$ 第 4 行赋值之前满足 $x = 0$ ，后置条件就成立。现在查看 $T_1$ ，我们发现 $x$ 确实可能被更新为 1，但关键在于，只有当 $z = 1$ 时才会如此。关于 $x$ 可被更新的条件约束，正是我们的分析为验证程序而要推断的性质。

我们的分析是线程模块化的。对于每个线程 $t$，我们都会针对每条指令处的程序状态推导一个断言，并将其称为证明概要。我们的目标是让这些断言分别过近似在任意线程交错下可在相应位置到达的状态。与许多其他线程模块化技术一样 $[13, 6, 3, 5, 8, 1]$,，我们通过为 $t$ 的环境中的每个线程生成一个保证条件 来实现这一点，该条件过近似了该线程可达状态转换的集合。在我们的分析中，这些保证条件表示为条件写入域中的元素。通过将它们并起来，

我们为 $t$ 得到一个可靠的依赖条件，它过近似了可能在 $t$ 的环境中执行的状态转换集合。为了确保 $t$ 的证明概要中的某个断言在所有线程交错下都成立，我们会弱化该断言，使其涵盖该依赖条件所允许的任意数量转换的后状态。现在，我们通过上述示例演示这一过程。

要初始化我们的分析，必须提供以下参数：

1. 一个带有格 $\mathcal{D}$的状态域。我们选择常量域，它是从变量到其常量值的部分映射。2. 变量集 $\mathcal{V}_0$ 和 $\mathcal{V}_1$，我们希望分别以它们为定义域来定义$T_0$ 和 $T_1$ 的依赖条件。我们选择 $\mathcal{V}_0 = \mathcal{V}_1 = \mathcal{V}$，因此实际上将所有程序变量都视为全局变量。3. 一个整数 $N$，用于指定稳定化函数的精度，其中 $0 \leq N \leq |\mathcal{V}|$。在我们的例子中，$|\mathcal{V}| = 3$，我们选择 $N = 3$ 以获得最高精度。我们将在第 4.2 节中说明这个数字的确切含义。

分析首先使用常量域推导 $T_0$ 的可达状态。在这一步中，我们不考虑环境干扰。$\mathcal{D}$ 的元素表示为从变量到值的映射；任何未包含的变量都不受约束。例如，$[r \mapsto 0]$ 表示这样一组状态：$r = 0$、$x$ 和 $z$ 可以取任意值。

```
T0:
    ⊤♯
1:  r := 0
    [r ↦ 0]
2:  if z == 0:
          [r ↦ 0, z ↦ 0]
3:        x := 0
          [r ↦ 0, z ↦ 0, x ↦ 0]
4:        r := x
          [r ↦ 0, z ↦ 0, x ↦ 0]
```

现在，我们推导一个保证条件，用来描述在 $T_0$ 中每个变量可在何种状态下被写入。通过检查证明概要中的断言，我们发现，对 $r$ 的赋值紧随断言 $\top^\sharp$ 之后，因此$r$ 可以在任意状态下由 $T_0$ 写入。这个事实由转换 函数表示：

$$\mathit{transitions}(\top^\sharp,\, r := 0) = \bot^\times[r \mapsto \top^\sharp]$$

其中，$f[x \mapsto y]$ 表示函数 $f$，并将 $x$ 映射到 $y$。例如，上述内容等价于：

$$[r \mapsto \top^\sharp,\, x \mapsto \bot^\sharp,\, z \mapsto \bot^\sharp].$$

同样，我们观察到，只有在满足 $r = 0 \wedge z = 0$ 时才能写入 $x$，而且从未写入过 $z$。将 转换 函数应用于每个前置条件 - 赋值对，并通过 $\sqcup^{\times}$ 合并结果，我们可以推导出保证条件 $G_0 \in \mathcal{I}$，它将每个变量映射到一个状态的过近似（作为 $\mathcal{D}$ 的元素），表示在这些状态下可以写入该变量：

$$G_0 = [r \mapsto \top^\sharp,\ x \mapsto [r \mapsto 0,\ z \mapsto 0],\ z \mapsto \bot^\sharp].$$

现在，我们将这个保证条件转换为 $T_1$ 的依赖条件。首先，将所有不在 $\mathcal{V}_1$ 中的变量映射到 $\top^\sharp$，并在每个写入条件中使用 $\lightning$ 对它们进行任意赋值，从而将它们从依赖条件中移除。由于 $\mathcal{V}_1 = \mathcal{V}$，这里无需进行任何操作。为了弱化依赖条件，使其能够涵盖任意数量的环境转移，下一步是推导其传递闭包。分析发现，由于在 $G_0$ 下允许以下两步转移，因此不满足传递性：

| $r$ | $x$ | $z$ |
|---|---|---|
| 1 | 0 | 0 |

$\rightarrow$

| $r$ | $x$ | $z$ |
|---|---|---|
| 0 | 0 | 0 |

$\rightarrow$

| $r$ | $x$ | $z$ |
|---|---|---|
| 0 | 1 | 0 |

第一次转移更新了 $r$，并且由于将 $r$ 映射到 $\top^\sharp$，因此该转移是允许的。第二次转移更新了 $x$，并且由于第二个状态满足 $x$ 的写入条件 $[r \mapsto 0,\ z \mapsto 0]$，因此该转移是允许的。然而，从第一个状态到第三个状态的转移不在我们的依赖条件涵盖范围内，因为它涉及对 $x$ 的更新，而第一个状态不满足 $x$ 的写入条件，因为 $r = 1$。为了解决这个问题，我们的分析使用 $close$ 函数弱化 $x$ 的写入条件：

$$R_1 = close(G_0) = [r \mapsto \top^\sharp,\ x \mapsto [z \mapsto 0],\ z \mapsto \bot^\sharp].$$

现在，我们推导 $T_1$ 的可达状态。不过，这次我们必须确保在 $R_1$ 下对每个生成的中间断言进行“稳定化”，以确保推导出的可达状态涵盖来自 $T_1$ 环境的所有可能干扰。由于 $R_1$ 是传递的，断言 $d_1 \in \mathcal{D}$ 可以通过推导出某个 $d_2 \in \mathcal{D}$ 来稳定化；该断言涵盖从 $d_1$ 中任意前状态出发、由 $R_1$ 所描述的一步内可达的所有状态，然后取 $d_1 \sqcup^\sharp d_2$。这可以通过应用 稳定化 函数实现，其中 稳定化$(R_1, d_1)$ 是一个大于 $d_1$ 的 $\mathcal{D}$ 元素，涵盖由 $R_1$ 所描述的所有一步转换。我们的分析会按照初始化时提供的参数 $N$ 所指定的精度执行此操作（见第 4.2 节）。我们推导出以下可达状态；在这个小例子中，它们在生成时就已经稳定：

```
T1:
    ⊤♯
1:  if z == 1:
        [z ↦ 1]
2:      x := 1
        [z ↦ 1, x ↦ 1]
```

现在，我们为 $T_1$ 生成一个保证 $G_1$：

$$G_1 = [r \mapsto \bot^\sharp,\ x \mapsto [z \mapsto 1],\ z \mapsto \bot^\sharp].$$

注意，这个保证捕获了我们期望的约束：只有当 $z = 1$ 时才会更新 $x$。这个保证本身就是传递的，而且由于 $\mathcal{V}_0 = \mathcal{V}$，我们可以立即推导出 $R_0$ 为 $G_1$。现在，我们在这个新的依赖条件下，从头重新构建 $T_0$ 的证明概要。在这个简单的示例中，稳定化后的断言与上一轮迭代完全相同，因此我们已经到达一个不动点，可以返回当前的依赖-保证条件和证明大纲。$T_0$ 和 $T_1$ 的派生后置条件的合取足以证明程序终止时 $r = 0$ 成立。

接下来的小节将介绍 稳定化、转换 和 闭合 函数，我们分别用它们在依赖条件下稳定化断言、从证明概要推导保证条件，以及为干扰域元素推导传递闭包。

### 4.2 稳定化 函数

本节将介绍干扰域函数：

$$stabilise : \mathcal{I} \times \mathcal{D} \to \mathcal{D}$$

然后为我们的条件写入域定义它。此函数的目的是捕获从某个 $d \in \mathcal{D}$ 所涵盖的任意前状态出发，经由某个 $i \in \mathcal{I}$（表示依赖）在一次转移内可达的所有后状态集合：

$$\gamma^\sharp(stabilise(i, d)) \supseteq \gamma^\sharp(d) \cup \{s_2 \in \mathcal{S} \mid \exists s_1 \in \gamma^\sharp(d) : (s_1, s_2) \in \gamma^\times(i)\}. \quad (1)$$

当 $i$ 是传递的，这足以确保 稳定化$(i, d)$ 能捕获从 $d$ 中的初始状态出发，在 $i$ 中经过任意次数转移后可达的所有状态，从而完整地考虑由 $i$ 表示的环境所造成的干扰。稳定化$(i, d)$ 还会捕获 $d$，这一事实考虑到了环境可能不采取任何步骤，因此也适用于非自反干扰域；这类域不会像条件写入域那样隐式捕获 $\{(s_1, s_2) \in \mathcal{S} \times \mathcal{S} \mid s_1 = s_2\}$。

然而，请注意，(1) 是一种过近似，因此可能存在转移 $(s_1, s_2) \in \gamma^\times(i)$，其中 $s_1 \notin \gamma^\sharp(d) \wedge s_1 \in \gamma^\sharp$(稳定化$(i, d)$)，但$s_2 \notin \gamma^\sharp$(稳定化$(i, d)$)。因此，尽管 稳定化$(i, d)$ 能可靠地捕获 $i$ 对 $d$ 造成的干扰影响，但它可能无法对 $i$ 中的转移闭合。这一局限对我们分析的实际影响是：重复应用 稳定化 可能会产生越来越弱的抽象状态，因此对于较长的线程，我们可以预期分析会更加保守，因为在生成证明概要的过程中会更频繁地调用稳定化。

大多数现有的线程模块化分析通过反复应用依赖条件，直至达到不动点，来补足这一缺失的闭包。在我们的框架中，这等价于将稳定化$(i, d)$定义为某个函数上的不动点

该函数满足 (1)。也就是说，我们迭代地弱化 $d$ ，直到 $i$ 中不再存在前状态属于 $d$ 、后状态却不属于 $d$的转换。除了实现这一闭包外，这种方法还有一项性能优势：为了准确捕获干扰， $i$ 无需具有传递性，因此可以避免为依赖条件求传递闭包。由此引出一个有趣的问题，而我们尚未见过文献讨论这个问题：为每个生成的中间断言求不动点，是否比为每个生成的依赖条件求传递闭包更耗时？我们通过实现分析的两种模式来研究这一问题：一种模式使依赖条件具有传递性，并将稳定化 简单定义为满足 (1) 的函数；另一种模式不使依赖条件具有传递性，而将稳定化 定义为第一种模式中定义的函数上的不动点。从现在起，我们用记号 $f^*$ 表示函数 $f$ 在第二种（非传递）模式下所采用的版本。

现在，我们来定义 稳定化$(i, d)$，使其满足 (1)。一种简单的定义方法是对 $d$ 中所有写入条件与 $d$相交的变量进行任意赋值：

$$d \not\downarrow \{v \in \mathcal{V} \mid d \sqcap^\sharp i(v) \neq \bot^\sharp\}.$$

直观地说，如果 $d$ 与 $i(v)$ 不相交，那么当程序处于 $d$所描述的任意状态时，环境都无法更新 $v$。但是，如果 $d$ 与$i(v)$ 相交，那么环境在处于 $d$ 所描述的状态时可能会更新 $v$ ；通过将 $d$调整为 $d \not\downarrow \{v\}$，我们就能捕获从 $d$所描述的任意状态出发，对 $v$ 进行任意更新所产生的所有可能后状态。

这个定义是可靠的（因为它满足 (1)），但也非常保守，因为我们没有考虑到这样一个事实：当 $v$ 在 $d$所描述的状态中被更新时，我们还必须处于 $i(v)$ 所描述的状态中。为了提高精度，我们可以不在 $d$中对变量 $v$进行任意赋值，而是在 $d \sqcap^\sharp i(v)$ 中对它们进行任意赋值，如下所示。

$$d \sqcup^\sharp \bigsqcup_{v \in \mathcal{V}}^\sharp (d \sqcap^\sharp i(v)) \not\downarrow \{v\}.$$

然而，这个定义并不可靠！我们未能考虑那些会同时更新多个变量的转移 $i$ 。为了保证可靠性，我们必须捕捉更新变量任意组合 $V \in \mathcal{P}(\mathcal{V})$ 的转移所产生的影响。为此，我们首先捕捉这样一组状态：在这些状态下，所有变量 $v \in V$ 都能在一次转移中更新，且转移发生时处于 $d$ 所捕捉的状态；这组状态是 $d$ 与所有这些变量的写入条件 $d \sqcap^\sharp \bigsqcap^\sharp_{v \in V} i(v)$ 的交。然后，我们通过对 $V$ 进行任意赋值来模拟变量更新的效果。因此，我们的新定义如下：

$$d \sqcup^\sharp \bigsqcup_{V \in \mathcal{P}(\mathcal{V})}^\sharp \left( d \sqcap^\sharp \bigsqcap_{v \in V}^\sharp i(v) \right) \not\downarrow V. \tag{2}$$

然而，每次调用稳定化 时都遍历 $\mathcal{V}$ 的幂集，效率低得无法接受，因此我们现在必须决定，是让分析更保守、更快速，还是更精确、更缓慢。我们不采用固定定义，而是接受一个

参数 $N$，让用户决定适当的权衡。对于基数为 $|V| \leq N$ 的所有变量集 $V$，我们会像式（2）那样，精确考虑更新 $V$ 的转移所产生的影响。对于其他规模更大的变量集，我们采用一种特殊技术来过近似其影响；该技术只要求遍历基数为 $N+1$ 的变量集。

为了将我们的定义形式化，我们先推导出状态集合 $X$：通过执行 $i$ 中更新不超过 $N$ 个变量的任意转移，可以到达这些状态。我们定义中的这一“精确”部分由 (2) 推导得出：

$$X = \bigsqcup_{V \in \mathcal{P}(\mathcal{V}),\, |V| \leq N}^{\sharp} \left( d \sqcap^{\sharp} \prod_{v \in V}^{\sharp} i(v) \right) \lightning V.$$

相应的“保守”部分则计算一个粗略的过近似，该过近似表示从 $d$ 经由 $i$ 中更新超过 $N$ 个变量的转换可达的状态集合。通过仅考虑基数为 $N+1$ 的变量集，而非所有基数为 $> N$ 的集合，可以使用以下技术高效地推导此过近似。我们首先收集集合 $\mathcal{V}_N$，其中包含所有满足以下条件的集合 $\{v_1, v_2, ..., v_{N+1}\}$：$v_1$、$v_2$、.....、$v_{N+1}$ 的写入条件之交与 $d$ 相交。这些是基数为 $N+1$ 的变量集，其中所有变量都可能在从由 $d$ 捕获的状态出发的一步 $i$ 内更新：

$$\mathcal{V}_N = \left\{ V \in \mathcal{P}(\mathcal{V}) \;\middle|\; |V| = N+1 \wedge \left( d \sqcap^{\sharp} \prod_{v \in V}^{\sharp} i(v) \right) \neq \bot^{\sharp} \right\}.$$

现在，我们收集状态集的一个过近似，其中 $d$ 中任意这些变量组合都可能被 $\mathcal{V}_N$ 写入：

$$\bigsqcup_{V \in \mathcal{V}_N}^{\sharp} d \sqcap^{\sharp} \prod_{v \in V}^{\sharp} i(v).$$

最后，我们对上述内容中出现在 $\mathcal{V}_N$ 里的所有变量进行任意赋值：

$$Y = \left( \bigsqcup_{V \in \mathcal{V}_N}^{\sharp} d \sqcap^{\sharp} \prod_{v \in V}^{\sharp} i(v) \right) \lightning \bigcup_{V \in \mathcal{V}_N} V.$$

在附录 A.1 中，我们证明 $Y$ 不仅过近似了会更新 $\mathcal{V}_N$ 中这些变量集的转移所对应的后状态，也过近似了 $\mathcal{V}$ 中基数为 $> N$ 的每个子集所对应的后状态。

我们将 稳定化$_N(i, d)$ 定义为 $d$ 与以下两者的并：精确的过近似 $X$（表示更新不超过 $N$ 个变量的转换的后状态），以及不那么精确的过近似 $Y$（表示更新超过 $N$ 个变量的转换的后状态）：

$$stabilise_N(i, d) \;\widehat{=}\; d \sqcup^{\sharp} X \sqcup^{\sharp} Y.$$

我们在附录 A.1 中证明，我们的定义满足 (1)。在分析的非传递模式下，我们将稳定化函数定义为上述表达式的最小不动点：

$$stabilise^{*}_{N}(i,d) \;\widehat{=}\; \mu d : stabilise_{N}(i,d).$$

如上所述，计算 稳定化$_N$(*i*, *d*) 在$N$ 较大时可能较慢，因为需要遍历 $\mathcal{P}(\mathcal{V})$ 的很大一部分。在实践中，我们可以实现一种简单的优化来缩小该域。对于任意变量集$V_0 \in \mathcal{P}(\mathcal{V})$，如果 $V_0$ 中变量的写条件的交为 $\bot^\sharp$，则可以忽略 $V_0$的任何超集。这是因为，由于 $V$ 是 $V_0$的超集， $V$ 中变量的写条件的交同样为 $\bot^\sharp$。该优化在附录 B.1 中形式化。

### 4.3 转换 函数

除了 稳定化 函数之外，我们的分析还需要一种从证明概要生成保证条件的方法。具体而言，我们需要该函数的定义：

$$transitions : \mathcal{D} \times Assign \to \mathcal{I}$$

直观上，转换(*d*, *A*) 刻画了在 $d$所捕获的任意状态下执行（局部或全局）赋值 $A$ 时可能发生的所有状态转换。形式上：

$$\gamma^{\times}(transitions(d,A)) \supseteq \left\{(s_1,s_2) \in \mathcal{S} \times \mathcal{S} \mid s_1 \in \gamma^{\sharp}(d) \wedge s_2 = exec(A,s_1)\right\}.$$

在我们的方法中，我们只需将每个被赋值变量映射到 $d$，并将其余变量映射到 $\bot^\sharp$：

$$transitions(d, \langle x_1,...,x_n := e_1,...,e_n\rangle) \;\widehat{=}\; \bot^{\times}[v \in \{x_1,...,x_n\} \mapsto d].$$

其中， $f[x \in X \mapsto y]$ 表示函数 $f$，其中每个 $x \in X$ 都映射到 $y$。

对其证明大纲中的每个前置条件-赋值对（$d$, $A$），将转换($d$, $A$)并起来，即可推导出整个线程的保证。我们在附录 A.2 中证明了该定义的可靠性。

### 4.4 闭合 函数

我们的分析要求依赖条件具有传递性。在附录 A.3 中，我们证明，当且仅当 $i \in \mathcal{I}$ 捕获了从状态 $s_1 \notin \gamma^\sharp$(*i*(*v*)) 到状态 $s_2 \in \gamma^\sharp$(*i*(*v*)) 的某个转移（$s_1$, $s_2$）时，这一性质才会被违反，其中 $v \in \mathcal{V}$ 为某个值。在这种情况下，$i$ 包含一个会修改 $v$ 的转移（$s_2$, $s_3$），但不可能包含（$s_1$, $s_3$），因为$s_1$ 不在 $v$ 的写入条件中。因此， $i$ 具有传递性，当且仅当对所有$v \in \mathcal{V}$：

$$\forall (s_1,s_2) \in \gamma^{\times}(i) : s_1 \notin i(v) \implies s_2 \notin i(v). \tag{3}$$

因此，我们定义一个函数：

$$close : \mathcal{I} \to \mathcal{I}$$

它对所有 $v \in \mathcal{V}$ 均满足式 (3)。

为了保留我们的依赖能够捕获其他线程保证条件中所有转换这一性质，我们还要求 $i \sqsubseteq^{\times}$ 闭合($i$)，即在推导$close(i)$时，我们只能弱化 （也就是说，向其中添加状态） $i$中变量所映射到的写条件。

现在，设想 $i$ 不是传递的，因此它捕获了一个转移 $(s_1, s_2)$，该转移从状态 $s_1 \notin \gamma^\sharp(i(v))$ 到状态 $s_2 \in \gamma^\sharp(i(v))$，其中 $v \in \mathcal{V}$ 为某个值。通过弱化写入条件来修正此类反例的唯一方法，就是将 $s_1$ 添加到 $\gamma^\sharp(i(v))$。因此，要为$i$ 推导传递闭包，取决于这样一种方法：对于某个 $v \in \mathcal{V}$，推导出所有状态 $s_1 \notin \gamma^\sharp(i(v))$ 的过近似，其中存在某个 $s_2 \in \gamma^\sharp(i(v))$，使得转移 $(s_1, s_2) \in i$ 成立。

一种非常粗略的方法可能是，简单地收集出现在 $i(v)$ 中的变量的写条件所捕获的全部状态。然而，由于我们在每次依赖条件的推导中只调用一次传递闭包过程，并且（尤其是由于 $\mathcal{D}$ 和 $\mathcal{I}$ 所提供的抽象）我们预计在分析过程中不会进行很多次此类推导，因此研究更精确的方法或许是有益的。

我们方法的直觉如下：假设存在如上所述的违反转移 $(s_1, s_2)$，它仅修改变量 $v'$。注意，$v'$ 必须是与 $v$ 不同的变量，因为 $s_1 \notin \gamma^\sharp(i(v))$，从而 $v$ 不能在状态 $s_1$ 中被修改。由于我们的转移仅修改 $v'$，我们必须有 $s_1$ 属于在 $s_2$ 中对 $v'$ 进行任意赋值所得到的状态集合：

$$s_1 \in \{s \in \mathcal{S} \mid \forall u \in \mathcal{V} \setminus \{v'\} : s(u) = s_2(u)\}$$

由于 $s_2 \in \gamma^\sharp(i(v))$，我们可以用 $i(v) \not\downarrow v'$ 得到该集合的一个过近似。此外，显然 $s_1$ 被 $i(v')$ 所捕获，因为 $i$ 捕获了 $(s_1, s_2)$，其中 $s_1(v') \neq s_2(v')$。综合这两个事实，我们有：

$$s_1 \in \gamma^\sharp(i(v) \not\downarrow \{v'\}) \cap \gamma^\sharp(i(v')).$$

在我们的分析中，可以将此集合的过近似推导为一个 $\mathcal{D}$ 元素：

$$i(v) \not\downarrow \{v'\} \sqcap^\sharp i(v').$$

直观地说，这表示 $i(v')$ 中的一个状态子集：当更新 $v'$ 时，这些状态可能会导致 $i(v)$ 中的后置状态。为了考虑其他任何变量的更新，我们对每个 $v' \in \mathcal{V} \setminus \{v\}$取上述结果的并，以此弱化 $i(v)$：

$$i(v) \sqcup^\sharp \bigsqcup_{v' \in \mathcal{V} \setminus \{v\}}^\sharp i(v) \not\downarrow \{v'\} \sqcap^\sharp i(v').$$

由于 $i(v) \not\downarrow \{v\} \sqcap^\sharp i(v) = i(v)$，我们可以通过将 $v$ 添加到并的域中来简化此定义：

$$\bigsqcup_{v' \in \mathcal{V}}^\sharp i(v) \not\downarrow \{v'\} \sqcap^\sharp i(v').$$

现在回想一下，我们上面的推理依赖于 $v'$ 是唯一被更新的变量，而我们的条件写入域能够捕获修改多个变量的转移。要真正过近似违反条件的前状态，我们实际上必须遍历 $\mathcal{V}$的整个幂集：

$$\bigsqcup_{V \in \mathcal{P}(\mathcal{V})}^{\sharp} i(v) \not\downarrow V \sqcap^{\sharp} \prod_{v' \in V}^{\sharp} i(v'). \tag{4}$$

不过，我们在附录 B.2 中讨论了能够显著缩小该搜索空间的优化。

现在，我们有了一种弱化 $i(v)$ 的方法，以捕获所有后状态属于 $i(v)$ 的转换在 $i$ 中对应的前状态。然而，添加到 $i(v)$ 中的新状态本身可能是 $i$ 所捕获的其他转换的后状态，因此，我们必须对 (4) 求不动点，才能针对 $v$ 得到 (3)。此外，通过弱化 $v$ 的写入条件而添加到 $i$ 中的新转换可能会使其他变量的 (3) 不再成立。因此，要用此方法计算传递闭包，我们必须针对所有 变量对 (4) 求不动点：

$$close(i) \;\widehat{=}\; \mu i : \lambda v : \bigsqcup_{V \in \mathcal{P}(\mathcal{V})}^{\sharp} i(v) \not\downarrow V \sqcap^{\sharp} \prod_{v' \in V}^{\sharp} i(v').$$

令 $i' =$闭合$(i)$。不动点计算终止后，我们有：

$$\forall v \in \mathcal{V} : i'(v) \sqsupseteq^{\sharp} \bigsqcup_{V \in \mathcal{P}(\mathcal{V})}^{\sharp} i'(v) \not\downarrow V \sqcap^{\sharp} \prod_{v' \in V}^{\sharp} i'(v')$$

我们在附录 A.3 中证明，这足以保证传递性。

## 5 依赖-保证生成

附录 C 对条件写入域中定义的数据结构和函数进行了完整总结。本节旨在形式化第 4.1 节所述的过程，从而展示如何将我们的域集成到完整的程序分析中，风格与 [12] 类似。

我们的分析涉及为每个线程迭代构建依赖条件 $R : \mathcal{T} \to \mathcal{I}$ 和保证条件 $G : \mathcal{T} \to \mathcal{I}$。我们的依赖条件直接根据生成的保证条件计算得出，使用 依赖 函数：

$$rely : \mathcal{T} \times (\mathcal{T} \to \mathcal{I}) \to \mathcal{I}$$

在我们分析的传递模式中，它被定义为所有环境保证之并的传递闭包：

$$rely(t, G) \;\widehat{=}\; close \circ reduce_t \left( \bigsqcup_{t' \in \mathcal{T} \setminus \{t\}}^{\times} G(t') \right)$$

其中归约$_t(i)$将 $i$ 弱化，以消除所有不在 $\mathcal{V}_t$中的变量：

$$reduce_t(i) \;\widehat{=}\; i\left[v \in \mathcal{V} \setminus \mathcal{V}_t \mapsto \top^\sharp\right]\left[v \in \mathcal{V}_t \mapsto i(v) \not\downarrow \mathcal{V} \setminus \mathcal{V}_t\right].$$

在第 4.2 节所述的非传递模式下运行时，我们只需跳过传递闭包步骤：

$$rely^*(t, G) \;\widehat{=}\; reduce_t\left(\bigsqcup_{t' \in \mathcal{T} \setminus \{t\}}^{\times} G(t')\right).$$

要为线程 $t \in \mathcal{T}$ 推导保证条件，我们使用具有格 $\mathcal{D}$ 的状态域，为 $t$ 生成线程局部证明概要，并削弱中间断言，以考虑 $t$ 的依赖所指定的干扰。随着可达状态的发现，我们使用 转换 函数推导它们所诱导的状态转换，并将这些转换收集到针对 $t$ 的新保证中。这一分析通过图 1 中定义的语义 $\mathbb{S}$ 形式化，其中 $\mathbb{S}(c,\ (d,\ r,\ g))$ 是一个元组，其第三个元素是相对于前置条件 $d$ 和依赖 $r$ 而言，程序 $c$ 的一个可靠保证条件。为提供简洁的形式化表示，我们在类型为 $\mathcal{D} \times \mathcal{I} \times \mathcal{I}$ 的元组上定义并运算符 $\sqcup$：

$$\sqcup : (\mathcal{D} \times \mathcal{I} \times \mathcal{I}) \times (\mathcal{D} \times \mathcal{I} \times \mathcal{I}) \to (\mathcal{D} \times \mathcal{I} \times \mathcal{I})$$
$$(d_1,\ r_1,\ g_1) \sqcup (d_2,\ r_2,\ g_2) \;\widehat{=}\; \left(d_1 \sqcup^\sharp d_2,\ r_1 \sqcup^\times r_2,\ g_1 \sqcup^\times g_2\right).$$

$$\mathbb{S} : inst \times (\mathcal{D} \times \mathcal{I} \times \mathcal{I}) \to \mathcal{D} \times \mathcal{I} \times \mathcal{I}$$
$$\mathbb{S}(c_1; c_2,\ X) \;\widehat{=}\; \mathbb{S}(c_2,\ \mathbb{S}(c_1,\ X))$$
$$\mathbb{S}(c : Assign,\ (d,\ r,\ g)) \;\widehat{=}\; \text{let } d' = stabilise_N(r,\ d) \text{ in } (\mathbb{S}^\sharp(c,\ d'),\ r,\ g \sqcup^\times transitions(d',\ c))$$
$$\mathbb{S}(\text{if } b \text{ then } c_1 \text{ else } c_2,\ X) \;\widehat{=}\; \text{let } X_1 = \mathbb{S}(c_1,\ \mathbb{B}(b,\ X)) \text{ in } \text{let } X_2 = \mathbb{S}(c_2,\ \mathbb{B}(\neg b,\ X)) \text{ in } X_1 \sqcup X_2$$
$$\mathbb{S}(\text{while } b \text{ do } c,\ X) \;\widehat{=}\; \mathbb{B}(\neg b,\ \mu X\ .\ X \sqcup \mathbb{S}(c,\ \mathbb{B}(b,\ X))$$
$$\mathbb{B}(b,\ (d,\ r,\ g)) \;\widehat{=}\; (\mathbb{B}^\sharp(b,\ stabilise_N(r,\ d)),\ r,\ g)$$

**图 1**。 用于生成保证条件的集合语义。

我们通过函数 $Analyse$ 定义整个分析，该函数接受程序前置条件，并返回该程序的一组依赖和保证条件：

$$Analyse : \mathcal{D} \to (\mathcal{T} \mapsto \mathcal{I}) \times (\mathcal{T} \mapsto \mathcal{I}).$$

使用条件写入域生成依赖-保证条件 15

其定义如下:

$$\mathit{Analyse}(\mathit{pre}) \;\widehat{=}\; \mu R, G :$$
$$(\lambda t \in \mathcal{T}\text{:依赖}(t,\, G) \quad \lambda t \in \mathcal{T}\text{: } \mathbb{S}(t, \text{(前, 依赖}(t,\, G) \quad \bot^{\times}))\text{。3)}$$

其中， $X.3$ 指元组 $X$的第三个元素。在分析的非传递模式下，使用函数依赖$^*$和稳定化$^*_N$ 分别替代依赖 和稳定化$_N$ 。

## 6 评估

我们使用一个简单的 Python 原型[2],实现了分析的两种模式，并使用两个状态域分别评估了每种模式：常量域，以及常量域的幂集完备 [12]。对于所有测试，我们选择$N = |\mathcal{V}|$ 和 $\mathcal{V}_t = \mathcal{V}$，使每个线程达到最高精度。我们在五个可使用条件写入域验证的并发程序上评估了分析，并将结果记录在表 1 中。所有实验均在一台配备 2.3 GHz Intel Core i5 处理器的 2019 年款 MacBook Pro 上进行。

| 程序 | 常量域 | | | | | | 常量域的幂集完备 | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | 非传递 | | | 传递 | | | 非传递 | | | 传递 | | |
| | 验证 | 操作 | 时间（秒） | 验证 | 操作 | 时间（秒） | 验证 | 操作 | 时间（秒） | 验证 | 操作 | 时间（秒） |
| reset | × | 123 | 0.0004 | × | 168 | 0.0005 | ✓ | 533 | 0.0020 | × | 786 | 0.0067 |
| 循环 | ✓ | 620 | 0.0014 | ✓ | 836 | 0.0022 | ✓ | 1019 | 0.0053 | ✓ | 1235 | 0.0064 |
| 互斥锁1 | ✓ | 67 | 0.0004 | ✓ | 105 | 0.0006 | ✓ | 237 | 0.0013 | ✓ | 342 | 0.0021 |
| 互斥锁2 | × | 48 | 0.0006 | × | 51 | 0.0003 | ✓ | 318 | 0.0018 | ✓ | 339 | 0.0016 |
| 自旋锁 | × | 336 | 0.0012 | × | 291 | 0.0009 | ✓ | 1291 | 0.0115 | ✓ | 2355 | 0.0116 |

**表 1.** 不同状态域和分析模式下的验证结果。

程序自旋锁 可在 [16] 中找到。所有其他程序均由作者编写，可与原型实现一同获取。我们在$Ver.$下记录程序是否通过我们的框架验证，在$Ops$下记录$\sqcup^\sharp$和 $\sqcap^\sharp$ 操作的数量，并在$Time$下记录性能耗时。

我们的初步结果表明，传递模式几乎总是比非传递模式慢，并且需要更多状态格运算。此外，传递模式的精度较低：它无法使用幂集完备化域验证$reset$，而非传递模式则可以。这在一定程度上证明了广泛采用不动点方法进行稳定化的合理性。不过，我们注意到，我们的大多数基准程序都是小型算法；这些结果未必适用于线程体更长的程序，因为在这类程序中$stabilise$ 调用与$rely$ 调用的比例更高。

[2] 可访问：https://doi.org/10.5281/zenodo.18589190。

## 7 相关工作

我们的工作受到 Miné [10, 9, 11] 以及 Monat 和 Miné [13] 所提出分析的启发，这些分析利用抽象解释生成依赖-保证条件。其中，[13] 中提出的框架与我们的工作最为相似：它基于一个用于表示状态的抽象域、另一个用于表示干扰的抽象域，以及他们各自定义的$stabilise$和$transitions$函数，用于在这两个域之间进行转换。然而，他们的分析要求使用标准数值抽象域，以带撇变量和不带撇变量来表示干扰，其中带撇变量表示干扰后状态中的变量值。这样既可以使用现有抽象域表示干扰，也可以采用部分定义的$stabilise$和$transitions$函数，用户只需为所选抽象域定义添加和移除某些变量（例如带撇变量）的函数即可。不过，我们发现，条件写入域无法简洁地表示为基于带撇变量和不带撇变量的数值域。原因之一是，该域对带撇变量和不带撇变量施加了不同的结构约束，例如，带撇变量只能被约束为等于其对应的不带撇变量（当该变量的写入条件为假时），否则则不受约束。此外，我们的框架允许用户完整定义$stabilise$ 和 $transitions$ 函数，从而支持更复杂的定义，例如我们的$stabilise$ 函数。最后，前述工作中的所有分析都通过对依赖条件求不动点来进行稳定化，而我们还提供了另一种分析模式，使依赖条件具有传递性，因此只需对依赖进行一步过度近似，就能处理干扰。

我们的分析以依赖-保证证明方法为基础 [7, 17], ，该方法已知比并发分离逻辑等其他逻辑更适合某些类别的程序 [14, 4]。许多生成依赖-保证条件的技术并不使用抽象解释，因此无法享有抽象域所提供的效率-精度权衡 [6, 3, 5,8]。据作者所知，现有采用抽象解释的技术仅限于对干扰的特定抽象，例如将变量映射到常量后状态值集合的非关系域 [1, 10], ，或基于带撇变量和不带撇变量的数值域 [13]。相比之下，我们将干扰抽象为**干扰域**，其可以表示依赖-保证条件的任意结构，并允许将这些干扰域与任何**状态域** 配对，只要该状态域实现了**稳定化**和**转换**定义中所使用的函数。

## 8 结论

在本工作中，我们引入了**干扰域**的概念，用于抽象线程或其环境可能执行的状态转换集合。干扰域由一个兼容的状态

域参数化，并定义为状态对集合上的一个格，用以表示转换；此外还包含稳定化和转换函数，分别用于应用和推导这些转换。随后，我们介绍了一个简单的干扰域示例，称为条件写入域，用于捕获仅约束变量在何种条件下更新、而不限制变量更新为何值的依赖-保证条件。最后，我们描述了一个简单的分析框架，展示如何利用干扰域为并发程序生成可靠的依赖-保证条件。

该框架的一个重大局限是目标语言过于简单。未来，我们希望扩展框架，以支持常见的语言特性，例如过程调用、动态创建线程和互斥锁。我们还希望构建一套更适用于复杂程序的干扰域，以处理数组和指针等复杂数据结构。为此，我们打算采用示例驱动的方法，通过检查实际使用的并发程序来识别有用的干扰域。最后，我们计划探索现有的静态分析工具，以便在其中实现我们的框架，从而提升性能，并更好地与现有分析方法进行比较。

**致谢**。 我们感谢 Kirsten Winter 对本文提出的深刻意见，并与我们进行了讨论。

**利益披露**。 作者声明，不存在与本文内容相关的竞争性利益。

## 参考文献

1. Carre, J., Hymans, C.：从单线程到多线程：一种高效的静态分析算法。CoRR **abs/0910.5833** (2009) 2. Cousot, P., Cousot, R.：抽象解释：通过构造或近似不动点，为程序静态分析建立统一的格模型。载于：POPL，第238–252页。ACM (1977) 3. Ezudheen, P., Neider, D., D'Souza, D., Garg, P., Madhusudan, P.：用于综合不变量与合约的 Horn-ice 学习。Proc. ACM Program. Lang. **2**(OOPSLA), 131:1–131:25 (2018) 4. Feng, X., Ferreira, R., Shao, Z.：并发分离逻辑与假设-保证推理之间的关系。载于：ESOP。《计算机科学讲义》，第4421卷，第173–188页。施普林格 (2007) 5. Flanagan, C., Qadeer, S.：线程模块化模型检查。载于：SPIN。《计算机科学讲义》，第2648卷，第213–224页。施普林格 (2003) 6. Gupta, A., Popeea, C., Rybalchenko, A.：Threader：一种面向多线程程序的基于约束的验证器。载于：CAV。《计算机科学讲义》，第6806卷，第412–417页。施普林格 (2011) 7. Jones, C.B.：迈向干扰程序开发方法的初步探索。ACM Trans. Program. Lang. Syst. **5**(4), 596–619 (1983)

8. Le, X., Sanán, D., Sun, J., Lin, S.：通过推断依赖-保证规范自动验证多线程程序。载于：ICECCS，第43–52页。IEEE (2020) 9. Miné, A.：《通过抽象解释对顺序程序和多线程程序进行静态分析》。载于：Reynier, P.A. (编)，《并行进程建模与验证第十届研习班》。法国马赛 (2012年12月)，https://inria.hal.science/hal- 00763076 10. Miné, A.：嵌入式实时并行 C 程序中运行时错误的静态分析。Log. Methods Comput. Sci.**8**(1) (2012) 11. Miné, A.：通过抽象解释进行关系式线程模块化静态值分析。载于：VMCAI。《计算机科学讲义》，第8318卷，第39–58页。施普林格 (2014) 12. Miné, A.：通过抽象解释静态推断数值不变量教程。Found. Trends Program. Lang. **4**(3-4), 120–372 (2017) 13. Monat, R., Miné, A.：利用关系式干扰抽象，对并发程序进行精确的线程模块化抽象解释。载于：VMCAI。《计算机科学讲义》，第10145卷，第386–404页。施普林格 (2017) 14. O'Hearn, P.W.：资源、并发与局部推理。Theor. Comput. Sci.**375**(1-3), 271–307 (2007) 15. Stølen, K.：一种开发完全正确的共享状态并行程序的方法。载于：Baeten, J.C.M., Groote, J.F. (编)，《CONCUR '91，并发理论第二届国际会议》。《计算机科学讲义》，第527卷，第510–525页。施普林格 (1991) 16. Winter, K., Coughlin, N., Smith, G.：面向并发程序的反向信息流分析。载于：CSF，第1–16页。IEEE (2021) 17. Xu, Q., de Roever, W.P., He, J.：用于验证共享变量并发程序的依赖-保证方法。Formal Aspects Comput. **9**(2), 149–174 (1997)

# A 证明

## A.1 稳定化的可靠性

**引理 1.** 对于任意 $i \in \mathcal{I}$ 和 $d \in \mathcal{D}$，我们有：

$$\gamma^\sharp(stabilise_N(i,d)) \supseteq \gamma^\sharp(d) \cup \{s_2 \in \mathcal{S} \mid \exists s_1 \in \gamma^\sharp(d) : (s_1, s_2) \in \gamma^\times(i)\}.$$

证明。为便于阅读，我们先定义变量集 $V$ 中某个 $i \in \mathcal{I}$ 的所有写入条件的交为 $M_i(V)$：

$$M_i : \mathcal{P}(\mathcal{V}) \to \mathcal{D}$$

$$M_i(V) \mathrel{\widehat{=}} \prod_{v \in V}^\sharp i(v).$$

稳定化 $_N$ 的定义因此为：

$$\begin{aligned}
&stabilise_N(i,d) \mathrel{\widehat{=}} \\
&\quad \text{let } X = \bigsqcup_{V \in \mathcal{P}(\mathcal{V}),\, |V| \le N}^\sharp \left(d \sqcap^\sharp M_i(V)\right) \lightning V \text{ in} \\
&\quad \text{let } \mathcal{V}_N = \left\{ V \in \mathcal{P}(\mathcal{V}) \;\middle|\; |V| = N+1 \wedge d \sqcap^\sharp M_i(V) \neq \bot^\sharp \right\} \text{ in} \\
&\quad \text{let } Y = \left( \bigsqcup_{V \in \mathcal{V}_N}^\sharp d \sqcap^\sharp M_i(V) \right) \lightning \bigcup_{V \in \mathcal{V}_N} V \text{ in} \\
&\quad d \sqcup^\sharp X \sqcup^\sharp Y.
\end{aligned}$$

该定义由一个“精确”组件 $X$ 和一个“保守”组件 $Y$ 组合而成。前者用于捕获修改不超过 $N$ 个变量的转移，后者用于捕获修改超过 $N$ 个变量的转移。首先，我们证明 $Y$ 的保守性。形式化地说：

$$\bigsqcup_{V \in \mathcal{P}(\mathcal{V}),\, |V| > N}^\sharp \left(d \sqcap^\sharp M_i(V)\right) \lightning V \sqsubseteq^\sharp Y.$$

由于对于任意 $d_1,\ d_2,\ d_3 \in \mathcal{D}$，我们都有 $d_1 \sqsubseteq^\sharp d_3 \wedge d_2 \sqsubseteq^\sharp d_3 \implies d_1 \sqcup^\sharp d_2 \sqsubseteq^\sharp d_3$，因此只需证明，对于任意满足 $|V'| > N$ 的 $V' \in \mathcal{P}(\mathcal{V})$：

$$\left(d \sqcap^\sharp M_i(V')\right) \lightning V' \sqsubseteq^\sharp Y.$$

回顾我们对 $\lightning$ 的约束：

$$\forall V \in \mathcal{P}(\mathcal{V}) : \bot^\sharp \lightning V = \bot^\sharp.$$

因此，如果 $d \sqcap^\sharp M_i(V') = \bot^\sharp$，那么上式可简化为 $\bot^\sharp \sqsubseteq^\sharp Y$，该式成立。现在，假设 $d \sqcap^\sharp M_i(V') \neq \bot^\sharp$。展开 $Y$ 后，我们的证明义务变为：

$$\big(d \sqcap^\sharp M_i(V')\big) \lightning V' \sqsubseteq^\sharp \left( \bigsqcup_{V \in \mathcal{V}_N}^\sharp d \sqcap^\sharp M_i(V) \right) \lightning \bigcup_{V \in \mathcal{V}_N} V. \tag{1}$$

由于 $|V'| \geq N+1$，它是其所有 $(N+1)$-子集 $V'_{sub}$ 的并集，形式化定义如下：

$$V'_{sub} = \{V \in \mathcal{P}(\mathcal{V}) \mid V \subseteq V' \wedge |V| = N+1\}.$$

此外，对于任意 $V \in V'_{sub}$，我们有 $M_i(V') \sqsubseteq^\sharp M_i(V)$，因此 $d \sqcap^\sharp M_i(V) \neq \bot^\sharp$。于是，根据 $\mathcal{V}_N$ 的定义：

$$V'_{sub} \subseteq \mathcal{V}_N. \tag{2}$$

由此可得：

$$V' = \bigcup_{V \in V'_{sub}} V \subseteq \bigcup_{V \in \mathcal{V}_N} V. \tag{3}$$

设 $V^-$ 是 $V'_{sub}$ 中的一个变量集。那么，由于 $V^-$ 是 $V'$ 的子集，我们有 $M_i(V') \sqsubseteq^\sharp M_i(V^-)$，因此由 (2) 可得：

$$d \sqcap^\sharp M_i(V') \sqsubseteq^\sharp d \sqcap^\sharp M_i(V^-) \sqsubseteq^\sharp \bigsqcup_{V \in \mathcal{V}_N}^\sharp d \sqcap^\sharp M_i(V). \tag{4}$$

回想一下，$\lightning$ 是单调的，即：

$$\forall V_1, V_2 \in \mathcal{P}(\mathcal{V}), d_1, d_2 \in \mathcal{D} : V_1 \subseteq V_2 \wedge d_1 \sqsubseteq^\sharp d_2 \implies d_1 \lightning V_1 \sqsubseteq^\sharp d_2 \lightning V_2.$$

因此，由 (3) 和 (4) 可推导出 (1)。所以 $Y$ 确实是保守的，我们可以相应地简化定义：

$$\begin{aligned}
&\mathit{stabilise}_N(i, d) \\
&= d \sqcup^\sharp X \sqcup^\sharp Y \\
&= d \sqcup^\sharp X \sqcup^\sharp \left( \bigsqcup_{V \in \mathcal{V}_N}^\sharp d \sqcap^\sharp M_i(V) \right) \lightning \bigcup_{V \in \mathcal{V}_N} V \\
&\sqsupseteq^\sharp d \sqcup^\sharp X \sqcup^\sharp \bigsqcup_{V \in \mathcal{P}(\mathcal{V}),\, |V| > N}^\sharp \big(d \sqcap^\sharp M_i(V)\big) \lightning V \\
&= d \sqcup^\sharp \left( \bigsqcup_{V \in \mathcal{P}(\mathcal{V}),\, |V| \leq N}^\sharp \big(d \sqcap^\sharp M_i(V)\big) \lightning V \right) \\
&\quad \sqcup^\sharp \left( \bigsqcup_{V \in \mathcal{P}(\mathcal{V}),\, |V| > N}^\sharp \big(d \sqcap^\sharp M_i(V)\big) \lightning V \right) \\
&= d \sqcup^\sharp \bigsqcup_{V \in \mathcal{P}(\mathcal{V})}^\sharp \big(d \sqcap^\sharp M_i(V)\big) \lightning V.
\end{aligned}$$

因此，我们的目标是证明：

$$\gamma^\sharp\left( d \sqcup^\sharp \bigsqcup_{V\in\mathcal{P}(\mathcal{V})}^\sharp \left(d \sqcap^\sharp M_i(V)\right) \not\downarrow V \right) \supseteq \gamma^\sharp(d) \cup \{s_2 \in \mathcal{S} \mid \exists s_1 \in \gamma^\sharp(d) : (s_1, s_2) \in \gamma^\times(i)\}. \tag{5}$$

情况 1：设 $s \in \gamma^\sharp(d)$。那么对于任意 $d' \in \mathcal{D}$，我们有 $s \in \gamma^\sharp(d \sqcup^\sharp d')$，因此 $s$ 属于 (5) 的左侧。情况 2：设 $s_2 \in \mathcal{S}$，其中 $\exists s_1 \in \gamma^\sharp(d) : (s_1, s_2) \in \gamma^\times(i)$。那么根据 $\mathcal{I}$ 的具体化函数，我们有：

$$\forall v \in \mathcal{V} : s_2(v) \neq s_1(v) \implies s_1 \in \gamma^\sharp(i(v)).$$

设 $P$ 是 $\mathcal{V}$ 的极大子集，其中 $\forall v \in P : s_2(v) \neq s_1(v)$。那么 $s_1 \in \gamma^\sharp(M_i(P))$，并且由于 $s_1 \in \gamma^\sharp(d)$，我们有 $s_1 \in \gamma^\sharp(d \sqcap^\sharp M_i(P))$。由于 $P$ 是如上定义的极大子集，我们有 $\forall v \in \mathcal{V} \setminus P : s_1(v) = s_2(v)$。现在回想一下我们对“$\not\downarrow$”算子的约束：

$$\gamma^\sharp(d \not\downarrow V) \supseteq \left\{s \in \mathcal{S} \mid \exists s_0 \in \gamma^\sharp(d) : \forall v \in \mathcal{V} \setminus V : s(v) = s_0(v)\right\}.$$

因此，在上式中取 $s_1$ 为 $s_0$，取 $d \sqcap^\sharp M_i(P)$ 为 $d$，可得：

$$s_2 \in \gamma^\sharp(d \sqcap^\sharp M_i(P)) \not\downarrow P.$$

因此，由于 $P \in \mathcal{P}(\mathcal{V})$，我们有：

$$s_2 \in \gamma^\sharp\left( \bigsqcup_{V\in\mathcal{P}(\mathcal{V})}^\sharp \left(d \sqcap^\sharp M_i(V)\right) \not\downarrow V \right)$$

因此，$s_2$ 属于 (5) 的左侧。

### A.2 转换的可靠性

**引理 2**。对于任意 $d \in \mathcal{D}$ 和赋值 $A = \langle x_1, ..., x_n := e_1, ..., e_n\rangle$，我们有：

$$\gamma^\times(\mathit{transitions}(d, A)) \supseteq \left\{(s_1, s_2) \in \mathcal{S} \times \mathcal{S} \mid s_1 \in \gamma^\sharp(d) \wedge s_2 = \mathit{exec}(A, s_1)\right\}.$$

证明。回忆转换的定义：

$$\mathit{transitions}(d, \langle x_1, ..., x_n := e_1, ..., e_n\rangle) \mathrel{\widehat{=}} \bot^\times[v \in \{x_1, ..., x_n\} \mapsto d]$$

以及 $\mathcal{I}$的具体化函数：

$$\gamma^\times(i) \mathrel{\widehat{=}} \left\{(s, s') \in \mathcal{S} \times \mathcal{S} \mid \forall v \in \mathcal{V} : s'(v) \neq s(v) \implies s \in \gamma^\sharp(i(v))\right\}.$$

只需证明，对于所有 $(s_1, s_2) \in \mathcal{S}\times\mathcal{S}$其中 $s_1 \in \gamma^\sharp(d) \wedge s_2 = \mathit{exec}(A, s_1)$，我们都有 $(s_1, s_2) \in \gamma^\times(\text{转换}(d, A))$。即：

$$\forall v \in \mathcal{V} : s_2(v) \neq s_1(v) \implies s_1 \in \gamma^\sharp(\bot^\times[v_0 \in \{x_1, ..., x_n\} \mapsto d](v)). \tag{1}$$

我们期望我们的具体语义满足：

$$x_1' = e_1 \wedge ... \wedge x_n' = e_n \wedge \forall v \notin \{x_1, ..., x_n\} : v' = v.$$

设 $v \in \mathcal{V}$ 是某个变量，其中 $s_2(v) \neq s_1(v)$。那么，由上文可得 $v \in \{x_1, ..., x_n\}$。因此：

$$\bot^\times[v_0 \in \{x_1, ..., x_n\} \mapsto d](v) = d$$

因此，(1) 中的结论可化简为 $s_1 \in \gamma^\sharp(d)$，这一点已得到证明。

### A.3 闭合的可靠性

**引理 3.** 设某个 $i \in \mathcal{I}$ 满足：

$$\forall v \in \mathcal{V} : i(v) \sqsupseteq^\sharp \bigsqcup_{V \in \mathcal{P}(\mathcal{V})}^\sharp i(v) \lightning V \sqcap^\sharp \prod_{v' \in V}^\sharp i(v'). \tag{1}$$

证明 $i$ 满足传递性：

$$\forall s_1, s_2, s_3 \in \mathcal{S} : (s_1, s_2) \in \gamma^\times(i) \wedge (s_2, s_3) \in \gamma^\times(i) \implies (s_1, s_3) \in \gamma^\times(i).$$

证明。 假设 $i$ 不是传递的。则必定存在状态 $s_1, s_2, s_3 \in \mathcal{S}$，其中：

$$(s_1, s_2) \in \gamma^\times(i) \wedge (s_2, s_3) \in \gamma^\times(i) \wedge (s_1, s_3) \notin \gamma^\times(i).$$

回顾 $\gamma^\times(i)$ 的定义：

$$\gamma^\times(i) \mathrel{\widehat{=}} \left\{(s, s') \in \mathcal{S} \times \mathcal{S} \mid \forall v \in \mathcal{V} : s'(v) \neq s(v) \implies s \in \gamma^\sharp(i(v))\right\}.$$

根据这一定义，由于 $(s_1, s_3) \notin \gamma^\times(i)$，必定存在某个变量，记为 $\hat{}v$，其中：

$$s_3(\hat{v}) \neq s_1(\hat{v}) \wedge s_1 \notin \gamma^\sharp(i(\hat{v})).$$

由于 $s_1 \notin \gamma^\sharp(i(\hat{}v))$，根据 $\gamma^\times(i)$ 的定义，我们有 $s_1(\hat{}v) = s_2(\hat{}v)$。因此，由于$s_1(\hat{}v) \neq s_3(\hat{}v)$，我们有 $s_2(\hat{}v) \neq s_3(\hat{}v)$，从而有 $s_2 \in \gamma^\sharp(i(\hat{}v))$。因此我们有：

$$s_1 \notin \gamma^\sharp(i(\hat{v})) \wedge s_2 \in \gamma^\sharp(i(\hat{v})). \tag{2}$$

直观地说，如果 $i$ 不是传递的，那么它刻画了从某个不在 $\hat{}v$ 的写入条件中的状态到一个处于该写入条件中的状态的转移。我们现在将证明，在给定 (1) 的情况下这是不可能的。首先，令 $V^\Delta$ 为在 $s_1$ 与 $s_2$ 中取值不同的所有变量的集合。于是我们有：

$$\forall v \in \mathcal{V} \setminus V^\Delta : s_1(v) = s_2(v) \tag{3}$$

$$s_1 \in \bigcap_{v \in V^\Delta} \gamma^\sharp(i(v)). \tag{4}$$

回顾我们对 $\lightning$ 的约束：对于任意 $d \in \mathcal{D}, V \in \mathcal{P}(\mathcal{V})$：

$$\gamma^\sharp(d \lightning V) \supseteq \left\{s \in \mathcal{S} \mid \exists s_0 \in \gamma^\sharp(d) : \forall v \in \mathcal{V} \setminus V : s(v) = s_0(v)\right\}.$$

将 $i(\hat{v})$ 代入 $d$，并将 $V^{\Delta}$ 代入 $V$，得到：

$$\gamma^{\sharp}\left(i(\hat{v}) \not\downarrow V^{\Delta}\right) \supseteq \left\{s \in \mathcal{S} \mid \exists s_0 \in \gamma^{\sharp}(i(\hat{v})) : \forall v \in \mathcal{V} \setminus V^{\Delta} : s(v) = s_0(v)\right\}.$$

现在，由 (2) 我们有 $s_2 \in \gamma^{\sharp}(i(\hat{v}))$，并由 (3) 可知 $\forall v \in \mathcal{V} \setminus V^{\Delta} : s_1(v) = s_2(v)$。因此，$s_1$ 是上述右侧集合的一个元素，从而：

$$s_1 \in \gamma^{\sharp}\left(i(\hat{v}) \not\downarrow V^{\Delta}\right). \tag{5}$$

由 (4) 和 (5) 我们有：

$$\begin{aligned}
s_1 &\in \gamma^{\sharp}\left(i(\hat{v}) \not\downarrow V^{\Delta}\right) \cap \bigcap_{v \in V^{\Delta}} \gamma^{\sharp}(i(v)) \\
&\subseteq \bigcup_{V \in \mathcal{P}(\mathcal{V})} \gamma^{\sharp}\left(i(\hat{v}) \not\downarrow V\right) \cap \bigcap_{v \in V} \gamma^{\sharp}(i(v)) \\
&\subseteq \gamma^{\sharp}\left( \bigsqcup_{V \in \mathcal{P}(\mathcal{V})}^{\sharp} i(\hat{v}) \not\downarrow V \sqcap^{\sharp} \prod_{v \in V}^{\sharp} i(v) \right) \\
&\subseteq \gamma^{\sharp}(i(\hat{v})) \text{ from } (1).
\end{aligned}$$

因此 $s_1 \in \gamma^{\sharp}(i(\hat{v}))$，这与 (2) 矛盾。

# B 优化

## B.1 计算稳定化的优化

如第 4.2 节所示，对于较大的 $N$，计算 稳定化$_N(i,\ d)$ 可能会很慢，因为需要遍历 $\mathcal{P}(\mathcal{V})$ 的很大一部分。实际上，我们可以通过简单的优化来缩小此域。对于任意变量集 $V_0 \in \mathcal{P}(\mathcal{V})$，如果该变量集中变量的写条件的交为 $\bot^{\sharp}$，则可以忽略 $V_0$ 的任何超集。形式上，我们可以忽略任何满足以下条件的 $V \in \mathcal{P}(\mathcal{V})$：

$$\exists V_0 \in \mathcal{P}(\mathcal{V}) : V_0 \subseteq V \wedge \prod_{v \in V_0}^{\sharp} i(v) = \bot^{\sharp}.$$

这是因为，既然 $V$ 是 $V_0$的超集，$V$ 中变量的写条件的交也为 $\bot^{\sharp}$：

$$\prod_{v \in V}^{\sharp} i(v) = \bot^{\sharp}$$

因此：

$$\left(d \sqcap^{\sharp} \prod_{v \in V}^{\sharp} i(v)\right) \not\downarrow V = \bot^{\sharp}.$$

要充分利用这一优化，最好采用自底向上遍历 $\mathcal{P}(\mathcal{V})$ 的实现。当遇到满足 $\prod_{v \in V_0}^{\sharp} i(v) = \bot^{\sharp}$ 的某个 $V_0 \in \mathcal{P}(\mathcal{V})$ 时，我们可以忽略 $V_0$ 的所有超集。

### B.2 计算闭合的优化

我们对闭合 的定义要求对 $\mathcal{P}(\mathcal{V})$ 求并。尽管这很慢，我们仍可以实施两项重要优化，大幅缩小这个域。

**优化 1** 我们只需考虑 $i(v)$ 中变量的幂集。也就是说，我们可以从域中移除任何包含某个 $u \in V$ 的变量集$V \in \mathcal{P}(\mathcal{V})$，使得 $i(v)\lightning\{u\} = i(v)$。这是因为，正如第3节中所述：

$$\forall V_1, V_2 \in \mathcal{P}(\mathcal{V}), d \in \mathcal{D} : d\lightning V_1 = d \implies d\lightning V_1 \cup V_2 = d\lightning V_2$$

因此，取 $d = i(v)$、 $V_1 = \{u\}$ 和 $V_2 = V \setminus \{u\}$：

$$i(v)\lightning V = i(v)\lightning(V \setminus \{u\})$$

因此：

$$i(v)\lightning V \sqcap^\sharp \prod_{v' \in V}^\sharp i(v') \sqsubseteq^\sharp i(v)\lightning(V \setminus \{u\}) \sqcap^\sharp \prod_{v' \in V \setminus \{u\}}^\sharp i(v').$$

由于域元素 $V \setminus \{u\}$ 所捕获的状态是 $V$ 所捕获状态的超集，因此我们可以安全地从域中移除 $V$。

**优化 2** 在遍历 $\mathcal{P}(\mathcal{V})$ 时，如果遇到某个 $V_0 \in \mathcal{P}(\mathcal{V})$，且 havoc 操作的结果完全包含 $V_0$ 中变量的写入条件的交，那么我们可以安全地忽略 $V_0$ 的任何严格超集 $V$。形式化地说，我们可以忽略任何满足以下条件的变量集 $V \in \mathcal{P}(\mathcal{V})$：

$$\exists V_0 \subset V : i(v)\lightning V_0 \sqsupseteq^\sharp \prod_{v' \in V_0}^\sharp i(v'). \tag{1}$$

这是因为：

$$i(v)\lightning V_0 \sqcap^\sharp \prod_{v' \in V_0}^\sharp i(v') = \prod_{v' \in V_0}^\sharp i(v')$$

因此，对于变量集 $V_0$，只需忽略 havoc 操作，并将交操作的结果并起来即可。现在注意，由于 $V \supset V_0$，我们有：

$$\prod_{v' \in V}^\sharp i(v') \sqsubseteq^\sharp \prod_{v' \in V_0}^\sharp i(v')$$

因此：

$$i(v)\lightning V \sqcap^\sharp \prod_{v' \in V}^\sharp i(v') \sqsubseteq^\sharp \prod_{v' \in V_0}^\sharp i(v').$$

由于域元素 $V_0$ 所捕获的状态是 $V$ 所捕获状态的超集，因此我们可以安全地从域中移除 $V$。请注意，使用此优化需要对 $\mathcal{P}(\mathcal{V})$ 进行自底向上遍历。

## C 条件写入域的形式化摘要

本节对条件写入域进行了完整的形式化总结。我们定义：

具体化函数 $\gamma^\times$： $\mathcal{I} \to \mathcal{P}(\mathcal{S} \times \mathcal{S})$：

$$\gamma^\times(i) \;\widehat{=}\; \left\{(s_1, s_2) \in \mathcal{S} \times \mathcal{S} \mid \forall v \in \mathcal{V} : s_2(v) \neq s_1(v) \implies s_1 \in \gamma^\sharp(i(v))\right\}.$$

顶元素（$\top^\times$）和底元素（$\bot^\times$）：

$$\top^\times \;\widehat{=}\; \lambda v \in \mathcal{V} : \top^\sharp$$
$$\bot^\times \;\widehat{=}\; \lambda v \in \mathcal{V} : \bot^\sharp.$$

并运算符 $\sqcup^\times$： $\mathcal{I} \times \mathcal{I} \to \mathcal{I}$，按分量定义：

$$i_1 \sqcup^\times i_2 \;\widehat{=}\; \lambda v \in \mathcal{V} : i_1(v) \sqcup^\sharp i_2(v).$$

交运算符 $\sqcap^\times$： $\mathcal{I} \times \mathcal{I} \to \mathcal{I}$，同样按分量定义：

$$i_1 \sqcap^\times i_2 \;\widehat{=}\; \lambda v \in \mathcal{V} : i_1(v) \sqcap^\sharp i_2(v).$$

函数 稳定化$_N$：$\mathcal{I} \times \mathcal{D} \to \mathcal{D}$，满足：

$$\gamma^\sharp(\mathit{stabilise}_N(i, d)) \supseteq \gamma^\sharp(d) \cup \{s_2 \in \mathcal{S} \mid \exists s_1 \in \gamma^\sharp(d) : (s_1, s_2) \in \gamma^\times(i)\}$$

定义如下：

$$\text{let } X = \bigsqcup^\sharp_{V \in \mathcal{P}(\mathcal{V}),\, |V| \leq N} \left( d \sqcap^\sharp \prod^\sharp_{v \in V} i(v) \right) \lightning V \text{ in}$$
$$\text{let } \mathcal{V}_N = \left\{ V \in \mathcal{P}(\mathcal{V}) \;\middle|\; |V| = N + 1 \wedge \left( d \sqcap^\sharp \prod^\sharp_{v \in V} i(v) \right) \neq \bot^\sharp \right\} \text{ in}$$
$$\text{let } Y = \left( \bigsqcup^\sharp_{V \in \mathcal{V}_N} d \sqcap^\sharp \prod^\sharp_{v \in V} i(v) \right) \lightning \bigcup_{V \in \mathcal{V}_N} V \text{ in}$$
$$d \sqcup^\sharp X \sqcup^\sharp Y.$$

函数 转换：$\mathcal{D} \times$赋值 $\to \mathcal{I}$，满足：

$$\gamma^\times(\mathit{transitions}(d, A)) \supseteq \{(s_1, s_2) \in \mathcal{S} \times \mathcal{S} \mid s_1 \in \gamma^\sharp(d) \wedge s_2 = \mathit{exec}(A, s_1)\}$$

并定义为：

$$\mathit{transitions}(d, \langle x_1, ..., x_n := e_1, ..., e_n \rangle) \;\widehat{=}\; \bot^\times[v \in \{x_1, ..., x_n\} \mapsto d].$$

满足以下条件的（弱化）传递闭包函数 闭合 : $\mathcal{I} \to \mathcal{I}$:

$$\begin{aligned} & close(i) \sqsupseteq^{\times} i \\ \wedge\, & \forall s_1, s_2, s_3 \in \mathcal{S} : (s_1, s_2) \in \gamma^{\times}(close(i)) \wedge (s_2, s_3) \in \gamma^{\times}(close(i)) \\ & \qquad \implies (s_1, s_3) \in \gamma^{\times}(close(i)) \end{aligned}$$

并定义为:

$$close(i) \;\widehat{=}\; \mu i : \lambda v : \bigsqcup_{V \in \mathcal{P}(\mathcal{V})}^{\sharp} i(v) \lightning V \sqcap^{\sharp} \bigsqcap_{v' \in V}^{\sharp} i(v').$$