arXiv:2301.07783v1 [cs.PL] 18 Jan 2023

通过抽象解释进行的可靠符号执行及其在安全领域的应用

Ignacio Tiraboschi[2], Tamara Rezk[1], 和 Xavier Rival[2]

[1] INRIA，蔚蓝海岸大学′，索菲亚安提波利斯，法国[2] INRIA 巴黎，DI ENS，高等师范学院′，PSL 大学′，法国国家科学研究中心 name.surname@inria.fr

摘要。 符号执行是一种程序分析技术，常用于判断程序是否违反属性；如果发现违规，还会生成能够触发违规的输入。在非干扰等安全属性的分析中，符号执行能够精确地找出不安全信息流对应的反例轨迹对；然而，它的可靠性仅限于某个界限，因此无法证明执行超出给定界限的程序正确。相比之下，基于抽象解释的静态分析能够保证可靠性，但通常无法提供反例轨迹对。本文提出将两者结合起来，以兼得其优势。我们通过一系列静态分析展示了这一方法，其中包括一种名为 RedSoundRSE、旨在验证非干扰的静态分析。RedSoundRSE 既能给出语义上可靠的结果，也能在某个界限内推导出反例轨迹对。它结合了符号执行与受著名约简积概念启发的抽象域。我们对 RedSoundRSE 进行形式化，并证明其可靠性以及其在某个界限内的相对精度。我们还提供了 RedSoundRSE 的原型实现，并在一组复杂示例上对其进行了评估。



1 引言

安全属性向来很难验证。尤其是，许多安全属性并非单次执行属性，而是超属性 [13]（也称为关系属性），这意味着有时需要提供多条执行轨迹作为反例，才能证伪这些属性。具体而言，非干扰 [26] 指出，在程序的任何一次执行中，高安全级信息都不应影响低安全级用户的观测

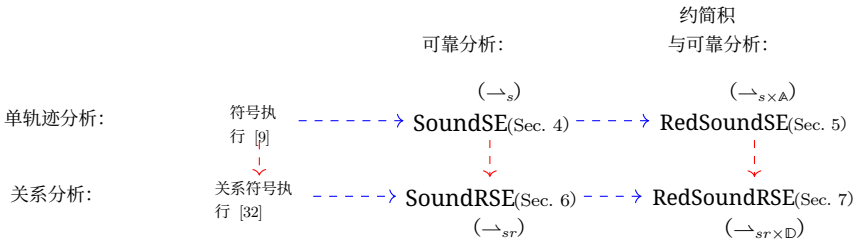

图1. 不同符号执行分析之间的关系。SE [9] 是传统的符号执行，RSE [32,34] 则是其针对关系属性的扩展。除带有不变式的 RSE [23], 外，SE 和 RSE 通常都不可靠。其余分析均是可靠的，也是我们的贡献：SoundSE 和 SoundRSE 不使用抽象解释，而 RedSoundSE 和 RedSoundRSE 可与不同的抽象域结合。红色虚线表示依赖关系：关系分析依赖于单轨迹分析。蓝色虚线表示对分析的增强。

。这一直是许多验证方法提案和工具（例如 [36,7,37,24,38,8,23,4,33]）的研究主题。

符号执行 [9,31]（SE）通常用于查找性质违例；经过针对关系属性的适当调整后，也可用于非干扰等策略。SE 归结为这样一种执行：变量最初取符号值，并通过这些符号值的表达式更新，而条件则被求值为符号路径守卫。该分析会使用 SMT 求解器等外部工具，剪除不可行路径，并尝试证明其余路径上的验证条件。SE 试图穷尽覆盖所有执行路径，但这只有在界限范围内才可行；存在无界循环时，成本很快就会变得高昂。

传统的符号执行不会对固定迭代次数界限之后的执行进行过近似。这意味着，当程序超出探索界限时，可靠性 便会丧失。可靠性保证：当分析得出所关注的属性成立时，被分析程序的具体语义必然满足该属性。由于没有进行过近似，当轨迹长度短于探索界限且违反了该属性时，SMT 求解器等工具可以为符号值提供实例，并支持重建反例轨迹。这对于安全性尤为重要，因为它有助于确认安全违例。我们将此类反例称为**反驳模型**。

要调整符号执行以处理关系属性 [32,34]，就需要跟踪多条轨迹，而不只是单条轨迹。下文将这种调整称为关系符号执行 （简称 RSE）。先前的工作 [23] 展示了如何将 RSE 与开发者提供的循环不变式结合起来，以恢复可靠性；代价是需要添加注释，并且当不变式不够强时会损失精度。

基于抽象解释的静态分析 [14]（人工智能）依赖一种抽象，该抽象由具体行为与抽象谓词之间的逻辑近似关系定义，并以牺牲完备性为代价，对程序语义进行可靠的过近似。然而，这种过近似意味着，即使在分析正确的程序时，分析也可能无法

得出肯定结论。此外，大多数静态分析实现都缺乏生成反例轨迹的能力。

本文在安全性场景下，形式化了一种组合分析技术，旨在结合符号执行与抽象解释各自的优势。我们首先展示如何对符号执行进行过近似以确保其可靠性，并将这种分析称为 SoundSE（见图 1）。我们使用 SoundSE 展示传统符号执行与不同抽象域的组合方式，并将得到的分析称为 RedSoundSE。针对关系属性的分析称为 RedSoundRSE，目标是验证非干扰性。它借鉴了关系符号执行的路径探索，并以 RedSoundSE 为参数；同时依赖基于抽象解释的静态分析，为所有程序报告可靠的结果。抽象能够尽早剪除不可行路径，并在探索界限耗尽时，计算出程序行为的可靠过近似。为此，RedSoundRSE 会自动将抽象域计算出的循环不变式注入关系存储中。依赖分析的结果不仅可用于填补符号执行无法充分探索路径时缺失的安全相关信息，还可借助（例如数值）状态抽象信息，改进从依赖分析中提取的符号信息。此外，我们的分析支持在不同抽象之间切换，并调整特定设置，例如循环展开深度（即符号执行保持精确的深度），使用户能够调整成本与精度之间的平衡。总而言之，我们提出了基于符号执行的验证方法，既可靠又精确，并能在给定界限内提供反驳模型。我们的贡献如图 1 所示，具体如下：

1. SoundSE 和 RedSoundSE：我们定义了一种可靠的符号执行分析，并将数值抽象域整合其中，以剪除可达路径。因此，我们使符号执行 [9] 变得可靠，同时保留分析发现反例的能力。2. SoundRSE 和 RedSoundRSE：我们定义了一种可靠的关系符号执行，并将其与依赖分析 [4] 结合，以提高后者的精度，同时保持可靠性。3. 我们使用 OCaml 对 RedSoundRSE 和 RedSoundSE 进行了原型实现，并通过一系列复杂示例展示了该方法既能可靠地判定安全程序是否满足非干扰，也能为不安全程序合成大小不超过给定界限的反例。

本文结构如下。第 2 节定义了一种基础语言以及贯穿全文的非干扰概念。第 3 节概述已有的分析方法，并重点介绍 Red-SoundRSE 的主要原理。第 4 节定义 SoundSE，这是一种可靠的单轨迹符号执行，为 RedSoundRSE 奠定基础；第 5 节介绍 RedSoundSE，这是 SoundSE 与状态抽象的一种新组合。第 6 节介绍 SoundRSE，第 7 节则通过加入依赖抽象对其进行扩展，从而得到 RedSoundRSE。第 8 节使用规模较小但颇具挑战性的示例评估我们的框架。最后，第 9 节讨论相关工作，第 10 节作出总结。附录包含本文中所有分析方法的规则。

2 语言与非干扰安全概念

本节介绍我们形式化分析方法所针对的语言和安全概念。令 V 和 X 分别为值集合和程序变量集合，$\oplus$、$\ominus$ 为二元运算符。布尔表达式**b** 是将比较运算符 $\ominus$应用于两个表达式后得到的结果，其值为布尔值 B = {**tt**、**ff**}。语句**s** 可以是空操作、赋值、条件语句或循环。最后，命令**c** 是由有限个语句组成的序列。程序 P 是一个二元组（**c** L），由命令**c**（程序主体）和低安全级变量 $L \subseteq$X 的集合构成；这些变量可公开观察（程序中出现的其他变量均为高安全级变量）。

$$\begin{array}{ll} \mathbf{e} ::= v\ (v \in \mathbb{V}) \mid x\ (x \in \mathbb{X}) \mid \mathbf{e} \oplus \mathbf{e} & \mathbf{b} ::= \mathbf{e} \ominus \mathbf{e} \\ \mathbf{s} ::= \texttt{skip} \mid x := \mathbf{e} \mid \texttt{if}\ \mathbf{b}\ \texttt{then}\ \mathbf{c}\ \texttt{else}\ \mathbf{c} \mid \texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c} & \mathbf{c} ::= \mathbf{s} \mid \mathbf{s}; \mathbf{c} \end{array}$$

语义。给定一个程序 (c L)，状态是一个二元组 (c μ)，其中 c 是一条命令，μ 是从 X 到 V 的函数，也就是一个存储。特别地，形如 (skip μ) 的状态是终态。我们分别用 M 和 S 表示存储集和状态集。我们用 $[\mathrm{x} \mapsto x, \mathrm{y} \mapsto y, \ldots]$ 显式枚举存储的内容，其中 $x, y, \ldots$ 是具体值。令 $(\rightarrow) \subseteq \mathrm{S} \times \mathrm{S}$ 表示小步操作语义（这是标准定义），并令 $\rightarrow^*$ 表示其自反传递闭包。

非干扰。 令 $=_L$ 为限制在L中低安全级变量上的存储相等关系。在本文余下部分，我们关注终止不敏感非干扰：

定义 *1*（终止不敏感非干扰）。程序 *(c L)* 是终止不敏感非干扰的，记作 $\mathcal{NI}_P^{T,I}$，当且仅当，对于所有存储$\mu_0, \mu_1, \mu_0', \mu_1' \in M$，$\mu_0 =_L \mu_1 \wedge$*(c* μ_0*)* $\rightarrow^*$*(skip* μ_0'*)* $\wedge$*(c* μ_1*)* $\rightarrow^*$ *(skip* μ_1'*)* $\Longrightarrow \mu_0' =_L \mu_1'$。

3 概述

本节将展示 RedSoundRSE 如何结合符号执行与抽象，以克服单独采用这两种方法各自的局限性。与本文其他部分一样，我们关注非干扰（NI），尽管同样的原理也适用于其他安全属性。

示例。 我们考虑图2所示的程序。从本质上说，程序 (a) 和 (b) 满足非干扰策略下的安全性，其中 priv 为高安全级，所有其他变量均为低安全级；而程序 (c) 则不安全。

在程序2(a)中，变量 y 被赋值为 5，且该赋值独立于 priv，因此程序是安全的。对于程序2(b)，设 μ_0，μ_1 为两个满足 $\mu_0 =_L \mu_1$ 的存储。由于μ_0 和 μ_1 是低相等的执行，因此它们不可能走不同的路径，而且循环执行的次数也相同。因此，程序是安全的。最后，程序 2(c)是不安全的，也就是说，它不满足非干扰。我们需要提供一个反例，其中包含两个从低相等存储 μ_0，μ_1 开始的执行，使得对应的输出存储 μ_0'，μ_1' 不低相等。我们考虑以下

```
1 如果(priv  > 0)
2   y  = 5;
3 else
4   y  = 5;
```
(a) 安全 program

```
1 当 (i < z) {
2   i =i  + 1;
3   priv  = priv + 5;
4 }
```
(b) 安全程序

```
1 当 (i  > priv){
2   i =i  + 1;
3   priv  = priv + 2;
4 }
```
(c) 不安全程序

```
1 如果(priv  < 0) priv  = 0;
2 当(i  < 10){
3   i  += 1; priv += 2;
4 }
5 如果(priv  >= 0) y += 1;
6 否则 y = 0;
```
(d) 安全程序需要——一个数值域。

图2. 示例程序。所有变量的类型均为整数类型，其中变量 priv 是安全的。

	安全吗？关系符号执行		依赖分析	RedSoundRSE
程序2(a)	Yes	✓ 安全	✗误报	✓ 安全
程序2(b)	Yes	✗误报	✓ 安全	✓ 安全
程序 2(c)	No	✓反驳模型	✓ 警报	✓反驳模型

表 1. 分析结果比较。符号 ✓ （分别表示 ✗）表示语义正确（分别表示语义不正确）的分析结果，结果包括安全性证明、警报（可能为误报）或反驳模型。

存储： $\mu_0 = [\mathrm{i} \mapsto 0,\ \mathrm{priv} \mapsto 0]$，以及 $\mu_1 = [\mathrm{i} \mapsto 0,\ \mathrm{priv} \mapsto -1]$。最后，计算得到的输出存储满足 $\mu_0'(\mathrm{i}) = 0 \neq \mu_1'(\mathrm{i}) = 1$，因此该程序违反了非干扰。

接下来的段落中，我们将研究验证方法对这三个程序的分析结果。

基于关系符号执行的验证。 符号存储（记作 ρ）将变量映射到由变量初始值构成的符号表达式。为避免混淆，我们用斜体等宽字体表示这些符号值，而程序变量则用正体等宽字体表示。例如，y 表示 y 的初始值。关系符号执行使用关于变量初始值的符号条件以及成对的符号存储来描述执行对。符号存储不足以抽象执行，因为它们无法表达约束。约束则由符号路径 π 提供，以此为存储补充上下文。符号存储与符号路径组成的二元组（ρ，π）称为**符号精确存储**。

例如，我们考虑程序2(a)。关系符号执行根据两次执行中 priv 的初始值的符号，揭示了四对路径。例如，其中一条分歧路径产生 $\pi = (priv_0 > 0 \wedge priv_1 \leq 0) \Longrightarrow ([\mathrm{y}_0 \mapsto 5,\ \ldots],\ [\mathrm{y}_1 \mapsto 5,\ \ldots])$，其中 y_0 和 y_1 表示两次执行中的程序变量 y，而 $priv_0$，$priv_1$ 表示 priv 的初始符号值。这个符号精确存储表明，流向 y 的信息流不存在，因为任何 SMT 求解器都能证明 $\mathrm{y}_0 = \mathrm{y}_1$。另外三对路径也得出类似结果，因此该程序被证明是安全的。

对于程序2(b)，循环的迭代次数没有上限，但关系符号执行只能覆盖有限次循环展开。这使得关系符号执行无法证明程序2(b)是安全的。

对于程序 2(c)，关系符号执行只会在某个界限内探索循环。假设该界限为 1（任何正值都能证明类似的结论），它可以通过计算出一条违反该性质的具体轨迹，确定程序不满足非干扰。该反例轨迹由 SMT 求解器计算得出，例如，$i_0 = i_1 = 0$、$priv_0 = 1$ 和 $priv_1 = -1$ 对应于前面给出的反例。

基于依赖抽象的验证。许多用于非干扰分析的静态分析都依赖某种形式的依赖抽象，例如 [4] 或 [28] 中形式化的抽象。我们简要概述 [4] 中的抽象。假设安全级别构成有序集合 {L，H}，且通过输入变量输入程序的每个值都被赋予一个安全级别。依赖关系记作 $l \rightsquigarrow$x with $l \in$ {L，H}，表示从级别 l 观察时，x 在两次执行中保持一致。此分析基于抽象解释，且是可靠的。

现在我们讨论图2中一些程序的分析。对于程序2(a)，分析确定这些赋值受 priv 的值控制，而 priv 初始为高安全级。随后，依赖关系 L$\rightsquigarrow$y 被丢弃，这表明 y 在不同执行中可能不一致。在程序2(b)中，循环条件只受 i 和 z 影响，而它们都是低安全级。于是，低安全级变量的赋值不受影响，i 和 z 仍保持低安全级，从而可以证明非干扰。

最后，程序 2(c) 并不安全；由于依赖分析是可靠的，该分析会根据非法的信息流舍弃依赖关系 L$\rightsquigarrow$i。

关系符号执行与依赖抽象的结合。如表 1 所示，关系符号执行无法精确处理程序 2(b)，而依赖抽象无法验证程序 2(a)，也无法为程序 2(c) 提供反例。RedSoundRSE 旨在交替运用这两种技术，以提高精度并剪除分支。

为此，RedSoundRSE 借鉴关系符号执行，对赋值和条件命令进行精确分析，并展开循环命令的迭代。具体而言，程序 2(a) 和 2(c) 的分析如上所示。然而，达到展开界限后，便使用依赖分析，以便在有限时间内可靠地推导出关于任意数量后续循环迭代的信息。实际上，当依赖信息证明某个循环不会导致给定低变量依赖于任何高变量时，就可以假定该变量在符号存储中的值相等。这个新值可能无法用初始值精确表示，因此可以用一个新符号近似表示。程序 2(b) 中的变量 i 就是这种情况。

如上所示，RedSoundRSE 能够精确分析图 2 中的前三个示例。

基于状态抽象的符号执行改进。 此前未曾考虑过的程序图2(d)，仅使用符号执行和依赖分析无法证明其满足非干扰。该程序是安全的，因为对 i 的赋值不依赖于 priv，并且循环结束后，y 受 priv 条件约束，而 priv 始终为正。

如程序2(b)所示，循环会导致符号执行在展开界限处停止。依赖信息可以证明不存在流向 i 的信息流，也可以证明第 8 行的 y 值不依赖于 priv。然而，

第 9 行的条件依赖于 priv，因此依赖分析无法证明第 10 行和第 12 行的赋值不会泄露信息。符号执行也无法成功，因为它缺乏推理循环退出时 priv 值的能力。

这类信息可以通过可达性静态分析计算得出。具体而言，基于区间抽象域的经典静态分析会计算所有数值变量的取值范围，并据此得出在此情形下 priv 为正，因此只能进入条件的真分支。集成非关系抽象域后，分析器可以自动剪除路径，从而提高精度。

这种人工智能与符号执行的结合称为 RedSoundSE，在第 5 节中定义，之后会集成到最终分析 RedSoundRSE 中。

4 SoundSE：可靠的符号执行

现在，我们定义一种名为 SoundSE 的符号执行，它不仅是 SoundRSE 的基础，也是 RedSoundSE（SoundSE 与抽象域的结合）的基础。

符号执行状态。 符号执行的核心原则是将程序变量映射为由符号值 组成的表达式，这些符号值表示程序变量的初始值。我们令 $\mathbb{V}=\{x,y,...\}$ 表示符号值的集合，并为清晰起见，记 x 为与程序变量 x 相关联的符号值（不要与具体值混淆）。符号存储 是一个从程序变量映射到符号表达式 的函数，其集合记为 E；也就是说，这些表达式与编程语言表达式的定义方式相同，只是使用符号值而非程序变量。我们将符号存储的集合记作 $\mathbb{M}=\mathcal{P}(\mathbb{X}\to\mathbb{E})$，并将 $[\mathtt{x}\rightsquigarrow\langle x\rangle,...]$ 记作一个显式给定的符号存储。为了恰当地关联符号存储与具体存储，我们需要建立符号值与具体值之间的关系。为此，我们将赋值定义为一个函数 $\nu:\mathbb{V}\longrightarrow\mathbb{V}$。此外，给定一个符号表达式 ε，我们令 $[\![\varepsilon]\!]$ 为一个偏函数，它将赋值 ν 映射为如下表达式求值所得的值：将 e 中的每个符号值 x 替换为 $\nu(x)$。现在，我们可以表示符号存储的具体化：

定义2（符号存储具体化）。符号存储具体化$\gamma_{\mathbb{M}}:\mathbb{M}\longrightarrow\mathcal{P}(\mathbb{M}\times(\mathbb{V}\to\mathbb{V}))$将符号存储映射为由存储和能够实现该存储的赋值构成的集合，即 $\gamma_{\mathbb{M}}(\rho)=\{(\mu,\nu)\mid\forall\mathtt{x}\in\mathbb{X}\ \mu(\mathtt{x})=[\![\rho(\mathtt{x})]\!](\nu)\}$。

要精确刻画执行路径的结果，符号存储过于抽象。因此，符号执行还会使用符号表达式来约束存储，该表达式称为符号路径，用于记录路径执行过程中遇到的条件。符号精确存储是一个二元组 $\kappa=(\rho,\pi)$，其中 $\rho\in\mathbb{M}$，且 π 为符号路径。我们用 K 表示符号精确存储的集合。其含义定义如下：

定义 3（符号精确存储具体化）。**符号精确存储具体化 $\gamma_{\mathbb{K}}:\mathbb{K}\longrightarrow\mathcal{P}(\mathbb{M}\times(\mathbb{V}\to\mathbb{V}))$ 定义为 $\gamma_{\mathbb{K}}(\rho,\pi)=\{(\mu,\nu)\in\gamma_{\mathbb{M}}(\rho)\mid[\![\pi]\!](\nu)=\mathtt{tt}\}$**。

$$\text{S-ASSIGN}\ \frac{(\mathbf{e},\rho)\vdash_s \varepsilon}{(\mathbf{x}:=\mathbf{e},(\rho,\pi))\rightharpoonup_s(\texttt{skip},(\rho[\mathbf{x}\rightsquigarrow\langle\varepsilon\rangle],\pi))}$$

$$\text{S-IF-T}\ \frac{(\mathbf{b},\rho)\vdash_s \beta \qquad \pi'\triangleq\pi\wedge\beta \qquad \mathbf{may}(\pi')}{(\texttt{if}\ \mathbf{b}\ \texttt{then}\ \mathbf{c}_0\ \texttt{else}\ \mathbf{c}_1,(\rho,\pi))\rightharpoonup_s(\mathbf{c}_0,(\rho,\pi))}$$

$$\text{S-LOOP-T}\ \frac{(\mathbf{b},\rho)\vdash_s \beta \qquad \pi'\triangleq\pi\wedge\beta \qquad \mathbf{may}(\pi')}{(\texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c},(\rho,\pi))\rightharpoonup_s(\mathbf{c};\ \texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c},(\rho,\pi))}$$

$$\text{S-LOOP-F}\ \frac{(\mathbf{b},\rho)\vdash_s \beta \qquad \pi'\triangleq\pi\wedge\neg\beta \qquad \mathbf{may}(\pi')}{(\texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c},(\rho,\pi))\rightharpoonup_s(\texttt{skip},(\rho,\pi))}$$

图 3. 符号执行步关系：若干选定规则

例 *1*（符号精确存储）。我们考虑程序2(a)。符号执行需要覆盖对应于条件语句各个分支的两条路径，即取决于 $priv$ 的符号。因此，符号执行应产生精确存储 $(\rho_0,\ priv > 0)$ 和 $(\rho_1,\ priv \le 0)$，其中$\rho_0=\rho_1=[\mathrm{y}\rightsquigarrow\langle 5\rangle,\ \mathrm{priv}\rightsquigarrow\langle priv\rangle]$。

符号执行步。 符号执行算法的核心部分是步关系，它紧密遵循程序的小步语义。我们通过状态之间的转换关系 $\rightharpoonup_s$ 来定义它，其中的状态由程序命令和符号精确存储构成。在写出分析 $\rightharpoonup_s$之前，我们需要先给出一些定义。

首先，我们定义符号存储中表达式或条件的符号求值，它会生成一个符号表达式。我们将 $(\mathbf{e}\ \rho)\vdash_s \varepsilon$ 记为在符号存储 ρ中将$\mathbf{e}$求值为符号表达式 ε。通常，这一步求值归结为将$\mathbf{e}$中的变量替换为它们在 ρ中所映射到的符号表达式，并可能进行一些简化。

其次，我们定义符号路径的保守可满足性测试。这一步通常由 SMT 求解器等外部工具执行，因此我们不在此详述其内部机制。需要注意的是，该测试可能会保守地返回“符号路径可能可满足”这一结果。当 π 可能可满足时，我们将其记为**可能**(π)。

现在我们来看图3中的规则。规则 S-ASSIGN 只需使用一个新的符号表达式更新所赋值变量对应的符号存储。在规则 S-IF-T中，如果守卫的求值结果 β 可满足，就会进入真分支，并将 β 添加到符号路径中。最后，对于循环，规则 S-LOOP-T 和 S-LOOP-F 遵循与规则S-IF-T 类似的原则。我们将执行步的可靠性形式化如下：

定理 *1*（单个符号执行步的可靠性）。设 $(c\ \mu)$ 和 $(c',\mu')\in S$ 是两个状态，满足 $(c\ \mu)\rightarrow(c',\mu')$，$\kappa\in K$ 是一个符号精确存储，并且ν 是一个赋值，满足 $(\mu,\nu)\in\gamma_{\mathbb{K}}(\kappa)$。那么，存在一个符号精确存储$\kappa'$，使得 $(\mu',\nu)\in\gamma_{\mathbb{K}}(\kappa')$ 且 $(c\ \kappa)\rightharpoonup_s(c',\kappa')$。

可靠的深度有界符号执行。 显然，对图3中定义的符号执行步关系进行穷举应用将无法终止。因此，常见的符号执行工具通常会在达到某种执行长度界限时中止探索

$$\text{S-NEXT}\ \frac{(\mathbf{c},\kappa)\rightharpoonup_s(\mathbf{c}',\kappa') \qquad \mathfrak{step}(\mathbf{c},\mathbf{c}',w)=(\mathbf{tt},w')}{(\mathbf{c},\kappa,w,b)\rightharpoonup_s(\mathbf{c}',\kappa',w',b)}$$

$$\text{S-APPROX-MANY}\ \frac{(\mathbf{c},\kappa)\rightharpoonup_s(\mathbf{c}',\kappa') \qquad \mathfrak{step}(\mathbf{c},\mathbf{c}',w)=(\mathbf{ff},w') \qquad \rho''=\mathfrak{modif}(\rho,\mathbf{c})}{(\mathbf{c},(\rho,\pi),w,b)\rightharpoonup_s(\texttt{skip},(\rho'',\pi),w',\mathbf{ff})}$$

图 4. SoundSE：可靠的有界符号执行步关系

。这种做法显然不可靠，因为更长的执行会被直接忽略。另一种方法是，对达到界限时可能可达的精确存储集合进行过近似。我们将在此对这种方法进行形式化。

本质上，符号状态需要增加两项信息：一个布尔型的所谓精度标志，用于说明符号执行是否因界限耗尽而进行了过度近似；以及一个称为计数器的界限控制字段。我们将计数器的集合定义为 W，其中包含一个特殊元素 $w_0 \in \mathsf{W}$，表示界限控制方面的初始计数器状态。为了对计数器进行操作，我们需要一个函数 $\mathfrak{step}$，其输入为两个命令$\mathbf{c}$、$\mathbf{c}'$以及一个计数器 w。该函数生成形式为 (b, w') 的结果，其中 b 是一个布尔值，w' 是下一个计数器。当且仅当从$\mathbf{c}$到$\mathbf{c}'$ 的一步操作能够在不耗尽迭代界限的情况下完成时，且新计数器为 w'时，值 b 为 $\mathbf{tt}$ 。如果 b为$\mathbf{ff}$，则表示已达到迭代上界，必须对该状态进行过度近似。

为执行过度近似，需要一个函数 $\mathfrak{modif}$ 。该函数接收一个符号存储和一条命令，并返回一个新的符号存储 ρ'，满足以下条件：– ρ' 将$\mathbf{c}$ 中被认为“已修改”的每个程序变量（通过集合的可靠过度近似来判定）映射到一个全新的符号值；– ρ' 将所有其他程序变量映射到其在原始存储中的对应值。

例 *2*（循环迭代界限）。限制符号执行最常见的方法，是将每个循环的迭代次数限制为预先定义的数值 k。此时，W由整数栈组成，w_0 为空栈，而 $\mathfrak{step}$ 在进入新循环时将零压入栈顶，在退出循环时弹出栈顶的值。更重要的是，当 $n \leq k$并进入下一次迭代时（规则 S-LOOP-T），它会递增栈顶的值 n；另一方面，当 $n > k$时，它会弹出 n 并返回$\mathbf{ff}$ 精度标志。

为确保终止，W 和 $\mathfrak{step}$ 应满足以下良基性性质：对于任意无限命令序列 $(\mathbf{c}_i)_i$，由 $\mathfrak{step}$ $(\mathbf{c}_i, \mathbf{c}_{i+1}, w_i) = (\mathbf{tt}\ w_{i+1})$ 定义的无限序列 $(w_i)_i$应当是平稳的，这一点在此处假设成立。

基于这些定义，深度有界符号执行 由定义在四元组上的转换关系给出，这些四元组由一个命令、一个符号状态、W 中的一个元素和一个布尔值组成，并统称为符号状态。我们为该关系重载记号 $\rightharpoonup_s$，其定义基于此前定义的 $\rightharpoonup_s$。规则见图 4：

– 规则 S-NEXT 执行符号执行的一个原子步骤，该步骤不需要过度近似；函数 $\mathfrak{step}$ 返回精度标志 b 和一个新的计数器；

– 规则 S-APPROX-MANY 执行全局近似步骤；实际上，当 $\mathfrak{step}$返回$\mathbf{ff}$时，函数 $\mathfrak{modif}$ 会应用于符号状态，以对

执行c任意数量步骤所产生的影响进行过近似；随着新的计数器状态，ff 精度也向前传播。

在良基性假设下，从任意初始符号状态出发，对可用的符号执行规则进行穷尽迭代将会终止，并产生有限多个符号状态。为表述该算法的可靠性，我们需要考虑函数 $\mathfrak{modif}$ 所创建的符号值，这意味着也需要扩展赋值。为此，我们用 $\nu \preceq \nu'$ 表示赋值 ν 的域包含于赋值 ν' 的域中，且 ν 和 ν' 在其域的交集上取值一致。现在我们得到以下可靠性陈述：

定理 *2*（任意序列单步符号执行的可靠性）。设 $(c\ \mu) \in S$ 为一个状态，μ' 为一个存储，且满足 $(c\ \mu) \to^* (skip\ \mu')$。设 $\kappa \in K$ 为一个符号精确存储，ν 为一个赋值，且满足 $(\mu,\nu) \in \gamma_{\mathbb{K}}(\kappa)$。设 $w \in W$ 为一个计数器。则存在一个符号精确存储 κ'、一个赋值 ν' 和一个计数器 $w' \in W$，使得 $\nu \preceq \nu'$、$(\mu',\nu') \in \gamma_{\mathbb{K}}(\kappa')$，且 $(c\ \kappa,w,b) \rightharpoonup_s^* (skip\ \kappa',w',b')$。

该定理可由定理 1（$\mathfrak{step}$ 返回tt的步骤）以及针对命令c 的全局归纳法证明，其中规则 S-APPROX-MANY 适用。

例 *3*（符号执行）。对于程序2(a)，符号执行会返回例 1 中所示的符号存储。我们采用例 2 中的界限，并考察程序2(b)。随后，符号执行会生成符号存储 $[\mathtt{z} \rightsquigarrow \langle z \rangle,\ \mathtt{i} \rightsquigarrow \langle i' \rangle,\ \mathtt{priv} \rightsquigarrow \langle priv' \rangle]$，其精度标志为ff，其中 i'、$priv'$ 是规则 S-APPROX-MANY生成的新符号值。

有界反驳。符号执行的一项非常理想的特性，是能够在给定界限内生成反例。这项特性源于有界反驳结果：当符号执行生成一个最终状态，且其最终精度标志为tt，并且符号路径可满足时，就能找到与之匹配的具体执行。SMT 求解器可以根据最终状态计算出一个反驳模型。

定理 *3*（有界反驳）。设 c 是一个命令，$\kappa, \kappa' \in K$ 是两个精确存储，$w, w' \in W$，且 $(c\ \kappa, w, tt) \rightharpoonup_s^* (skip\ \kappa', w', tt)$。那么，对于所有 $(\mu',\nu') \in \gamma_{\mathbb{K}}(\kappa')$，都存在 $(\mu,\nu) \in \gamma_{\mathbb{K}}(\kappa)$，使得 $(c\ \mu) \to^* (skip\ \mu')$。

这一结果源于规则 S-APPROX-MANY 在符号执行过程中从未被应用这一事实，以及对 S-NEXT 步骤序列进行归纳。

例 *4*（界限内的符号执行完备性）。我们考虑例 3 中讨论的情形。使用例 2 中的界限处理，程序2(a) 得出的结果是完备的，而程序2(b) 得出的结果则会生成某个精度标志为ff的最终符号状态，因此定理 3 不适用于该状态。

5 RedSoundSE：可靠符号执行与抽象状态的结合

现在，我们扩展 SoundSE，使其能够利用抽象解释推断出的属性。这种结合了符号执行的方法称为 RedSoundSE，名称体现了 SoundSE 与基于抽象解释的分析之间的约简积。

$$\text{A-ASSIGN}\ \frac{a' \triangleq \mathfrak{assign}_{\mathbf{x},\mathbf{e}}(a)}{(\mathbf{x}:=\mathbf{e},a)\rightharpoonup_{\mathbb{A}}(\texttt{skip},a')} \qquad \text{A-IF-T}\ \frac{a' \triangleq \mathfrak{guard}_{\mathbf{b}}(a) \qquad a' \neq \bot}{(\texttt{if}\ \mathbf{b}\ \texttt{then}\ \mathbf{c}_0\ \texttt{else}\ \mathbf{c}_1,a)\rightharpoonup_{\mathbb{A}}(\mathbf{c}_0,a')}$$

(a) Abstract execution step selected rules

$$\text{S-A-NEXT}\ \frac{(\mathbf{c},\kappa,w,b)\rightharpoonup_s(\mathbf{c}',\kappa',w',b) \qquad \mathfrak{step}(\mathbf{c},\mathbf{c}',w)=(\mathbf{tt},w') \qquad (\mathbf{c},a)\rightharpoonup_{\mathbb{A}}(\mathbf{c}',a') \qquad (\kappa'',a'') \triangleq \mathfrak{reduction}(\kappa',a')}{(\mathbf{c},\kappa,a,w,b)\rightharpoonup_{s\times\mathbb{A}}(\mathbf{c}',\kappa'',a'',w',b)}$$

$$\text{S-A-APPROX-MANY}\ \frac{(\mathbf{c},\kappa)\rightharpoonup_s(\mathbf{c}',\kappa') \qquad \mathfrak{step}(\mathbf{c},\mathbf{c}',w)=(\mathbf{ff},w') \qquad \kappa''=\mathfrak{modif}(\kappa,\mathbf{c}) \qquad a'=[\![\mathbf{c}]\!]^\sharp_{\mathbb{A}}(a) \qquad (\kappa''',a''') \triangleq \mathfrak{reduction}(\kappa'',a')}{(\mathbf{c},(\kappa,a),w,b)\rightharpoonup_{s\times\mathbb{A}}(\texttt{skip},(\kappa''',a'''),w',\mathbf{ff})}$$

(b) Product of symbolic execution and static analysis

图 5. 抽象执行步骤及其与符号执行的结合

存储的抽象与静态分析。 下文中，我们假设已固定一个描述存储集合的抽象域 [14] $\mathbb{A}$，以及一个具体化函数$\gamma_{\mathbb{A}}: \mathbb{A} \longrightarrow \mathcal{P}(\mathbb{M})$。我们假设存在一个元素 $\bot \in \mathbb{A}$，使得 $\gamma_{\mathbb{A}}(\bot)=\emptyset$。此外，对于基本操作，我们要求以下两个可靠的抽象后置条件函数。函数 $[\![\ ]\!]$ 将重载，以便在存储中用变量 $\mathbf{x}$ 对应的值替换变量 $\mathbf{x}$，即 $\mu(\mathbf{x})$:

– 抽象赋值 $\mathfrak{assign}_{\mathbf{x},\ \mathbf{e}}:\mathbb{A}\longrightarrow\mathbb{A}$ 由变量 $\mathbf{x}$ 和表达式 $\mathbf{e}$参数化，并满足 $\forall a\in\mathbb{A}$，$\{\mu[\mathbf{x}\mapsto[\![\mathbf{e}]\!](\mu)]\,|\,\mu\in\gamma_{\mathbb{A}}(a)\}\subseteq\gamma_{\mathbb{A}}(\mathfrak{assign}_{\mathbf{x},\ \mathbf{e}}(a))$.

–抽象条件 $\mathfrak{guard}_{\mathbf{b}}:\mathbb{A}\longrightarrow\mathbb{A}$ 由布尔表达式$\mathbf{b}$参数化，并满足 $\forall a\in\mathbb{A}$，$\{\mu\in\gamma_{\mathbb{A}}(a)\,|\,[\![\mathbf{b}]\!](\mu)=\mathbf{tt}\}\subseteq\gamma_{\mathbb{A}}(\mathfrak{guard}_{\mathbf{b}}(a))$.

基于这些操作，定义可靠的抽象执行步骤关系$\rightharpoonup_{\mathbb{A}}$ 十分简单。我们在图 5(a) 中展示了两条规则。这些规则与$\rightarrow$ （第 2 节）中的规则相对应，并且相对于它们是可靠的。下文假设 $\mathbb{A}$ 是分析的一个参数。它可以由任意数值抽象构成，例如区间抽象域 [14] 或凸多面体域 [16]。此外，应用标准扩宽技术 [14] 可以定义一个静态分析函数 $[\![\mathbf{c}]\!]^\sharp_{\mathbb{A}}:\mathbb{A}\longrightarrow\mathbb{A}$，该函数是可靠的，即对于所有命令 $\mathbf{c}$ 和所有抽象状态 a, $\{\mu'\in\mathbb{M}\,|\,\exists\mu\in\gamma_{\mathbb{A}}(a),\ (\mathbf{c}\ \mu)\rightharpoonup_s^*(\text{空操作}\ \mu')\}\subseteq\gamma_{\mathbb{A}}([\![\mathbf{c}]\!]^\sharp_{\mathbb{A}}(a))$

符号精确存储与抽象状态的约简积。 约简积 [15]旨在精确表示由不同抽象域表达的约束的合取。我们将精确积存储定义为一对 $(\kappa,a)\in\mathbb{K}\times\mathbb{A}$。在我们的情形中，定义需要稍作调整，因为符号执行与抽象域 $\mathbb{A}$ 抽象的对象并不完全相同:

定义 4（积域）。积抽象域由集合 $\mathbb{K}\times\mathbb{A}$ 以及具体化函数 $\underline{\gamma_{\mathbb{K}\times\mathbb{A}}}:\mathbb{K}\times\mathbb{A}\longrightarrow\mathcal{P}(\mathbb{M}\times(\mathbb{V}\to\mathbb{V}))$ 构成，其定义如下： $\underline{\gamma_{\mathbb{K}\times\mathbb{A}}}:(\kappa,a)\longmapsto\{(\mu,\nu)\in\gamma_{\mathbb{M}}(\kappa)\,|\,\mu\in\gamma_{\mathbb{A}}(a)\}$

在精确积存储 (κ,a) 中，目标是通过在 κ 和 a 之间交换信息来提高精度。这通过一个**归约**函数实现，该函数用具体化结果相同、但支持更精确分析操作的另一个抽象元素重写原抽象元素。这意味着 $(\gamma_{\mathbb{K}\times\mathbb{A}}\circ\mathfrak{reduction})(\kappa,a) = \gamma_{\mathbb{K}\times\mathbb{A}}(\kappa,a)$。这要求抽象域 A 支持一个函数 $\mathfrak{constr}$，将抽象状态 a 映射为关于程序变量且由 a 蕴含的逻辑公式，即如果 $\mu\in\gamma_{\mathbb{A}}(a)$，那么 μ 满足公式 $\mathfrak{constr}(a)$。某些抽象域——特别是区间和抽象多面体——使用基于约束合取的内部表示，在这种情况下，$\mathfrak{constr}$ 是显然的。然后，$\mathfrak{reduction}:\mathbb{K}\times\mathbb{A}\longrightarrow\mathbb{K}\times\mathbb{A}$ 定义为：

$$\mathfrak{reduction}((\rho,\pi),a)\triangleq((\rho,\pi'),a)\quad\text{where}\quad\pi'\triangleq\pi\wedge\mathfrak{constr}(a)[\vec{\mathtt{x}}\mapsto\rho(\vec{\mathtt{x}})]$$

请注意，在上述定义中，$[\vec{}\mathtt{x}\mapsto\rho(\vec{}\mathtt{x})]$ 表示将 $\mathfrak{constr}(a)$中出现的每个程序变量替换为其在 ρ中的定义；这一步源于以下事实：a 约束程序变量，而 π 约束赋值。这个通用归约函数可以细化为一个更精确的函数，在该函数中，生成的符号路径会被简化，甚至可能简化为**ff** 公式。此外，这种归约只会修改符号路径 π，但也可以定义一种同时重写抽象状态 a的归约操作。

约化积符号执行。 乘积分析（即 RedSoundSE）采用对图 4 中符号执行函数的扩展形式。新状态仍为 4 元组，但符号精确存储分量 κ 现在替换为精确积存储（κ，a）。这些状态之间的转移关系 $\rightharpoonup_{s\times\mathbb{A}}$ 由图 5(b) 所示的两条规则构成，它们扩展了图 4 中的规则。在规则 S-A-MANY 中（当达到探索界限时应用），除 $\mathfrak{modif}$之外，还会对抽象状态计算循环，然后应用归约函数。

在这两种情况下，都可以应用可靠的 $\mathfrak{reduction}$ 运算符。实践中，为了提高效率，可以采用惰性方式计算并应用该运算符，也就是说，只在特定步骤（通常是 S-A-MANY 和分支命令）中才进行计算和应用。

示例 *5*（积分析）。对于程序 2(d)，假设 $\mathsf{i}<10$，那么退出循环时，区间抽象状态将包含两个约束 $a=\{\mathsf{i}=10;\ \mathsf{priv}\geq 2\}$。假设存在符号精确存储 $\kappa=(\rho,\ \pi)$，且满足 $\rho=[\mathsf{i}\rightarrow i;\ \mathsf{priv}\rightarrow priv]$，则可以将抽象约束拟合到符号路径 π'，如下所示：$\pi'\triangleq\ \pi\ \wedge\ i=10\ \wedge\ priv\geq 2$。更详细的执行轨迹见附录 A。

可靠性与反驳性质。上一段定义的 RedSoundSE 分析满足与标准符号执行相同的可靠性（定理 2）和反驳性质（定理 3），因此不再重复给出这些定理。

6 SoundRSE：可靠的关系符号执行

如第 3 节所述，非干扰等安全属性需要对执行轨迹对进行推理，因此我们现在建立一种可靠的关系符号执行技术，用于构造执行对。我们将此分析称为 SoundRSE。

假设。为简化记号，我们假设本节及下一节中的界限计数器步进函数 $\mathfrak{step}$ 仅影响循环，即当$\mathbf{c}$ 不是循环命令时，$\mathfrak{step}(\mathbf{c},\mathbf{c}',w)=w$。此外，在以下定义中，我们不包含与数值抽象状态的乘积（如第 5 节所示）。由于可以无缝地将其加入，我们在此省略，以简化形式化表述。

精确关系存储。我们首先定义关系表达式、关系存储和精确关系存储的概念。

定义 5（关系存储与精确关系存储）。关系符号表达式 是由以下文法定义的一个元素：$\tilde{\varepsilon} ::= \langle\varepsilon\rangle \mid \langle\varepsilon \mid \varepsilon\rangle$ 其中 ε 遍历符号表达式集合 $\mathbb{E}$。我们用$\mathbb{E}_2$ 表示关系符号表达式的集合。关系符号存储 $\tilde{\rho}$ 是一个从变量到关系符号表达式的函数。我们用$\mathbb{M}_2 = \mathbb{X} \to \mathbb{E}_2$ 表示它们构成的集合。最后，精确关系存储 $\tilde{\kappa}$ 是一个对$(\tilde{\rho},\pi) \in \mathbb{K}_2$。

在定义 $\mathbf{M}_2$ 和 $\mathbf{K}_2$, 的具体化之前，我们需要介绍两个操作：

– 投影 Π_0,Π_1 将关系符号存储映射为符号存储。它们按逐点方式定义如下：若 $\tilde{\rho}(\mathbf{x})=\langle\varepsilon\rangle$，则 $\Pi_0(\tilde{\rho})(\mathbf{x})=\Pi_1(\tilde{\rho})(\mathbf{x})=\varepsilon$；若 $\tilde{\rho}(\mathbf{x})=\langle\varepsilon_0 \mid \varepsilon_1\rangle$，则 $\Pi_0(\tilde{\rho})(\mathbf{x})=\varepsilon_0$ 且 $\Pi_1(\tilde{\rho})(\mathbf{x})=\varepsilon_1$。我们扩展 Π_0,Π_1 这一记法，也将其应用于双重符号表达式：$\Pi_0(\langle\varepsilon\rangle)=\Pi_1(\langle\varepsilon\rangle)=\varepsilon$；若 $\tilde{\varepsilon}=\langle\varepsilon_0 \mid \varepsilon_1\rangle$，则 $\Pi_0(\tilde{\varepsilon})=\varepsilon_0$ 且 $\Pi_1(\tilde{\varepsilon})=\varepsilon_1$。

– 两个符号存储 ρ_0 和 ρ_1 的配对 $(\!|\rho_0 \mid \rho_1|\!)$ 是一个关系符号存储，其定义如下：对于所有变量 $\mathbf{x}$,

$$(\!|\rho_0 \mid \rho_1|\!)(\mathbf{x}) = \begin{cases} \langle\varepsilon\rangle & \text{if } \rho_0(\mathbf{x}) \text{ and } \rho_1(\mathbf{x}) \text{ are provably equal to } \varepsilon \in \mathbb{E} \\ \langle\rho_0(\mathbf{x}) \mid \rho_1(\mathbf{x})\rangle & \text{otherwise} \end{cases}$$

其中，“可证明相等”的概念可以简化为符号表达式在语法上的相等，也可以涉及外部证明工具。

现在可以定义具体化函数 s:

定义 6（具体化函数）。关系存储的具体化$\gamma_{\mathbb{M}_2}$ 以及精确关系存储 $\gamma_{\mathbb{K}_2}$ 的具体化定义如下：

$$\begin{aligned} \gamma_{\overline{\mathbb{M}}_2} : \overline{\mathbb{M}}_2 &\longrightarrow \mathcal{P}(\mathbb{M}\times\mathbb{M}\times(\overline{\mathbb{V}}\to\mathbb{V})) \\ \tilde{\rho} &\longmapsto \{(\mu_0,\mu_1,\nu) \mid \forall \mathbf{x}\in\mathbb{X}, \forall i\in\{0,1\}, \mu_i(\mathbf{x}) = [\![\Pi_i(\tilde{\rho})(\mathbf{x})]\!](\nu)\} \\ \gamma_{\mathbb{K}_2} : \mathbb{K}_2 &\longrightarrow \mathcal{P}(\mathbb{M}\times\mathbb{M}\times(\overline{\mathbb{V}}\to\mathbb{V})) \\ (\tilde{\rho},\pi) &\longmapsto \{(\mu_0,\mu_1,\nu)\in\gamma_{\overline{\mathbb{M}}_2}(\tilde{\rho}) \mid [\![\pi]\!](\nu) = \boldsymbol{tt}\}. \end{aligned}$$

例 *6.* 我们考虑程序2(a)（SoundSE 已在例 1 中讨论）。为了涵盖低变量 y 取相同值、但高变量 priv 可能取不同值的执行对，关系符号执行应涵盖四对路径。这四条路径具有相同的关系符号存储[priv $\rightsquigarrow\langle \boldsymbol{priv}_0 \mid \boldsymbol{priv}_1\rangle$，$\mathrm{y}\rightsquigarrow\langle 5\rangle$]，且仅在符号路径部分上有所不同。例如，当第一个执行采取条件的真分支，而第二个执行采取假分支时，符号路径为 $\boldsymbol{priv}_0 > 0 \wedge \boldsymbol{priv}_1 \leq 0$。

$$\text{SR-EXIT}\ \frac{}{(\texttt{skip} \bowtie \texttt{skip}, (\tilde{\rho}, \pi), w, b) \rightharpoonup_{sr} (\texttt{skip}, (\tilde{\rho}, \pi), w, b)}$$

$$\text{SR-COMP-R}\ \frac{(\mathbf{c}_1, (\Pi_1(\tilde{\rho}), \pi), w, b) \rightharpoonup_{s} (\mathbf{c}_1', (\rho_1', \pi'), w', b')}{(\texttt{skip} \bowtie \mathbf{c}_1, (\tilde{\rho}, \pi), w, b) \rightharpoonup_{sr} (\texttt{skip} \bowtie \mathbf{c}_1', ((\!|\Pi_0(\tilde{\rho}) \,|\, \rho_1'|\!), \pi'), w', b')}$$

$$\text{SR-COMP-L}\ \frac{(\mathbf{c}_0, (\Pi_0(\tilde{\rho}), \pi), w, b) \rightharpoonup_{s} (\mathbf{c}_0', (\rho_0', \pi'), w', b')}{(\mathbf{c}_0 \bowtie \mathbf{c}_1, (\tilde{\rho}, \pi), w, b) \rightharpoonup_{sr} (\mathbf{c}_0' \bowtie \mathbf{c}_1, ((\!|\rho_0' \,|\, \Pi_1(\tilde{\rho})|\!), \pi'), w', b')}$$

$$\text{SR-IF-TF}\ \frac{(\mathbf{b}, \tilde{\rho}) \vdash_{\mathrm{sr}} \tilde{\beta} \qquad \pi' = \pi \wedge \Pi_0(\tilde{\beta}) \wedge \neg \Pi_1(\tilde{\beta}) \qquad \mathbf{may}(\pi')}{(\texttt{if}\ \mathbf{b}\ \texttt{then}\ \mathbf{c}_0\ \texttt{else}\ \mathbf{c}_1, (\tilde{\rho}, \pi), w, b) \rightharpoonup_{sr} (\mathbf{c}_0 \bowtie \mathbf{c}_1, (\tilde{\rho}, \pi'), w, b)}$$

$$\text{SR-APPROX-MANY}\ \frac{\begin{array}{c}\mathfrak{step}(\texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c}, (\mathbf{c};\ \texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c}), w) = (\mathbf{ff}, w') \\ \tilde{\rho}'' = \mathfrak{modif}(\tilde{\rho}, \mathbf{c}) \qquad (\mathbf{b}, \tilde{\rho}'') \vdash_{\mathrm{sr}} \langle \beta_0, \beta_1 \rangle \qquad \pi' \triangleq \pi \wedge \neg\beta_0 \wedge \neg\beta_1\end{array}}{(\texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c}, (\tilde{\rho}, \pi), w, b) \rightharpoonup_{sr} (\texttt{skip}, (\tilde{\rho}'', \pi'), w', \mathbf{ff})}$$

图 6. SoundRSE：关系符号执行步关系的若干选定规则。

关系符号执行算法。 由于 SoundRSE 旨在描述执行对，因此应考虑两个执行遵循不同控制流路径的情况。因此，关系符号状态可以由一条命令组成（当两个执行遵循相同路径时），也可以由两条命令组成（当它们分歧时）。我们分别将这两类状态记为（$\mathbf{c}\ \tilde{\kappa},\ w,\ b$）和（（$\mathbf{c}_0 \bowtie \mathbf{c}_1$）；$\mathbf{c}_2,\ \tilde{\kappa},\ w,\ b$）；在后一种状态中，$\mathbf{c}_0$（分别为 $\mathbf{c}_1$）表示第一个（分别为第二个）执行的控制状态，它们之后会在$\mathbf{c}_2$处汇合。组件 w 和 b 的含义与第 4 节中相同。初始状态属于前一种类型。

我们用 $\rightharpoonup_{sr}$ 表示关系符号执行步关系。图 6 展示了其中一些具有代表性的规则。规则 SR-APPROX-MANY 描述了一种为确保终止而执行近似的情形，并将 $\mathfrak{modif}$ 直接扩展到关系符号状态。

可靠性与反驳性质。SoundRSE 继承了与 SoundSE 类似的可靠性与反驳性质，如以下定理所示。

定理 *4*（可靠性）。设 $\tilde{\kappa} \in K2$、$w \in W$ 和 $b \in B$。令 $(\mu_0, \mu_1, \nu) \in \gamma_{\mathbb{K}_2}(\tilde{\kappa})$，并假设存储 μ_0'、μ_1' 满足 $(c\ \mu_0) \rightarrow^* (skip\ \mu_0')$ 且 $(c\ \mu_1) \rightarrow^* (skip\ \mu_1')$。那么，存在 $\tilde{\kappa}' \in K_2$，赋值 ν' 和计数器状态 $w' \in W$，使得 $\nu \preceq \nu'$、$(\mu_0', \mu_1', \nu') \in \gamma_{\mathbb{K}}(\tilde{\kappa}')$，且 $(c\ \tilde{\kappa}, w, b) \rightharpoonup_{sr}^* (skip\ \tilde{\kappa}', w', b')$。

定理 *5*（有界反驳）。设 c 是一个命令，$\tilde{\kappa}, \tilde{\kappa}' \in K_2$ 是两个精确存储，$w, w' \in W$，使得 $(c\ \kappa, w, tt) \rightharpoonup_{sr}^* (\text{空操作}\ \kappa', w', tt)$。那么，对于所有 $(\mu_0', \mu_1', \nu') \in \gamma_{\mathbb{K}_2}(\kappa')$，存在 $(\mu_0, \mu_1, \nu) \in \gamma_{\mathbb{K}_2}(\kappa)$，使得 $(c\ \mu_0) \rightarrow^* (\text{空操作}\ \mu_0')$ 且 $(c\ \mu_1) \rightarrow^* (\text{空操作}\ \mu_1')$。

SoundRSE 分析与非干扰。现在我们假设有一个程序（$\mathbf{c}\ L$），并展示如何应用 SoundRSE 分析来尝试证明非干扰。分析按以下步骤进行：

1. 构造初始存储 $\tilde{\rho}_0$，使得对于 $\mathbf{c}$ 中出现的所有变量 $\mathbf{x}$，$\tilde{\rho}_0(\mathbf{x}) = \langle x \rangle$（分别为 $\tilde{\rho}_0(\mathbf{x}) = \langle x_0 \,|\, x_1 \rangle$），若 $\mathbf{x} \in L$（分别为 $\mathbf{x} \notin L$），且其中 x 是一个新符号值（分别为 x_0，x_1 是新符号值）。

2. 从初始状态 (c,($\tilde{\rho}_0$,tt) w_0,tt) 出发，穷尽应用语义规则；我们用 $\mathcal{O}$ 表示带有其精度标志的最终精确关系存储集合：$\mathcal{O} \triangleq \{(\tilde{\kappa}, b) \mid \exists w \in \mathrm{W}, (\mathrm{c}, (\tilde{\rho}_0, \mathrm{tt})\ w_0, \mathrm{tt}) \rightarrow_{sr} (\text{空操作 } \tilde{\kappa}, w, b)\}$.

3. 尝试证明非干扰 对 $\mathcal{O}$ 中的每条符号路径，使用外部工具（例如 SMT 求解器）进行检查；更准确地说，给定 $((\tilde{\rho}, \pi)\ b) \in \mathcal{O}$,

– 如果 π 不可满足，则该路径不可行，可以忽略；–如果可以证明，对于所有变量 $\mathrm{x} \in L$，都存在唯一值，即 $\Pi_0(\tilde{\rho})(\mathrm{x}) = \Pi_1(\tilde{\rho})(\mathrm{x})$，那么该程序就是非干扰的；– 如果能找到一个赋值 ν，使得 $[\![\pi]\!](\nu) = \mathbf{tt}$（该路径可满足），并且存在一个变量 $\mathrm{x} \in L$，使得 $[\![\Pi_0(\tilde{\rho})(\mathrm{x})]\!](\nu) \neq [\![\Pi_1(\tilde{\rho})(\mathrm{x})]\!](\nu)$，且 $b = \mathbf{tt}$，那么 ν 会给出一个反例，从而证伪非干扰；– 最后，如果 $b = \mathbf{ff}$，且上述情况均未发生，则无法对该路径给出确定结论。

总而言之，分析器要么证明非干扰（当所有路径均不可满足或均满足非干扰时），要么提供一个能够反驳非干扰的赋值（当至少一条路径上可以找到这样的赋值时），要么无法得出结论。找到反例时，该反例实际上定义了一次真实攻击。

示例 7（非干扰）。对于程序2(a)，所有路径均低相等。对于程序 2(c)，只要展开界限设为任意严格正整数，分析就会计算出至少一条干扰路径；在这种情况下，即使使用基础的 SMT 求解器，也能合成出类似第 3 节所示的模型。最后，结合依赖分析抽象解释的关系符号执行与值抽象域（例如区间）的约简积，可以证明图 2(d) 中的程序满足非干扰（第 5 节）。

7 RedSoundRSE：SoundRSE 与依赖分析抽象解释的积

如第 3 节所述，对于图2(b)中的某些程序，使用传统的依赖分析进行分析，其精度可能高于有界符号执行（第 4 节）。本节将建立一种新颖的抽象的积形式，以利用这种精度提升。此积的概念具有通用性，无需固定某种特定的依赖抽象。我们将本节介绍的最终分析称为 RedSoundRSE。

依赖抽象与静态分析。 尽管依赖抽象可以采取多种形式，但它们都刻画了通过比较执行对而观察到的信息流。例如， [4] 使用安全级别格，抽象元素将每个级别映射到一组变量。当更高级别变量的输入值发生变化时，这些变量保持不变。其他研究则使用关系抽象域，其中“关系”意味着在执行对之间维护关系。因此，我们可以用存储对的抽象来刻画此类分析：

定义 7（依赖抽象与分析）。A依赖抽象由一个从安全级别映射到变量的抽象格$\mathbb{D}$ 以及一个具体化函数构成，

具体化函数

$$\gamma_{\mathbb{D}} : \mathbb{D} \longrightarrow \mathcal{P}(\mathbb{M} \times \mathbb{M})$$
$$d \longmapsto \{(\mu_0, \mu_1) \in \mathbb{M} \times \mathbb{M} \mid \mu_0 =_{d(\mathbb{L})} \mu_1\}$$

可靠的依赖分析由函数 $[\![c]\!]^\sharp\ D : D \to D$ 定义，使得对于所有 $d \in D$，$(\mu_0, \mu_1) \in \gamma_{\mathbb{D}}(d)$，$\{(\mu_0', \mu_1') \in M \times M \mid \forall i \in \{0,1\}, (c\ \mu_i) \to (\text{空操作}\ \mu_i')\} \subseteq \gamma_{\mathbb{D}} \circ [\![c]\!]^\sharp\ D(d)$。

例 8（基于标准依赖的抽象 [4]）。[4] 的抽象是定义 7 的一个实例。令 {L, H} 为安全级别集合。假设初始抽象状态 d 能够捕获某个程序（$\mathbf{c}\ L$）中低级相等的具体存储对。应用依赖分析后，如果最终依赖状态对每个初始低变量都有低级依赖，则该程序是非干扰的。

实践中，这类信息通过前向抽象解释计算得出：对表达式和条件使用语法依赖，并保守地假设条件可能向其守卫的任何操作产生（隐式）流。

我们注意到，定义 7 不仅适用于 [4] 这类依赖抽象。特别是，[22] 提出了一种语义补丁分析，它也可以通过使用关系抽象域关联执行对，应用于安全属性；这类分析使用的抽象也会像定义 7 中那样进行写入。下文中，我们假设已确定一种可靠的依赖分析。

符号执行与依赖分析的乘积。现在，我们将依赖分析与符号执行结合起来。对于大多数语句，图 6 中定义的 SoundRSE 规则不会引入精度损失。一个显著的例外是达到执行界限的情况，如规则 SR-APPROX-MANY 所示。因此，组合分析的原则是用另一条规则替换这条不精确的规则，利用依赖分析的结果来强化关系存储。首先，我们介绍两个操作，以可靠的方式将信息传入依赖抽象域并从中传出：

定义 8（信息转换与依赖抽象）。从符号到依赖的转换是一个函数 $\tau_{s \to \mathbb{D}} : \mathrm{M}_2 \to \mathrm{D}$，在以下意义下是可靠的：$\forall \tilde{\rho} \in \mathrm{M}2, \forall (\mu_0, \mu_1, \nu) \in \gamma_{\mathbb{M}_2}(\tilde{\rho}),\ (\mu_0, \mu_1) \in \gamma_{\mathbb{D}} \circ \tau_{s \to \mathbb{D}}(\tilde{\rho})$。依赖信息的提取是一个函数 $\lambda_{\mathbb{D} \to \mathbb{L}} : \mathrm{D} \to \mathcal{P}(\mathrm{X})$，在以下意义下是可靠的：$\forall d \in \mathrm{D}\ \forall (\mu_0, \mu_1) \in \gamma_{\mathbb{D}}(d), \mu_0 =_{\lambda_{\mathbb{D} \to \mathbb{L}}(d)} \mu_1$

直观地说，$\tau_{s \to \mathbb{D}}$ 应当计算出一个依赖抽象域元素，该元素表达的性质由其所应用的关系符号存储蕴含。在例 8 的设定中，实现这一点的一种直接方法是将 $\tilde{\rho}$ 映射到一个元素 d，该元素将 L 映射到集合：$\{\mathrm{x} \in \mathrm{X} \mid \textbf{可能}(\Pi_0(\tilde{\rho})(\mathrm{x}) = \Pi_1(\tilde{\rho})(\mathrm{x}))\}$

当 $\tilde{\rho}(\mathrm{x}) = \langle \varepsilon \rangle$ 时，这个等式显然成立；当 $\tilde{\rho}(\mathrm{x}) = \langle \varepsilon_0 \mid \varepsilon_1 \rangle$ 时，需要借助 SMT 求解器等外部工具来证明该等式 $\varepsilon_0 = \varepsilon_1$。类似地，函数 $\lambda_{\mathbb{D} \to \mathbb{L}}$ 会提取一组变量，这些变量被证明会被其参数保持为低安全级别。在例 8 的设置中，这相当于返回 $d(\mathrm{L})$。

现在我们介绍组合分析。符号执行步 SR-APPROX-MANY-DEP 如图 7 所示，并取代规则 SR-APPROX-MANY（图 6）。当

$$\text{SR-APPROX-MANY-DEP}\ \frac{\begin{array}{c}\mathfrak{step}(\texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c},(\mathbf{c};\ \texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c}),w)=(\mathbf{ff},w')\\ d=[\![\texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c}]\!]^{\sharp}_{\mathbb{D}}(\tau_{s\to\mathbb{D}}(\tilde{\rho}))\qquad \tilde{\rho}''=\mathfrak{modify}_{\mathbb{D}}(\tilde{\rho},\mathbf{c},\lambda_{\mathbb{D}\to\mathbb{L}}(d))\\ (\mathbf{b},\tilde{\rho}'')\vdash_{\mathrm{sr}}\langle\beta_0,\beta_1\rangle\qquad \pi'\triangleq\pi\wedge\neg\beta_0\wedge\neg\beta_1\end{array}}{(\texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c},(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr\times\mathbb{D}}(\texttt{skip},(\tilde{\rho}'',\pi'),w',\mathbf{ff})}$$

图 7. RedSoundRSE：符号执行近似及其与依赖信息的乘积。

循环语句达到执行界限时，它会根据将 $\tau_{s\to\mathbb{D}}$ 应用于关系符号存储所得的依赖状态，对整个循环进行依赖分析。然后，它应用 $\lambda_{\mathbb{D}\to\mathbb{L}}$，得出经依赖分析证明为低的变量集合。最后，它根据所确定的变量集合修改变量，从而计算出新的关系符号存储：

– 如果根据 $\lambda_{\mathbb{D}\to\mathbb{L}}$ 输出可证明变量 x 是低安全级的，$\mathfrak{modify}_{\mathbb{D}}$ 则合成一个新的符号值 $\mathfrak{x}_{\mathrm{new}}$ 并将其映射到 $\langle\mathfrak{x}_{\mathrm{new}}\rangle$；– 如果无法证明变量 x 是低安全级的，$\mathfrak{modify}_{\mathbb{D}}$ 则合成两个新的符号值$\mathfrak{x}_{\mathrm{new0}}$、$\mathfrak{x}_{\mathrm{new1}}$，并将 x 映射到 $\langle\mathfrak{x}_{\mathrm{new0}}\,|\,\mathfrak{x}_{\mathrm{new1}}\rangle$。

备注 *1*（约简积性质）。我们强调，规则 SR-APPROX-MANY-DEP在分析过程中可能会被多次应用，基本上每当分析循环语句时都会应用。由于抽象迭代，其应用次数通常远远多于程序中的循环命令数量。因此，我们的分析不能被视为固定的分析序列。这种分解方式（例如先运行依赖分析，再运行符号执行）的精度严格低于我们基于约简积的方法。

可靠性与反驳性质。假设依赖分析和转换操作是可靠的，则组合符号执行也是可靠的，因此定理 4 仍然成立。此外，定理 5 的反驳性质也成立。

示例 *9*（组合分析）。我们考虑程序2(b)。如第 3 节所述，循环语句可能执行无界次，因此关系符号执行会应用规则 SR-APPROX-MANY-DEP。通过 $\tau_{s\to\mathbb{D}}$ 计算得到的循环初始依赖抽象元素将 L 映射到 {i，z}，并将 H 映射到所有变量。循环的依赖分析返回相同的元素。因此，由 $\lambda_{\mathbb{D}\to\mathbb{L}}$ 返回的低安全级变量集合为 {i，z}，这使得我们能够计算出精确的关系符号存储，并成功验证该程序是非干扰的。

8 比较

本节将我们的分析方法彼此进行比较，并与 Assaf 等人 [4] 的依赖分析进行比较。为此，我们实现了所有分析方法的原型。我们的目标并非在大型代码库上评估这些分析方法，而是基于一些规模虽小、但对典型非干扰分析器而言颇具挑战性的程序，考察它们之间的差异。

```
如果(priv  > 0)
  i = 0;
else
  i = 0;
当(i  < 10){
  i += 1;
  priv += 5;
}
```
(a) 安全

```
i = 0; w = 2;
x = 100;
当(i  <x) {
  如果(x  <= 0)
    w =priv;
  i += 2;
  x += 1;
}
```
(b) 安全

```
i = 0;
当(i < 3) {
    y0 = y1;
    y1 = y2;
    y2 = priv;
    i += 1;
}
y1 = 0; y2 = 0;
```
(c) 不安全

```
i = 0;
当(i  < 100){
    如果(priv  > 0)
        y = 5;
    i += 1;
}
```
(d) 不安全

图 8. 展示分析器不同属性的程序。变量 priv 属于高安全级。

关系分析		𝔻	SoundRSE		RedSoundRSE (D)	
关系分析输入：		None	SoundSE	RedSoundSE	SoundSE	RedSoundSE
程序	安全?					
图 2(a)	Yes	✗误报 ✓安全		✓安全 (**I**,**P**)	✓安全	✓安全 (**I**,**P**)
图 2(b)	Yes	✓安全	✗误报	✓安全 (**P**)	✓安全	✓安全 (**I**,**P**)
图 2(d)	Yes	✗误报 ✗误报		✓安全 (**I**,**P**)	✗误报	✓安全 (**I**,**P**)
图8(a)	Yes	✗误报 ✗误报		✗误报	✓安全	✓安全 (**I**,**P**)
图8(b)	Yes	✗误报 ✗误报		✗误报	✗误报	✓安全 (**I**,**P**)
图2(c)	No	✓警报	✓反驳模型	✓反驳模型 ✓反驳模型 ✓反驳模型		
图8(c)	No	✓警报	✓反驳模型	✓反驳模型 ✓反驳模型 ✓反驳模型		
图8(d)	No	✓警报	✓警报	✓警报	✓警报	✓警报

表2. 分析组合的评估与比较。 𝔻 表示对 [4] 的依赖分析。符号 ✓（分别表示 ✗）表示语义正确（分别表示不正确）的分析结果，其形式可以是安全性证明、警报（可能是误报）或反驳模型。对于RedSoundSE列，当分析成功证明非干扰时，我们用 **I**（分别用 **P**）表示使用区间（分别表示使用多面体）域。

实现。 我们为本文提出的分析方法以及依赖分析、区间分析和凸多面体分析实现了原型。该原型使用 OCaml 编写，约有 4k 行代码；数值域采用 Apron 库 [29]，并使用 Z3 SMT 求解器 [21]。通过为 SoundSE 和 RedSoundSE 定义共享接口，RedSoundRSE 的实现可以参数化地使用这两者。我们提供了该实现的实现制品。

评估。 我们通过在一组复杂示例上进行评估，比较了三种采用不同单轨迹分析的关系技术。结果如表2所示。下文将非干扰程序与非非干扰程序分开讨论。对于后者，我们考察分析的反驳能力。

不同关系分析的验证能力比较。 图2中的程序已在第 3 节中说明，我们的原型也验证了这些结果，结果汇总于表2。

在程序 8(a) 中，第一个条件使依赖分析失效，因为它会将变量 i 视为高安全级。如果迭代上界低于 10，该程序也将无法通过 SoundRSE 的验证：在这种情况下，i 将被赋予一个新符号

值，因此会被判定为高安全级。相比之下，RedSoundRSE 可以确定循环中 i 的值不依赖于 priv。

程序 8(b) 更为复杂。要证明其满足非干扰，需要同时进行数值抽象和依赖抽象。分析将（保守地）判定循环中有三个变量被修改：x、i和w。即使变量i和x 都被修改，依赖分析仍能判定它们是低安全级。然而，由于w 依赖于x，且 x 的确切值未知，因此无法判定w 是低安全级。加入数值域后，就能轻松跟踪到 x 的值始终为正，这意味着if 语句永远不会执行。

不同关系分析的反驳能力比较。由于 SoundRSE 和 RedSoundRSE 只将循环展开有限次，因此对于某些不安全程序，可以找到反驳模型，而对于另一些程序则无法做到。请注意，要通过模型反驳一个程序，符号执行不能进行任何过近似；也就是说，分析发现违规时，精度标志必须为 false。因此，SoundRSE 对不安全程序的结果与依赖符号执行的不同组合的结果相似，如图 2所示。对于程序 2(c)，迭代一次即可找到一个赋值：$\nu(i_0)=\nu(i_1)=1$ 和 $\nu(priv_0)=0, \nu(priv_1)=1$。对于程序 8(c)，若将迭代界限设为 4 或更高，就能找到一个模型。赋值 ν 只需将变量 priv 映射到两个不同的值：$\nu(priv_0)\neq\nu(priv_1)$。对于程序 8(d)，只要用户设定的界限低于 100，执行就必须进行过近似，从而丧失反驳能力。

评估结论。 我们评估并相互比较了这些分析，也将其与最先进的依赖关系分析进行了比较 [4] ，评估对象是一组包含 8 个复杂示例的集合。结果表明，与依赖关系分析不同， [4], 这些分析继承了符号执行在给定界限内提供反驳模型的能力。此外，与所有其他参与比较的分析不同，使用 RedSoundSE 实例化的 RedSoundRSE 能够可靠地验证所有示例，汇总结果见表2。

局限性。 由于 RedSoundSE 可靠且自动化，因此必然无法实现完备性（根据赖斯定理 [30,3]）。作为补偿，我们提供了界限内的完备性。另一个较为微妙的局限是，数值抽象是在单次符号执行（RedSoundSE）层面应用的。这意味着这些抽象无法追踪执行之间的关系，只能处理局部约束。

9 相关工作

超属性非干扰最初由 Goguen 和 Meseguer 定义 [26],，随后也以去分类这一属性名称推广到更强大的攻击者模型。我们请读者参阅一篇关于截至 2005 年去分类策略的综述 [37] 。正如引言中所述，非干扰并非安全属性，而是一种安全超属性 [13],，亦称超安全性。文献中的多项研究表明，超安全性验证可以归约为安全属性验证 [7,20,39,13],

然而，这种归约在实践中并非总是高效的 [39]。我们的工作并未将非干扰归约为安全属性验证，而是采用关系分析。我们仅通过非干扰展示研究结果，但只要存在可靠的抽象域，这种方法就很容易推广到限制更宽松的去分类属性。

符号执行。 符号执行是一种静态分析技术，诞生于 20 世纪 70 年代 [9,31]，如今已应用于多种流行的测试工具，例如 KLEE [11]和美国国家航空航天局的 Symbolic PathFinder [35], 等。符号执行的首要目标和优势在于找出导致反例的路径，从而生成具体输入值来执行该路径。当检测到漏洞时，这对于调试并确认攻击的可行性尤为重要。

Alatawi 等人 [2] 利用人工智能提升以路径覆盖为目标的动态符号执行的精度。他们的方法首先使用人工智能分析程序，以捕获间接依赖关系，从而增强路径谓词。此外，他们的分析无法保持可靠性（也无法保持完备性）。与此同时，我们的方法在抽象域与符号执行之间持续交替，在一定界限内保持可靠性和完备性。最后，Alatawi 等人 [2]不分析非干扰等关系属性，只分析安全属性。

本文其余部分将重点讨论关系安全属性的静态分析技术：关于符号执行的更广泛讨论，请参阅一篇综述 [10]（截至 2011 年），以及一篇对截至 2013 [12] 时 SE 实践挑战的深入探讨。

关系符号执行。 为了将 SE 应用于非干扰等安全属性，Milushev 等人 [32] 提出了一种关系符号执行（RSE）形式，利用 KLEE 通过一种称为自组合 [7,20,39] 的技术分析非干扰，将程序 p 的关系属性归约为对 p 进行变换后所得程序的安全属性。最近，Daniel 等人优化了 RSE，使其适用于二进制代码，从而分析常数时间 [17] 和推测常数时间 [18,19] 等关系属性，并在真实世界的密码学库中发现了对这些属性的违反。所有这些方法都基于纯（关系）SE 静态技术，因此无法像我们的方法那样，在固定界限之外恢复可靠性。与 RedSoundRSE 最接近的工作是 RelSym [23]，它支持交互式反驳和可靠性。为了恢复可靠性，Chong 等人 [23] 提出在手动标注了循环不变式的程序上使用 RelSym。只有当不变式足够强时，反驳的精度才能得到保证，而工具本身无法判定这一点。在其他情况下，精度均无法保证。相比之下，我们的不变式由人工智能自动生成，反驳的精度始终能够在一个界限内得到保证，而该界限由我们的工具自动计算。

针对超属性的可靠静态分析 正如引言中所述，已有许多针对关系安全属性的可靠验证方法。关于这一主题，读者可参阅一篇出色的综述 [36]，其内容截至 2003年。2003年之后，研究者提出了若干针对类似非干扰的属性的可靠（半）静态验证方法，包括采用类型系统（例如 [6,24]）、混合类型（例如 [38]）、关系逻辑（例如 [1]）、模型检测（例如 [27,5]）以及纯人工智能方法 [4]。我们将重点介绍

基于人工智能的方法，因为它们与我们的工作最为接近。Giacobazzi 和 Mastroeni [25] 为攻击者对程序秘密的观察定义了抽象，并设计了基于人工智能、针对执行集合（而非关系执行）的可靠自动程序分析。Assaf 等人 [4] 首创了在人工智能框架内完整表达超属性的方法，其方式是定义一个直接近似目标超属性的伽罗瓦连接。我们将 Assaf 等人 [4]的抽象域与符号执行结合，以获得 RedSoundSE。请注意，由于 Assaf 等人 [4] 的框架依赖于不完备抽象，其分析既无法进行精确反驳，也无法提供反驳模型。据我们所知，此前尚无工作将抽象域与符号执行结合以实现可靠性。

10 结论

在本研究中，我们提出了一系列分析方法，如图1所示，将符号执行与人工智能相结合。这些分析方法是可靠且精确的，并能够在给定界限内合成反例。我们对这些分析方法、若干人工智能域以及一种用于验证非干扰的依赖分析进行了原型实现。表2总结了我们的结果，结果表明，对于一组具有挑战性的非干扰示例，我们的分析方法优于依赖分析，能够精确地留空并可靠地得出程序是否满足非干扰的结论，还能在给定界限内提供反驳模型。鉴于这些令人鼓舞的结果，我们计划在未来推广目标安全属性，并使这些分析方法能够扩展到其他语言。

致谢 作者感谢匿名评审提出的意见，这些意见有助于改进论文。本项目由 INRIA Challenge SPAI 和 VeriAMOS ANR项目资助。本研究还得到 ANR17-CE25-0014-01 CISC 项目的部分支持。我们也感谢 Josselin Giet 和 Adam Khayam 提供的见解。

参考文献

1. A. Aguirre、G. Barthe、M. Gaboardi、D. Garg 和 P. Strub。高阶程序的关系逻辑。 *Proc. ACM Program. Lang.*，（ICFP）：21:1–21:29，2017年。2. E. Alatawi、H. Søndergaard 和 T. Miller。利用抽象解释实现高效的动态符号执行。载于 G. Rosu、M. D. Penta 和 T. N. Nguyen 编，*Proceedings of the 32nd IEEE/ACM International Conference on AutomatedSoftware Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*年，第 619–624 页。IEEE计算机学会，2017年。3. A. Asperti 和 C. Armentano。数论中的一页。 *J. Formaliz. Reason.*，1 1 1( ): – 23，2008年。4. M. Assaf、D. A. Naumann、J. Signoles、 ´E. Totel 和 F. Tronel。超收集语义及其在信息流静态分析中的应用。载于 *Symposium on Principles of Programming Languages (POPL)* 874 887 2017 ，第 – 页。ACM，1月。5. M. Backes、B. K¨opf 和 A. Rybalchenko。信息泄漏的自动发现与量化。载于 *30th IEEE Symposium on Security and Privacy (S&P 2009), 17-20 May 2009 Oakland California USA* 141 153, 2009 ， ,,，第 – 页。

6. A. Banerjee、D. A. Naumann 和 S. Rosenberg。富有表现力的去分类策略与模块化静态强制执行。载于 *2008) 18-21 May 2008 Oakland California USA* *2008 IEEE Symposium on Security and Privacy (SP* 2008 , , , , 。IEEE 计算机学会，。7. G. Barthe 、 P. R. D'Argenio 和 T. Rezk。通过自组合实现安全信息流。载于 *Proceedings of the IEEE Computer Security Foundations Workshop (CSF)*，第 17 卷，第 100–114 页，2004年。8. N. Bielova 和 T. Rezk。信息流监控器分类法。载于 *Principles of Security and Trust - 5th International Conference, POST 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016,Eindhoven, The Netherlands, April 2-8*，2016年， *Proceedings*，《计算机科学讲义》，第 46–67 页。Springer，2016年。9. R. S. Boyer、B. Elspas 和 K. N. Levitt。SELECT——一种通过符号执行进行程序测试和调试的形式系统。载于 *Proceedings of the International Conference on Reliable Software 1975, Los Angeles, California, USA, April 21-23*，1975年，第 234–245 页。ACM，1975年。10. C. Cadar、P. Godefroid、S. Khurshid、C. S. Pasareanu、K. Sen、N. Tillmann 和 W. Visser。面向软件测试的符号执行实践：初步评估。载于 *Proceedings of the 33rd International Conference on Software Engineering ICSE* , , , , , , *2011 Waikiki Honolulu HI USA* *2011* 1066 1071 2011 *May 21-28,* , 第 – 页。ACM，。11. C. Cadar 和 M. Nowack。 2019 *International* 中的 KLEE 符号执行引擎。*Journal of Software Tools Technol. Transf.* 2021 , 。12. C. Cadar 和 K. Sen。软件测试的符号执行：三十年回顾。*Communications of the ACM*，第 82–90 页，2013年。13. M. R. Clarkson 和 F. B. Schneider。超属性。载于 *Computer Security Foundations Symposium (CSF)* *Proceedings of the IEEE* 51 65 2008 , 第 – 页。IEEE，。14. P. Cousot 和 R. Cousot。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 *Symposium on Principles of Programming Languages (POPL)* 238 252 1977 , 第 – 页。ACM，。15. P. Cousot 和 R. Cousot。程序分析框架的系统化设计。载于 *Symposium on Principles of Programming Languages (POPL)*。ACM，1979年。16. P. Cousot 和 N. Halbwachs。自动发现程序变量之间的线性约束。载于 *Symposium on Principles of Programming Languages (POPL)*，第 84–97 页。ACM，1978年。17. L. Daniel、S. Bardin 和 T. Rezk。Binsec/rel：面向二进制级常数时间的高效关系符号执行。载于 *2020 IEEE Symposium on Security and Privacy, SP 2020 San Francisco CA USA May 18-21 2020* 1021 1038, 2020 , , , , , , 第 – 页。18. L. Daniel、S. Bardin 和 T. Rezk。追捕幽灵——使用 haunted relse 对 Spectre 进行高效关系符号执行。载于 *28th Annual Network and Distributed System Security Symposium NDSS 2021 virtually February 21-25 2021* , , , , 。互联网协会，2021年。19. L. Daniel、S. Bardin 和 T. Rezk。关于 Spectre 二进制级符号分析器实验评估的思考。载于 *Post-proceedings of the LASER@NDSS 2021* 2022 。互联网协会，。20. ´A. Darvas、R. H¨ahnle 和 D. Sands。通过定理证明分析安全信息流的方法。载于 D. Hutter 和 M. Ullmann 编， *Security in PervasiveComputing, Second International Conference, SPC 2005, Boppard, Germany, April 6-8*，2005 193 209 2005， *Proceedings*，《计算机科学讲义》，第 – 页。Springer，。21. L. de Moura 和 N. Bjørner。Z3：一种高效的 SMT 求解器。载于 C. R. Ramakrishnan 和 J. Rehof 编， *Tools and Algorithms for the Construction and Analysis of Systems*，第 337–340 页，柏林，海德堡，2008年。Springer Berlin Heidelberg。

22. D. Delmas 和 A. Min´e。使用数值抽象解释分析软件补丁。载于 B. E. Chang 编，*Static Analysis Symposium (SAS)*，*LNCS* 第 11822 卷，第 225–246 页。Springer，2019。23. G. P. Farina、S. Chong 和 M. Gaboardi。关系符号执行。载于 E. Komendantskaya 编，*Proceedings of the 21st International Symposium on Principles and Practice of Programming Languages*，*PPDP 2019*，*Porto*，*Portugal*，*October 7-9 2019* 10 1 10 14 2019 ，第 : – : 页。ACM。24. C. Fournet、J. Planul 和 T. Rezk。同态加密的信息流类型。载于 Y. Chen、G. Danezis 和 V. Shmatikov 编，*Proceedings of the18th ACM Conference on Computer and Communications Security*，*CCS 2011*，*Chicago*，*Illinois USA October 17-21 2011* 351 360, 2011 ，第 – 页。25. R. Giacobazzi 和 I. Mastroeni。抽象非干扰：通过抽象解释参数化非干扰。载于 *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*，*POPL 2004*，*Venice*，*Italy January 14-16 2004* 186 197 2004 ，第 – 页。ACM。26. J. A. Goguen 和 J. Meseguer。安全策略与安全模型。载于 *IEEE Symposium on Security and Privacy Oakland* 11 20 1982 ，第 – 页。IEEE计算机学会。27. M. Huisman、P. Worah 和 K. Sunesen。观察确定性的时序逻辑刻画。载于 *19th IEEE Computer Security Foundations Workshop (CSFW'06)* 2006 。28. S. Hunt 和 D. Sands。论流敏感安全类型。载于 *Symposium on Principles of Programming Languages (POPL)* 79 90 2006 ，第 – 页。ACM。

29. B. Jeannet 和 A. Min´e。Apron：用于静态分析的数值抽象域库。载于 A. Bouajjani 和 O. Maler 编，*Computer Aided Verification 21st International Conference CAV 2009 Grenoble France June 26 - July 2 2009. Proceedings* ，*Lecture Notes in Computer Science* 第 5643 卷，第 661–667 页。Springer，2009年。30. H. R. Jr. *Theory of recursive functions and effective computability (Reprint from 1967)*。麻省理工学院出版社，1987年。31. J. C. King。符号执行与程序测试。*Communications of the ACM*，19 7 385 394，1976 (): –。32. D. Milushev、W. Beck 和 D. Clarke。通过符号执行实现非干扰。载于 *Formal Techniques for Distributed Systems Joint 14th IFIP WG 6.1 International Conference FMOODS 2012 and 32nd IFIP WG 6.1 International Conference FORTE 2012 Stockholm Sweden June 13-16 2012. Proceedings* ，计算机科学讲义，第 152–168 页。Springer，2012年。33. M. Ngo、N. Bielova、C. Flanagan、T. Rezk、A. Russo 和 T. Schmitz。动态信息流控制的一个更佳方面。载于 P. Champin、F. Gandon、M. Lalmas 和 P. G. Ipeirotis 编，*2018 WWW 2018 Lyon France April 23-27 2018* 731 739, 2018 ，第 – 页。34. H. Palikareva 、 T. Kuchta 和 C. Cadar。疑云重重：测试软件版本之间的差异。载于 *Proceedings of the 38th International Conference on Software Engineering ICSE 2016 Austin TX USA May 14-22 2016* 1181 1192 ，第 – 页。ACM，2016年。35. C. S. Pasareanu、P. C. Mehlitz、D. H. Bushnell、K. Gundy-Burlet、M. R. Lowry、S. Person 和 M. Pape。结合单元级符号执行与系统级具体执行来测试美国国家航空航天局软件。载于 B. G. Ryder 和 A. Zeller 编，*Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis ISSTA 2008 Seattle WA USA July 20-24 2008* 15 26 2008 ，第 – 页。ACM。

36. A. Sabelfeld 和 A. C. Myers。基于语言的信息流安全。*IEEE Journal Sel. Areas Commun.* 21 1 5 19, 2003 ，( ): –。

37. A. Sabelfeld 和 D. Sands。去分类的维度与原则。载于 *18th IEEE Computer Security Foundations Workshop, (CSFW-18 2005), 20-22 June 2005, Aix-en-Provence France* 255 269 2005 ，，第 – 页。IEEE计算机学会。38. J. F. Santos、T. P. Jensen、T. Rezk 和 A. Schmitt。类 JavaScript 语言中安全信息流的混合类型判定。载于 *Trustworthy Global Computing - 10th International Symposium TGC 2015 Madrid Spain August 31 - September 1 2015 Revised Selected* ，，，，，，计 *Papers* 63 78 2015 算机科学讲义，第 – 页。Springer。39. T. Terauchi 和 A. Aiken。将安全信息流视为安全性问题。载于 *Static Analysis, 12th International Symposium, SAS 2005, London, UK, 2005* 年， *Proceedings*, *Lecture Notes in Computer Science* 第 3672 卷，第 352–367 页。Springer，2005年。

使用区间的 RedSoundSE 对程序 2(d) 的一条轨迹

本节旨在展示程序 2(d) 的一条符号轨迹的执行过程。初始精确存储 κ 将记录变量 i和 y的初始低等价。抽象状态为 a。乘积存储的变化以红色标记。

$$\kappa = \begin{cases} \rho = [\mathtt{i} \to \langle i_0 \rangle, \mathtt{y} \to \langle y_0 \rangle, \mathtt{priv} \to \langle priv_0 \rangle] \\ \pi = \mathbf{tt} \\ a_l = [\,] \end{cases}$$

在第 3 行，由于 priv 不受约束，语义可以选择任意一条路径。假设我们的轨迹遵循规则 S-IF-T。那么，到第 5 行时，状态如下。

$$\kappa = \begin{cases} \rho = [\mathtt{i} \to \langle i_0 \rangle, \mathtt{y} \to \langle y_0 \rangle, \mathtt{priv} \to \langle 0 \rangle] \\ \pi = priv_0 < 0 \\ a = [\mathtt{priv} = 0] \end{cases}$$

由于这个循环的迭代次数没有上界，我们知道将进行过度近似。假设迭代上界为 1（即语义在进行过度近似之前最多执行一次循环），并且 i<10。执行完一次完整迭代后，将得到以下符号状态。

$$\kappa = \begin{cases} \rho = [\mathtt{i} \to \langle i_0 + 1 \rangle, \mathtt{y} \to \langle y_0 \rangle, \mathtt{priv} \to \langle 2 \rangle] \\ \pi = i_0 < 10 \wedge priv_0 < 0 \\ a = [\mathtt{priv} = 0; \mathtt{i} < 11] \end{cases}$$

现在已达到迭代次数上限，下一步是对循环进行过度近似。对于这个例子，接下来将展示归约之前的状态。请注意，π 中的新约束是对守卫取反的结果。

$$\kappa = \begin{cases} \rho = [\mathtt{i} \to \langle i_1 \rangle, \mathtt{y} \to \langle y_0 \rangle, \mathtt{priv} \to \langle priv_1 \rangle] \\ \pi = i_1 \geq 10 \wedge i_0 < 10 \wedge priv_0 < 0 \\ a = [\mathtt{priv} \geq 2; \mathtt{i} = 10] \end{cases}$$

由于变量 i 和 priv 被修改，因此赋予了新的符号值。这会造成很大的不准确性，但抽象状态可以弥补。通过化简，我们将 a 到 π 的约束添加进来。

$$\kappa = \begin{cases} \rho = [\mathtt{i} \to \langle i_1 \rangle, \mathtt{y} \to \langle y_0 \rangle, \mathtt{priv} \to \langle priv_1 \rangle] \\ \pi = priv_1 \geq 2 \wedge i_1 = 10 \wedge i_1 \geq 10 \wedge i_0 < 10 \wedge priv_0 < 0 \\ a = [\mathtt{priv} \geq 2; \mathtt{i} = 10] \end{cases}$$

通过归约，我们获得了有关 i低等值的信息，同时也获得了有关 priv为正的信息。最后，最后一个 `if` 语句不会执行。

B SE步关系

本节展示了 SE（标准的非可靠符号执行）的全部规则。

$$\text{S-ASSIGN}\ \frac{(\mathbf{e},\rho)\vdash_{\mathrm{s}}\varepsilon}{(\mathbf{x}:=\mathbf{e},(\rho,\pi))\rightharpoonup_s(\texttt{skip},(\rho[\mathbf{x}\rightsquigarrow\langle\varepsilon\rangle],\pi))}$$

$$\text{S-SEQ-EXIT}\ \frac{}{(\texttt{skip};\ \mathbf{c}_1,\kappa)\rightharpoonup_s(\mathbf{c}_1,\kappa)}\qquad \text{S-SEQ}\ \frac{(\mathbf{c}_0,\kappa)\rightharpoonup_s(\mathbf{c}_0',\kappa')}{(\mathbf{c}_0;\ \mathbf{c}_1,\kappa)\rightharpoonup_s(\mathbf{c}_0';\ \mathbf{c}_1,\kappa')}$$

$$\text{S-IF-T}\ \frac{(\mathbf{b},\rho)\vdash_{\mathrm{s}}\beta\qquad \pi'\triangleq\pi\wedge\beta\qquad \mathbf{may}(\pi')}{(\texttt{if}\ \mathbf{b}\ \texttt{then}\ \mathbf{c}_0\ \texttt{else}\ \mathbf{c}_1,(\rho,\pi))\rightharpoonup_s(\mathbf{c}_0,(\rho,\pi))}$$

$$\text{S-IF-F}\ \frac{(\mathbf{b},\rho)\vdash_{\mathrm{s}}\beta\qquad \pi'\triangleq\pi\wedge\neg\beta\qquad \mathbf{may}(\pi')}{(\texttt{if}\ \mathbf{b}\ \texttt{then}\ \mathbf{c}_0\ \texttt{else}\ \mathbf{c}_1,(\rho,\pi))\rightharpoonup_s(\mathbf{c}_1,(\rho,\pi))}$$

$$\text{S-LOOP-T}\ \frac{(\mathbf{b},\rho)\vdash_{\mathrm{s}}\beta\qquad \pi'\triangleq\pi\wedge\beta\qquad \mathbf{may}(\pi')}{(\texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c},(\rho,\pi))\rightharpoonup_s(\mathbf{c};\ \texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c},(\rho,\pi))}$$

$$\text{S-LOOP-F}\ \frac{(\mathbf{b},\rho)\vdash_{\mathrm{s}}\beta\qquad \pi'\triangleq\pi\wedge\neg\beta\qquad \mathbf{may}(\pi')}{(\texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c},(\rho,\pi))\rightharpoonup_s(\texttt{skip},(\rho,\pi))}$$

C SoundSEstep关系

本节通过符号执行展示附录 B 中 SoundSE 的完整规则集。

$$\text{S-NEXT}\ \frac{(\mathbf{c},\kappa)\rightharpoonup_s(\mathbf{c}',\kappa')\qquad \mathfrak{step}(\mathbf{c},\mathbf{c}',w)=(\mathbf{tt},w')}{(\mathbf{c},\kappa,w,b)\rightharpoonup_s(\mathbf{c}',\kappa',w',b)}$$

$$\text{S-APPROX-MANY}\ \frac{(\mathbf{c},\kappa)\rightharpoonup_s(\mathbf{c}',\kappa')\qquad \mathfrak{step}(\mathbf{c},\mathbf{c}',w)=(\mathbf{ff},w')\qquad \rho''=\mathfrak{modif}(\rho,\mathbf{c})}{(\mathbf{c},(\rho,\pi),w,b)\rightharpoonup_s(\texttt{skip},(\rho'',\pi),w',\mathbf{ff})}$$

D 抽象步关系

本节展示 RedSoundSE 中所用抽象分析的完整规则集。

$$\text{A-ASSIGN}\ \frac{a' \triangleq \mathfrak{assign}_{\mathbf{x},\mathbf{e}}(a)}{(\mathbf{x} := \mathbf{e}, a) \rightharpoonup_{\mathbb{A}} (\texttt{skip}, a')} \qquad \text{A-IF-T}\ \frac{a' \triangleq \mathfrak{guard}_{\mathbf{b}}(a) \qquad a' \neq \bot}{(\texttt{if}\ \mathbf{b}\ \texttt{then}\ \mathbf{c}_0\ \texttt{else}\ \mathbf{c}_1, a) \rightharpoonup_{\mathbb{A}} (\mathbf{c}_0, a')}$$

$$\text{A-IF-F}\ \frac{a' \triangleq \mathfrak{guard}_{\neg\mathbf{b}}(a) \qquad a' \neq \bot}{(\texttt{if}\ \mathbf{b}\ \texttt{then}\ \mathbf{c}_0\ \texttt{else}\ \mathbf{c}_1, a) \rightharpoonup_{\mathbb{A}} (\mathbf{c}_0, a')}$$

$$\text{A-SEQ-EXIT}\ \frac{}{(\texttt{skip};\ \mathbf{c}_1, a) \rightharpoonup_{\mathbb{A}} (\mathbf{c}_1, a)} \qquad \text{A-SEQ}\ \frac{(\mathbf{c}_0, a) \rightharpoonup_{\mathbb{A}} (\mathbf{c}_0', a')}{(\mathbf{c}_0;\ \mathbf{c}_1, a) \rightharpoonup_{\mathbb{A}} (\mathbf{c}_0';\ \mathbf{c}_1, a')}$$

$$\text{A-LOOP-F}\ \frac{a' \triangleq \mathfrak{guard}_{\neg\mathbf{b}}(a) \qquad a' \neq \bot}{(\texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c}_0, a) \rightharpoonup_{\mathbb{A}} (\texttt{skip}, a')}$$

$$\text{A-LOOP-T}\ \frac{a' \triangleq \mathfrak{guard}_{\mathbf{b}}(a) \qquad a' \neq \bot}{(\texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c}_0, a) \rightharpoonup_{\mathbb{A}} (\mathbf{c};\ \texttt{while}\ \mathbf{b}\ \texttt{do}\ \mathbf{c}_0, a')}$$

E RedSoundSE 步进关系

RedSoundSE 由附录 B、附录 C 和附录 D 中的规则定义。

$$\text{S-A-NEXT}\ \frac{\begin{array}{c}(\mathbf{c}, \kappa, w, b) \rightharpoonup_s (\mathbf{c}', \kappa', w', b) \\ \mathfrak{step}(\mathbf{c}, \mathbf{c}', w) = (\mathbf{tt}, w') \qquad (\mathbf{c}, a) \rightharpoonup_{\mathbb{A}} (\mathbf{c}', a') \qquad (\kappa'', a'') \triangleq \mathfrak{reduction}(\kappa', a')\end{array}}{(\mathbf{c}, \kappa, a, w, b) \rightharpoonup_{s \times \mathbb{A}} (\mathbf{c}', \kappa'', a'', w', b)}$$

$$\text{S-A-APPROX-MANY}\ \frac{\begin{array}{c}(\mathbf{c}, \kappa) \rightharpoonup_s (\mathbf{c}', \kappa') \qquad \mathfrak{step}(\mathbf{c}, \mathbf{c}', w) = (\mathbf{ff}, w') \\ \kappa'' = \mathfrak{modif}(\kappa, \mathbf{c}) \qquad a' = [\![\mathbf{c}]\!]^\sharp_{\mathbb{A}}(a) \qquad (\kappa''', a''') \triangleq \mathfrak{reduction}(\kappa'', a')\end{array}}{(\mathbf{c}, (\kappa, a), w, b) \rightharpoonup_{s \times \mathbb{A}} (\texttt{skip}, (\kappa''', a'''), w', \mathbf{ff})}$$

F 关系符号执行与 SoundRSE 步关系

本节展示 SoundRSE 的全部规则。通过移除规则 SR-APPROX-MANY，以及移除计数器和布尔标志，即可得到 RSE；RSE 是 SoundRSE 的子集。

$$\textsc{SR-ASSIGN}\ \frac{(\mathbf{e},\tilde{\rho})\vdash_{\mathrm{sr}}\tilde{\varepsilon}}{(\mathbf{x}:=\mathbf{e},(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr}(\texttt{skip},(\tilde{\rho}[\mathbf{x}\rightsquigarrow\langle\tilde{\varepsilon}\rangle],\pi),w,b)}$$

$$\textsc{SR-SEQ-EXIT}\ \frac{}{(\texttt{skip};\,\mathbf{c}_1,\tilde{\kappa})\rightharpoonup_{sr}(\mathbf{c}_1,\tilde{\kappa})}\qquad \textsc{SR-SEQ}\ \frac{(\mathbf{c}_0,\tilde{\kappa})\rightharpoonup_{sr}(\mathbf{c}_0',\tilde{\kappa}')}{(\mathbf{c}_0;\,\mathbf{c}_1,\tilde{\kappa})\rightharpoonup_{sr}(\mathbf{c}_0';\,\mathbf{c}_1,\tilde{\kappa}')}$$

$$\textsc{SR-IF-TT}\ \frac{(\mathbf{b},\tilde{\rho})\vdash_{\mathrm{sr}}\tilde{\beta}\qquad \pi'=\pi\wedge\Pi_0(\tilde{\beta})\wedge\Pi_1(\tilde{\beta})\qquad \mathbf{may}(\pi')}{(\texttt{if }\mathbf{b}\texttt{ then }\mathbf{c}_0\texttt{ else }\mathbf{c}_1,(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr}(\mathbf{c}_0,(\tilde{\rho},\pi'),w,b)}$$

$$\textsc{SR-IF-TF}\ \frac{(\mathbf{b},\tilde{\rho})\vdash_{\mathrm{sr}}\tilde{\beta}\qquad \pi'=\pi\wedge\Pi_0(\tilde{\beta})\wedge\neg\Pi_1(\tilde{\beta})\qquad \mathbf{may}(\pi')}{(\texttt{if }\mathbf{b}\texttt{ then }\mathbf{c}_0\texttt{ else }\mathbf{c}_1,(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr}(\mathbf{c}_0\bowtie\mathbf{c}_1,(\tilde{\rho},\pi'),w,b)}$$

$$\textsc{SR-IF-FT}\ \frac{(\mathbf{b},\tilde{\rho})\vdash_{\mathrm{sr}}\tilde{\beta}\qquad \pi'=\pi\wedge\neg\Pi_0(\tilde{\beta})\wedge\Pi_1(\tilde{\beta})\qquad \mathbf{may}(\pi')}{(\texttt{if }\mathbf{b}\texttt{ then }\mathbf{c}_0\texttt{ else }\mathbf{c}_1,(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr}(\mathbf{c}_1\bowtie\mathbf{c}_0,(\tilde{\rho},\pi'),w,b)}$$

$$\textsc{SR-IF-FF}\ \frac{(\mathbf{b},\tilde{\rho})\vdash_{\mathrm{sr}}\tilde{\beta}\qquad \pi'=\pi\wedge\neg\Pi_0(\tilde{\beta})\wedge\neg\Pi_1(\tilde{\beta})\qquad \mathbf{may}(\pi')}{(\texttt{if }\mathbf{b}\texttt{ then }\mathbf{c}_0\texttt{ else }\mathbf{c}_1,(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr}(\mathbf{c}_0,(\tilde{\rho},\pi'),w,b)}$$

$$\textsc{SR-LOOP-TT}\ \frac{\mathfrak{step}(\mathbf{c},\mathbf{c}',w)=(\mathbf{tt},w')\qquad (\mathbf{b},\tilde{\rho})\vdash_{\mathrm{sr}}\tilde{\beta}\qquad \pi'=\pi\wedge\Pi_0(\tilde{\beta})\wedge\Pi_1(\tilde{\beta})\qquad \mathbf{may}(\pi')}{(\texttt{while }\mathbf{b}\texttt{ do }\mathbf{c}_0,(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr}(\mathbf{c}_0;\,\texttt{while }\mathbf{b}\texttt{ do }\mathbf{c}_0,(\tilde{\rho},\pi),w',b)}$$

$$\textsc{SR-LOOP-TF}\ \frac{\mathfrak{step}(\mathbf{c},\mathbf{c}',w)=(\mathbf{tt},w')\qquad (\mathbf{b},\tilde{\rho})\vdash_{\mathrm{sr}}\tilde{\beta}\qquad \pi'=\pi\wedge\Pi_0(\tilde{\beta})\wedge\neg\Pi_1(\tilde{\beta})\qquad \mathbf{may}(\pi')}{(\texttt{while }\mathbf{b}\texttt{ do }\mathbf{c}_0,(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr}((\mathbf{c}_0;\,\texttt{while }\mathbf{b}\texttt{ do }\mathbf{c}_0)\bowtie\texttt{skip},(\tilde{\rho},\pi'),w',b)}$$

$$\textsc{SR-LOOP-FT}\ \frac{\mathfrak{step}(\mathbf{c},\mathbf{c}',w)=(\mathbf{tt},w')\qquad (\mathbf{b},\tilde{\rho})\vdash_{\mathrm{sr}}\tilde{\beta}\qquad \pi'=\pi\wedge\neg\Pi_0(\tilde{\beta})\wedge\Pi_1(\tilde{\beta})\qquad \mathbf{may}(\pi')}{(\texttt{while }\mathbf{b}\texttt{ do }\mathbf{c}_0,(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr}(\texttt{skip}\bowtie(\mathbf{c}_0;\,\texttt{while }\mathbf{b}\texttt{ do }\mathbf{c}_0),(\tilde{\rho},\pi'),w',b)}$$

$$\textsc{SR-LOOP-FF}\ \frac{\mathfrak{step}(\mathbf{c},\mathbf{c}',w)=(\mathbf{tt},w')\qquad (\mathbf{b},\tilde{\rho})\vdash_{\mathrm{sr}}\tilde{\beta}\qquad \pi'=\pi\wedge\neg\Pi_0(\tilde{\beta})\wedge\neg\Pi_1(\tilde{\beta})\qquad \mathbf{may}(\pi')}{(\texttt{while }\mathbf{b}\texttt{ do }\mathbf{c}_0,(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr}(\texttt{skip},(\tilde{\rho},\pi'),w',b)}$$

$$\textsc{SR-EXIT}\ \frac{}{(\texttt{skip}\bowtie\texttt{skip},(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr}(\texttt{skip},(\tilde{\rho},\pi),w,b)}$$

$$\textsc{SR-COMP-R}\ \frac{(\mathbf{c}_1,(\Pi_1(\tilde{\rho}),\pi),w,b)\rightharpoonup_{s}(\mathbf{c}_1',(\rho_1',\pi'),w',b')}{(\texttt{skip}\bowtie\mathbf{c}_1,(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr}(\texttt{skip}\bowtie\mathbf{c}_1',(\langle\!|\Pi_0(\tilde{\rho})\,|\,\rho_1'|\!\rangle,\pi'),w',b')}$$

$$\textsc{SR-COMP-L}\ \frac{(\mathbf{c}_0,(\Pi_0(\tilde{\rho}),\pi),w,b)\rightharpoonup_{s}(\mathbf{c}_0',(\rho_0',\pi'),w',b')}{(\mathbf{c}_0\bowtie\mathbf{c}_1,(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr}(\mathbf{c}_0'\bowtie\mathbf{c}_1,(\langle\!|\rho_0'\,|\,\Pi_1(\tilde{\rho})|\!\rangle,\pi'),w',b')}$$

$$\textsc{SR-APPROX-MANY}\ \frac{\begin{array}{c}\mathfrak{step}(\texttt{while }\mathbf{b}\texttt{ do }\mathbf{c},(\mathbf{c};\,\texttt{while }\mathbf{b}\texttt{ do }\mathbf{c}),w)=(\mathbf{ff},w')\\ \tilde{\rho}''=\mathfrak{modif}(\tilde{\rho},\mathbf{c})\qquad (\mathbf{b},\tilde{\rho}'')\vdash_{\mathrm{sr}}\langle\beta_0,\beta_1\rangle\qquad \pi'\triangleq\pi\wedge\neg\beta_0\wedge\neg\beta_1\end{array}}{(\texttt{while }\mathbf{b}\texttt{ do }\mathbf{c},(\tilde{\rho},\pi),w,b)\rightharpoonup_{sr}(\texttt{skip},(\tilde{\rho}'',\pi'),w',\mathbf{ff})}$$

G RedSoundRSE步关系

RedSoundRSE 由附录 F 的规则以及规则 SR-APPROX-MANY-DEP 定义。

$$\frac{\mathfrak{step}(\text{当 b 执行 c 时，(c；当 b 执行 c)}\ w) = (\mathbf{ff}\ w') \quad d = [\![\text{当 b 执行 c}]\!]^\sharp\ \mathrm{D}(\tau_{s\to\mathbb{D}}(\tilde{\rho})) \quad \tilde{\rho}'' = \mathfrak{modify}_{\mathbb{D}}(\tilde{\rho}, \text{c}\ \lambda_{\mathbb{D}\to\mathbb{L}}(d)) \quad \text{SR-APPROX-MANY-DEP}\ (\text{b}\ \tilde{\rho}'') \vdash_{\mathrm{sr}} \langle \beta_0, \beta_1 \rangle \quad \pi' \triangleq \pi \wedge \neg\beta_0 \wedge \neg\beta_1}{(\text{当 b 执行 c}, (\tilde{\rho}, \pi)\ w, b) \rightharpoonup_{sr\times\mathbb{D}} (\text{空操作}, (\tilde{\rho}'', \pi')\ w', \mathbf{ff})}$$