非确定性程序假设下的静态分析

Abdullah H. Rasheed

arXiv:2602.07324v1 [cs.PL] 7 Feb 2026

摘要

静态分析通常以牺牲精度为代价来保证可靠性和自动化。因此，其适用场景仅限于精度损失不会造成严重影响的情况。本文提出并规范了一种静态分析，该分析接受用户提供的、局部适用于各程序位置的程序假设。这些假设可用于弥补静态分析的精度损失，从而使其能够应用于范围广得多的场景。分析器以非确定性方式采纳这些假设，因此可以建立一个函数，将已采纳假设的集合映射为基于这些假设得到的分析结果。我们还通过两种方式展示了这一函数的实用性；两者均表明，该函数能够在假设的搜索空间上进行优化，而若没有所规范的分析，这种优化将不可行。

1 引言

静态分析是一种广泛用于错误检测、编译器级优化等领域的方法。该技术无需执行程序，即可确定给定程序的相关性质。这类分析通常被归类为“可靠的过近似”，因为它们可能产生假阳性。例如，一种可靠的错误检测静态分析会报告所有错误实例，但对其中一些实例可能无法确定。此类不确定情况通常称为假阳性。相反，如果分析器报告未发现任何错误，那么程序中确实不存在错误。

一般而言，可靠的静态分析器必须在自动化、精度和可扩展性之间取得平衡。虽然“假设—保证”推理可以通过允许局部细化来牺牲可靠性，从而提高精度和可扩展性，但现有框架要求假设必须先验地固定。此外，这种方法仍将环境视为静态输入，因此面对不确定性时十分脆弱。这限制了此类分析在需要优化和综合的任务中的效力，因为分析器若不先确定某个假设，就无法推理该假设所能提供的保证，进而导致代价高昂的反复试错循环。通过对假设进行非确定性选择来开展分析的方法，可以弥补这一缺口，提供缺失的“保证”信息，并支持针对候选假设集合进行优化。更形式化地说，我们提出一种静态分析的符号化参数化方法，将域 Σ提升为函数域$2^{\mathcal{A}} \to \Sigma$ ，其中 $\mathcal{A}$ 是在程序位置处标注的一组用户提供的假设。这立即提供了一个预言机，可针对任意所需的假设集合，获取相应假设下的静态分析结果。我们还表明，这种参数化方法能够基于抽象解释器实现，同时保持可靠性。

许多现有工作旨在解决类似的目标，但我们的方法因其以语义性的运行时假设来限定验证而仍然新颖。这不同于轨迹划分等已知方法，后者通过在条件路径（或一般而言的控制流历史）上划分状态空间来恢复精度 [16]。现有的提升的静态分析与我们的方法高度相关，但划分仅限于句法上的编译时变化性（在构建时定义的布尔特征） [5, 13]。即使支持运行时变化性的方法，也仅限于对环境值进行划分，而非运行时程序假设 [8]。诸如析取抽象域之类的方法 [15, 17, 22, 10] 通过将状态表示为抽象元素的析取（有时编码路径条件）来获得高精度，但它们会遭受路径爆炸，且其析取是“盲目的”。它们可能知道状态是 $A \vee B$，但它们失去了为何的原因，而我们的方法用假设对析取进行参数化（实际上，$(Assumption_1 \to A) \wedge (Assumption_2 \to B)$）。这使我们能够维持环境、运行时假设与程序状态之间的因果联系。

与需要进行指数级重新分析的朴素方法不同，我们采用单遍分析，并运用两种策略来提升可扩展性。第一种策略是对参数化状态强制采用范式。通过将假设空间划分为等价类，我们得以维护一种唯一且紧凑的表示，合并冗余的分析结果，从而能够针对大量候选假设进行可扩展的推理。第二种策略则为开发者提供一个可调节的“旋钮”，以便在可扩展性和精度之间取得平衡，并可按需降低最坏情况运行时间。具体而言，这种策略采用近似合并，通过连接各等价类的分析结果来合并等价类，从而产生过近似。此缩减可以任意应用，从而实现可控的 近似。

为了说明　　参数化分析的实用性在于，我们证明它简化了一个假设

综合问题：给定一个具有假设空间（候选假设）且包含许多分散在各处的断言语句的程序，找出一组最小假设，使程序满足所有断言。正如我们在论文中所讨论的，这一问题的简化也意味着许多其他相关问题的简化。

我们还探讨假设集中的潜在“不一致”问题：首先证明，参数化分析结果支持在 2^A 上进行不动点计算。这使我们能够采用多种已知的有原则的技术来获取有用信息。具体而言，我们使用非单调逻辑中常见的不动点方法，找出一致和不一致的假设集。

总之，本文的主要贡献如下：

- **参数化分析**。 我们设计并形式化了一种方法，可扩展现有静态分析以支持“假设”语句，并对分析进行参数化，使其能够将假设子集映射为相应假设下的分析结果。该分析通过单次遍历完成，并使用范式以唯一且紧凑的形式表示参数化程序状态。

- **正确性证明与抽象解释**。 我们给出了一个正确性证明，其内容为：“对于程序中的每个假设语句子集 A，参数化分析在假设 A 下提供的分析结果，与原始分析器提供的结果相同。”我们还将此定理加以扩展，证明对可靠的抽象解释器进行参数化后，可得到一个相对于所有假设集均可靠的分析。

- **可靠的参数化近似**。 为避免参数化分析在最坏情况下（相对于假设数量）具有指数级复杂度，我们概述一种通过近似加快计算的方法。我们还证明了此操作的可靠性，并讨论如何尽可能减少精度损失。

- **假设合成**。 为展示此分析的适用性，我们说明如何将其应用于程序中的局部条件合成问题，以满足多个断言语句。此应用还展示了不可靠性并非总是问题的情形。

- **假设一致性界定**。 我们还展示了如何利用参数化分析来剪除假设集中的不一致性。也就是说，这使得找到不“自相矛盾”的假设集变得更容易。这是因为，借助预言机，可以获得任意给定假设子集下的分析结果，从而利用该预言机进行不动点计算。这项技术借鉴了知识表示领域中用于非单调逻辑的技术。

2 背景

一个程序表示为一个控制流图（CFG），其中 n 节点表示语句，有向边表示控制流。图1展示了一个书写形式的程序及其对应的控制流图。

节点根据语言语义的定义转换程序状态。例如，对应语句 $x := 5$ 的节点可能会将程序状态 σ 转换为 $\sigma[x \mapsto 5]$，其中 $\sigma \in Var \hookrightarrow \mathbb{Z}$。

收集语义 描述每个程序位置（控制流图节点）上可能出现的所有程序状态。本质上，如果 $Concrete$ 是具体程序状态的集合，那么收集语义 CS 定义在 $\mathcal{P}(Concrete)$ 上，使得对于任意控制流图节点 v，$CS(v) \subseteq Concrete$ 表示程序在 v 处可能处于的状态集合。语句也会相应地转换收集语义。例如，如果 $Concrete$ 由从变量到整数的映射构成，那么语句 $x := 5$ 可能会将具体状态集合 $S \subseteq Concrete$ 转换为集合 $\{\sigma[x \mapsto 5] \mid \sigma \in S\}$。

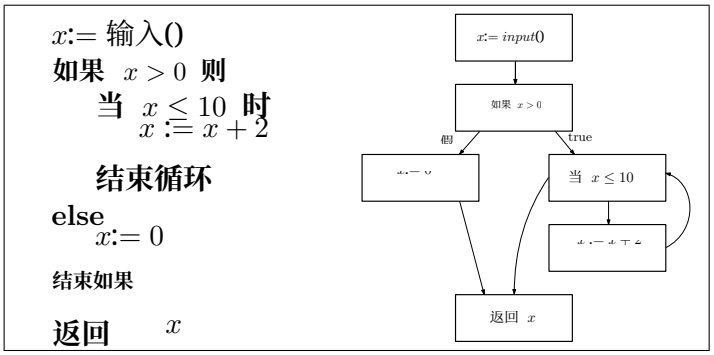

图1：示例程序及其对应的控制流图。

如果存在一条从控制流图节点 v 到另一节点 v'的有向边，我们将记为 $v \to v'$。如果状态$\sigma \in \Sigma$ 能够触发该转移，则我们记为 $v \to_\sigma v'$。 v 相对于具体状态集合 S 的前驱节点集合定义为

$$pre(v, S) = \{v' \mid \exists \sigma \in S : v' \to_\sigma v\}$$

并且我们将令 $pre(v) = \{v' \mid \exists \sigma \in CS(v') : v' \to_\sigma v\}$。就本文而言，$pre(v)$的定义可以保持抽象，甚至可以由过近似 $\{v' \mid v' \to v\}$来定义。

进入控制流图节点 v 的可能“传入”状态集合定义为

$$CS_{pre}(v) = \{\sigma \in CS(v') \mid v' \in pre(v)\}$$

我们将用 Σ 表示一个代表（抽象）程序状态的完备格。类似地，我们将用 $\Sigma^\sharp$ 表示通过伽罗瓦连接（稍后定义）与 Σ 相关联的抽象。

在本文中，我们将使用以下方程组来对静态分析进行建模。设 C_v 为附着于 v 的约束变量。该变量表示在程序中执行 v 之后直接得到的（抽象）程序状态。这些约束变量各自具有如下约束

$$C_v = \bigsqcup_{v' \in pre(v)} t_v(C_{v'})$$

其中某个变换器 t_v 表示 v 的语义。注意，这是关于所有约束变量的函数。也就是说，我们可以写成

$$\begin{aligned} C_{v_1} &= f_{v_1}(C_{v_1}, \ldots, C_{v_n}) \\ &\vdots \\ C_{v_n} &= f_{v_n}(C_{v_1}, \ldots, C_{v_n}) \end{aligned}$$

我们可以进一步将其简化为一个函数 $\vec{f}: \Sigma^n \to \Sigma^n$:

$$\vec{f}(\sigma_1, \ldots, \sigma_n) = (f_{v_1}(\sigma_1, \ldots, \sigma_n), \ldots, f_{v_n}(\sigma_1, \ldots, \sigma_n))$$

然后，分析结果为 lfp($\vec{f}$)。简写为，我们将把 $[\![P]\!]$ 记作程序 a 的分析结果

程序 P。

设 Σ 和 $\Sigma^\sharp$ 是完备格，且 $\alpha : \Sigma \to \Sigma^\sharp$ 和 $\gamma : \Sigma^\sharp \to \Sigma$。如果对于所有 $\sigma \in \Sigma$ 和 $\sigma^\sharp \in \Sigma^\sharp$，$\alpha(\sigma) \sqsubseteq^\sharp \sigma^\sharp \Leftrightarrow \sigma \sqsubseteq \gamma(\sigma^\sharp)$，则称 α 和 γ 构成伽罗瓦连接。若对于所有程序 P，$\alpha([\![P]\!]) \sqsubseteq [\![P]\!]^\sharp$，则称 $\Sigma^\sharp$ 上的分析 $[\![\cdot]\!]^\sharp$ 是**不动点可靠**的。本文中，我们将简称其为可靠，以表示不动点可靠。我们还将分别用 $\sqsubseteq, \sqcup, \sqcap$ 表示格的偏序关系、并运算和交运算；除非上下文不明确，否则省略表示格的额外记号，以使记号不那么繁杂。

设 L 是一个完备格。函数 $f\colon L \to L$ 若满足 $\forall x, y \in L\colon x \sqsubseteq y \Rightarrow f(x) \sqsubseteq f(y)$，则称其为单调函数；若满足 $\forall x, y \in L\colon x \sqsubseteq y \Rightarrow f(y) \sqsubseteq f(x)$，则称其为反单调函数。我们称 $f^0(x) = x$，并且当 $k > 0$ 时，称 $f^k(x) = f(f^{k-1}(x))$。Knaster-Tarski 不动点定理指出，对于任意单调的 f，其最小不动点 $\mathbf{lfp}(f)$ 和最大不动点 $\mathbf{gfp}(f)$ 必然存在 [20]。对于任意函数 $f\colon L \to L$，我们用 $\mathbf{fix}(f)$ 表示其不动点集合。对于单调的 f，若对每个 ω-链 $x_0 \sqsubseteq x_1 \sqsubseteq \ldots$（其上确界存在；在完备格中这总是成立），下列等式都成立，则称其为ω-连续函数：

$$f\left(\bigsqcup_{i\in\omega} x_i\right) = \bigsqcup_{i\in\omega} f(x_i)$$

我们假设静态分析的所有前向变换器 f_{v_i} 都是 ω-连续的（因此要求 t_{v_i}中的 ω-连续性）。稍后我们将讨论，这并非必要要求，只是为了简化证明。

最后，根据 Kleene 不动点定理，如果 f 是 ω-连续的，那么 $\mathbf{lfp}(f) = \bigsqcup_{k\geq 0} f^k(\bot)$，其中$\bot = \bigsqcup \emptyset$（格的底元素）[19]。

3 程序假设

现在，我们考虑为一种编程语言扩展语句**“假设** p”。直观而言，该语句用于限制 执行集合，使其仅包含满足 p的执行，从而对程序状态施加 p。我们不再让 p 成为关于程序状态的谓词，而是要求它表示一个程序状态 π_p。如果 $\sigma \sqsubseteq \pi_p$，则称状态 σ 满足 p。例如，在收集语义中，p 可以是具体状态的集合。于是，假设语句会将具体状态集合限制为包含于 p中的那些状态。当然，在程序中手动写出这些内容并不现实；不过我们将看到，在许多抽象域中，这种做法相当自然。

设 v 是表示假设语句 **假设** p 的控制流图节点。假设语句语义的更形式化定义如下：

$$CS(v) = \{\sigma \in CS_{pre}(v) \mid \sigma \sqsubseteq \pi_p\}$$

要在程序状态 σ 上施加假设 p，只需执行交运算$\sigma \sqcap \pi_p$。这会限制 σ，迫使其满足 p（因为 $\sigma \sqcap \pi_p \sqsubseteq \pi_p$）。请注意，假设的表达能力与所选域的表达能力直接相关。

示例。 假设程序状态属于 $Interval$ 域（从变量到整数区间的偏函数）。我们可以写出如下形式的假设：**假设** $x \geq 3$（对应于状态 $\pi_{x\geq 3} = [x \mapsto [3, \infty]]$），或**假设** $y \leq 7$（对应于状态 $\pi_{y\leq 7} = [y \mapsto [-\infty, 7]]$）。

我们还可以使用逻辑连接词，例如**假设** $x \geq 3 \wedge x \leq 10$ 以及其他逻辑连接词，并以任意方式组合 $\wedge, \vee, \neg$。设 $\sigma = [x \mapsto [-12, 8], y \mapsto [-\infty, 13]]$。为强制满足**假设** $x \geq 3 \wedge x \leq 10$，我们可以应用前文所述的交：

$$\sigma \sqcap \pi_{x \geq 3 \wedge x \leq 10} = \sigma \sqcap [x \mapsto [3, 10]] = [x \mapsto [3, 8], y \mapsto [-\infty, 13]]$$

并且该映射现在满足 $x \geq 3 \wedge x \leq 10$。

考虑 $\sigma \sqcap \pi_p$ 为"不可行"的情况。例如，如果 p 为 $x \geq 3$ 和 $\sigma = [x \mapsto [-\infty, 2]]$，$\sigma \sqcap \pi_p = [x \mapsto \emptyset]$，则它不是一个可行的程序状态。在这种情况下，我们有一个证明 表明，该假设在此位置和状态下为假。识别不可行状态将在之后搜索一致的假设集时发挥作用。我们将用谓词 $Feasible_p(\sigma)$表示 σ 对于 p 是否可行（$\sigma \sqcap \pi_p$ 是否为可行的程序状态）。我们还假设 $Feasible_p$是单调的：如果 $\sigma_1 \sqsubseteq \sigma_2$且 $Feasible_p(\sigma_1)$，那么 $Feasible_p(\sigma_2)$。

在定义新的分析之前，我们将先为上述分析设置中的假设语句定义变换器。这是为了完整性，并为之后的正确性证明做准备。设 v 为语句**假设** p对应的控制流图节点。我们定义

$$t_v(\sigma) = \sigma \sqcap \pi_p$$

其语义是将假设 p 强制应用于输入状态，如前所述。

在本文余下部分中，我们用 $\mathcal{A}$ 表示给定程序中代表假设语句的控制流图节点的（有限）集合。对于所有 $a \in \mathcal{A}$，我们还将用命题变量 $\llparenthesis a \rrparenthesis$ 表示 a 是否被接受为假设。这样，我们就能以符号方式表示关于假设集的条件。

4 参数化分析

现在，我们希望对一个程序进行静态分析，得到一个函数 $F\colon 2^{\mathcal{A}} \to \Sigma^n$。也就是说，静态分析的结果应能回答这样一个问题："对于假设集 $A \subseteq \mathcal{A}$，在 A 下的静态分析结果是什么？"（即 $F(A)$）。我们也希望避免针对每个 $A \subseteq \mathcal{A}$ 分别运行静态分析。本节将概述一种单遍分析方法，该方法能够在回答这一问题的同时进行一些计算优化。然而，在保持完全精度的前提下，最坏情况下关于 $|\mathcal{A}|$ 的指数级开销无法避免；不过，我们稍后将证明，牺牲一些精度即可实现亚指数级运行时间。

4.1 参数化抽象域

我们定义的分析通过对已有分析进行"参数化"来实现。现在，先假设静态分析作用于单个具体域 Σ。之后我们会说明，为了提高计算可行性，这些方法也可以建立在抽象解释器之上。假设我们有一个可靠的前向变换器 $\vec{f}\colon (\Sigma)^n \to (\Sigma)^n$，使得 $[\![P]\!] = \mathrm{lfp}(\vec{f})$。

我们不再在 Σ 中的每个位置跟踪单个抽象状态，而是为每个位置跟踪一组规则。每条规则都具有

- A 条件 ϕ_i 基于假设。该规则仅适用于满足 ϕ 的 $A \subseteq \mathcal{A}$ $_i$.

- Ar 果 $\sigma_i \in \Sigma$ 对应于在条件 ϕ下成立的抽象状态 $_i$.

我们可以将 ϕ_i 视为原子命题来自 $\mathcal{A}$ 的布尔表达式，或者等价地视为 $\mathcal{A}$ 的子集。为方便起见，我们会交替使用这两种视角，但在实现中，使用前一种表示更为合理。形式上，我们将抽象域改为参数化版本

$$Par(\Sigma) = \{\{\phi_i, \sigma_i\}_i \mid \phi_i \subseteq 2^{\mathcal{A}} \wedge \sigma_i \in \Sigma\}$$

我们还将进一步加强这一要求：对于任意的 $\{\phi_i, \sigma_i\}_i \in Par(\Sigma)$， $\{\phi_i\}_i$ 都构成 $2^{\mathcal{A}}$ 的不相交划分。这确保了对于所有 $A \subseteq \mathcal{A}$，恰好只有一条规则适用于 A。在这一限制下，我们可以将 $\{\phi_i\}_i$ 视为假设的等价类，也就是说，如果A_1， $A_2 \in \phi_i$，那么在静态分析中，它们在局部是等价的（它们在该位置得出相同的分析结果 σ_i ）。

如预期， $Par(\Sigma)$ 中的状态表示 $2^{\mathcal{A}} \to \Sigma$ 中的函数。也就是说，对于 $X \in Par(\Sigma)$，其代表函数是

$$\rho[\![X]\!](A) = \sigma_i \text{ such that } (\phi_i, \sigma_i) \in X \wedge A \in \phi_i$$

这是定义良好的，因为我们要求 $\{\phi_i\}_i$ 构成 $2^{\mathcal{A}}$ 的不相交划分。请注意，将函数上的典范逐点偏序提升到 $Par(\Sigma)$ 上会产生问题，因为同一个函数可以由 $Par(\Sigma)$ 中的多个状态表示（因此不满足反对称性）。这个问题很快就会得到解决。

另一个问题是， X 仍可能非常庞大。下面我们将介绍一些方法，在不改变其代表函数的前提下缩减 X 的大小。这些方法无法解决最坏情况；该问题将在第4.5节讨论。

4.2 单射范式

我们希望为 $X \in Par(\Sigma)$ 找到一种唯一且“最紧凑”的表示。我们用 $X \sim Y$ 表示 $\rho[\![X]\!] = \rho[\![Y]\!]$（语义等价）。请注意， $Par(\Sigma)/\sim$ 与 $2^{\mathcal{A}} \to \Sigma$ 同构。现在，我们为 $Par(\Sigma)$ 定义一种范式，使每个语义等价类都有唯一的最紧凑代表。

定义 1 （单射范式（INF））。 $X \in Par(\Sigma)$ 属于 INF，当且仅当对于每个 (ϕ_i, σ_i)，$(\phi_j, \sigma_j) \in X$，满足 $i \neq j$、 $\sigma_i \neq \sigma_j$ 和 $\phi_i \neq \emptyset$。

换言之， X 属于 INF，当且仅当由 $h(\phi, \sigma) = \sigma$ 定义的函数 $h : X \to \Sigma$ 是单射，且 $image(\rho[\![X]\!]) = image(h)$。直观而言，INF 通过两种方式保持表示紧凑：

1. 每个分析结果只能对应一个等价类。如果 $\sigma_i = \sigma_j$，那么 ϕ_i 和 ϕ_j 描述的是同一分析结果的条件，这是冗余的。2. 每条规则都应当能够被调用。如果 ϕ_i （将 ϕ_i 视为谓词）不可满足，那么相应的规则将永远不会被调用，因此是冗余的。

对于任意 $X \in Par(\Sigma)$，我们都可以通过反复执行两种操作将其化简为 INF形式。第一种是精确合并，可对任意两条具有相同结果状态的规则 (ϕ_i, σ),$(\phi_j, \sigma) \in X$ 执行，将它们合并为一条规则 $(\phi_i \vee \phi_j, \sigma)$。第二种操作是冗余

消除，可对满足 ϕ 不可满足条件的任意规则 $(\phi, \sigma) \in X$ 执行，方法是将其从 X中移除。

示例。 考虑以下程序：

示例 1

v_1: x:=input()
v_2: **假设** $x > 0$
v_3: $x := 5$
v_4: **假设** $x = 0$

并且假设我们在 $Interval$ 域上工作，只跟踪 x。还假设进入 v_3时，参数化状态由两条规则组成：$(\llbracket v_2 \rrbracket, [1, \infty])$ 和 $(\neg\llbracket v_2 \rrbracket, [-\infty, \infty])$。正如下一节将会看到的，暂且将其视为黑箱，对这个状态应用 $x := 5$ 会得到规则 $(\llbracket v_2 \rrbracket, [5, 5])$ 和 $(\neg\llbracket v_2 \rrbracket, [5, 5])$，因为无论采用何种假设， x 的值都被设为 5。在这种情况下，我们不再需要跟踪多条规则，而是可以应用精确合并，得到单条规则 $(\llbracket v_2 \rrbracket \vee \neg\llbracket v_2 \rrbracket, [5, 5])$。

为了说明冗余消除，假设进入 v_2 时，参数化状态中包含规则 $(\neg\llbracket v_2 \rrbracket, [-\infty, \infty])$（可能是通过在一次 while 循环的迭代中经过 v_2 得到的）。那么，为了涵盖我们接受 v_2 处假设的情况（下一节将对此进行说明，但目前仍将其视为黑箱），我们可以将这条规则转换为规则 $(\neg\llbracket v_2 \rrbracket \wedge \llbracket v_2 \rrbracket, [1, \infty])$。显然，无论选择何种假设，都无法满足 $\neg\llbracket v_2 \rrbracket \wedge \llbracket v_2 \rrbracket$，因此可以消除这条规则。

引理 2. 对任意有限的 $X \in Par(\Sigma)$ 应用精确合并和冗余消除，直至达到不动点，得到状态 $X \in Par(\Sigma)$，使得 X 属于 *INF*，且 $X \sim X$。

证明。 在不动点处，精确合并和冗余消除已无法继续应用。ˆ 因此，结果状态 X 不再存在任何结果状态相同的规则对，也不再存在任何条件不可满足的规则。这符合 INF 的定义，ˆ 因此 X 属于 INF。

ˆ 要证明 $X \sim X$，只需证明，应用一次精确合并或冗余消除后，这种等价关系仍然成立。

假设存在一对规则 (ϕ_i, σ)，$(\phi_j, \sigma) \in X$，它们可以通过精确合并合并为一个新状态 X'。那么，如果 $A \in \phi_i \vee \phi_j$，我们有 $\rho[\![X]\!](A) = \sigma = \rho[\![X']\!](A)$。如果 $A \notin \phi_i \vee \phi_j$，则等式显然成立，因此 $X \sim X'$。

现在，反过来假设 X' 是通过对 $(\phi, \sigma) \in X$应用冗余消除得到的。在这种情况下， ϕ 为空，因此对于所有 A， $\rho[\![X]\!](A) = \rho[\![X']\!](A)$，因为 $A \notin \phi$。因此， $X \sim X'$。

ˆ 归纳地，这意味着 $X \sim X$。 □

请注意，前一个引理并未要求以任何特定顺序执行精确合并或冗余消除。在下面引理的证明中，我们还将证明，无论以何种顺序执行这些操作，最终总会得到相同的状态。

ˆ ˆ**引理 3**。 对于所有 $X \in Par(\Sigma)$，存在唯一的 $X \in Par(\Sigma)$，使得 X属于 *INF*，且ˆ$X \sim X$。

证明。由引理 2 可保证存在性。为证明唯一性，假设存在两个状态 $X_1, X_2 \in Par(\Sigma)$ 属于 INF，使得 $X_1 \sim X \sim X_2$ 且 $X_1 \neq X_2$。令 $X_1 = \{\phi_i, \sigma_i\}_i$ 且 $X_2 = \{\psi_i, \delta_i\}$。由于 $X_1 \sim X_2$，对于所有 $A \subseteq \mathcal{A}$，我们有 $\rho[\![X_1]\!](A) = \rho[\![X_2]\!](A)$，这意味着存在唯一的（由 INF 的第一个条件保证）$(\phi, \sigma) \in X_1$ 和 $(\psi, \delta) \in X_2$，使得 $\sigma = \delta$ 且 $A \in \phi$ 且 $A \in \psi$。必定存在这样的 A，使得 $\phi \neq \psi$；否则将意味着 $X_1 = X_2$ 或 X_1（不失一般性）存在一对 (ϕ_i, σ_i)，使得 ϕ_i 不可满足（σ_i 无法由代表函数实现）。后一种可能性与 INF 形式矛盾，前一种可能性则与 $X_1 \neq X_2$ 这一假设矛盾。因此，存在这样一对规则，使得 $\phi \neq \psi$。这意味着存在某个 $A' \in \phi$，但 $A' \notin \psi$（不失一般性）。于是，

$$\rho[\![\hat{X}_1]\!](A') = \sigma = \delta \neq \rho[\![\hat{X}_2]\!](A')$$

由于 X_2 属于 INF，因此它不能有另一条以 δ 为结果状态的规则。这与 $X_1 \sim X_2$ 矛盾，由此得证。 □

推论 4. 对于所有 $X, Y \in Par(\Sigma)$，$X \sim Y$ 当且仅当 $X = Y$。

显然，将 $Par(\Sigma)$ 限制在 INF 状态上，等价于 $Par(\Sigma)/\sim$；其中 $Par(\Sigma)/\sim$ 与 $2^{\mathcal{A}} \to \Sigma$ 之间的同构由 ρ 定义。现在，我们可以在参数化域上使用来自 $2^{\mathcal{A}} \to \Sigma$ 的典范偏序，从而得到一个完备格。

4.3 分析约束函数

如前所述，我们假设已有一种分析，它定义为前向变换器 $\rightharpoonup f$ 的不动点，该变换器可展开为

$$\vec{f}(\sigma_1, \ldots, \sigma_n) = (f_{v_1}(\sigma_1, \ldots, \sigma_n), \ldots, f_{v_n}(\sigma_1, \ldots, \sigma_n)).$$

为了将前向变换器提升到 $Par(\Sigma)^n$，我们首先对所有不是假设语句的 v_i 对约束函数 f_{v_i} 进行平凡提升。也就是说，如果 $v \notin \mathcal{A}$，那么我们将其提升为函数 $pf_v : Par(\Sigma)^n \to Par(\Sigma)$，方法是将 t_v 提升为

$$pt_v(X) = \{(\phi_i, t_{v_i}(\sigma_i)) \mid (\phi_i, \sigma_i) \in X\}$$

也就是说，v_i 以通常转换程序状态的方式转换所有给定规则的结果状态。如果 v_i 是一个假设语句，我们就需要修改规则集合。对于 $a \in \mathcal{A}$，我们将把 $splitPair_a(\phi, \sigma)$定义为满足以下条件的最小函数

$$\begin{aligned} Sat(\phi \wedge (\!|a|\!)) &\implies (\phi \wedge (\!|a|\!), \sigma \sqcap \pi_a) \in splitPair_a(\phi, \sigma) \\ Sat(\phi \wedge \neg(\!|a|\!)) &\implies (\phi \wedge \neg(\!|a|\!), \sigma) \in splitPair_a(\phi, \sigma) \end{aligned}$$

直观地说，如果条件 $\phi \wedge (\!|a|\!)$ 可满足（可实现），那么我们可以非确定性地选择强制采用假设 a。为此，我们更新条件，以表明这条新规则以使用 a为条件，并按前文所述，在结果状态 σ 上强制采用该假设。对于 $\phi \wedge \neg(\!|a|\!)$，也可以采用类似的思路：我们表明并未采用该假设，并保持结果状态不变（因为没有强制采用任何假设）。需要注意的是，$\neg(\!|a|\!)$ 表示在此分支中不考虑 a 这一事实，而不是

说 a 中描述的假设为假。事实上，除非 Σ 是一个布尔格（此时可以通过补集π_p限制程序状态），否则可能无法强制使 a 为假。

综合考察这两种情况，如果 $splitPair_a(\phi,\ \sigma) = \{(\phi \wedge (\!|a|\!),\ \sigma \sqcap \pi_p),\ (\phi \wedge \neg(\!|a|\!),\ \sigma)\}$，显然我们正在对输入规则进行拆分。这表示非确定性选择：要么接受该假设，要么不接受。在实现中，$splitPair_a$ 可以检查 $\phi \Rightarrow (\!|a|\!)$（第二种情况也类似）。如果满足这一条件，就无需合取 $(\!|a|\!)$，从而使条件更简洁、复杂度更低。

为了对参数化状态 $X \in Par(\Sigma)$ 进行拆分，我们定义其自然扩展

$$split_a(X) = \bigcup_{(\phi,\sigma)\in X} splitPair_a(\phi,\sigma)$$

需要注意的是，如果 $\{\phi_i\}_i$ 是 $2^{\mathcal{A}}$的不相交划分，那么 $split_a$ 也会生成构成 $2^{\mathcal{A}}$的不相交划分的条件。这一性质使我们能够放心地断言，$split_a$ 确实是 $Par(\Sigma) \to Par(\Sigma)$ 中的一个映射，而且这一不相交划分要求绝不会被违反。

引理5。对于任意 $a \in \mathcal{A}$和 $X \in Par(\Sigma)$，令 $split_a(X) = \{(\phi'_i, \sigma'_i)\}_i$。那么，$\{\phi'_i\}_i$是 $2^{\mathcal{A}}$ 的不相交划分，因此 $split_a(X) \in Par(\Sigma)$。

证明。令 $X = \{(\phi_i, \sigma_i)\}_i$。那么，$\{\phi_i\}_i$ 是 $2^{\mathcal{A}}$ 的不相交划分。只需证明，对于所有 $A \subseteq \mathcal{A}$，存在唯一的 ϕ'_i 使得 $A \in \phi'_i$。我们先证明存在性。令 ϕ_i 为 $\{\phi_i\}_i$ 中满足 $A \in \phi_i$ 的唯一条件。那么不失一般性，若 $a \in A$，则 $A \in (\phi_i \wedge (\!|a|\!))$，这也意味着 $Sat(\phi_i \wedge (\!|a|\!))$，因此 $\phi_i \wedge (\!|a|\!) \in \{\phi'_i\}_i$，由此可知存在性成立。对于唯一性，注意到由于对于所有 $j \neq i$，$A \notin \phi_j$ $A \notin \phi_j \wedge (\!|a|\!)$ 且 $A \notin \phi_j \wedge \neg(\!|a|\!)$。由于这涵盖了 $\{\phi'_i\}_i$ 中所有其他可能的条件（除 $\phi_i \wedge \neg(\!|a|\!)$ 外；A 不包含在其中，因为 $a \in A$），因此唯一性成立。 □

现在可以定义假设语句的变换器。设 v 为假设语句的控制流图节点。我们定义 pt_v 为

$$pt_v(X) = split_v(X)$$

并将所有节点的约束函数 pf_v 定义为

$$pf_v(X_1,\ldots,X_n) = INF\left(\bigsqcup_{v_i\in pre(v)} pt_v(X_i)\right)$$

也就是说，每当遇到假设语句时，我们就针对所有输入状态对该假设进行非确定性分情况（并压缩为 INF）。新的方程组变为

$$\vec{pf}(X_1,\ldots,X_n) = (pf_{v_1}(X_1,\ldots,X_n),\ldots,pf(X_1,\ldots,X_n))$$

其最小解 $\mathrm{lfp}(\vec{pf}) \in Par(\Sigma)^n$ 表示了 $2^{\mathcal{A}} \to \Sigma$ 中每个控制流图节点所对应的期望函数。现在我们证明这一结论的正确性。

4.4 正确性

设 P 是一个程序。我们用 $P|_A$ 表示 $A \subseteq \mathcal{A}$，意指只保留假设节点 A 的程序 P。也就是说，所有不在 A 中的假设语句都将替换为 **假设** $true$。我们还将用

$[\![P]\!]_{par}$ 表示对分析 $[\![\cdot]\!]$ 进行参数化后所得参数化分析结果的代表函数（如上一节所述）。下述定理指出，对于任意选择的假设 $A \subseteq \mathcal{A}$，参数化分析都能正确计算程序 P 在假设 A 下的静态分析结果。

定理 6。 给定任意程序 P以及任意一组假设 $A \subseteq \mathcal{A}$, $[\![P]\!]_{par}(A) = [\![P|_A]\!]$。

证明。 由于 $[\![P]\!]_{par} = \bigsqcup_{k\geq 0} \vec{pf}^{\,k}(\bot)$ 和 $[\![P|_A]\!] = \bigsqcup_{k\geq 0} \vec{f}_A^k(\bot)$（根据 ω-连续性假设），其中 $\vec{f}_A$ 是 $P|_A$ 的前向变换器，因此只需证明对于所有 $k \geq 0$, $\vec{pf}^{\,k}(\bot) = \vec{f}_A^k(\bot)$。我们将通过归纳法来证明。

基础情形 $k = 0$ 显然成立。令 $k \geq 1$。那么，根据归纳假设，我们有$\rho[\![\vec{pf}^{\,k-1}(\bot)]\!](A) = \vec{f}_A^{k-1}(\bot)$。令 $\vec{f}_A^{k-1}(\bot) = (\sigma_1, \ldots, \sigma_n)$ 和 $\vec{pf}^{\,k-1}(\bot) = (X_1, \ldots, X_n)$，并令

$$\vec{pf}^{\,k}(\bot) = (X_1', \ldots, X_n') = (pf_{v_1}(X_1, \ldots, X_n), \ldots, pf_{v_n}(X_1, \ldots, X_n))$$
$$\vec{f}_A^k(\bot) = (\sigma_1', \ldots, \sigma_n') = (f_{v_1}(\sigma_1, \ldots, \sigma_n), \ldots, f_{v_n}(\sigma_1, \ldots, \sigma_n))$$

我们将证明，对于所有 $i \in [n]$, $\rho[\![X_i']\!](A) = \sigma_i'$。需要考虑两种情况。第一种情况是， v_i 不是假设语句（$v_i \notin \mathcal{A}$）。这里,

$$\sigma_i' = \bigsqcup_{v_j \in pre(v_i)} t_{v_i}(\sigma_j) = \bigsqcup_{v_j \in pre(v_i)} t_{v_i}(\rho[\![X_j]\!](A))$$

根据归纳假设，以及

$$X_i' \sim \bigsqcup_{v_j \in pre(v_i)} pt_{v_i}(X_j)$$

（根据引理2），因此

$$\rho[\![X_i']\!](A) = \bigsqcup_{v_j \in pre(v_i)} \rho[\![pt_{v_i}(X_j)]\!](A) = \bigsqcup_{v_j \in pre(v_i)} t_{v_i}(\rho[\![X_j]\!](A))$$

根据同构 orphism of ρ 以及 $v_i \notin \mathcal{A}$的 pt_{v_i} 定义。因此，在这种情况下， $\rho[\![X_i']\!](A) = \sigma_i'$.

第二种情况是 $v_i \in \mathcal{A}$。对于所有 j，令（ϕ, σ_j）$\in X_j$ 为满足 $A \in \phi$ 的唯一规则（结果状态为 σ_j，因为 $\rho[\![X_j]\!](A) = \sigma_j$）。那么，如果 $v_i \in A$,

$$\sigma_i' = \bigsqcup_{v_j \in pre(v_i)} \sigma_j \sqcap \pi_p$$

and

$$\rho[\![X_i']\!](A) = \bigsqcup_{v_j \in pre(v_i)} \rho[\![split_{v_i}(X_j)]\!](A) = \bigsqcup_{v_j \in pre(v_i)} \sigma_j \sqcap \pi_p$$

如果 $v_i \notin A$，我们同样有

$$\sigma_i' = \bigsqcup_{v_j \in pre(v_i)} \sigma_j$$
$$\rho[\![X_i']\!](A) = \bigsqcup_{v_j \in pre(v_i)} \sigma_j$$

在这两种情况下，$\rho[\![X'_i]\!](A) = \sigma'_i$。 □

备注。 对前向变换器的 ω-连续性要求，有助于证明分析相对于其所分析的域是精确的，而且并非限制性要求。然而，当分析希望使用拓宽等无法计算前向变换器最小不动点的技术时，这一要求似乎具有限制性。尽管我们没有对此进行完全形式化，但正如证明中所示，这些情况实际上并不会破坏定理，因为在 A 下，参数化分析与原始分析无论前向变换器受到何种限制，都会“同步”进行。在这些情况下，虽然它们不会像定理所假设的那样计算最小不动点，但只要它们是迭代的，分析结果就会相同。

4.4.1 抽象解释器情形下的可靠性

在实践中，我们希望对抽象域而非具体域进行参数化。当然，我们可以简单地用抽象域 $\Sigma^\sharp$替换 Σ 。我们应当确保所得的参数化分析结果相对于具体域下的参数化分析结果是可靠的，其中 $\alpha : \Sigma \to \Sigma^\sharp$ 是相应伽罗瓦连接的抽象函数。

设 $[\![\cdot]\!]^\sharp$ 为一次可靠抽象解释的结果，并设 $[\![\cdot]\!]^\sharp_{par}$ 为对该可靠抽象解释进行参数化后得到的分析结果。

定理 7. 给　　定任意程序 P 以及任意假设集合 $A \subseteq \mathcal{A}$，　$\alpha([\![P|_A]\!]) \sqsubseteq [\![P]\!]^\sharp_{par}(A)$

证明。由 Theo 得出 由 6，我们有 $[\![P]\!]^\sharp_{par}(A) = [\![P|_A]\!]^\sharp$。然后，由于 $[\![\cdot]\!]^\sharp$相对于 $[\![\cdot]\!]$是可靠的，我们有 e

$$\alpha([\![P|_A]\!]) \sqsubseteq [\![P|_A]\!]^\sharp = [\![P]\!]^\sharp_{par}(A)$$

□

4.5 近似合并

正如前文经常提到的，在保持精度的同时，我们无法避免指数级增长。因此，我们将牺牲一些精度，以换取亚指数级的状态表示。

在第 4.2 节中，我们讨论了将参数化状态 $X \in Par(\Sigma)$ 转换为等价且更紧凑形式（INF）的方法。我们将介绍一种类似于精确合并的新归约操作，但它适用于任意一对规则。

近似合并。 给定两条规则 $(\phi_i,\ \sigma_i)$，$(\phi_j,\ \sigma_j) \in X$，我们可以将它们合并为一条规则 $(\phi_i \vee \phi_j,\ \sigma_i \sqcup \sigma_j)$。直观而言，这条规则表示：“如果 A 满足 ϕ_i 或 ϕ_j，那么它的结果状态要么是 σ_i，要么是 σ_j。”当然，由于合并状态无法还原出两个原始状态，因此会丢失部分信息。

应用近似合并的结果是初始状态的过近似。

定理 8. 设 $X \in Par(\Sigma)$ 非空。然后，对 X 中的任意两条规则应用近似合并，会得到一个状态 X'，使得 $X \sqsubseteq X'$。

证明。 假设我们对 $(\phi_i, \sigma_i), (\phi_j, \sigma_j) \in X$应用近似合并，得到 $(\phi_i \vee \phi_j, \sigma_i \sqcup \sigma_j)$。那么，对于所有 $A \subseteq \mathcal{A}$，若 $A \in \phi_i \vee \phi_j$，显然有

$$\rho[\![X]\!](A) \sqsubseteq \sigma_i \sqcup \sigma_j = \rho[\![X']\!](A)$$

另一方面，如果 $A \notin \phi_i \vee \phi_j$，那么 $\rho[\![X]\!](A) = \rho[\![X']\!](A)$。 □

可以随意决定是否应用近似合并，但更好的做法是明智地应用。理想情况下，我们希望尽可能少地损失精度。假设我们有某个函数 $Loss(\sigma_i, \sigma_j)$，它可以衡量这些输入状态与其连接 $\sigma_i \sqcup \sigma_j$ 之间损失的精度。理想情况下，在执行近似合并时，我们希望选择能使$Loss(\sigma_i, \sigma_j)$最小的 (ϕ_i, σ_i),(ϕ_j, σ_j)。未来的工作可以考虑设计数据结构，利用某些域的结构，高效地支持此操作。

补充说明。 近似合并的使用可以与拓宽相比较。拓宽牺牲精度以换取终止性和速度，而近似合并则牺牲精度（但当 $\mathcal{A}$ 有限时，不会影响终止性）。幸运的是，在大多数应用中，假设的数量足够少，而且 INF 约简能够完成足够多的工作，从而使表示的计算开销保持在合理范围内。因此，在大多数情况下，近似合并是一种约简，只需在开发者确定的某个阈值处保守地应用。还应谨慎避免过早应用近似合并，因为过早使用可能导致精度损失，而这些损失本可在后续迭代中通过精确合并避免。例如，假设 X 中包含的规则过多，因此应用了近似合并。如果没有应用近似合并，那么再迭代一次后， X 中的几乎所有规则都有可能获得相同的结果状态，从而实现大规模的精确合并。这样一来，这次近似合并就可谓徒劳无功。这与过早应用拓宽所造成的后果类似。

5 假设合成

虽然非确定性程序假设下的静态分析可用于许多场景，但本节将探讨一个示例用例。考虑一个包含多个断言语句的程序。我们的目标是找到一组假设 $A \subseteq \mathcal{A}$，使每个断言都得到满足（如果存在这样的一组假设）。

在完成分析并获得$[\![P]\!]_{par}$后，这个问题的解决方案相当直接。令 v 为断言语句 **assert** φ_v对应的控制流图节点， X_v 为 v处的分析结果。只需找到一条规则 $(\phi_i, \sigma_i) \in X_v$，使得 $\sigma_i \models \varphi$。然后，对于所有 $A \in \phi_i$，采用假设 A 即可使程序满足该位置处的断言。一般而言， ϕ_i 描述了这样一类假设集。要找到一个满足所有 断言的假设集（如果存在），我们先找出满足每个单独断言的所有假设集，再从中找出一个共同的假设集。更正式地说，能够（根据分析结果）证明满足所有断言的假设集为

$$Syn(P) = \bigcap_{\text{assertions } v} \bigcup_{\substack{(\phi_i, \sigma_i) \in X_v \\ \sigma_i \models \varphi_v}} \phi_i$$

这个问题的解决方案之所以如此简单，是因为我们在需要确定任何假设之前，已经掌握了每个假设集的保证。假设不再需要“推导”。相反，我们一开始就拥有所有必要信息（当然，在有限的用户提供的候选假设集上进行分析，也大大简化了问题）。然而，使用可靠的过近似（例如抽象解释或大量使用近似合并所产生的过近似）时，分析提供的信息可能过于粗略，无法证明或证伪断言。在这种情况下，我们可以给出“未知”的结论。这里，我们维持了可靠性：只要返回一个假设集，它就确实能证明断言为真。否则，我们可以输出“未知”（或者，如果能够证伪所有假设集对应的断言，则可以输出“不可能”）。

下面的定理通过指出，返回的任何一组假设都确实能在原始分析的语义下使断言为真，从而将这一点形式化 $[\![\cdot]\!]$。下面的定理以 $Syn(P)$ 非空为前提。

定理 9。 设 $A \in Syn(P)$ 和 $[\![P|_A]\!] = (\sigma_{v_1}, \ldots, \sigma_{v_n})$。那么，对于所有表示断言 断言 φ_{v_i} 的控制流图节点 v_i， $\sigma_{v_i} \models \varphi_{v_i}$。

证明。设 v 是断言语句 assert φ_v 所对应的控制流图节点。那么，由于 $A \in Syn(P)$，我们有 $A \in \cup\{\phi_i \mid (\phi_i, \sigma_i) \in X_v \wedge \sigma_i \models \varphi_v\}$，其中 X_v 是 $[\![P]\!]_{par}$ 中 v 处的分析结果。因此， $\exists(\phi_i, \sigma_i) \in X$，使得 $\sigma_i \models \varphi_v$ 且 $A \in \phi_i$。由于 $A \in \phi_i$，我们有 $\rho[\![X_v]\!](A) = \sigma_i$，因此 $\rho[\![X_v]\!](A) \models \phi_i$。根据定理 6， $\rho[\![X_v]\!](A) = \sigma_v$，所以 $\sigma_v \models \varphi_v$。 □

考虑一个经典问题：“为给定程序 P 寻找一个前置条件，使得给定的后置条件 φ得到满足。”一般来说，这是此前讨论的问题的一个较弱版本。虽然这两个问题可以相互编码为对方的形式，但经典问题需要使用析取及其他额外的复杂机制来编码路径依赖，而且无法恢复路径信息。另一方面，我们讨论的问题能够自然地处理路径依赖和路径信息，因为 assume/assert 语句可以放置在程序的任何位置，甚至是条件语句内部。因此，assume/assert 语句本身就包含路径信息，无需使用析取来表示经典问题必须表示的语句。这表明，我们的参数化分析通常适用于范围更广的问题，因为许多问题都可以编码为我们所描述的假设综合问题形式，而不会增加额外的复杂性。

备注。 假设和断言在表达能力上有所不同。假设仅限于分析域（抽象域）中的程序状态，而断言可以是任何能够由程序状态证明 的内容。例如， $Interval$ 域无法表达假设 $x \leq y$，但在程序状态满足 $[x \mapsto [a, b], y \mapsto [c, d]$和 $b \leq c$时，它可以证明 断言 $x \leq y$。

请注意，只有分析具有足够的精度，才能完成这些证明；而通过用假设强化程序状态，我们可以提高精度。因此，精度和证明能力受两方面影响：假设和所选的抽象域。假设可以提供足够的精度来证明更强的性质，而抽象域则可以提供足够的表达能力来证明更多种类的性质。

最后需要指出的是，这个问题所表达的内容很重要：令 $A \in Syn(P)$。那么，如果 假设 A 为真，那么所有断言都为真。这是一个条件

语句，其中 A 中假设的有效性未必得到保证。由于问题是以这种方式表述的，静态分析中的不可靠性不会造成问题。然而，如果这些假设与真实的可达程序状态相矛盾，那么了解这一点可以帮助开发人员在大型代码库中发现关键问题，甚至证明给定假设的互斥性质。在下一节中，我们将介绍一个有原则的框架，用于消除这类矛盾。

6 一致假设集

如前所述，不可靠性并不总是个问题，因为参数化分析结果的形式为"如果 假设 A 为真，那么 v 处的程序状态为 σ。"不过，有时消除假设的不一致 集合也很有用。本节将说明，参数化分析提供了一种简便的方法来实现这一点。

如果静态分析在 A 下既不能证伪 A 中的任何假设，也不能允许在 $\mathcal{A}$ 中加入任何新假设，我们就称假设集 $A \subseteq \mathcal{A}$ 为一致。令 $\Phi(A)$ 表示 $\mathcal{A}$ 中在 A 下的静态分析未能驳斥的假设集。一致集于是 Φ 的不动点。更正式地说，对于语句**假设** p，令

$$\zeta_p(X) = \{\phi \mid (\phi, \sigma) \in X \wedge \neg Feasible_p(\sigma)\}$$

这是 X 中使得相对于假设 p 而言程序状态不可行的条件集。于是，驳斥 假设 $a \in \mathcal{A}$ 的假设集为

$$B_a(X_{v_1}, \ldots, X_{v_n}) = \bigcup_{\phi_i \in \zeta_p(X_a)} \phi_i$$

其中 a 是 语句对应的控制流图节点 **假设** p。最后， Φ可形式化定义为 s

$$\Phi(A) = \{a \in \mathcal{A} \mid A \notin B_a([\![P|_A]\!])\}$$

也就是说， $\Phi(A)$ 是未被 A 驳斥的假设集。仔细考察这个函数可知， Φ 是反单调的。这是因为更大的 A 会产生更精确的分析（对所有 $a \in \mathcal{A}$而言，更精确的 X_a ），从而更容易驳斥更多假设（$Feasible$的单调性），因此剩余的假设更少。其对偶陈述也成立。

由于 Φ 是反单调的， Φ^2 是单调的，因此根据 Knaster-Tarski 不动点定理，它存在一个最小不动点。此外，已知 $\text{lfp}(\Phi^2) \subseteq \text{fix}(\Phi)$ 且 $\text{fix}(\Phi) \subseteq \Phi(\text{lfp}(\Phi^2))$（并且 $\Phi(\text{lfp}(\Phi^2)) = \text{gfp}(\Phi^2)$）。这为 Φ （一致假设集）的不动点提供了上界和下界。令 $\mu = \text{lfp}(\Phi^2)$， $\nu = \text{gfp}(\Phi^2)$。这些界使我们能够作出某些推断，例如：

- 如果 $a \in \mu$，那么 a 会出现在每一个一致的假设集中。
- 如果 $a \in \nu$，那么 a 会出现在某个一致的假设集中。
  - 如果 $a \notin \nu$，那么 a 永远不会出现在任何一致的假设集中。

这提供了一种有理有据的方法，引导工具采用一致的假设。请注意，这种不动点计算之所以可行，只是因为我们在 $2^{\mathcal{A}} \to \Sigma$ 中得到了相应的分析函数。

否则，计算 $\Phi(A)$ 就需要在 A 下重新运行分析，而这通常不可行（因为 Φ 可能需要迭代许多次才能达到 $\mathrm{lfp}(\Phi^2)$）。

还需注意，μ 和 ν 可能形成一个非常宽松的界限。事实上，即使合并了不动点，$(\mu,\ \nu)$ 也可能跨越整个格。已有研究探讨了如何高效地获得更紧的界限 [2]„ 从而使这种一致性搜索更可行、更实用。

7 相关工作

提升式静态分析旨在解决类似的问题，即分析以环境为参数的程序。提升式静态分析的主要应用领域是软件产品线（SPL），它通过提升分析来支持语法层面的编译时变化性，将其作为静态配置选项“ifdef FEATURE” [7, 5, 9]。在 [8]，中，配置选项通过将特性视为一等程序变量而放宽为动态选项。在这两种情况下，这些工作处理的都是特性，而我们的工作处理的是假设。两者存在一个关键区别：提升式静态分析中的特性通常是全局变量或作用域变量，旨在充当“配置”；而我们工作中的假设是在任意程序点引入的语义约束。这些并非程序员设计的“变量特性”，而是由分析器以非确定性方式处理的形式约束。此外，我们方法中的一致性取决于语义，而非固定的“特性模型”。我们的近似合并还支持对假设路径进行可靠合并，并提供精度/可扩展性调节参数。这些差异体现了从软件工程向形式化验证的一项重要技术迁移。

轨迹划分是一种经过充分研究的方法，通过根据控制流决策的历史区分执行轨迹，来缓解精度损失 [16]。我们将这种划分理念从控制流扩展到了假设空间。轨迹划分根据条件语句（如 if (c) ...）划分状态，而我们的方法则根据假设（如 assume (P) ...）划分状态。这种划分使分析状态不仅取决于程序的内部历史，还取决于额外的“假设行为”。这标志着一种转变：从优化 控制 转向优化上下文。

假设-保证（AG）推理中的假设合成已有多种方法，各有优点。一个著名示例是基于学习的假设-保证自动化 [4, 3, 14, 12], ，它在学习器与教师之间通过“猜测并检验”的循环，将假设学习为正则语言。尽管已有一些优化，这种学习过程的计算开销仍然很大，而且主要适用于有限状态性质。我们的方法消除了这类学习算法中常见的迭代循环（通常是 L^* 学习），转而执行符号化假设传播。通过在非确定性假设下对组件进行参数化分析（AG 致力于验证系统各组件的性质），我们的方法计算出组件行为相对于假设空间的完整函数。随后，证明整个系统性质的任务便从模型检测转变为对各组件参数化结果进行一致性检查（通过域交、可满足性或其他方式）。因此，相较于标准技术的“黑盒”学习，我们的方法是一种“白盒”方法。它利用源代码的静态分析，明确推导出假设与保证之间的依赖结构。

我们将分析应用于假设合成，这与现有的溯因推理指导方法 [18, 1, 6]相近。虽然与这些方法相比，我们的方法在精度/可扩展性权衡方面更为严格，但它保留的是一组充分条件，而非单一的最弱条件。

通过跟踪条件依赖在每个程序点的演变情况，我们实际上维护了一个由抽象语义假设（而非原始逻辑）参数化的“运行中最弱前置条件”。这也体现了我们的方法与其他模型检测指导方法[21, 11]之间的差异。这一区别也揭示了不同类型方法之间的权衡。

8 结论

我们定义了一种基于参数化抽象域的分析，它等价于一个函数：将程序假设集映射为相应假设下的分析结果。用户在程序位置提供这些假设，分析器则以非确定性方式处理它们。为使参数化状态表示在计算上可行，我们在参数化域上强制采用唯一的紧凑表示。这也使其具备完备格的性质。我们提出了一种对参数化状态进行近似的方法，以换取更高的可扩展性，从而允许开发者自行调节精度/可扩展性。为展示参数化分析的实用性，我们说明了它如何通过将假设综合问题转化为搜索空间上的优化问题来简化该问题，而无需再发现这个搜索空间。我们还展示了分析结果具有函数形式这一特点如何支持有效的不动点计算，并利用这一点来指导对一致的假设集的搜索。

参考文献

[1] Aws Albarghouthi、Isil Dillig 与 Arie Gurfinkel。最大规范合成。*SIGPLANNot.*，51(1):789–801，2016年1月。[2] 匿名作者。论文标题，2026年。审稿中的稿件。[3] Colin Blundell、Dimitra Giannakopoulou 与 Corina S. Pˇasˇareanu。假设-保证测试。载于*2005*年基于组件系统规范与验证会议论文集，SAVCBS ’05，第1–es页，美国纽约州纽约，2005年。计算机协会。[4] Sagar Chaki 与 Ofer Strichman。基于优化 l* 的假设-保证推理。载于 Orna Grumberg 与 Michael Huth 编，系统构造与分析工具及算法，第276–291页，柏林，海德堡，2007年。Springer Berlin Heidelberg。[5] Sheng Chen 与 Martin Erwig。基于类型的程序族参数化分析。载于第*19*届 *ACM SIGPLAN* 函数式编程国际会议论文集，ICFP ’14，第39–51页，美国纽约州纽约，2014年。计算机协会。[6] Isil Dillig、Thomas Dillig 与 Alex Aiken。使用溯因推理的自动错误诊断。*SIGPLAN Not.*，47(6):181–192，2012年6月。[7] Aleksandar S. Dimovski。使用二元决策图抽象域的提升式静态分析。载于第*18*届 *ACM SIGPLAN* 生成式编程国际会议：概念与经验论文集，GPCE 2019，第102–114页，美国纽约州纽约，2019年。计算机协会。

[8] Aleksandar S. Dimovski 与 Sven Apel。基于抽象解释的动态程序族提升式静态分析（工件）。*Dagstuhl* 工件系列，7(2):6:1–6:6，2021年。[9] Aleksandar S. Dimovski、Sven Apel 与 Axel Legay。用于数值程序族静态分析的多个提升式抽象域。《计算机编程科学》，213:102725，2022年。[10] Roberto Giacobazzi 与 Francesco Ranzato。析取抽象解释的最优域。《计算机编程科学》，32(1):177–210，1998年。第六届欧洲编程研讨会。[11] Dimitra Giannakopoulou、Corina S. P˘as˘areanu 与 Howard Barringer。使用自动生成的假设进行组件验证。《自动化软件工程》，12(3):297–320，2005年7月。[12] Wonhong Nam 与 Rajeev Alur。带自动分解的基于学习的符号化假设—保证推理。载于 Susanne Graf 与 Wenhui Zhang 编，自动化验证与分析技术，第170–185页，柏林，海德堡，2006年。Springer Berlin Heidelberg。[13] D. L. Parnas。论程序族的设计与开发。*IEEE Trans. Softw. Eng.*，2(1):1–9，1976年1月。[14] Corina S. P˘as˘areanu、Dimitra Giannakopoulou、Mihaela Gheorghiu Bobaru、Jamieson M. Cobleigh 与 Howard Barringer。学会分而治之：应用 L*算法实现假设-保证推理的自动化。《系统设计中的形式化方法》，32(3):175–205，2008年6月。[15] Corneliu Popeea 与 Wei-Ngan Chin。推断析取后置条件。载于 Mitsu Okada 与 Ichiro Satoh 编，计算机科学进展——*ASIAN 2006*。安全软件及相关问题，第331–345页，柏林，海德堡，2007年。Springer Berlin Heidelberg。[16] Xavier Rival 与 Laurent Mauborgne。轨迹划分抽象域。*ACM Trans.Program. Lang. Syst.*，29(5):26–es，2007年8月。[17] Sriram Sankaranarayanan、Franjo Ivan˘ci´c、Ilya Shlyakhter 与 Aarti Gupta。析取数值域中的静态分析。载于 Kwangkeun Yi 编，静态分析，第3–17页，柏林，海德堡，2006年。Springer Berlin Heidelberg。[18] Simon Spies、Lennard G¨aher、Michael Sammler 与 Derek Dreyer。Quiver：Coq 中分离逻辑规范的引导式溯因推理。*Proc. ACM Program. Lang.*，8(PLDI)，2024年6月。[19] V. Stoltenberg-Hansen、I. Lindstr¨om 与 E. R. Griffor。域的数学理论。剑桥理论计算机科学丛书。剑桥大学出版社，1994年。[20] Alfred Tarski。格论不动点定理及其应用。《太平洋数学杂志》，5:285–309，1955年。[21] Dong Wang 与 J. Levitt。面向基于断言的形式化验证的自动假设-保证分析。载于*ASP-DAC 2005*论文集。亚洲及南太平洋设计自动化会议，*2005*年。，第1卷，第561–566页，第1卷，2005年。

[22] Han Xu、Liqian Chen、Guangsheng Fan、Banghu Yin 和 Ji Wang。具有析取式右端区间的线性模板抽象域。载于 可信赖软件工程。理论、工具与应用：第九届国际研讨会，*SETTA 2023*，中国南京，*2023*年*11*月*27*日至*29*日，论文集，第314–330页，柏林、海德堡，2023年。施普林格出版社。