arXiv:2601.14059v1 [cs.PL] 20 Jan 2026

在 Stainless 中验证浮点程序

Andrea Gilot*✉ Axel Bergström* Eva Darulova

乌普萨拉大学
{andrea.gilot,axel.bergstrom,eva.darulova}@it.uu.se

摘要。 我们为 Stainless 演绎式验证器扩展了浮点支持，为包含多态、递归和高阶函数的 Scala 子集提供了首个针对浮点数的自动化验证支持。我们采用 KeY 验证器中的最新方法，对数学函数的推理进行公理化；但我们更进一步，支持 Scala 数学 API 中的所有函数，并在 Stainless 自身中验证这些公理相对于实际实现的正确性。我们在一组新的基准测试上验证了 Stainless 的浮点支持能力。这些基准测试取样自 GitHub 上的真实世界代码，结果表明，对于大多数受支持的函数，该工具能够验证有关输出值域或不存在特殊值等方面的规格说明；当规格说明不成立时，还能生成反例。

关键词： 浮点算术 · 超越函数 · 演绎式验证

1 简介

浮点算术在现代软件中无处不在。尽管人们常在科学计算、嵌入式系统和机器学习的语境下讨论它，但它首先广泛存在于日常用户代码中 [18]。然而，验证工具并非总是支持浮点推理，这限制了它们在实践中的应用。例如，LiquidHaskell [40] 缺少对浮点数的支持，这已被认为是其在现实世界中适用性的一项限制 [39]。

本文将位精确浮点推理引入 Stainless [20]，这是一个用于 Scala 程序的演绎式验证框架。此次集成面临的两大主要挑战是正确性和性能。为确保正确性，实现必须仔细处理 JVM 与 SMT-LIB 之间的语义差异，以及相等的非标准行为。

此外，已知在 SMT 求解器中对浮点算术进行推理代价高昂；因此，在条件允许时，需要使用专门的 SMT-LIB 结构谨慎地编码验证条件。

我们首先复现了 KeY 中近期引入的浮点支持 [1], ，以此证明这种方法可以推广到其他验证器。我们的集成将这项支持扩展到了 Stainless 所处理的 Scala 语言子集，

* 这些作者对本研究作出了同等贡献。

使得验证程序能够将浮点计算与高阶函数、代数数据类型等特性结合起来，而这些特性在 KeY 中并不受支持。我们解决了将通常被假定为自反的多态相等与浮点推理结合起来的难题；在浮点推理中，这一假设并不成立（因为 `NaN != NaN`）。忽略这一不匹配可能导致不健全性，因为针对多态类型证明的定理可能会被不安全地实例化为浮点类型。

我们为 Stainless 实现了一个标准数学库，其 API 与 Scala 标准库 API 一致，并在*Stainless* 本身中验证了此前仅在 KeY 中公理化或尚不支持的规格说明例如，规定 `sin` 返回的值介于 -1 和 1 之间。虽然这些规格说明并不完整，但有时足以用于验证包含超越函数的用户代码。SMT 求解器不支持超越函数，而内联数学库函数的实现通常会导致求解时间长得无法接受。

为尽量减少用户注释工作量，Stainless会自动生成验证条件，以捕获浮点行为中一些微妙之处；这些特性有悖直觉，即使是经验丰富的开发人员也常常会误解 [14]。具体而言，它会强制执行浮点比较的非-`NaN` 前置条件，并验证浮点类型与整数类型之间类型转换的安全性。

我们使用 KeY 提出的基准测试评估 Stainless [1]，结果表明其浮点推理支持能力相当。我们还使用从开源 GitHub 仓库中挖掘出的 103 个真实世界浮点函数评估 Stainless，找出实际应用场景以及仍待解决的挑战——无论对 Stainless 还是其他工具而言，全面支持浮点验证都面临这些挑战。

我们的结果表明，Stainless 成功验证了来自 KeY 的所有基准测试，并对 82% 的用户基准测试完成了验证或生成了反例。通过数学库中已验证的性质对超越函数进行推理，足以处理一半以上的这些用户基准测试，这既凸显了该方法在实践中的有效性，也揭示了其当前的局限性。最后，针对反直觉浮点行为的自动检查能够检测出用户代码中所有经检查的问题，因而有效识别潜在漏洞。

贡献。总而言之，我们做出了以下贡献：

– 我们为 Stainless 验证器扩展了浮点数支持[1]，将近期的 SMT 求解器 Bitwuzla [33] 用作专门进行浮点推理的后端，并加入了实用性功能，例如自动检查违反直觉的浮点行为（第 3 节）。– 我们为 Stainless 实现了一个标准数学库，其实现紧密遵循 OpenJDK 数学库，并使用 Stainless 本身验证了该库的数学性质（第 4 节）。– 我们在 KeY 提出的基准测试集以及从 GitHub[2] 收集的用户编写浮点代码上评估了 Stainless（第 5 节）。

[1] 已合并至 Stainless 主代码库（`https://github.com/epfl-lara/stainless`）。

[2] `https://github.com/fxpl/stainless-float-benchmarks`

```scala
case class StormDay(moves: Int, errors: Int) {
  require(moves >= errors && errors >= 0 && moves > 0)  // class invariant

  def accuracyPercent: Float = {
    100 * (moves - errors) / moves.toFloat
  }.ensuring(result => 0 <= result && result <= 100)  // postcondition
}

def gradient(prediction: Double, label: Double): Double = {
  require(prediction.isFinite && label.isFinite)  // precondition
    - 4.0 * label / (1.0 + math.exp(2.0 * label * prediction))
}.ensuring(result => !result.isNaN)

def limit(maxMagnitude: Double): Vector = {
  if (getMagnitude > maxMagnitude) // NaN check inserted here
    getNewMagnitude(maxMagnitude)
  else Vector(x, y)
}
```

图 1：从 GitHub 提取的浮点程序示例

2 概述

此前，Stainless 不支持浮点推理。本节将概述 Stainless，以及本工作引入的浮点支持。Stainless 是一个面向 Scala 程序的基于契约的验证器 [20]。每个 Stainless 程序都是有效的 Scala 代码，并且可以像其他 Scala 程序一样在 JVM 上编译和执行。图 1 中的类 `StormDay` 展示了 Lichess 代码库中的一个片段；Lichess 是一个在线国际象棋平台。函数`accuracyPercent` 通过统计一系列国际象棋走法中的正确走法数量来计算玩家的准确率。在假设走法数量严格为正且大于错误数量的情况下（这一条件在 **`require`** 子句的类不变式中表达），我们可以在后置条件（**`ensuring`**子句）中规定，结果必须位于区间 $[0, 100]$ 内。

在验证过程中，Stainless 会生成由 SMT 求解器求解的验证条件。在本例中，Stainless 报告了反例`StormDay(1073741832, 730144766)`；对于该反例，函数返回 `-2.9586256E-5`，违反了后置条件。出现这一结果的原因是，左侧乘法使用整数算术，发生溢出后变为负值，随后又进行了除法。将 `accuracyPercent` 的函数体重写为 `100 * ((moves - errors)/ moves.toFloat)`，会强制先执行浮点除法，再乘以 100，从而保证结果始终处于预期值域内。

图 1 中的函数 `gradient` 实现了 Apache Spark [3]，中的梯度计算。Apache Spark 是一款用于大规模数据处理的 Scala 框架，也是 GitHub 上最受欢迎的 Scala 库。对于这个示例，我们指定输入均为有限值，即，它们既不是浮点特殊值 `+/-Infinity`

也不是 NaN。例如，溢出会导致浮点值变为无穷大，而数学上无效的运算则会产生“非数”值，例如， sqrt(-1.0)。Stainless 尝试验证该函数不会返回 NaN，这可以理解为它不会遇到错误。在运行这段代码时，Stainless 报告了一个反例：label 被赋予了一个非常大的值（与 2 相乘时会导致溢出），而 prediction 被设为 0，进而导致 Infinity * 0 = NaN。再次将乘法从左结合改为右结合即可解决问题，并确保符合规范，这一点已由 Stainless 验证。

这个示例给验证器带来了两项挑战：检测 NaN 值，以及指数函数的存在。检测 NaN 值对开发者和静态分析器而言都很困难，因为要精确刻画这些值，需要进行位级推理。若涉及超越函数，难度会进一步增加；本例中的 **math.exp** 就是如此。这些函数并未得到 SMT 求解器的原生支持，而且其实现通常过于复杂，无法内联。

Stainless 支持对 Scala 标准库中的数学函数进行推理，例如三角函数、指数函数或对数函数。沿用 KeY 的方法 [1], ，这些函数被建模为公理化的无解释符号。这种抽象使 Stainless 无需完整的语义建模，即可对特殊值（例如 NaN）的出现进行基本推理。与 KeY 不同，我们使用 Stainless 本身验证这些公理与实际实现的一致性——这也对其浮点支持进行了压力测试，因为这些函数在本质上具有很强的数值特性；更多详情请参见第 4 节。

特殊值相关的漏洞往往源于缺少 NaN 检查 [13]。例如，NumPy 先前的 max 函数中存在一个漏洞：当输入数组中出现 NaN 值时，该函数会返回一个任意数，而不是最大值。这类漏洞的根源在于 NaN反直觉的行为：任何涉及NaN 的比较结果都为假，包括 NaN == NaN。为避免此类错误未被发现，Stainless 会自动为所有比较和相等操作插入 NaN 检查，确保两个操作数都不是 NaN。

图 1 中的函数 limit 会在二维向量的模长超过给定阈值时将其限制在该阈值内。如果输入的阈值为 NaN，比较结果会为假，导致函数返回原向量，从而实际上忽略了无效输入。插入 NaN检查后，Stainless 会自动检测此类情况并报告相应的反例。如果有意进行与NaN的比较，用户可以显式禁用这些检查。

Stainless 还会在将浮点值转换为整数时插入检查，以验证输入参数是否处于正确的值域内。在 Scala 中，类型转换 NaN会得到 0，而超出值域的值会被限制为可表示的 32 位或 64 位整数的最大值或最小值。不过，对于 16 位和 8 位整数，超出值域的值会按目标值域取模回绕，从而引入另一种非预期行为。

我们的示例和评测基准中检查的大多数属性都涉及范围分析、符号确定以及特殊值检测。Stainless 也能够推理浮点数等式，但这类情况较为少见。

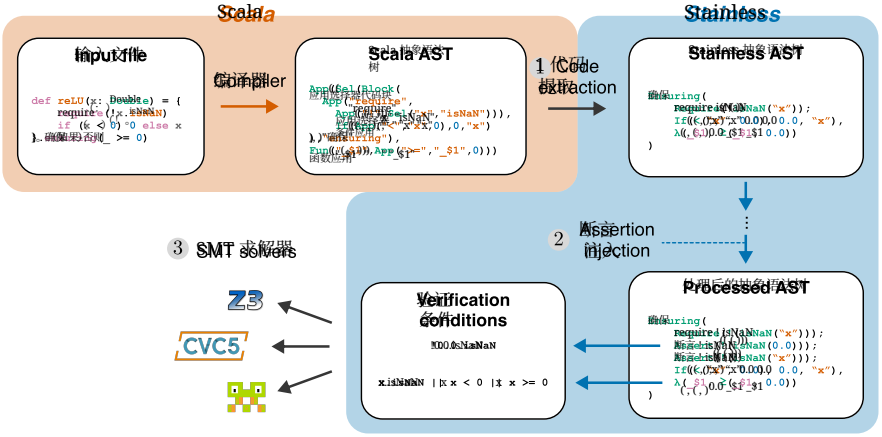


图 2：Stainless 的简化架构

这主要是因为，在理想规格中对实数而言相等的表达式，通常会由于舍入而在浮点数上不再相等。

除了这些高层次的浮点数属性之外，Stainless 还能够推理浮点值的位级属性。第4节中的案例研究要求我们进行推理，例如，指数上的位级操作如何影响相应的浮点值。

3 Stainless 中浮点数支持的实现

我们在 Stainless 中实现了这一扩展。Stainless 采用 Scala 实现，因此我们尽可能复用现有基础设施。图2展示了 Stainless 的概览。Stainless 处理 Scala编译器生成的抽象语法树。随后，它分多个阶段处理这些抽象语法树，包括反别名处理、类型编码和终止度量推断。在流水线末端，Stainless 生成验证条件，并将其交由 SMT 求解器求解。需要注意的是，Scala编译器保留浮点数语义（除编译器实现中的漏洞外），不会因浮点数舍入的变化而影响验证结果。我们的扩展复用了大部分现有基础设施，只对流水线中的少数阶段进行了修改，从而体现了 Stainless 基础设施的模块化特性。

现在，我们将详细介绍为支持浮点运算而对 Stainless 所做的修改，这些修改既保证了可靠性，也提升了效率；我们认为，它们对其他工具和开发者最有参考价值。

代码提取。（1）过程的一部分是将 Scala 抽象语法树降低为 Stainless 抽象语法树。Scala 提供高级构造，而 Stainless 使用专为验证定制的中间表示，因此需要显式编码。默认情况下，Stainless 会内联非递归表达式，并表示

算术、比较和类型转换等基本操作时使用专用节点。然而，SMT-LIB 定义了更复杂的运算符，例如平方根、最小值/最大值和舍入（`ceil`、 `floor`、 `rint`）。将 Scala 调用直接映射到这些运算符可以减少验证条件的规模和复杂性，因此我们会对这类函数禁用内联，并直接对其进行转换。例如，Scala 通过将输入转换为二进制、应用掩码，再重建浮点值来实现浮点绝对值函数。我们不内联这种位级实现，而是将对 `abs` 的调用直接映射到相应的 SMT-LIB 运算符。

JVM 与 SMT-LIB 之间的语义差异也带来了挑战。其中一个差异涉及取模运算符：在 Scala 中， `14.5 % 1.5` 的求值结果为 `1.0`，而在 SMT-LIB 中则得到 `-0.5`。修正这一差异（例如通过添加偏移量）会引入舍入误差，因此 Stainless 目前不支持浮点取模。为了评估这一限制的影响，我们分析了从 GitHub 挖掘出的 30,421 个浮点函数（第 5.1 节）。其中只有 108 个（0.35 %）使用了取模，通常参数为 1（用于提取小数部分）或 360（用于角度计算）。在这些情况下，如果愿意接受不同的浮点舍入行为，可以用 `x - n * floor(x / n)` 替换 `x % n` ，从而避免取模运算（尽管这两个表达式在实数范围内数学等价，但它们在浮点算术中的舍入误差累积方式不同）。由于 Scala 运行在 JVM 上，KeY 也受到同样的限制，因此同样不支持浮点取模。

`NaN` 值的表示方式存在第二个差异。在 IEEE 754 中，`NaN`的所有指数位都被设为 1，且有效数中有一个或多个非零位。这会产生多个 `NaN` 值的编码，其中变化的有效数位构成负载。JVM 会保留这些负载，而 SMT-LIB 使用单一的规范 `NaN` 值。因此，公开浮点数二进制编码的函数在语义上会产生分歧，因为 SMT-LIB 不区分不同的二进制 `NaN`编码。为避免不健全性，Stainless 将二进制转换建模为未解释函数；当其结果被转换回浮点数值时，该结果等于原始浮点数值。

相等与多态。 SMT-LIB 标准区分按位相等和 IEEE 754 相等： `+0` 等于 `-0` ，而 `NaN` 不等于任何值，包括其自身。相等关系的非自反性在与参数多态结合时会带来问题。在 Stainless 中，对类型变量进行推理依赖于这样的假设：相等关系具有自反性、对称性和传递性。当这类类型变量被实例化为浮点数类型，或被实例化为重载了相等运算的用户定义类型时，这些假设可能导致不健全性。例如，Stainless 可能会错误地证明 `NaN == NaN`。

一种可行的解决方案是放弃所有关于参数化相等性的假设，转而将其视为无解释符号，就像 Flux [26] 中的做法一样。尽管这能保证可靠性，但会破坏向后兼容性，因为许多此前在 Stainless 中验证通过的程序将无法再通过验证。

我们采用一种更务实的方法。默认情况下，Stainless 只会将类型变量实例化为其相等满足自反性、对称性和

传递性的类型。如果用户希望使用 `Float`或 `Double`实例化多态表达式，就必须为类型变量添加注解 `@noeq`。随后，Stainless 会将带注解类型变量上的相等视为未解释谓词，从而避免因假定相等具有代数性质而导致的不健全性。

即使不考虑 `NaN`S，IEEE 754 相等也不一定意味着行为等价。例如， `+0.0`和 `-0.0` 比较结果相等，尽管 `.isPositive`对前者返回 true，对后者返回 false。Stainless 能正确处理这种行为。

断言注入。(2) 我们在工[illegible]中进行最大幅度修改的阶段之一是断言注入。对于整数算术，Stainless 默认会插入检查，以确保运算不会溢出且不会发生除以零的情况。可以通过命令行选项全局禁用此行为，也可以通过包装器在局部禁用。断言会在 AST 遍历期间根据正在处理的节点注入。

我们通过为浮点运算添加额外检查来扩展这一机制。具体而言，我们会在比较操作上添加 `NaN`检查，并为转为整数的类型转换添加安全检查。处理比较节点（包括相等判断）时，Stainless 现在会生成一个验证条件，确保两个操作数都不是`NaN`。遍历转为整数的类型转换时，Stainless 会生成两个验证条件：一个确保该值不是 `NaN`，另一个确保浮点值处于整数域范围内。正如我们的实证评估（第 5.4 节）所示，这些检查可以检测出用户代码中的真实漏洞。

有人可能会认为，应全局检查 NaN 值，例如，禁止它们出现在任何函数参数中。从正确性的角度来看，这种做法颇具吸引力，但也会引发对工具易用性的担忧。过多的检查可能会使 SMT 求解器不堪重负，并大幅增加验证时间。此外，这些检查应对受检函数或表达式的输入作出何种假设，也尚不明确。另外，在某些情境下， `NaN` 值可能是可接受的（例如，用于描述定义域外行为）。在这种情况下，用户应保留逐函数禁用这些检查的能力。另一种做法是提供细粒度控制，例如支持不同的严格程度级别或可配置选项。要评估这些易用性方面的权衡，需要开展用户研究；我们将其留待未来工作。

加入 ***Bitwuzla***。(3)我们[illegible]展了 Stain- less 中可用的 SMT 求解器集合，将 Bitwuzla 纳入其中 [33]。Bitwuzla 对浮点运算和位向量提供了强大的原生支持，在处理这些类型的查询时，其性能优于其他求解器（第 5.3 节）。然而，它不支持代数数据类型（ADTs），而 Scala 程序中广泛使用了这类类型。对于涉及 ADTs 的验证条件，Stainless 会退而使用通用求解器，例如 Z3 [32] 或 cvc5 [5]。

4 案例研究：经验证的标准数学库

浮点程序通常依赖标准数学库，尤其是用于超越函数的库 [18]。然而，这些

函数的实现目前往往过于复杂，SMT 求解器无法直接对其进行推理。相反，面向 Java 的 KeY 验证器会对这些函数进行公理化 [1], ，仅公开部分属性，例如其输出值域或在特殊输入值下的行为。我们为 Stainless 采用这种基于属性的方法，避免内联底层实现，从而使依赖这些函数的程序仍可进行高效验证。

Stainless 比 KeY 更进一步：它验证这些公开性质对于与 OpenJDK 数学库等价或近似的实现是否成立。OpenJDK 数学库基于广泛使用的 FdLibm C 库 [38]。具体而言，我们在 Stainless 中实现了一个经过验证的数学库，其 API 与 Scala 标准库 API 一致。非超越函数使用现有的 Scala 实现，并在验证过程中内联。我们扩展了所支持的超越函数范围，超出 KeY 中公理化的函数，涵盖完整的 Scala 数学库，包括双曲函数、对数、立方根和斜边函数。我们还验证了其他性质，例如符号保持以及与恒等函数的关系。附录中的表 3 总结并比较了 KeY 和 Stainless 所支持的性质。用户可以选择让 Stainless 通过这些已验证性质对超越函数进行推理，也可以选择内联其实现。

对于大多数超越函数（18 个中有 15 个），Stainless 中的实现会将 FdLibm 代码转换为静态单赋值（SSA）形式，其中每个变量只赋值一次。这种转换消除了可变性，使我们能够绕过 Stainless 对可变状态的编码；否则，这种编码会增加验证条件的数量并降低求解器性能。我们手动执行 SSA 转换，并通过测试转换后的代码与原始实现对十亿个按数量级分层的随机输入，以及零值、无穷值、`NaN`S 和 π 的倍数等特殊值的处理情况，来验证转换结果。我们还对次正规数进行过采样，因为已有研究表明它们是浮点代码中漏洞的常见来源 [13]。由于 Stainless 的契约默认会编译为运行时断言，对这些实现进行模糊测试也能检查已证明的性质是否对测试输入成立。

三角函数。 三角函数 `sin`、 `cos` 和 `tan` 的实现分为两个不同的步骤：1. 进行区间缩减，计算输入对 $\pi/2$ 的取模结果；2. 根据缩减后的自变量计算函数值。第二步的性质可以采用与其他 FdLibm 函数相同的方法进行验证。由于 SMT 求解器性能不佳，需要进行更大范围的修改，才能证明第一步返回的值处于预期值域内。

取模 $\pi/2$ (RemPio2) 余数运算分为几种情况：无穷大或 `NaN`值、较小的输入、中间范围的输入（幅值为 $\sim\leq 2^{20}\pi/2$），以及最后一种能够处理所有有限浮点数的情况。Stainless 可以轻松处理前两种情况，但由于 SMT 求解器性能不佳，我们未能完成中间幅值情况的证明。因此，我们从实现中移除了中间值域的情况，转而对中间输入也使用来自

最后一种情况所采用的算法。我们将介绍这种情况所用的算法，以及为证明所做的一些修改。

无法使用普通浮点运算计算余数，因为这样会导致精度低得无法接受。相反，该算法使用浮点值数组或整数数组来表示数值，以达到高于普通双精度浮点数的精度。将输入转换为这种形式后，依次进行以下操作：1. 乘以 $2/\pi$ 展开式中与输入大小相关的一部分；2. 去掉整数部分；3. 乘以 $\pi/2$ 展开式的一个前缀；4. 将数组压缩为一个双精度浮点数，并附带一个补偿项（即另一个双精度浮点数，用于表示结果中幅度太小、无法由第一个浮点数表示的部分）。经过我们的修改，Stainless 可以在大约 20 分钟内证明该结果位于 $[-\pi/4, \pi/4]$ 的预期值域内。

这些数组上的算术运算通过循环实现。由于 Stainless 是一种函数式验证器，因此原生不支持循环。Stainless 会将循环转换为递归函数，并返回一个元组，其中包含循环所修改的所有变量的更新值。在 SMT-LIB 中，这些元组被编码为代数数据类型。

这意味着我们无法依赖 Bitwuzla 的高性能，因为它不支持代数数据类型。相反，我们必须通过添加额外的断言，有时展开几次循环迭代，来帮助 cvc5 求解验证条件。这两种做法都是为了提升求解器性能。我们还修改了将输入转换为数组形式的方式，确保相同的数组索引始终表示大小相同的值。这样一来，循环不变式便可从由不同情况组成的析取式简化为单一情况。

原始实现会根据需要，使用 $2/\pi$ 展开式中可变数量的比特；如果发现精度不足，就用更多比特重新执行部分计算。由于在 Stainless 中证明这一过程会终止极具挑战性，我们通过使用 $2/\pi$ 的固定数量比特简化了实现。

最后，我们还修改了实现第 4 步压缩操作的循环，使其能够满足一个简单的不变式。我们的循环实现基于 Ogita 等人的算法 4.1 [35],，但使用 `FastTwoSum` 函数，而非 `TwoSum` 函数。这些修改共同使我们能够证明结果处于预期值域内，从而证明三角函数的取值也处于预期值域内。

除了前文所述的模糊测试外，我们还对所有单精度浮点数和 100 亿个随机双精度值进行了三角函数测试。由于该实现已不再与原始实现逐位等价，我们必须修改三角函数的测试方式。在所有受测输入上，Stainless 的三角函数实现与 OpenJDK 的实现之间的误差不超过最后一位单位（ulp）。

5 评估

我们在三组基准测试上评估了对 Stainless 的扩展，以探究以下研究问题：

研究问题 1： Stainless 在多大程度上能够验证真实世界浮点程序的正确性，或在验证失败时生成反例？
研究问题 2： 对于从真实世界代码生成的浮点查询，所支持的 SMT 求解器的性能相比如何？
研究问题 3： 针对比较的非数检查和类型转换安全检查，能否检测出真实世界代码中可能存在的漏洞？

5.1 实验设置

我们使用三组基准测试来代表真实世界中的浮点程序：用于评估 Stainless 标准数学库中的 KeY [1],函数的基准测试（第 4 节），以及从 GitHub 上的开源 Scala 项目中提取的浮点函数。

在本节中，我们将每个带注释的源文件 视为一个基准测试。每个基准测试可能包含多个带注释的函数，而每个函数又可能生成多个验证条件（VC）。

所有实验均在 Ubuntu 24.02 上运行，硬件配置为 AMD Ryzen 5900X CPU（12 核，基准时钟频率为 3.7 GHz）和 128 GB 内存。我们使用三个 SMT 求解器：Z3 [32] （版本 4.13.0）、cvc5 [5]（版本 1.1.2）和 Bitwuzla [33] （由最新的 GitHub 提交 [34] 构建）。所有求解器均以单线程运行。

对于 KeY 基准测试，我们禁用 Stainless 面向易用性的 `NaN` 检查以及针对类型转换的安全检查，因为 KeY 不包含这些功能。对于基于 FdLibm 的基准测试，我们也禁用这些检查，以避免这些检查扩大求解器上下文、进而降低求解器性能。对于实证 GitHub 基准测试，我们则保留这些检查。

KeY 基准测试。 由于 Stainless 主要面向函数式程序，我们将基准测试调整为更符合函数式风格的形式。具体来说，我们将类设为不可变，将可变方法改写为返回新实例，并用元组替换较小的定长数组。此外，我们将紧密耦合的 Java 源文件合并为单个 Scala 文件，因此有些 Scala 基准测试对应于原 Java 测试套件中的多个类或文件。

与 Stainless 不同，KeY 验证器支持每个方法有多个契约。我们通过以下方式转换这类 KeY 基准测试：(1) 将所有契约共有的前置条件添加为 Stainless 中的前置条件；(2) 将每个 KeY 契约编码为 Stainless 中的蕴含，并将它们合并为一个合取后置条件，同时省略重复的前置条件。如果 KeY 基准测试中的某个方法有多个契约，且其中一个无效，我们会生成该方法的两个版本，以避免使用无效契约来证明其他契约：一个版本只包含有效契约，另一个版本则包含无效契约。KeY 基准测试中没有任何方法包含多个无效契约。

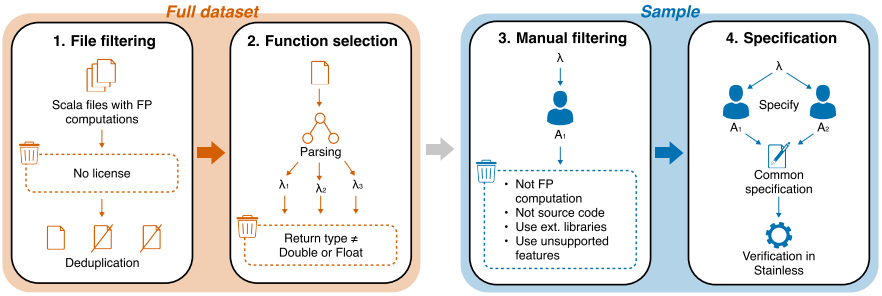


图 3：从 GitHub 提取基准测试的方法概览

FdLibm 基准测试。第 4 节中介绍的案例研究构成了第二组基准测试。数学库中的每个函数都会转换为一个基准测试，要求证明关于该函数的所有引理，并假定库中其他函数的所有引理均正确。有两个例外：正弦函数和余弦函数的大部分代码相同，因此合并为一个基准测试；余数取模-$\pi/2$(RemPio2) 运算（用于实现正弦、余弦和正切函数）则作为独立的基准测试。

实证基准测试。此前的工作 [18] 表明，常见的浮点验证基准测试无法准确反映真实世界的程序。在该研究中，作者从 GitHub 提取了一个数据集，其中包含使用静态类型语言编写、涉及浮点计算的源文件。我们基于此数据集，使用其中的 Scala 子集，为 Stainless 推导实证基准测试。图 3 概述了我们提取并指定这些基准测试的整体方法。

我们选取所有采用开源许可证的 Scala 文件，删除重复项以减少大规模代码重复 [28], ，并提取所有返回浮点值的函数。从中均匀抽样 500 个函数，并手动排除：

– 178 个返回浮点值但不执行任何浮点计算的函数（例如简单的获取器）。– 54 个不属于代码仓库源代码的函数（例如属于测试套件的函数）。– 97 个直接或间接调用外部库函数的函数（例如 Breeze、Apache Spark）。– 70 个使用 Stainless 不支持的语言特性的函数，例如宏、并发或有界多态。在可能的情况下，我们会通过移除不受支持的特性来调整这些函数，同时不改变其语义。

筛选后，我们从 41 个不同的代码仓库中获得了 103 个函数。由于函数之间可能存在依赖关系，我们将同一源文件中的函数及其依赖项归为一个验证基准测试。

前两位作者分别为这些函数编写规格说明，依据包括代码内文档、使用场景、已发表算法的参考资料以及语义线索（例如，假定名为 `count` 的变量为非负数）。我们系统地检查代码在有效输入下是否可能返回非数等特殊值。随后，标注者比较各自的规格说明，并通过讨论解决分歧。如果无法达成共识，则保留两份规格说明并分别进行验证。若 Stainless 在验证某个基准测试时产生反例，我们会手动修正实现并重新尝试验证。由此得到两个不同的基准测试：一个代表原始实现，另一个则包含修正后的实现。

最终的实证基准测试套件由来自 52 个唯一源文件的 79 个基准测试组成；其中 9 个源文件存在尚未解决的规范分歧，另有 18 个基准测试源于对反例的修正。

5.2 研究问题 1：完整基准测试上的验证结果

我们通过对所有基准测试集运行端到端实验，评估 Stainless 中的浮点支持。我们将 Stainless 配置为采用 cvc5、Bitwuzla 和 Z3 进行组合求解；对于 KeY 基准测试和实证基准测试，每个验证条件的超时设为 300 秒。对于数值密集度更高的 FdLibm 基准测试，我们不设置超时，因为这些测试旨在确立有关库函数的特定引理的正确性，这要求所有验证条件都完成。我们没有与 KeY 或 Frama-C 进行实证比较（例如，测量性能），因为这些工具面向不同的编程范式。此类比较需要对基准测试进行大量修改，正如我们对 KeY 基准测试的翻译所表明的那样，这会使结果难以解读。

为避免执行问题，评估中排除了三个基准测试。其中两个属于实证基准测试：一个会导致 Stainless 的终止推断机制发生栈溢出，另一个会使 Stainless 冻结，需要手动中断。我们还排除了 FdLibm RemPio2 基准测试，因为它会在评估机器上触发内存溢出错误。不过，在用于 FdLibm 案例研究的笔记本电脑上，该基准测试可以成功完成验证，且不会占用异常多的内存。我们将进一步调查这些问题。

表1总结了 KeY 和 FdLibm 基准测试以及部分实证基准测试的评估结果。所有实证基准测试的完整统计数据见附录中的表4。对于每个基准测试，我们报告五次独立运行中 Stainless 端到端实际运行时间的最小值、中位数和最大值。我们还列出了有效和无效验证条件的总数，以及被报告为未知或导致超时的验证条件数量。此外，对于中位数运行，我们报告了每个求解器所解决的有效和无效验证条件数量。

对于 KeY 和 FdLibm 基准测试集，所有验证条件均已成功证明为有效或无效。对于实证基准测试，在 79 个实证案例中，有 6 个出现了至少一个未知结果或超时：其中 3 个基准测试遇到

表1：端到端实验五次运行的汇总统计。最小值/中位数/最大值列汇总了端到端执行时间。#VCs 列显示在执行时间为中位数的那次运行中，每个基准测试得到有效（/无效/未知/超时）结果的验证条件数量。cvc5、Bitwuzla 和 Z3 列显示在同一次中位数运行中，各求解器证明为有效（/无效）的验证条件数量。#fns 列显示对超越函数数学库函数的调用次数（若非零）。

(a) KeY 基准测试（300 秒超时）

基准测试	最小值	中位数	max	#验证条件	cvc5	Bitwuzla	Z3	#fns
CartesianPolar	12.92	13.05	13.15	3	3	0	0	3
复数	94.74	95.62	96.06	9/1/0/0	9/1	0	0	3
FPLoop	4.48	4.55	4.67	30/3/0/0	13	12/3	5	
Matrix2	2.50	2.53	2.61	1	1	0	0	
Matrix3	129.44	130.34	130.65	1/1/0/0	1/1	0	0	
PineBench	234.66	245.32	247.01	105/14/0/0	58/2	46/12	1	
矩形	10.53	10.55	10.72	1/1/0/0	0/1	0	1	
旋转	21.05	21.17	21.30	6	4	2	0	

（b）FdLibm 基准测试（无超时）

基准测试	最小值	中位数	max	#验证条件	cvc5	Bitwuzla	Z3	#fns
Acos	33.72	35.60	42.28	11	1	6	4	
Asin	105.46	171.82	859.33	8	1	6	1	
Atan	62.43	64.44	70.88	10	0	8	2	
双参数反正切	8.40	8.47	8.56	28	0	7	21	2
Cbrt	667.36	913.66	1178.50	11	1	7	3	
Cosh	40.07	41.85	45.40	9	1	7	1	4
Exp	49.85	52.04	152.58	16	0	11	5	
指数函数减一	3224.93	3969.72	4189.57	24	1	19	4	
斜边函数	16.88	17.49	18.24	7	0	6	1	
Log	140.12	206.87	741.37	16	1	14	1	
常用对数	4.10	4.21	4.22	6	0	4	2	1
自然对数加一	22.78	120.52	249.31	16	1	13	2	
Pow	9.40	9.57	9.63	24	1	21	2	
正弦余弦	194.80	220.15	511.64	42	12	3	27	
Sinh	111.82	119.59	125.88	9	1	8	0	3
Tan	17.80	17.93	18.09	13	3	4	6	
Tanh	4.35	4.42	4.59	6	0	5	1	2

(c) 按中位时间均匀选取的实证基准测试子集 按中位时间均匀选取的基准测试（30 秒超时）

基准测试	最小值	中位数	max	验证条件数	cvc5	Bitwuzla	Z3	#fns
HTMLComp.Position _OK	3.30	3.35	3.43	2	1	1	0	
流式输入输出 _CE	3.58	3.60	3.65	2/1/0/0	1/1	1	0	
边界框 _a1_OK	3.85	3.89	4.00	4	2	0	2	
曲线下面积 _OK	4.16	4.21	4.28	11	10	0	1	
数据辅助器 _OK	4.69	4.79	4.88	21	4	0	17	
玩家 _a0_OK	6.26	6.38	6.54	21	16	0	5	
边界 _OK	7.74	7.83	7.87	77	14	0	63	
中介 _a1_OK	9.41	9.50	9.71	56	10	0	46	
方差 _CE	15.25	15.44	15.61	37/2/0/0	18/2	4	15	
二维向量 _a0_CE	33.42	33.55	33.75	7/2/0/0	3/2	0	4	
公共 _TO	727.42	748.05	916.92	46/0/1/2	15	5	26	1

表 2：各求解器解决的验证条件数量

	#VCs	cvc5	Z3	Bitwuzla	任意求解器
KeY	176	176（100%）	150（85.2%）	118（67.0%）	176（100%）
FdLibm	256	227（88.7%）	200（78.1%）	217（84.8%）	254（99.2%）
实证	1600	1595（99.7%）	1528（95.5%）	489（30.6%）	1596（99.8%）

至少有一个未知结果，另有 4 个至少遇到一次超时。在 79 个实证基准测试中，有 73 个的所有验证条件均成功判定为有效或无效。

人工检查发现，对于 Stainless 判定为无效的 8 个实证基准测试，报告的反例都是伪反例。[3] 这些基准测试包含超越函数，Stainless 将其视为不透明函数。也就是说，代码库的其他部分只能使用这些函数的契约，而无法访问其实现。函数调用会被转换为 SMT 求解器中的未解释函数应用，并且仅受契约约束。如果契约不足以证明某个验证条件，求解器就可能构造出一个满足契约、却违反函数预期语义的模型，从而导致伪反例。在另外 11 个使用超越函数的实证基准测试中，1 个超时，5 个通过验证，另有 5 个产生了非伪反例。总的来说，对于使用超越函数的 19 个基准测试，KeY 引入的公理化方法在其中 8 个上并不充分，也就是说，这一比例相当高。

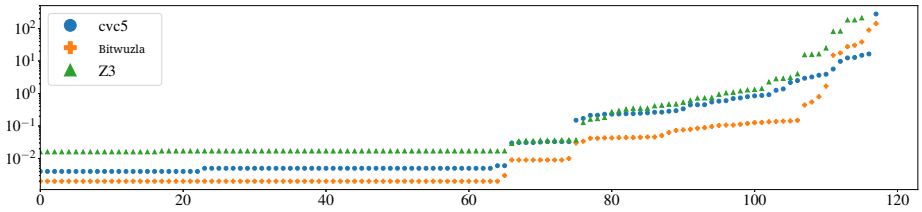
研究问题 1 的答案：我们的结果表明，对于自包含的真实世界浮点程序，Stainless 通常能够成功完成验证或生成反例（占实证案例的 82%）。在某些情况下，我们观察到验证时间较长且方差较大，这在数值计算密集度更高的 FdLibm 基准测试中尤为明显。超越函数仍然是一项挑战：案例研究中引入的引理对其行为进行了过度近似，导致多个基准测试中出现伪反例（19 个中有 8 个）。

5.3 研究问题 2：验证条件的求解器比较

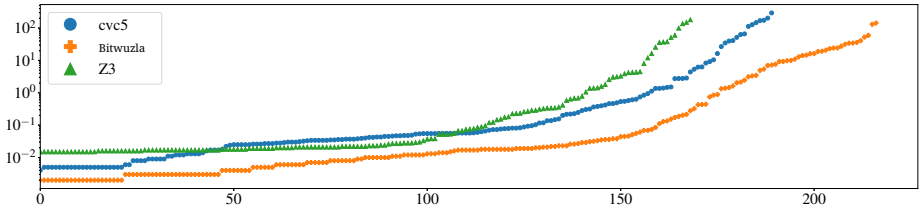
为了评估和比较 Stainless 所支持的 SMT 求解器的性能，我们将 Stainless 配置为针对每个基准测试中的每个验证条件生成一个 SMT-LIB 文件。我们对每个生成的 SMT-LIB 文件分别运行每个求解器五次，并测量总执行时间。对于由 KeY 和 FdLibm 基准测试生成的验证条件，我们使用 300 秒的超时时间。鉴于实证基准测试数量庞大，我们对这组基准测试采用较短的 120 秒超时时间。

表 2 报告了 cvc5、Z3 和 Bitwuzla 在所有基准测试类别中解决的验证条件数量。该表还列出了至少由三个求解器之一解决的验证条件总数。与端到端评估相比，我们

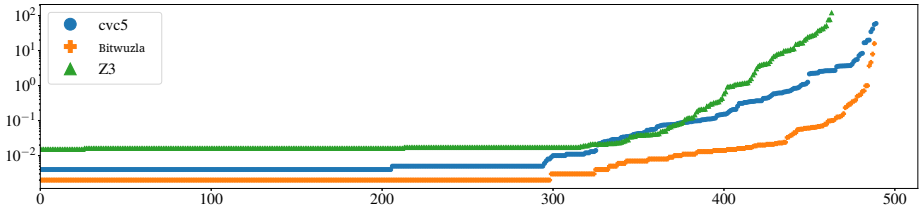
[3] 对于某一次特定运行，无法保证 SMT 求解器在不同运行中产生相同的反例。



(a) KeY 基准测试（300 秒超时）



(b) FdLibm 基准测试（300 秒超时）



（c）实证基准测试（120 秒超时）

图 4：所有求解器均支持的验证条件的求解器运行时间

观察到未解决的验证条件更少，因为未知结果可能源自 Stainless 本身；在这种情况下，我们不会从 SMT 求解器观察到任何未知结果。结果表明，组合求解方法是有效的，几乎所有验证条件都至少由一个求解器解决。

Bitwuzla 未解决的大多数验证条件对应的是错误，而非性能限制：对于 58 个 KeY 基准测试、37 个 FdLibm 基准测试和 1107 个实证基准测试，它会产生错误，而 cvc5 和 Z3 则不会遇到任何错误。我们认为，所有这些错误都源于对代数数据类型和整数理论缺乏支持。因此，我们将求解器之间的比较限制在 Bitwuzla 不会遇到任何错误的验证条件子集上。图 4 展示了三个求解器在该子集上的性能，附录中的图 5 则给出了所有验证条件对应的图表。

研究问题 2 的回答：对于 Bitwuzla 未遇到任何错误的情况，我们得出了清晰的求解器排名：Bitwuzla 的表现优于 cvc5，尤其是在计算密集型的 FdLibm 基准测试中；而 cvc5 的表现优于 Z3。由于 Bitwuzla 不支持代数数据类型或整数理论，因此总体而言，cvc5 仍能求解更多验证条件。

5.4 RQ3：NaN 检查和类型转换安全检查的影响

在实证基准集中包含的 103 个函数中，有 9 个涉及浮点比较。在所有这些情况下，对 NaN值的处理都不完善。在其中两个函数中，这一遗漏会导致实现即使在输入为 NaN时也返回有效输出，从而实际上悄然忽略 NaN 输入。在一个案例中，代码注释明确假定假比较意味着输入有效且处于某个值域内，但当输入为 NaN 时，这一性质并不成立；我们在附录的图 6 中展示了这个函数。

在大多数函数中，函数参数都会与常量阈值进行比较。在此类情况下，假设输入不为NaN ，就足以保证不会进行错误的比较。当函数缺少这一假设时，Stainless 能够成功检测并报告潜在的 NaN 比较。

实证基准集中有两个函数会进行浮点数到整数的类型转换。二者都不安全，Stainless 会报告具体反例，这些反例会触发 NaN到整数的转换或越界整数类型转换。

RQ3 的回答：自动化的 NaN检查和类型转换安全检查能够有效识别用户代码中的浮点相关错误。在我们的实证基准集中，Stainless 成功检测出所有不安全的类型转换和潜在的 NaN 比较，并为每种情况生成具体反例。

6 相关工作

据我们所知，我们是首个将浮点推理引入 Scala 演绎式验证器的团队。Stainless 通过验证条件实现浮点支持，并使用浮点 SMT-LIB 理论 [36] 由 SMT 求解器进行求解；这一支持沿袭了 KeY 在 Java 中、Why3 [15] 在 C [11] 和 Ada [16], 中，以及 Boogie 中间语言 [27]中的相关支持。与这些工具相比，Stainless 为具备不同特性集合的输入语言提供浮点支持，这些特性包括多态、递归、高阶函数和类不变量。此外，它还加入了自动化 NaN 检查，相较于现有验证器降低了注解负担。KeY 对标准数学库 [1]中的部分超越函数进行了公理化处理。相比之下，Stainless 覆盖了整个标准数学库，并且对于大多数函数，能够根据实际实现本身验证公理。对于其他基于 JVM 的语言，演绎式验证器仅提供有限支持：VeriFast [24]将浮点运算视为实数，而 OpenJML [10] 和 LiquidJava [17]虽然能够解析浮点程序，却无法对其进行推理。

基于抽象解释的技术 [9,25] 原则上可以自动证明浮点代码中不存在特殊值。然而，要将这些分析有效地应用于现实世界的程序，通常需要用户进行大量调优，例如选择抽象域或加宽阈值 [7]。Ariadne [6] 结合符号执行、实数 SMT 求解和测试，寻找会触发浮点异常的输入，包括溢出和无效运

算。我们的工作涵盖了这类分析：Stainless 不仅通过 SMT 求解生成反例，还能证明不存在此类异常。

除了证明不会出现特殊值，近期研究还开发了静态分析方法，用于界定浮点舍入误差 [12, 19, 29, 31, 37]。然而，这些分析目前针对的是用领域特定语言编写的小型算术内核，并且不支持主流编程语言的特性，例如用户定义数据类型、对象和递归。

Stainless 使用自身的浮点支持，验证标准数学库函数的公理是否符合其实现。Bagnara 等人 [4] 采用另一种结合符号执行、抽象解释和穷举测试的方法，验证 `math.h` C 库中函数的性质，例如分段单调性，以及在有效输入下不会产生 `NaN` 结果。另一类互补研究则使用交互式定理证明器（例如 HOL、Rocq）验证超越函数的浮点实现的误差界 [2, 8, 21–23, 30]；此类证明需要大量数学工具，并要求用户具备丰富的专业知识且投入大量交互工作。Harrison 指出，每项此类证明所需的人工工作量可能从数周到数月不等 [21]。

7 结论

要在支持类、多态和高阶函数等高级语言特性的程序演绎式验证器中支持浮点推理，需要施加特定限制，例如限制多态相等的实例化，或通过未解释函数削弱相等。通过扩展 Stainless，我们复现了 KeY 的验证结果，并进一步验证了 Scala 标准数学库的一个大型子集；KeY 此前只能对其中部分内容进行公理化，或不支持这些内容。我们还展示了 Stainless 能够验证从 GitHub 抽取的用户代码。

未来的工作包括检测并消除由超越函数导致的伪反例。检测到伪反例时（例如通过求值程序），一种可能的策略是内联完整的函数实现，然后重新尝试验证。然而，当前的 SMT 求解器通常无法高效地推理超越函数的完整实现，因此这种方法在实践中往往效果不佳。通过添加更多性质来强化契约可以缓解这一问题，但无法彻底消除其根本原因。未来的工作还包括研究可用性方面的权衡，例如增加更多自动检查，并设计更易读的规格机制。

致谢。 由欧盟（欧洲研究理事会，HORNET，101163629）共同资助。但本文所表达的观点和意见仅代表作者本人，不一定反映欧盟或欧洲研究理事会的观点。欧盟和资助机构均不对此承担责任。

利益披露。 作者声明不存在与本文内容相关的任何利益冲突。

8 数据可用性声明

我们为 Stainless 提供的浮点支持位于 Stainless 代码仓库的主分支中：https://github.com/epfl-lara/stainless。我们还提供了一个 Zenodo 工件（https://doi.org/10.5281/zenodo.17486575），其中包含该实现、本文使用的所有基准测试、收集的评估结果以及用于复现实验的脚本。

参考文献

1. Abbasi, R., Schiffl, J., Darulova, E., Ulbrich, M., Ahrendt, W.：在 KeY 中结合基于规则和 SMT 的推理来验证浮点 Java 程序。《国际软件工具与技术转移杂志》**25**(2)，185–204（2023 https://doi.org/10.1007/s10009-022-00691-x）。. Akbarpour, B., Abdel-Hamid, A.T., Tahar, S., Harrison, J.：使用 HOL 验证 IEEE-754 浮点指数函数的合成实现。《计算机杂志》**53** 4 465 488 2009 https://doi.org/10.1093/comjnl/bxp023 **( )**，– （）。. Apache 软件基金会：Apache Spark。（），访问日期：2025年10月14日 4. Bagnara, R., Chiari, M., Gori, R., Bagnara, A.：使用 math.h/cmath 函数验证浮点 C/C++ 程序的实用方法。《ACM 软件工程方法学汇刊》**30** 1 2021 https://doi.org/10.1145/3410875 **( )**（）。. Barbosa, H., Barrett, C., Brain, M., Kremer, G., Lachnitt, H., Mann, M., Mohamed, A., Mohamed, M., Niemetz, A., Nötzli, A., Ozdemir, A., Preiner, M., Reynolds, A., Sheng, Y., Tinelli, C., Zohar, Y.：cvc5：一种多功能、工业级的 SMT 求解器。载于：系统构造与分析工具和算法（TACAS）（2022 https://doi.org/10.1007/978-3-030-99524-9_24）。. Barr, E.T., Vo, T., Le, V., Su, Z.：自动检测浮点异常。载于：编程语言原理研讨会（POPL）论文集（2013 https://doi.org/10.1145/2429069.2429133）。. Blanchet, B., Cousot, P., Cousot, R., Feret, J., Mauborgne, L., Miné, A., Monniaux, D., Rival, X.：面向大型安全关键软件的静态分析器。载于：编程语言设计与实现会议（PLDI）论文集（2003 https://doi.org/10.1145/781131.781153）。. Boldo, S., Filliâtre, J.C., Melquiond, G.：结合 Coq 和 Gappa 认证浮点程序。载于：符号计算与机械化推理集成研讨会（Calculemus），作为智能计算机数学会议（CICM）的一部分举行（2009 https://doi.org/10.1007/）。9. 978-3-642-02614-0_10 Chen, L., Miné, A., Cousot, P.：可靠的浮点多面体抽象域。载于：编程语言与系统（APLAS）（2008）。https://doi.org/10.1007/978-3-540-89330-1_210. Cok, D.R.：OpenJML：通过扩展 OpenJDK 为 Java 7 提供 JML。载于：NASA 形式化方法（2011 https://doi.org/10.1007/978-3-642-20398-5_35）。. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.：Frama-C——软件分析视角。载于：国际软件工程与形式化方法会议（SEFM）（2012 https://doi.org/10.）。 1007/978-3-642-33826-7_16

12. Darulova, E., Izycheva, A., Nasir, F., Ritter, F., Becker, H., Bastian, R.：Daisy——数值程序分析与优化框架（工具论文）。载于：系统构造与分析工具和算法（TACAS）（2018）。 https://doi.org/10.1007/978-3-319-89960-2_1513. Di Franco, A., Guo, H., Rubio-González, C.：真实世界数值错误特征的全面研究。载于：国际自动化软件工程会议（ASE）（2017）。 https://doi.org/10.1109/ASE.2017.811566214. Dinda, P., Hetland, C.：开发者理解 IEEE 浮点数吗？载于：国际并行与分布式处理研讨会（IPDPS）（2018）。https://doi.org/10.1109/IPDPS.2018.0006815. Filliâtre, J.C., Paskevich, A.：Why3——程序与证明器相遇之处。载于：欧洲编程研讨会（ESOP），编程语言与系统（2013）。https://doi.org/10.1007/978-3-642-37036-6_816. Fumex, C., Marché, C., Moy, Y.：浮点程序验证的自动化。载于：经验证的软件：理论、工具与实验（VSTTE）（2017）。https://doi.org/10.1007/978-3-319-72308-2_717. Gamboa, C., Canelas, P., Timperley, C., Fonseca, A.：面向易用性的 Java 液体类型设计。载于：国际软件工程会议（ICSE）（2023）。 https://doi.org/10.1109/ICSE48619.2023.0013218. Gilot, A., Wrigstad, T., Darulova, E.：静态类型语言中浮点数使用情况的大规模研究。CoRR **abs/2509.04936**（2025）。 https://doi.org/10.48550/ARXIV.2509.0493619. Goubault, E., Putot, S.：有限精度计算的静态分析。载于：验证、模型检查与抽象解释（VMCAI）（2011）。 https://doi.org/10.1007/978-3-642-18275-4_1720. Hamza, J., Voirol, N., Kunčak, V.：System FR：Stainless 验证器的形式化基础。ACM《编程语言会议论文集》**3**（OOPSLA）（2019）。 https://doi.org/10.1145/336059221. Harrison, J.：HOL Light 中的浮点验证：指数函数。《系统设计中的形式化方法》**16**(3)，271–305（2000）。 https://doi.org/10.1023/A:100871290715422. Harrison, J.：三角函数的形式化验证。载于：计算机辅助设计中的形式化方法（FMCAD）（2000 https://doi.org/10.）。1007/3-540-40922-X_1423. Harrison, J.：平方根算法的形式化验证。《系统设计中的形式化方法》**22**(2)，143–153（2003）。 https://doi.org/10.1023/A:1022973506233 24. Jacobs, B., Smans, J., Philippaerts, P., Vogels, F., Penninckx, W., Piessens, F.：VeriFast：适用于 C 和 Java 的强大、可靠、可预测且快速的验证器。载于：NASA 形式化方法（2011）。 https://doi.org/10.1007/978-3-642-20398-5_425. Jeannet, B., Miné, A.：Apron：用于静态分析的数值抽象域库。载于：计算机辅助验证（CAV）（2009 https://doi.org/10.1007/）。978-3-642-02658-4_5226. Lehmann, N., Geller, A.T., Vazou, N., Jhala, R.：Flux：面向 Rust 的液体类型。ACM《编程语言会议论文集》**7**（PLDI）（2023）。 https://doi.org/10.1145/359128327. Leino, K.R.M.：这就是 Boogie 2。 https://www.microsoft.com/en-us/research/publication/this-is-boogie-2-2/ 2008 2025 10 14 （），访问日期：--。.28 Lopes, C.V., Maj, P., Martins, P., Saini, V., Yang, D., Zitny, J., Sajnani, H., Vitek, J.：DéjàVu：GitHub 上重复代码的分布图。ACM《编程语言会议论文集》**1**（OOPSLA）（2017）。 https://doi.org/10.1145/3133908

29. Magron, V., Constantinides, G., Donaldson, A.：使用半定规划的经认证的舍入误差界。ACM 数学软件汇刊 **43(4)**（2017年）。 https://doi.org/10.1145/301546530. Melquiond, G.：Coq 系统中的浮点算术。《信息与计算》**216**，14–23（2012年）。 https://doi.org/10.1016/j.ic.2011.09.005，特刊：第 8 届实数与计算机会议 31. Moscato, M., Titolo, L., Dutle, A., Muñoz, C.A.：通过静态分析自动估计经验证的浮点舍入误差。载于：计算机安全性、可靠性与安全性（SAFECOMP）（2017年）。 https://doi.org/10.1007/978-3-319-66266-4_1432. de Moura, L., Bjørner, N.：Z3：一种高效的 SMT 求解器。载于：系统构造与分析工具和算法（TACAS）（ 2008 https://doi.org/10. ）。 1007/978-3-540-78800-3_24
33. Niemetz, A., Preiner, M.：Bitwuzla。载于：计算机辅助验证（CAV）（2023年）。 https://doi.org/10.1007/978-3-031-37703-7_134. Niemetz, A., Preiner, M.：Bitwuzla 公共代码库——用于实验的提交版本。https://github.com/bitwuzla/bitwuzla/tree/ 929af621f6fe65c78236df70b8eb9343a5914d9f 2025 2025 10 14 ( )，访问日期：--35
-。Ogita, T., Rump, S.M., Oishi, S.：精确的求和与点积。《 SIAM 科学计算杂志》 **26** 6 1955 1988 2005 https://doi.org/10.1137/030601818 ( )，–36
（）。. Rümmer, P., Wahl, T.：二进制浮点算术的 SMT-LIB 理论。载于：可满足性模理论国际研讨会（SMT）（2010年）37. Solovyev, A., Baranowski, M.S., Briggs, I., Jacobsen, C., Rakamarić, Z., Gopalakrishnan, G.：利用符号泰勒展开严格估计浮点舍入误差。《ACM 编程语言与系统汇刊》**41(1)**（2018年）。 https://doi.org/10.1145/3230733 38. 太阳微系统公司：FDLIBM 5.3 版。 https://www.netlib.org/fdlibm/ （2022年），访问日期：2025年10月14日 39. Vazou, N., Seidel, E.L., Jhala, R.：LiquidHaskell：现实世界中的细化类型实践。载于：Haskell 研讨会（2014年）。 https://doi.org/10.1145/2633357.263336640. Vazou, N., Seidel, E.L., Jhala, R., Vytiniotis, D., Peyton-Jones, S.：Haskell 的细化类型。载于：函数式编程国际会议（ICFP）（2014年）。 https://doi.org/10.1145/2628136.2628161

附录

表3：KeY 中的超越函数公理与 Stainless 中的已验证性质之比较

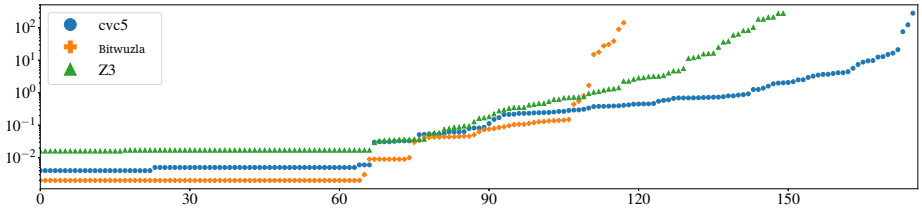
函数 KeY（公理）		Stainless（已验证）
cos	非数、值域	非数，值域
sin	非数，f(±0)，值域	非数，f(±0)，值域
tan	f(非数)，f(±0)，值域	f(非数)，f(±0)，值域
asin	f(非数)，f(0)，值域	非数，f(±0)，f(±1)，值域
acos	f(非数)，f(0)，值域	非数，f(±0)，f(±1)
atan	f(非数)，f(0)，值域	非数，f(±0)，f(±∞)，符号，值域
双参数反正切	f(非数, _), f(_, 非数)，值域	非数、所有特殊值对、符号、值域
斜边函数	不支持	非数、所有特殊值对、值域
cbrt	不支持	非数、f(±0)、f(±1)、f(±∞)、符号、 $f(x) \leqslant x$
pow	所有特殊值对	所有特殊值对
exp	f(非数)、f(±∞)	非数，f(±0)，f(±∞)，值域
指数函数减一	不支持	非数，f(±0)，f(±∞)，值域
log	不支持	非数，f(±0)，f(1)，f(±∞)，值域， $f(x) \leq x - 1$
自然对数加一	不支持	非数，f(±0)，f(−1)，f(±∞)，值域， $f(x) \leq x$
常用对数	不支持	非数，f(±0)，f(1)，f(±∞)，值域
sinh	不支持	非数，f(±0)，f(±∞)，符号
cosh	不支持	非数，f(±0)，f(±∞)，值域
tanh	不支持	非数，f(±0)，f(±∞)，符号，值域

图例：
非数：函数输出何时为 NaNf 的精确刻画（□）：输入 □ 时的预期结果所有特殊值对：NaN、 ±0 和 ±∞ 的所有组合的预期结果符号：结果符号的刻画值域：结果的下界和/或上界f（x） $\leqslant x$：函数高于或低于其输入时的刻画

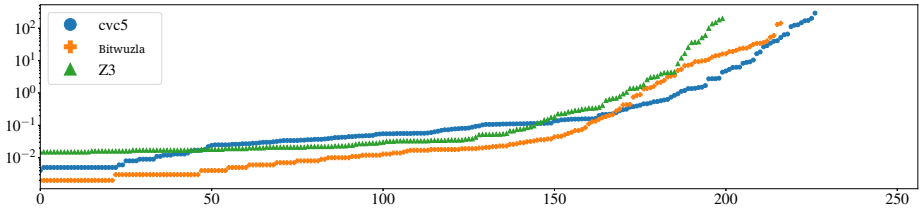
表4：实证基准测试中端到端实验五次运行的汇总统计。min/median/max 列汇总了端到端执行时间。#vcs 列显示中位数运行中被证明有效的验证条件总数；如果有些验证条件未被证明有效，则显示有效/无效/未知/超时结果的数量。cvc5、Bitwuzla 和 Z3 列显示各求解器在中位数运行中证明有效的验证条件数量，或有效/无效验证条件的数量。#fns 列显示超越函数数学库函数的调用次数（非零时）。

基准测试		最小值	中位数	max	#vcs	cvc5	Bitwuzla	Z3	#fns
绝对误差	_OK	3.35	3.37	3.41	2	0	2	0	
曲线下面积	_CE	4.18	4.22	4.38	9/1/0/0	7/1	0	2	
曲线下面积	_OK	4.16	4.21	4.28	11	10	0	1	
边界	_OK	7.74	7.83	7.87	77	14	0	63	
边界框	_a0_OK	4.01	4.07	4.11	6	4	0	2	
边界框	_a1_OK	3.85	3.89	4.00	4	2	0	2	
Cast	_OK	3.57	3.59	3.63	4	0	4	0	
公共	_TO	727.42	748.05	916.92	46/0/1/2	15	5	26	1
数据辅助器	_OK	4.69	4.79	4.88	21	4	0	17	
数据集成本	_false _CE	32.33	32.47	32.71	12/2/0/0	2/2	0	10	1
Exp.Mutagen	_false _CE	16.97	17.13	17.30	11/2/0/0	2/2	0	9	1
FlinkStatistic	_OK	4.24	4.29	4.34	10	5	0	5	
半精度浮点数	_a0_OK	4.27	4.34	4.38	17	7	0	10	
半精度浮点数	_a1_OK	4.50	4.58	4.62	21	16	2	3	
FollowTargetRule	_交叉熵	100.47	100.71	100.76	15/9/0/0	10/9	1	4	4
FollowTargetRule	_确定	104.89	104.91	105.39	32/3/0/0	20/3	2	10	4
GCode	_a0_CE	3.86	3.88	3.95	2/1/0/0	0/1	1	1	
GCode	_a0_OK	3.92	3.94	4.00	4	1	1	2	
GCode	_a1_OK	3.80	3.90	4.00	3	1	0	2	
HTMLComp.Position	_OK	3.30	3.35	3.43	2	1	1	0	
HyperLogLogPlusPlus	_CE	46.40	46.77	46.88	4/4/0/0	2/4	0	2	2
简介	_OK	3.40	3.51	3.53	2	0	2	0	
保序回归	_CE	5.02	5.11	5.68	15/1/0/0	1	0/1	14	
保序回归	_OK	6.43	6.70	7.53	31	5	11	15	
杰卡德相似度	_OK	10.48	10.63	10.78	16	12	0	4	
杰卡德相似度	_UNK	104.82	109.56	428.60	10/0/2/0	6	0	4	
厨房	_OK	4.70	4.80	4.98	5	1	0	4	
Kmath	_a0_OK	8.04	8.25	8.48	12	4	5	3	3
Kmath	_a1_OK	8.68	8.77	8.82	12	4	5	3	3
LDA优化器	_false _CE	6.68	6.77	6.89	3/2/0/0	0/2	0	3	1
List	_OK	6.95	7.03	7.21	18	4	0	14	
对数损失	_CE	6.90	6.99	7.10	24/2/0/0	1	8/2	15	6
对数损失	_OK	6.69	6.83	6.87	26	2	9	15	6
MCOpt.DeepStructure	_OK	3.60	3.68	3.71	4	4	0	0	
中介	_a0_OK	4.83	4.89	5.00	13	4	0	9	
中介	_a0_UNK	6.72	6.91	7.03	7/0/3/0	1	0	6	
中介	_a1_OK	9.41	9.50	9.71	56	10	0	46	
牛顿平方	_CE	54.00	54.47	54.64	19/3/0/0	6	13/3	0	
本体统计	_a0_OK	3.51	3.56	3.68	5	1	4	0	
本体统计	_a1_OK	3.73	3.84	3.91	2	0	2	0	
优化器	_OK	3.67	3.71	3.79	7	1	1	5	
成对评分器	_OK	5.39	5.49	5.56	18	6	0	12	
玩家	_a0_OK	6.26	6.38	6.54	21	16	0	5	

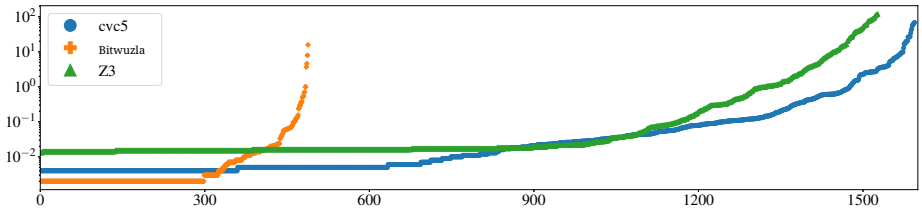
玩家	_a1	_OK	7.65	7.87	8.03	29	26	0	3	
Polyn.Mutagen	_false	_CE	14.29	14.37	14.53	9/2/0/0	1/2	0	8	2
多项式性能		_CE	4.33	4.38	4.40	12/2/0/0	0	4/2	8	
多项式性能		_TO	430.95	432.50	435.50	16/0/0/1	1	6	9	
预测指标		_CE	19.13	19.14	19.29	11/2/0/0	2/2	4	5	2
预测指标		_OK	15.69	15.83	16.10	17	4	0	13	2
配置文件		_CE	3.99	4.11	4.19	4/3/0/0	3/1	0/2	1	
配置文件		_OK	4.03	4.10	4.29	9	4	1	4	
矩形		_OK	28.62	28.70	28.81	12	11	0	1	
矩形边框	_a0	_OK	11.86	12.28	12.34	30	12	8	10	
矩形边框	_a1	_OK	7.90	8.02	8.07	27	15	6	6	
Redis存储		_OK	3.53	3.59	3.65	4	2	2	0	
RegularPayGrantCalc.		_OK	4.39	4.43	4.51	14	2	9	3	
SamplingUtils		_CE	12.52	12.80	13.76	46/2/0/0	11	33/2	2	2
SamplingUtils	_假	_CE	13.56	14.02	14.62	50/1/0/0	19	29/1	2	2
会话		_CE	30.77	33.12	35.06	21/2/0/0	6	15/2	0	
速度控制		_OK	5.56	5.61	5.76	65	24	41	0	
球形	_false	_CE	21.40	23.21	23.29	73/11/0/0	21	46/11	6	29
StatCounter		_确定	318.74	318.82	318.93	19/0/0/1	8	0	11	
StatCounter		_TO	318.73	318.81	319.04	17/1/0/1	6/1	0	11	
StormDay		_CE	3.95	3.98	4.02	2/3/0/0	0/2	0	2/1	
StormDay		_OK	4.81	4.92	4.97	7	3	0	4	
流式直方图		_CE	8.68	8.80	8.95	10/1/0/0	1/1	0	9	
流式直方图		_OK	8.78	8.85	9.53	15	5	0	10	
流式输入输出		_CE	3.58	3.60	3.65	2/1/0/0	1/1	1	0	
流式输入输出		_OK	3.38	3.47	3.54	3	1	2	0	
方差		_CE	15.25	15.44	15.61	37/2/0/0	18/2	4	15	
方差		_OK	13.81	13.93	14.12	43	19	4	20	
二维向量	_a0	_CE	33.42	33.55	33.75	7/2/0/0	3/2	0	4	
二维向量	_a0	_OK	33.64	33.72	34.12	16	8	3	5	
二维向量	_a1	_CE	34.43	34.67	34.77	6/3/0/0	2/3	1	3	
二维向量	_a1	_false _CE	6.08	6.12	6.23	18/1/0/0	6/1	0	12	1
向量	_false	_CE	42.87	43.45	44.02	201/3/0/0	64/1	0	137/2	2
加权最小二乘法		_OK	15.87	15.90	16.03	17	7	0	10	
gui		_OK	3.51	3.63	3.65	3	3	0	0	
软件包		_OK	3.73	3.79	3.80	6	6	0	0	



(a) KeY 基准测试（300 秒超时）



(b) FdLibm 基准测试（300 秒超时）



(c) 实证基准测试（超时 120 秒）

图 5：所有验证条件的求解器运行时间

```
def getPortionOutsideBorder(position: Double, minReachablePosition: Double,
     maxReachablePosition: Double): Double = {
        position match {
                case p if p < minReachablePosition => minReachablePosition - p
                case p if p > maxReachablePosition => p - maxReachablePosition
                case _ => 0 // the entity is inside the map edge => radius portion
                     outside the map is 0
        }
}
```

图 6：包含浮点比较的用户代码示例，其中一条代码注释错误地假设，假比较意味着输入有效（非 NaN）