验证金融科技系统中的数据约束等价性

Chengpeng Wang*、Gang Fan†、Peisen Yao‡、Fuxiong Pan†和Charles Zhang*
*香港科技大学计算机科学与工程系，中国香港。电子邮箱：{cwangch, charlesz}@cse.ust.hk,
†蚂蚁集团，中国深圳。电子邮箱：{fangang, fuxiong.pfx}@antgroup.com,
‡浙江大学，中国杭州。电子邮箱：pyaoaa@zju.edu.cn

arXiv:2301.11011v1 [cs.PL] 26 Jan 2023

摘要——**数据约束广泛应用于金融科技系统，用于监控数据一致性并诊断异常数据操作。然而，在开发周期中，许多等价数据约束被重复创建，导致金融科技系统运行变慢并产生不必要的警报。我们提出了 EQDAC，这是一种高效的判定过程，用于确定数据约束等价性。我们首先提出用于语义编码的符号表示，然后引入两种轻量级分析方法，分别用于反驳和证明等价性，并证明它们可在多项式时间内完成。我们在一个金融科技系统的 30,801 个数据约束上评估了 EQDAC。结果表明，EQDAC 在三小时内检测出 11,538 组等价数据约束。它还支持高效的等价搜索，平均时间开销为 1.22 秒，使系统能够在提交新数据约束时对其进行检查。**

索引词—**等价性验证、数据约束、金融科技系统**

I. 引言

随着电子商务的快速发展，金融科技系统已成为工业生产中日益重要的组成部分。它们由一簇数据库支持的应用程序构成，用于处理大量敏感数据 [1]。任何错误的数据值都可能导致系统异常行为，并造成无法估量的经济损失。为确保可靠性，通常的做法是将目标属性规定为数据约束 [2],[3] ，以进行运行时检查。如果违反了某项数据约束，开发人员便可收到警报，以便进一步诊断。

遗憾的是，开发人员不断提交数据约束，使其迅速累积，甚至可能造成冗余。2022年第一季度，全球金融科技公司A的103名开发人员提交了2,306条数据约束。由于不了解此前的提交内容，他们创建了等价的数据约束，这些约束逐渐成为技术债务 [4],，浪费计算资源并增加系统维护的负担。为消除冗余，开发人员希望在提交新约束之前检索现有的等价数据约束，从而避免重复提交。此外，质量保证团队也迫切希望定期检查数据约束库，依据等价关系寻找更多优化机会。因此，验证金融科技系统中的数据约束等价性，有助于更好地进行维护。

目标与挑战。我们的目标是设计一种判定过程，用于确定两个数据约束是否等价。然而，要找到符合工业需求的解决方案，面临着极大的挑战。首先，判定过程必须具备极高的效率，因为金融科技系统通常包含数万条数据约束，这会大幅加剧效率瓶颈。判定过程中的任何低效之处，都可能给采用该方法带来沉重负担。其次，确保可靠性并尽可能全面地证明等价性至关重要。否则，就可能删除必要的数据约束或遗漏等价的数据约束，分别导致财务损失或错失进一步优化的机会。实际上，数据约束涉及针对不同数据类型的各种操作，这增加了同时实现这些目标的难度。

现有工作。关于等价性验证，已有两条研究路线。一类技术利用指定的重写规则，检查程序是否能够通过项重写 [5], [6]转换为另一个程序。尽管重写规则在理论上能够确保可靠性，但它们只能识别形式受限的等价模式 [7],，而应用重写规则时庞大的搜索空间也会带来很大的开销 [8]。另一类方法利用逻辑公式编码程序语义，并通过调用 SMT 求解器 [9]–[11]执行符号推理。该方法为验证等价性提供了一种通用方案，但 SMT 求解器在推理大量数据约束时效率不够高。在等价聚类和搜索过程中，求解器必须被调用数千次，开销不断累积，最终降低整体效率 [12]。

洞察与解决方案。我们的核心思路源于两个重要观察。首先，不等价的数据约束通常包含不同的变量、字面量或运算符。例如，图1a 中的数据约束检查表*t*中的属性*oid*和*in*，而图1c 中的数据约束则检查属性*iid* 和*new* 。词汇差异有助于生成具体值，使两个数据约束得出不同的计算结果。其次，等价的数据约束通常具有相似的句法结构。例如，图1b 和图1d 中的数据约束仅在消除用户定义变量后，断言、分支和可交换操作数的顺序上有所不同

```
s = 'IN';
if(包含(t.ty,s))
   ASSERT(t.in  > 0);
else
   ASSERT(t.out  > 0);
ASSERT(t.amt  > 0);
ASSERT(t.oid !=  0);
```

(a)

```
如果(包含(t.ty,'IN')){
   断言(t.old  == t.new - t.in);
}否则{
   断言(t.old  == t.new  + t.out);
}
断言(t.oid !=  0);
断言(t.iid !=  0);
```

(b)

```
s = 'IN';
如果(否、包含(t.ty,s))
   assert(t.out  > 0);
else
   assert(t.new  > 0);
assert(t.amt  > 0);
ASSERT(t.iid !=  0);
```

(c)

```
ASSERT(t.iid !=  0);
ASSERT(t.oid !=  0);
如果(不 包含(t.ty,'IN'))
   现金  = t.out  + t.new;
else
   现金  = t.new - t.in;
断言(现金  == t.old);
```

(d)

图 1：数据约束示例

。同构的句法结构是其等价性的见证。因此，我们可以利用词汇差异和句法同构，高效地反驳和证明等价性，从而避免不必要的 SMT 求解，提升性能。

基于这一洞见，我们提出 EQDAC，这是一种用于等价性验证的高效判定过程。我们建立一阶逻辑（FOL）公式作为符号表示，以描述其语义。为反驳等价性，我们进行差异分析，探索符号表示并生成变量的具体值，使一个数据约束成立而另一个数据约束被违反。为证明等价性，我们进行同构分析，采用树同构算法 [13]检查这两个符号表示能否通过重新排列子句和可交换项相互转换。我们将这两种分析与 SMT 求解相结合，以判定符号表示的逻辑等价性，最终得到一个分三阶段的判定过程。

我们实现了 EQDAC，并在 Company A 的一个金融科技系统上对其进行评估。该系统总共维护着 30,801 个数据约束。借助 EQDAC，我们发现系统中有 11,538 个数据约束至少存在一个等价变体，这表明有 7,842 个数据约束是冗余的。EQDAC 在三小时内完成等价聚类，等价搜索每个数据约束耗时 1.22 秒。除 SMT 求解外，EQDAC 的各个阶段均可证明能在多项式时间内完成。对于给定的数据约束语法，我们还从理论上证明了 EQDAC 的可靠性和完备性。我们的工作已跨越了从开发仅用于学术研究的判定过程到

一个足够实用、可供部署的方案。

总之，我们的主要贡献如下：

•我们对数据约束等价问题进行了形式化定义，该问题对金融科技系统维护至关重要。

•我们提出了一种可靠且完备的判定过程 EQDAC，用于确定数据约束等价性，从而高效支持等价聚类和搜索。

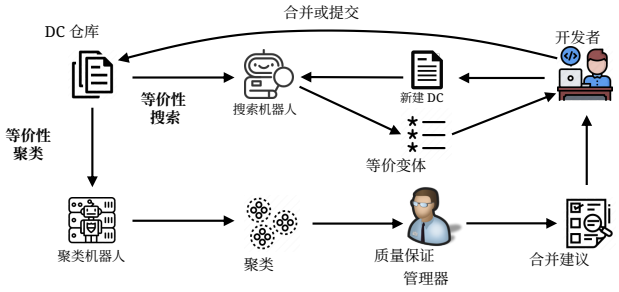


图 2：等价搜索与聚类工作流

•我们实现了 EQDAC，并在一家拥有 10 亿活跃用户的全球金融科技公司的数据约束上对其进行了评估。结果表明，它能够高效地检测出大量等价数据约束。

II. 背景与动机

本节介绍背景，并阐述我们工作的动机。

A. 金融科技系统中的等价数据约束

金融科技系统通常由一个数据库支持的应用程序簇组成，这些应用程序处理大量数据，使数据记录呈现特定属性。为提高可靠性，开发人员通常会指定数据约束来描述目标属性，并建立数据协调（DR）平台 [14]，以便在运行时对其进行检查。一旦违反数据约束，开发人员便能收到详细的运行时信息，以指导进一步的系统诊断。

实际上，许多开发团队会不断向中央 DR 平台提交数据约束。例如，Company A 约有 100 个团队每天都会主动向该平台提交数据约束。由于不了解已有的提交内容，开发人员常会提交与现有约束等价的数据约束。此外，开发人员往往不愿删除约束，因为他们不想冒漏掉数据错误的风险。最终，等价数据约束的累积成为金融科技系统的技术债务 [4]：

• DR 平台会冗余地检查等价数据约束，从而造成不必要的资源消耗，例如 CPU时间、磁盘输入输出和网络流量。

•如果违反等价数据约束，系统会向开发人员发出多条警报，这需要为诊断分配更多时间，从而延长金融科技系统的开发周期。

• 如果表结构或目标属性发生更改，则应更新所有等价数据约束，这会增加与 DR 平台交互所需的额外工作量。

因此，在维护过程中处理等价数据约束至关重要，这有助于节省资源，并简化金融科技系统的调试和重构。

B. 解决等价数据约束

为解决技术债务，Company A 的开发人员提出了两项需求，即等价聚类和

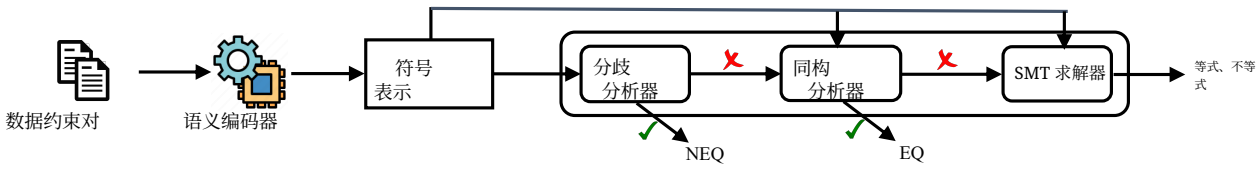


图 3：判定过程 EQDAC 的示意图

等价搜索，以处理等价数据约束。具体而言，他们希望将两个机器人集成到 CI/CD 工作流中 [15]，具体应用于金融科技系统。

- 等价搜索：开发者向机器人提交一条新的数据约束，以搜索现有的等价变体。等价变体列表有助于开发者决定是否将其与任何现有变体合并。工作流程如图 2 上半部分所示。
- 等价聚类：质量保证（QA）经理将所有数据约束导出给机器人，由机器人将数据约束划分为等价簇。随后，质量保证经理总结合并建议，并将其发送给开发人员作进一步确认。图2下半部分展示了等价聚类的工作流程。

通常，两个机器人分别从两个角度解决冗余问题。首先，等价搜索会即时检查新提交的数据约束，使开发人员能够尽可能避免冗余。其次，等价聚类支持对整个数据约束代码库进行夜间扫描。质量保证经理可以检查聚类信息，寻找合并等价数据约束的机会。在开发周期中，这两个机器人可在 CI/CD 工作流中充当防范冗余问题的两道防线。

要实现整个工作流程的自动化，我们需要一种高效的判定过程，以验证两个数据约束是否等价。具体而言，两个机器人将分别在聚类和搜索过程中调用该判定过程来确定等价性。在这项工作中，我们旨在设计一种有效的解决方案，用于验证数据约束等价性，并借助我们的判定过程促进数据约束维护。

III. EQDAC 概述

本节通过一个动机示例展示我们的洞见（§III-A），并概述我们的判定过程（§ III-B）。

A. 动机示例

在工业场景中验证数据约束等价性并非易事。首先，由于数据约束数量庞大，判定过程的成本可能会显著累积 [16]。其次，如果判定过程不具备可靠性或完备性，可能会错误剪除必要的数据约束，或遗漏等价的数据约束，从而增加数据安全风险，并错失优化机会。因此，我们需要同时确保判定过程的可靠性、完备性和效率。

图1 展示了四个数据约束示例。具体而言，图1a和图1c 分别描述了表 t 的三个属性在两种情况下具有正值，以及*oid* 和*iid* 的值分别不为 0 的属性。此外，图 1b 和图 1d 描述了账户余额的变动等于转账金额，且两个账户的标识符（即 iid 和 oid）不为 0 的属性。根据这些示例，我们可以得到以下两个重要观察结果：

- 非等价的数据约束往往包含不同的词法标记，例如数据库属性、字面量和运算符。例如，图1a和图 1c中的数据约束涉及不同的属性。生成表属性的值很可能会使它们的求值结果不同。
- 消除用户定义变量后，等价的数据约束通常仅在可交换操作数的顺序和独立语句的顺序上有所不同。例如，图 1b 和图1d中的数据约束具有同构的句法结构，这意味着它们等价。

基于这些观察，我们发现，词法差异和句法同构分别使我们能够高效地反驳和证明等价性。如果我们能生成使两个数据约束的求值结果不同的“良好”具体值，或找到句法结构之间的同构关系，就可以避免 SMT 求解并实现高效率。

B. 判定过程概述

根据我们的洞见，我们设计了一种高效、可靠且完备的判定过程，用于验证数据约束等价性。为描述数据约束的语义，我们提出语义编码，将其构造为形式受限的一阶逻辑公式，作为其符号表示，从而消除用户定义变量（例如图中的现金 （见图1d））。基于这些符号表示，我们的判定过程分三个阶段进行，如图3所示。

- 差异分析在词汇差异的指引下探索符号表示，旨在生成使数据约束求值结果不同的具体值。例如，它探索图1a中最后一条断言所导出的子句，并将属性*oid*赋值为 0，以违反该断言。此外，它还将图1c中的变量具体化，使数据约束得到满足。
- 同构分析会构建符号表示的解析树，并检查这些解析树是否同构。该分析会忽略可交换结构（例如独立语句和可交换操作数）的顺序。例如，它会发现

$\mathcal{V} := v_d \mid x\mathcal{L} := \{l_i \mid i \geq 1\}\mathcal{A} := l \mid v_d \mid a_1 \oplus a_2\mathcal{C}:$
$= a_1 \odot a_2 \mid x_1 \odot x_2 \mid a \odot x \mid x \odot a \mid p(v,l) \mid p(v_1,v_2)\mathcal{B}:$
$= c \mid b_1 \text{且} \ b_2 \mid b_1 \text{或} \ b_2 \mid \text{非} \ b \mid \text{ite}_b(c_0,b_1,b_2)\mathcal{S}:$
$= x = a \mid \text{断言}(b) \mid s_1; s_2 \mid \text{ite}_s(c_0,s_1,s_2)\mathcal{R} := s+\oplus:$
$= + \mid - \mid \times \mid \div\odot := > \mid < \mid \geq \mid \leq \mid == \mid \neq$
$\mathcal{P} := \{\text{前缀匹配、后缀匹配、包含、等于}\}$

图 4：数据约束的语法

图 1b 和图1d中的同构结构，模糊断言的顺序以及 + 和 == 的操作数。

- 如果前两种分析既无法反驳也无法证明等价性，我们就调用 SMT 求解器来检查符号表示的逻辑等价性。为确保可靠性和完备性，我们在位向量、浮点算术和字符串的组合理论中，使用一个可判定片段进行 SMT 编码。

除了可靠性和完备性之外，EQDAC 还具有复杂度方面的理论保证。符号表示构建、差异分析和同构分析的时间复杂度相对于数据约束抽象语法树的大小均为多项式时间。我们的评估也为 EQDAC 在等价聚类和搜索方面的高效率提供了有力证据。

IV. 问题表述

本节介绍语法（§ IV-A），并阐述数据约束等价性问题（§IV-B）。

A. 数据约束语法

图 4 总结了语法。变量 变量 是一个数据变量 $v_d \in \mathcal{V}_d$，表示表属性的值，或是一个用户定义变量 $x \in \mathcal{V}_u$，用于临时存储值。其值可以是有限长度整数、浮点数或字符串。字面量 字面量 是一个常量值。算术表达式算术表达式可以是字面量、数据变量或复合算术表达式。比较表达式 比较表达式 用于比较算术表达式和用户定义变量，或使用谓词检查字符串 $p \in \mathcal{P}$。布尔表达式 布尔表达式 是比较表达式或由逻辑连接词连接的复合表达式。语句 语句 是赋值、断言、顺序执行或**ite**$_s$ 语句。特别是，**ite**$_b$ 表达式和 **ite**$_s$ 语句中的条件仅涉及数据变量。最后，数据约束 数据约束 由有限条语句组成。其所有断言都应在给定数据库表的情况下成立。

该语法具有足够的表达能力，可以描述真实场景中的目标属性。它涵盖了 [3],中的所有模式，例如值比较、条件比较等。此外，用户定义的变量支持灵活地编写数据约束。算术运算和字符串谓词支持表达复杂属性，例如比较现金金额之和以及匹配字符串变量。

B. 数据约束等价问题

在陈述问题之前，我们先介绍解释和语义等价的概念如下。

定义 1. 解释 I 将每个数据变量 v_d映射到其定义域中的一个值。若在 I下所有断言均成立，则 I 是数据约束 r的一个模型，记作 $I \models r$。

例 1。 以下解释 I 是图中数据约束的一个模型1a。

$$I = [t.ty \mapsto \text{'IN'}, t.in \mapsto 1, t.out \mapsto 0, t.oid \mapsto 1, t.amt \mapsto 1]$$

解释表示表属性的取值。数据约束会导出一个解释集合，使其中的断言成立。形式上，我们定义语义等价性。

定义 2。 数据约束 r_1 和 r_2 在语义上等价，记作 $r_1 \simeq r_2$，当且仅当

$$\forall I : I \models r_1 \Leftrightarrow I \models r_2$$

例 2。 基于例 1，我们可以构造

$$I' = I[t.new \mapsto 0, t.iid \mapsto 1]$$

I' 不是图中数据约束的模型 1c，而它是图中数据约束的模型1a，这表明它们在语义上并不等价。

在本研究中，我们旨在提出一种判定过程，以验证对于给定的数据约束对 r_1 是否与 r_2 在语义上等价 (r_1, r_2)。然而，要找到一种可靠、完备且高效的解决方案颇具挑战性。从理论上讲，任何 SAT 问题实例 [17] 都可以通过在多项式时间内构造两个合适的数据约束，归约为我们问题的一个实例。形式上，我们将该问题的复杂性障碍表述如下。

定理 1. 数据约束等价问题是 NP 困难。

概述。 为验证等价性，我们提出一种符号表示来编码语义（§ V），并设计了一个高效的判定过程（§ VI）。具体而言，我们引入轻量级推理，以高效地反驳和证明等价性，这是我们的主要技术贡献。通过将轻量级推理与基于SMT的分析相结合，我们的判定过程具有可靠性和完备性，并能高效支持等价聚类和搜索。

V. 语义编码

本节介绍用于描绘语义的符号表示（§ V-A），阐述符号求值（§ V-B），并在最后总结其优势（§V-C）。

A. 符号表示

数据约束本质上是一个以数据变量为输入的程序。数据变量的值决定所有变量和表达式的值。基于这一直觉，我们提出符号项和条件的概念，用于描述变量和表达式的值。

定义 3. 符号项 τ 表示变量或字面量的值，形式如下：

- $\tau := v_d$ 或 $\tau := l$ 分别是数据变量或字面量。
- $\tau := \tau_1 \oplus \tau_2$ 是一个带有算术运算符的复合项。

定义 4。 符号条件 ϕ 是一阶逻辑公式，形式如下之一：

- 原子条件是两个符号项的算术比较或字符串比较，即 $\phi := \tau_1 \odot \tau_2$ 或$\phi := p(\tau_1, \tau_2)$，其中 $p \in \mathcal{P}$ 是字符串谓词。
- 复合条件是含有逻辑连接词的一阶逻辑公式，即 $\phi := \phi_1 \wedge \phi_2$、$\phi := \phi_1 \vee \phi_2$或 $\phi := \neg\phi_0$。

示例 3。 在图 1d中，现金 的值可以用项 $t.out + t.new$ 和 $t.new - t.in$表示。 $\textbf{ite}_s$ 语句的条件编码为 ¬包含$(t.ty,$ ‘IN’)。

符号项表示变量、字面量和算术表达式的值，而符号条件则对布尔表达式的值进行编码，为定义符号表示提供基础。

定义 5. 对于数据约束 r，其符号表示是满足以下条件的符号条件 φ

- 对于任意解释 I， $I \models r$ 当且仅当 $I \models \varphi$。
- 否定仅出现在字符串原子约束之前。

直观而言，符号表示通过一阶逻辑公式忠实地编码语义，该公式仅涉及数据变量，并排除冗余的否定。它抽象掉用户定义的变量，并有效消除否定方面的语法差异，使我们能够为等价性验证设计轻量级推理方法。接下来，我们将详细介绍如何构造符号表示。

B. 符号求值

现在，我们提出符号求值来构造符号表示。基本上，符号求值分为两个阶段，分别用于收集每个断言中布尔表达式的值，以及消除不必要的否定。在深入探讨细节之前，我们先介绍符号状态的概念。

定义 6。 给定数据约束 r，程序位置 ℓ 处的符号状态 $\mathbf{S}$ 为 **(E Φ)**，其中

- 一个环境 $\mathbf{E}$ 将变量 v 或算术表达式 e 映射到项-条件对集合 $\{(\tau, \phi)\}$，表示当 ϕ 成立时， v 或 e 的求值结果与 τ 相同。
- 一个性质 $\mathbf{\Phi}$ 是一个符号条件，用于概括程序位置 ℓ 之前 r 中的断言。

例 4。 图 1d中的第一个断言之后，我们有$\mathbf{E} = [t.iid \mapsto \{(t.iid, T)\},\ 0 \mapsto \{(0, T)\}]$ $\mathbf{\Phi} = (t.iid \neq 0)$

现在我们介绍符号求值的技术细节。在第一阶段，我们对变量和表达式求值，以获得一个描述语义的一阶逻辑公式，该公式仅与数据变量相关。具体而言，我们在图 5和图6中定义求值规则。

$$\text{ASSIGN}\ \frac{\mathbf{E} \vdash_e a \rightsquigarrow V \quad \mathbf{E}' = \mathbf{E}[v \mapsto V]}{\mathbf{E}, \mathbf{\Phi} \vdash v = a \rightsquigarrow \mathbf{E}', \mathbf{\Phi}}$$

$$\text{ASSERT}\ \frac{\mathbf{E} \vdash_b b \rightsquigarrow \psi \quad \mathbf{\Phi}' = \mathbf{\Phi} \wedge \psi}{\mathbf{E}, \mathbf{\Phi} \vdash \textbf{assert}(b) \rightsquigarrow \mathbf{E}, \mathbf{\Phi}'}$$

$$\text{SEQ}\ \frac{\mathbf{S} \vdash s_1 \rightsquigarrow \mathbf{S}_1 \quad \mathbf{S}_1 \vdash s_2 \rightsquigarrow \mathbf{S}'}{\mathbf{S} \vdash s_1; s_2 \rightsquigarrow \mathbf{S}'}$$

$$\text{ITE-S}\ \frac{\begin{array}{c}\mathbf{E} \vdash_b c_0 \rightsquigarrow \gamma_1 \quad \gamma_2 = \neg\gamma_1 \quad \mathbf{E}, \mathbf{\Phi} \vdash s_i \rightsquigarrow \mathbf{E}_i, \mathbf{\Phi}_i \\ \mathbf{E}' = [u \mapsto \bigcup_{i=1}^{2} \{(\tau_i, \phi_i \wedge \gamma_i) | (\tau_i, \phi_i) \in \mathbf{E}_i(u)\}]\end{array}}{\mathbf{E}, \mathbf{\Phi} \vdash \textbf{ite}_s(c_0, s_1, s_2) \rightsquigarrow \mathbf{E}', ite(\gamma_1, \mathbf{\Phi}_1, \mathbf{\Phi}_2)}$$

图 5：语句的求值规则

$$\text{VAR}\ \frac{u \in \mathcal{L} \cup \mathcal{V}_d \quad U = \{(u, T)\}}{\mathbf{E} \vdash_e u \rightsquigarrow U}$$

$$\text{AE}\ \frac{\begin{array}{c}a_i \in \mathcal{A} \quad \mathbf{E} \vdash_e a_i \rightsquigarrow U_i \\ A = \{(t_1 \oplus t_2, \phi_1 \wedge \phi_2) \mid (t_i, \phi_i) \in U_i\}\end{array}}{\mathbf{E} \vdash_e a_1 \oplus a_2 \rightsquigarrow A}$$

$$\text{ACmp}\ \frac{\begin{array}{c}u_i \in \mathcal{A} \cup \mathcal{V}_u \quad \mathbf{E} \vdash_e u_i \rightsquigarrow U_i \\ B = \{(t_1 \odot t_2) \wedge \phi_1 \wedge \phi_2 \mid (t_i, \phi_i) \in U_i\}\end{array}}{\mathbf{E} \vdash_b u_1 \odot u_2 \rightsquigarrow \bigvee_{\phi \in B} \phi}$$

$$\text{ITE-E}\ \frac{\mathbf{E} \vdash_b c_0 \rightsquigarrow \gamma_0 \quad \mathbf{E} \vdash_b b_i \rightsquigarrow \gamma_i}{\mathbf{E} \vdash_b \textbf{ite}_b(c_0, b_1, b_2) \rightsquigarrow (\gamma_1 \wedge \gamma_0) \vee (\gamma_2 \wedge \neg\gamma_0)}$$

Fig. 6 表达式求值规则1 i i

- 规则 赋值 使用 变量 和算术表达式规则 （见图 6）对右侧表达式求值，并对 $\mathbf{E}$执行强更新，从而确保用户定义变量 v 与表达式 a 具有相同的值。该规则成功对用户定义变量求值，使符号项仅与数据变量相关。
- 规则 ASSERT 将布尔表达式 b 求值为符号条件 ψ。然后，它通过逻辑合取将 ψ 与原始属性 $\mathbf{\Phi}$ 连接起来。由此形成一个累积断言条件的属性。
- SEQ 和 ITE-S 规则的定义十分直接。SEQ 依次应用两个组成部分的求值规则。ITE-S 分别对两种情况求值，并根据分支条件合并两个符号状态。

由于篇幅有限，我们省略字符串比较和其他复合布尔表达式的求值规则；这些规则与 ACmp 和 ITE-E 规则类似。根据这些规则，我们逐步对数据约束求值。初始时，符号状态是一个空映射与一个真 值组成的对。沿控制流路径应用各语句的规则后，我们得到每个程序位置处的符号状态，并最终用出口处的属性 $\mathbf{\Phi}_e$ 汇总所有断言，该属性描述了数据约束的语义。

示例 5。 考虑图 1d 中的数据约束。我们在其退出处得到 $\mathbf{\Phi} = \phi_1 \wedge ((\phi_2 \wedge \phi_4) \vee (\phi_3 \wedge \neg\phi_4))$，其中$\phi_1 = (t.iid \neq 0) \wedge (t.oid \neq 0)$ $\phi_2 = (t.out + t.new = t.old) \phi_3 = ($ $t.new - t.in = t.old)$ $\phi_4 = \neg$包含$(t.ty$、‘IN’)

在第二阶段，我们消除 $\mathbf{\Phi}_e$ 中不适用于原子字符串约束的否定。具体而言，我们首先将 $\mathbf{\Phi}_e$ 转换为否定范式（NNF），在该范式中，否定仅适用于原子公式。然后，我们通过更改比较运算符，消除每个原子算术约束前的否定，例如，将$\neg(t.a \geq t.b)$转换为 $t.a < t.b$。值得注意的是，上述转换可以通过对 $\mathbf{\Phi}_e$的解析树执行广度优先搜索来实现，同时动态构建符号表示。总体时间复杂度与 $\mathbf{\Phi}_e$的大小呈线性关系。

示例 6。在示例 5中，我们消除否定，得到符号表示 $\varphi = \phi_1 \wedge ((\phi_2 \wedge \phi_4) \vee \phi')$，其中$\phi' = (t.new - t.in = t.old) \wedge$包含$(t.ty$、'IN'$)$。

C. 总结

符号表示本质上是数据变量的布尔函数，具有以下三项优势：

- 符号表示保留了数据变量、字面量和运算符方面的词法差异，这些差异可能表明它们并不等价。
- 符号求值会对用户定义的变量进行求值，消除变量名称上的差异，因为这些差异不会影响语义。
- 消除不必要的否定可以规范化一阶逻辑公式，并为更多等价的数据约束生成同构的符号表示。

因此，语义编码揭示了词汇差异和句法同构，以便进行轻量级推理，从而高效地反驳和证明等价性（§ VI-A§ VI-B）。

VI. 判定过程

本节首先介绍差异分析（§ VI-A）和同构分析（§ VI-B），分别用于高效地反驳和证明等价性。随后，我们将这两种分析与 SMT 求解相结合，以建立判定过程（§ VI-C）。下文中，我们用 r_1 和 r_2 表示数据约束，并用 φ_1 和 φ_2 表示其符号表示，以供说明。

A. 差异分析

根据定义 5，φ_1 和 φ_2 忠实地描述了两个数据约束的语义。如果存在某个解释 I 使它们求值得到不同的真值，我们便可以安全地反驳其等价性。然而，要高效地获得所需的解释并非易事。随机采样可能会在多次尝试失败后成功命中所需的解释，从而显著降低效率。为解决这一问题，我们尝试探索 φ_1 和 φ_2 的特定布尔结构，并对这些结构中的数据变量进行具体化。形式上，我们引入自由度来指导探索。

定义 7. 对于两个符号表示 φ_1 和 φ_2，出现在 φ_1 中的子句 ϕ 的自由度为

$$\mathcal{DF}(\phi \mid \varphi_1, \varphi_2) = \frac{1}{h(\phi)} \cdot \sum_{M \in \{V_d, L, O\}} |M(\phi) \setminus M(\varphi_2)|$$

算法 1： 差异分析

输入： φ_1，φ_2：两个符号表示；
输出： 是否 $\exists I:\ \neg(I \models \varphi_1 \leftrightarrow I \models \varphi_2)$

```
1  对每个(φ1, φ2) ∈ {(φ1, φ2), (φ2, φ1)} 执行
2      I ← ⊥;
3      status ← T;
4      探索 (φ1, φ1, φ2, F) ;
5      探索 (φ2, φ2, φ1, T) ;
6      如果 status是 T 则
7          返回 真;
8  返回未知;
9  过程 探索 (φ, φ, φ′, tv)
10     若 status则
11         若 φ 是原子的则
12             若 自由变量(φ, I) ≠ ∅ 则
13                 I ← 具体化(φ, tv);
14             else
15                 status ← 检查(I ⊨ φ = tv);
16         否则若 (LC(φ) tv) ∈ {(∧, T), (∨, F)} 则
17             对每个 φi ∈ C(φ) 执行
18                 探索 (φi, φ, φ′, tv) ;
19         else
20             φ′ ← 最大化参数φi∈C(φ) DF(φi | φ, φ′);
21             探索 (φ′, φ, φ′, tv) ;
```

$h(\psi)$ 是 ψ的解析树高度。 $V_d(\psi)$ 包含 ψ 中的数据变量，但不包括算术操作数。 $L(\psi)$ 和 $O(\psi)$ 分别包含 ψ中的字面量和运算符。

直观地说，自由度越大，使 ϕ 求值为目标真值的可能性就越高：

- 首先， $h(\phi)$的值越小，就越有机会通过较少的探索找到所需的解释。
- 其次， $|M(\phi) \setminus M(\varphi_2)|$的值越大，表示 ϕ中存在更多 φ_2所没有的独特词法标记。 ϕ 和 φ_2 中的数据变量具体化过程交织得更少。
- 第三， $V_d(\phi)$不包括算术操作数，因为算术运算会增加具体化的难度。

示例 7。考虑图 1a 和图1c中的数据约束。根据 § V，它们的符号表示如下：

$\varphi_1 = ((t.in > 0 \wedge \phi_c) \vee (t.out > 0 \wedge \neg\phi_c)) \wedge \phi_a \wedge \phi_o\varphi_2 = ((t.out > 0 \wedge \neg\phi_c) \vee (t.new > 0 \wedge \phi_c)) \wedge \phi_a \wedge \phi_i$其中 $\phi_a = (t.amt > 0)$、$\phi_o = (t.oid \neq 0)$、 $\phi_i = (t.iid \neq 0)$，且 $\phi_c =$ 包含$(t.ty$, 'IN'$)$。令 ϕ 表示 φ_1的第一个子句。我们有 $V_d(\phi) \setminus V_d(\varphi_2) = \{t.in\}$， $L(\phi) \subseteq L(\varphi_2)$，以及$O(\phi) \subseteq O(\varphi_2)$。因此，我们有 $\mathcal{DF}(\phi \mid \varphi_1, \varphi_2) = \frac{1}{3}$。类似地，我们有 $\mathcal{DF}(\phi_a \mid \varphi_1, \varphi_2) = 0$和 $\mathcal{DF}(\phi_o \mid \varphi_1, \varphi_2) = 1$。

基于自由度，我们提出差异分析来生成所需的解释。算法 1 展示了其技术细节。它接收两个符号表示φ_1 和 φ_2，并尝试生成所需的解释，以强制它们得出不同的评估结果（第 1–7 行）。该函数

探索按层级遍历子句（第 9–21 行），并采用特定策略处理三类子句 ϕ：

- 如果 ϕ 是原子的，我们就将自由变量具体化，使 ϕ求值为 tv（第12–13行）。如果没有自由变量，我们就检查 ϕ 在 I 下是否求值为 tv （第15行）。
- 如果 ϕ 与 $\wedge$ 相连且 tv 为真，或者 ϕ 与 $\vee$ 相连且 tv 为假，我们就探索 ϕ 中的所有子句，并强制它们求值为 tv （第16–18行）。
- 否则，我们选择自由度最大的子句 ϕ'，并强制其求值为 tv（第20–21行）。

如果每个子句都求值为目标值，算法 1 就会找到所需的解释，从而反驳等价性。

示例8。 在示例 7中， φ_1 与逻辑合取相连。如果要使 φ_1 求值为 假，我们只需选择并探索其中一个子句。第三个子句 ϕ_o 的自由度比其他两个更大，因此我们选择它，并将 0 赋给 $t.oid$。同样，我们可以强制φ_2 求值为 真，最终反驳等价性。

最后值得一提的是，词法标记不同的数据约束通常并不等价，但仅凭词汇差异直接否定等价性并不可靠。相比之下，我们的差异分析实质上利用词汇差异来引导解释生成，从而能够可靠地否定等价性。

B. 同构分析

作为一阶逻辑公式，如果我们能够通过对 φ_1中的可交换子公式和项重新排序，将 φ_1 转换为 φ_2 ，那么符号表示 φ_1 和φ_2 在逻辑上等价。例如，一阶逻辑公式的求值不受逻辑析取和合取所连接的子句顺序影响。此外，对可交换算术运算符（如加法和乘法）的操作数进行任意排列，始终会得到逻辑等价的公式。换言之，通过识别 φ_1 与 φ_2 之间的同构，我们可以可靠地证明数据约束等价性。

基于上述关键思想，我们提出同构分析，以确定 φ_1 和 φ_2 的解析树是否同构，其形式化描述见算法2。算法 [13] 2 使用 AHU 算法进行树同构检验；若解析树同构，该算法即可证明数据约束等价性（第 1–2 行）。具体而言，函数 *SCT* 和*STT* 对符号表示中的子句和项进行处理，并以自顶向下的方式.....
向下的方式，分别创建解析树中的树节点和叶节点。

- 处理非原子公式 φ 时，*SCT* 会创建一个树节点来存储逻辑连接词，并附加其所有子句的解析树（第 5–6 行）。
- 对于原子条件，如果比较运算符属于 $\{=, \neq\}$ ，或者字符串谓词为等于，*SCT* 就会创建一个树节点（第 8–9 行）。否则，它会添加一个叶节点，使子树不可交换（第 10–14 行）。值得注意的是，它会规范化不等式，以便仅使用 $>$ 和 $\geq$ 来强制执行，从而有助于发现更多等价不等式。

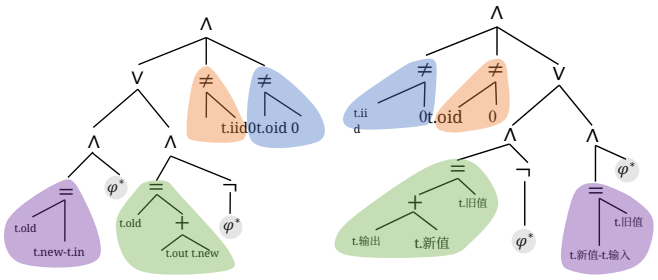
算法 2： 同构分析

输入： φ_1， φ_2：两个符号表示；
输出： 是否 $\forall I$： $I \models \varphi_1 \leftrightarrow I \models \varphi_2$

1 **若**AHUcheck($SCT(\varphi_1)$， $SCT(\varphi_2)$) **则**
2 　**返回** *true*;
3 **返回***unknown*;
4 **过程 SCT** (ϕ)
5 　**如果** ϕ： $\phi_1 \circledast \cdots \circledast \phi_k$ 且 $\circledast \in \{\wedge, \vee, \neg\}$ **则**
6 　　**返回** 树($\circledast$， $\{SCT(\phi_i) \mid 1 \leq i \leq k\}$);
7 　**否则如果** ϕ： $\tau_1 \circledast \tau_2$ 或 ϕ： $\circledast(\tau_1, \tau_2)$ **则**
8 　　**如果** $\circledast \in \{=, \neq, equals\}$ **则**
9 　　　**返回** 树($\circledast$, {$STT(\tau_1)$,$STT(\tau_2)$});
10 　　**否则如果** $\circledast \in \{<, \leq\}$ **则**
11 　　　$\circledast' \leftarrow$翻转($\circledast$);
12 　　　**返回** 叶节点($\circledast'$,$STT(\tau_1)$,$STT(\tau_2)$);
13 　　**else**
14 　　　**返回** 叶节点($\circledast$,$STT(\tau_1)$,$STT(\tau_2)$);
15 **过程 STT** (τ)
16 　**如果** $\mathrm{Op}(\tau) = \{\circledast\}$ 且 $\circledast \in \{+, *\}$ **则**
17 　　**返回** 树($\circledast$, 操作数(τ));
18 　**否则如果** τ： $\tau_1 \oplus \tau_2$ **则**
19 　　**返回** 叶节点($\oplus$,$STT(\tau_1)$,$STT(\tau_2)$);
20 　**else**
21 　　**返回** 叶节点(τ);

- *STT* 会在某个项 τ 仅使用加法或乘法时构造一个树节点（第 16–17 行）。对于其他情况，*STT* 会创建一个叶节点（第 18–21 行）。

示例 9。 图 7a和图 7b 分别展示了图1b和图 1d中数据约束的符号表示所对应的解析树。 φ^* 表示 包含($t.ty$、‘I N’)。它们的同构证明了数据约束等价性。



(a) (b)

图 7：两个同构的解析树

值得注意的是，Alg. 2中的 AHU 算法与标准算法略有不同 [13]。原本，AHU 算法会按层级对子树排序，因为子树的顺序并不重要。然而，在我们的情形中，只有树节点的子树可以任意置换。因此，我们修改了 AHU 算法，使其适用于同构分析，不对每个叶节点的子树进行排序。

算法 3： 判定过程

输入： r_1，r_2：两个数据约束；
输出： $r_1 \simeq r_2$ 是否成立

```
1 φ1, φ2 ← 获取符号代表(r1, r2);
2 如果 发散 (φ1, φ2) 为真 则
3     返回 false;
4 如果 同构 (φ1, φ2) 为真 则
5     返回 真;
6 返回 (SMT 求解(¬(φ1 ↔ φ2)) 不可满足);
```

C. 使用 EQDAC 进行等价性验证

将上述分析与 SMT 求解相结合，我们得到 Alg. QDAC 中的判定过程 E3。我们首先通过语义编码构造符号表示 φ_1 和 φ_2。如果通过前两种分析（第 2–5 行）无法反驳或证明它们等价，SMT 求解器就会在一般情形下检查 φ_1 和 φ_2 在逻辑上是否等价（第 6 行）。值得注意的是，函数 发散 会在第 2 行调用 Alg.1，并且最多两次探索 φ_1 和 φ_2 的布尔结构，从而确保第一阶段的效率。

差异分析和同构分析分别对等价性进行过度近似和不足近似。尽管这些分析并不总能确定等价性，但正如我们的评估所证明的，它们在实践中能够处理大量数据约束。形式上，我们提出两个定理来阐明理论保证，其证明见附录。

定理 2. 算法 3中第 6 行之前的步骤以多项式时间运行，时间复杂度为 N，其中 N 是两个数据约束的抽象语法树节点数的上界。

定理 3. 对于图 4中的语法，当算法 3返回 true 时，数据约束在语义上等价，反之亦然。

七、实现

我们已使用 Python 实现 EQDAC，并将其部署于金融科技公司 A。EQDAC 首先生成数据约束的抽象语法树，然后将其转换为符号表示。我们利用 Z3 SMT 求解器 [18] 支持第三阶段的 SMT 求解。具体而言，我们使用位向量、浮点算术和字符串理论，分别对有限长度整数、浮点数和字符串类型中的变量与字面量进行编码。

基于 EQDAC，我们进一步实现了两个机器人，如图 2 所示，分别用于进行等价聚类和搜索。在等价聚类中，我们通过逐对调用 EQ-DAC 来验证数据约束的等价性。具体而言，我们缓存每个数据约束的符号表示，以避免在不同调用中重复构建。类似地，我们检查等价性

在等价搜索中，将新的数据约束与每个现有约束依次进行比较，并且只为每个数据约束生成一次符号表示。

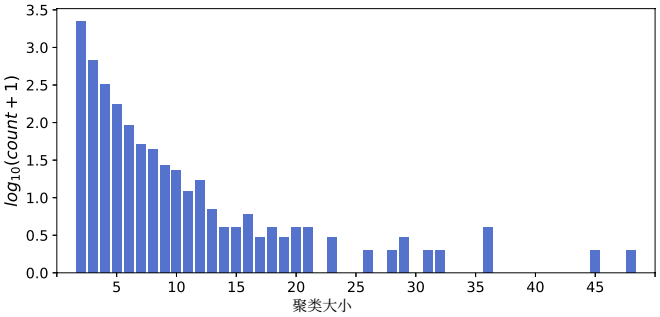


图 8：聚类的数量和大小

八、评估

为量化其有效性和效率，我们通过研究以下问题来评估 EQDAC 在金融科技系统数据约束上的表现：

- **(RQ1)** 识别出了多少个等价数据约束?
- **(RQ2)** EQDAC 在等价聚类和搜索方面的效率如何?
- **(RQ3)**三个阶段各自有多重要?

研究对象。 我们从 Company A 的一个金融科技系统中收集了 30,801 个数据约束，其语法如图 4 所示。平均而言，一个数据约束包含 9.4 个数据约束和 17.6 代码行。尽管平均规模适中，我们仍需高效处理大量数据约束，这在工业场景中虽非易事，却至关重要。最后，还有 1,497 个数据约束不符合我们的语法，因此未被选为研究对象。它们主要包含高级字符串操作，例如子字符串 和 *replaceAll*，以及系统调用，例如*getTimeZone*。在本研究中，EQDAC 专注于符合给定语法的数据约束，涵盖了 Company A 金融科技系统中大多数数据约束（95.4%）。

环境。 所有实验均在一台配备 40 个 Intel(R) Xeon(R) CPU E5-2698 v4 @ 2.20 GHz 处理器和 512 GB 物理内存的 64 位机器上进行。我们使用默认选项调用 Z3 SMT 求解器。实验的时间限制为 12 小时，内存限制为 16 GB。

可用性。 我们在 GitHub代码仓库 [19] 中发布了代码和示例约束。由于保密协议，无法分享完整的数据约束集。

A. 等价数据约束识别

为回答第一个问题，我们通过两两验证数据约束是否等价，对 30,801 条数据约束进行评估。具体而言，我们将每一对数据约束输入 EQDAC，以确定它们是否等价。

结果。 我们发现，有 11,538 个数据约束（37.5%）存在一个或多个等价变体，形成了 26,789 个等价对和 3,696 个等价簇。具体而言，我们可以在每个等价簇中保留一个数据约束，从而消除 7,842 个数据约束，同时不影响数据协调的有效性。由于权限有限，我们抽取了部分数据约束进行采样，并测量避免检查冗余约束后 CPU 时间的减少幅度。

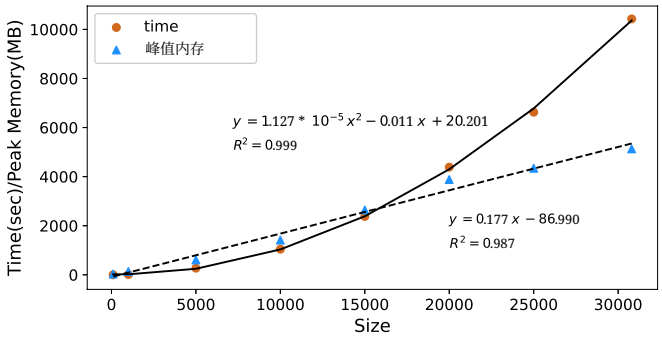


图 9：等价聚类的时间和内存开销

结果显示，CPU 时间减少比例达到 15.48%。根据专家的反馈，任何程度的减少都能在长期内为整体系统带来显著效益，因为在漫长的开发周期中，数据约束会频繁地在线检查。

我们还统计了每个簇中的数据约束数量，其中簇内的数据约束彼此等价。图8 显示，簇的大小介于 2 到 48 之间。具体而言，大小为 2 的簇有 2,233 个。对于大小为 48 的最大簇，任何一个数据约束遭到违反都会产生 48 条警报。因此，识别等价数据约束能够为复用检查结果提供实用指导，并支持消除冗余警报。

B. 性能评估

我们研究 EQDAC 在等价聚类和等价搜索中的时间消耗与内存使用情况。实验配置如下。

- 等价聚类：为量化对不同大小的数据约束集进行聚类的成本，我们构建了八个数据约束集，其大小从 100 到 30,801 不等，并测量聚类所需的时间和内存。所有数据约束均为随机选取。
- 等价搜索：我们从 30,801 个数据约束中选取 1,000 个作为最近提交的约束，并将其余约束视为已有约束。具体而言，选取的约束中有一半至少与其余集合中的一个数据约束等价，以便量化最坏情况下等价搜索的成本。

结果。如图9所示，EQDAC 在 5.01 GB 的峰值内存范围内，用时 2.89 小时完成了对 30,801 个数据约束的分析。我们通过回归分析来量化可扩展性。由于我们只为每个数据约束构建一次符号表示，并以成对方式调用判定过程，因此分别选取二次函数和线性函数作为时间开销和内存开销回归分析的模板。内存和时间的 R-平方值分别为 0.987 和 0.999。此外，二次项和线性项的系数都相当小，表明开销增长缓慢。总之，EQDAC 支持可扩展的等价聚类。

表 I：等价聚类的统计数据

变体	时间（小时）	内存（GB）	等式对数	冗余数
EQDAC-ND	OOT	7.27	141	53
EQDAC-NI	4.48	6.80	26,789	7,842
EQDAC-NS	2.13	3.94	25,952	7,296
EQDAC	2.89	5.01	26,789	7,842

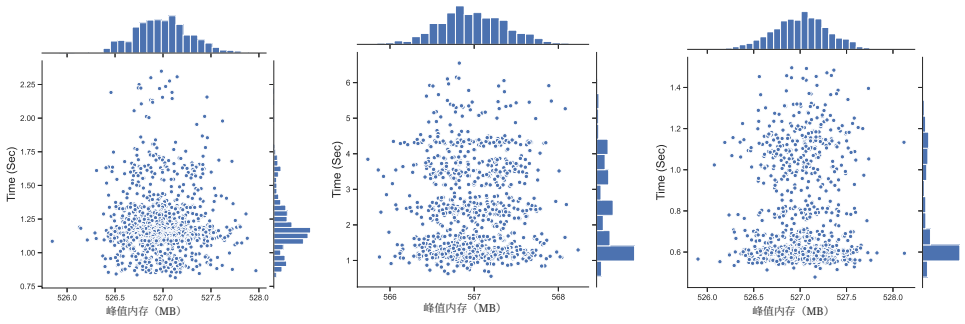
图10a 展示了等价搜索的开销。所有分析均在 528 MB 的峰值内存范围内于 2.5 秒内完成。具体而言，内存开销差异很小，介于 525.85 MB 和 527.87 MB 之间，而时间开销的变化相对较大。对若干数据约束的分析需要进行 SMT 求解，因此增加了时间开销。通常，大多数情况都能在 1.5 秒内完成分析，平均时间开销仅为 1.22 秒。因此，EQDAC 能够高效地搜索等价数据约束，这对维护至关重要。

C. 消融研究

我们设置了三种消融变体，分别是 EQDAC-ND、EQDAC-NI 和 EQDAC-NS，它们分别跳过差异分析、同构分析和 SMT 求解。前两种消融变体是可靠且完备的，而 EQDAC-NS 由于不完备，可能返回未知 。

结果。 表 I展示了等价聚类中消融研究的结果。可以看出，EQDAC-ND 在 12 小时内未能完成对 30,801 个数据约束的分析，且其峰值内存接近 7.27 GB。具体而言，EQDAC-ND 仅完成了将七个数据约束与其余数据约束逐对比较，发现了 141 个等价对和 53 个冗余数据约束。EQDAC- NI 发现的等价对与 EQDAC 相同。然而，它必须对所有未被差异分析否定等价性的约束对执行 SMT 求解，因此时间开销增加至 4.48 小时。EQDAC-NS 跳过 SMT 求解，耗时和内存占用均低于 EQDAC；其差异分析和同构分析分别耗时 1.78 小时和 0.35 小时。它未能发现 837 个等价对，因此遗漏了 546 个冗余数据约束。特别是，即使符号表示具有相同的数据变量、字面量和运算符集合，我们的差异分析仍识别出 38,964 个非等价对。因此，差异分析不仅能可靠地否定等价性，还能进一步否定更多非等价对。

图 10展示了等价搜索的开销。在每项等价搜索任务中，EQDAC-NI 的开销都更高，因为 SMT 求解器需要消耗更多资源来证明等价性。具体而言，其平均时间开销为 2.53 秒，峰值内存达到 566.98 MB。在最坏情况下，完成一个数据约束的等价搜索需要 6.56 秒，降低了其在实际生产中的可用性。EQDAC-NS 耗时较少，因为它并非在所有情况下都会调用 SMT 求解器。然而，由于完备性不足，它无法识别 37 个数据约束的等价变体。
EQDAC-ND 未能完成等价



(a) (b) (c)

图 10：EQDAC、EQDAC-NI 和 EQDAC-NS 的时间和内存开销

```
/* 数据约束 1 */
ASSERT(t.id != t.pid);
ASSERT(ut.oid != ut.iid);
如果(t.id == ut.oid){
   ASSERT(t.pid == ut.iid);
} else {
   ASSERT(t.id == ut.iid);
   ASSERT(t.pid == ut.oid);}

/* 数据约束 2 */
如果(t.id == ut.iid){
   断言(ut.oid ==t.pid);
}否则{
   断言(t.id == ut.oid);
   断言(t.pid == ut.iid);}
断言(ut.iid != ut.oid);
断言(t.pid != t.id);
```

图 11：案例研究示例

搜索：在给定的时间预算内，无法完成 1,000 个数据约束的等价搜索。它必须调用 SMT 求解器来证明非等价性，导致总体时间开销无法接受。

案例研究。图 11 展示了通过 SMT 求解发现的一对等价项。两个数据约束都检查收入账户和支出账户的标识符是否与交易中的标识符匹配。遗憾的是，我们无法从其符号表示的解析树中推导出等价性。相反，我们必须以关系方式推理多个断言。顺序执行中的两个断言是两个 **ite**s 语句等价的前提，而这超出了同构分析的能力范围。

D. 讨论

接下来，我们将讨论用户反馈、有效性威胁、局限性和未来工作。

用户反馈。EQDAC 已集成到 Company A 的生产线中，作为 CI/CD 工作流中两个机器人的核心构建模块。为收集用户反馈，我们向公司论坛中的开发人员和质量保证经理发放了问卷，并获得了用户的一致好评。例如，一位开发者在论坛中对搜索机器人作出了如下评论，表达了他对即时响应和实用结果的赞赏。

> “搜索体验真流畅！我一直期待有这样一款用于数据约束维护的助手。结果通常只需一两秒就能检索出来，有助于合并数据约束。”

效度威胁。效度威胁之一在于，生成数据约束的方式是否会影响我们的结果。开发人员之间沟通不畅，可能会增加等价数据约束的数量，因为他们并不知道其他人提交的数据约束。对于只有少量数据约束的小型金融科技系统，消除冗余带来的益处可能并不显著。EQDAC 主要面向拥有数千条数据约束的系统，在提升其可维护性方面展现出巨大潜力。

局限性与未来工作。首先，我们的语法不包括若干字符串操作。从理论上讲，求解一般的字符串约束是不可判定的 [20]–[22]，而 EQDAC 的前两个阶段在存在高级字符串操作的情况下仍可运行。尽管根据我们的经验，这些操作并不常见，但对更多字符串操作进行推理将很有意义。其次，本研究中的 EQDAC 聚焦于等价关系。考察某个数据约束是否涵盖其他约束以实现合并 [23]，是有意义的。第三，EQDAC 目前分析金融科技系统中的数据约束。将 EQDAC 扩展至其他定义域，以支持轻量级等价性检查，例如 SQL 查询 [9] 和数据库支持的程序 [24]，将很有前景。

IX. 相关工作

A. 程序等价性检查

关于程序等价性检查的研究成果十分丰富，这也是许多应用中的关键构建模块，例如翻译验证 [7]，[25]和程序综合 [26]–[28]。一类研究通过证明特定的验证条件，将等价性检查转化为验证问题，例如关系验证 [29]–[34]。类似方法还包括对无循环程序进行符号执行 [27]，[35]–[38]。不同于现有针对复杂程序结构的研究，EQDAC 聚焦于金融科技系统中的领域特定语言，优先追求效率，而非处理灵活程序语法的能力。

另一类研究通过项重写 [6]，[7]，[39]，[40]来证明程序等价性。这种方法的有效性很大程度上取决于重写规则的质量。首先，如果规则集中包含错误规则或遗漏了正确规则，可能会牺牲可靠性或完备性 [41]。其次，在重写规则数量庞大时，可能会遭遇阶段排序问题 [8] 。为提高复杂度表现， [42]限制了重写规则的形式，并采用树同构算法检查语法同构。EQDAC 在证明等价性方面与 [42] 相似，但我们在同构分析中考虑了更多程序结构，例如算术运算符和字符串谓词，从而提升了其在实践中的能力。

B. SQL 查询等价性检查

验证 SQL 查询等价性是学术界和工业界的重要课题。当前最先进的方法专注于特定形式的 SQL 查询 [43]，并采用代数推理技术 [6], [40], [44] 或符号推理 [9], [10], [45] 进行等价性验证。通常，UDP [40]利用 U-半环有效地编码 SQL 的包语义，并检查两个代数结构之间的同构。然而，它无法处理三值逻辑等高级特性，而且在同构检查过程中还受到低效追赶过程的困扰 [46]。相比之下，EQUITAS [9]使用一阶逻辑公式编码语义，并借助求解器判定等价性，能够处理比 UDP 更多的 SQL 特性 [47] 。

在我们的工作中，EQDAC 面向的是数据约束的等价性，而非 SQL 查询；不过，在技术设计方面，它与现有工作具有相似之处。具体而言，它抽象掉可交换结构的顺序，并利用求解器来判定两个一阶逻辑公式的等价性。因此，它避免了低效的追赶过程，并发挥了 SMT 求解的能力。

C. SMT 求解优化

已有大量文献研究如何利用程序特征来引导 SMT 求解。一类典型研究通过执行语义等价变换来降低 SMT 求解的开销 [12], [48]–[51]，例如， [50] 利用上下文信息简化数组约束，将具有符号索引的数组操作转换为仅涉及常量索引的操作。另一类文献利用语义信息来优化 SMT 求解算法 [52]–[55]。例如， [51], [52] 利用控制流信息来指导 CDCL(T) 风格 SMT 求解中的分支策略。类似地，我们的符号表示保留了程序语法特征，从而支持通过同构分析高效证明等价性。

除了加快求解过程之外，缓存中间结果（例如不可满足核 [56] 和语法特征 [57], ）也可以避免调用求解器。GREEN 检查约束的语法等价性，并复用等价约束先前的求解结果 [57]。这一做法与 EQDAC 的同构分析思路相似，而 EQDAC 还会进行差异分析，以高效发现非等价对，从而提高数据约束维护应用在现实场景中的效率。

X. 结论

我们提出了 EQDAC，这是一种高效、可靠且完备的判定过程，用于验证金融科技系统中的数据约束等价性。它支持全球金融科技公司生产线上的两类典型应用：等价聚类和搜索。EQDAC 能够高效处理大量数据约束，从而提升数据约束维护的效率。我们相信，EQDAC 背后的思路还能进一步推动其他领域的等价性检查。

致谢

我们感谢匿名评审对本文早期稿件提出的宝贵意见，这些意见有助于改进论文的表达。我们也感谢肖肖博士分享的真知灼见。作者获得了香港研究资助局和创新科技署提供的 RGC16206517、ITS/440/18FP 和 PRP/004/21FX 项目资助，以及蚂蚁集团的支持和微软、华为的捐赠。Peisen Yao 为通讯作者。

参考文献

[1] Feifei Li. 阿里巴巴云原生数据库系统：机遇与挑战。*VLDB*基金会论文集，12(12):2263–2272，2019年。[2] Juan Manuel Florez、Laura Moreno、Zenong Zhang、Shiyi Wei 和 Andrian Marcus。Java 中数据约束实现的实证研究。*Empir. Softw. Eng.*，27(5):119，2022年。[3] Juan Manuel Florez、Jonathan Perry、Shiyi Wei 和 Andrian Marcus。利用细粒度代码模式检索数据约束实现。载于第*44*届 *IEEE/ACM* 国际软件工程会议（*ICSE 2022*），美国宾夕法尼亚州匹兹堡，*2022*年*5*月*25*日至*27*日，第1893–1905页。ACM，2022年。[4] Neil Ernst、Rick Kazman 和 Julien Delange。实践中的技术债务：如何发现并修复。麻省理工学院出版社，2021年。[5] Andr´es Letelier、Jorge P´erez、Reinhard Pichler 和 Sebastian Skritek。语义网查询的静态分析与优化。*ACM Trans. Database Syst.*，38(4):25:1–25:45，2013年。[6] Shumo Chu、Konstantin Weitz、Alvin Cheung 和 Dan Suciu。Hottsql：使用单值SQL语义证明查询重写。载于 Albert Cohen 和 Martin T. Vechev（编），第*38*届 *ACM SIGPLAN* 编程语言设计与实现大会论文集（*PLDI 2017*），西班牙巴塞罗那，*2017*年*6*月*18*日至*23*日，第510–524页。ACM，2017年。[7] George C. Necula。优化编译器的翻译验证。载于 Monica S. Lam（编），*2000*年 *ACM SIGPLAN* 编程语言设计与实现大会（*PLDI*）论文集，加拿大不列颠哥伦比亚省温哥华，*2000*年*6*月*18*日至*21*日，第83–94页。ACM，2000年。[8] Max Willsey、Chandrakana Nandi、Yisu Remy Wang、Oliver Flatt、Zachary Tatlock 和 Pavel Panchekha。egg：快速且可扩展的等式饱和。*ACM* 编程语言汇刊，5(POPL):1–29，2021年。[9] Qi Zhou、Joy Arulraj、Shamkant B. Navathe、William Harris 和 Dong Xu。利用可满足性模理论自动验证查询等价性。*VLDB*基金会论文集，12(11):1276–1288，2019年。[10] Qi Zhou、Joy Arulraj、Shamkant Navathe、William Harris 和 Jinpeng Wu。基于符号方法证明包语义下查询等价性。*arXiv* 预印本 *arXiv:2004.00481*，2020年。[11] Sahar Badihi、Faridah Akinotcho、Yi Li 和 Julia Rubin。Ardiff：通过对公共代码进行迭代抽象与细化来扩展程序等价性检查。载于 Prem Devanbu、Myra B. Cohen 和 Thomas Zim- mermann（编），*ESEC/FSE '20*：第*28*届 *ACM* 欧洲软件工程联合会议暨软件工程基础研讨会，美国线上会议，*2020*年*11*月*8*日至*13*日，第13–24页。ACM，2020年。[12] Malik Bouchet、Byron Cook、Bryant Cutler、Anna Druzkina、Andrew Gacek、Liana Hadarean、Ranjit Jhala、Brad Marshall、Daniel Peebles、Neha Rungta、Cole Schlesinger、Chriss Stephens、Carsten Varming 和 Andy Warfield。阻止公共访问：访问控制策略的信任与安全验证。载于 Prem Devanbu、Myra B. Cohen 和 Thomas Zim- mermann（编），*ESEC/FSE '20*：第*28*届 *ACM* 欧洲软件工程联合会议暨软件工程基础研讨会，美国线上会议，*2020*年*11*月*8*日至*13*日，第281–291页。ACM，2020年。[13] Alfred V. Aho、John E. Hopcroft 和 Jeffrey D. Ullman。计算机算法的设计与分析。Addison-Wesley，1974年。[14] Abdelghani Bakhtouchi。数据协调与融合方法：综述。*Applied Computing and Informatics*，2020年。[15] Fiorella Zampetti、Salvatore Geremia、Gabriele Bavota 和 Massimil- iano Di Penta。CI/CD流水线的演进与重构：一项定性与定量研究。载于*IEEE*国际软件维护与演化会议（*ICSME 2021*），卢森堡，*2021*年*9*月*27*日至*10*月*1*日，第471–482页。IEEE，2021年。

[16] Junwen Yang、Utsav Sethi、Cong Yan、Alvin Cheung 和 Shan Lu。管理数据库支持的 Web 应用程序中的数据约束。载于 Gregg Rothermel 和 Doo-Hwan Bae （编），ICSE '20：第 42 届国际软件工程会议，韩国首尔，2020年6月27日至7月19日，第1098–1109页。ACM，2020年。[17] Stephen A. Cook。定理证明过程的复杂度。载于 Michael A. Harrison、Ranan B. Banerji 和 Jeffrey D. Ullman（编），第3届 ACM年度计算理论研讨会论文集，1971年5月3日至5日，美国俄亥俄州沙克高地，第151–158页。ACM，1971年。[18] Leonardo Mendonc¸a de Moura 和 Nikolaj Bjørner。Z3：高效的 SMT 求解器。载于 C. R. Ramakrishnan 和 Jakob Rehof（编），《系统构造与分析工具和算法》，第14届国际会议（TACAS 2008），作为《欧洲软件理论与实践联合会议》（ETAPS 2008）的一部分在匈牙利布达佩斯举行，2008年3月29日至4月6日。论文集，计算机科学讲义第4963卷，第337–340页。施普林格，2008年。[19] EQDAC。数据约束等价性验证。https://github. com/EqDAC/EqDACTool，2022年。 [线上；访问日期：9月7日，2022]。[20] Taolue Chen、Yan Chen、Matthew Hague、Anthony W Lin 和 Zhilin Wu。带有 replaceAll 函数的字符串约束有哪些是可判定的？ACM 编程语言汇刊，2(POPL):1–29，2017年。[21] Adam Kiezun、Vijay Ganesh、Shay Artzi、Philip J. Guo、Pieter Hooimeijer 和 Michael D. Ernst。HAMPI：用于字符串上的字方程、正则表达式和上下文无关文法的求解器。ACM Trans. Softw. Eng. Methodol.，21(4):25:1–25:28，2012年。[22] Taolue Chen、Matthew Hague、Anthony W Lin、Philipp R¨ummer 和 Zhilin Wu。用于处理字符串的复杂操作程序路径可行性的判定过程。ACM 编程语言汇刊，3(POPL):1–30，2019年。[23] Marcelo Sousa、Isil Dillig、Dimitrios Vytiniotis、Thomas Dillig 和 Christos Gkantsidis。使用用户定义函数合并查询。载于 Michael F. P. O'Boyle 和 Keshav Pingali（编），ACM SIGPLAN 编程语言设计与实现大会（PLDI '14），英国爱丁堡，2014年6月9日至11日，第554–564页。ACM，2014年。[24] K. Venkatesh Emani、Karthik Ramachandra、Subhro Bhattacharya 和 S. Sudarshan。从数据库应用程序中的命令式代码提取等价 SQL。载于 Fatma ¨Ozcan、Georgia Koutrika 和 Sam Madden（编），2016年国际数据管理会议论文集（SIGMOD 2016 年会），美国加利福尼亚州旧金山，2016年6月26日至7月1日，第1781–1796页。ACM，2016年。[25] Thomas Arthur Leck Sewell、Magnus O. Myreen 和 Gerwin Klein。经验证的操作系统内核的翻译验证。载于 Hans-Juergen Boehm 和 Cormac Flanagan（编），ACM SIGPLAN 编程语言设计与实现大会（PLDI '13），美国华盛顿州西雅图，2013年6月16日至19日，第471–482页。ACM，2013年。[26] Eric Schkufza、Rahul Sharma 和 Alex Aiken。随机超优化。载于 Vivek Sarkar 和 Rastislav Bod´ık（编），《编程语言与操作系统的体系结构支持》（ASPLOS 2013），美国得克萨斯州休斯敦，2013年3月16日至20日，第305–316页。ACM，2013年。[27] Malavika Samak、Deokhwan Kim 和 Martin C. Rinard。综合替换类。ACM 编程语言汇刊，4 33，2020 52 1 52 (POPL): : – : 。[28] Chandrakana Nandi、Max Willsey、Adam Anderson、James R. Wilcox、Eva Darulova、Dan Grossman 和 Zachary Tatlock。利用等式饱和与逆变换合成结构化 CAD 模型。载于 Alastair F. Donaldson 和 Emina Torlak（编），第41届 ACM SIGPLAN 编程语言设计与实现国际会议论文集（PLDI 2020），英国伦敦，2020年6月15日至20日，第31–44页。ACM，2020年。[29] Federico Mora、Yi Li、Julia Rubin 和 Marsha Chechik。客户端特定等价性检查。载于 Marianne Huchard、Christian K¨astner 和 Gordon Fraser（编），第33届 ACM/IEEE 自动化软件工程国际会议论文集（ASE 2018），法国蒙彼利埃，2018年9月3日至7日，第441–451页。ACM，2018年。[30] Anna Trostanetski、Orna Grumberg 和 Daniel Kroening。针对程序版本语义差异的模块化按需分析。载于《国际静态分析研讨会》，第405–427页。施普林格，2017年。

[31] Hiroshi Unno、Tachio Terauchi 和 Eric Koskinen。基于约束的关系验证。发表于计算机辅助验证国际会议，第742–766页。施普林格，2021年。[32] Marcelo Sousa 和 Isil Dillig。用于验证 k 安全属性的笛卡尔霍尔逻辑。发表于 Chandra Krintz 和 Emery D. Berger 编，第*37*届 *ACM SIGPLAN* 编程语言设计与实现大会论文集，*PLDI 2016*，圣巴巴拉，加利福尼亚州，美国，*2016*年*6*月*13*日至*17*日，第57–69页。ACM，2016年。[33] Berkeley R. Churchill、Oded Padon、Rahul Sharma 和 Alex Aiken。用于等价性检查的语义程序对齐。发表于 Kathryn S. McKinley 和 Kathleen Fisher 编，第*40*届 *ACM SIGPLAN* 编程语言设计与实现大会论文集，*PLDI 2019*，凤凰城，亚利桑那州，美国，*2019*年*6*月*22*日至*26*日，第1027–1040页。ACM，2019年。[34] Timos Antonopoulos、Paul Gazzillo、Michael Hicks、Eric Koskinen、Tachio Terauchi 和 Shiyi Wei。用分解取代自组合，以证明不存在时序信道。发表于第*38*届 *ACM SIGPLAN* 编程语言设计与实现大会论文集，PLDI 2017，第362–375页，纽约，纽约州，美国，2017年。计算机协会。[35] Suzette Person、Matthew B. Dwyer、Sebastian G. Elbaum 和 Corina S. Pasareanu。差分符号执行。发表于 Mary Jean Harrold 和 Gail C. Murphy 编，第*16*届 *ACM SIGSOFT*软件工程基础国际研讨会论文集，*2008*年，亚特兰大，佐治亚州，美国，*2008*年*11*月*9*日至*14*日，第226–237页。ACM，2008年。[36] Shuvendu K Lahiri、Chris Hawblitzel、Ming Kawaguchi 和 Henrique Rebˆelo。Symdiff：一种面向命令式程序的语言无关语义差异工具。发表于计算机辅助验证国际会议，第712–717页。施普林格，2012年。[37] Sahar Badihi、Faridah Akinotcho、Yi Li 和 Julia Rubin。Ardiff：通过对公共代码进行迭代抽象与细化来扩展程序等价性检查。发表于第*28*届 *ACM* 欧洲软件工程会议与软件工程基础研讨会联合会议论文集，第13–24页，2020年。[38] Jia Chen、Jiayi Wei、Yu Feng、Osbert Bastani 和 Isil Dillig。使用强化学习进行关系验证。*ACM* 编程语言汇刊，3(OOPSLA)：1–30，2019年。[39] Benjamin Goldberg、Lenore D. Zuck 和 Clark W. Barrett。深入循环：优化编译器翻译验证中的实际问题。电子理论计算机科学札记，132(1)：53–71，2005年。[40] Shumo Chu、Brendan Murphy、Jared Roesch、Alvin Cheung 和 Dan Suciu。用于判定 SQL 查询语义等价的公理化基础与算法。*VLDB*基金会论文集，11 1495，2018年 11 1482( )：–。[41] Chandrakana Nandi、Max Willsey、Amy Zhu、Yisu Remy Wang、Brett Saiki、Adam Anderson、Adriana Schulz、Dan Grossman 和 Zachary Tatlock。使用等式饱和进行重写规则推断。*ACM* 编程语言汇刊，5(OOPSLA)：1–28，2021年。[42] Simon Guilloud 和 Viktor Kuncak。对正交补双半格进行等价性检查，时间复杂度为对数线性。发表于 Dana Fisman 和 Grigore Rosu 编，系统构造与分析工具和算法——第*28*届国际会议，*TACAS 2022*，作为欧洲软件理论与实践联合会议 *ETAPS 2022* 的一部分举行，慕尼黑，德国，*2022*年*4*月*2*日至*7*日，论文集，第二部分，计算机科学讲义第13244卷，第196–214页。施普林格，2022年。[43] Phokion G. Kolaitis 和 Moshe Y. Vardi。合取查询包含与约束满足。发表于 Alberto O. Mendelzon 和 Jan Paredaens 编，第十七届 *ACM SIGACT-SIGMOD-SIGART* 数据库系统原理研讨会论文集，*1998*年*6*月*1*日至*3*日，西雅图，华盛顿州，美国，第205–213页。ACM出版社，1998年。[44] Shumo Chu、Daniel Li、Chenglong Wang、Alvin Cheung 和 Dan Suciu。Cosette 自动化 SQL 证明器演示。发表于 Semih Salihoglu、Wenchao Zhou、Rada Chirkova、Jun Yang 和 Dan Suciu 编，*2017*年 *ACM* 国际数据管理会议论文集，*SIGMOD 2017*会议，*19*，*2017*年，*14*1591 1594 2017芝加哥，伊利诺伊州，美国，*5*月 -第–页。ACM，。[45] Chenglong Wang、Alvin Cheung 和 Rastislav Bod´ık。加速关系查询的符号推理。*ACM* 编程语言汇刊，2(OOPSLA)：157:1–157:25，2018年。[46] Ronald Fagin、Phokion G. Kolaitis、Ren´ee J. Miller 和 Lucian Popa。数据交换：语义与查询应答。发表于 Diego Calvanese、Maurizio Lenzerini 和 Rajeev Motwani 编，数据库理论——

ICDT 2003，第九届国际会议，锡耶纳，意大利，*2003*年*1*月*8*日至*10*日，论文集，第2572卷，收录于计算机科学讲义，第207–224页。施普林格，2003年。

[47] Qi Zhou、Joy Arulraj、Shamkant Navathe、William Harris 和 Jinpeng Wu。SIA：使用学习谓词优化查询。载于 Guoliang Li、Zhanhuai Li、Stratos Idreos 和 Divesh Srivastava（编），*SIGMOD '21*：国际数据管理会议，线上会议，中国，*2021*年*6*月*20*日至*25*日，第 2169–2181 页。ACM，2021年。

[48] Dongpeng Xu、Binbin Liu、Weijie Feng、Jiang Ming、Qilong Zheng、Jing Li 和 Qiaoyan Yu。提升 SMT 求解器处理混合位运算-算术表达式的性能。载于 Stephen N. Freund 和 Eran Yahav（编），*PLDI '21*：第*42*届 *ACM SIGPLAN* 编程语言设计与实现国际会议，线上会议，加拿大，*2021*年*6*月*20*日至*25*日，第 651–664 页。ACM，2021年。

[49] Cristian Cadar，. 符号执行的定向程序转换 -。载于 Elisabetta DiNitto、MarkHarman 和 Patrick Heymans（编），*2015*年第十届软件工程基础联合会议论文集，*ESEC/FSE 2015*，意大利贝加莫，*2015*年*8*月*30*日至*9*月*4*日，第906–909页。ACM，2015年。

[50] David Mitchel Perry、Andrea Mattavelli、Xiangyu Zhang 和 Cristian Cadar。加速符号执行中的数组约束。载于 Tevfik Bultan 和 Koushik Sen（编），第*26*届 *ACM SIGSOFT* 软件测试与分析国际研讨会论文集，圣巴巴拉，加利福尼亚州，美国，*2017*年*7*月*10*日至*14*日，第68–78页。ACM，2017年。[51] Jianhui Chen 和 Fei He。在程序验证的 SMT 求解中利用控制流知识。*ACM* 软件工程与方法学汇刊，30(4):41:1–41:26，2021年。

[52] Jianhui Chen 和 Fei He。用于程序验证的引导式 SMT 控制器。在 Marianne Huchard、Christian K¨astner 和 Gordon Fraser（编），《第 *33* 届 *ACM/IEEE* 自动化软件工程国际会议论文集》，*ASE 2018*，蒙彼利埃，法国，*2018*年*9*月*3*日至7日，第 351–361 页。ACM，2018年。

[53] Hongyu Fan、Weifeng Liu 和 Fei He。用于多线程程序验证的引导式 SMT 求解关系。载于 Jaejin Lee、Kunal Agrawal 和 Michael F. Spear（编），*PPoPP '22*：第 *27* 届 *ACM SIGPLAN* 并行程序原理与实践研讨会，首尔，大韩民国，*2022*年*4*月*2*日至*6*日，第 163–176 页。ACM，2022年。

[54] Qingkai Shi、Peisen Yao、Rongxin Wu 和 Charles Zhang。无需路径条件的路径敏感稀疏分析。载于第*42*届 *ACM SIGPLAN* 编程语言设计与实现国际会议论文集，第 930–943 页，2021年。[55] Ziqi Shuai、Zhenbang Chen、Yufeng Zhang、Jun Sun 和 Ji Wang。用于符号执行的类型与区间感知的数组约束求解。载于第*30*届 *ACM SIGSOFT* 软件测试与分析国际研讨会论文集，第 361–373 页，2021年。[56] Jan Mr´azek、Martin Jon´as、Vladim´ır Still、Henrich Lauko 和 Jiri Bar- nat。在 symdivine 中优化和缓存 SMT 查询——（竞赛贡献）。载于 Axel Legay 和 Tiziana Margaria 编，系统构造与分析工具和算法——第*23*届国际会议（*TACAS 2017*），作为欧洲软件理论与实践联合会议（*ETAPS 2017*）的一部分于*2017*年*4*月*22*日至*29*日在瑞典乌普萨拉举行，论文集，第二部分，卷 10206，计算机科学讲义，第 390–393 页，2017年。[57] Willem Visser、Jaco Geldenhuys 和 Matthew B. Dwyer。Green：在程序分析中缩减、重用和回收约束。载于 Will Tracz、Martin P. Robillard 和 Tevfik Bultan 编，第*20*届 *ACM SIGSOFT* 软件工程基础研讨会（*FSE-20*），*SIGSOFT/FSE'12*，*2012*年*11*月*11*日至*16*日，美国北卡罗来纳州卡里，第 58 页。ACM，2012 年。

附录

在本补充材料中，我们提供论文中各定理的证明。

A. 复杂性障碍

下面我们给出定理1的证明。

证明。 我们只需证明，可以在多项式时间内将 SAT 问题的任意实例归约为数据约束等价问题的一个实例。

考虑任意一个命题逻辑公式 ψ，其中包含 n 个变量，记作 a_i $(1 \le i \le n)$。首先，我们构造一个包含 n 个属性的数据库表，即 v_i，其中 $1 \le i \le n$。同时，我们引入一个包含 n 个互不相同常量的常量集，记作 $L = \{\ell_1, \ell_2, \cdots, \ell_n\}$。接着，我们构造断言语句 **断言(**$f(v_1, v_2, \cdots, v_n)$**)**。其中，布尔表达式 $f(v_1, v_2, \cdots, v_n)$ 是通过将 ψ 中的 a_i 替换为相等约束 $v_i == \ell_i$ 构造而成的。这样的断言语句恰好就是我们所需的数据约束 r。显然，ψ 不可满足，当且仅当 r 在语义上等价于

$$\textbf{assert}(f(v_1, v_2, \cdots, v_n) \textbf{ and } \neg f(v_1, v_2, \cdots, v_n))$$

该归约可在线性时间内完成，时间复杂度与公式 ψ 的大小成线性关系。因此，数据约束等价性问题是 NP 困难的。□

B. 复杂度

在证明定理 2之前，我们首先提出并证明以下两个引理。

引理 1。 我们将符号条件 ϕ 的大小定义如下：

$$\delta(\phi) = \begin{cases} 1 & \phi \text{ is atomic} \\ 1 + \delta(\phi_0) & \phi = \neg\phi_0 \\ \delta(\phi_1) + \delta(\phi_2) & \phi = \phi_1 \vee \phi_2 \text{ or } \phi = \phi_1 \wedge \phi_2 \end{cases}$$

给定任意数据约束 r，用 N表示其抽象语法树的节点数。其符号表示$\delta(\varphi)$ 的大小关于 N 是多项式的。

证明。 为便于说明，我们引入两个函数 α 和 β:

- α**(E** e**)** 是 e 可能等于的项的数量。
- β**(E** e**)** 是在 e 等于某个特定项的条件下，符号条件的最大大小。即，β**(E** e**)** $=\max_{(\tau,\phi)\in\mathbf{E}(e)} \delta(\phi)$。

根据图 5 和图 6，**E** 仅由以下规则更新：赋值、SEQ和ITE-S。

- 考虑应用赋值规则后的 $\alpha(\mathbf{E}', v)$ 和 $\beta(\mathbf{E}', v)$。如果赋值规则应用变量规则，则有

$$\alpha(\mathbf{E}', v) = 1, \quad \beta(\mathbf{E}', v) = 1$$

如果赋值规则应用算术表达式规则，则有

$$\alpha(\mathbf{E}', v) = 1, \quad \beta(\mathbf{E}', v) = \beta(\mathbf{E}, a_1) + \beta(\mathbf{E}, a_2)$$

- 应用 ITE-S 规则后，对于任意 $e \in dom(\mathbf{E}')$，有

$$\alpha(\mathbf{E}', e) = O(\alpha(\mathbf{E}_1, e) + \alpha(\mathbf{E}_2, e))$$

$$\beta(\mathbf{E}', e) = \max_{i\in\{1,2\}} \beta(\mathbf{E}_i, e) = O(\beta(\mathbf{E}_1, e) + \beta(\mathbf{E}_2, e))$$

- 应用规则 SEQ 后，相关规则的效果会累积。

根据上述等式，我们可以发现，$\alpha(\mathbf{E}\ e)$ 和$\beta(\mathbf{E}\ e)$ 都与规则应用的次数呈线性关系。因此，对于任何程序位置，我们都有

$$\max_{e \text{ in } r} \alpha(\mathbf{E}, e) = O(N), \quad \max_{e \text{ in } r} \beta(\mathbf{E}, e) = O(N)$$

现在，我们可以估计 $\delta(\varphi)$ 的上界。根据图 5 和图 6中的规则，$\mathbf{\Phi}$ 仅由以下规则更新：ASSERT、ITE-S和SEQ。

- 考虑应用规则 ASSERT 后的 $\delta(\mathbf{\Phi}')$。如果规则 ASSERT 应用规则 ACmp，则有

$$\delta(\mathbf{\Phi}') - \delta(\mathbf{\Phi}) = \sum_{(t_i,\phi_i)\in\mathbf{E}_i(u_i)} (1+\delta(\phi_1)+\delta(\phi_2)) = O(N^2)$$

如果规则 ASSERT 应用规则 ITE-E，则有

$$\delta(\mathbf{\Phi}') - \delta(\mathbf{\Phi}) = \delta(\gamma_1) + \delta(\gamma_2) + 3$$

注意，γ_1 和 γ_2 是通过应用规则 ACmp 和 ITE-E 得到的。将上述两个等式相加，可得

$$\delta(\mathbf{\Phi}') - \delta(\mathbf{\Phi}) = O(N^3)$$

- 应用规则 ITE-S 后，有如下关系：

$$\delta(\mathbf{\Phi}') = 3 + \delta(\mathbf{\Phi}_1) + \delta(\mathbf{\Phi}_2)$$

- 规则 SEQ 累积了作用于 $\delta(\mathbf{\Phi})$ 的影响。

将上述所有等式相加，得到

$$\delta(\mathbf{\Phi}_e) = O(N^4)$$

注意，否定消去可以减小一阶逻辑公式的大小。因此，我们有

$$\delta(\varphi) \le \delta(\mathbf{\Phi}_e) = O(N^4)$$

值得注意的是，$\max_{e \text{ 中的 } r} \alpha(\mathbf{E}\ e)$、$\max_{e \text{ 中的 } r} \beta(\mathbf{E}\ e)$ 和 $\delta(\mathbf{\Phi}_e)$ 的估计上界并不紧确。为便于说明，我们仅尝试用 N 的多项式函数对它们进行界定。这些上界还可以通过特定抽象语法树节点数量的多项式函数进一步加强。 □

引理 2. 给定一个数据约束 r，其符号表示 φ 可以在关于 N 的多项式时间内构造，其中 N 是 r 的抽象语法树的节点数。

证明。 我们考察图5和图 6中的规则的时间复杂度。规则 ACmp 和 算术表达式规则 需要迭代两个集合并逐对处理，因此可在 $O(N^2)$时间内应用。其他规则均可在 $O(1)$ 时间内应用，因为它们只需构造常数个一阶逻辑公式。上述每条规则最多应用 $O(N)$ 次。因此，最终可以在 $O(N^3)$内构造符号表示。 □

现在，我们给出定理 2 的证明如下。

证明。 我们通过证明三个部分来证明该定理。

- 首先，我们可以根据引理 φ_1 2得出 φ_2 和 N 可以在多项式时间内构造出来。
- 其次，差异分析实际上遍历了 φ_1 和 φ_2的解析树，其大小都关于 N是多项式级的。引理 1 指出 $\delta(\varphi_1)$ 和 $\delta(\varphi_2)$ 关于 N都是多项式级的。因此，差异分析也能在多项式时间内完成。
- 第三，函数 *SCTree* 在$O(\delta(\varphi_1) + \delta(\varphi_2))$ 时间内构造解析树，其时间复杂度为多项式级别，相对于 N。AHU 算法也能在 $O(M)$ 时间内运行，其中 M 是树的节点数。根据引理 1，M 相对于 N是多项式级别。因此，同构分析可在多项式时间内完成。

因此，算法 3中第 6 行之前的步骤运行时间相对于 N是多项式级别。 □

在本节末尾，我们想强调以下两点。第一，在证明这两个引理时，我们省略了图 5 和图 6 中未展示的若干规则的讨论，例如，使用逻辑连接词计算布尔表达式的规则。不过，相关论证与证明中讨论的规则类似。事实上，对于图 4所示语法中的任意数据约束，这两个引理都成立。第二，我们没有给出复杂度的紧确估计。如引理 1的证明所示，如果要给出紧确上界，$\delta(\mathbf{\Phi}_e)$ 的上界将相当复杂，还需考虑不同类型程序结构的数量。在本文中，我们仅旨在说明，SMT 求解之前的步骤可在多项式时间内完成，而从理论上讲，SMT 求解可能会消耗指数级时间。

C. 可靠性与完备性

本节中，我们证明其可靠性与完备性。

证明。如果两个数据约束 r_1 和 r_2 在语义上等价，差异分析不会返回 *true*，因为它无法找到一种解释，使它们的符号表-示 φ_1 和 φ_2 求值后得到不同的真值。同时，φ_1 和 φ_2 本质上是位向量理论、浮点算术理论和字方程片段中的一阶逻辑公式，这些公式在理论上是可判定的。因此，SMT 求解必定会终止并返回 不可满足，使算法 3 返回*true*。

如果算法 3返回 *true*，则同构分析返回*true* ，或者 SMT 求解返回 不可满足。在第一种情况下，两个符号表示 φ_1 和 φ_2的解析树同构，这表明在给定解释下，它们的求值必定得到相同的真值，因此这些数据约束在语义上等价。在第二种情况下，我们有 φ_1和 φ_2 逻辑等价，这意味着数据约束等价性成立。因此，EQDAC 是可靠且完备的。 □