arXiv:2301.10556v1 [cs.LO] 25 Jan 2023

显式依赖综合*

Priyanka Golia[1,2],Subhajit Roy[1],和 Kuldeep S. Meel[2]

[1] 印度坎普尔理工学院，印度[2] 新加坡国立大学，
新加坡

摘要。 量化布尔公式（QBF）通过引入量化扩展了命题逻辑 $\forall, \exists$。在 QBF 中，作用域内的每个存在量化变量都可以依赖于所有全称量化变量。依赖量化布尔公式（DQBF）则限制存在量化变量的依赖关系。在 DQBF 中，存在量化变量显式依赖于一部分全称量化变量，这些依赖关系称为亨金依赖。给定输入与输出之间的布尔规约，亨金综合问题是将每个输出变量综合为其亨金依赖的函数，使规约得以满足。亨金综合有广泛的应用，包括部分电路验证、控制器综合和电路可实现性。

本文提出了一种名为 Manthan3 的数据驱动方法，用于 Henkin 综合。通过对来自以往 DQBF 求解竞赛的 563 个以上实例进行广泛评估，我们证明了 Manthan3 的性能可与最先进的工具相媲美。此外，Manthan3 还能为 26 个基准测试综合出 Henkin 函数，而现有最先进技术均无法完成这项任务。

1 引言

量化布尔公式（QBF）通过为命题变量引入全称量词（$\forall$）和存在量词（$\exists$），扩展了命题逻辑。在 QBF 中，存在量化变量可以依赖其作用域内的所有全称量化变量。另一方面，Henkin 量词（通常称为分支量词）推广了标准量化，并允许显式声明依赖关系 [30]。为命题逻辑引入 Henkin 量词后，便得到所谓的依赖量化布尔公式（DQBF）。在 DQBF 中，存在量化变量可以依赖预先定义的全称量化变量子集，这些依赖关系称为 Henkin 依赖。例如，$\phi: \forall x_1, x_2 \exists^{x_1} y_1 . \varphi(x_1, x_2, y_1)$ 是一个 DQBF 公式，其中 φ 是某个无量词布尔公式，而存在量化变量 y_1 仅允许依赖于 x_1，后者是与y_1对应的 Henkin 依赖。依赖规约量化称为 Henkin 量词 [30]。

*将发表于 DATE 2023。Manthan3 已开源，地址为 https://github.com/meelgroup/manthan。

这些显式依赖关系使 DQBF 比 QBF 具有更简洁的表达能力。然而，已有研究证明 DQBF 属于 NEXPTIME 完备 [35], 复杂度类，而 QBF仅为PSPACE 完全 [15]。尽管计算复杂度有所提高，DQBF 仍有广泛的应用，例如工程变更排序 [28], 、拓扑约束综合 [6], 、部分函数等价性检查 [16], 、为不完全信息博弈寻找策略 [35], 、控制器综合 [9], 、电路可实现性 [6], 以及线性时序逻辑片段的综合 [10]。

DQBF 可满足性是一类判定问题，旨在回答以下问题：是否存在一个函数，分别对应每个存在量化变量，并以其亨金依赖为自变量，使得将这些函数代入存在量化变量的位置后，所得公式成为重言式？由于 DQBF 能够表示的应用种类繁多，近年来人们对 DQBF 求解的兴趣日益高涨 [16,14,40,43,44]。在许多情况下，仅仅给出真/假答案并不足够，因为人们往往还希望确定与这些函数对应的定义。例如，在工程变更排序（ECO）中，除了了解给定电路能否修正以满足黄金规约之外，人们也会希望推导出相应的修补函数 [28]。鉴于这些函数依赖关系的命名方式，我们将这类修补函数称为亨金函数。

近年来，人们对亨金函数综合问题的兴趣日益浓厚。当前最先进的技术 HQS2 [44] 和 Pedant [40] 除了解决 DQBF 外，还能为真 DQBF 综合亨金函数。HQS2 [45] 通过对 DQBF 实例进行一系列变换来消除量词，从而为真实例综合亨金函数；而 Pedant [40] 则采用基于插值的定义提取，并调用各种 SAT 神谕来综合亨金函数。尽管多年来取得了显著进展，许多现实世界中的实例仍超出了亨金函数综合引擎的能力范围。

在这项工作中，我们迈出一步，拓展亨金综合的边界。为此，我们提出了一种名为 Manthan3 的新型亨金函数综合框架。Manthan3 采用了与现有技术截然不同的方法，将机器学习的进展与自动推理相结合。具体而言，Manthan3 使用约束采样生成数据，随后将数据输入机器学习算法，使其根据每个存在量化变量的亨金依赖学习候选函数。接着，Manthan3 使用 SAT 求解器检查综合所得候选项的正确性。如果候选项验证检查未通过，Manthan3 就会执行反例驱动的候选修复。此外，Manthan3 采用基于最大可满足性求解器的方法，找出需要修复的候选项，并使用证明引导策略构造良好的修复。

为展示 Manthan3 的实际效率，我们在包含 563个实例 的基准测试套件上，将其与此前最先进的技术 HQS2 和 Pedant 进行了广泛比较。实证评估表明，Manthan3 的性能具有竞争力，并为

Henkin 综合器组合做出了重要贡献。在至少有一种工具能够求解的 204 个基准测试中，Manthan3 在其中 42 个实例上实现了最短的综合时间。此外，对于 26 个实例，Manthan3 能够综合出 Henkin 函数，而其他最先进的函数综合引擎均无法完成综合。

2 预备知识

我们用小写字母表示命题变量，用大写字母表示变量集合。文字是一个变量或其否定，子句则被视为文字的析取。由子句合取表示的公式 φ 称为合取范式（CNF）。$Vars(\varphi)$表示出现在 φ中的变量集合。公式 φ 的一个满足赋值（σ）将$Vars(\varphi)$映射到 $\{0,1\}$ ，使得在 σ下， φ 的值为真。我们用 $\sigma \models \varphi$ 表示 φ的一个满足赋值 σ 。对于变量集合V，我们用 $\sigma[V]$ 表示 σ 在 V上的限制。如果 φ 在$Vars(\varphi)$的所有可能赋值下都为真，则 φ 称为重言式。

均匀采样器从公式的解空间中均匀随机地采集所需数量的满足赋值。我们用 UnsatCore 表示不可满足核，它是 φ 的一个子句子集，该子句集不存在满足赋值。对于一个合取范式公式，若其中一组子句被视为硬约束，其余子句被视为软约束，则最大可满足性求解器会尝试找到一个满足赋值，使其满足所有硬约束，并使满足的软约束数量最大化。

公式 ϕ若可表示为ϕ： $\forall x_1 \ldots x_n \exists^{H_1} y_1 \ldots \exists^{H_m} y_m \varphi(X, Y)$，其中 $X = \{x_1, \ldots, x_n\}$， $Y = \{y_1, \ldots, y_m\}$且 $H_i \subseteq X$表示 y_i的依赖集，也就是说，变量 y_i只能依赖于 H_i。每个 H_i称为 Henkin 依赖，每个量词 $\exists^{H_i}$ 称为 $Henkin$ 量词 [22]。

若存在函数 f_i： $\{0,1\}^{|H_i|} \mapsto \{0,1\}$，使得对于每个存在量化变量 y_i，将每个 y_i替换为其对应函数 f_i后得到的 $\varphi(X, f_1(H_1).....,\ f_m(H_m))$ 是一个重言式，则称 DQBF ϕ 为真。给定一个 DQBF ϕ，DQBF 可满足性问题就是判断给定的 DQBF 为真还是为假。

问题陈述：给定一个真 DQBF $\forall x_1 \ldots x_n \exists^{H_1} y_1 \ldots \exists^{H_m} y_m\ \varphi(x_1, \ldots x_n, y_1, \ldots, y_m)$，其中 $x_1, \ldots, x_n \in X$, $y_1, \ldots, y_m \in Y$, $H_i \subseteq X$，Henkin 综合问题是综合一个函数向量 $\boldsymbol{f}$: $\langle f_1, \ldots, f_m \rangle$，使得$\varphi(X, f_1(H_1) \ldots, f_m(H_m))$ 是一个重言式。

$\boldsymbol{f}$ 称为 Henkin函数向量，其中每个 f_i 都是一个 Henkin 函数。我们交替使用$\forall X \exists^{H_1} y_1 \ldots \exists^{H_m} y_m\ \varphi(x_1, \ldots, x_n, y_1, \ldots, y_m)$ 和 $\forall X \exists^{H} Y \varphi(X, Y)$。

Henkin 综合推广了 Skolem 综合，其中 $H_1 = \ldots = H_m = X$。在这种情况下，可以省略 H_i 的使用，直接将 ϕ 表示为$\forall X \exists Y \varphi(X, Y)$。在这种情况下， $\boldsymbol{f}$称为 Skolem 函数向量，使得$\forall X(\exists Y \varphi(X, Y) \leftrightarrow \varphi(X, \boldsymbol{f}))$。

3 相关工作

量化布尔公式求解领域已有大量研究， [26,25,32,33,37,38,39],近年来，DQBF 求解也受到了广泛关注[14,16,17,42,40,43,44]。Frohlich 等人提出了首个基于 DPLL 的 DQBF 可满足性求解方法 [13]。沿着相似的思路，Tentrup 和 Rabe 最近 [43]提出了将子句抽象用于 DQBF 求解的想法。Gitina 等人 [16]提出了一种基本的变量消去策略来求解依赖量化布尔公式实例：通过消去导致非线性依赖关系的一组变量，将依赖量化布尔公式实例转换为量化布尔公式。该策略随后通过若干优化得到了进一步改进，见 [17]。Frohlich 等人提出了类似的方法 [14] ，通过对每个子句进行局部全称展开，将依赖量化布尔公式实例转换为可满足性实例。

Wimmer 等人 [44] 提出了一种方法，可从基于变量消去的 DQBF 求解技术的 DQBF 求解器中获取 Henkin 函数 [17,14]。基于消去的 DQBF 求解器会对 DQBF 实例执行一系列消去量词的转换，以获得一个可满足实例；在此过程中，它们会得到一系列等可满足公式 φ_1， φ_2，……， φ_k，其中 φ_i 公式是由 φ_{i-1}转换得到的。他们证明，对于真 DQBF 实例，可以从 φ_i 中获得 φ_{i-1} 的 Henkin 函数。

最近，Reichl、Slivovsky 和 Szeider [40] 提出了一种不同的 DQBF 求解方法，即基于插值的定义提取。他们引入仲裁变量 ，用于表示存在变量在其依赖集的所有赋值下、且该变量未被唯一确定时的取值。随后，该方法会提取存在变量关于其依赖集和这些仲裁变量的定义。所提出的方法在设计上具有可认证性，并会返回 Henkin 函数。

DQBF 领域最早的研究聚焦于 2-QBF 这一特殊情形，涉及判定问题和函数综合问题。在 2-QBF 的背景下，函数综合问题称为 Skolem 综合。最早的 Skolem 综合研究仅限于给定的 2-QBF 公式为 True [7,23,27] 的情形；后续研究则扩展到一般的 2-QBF，因此提出了许多适用范围广泛的技术，例如基于 CEGAR 的方法 [3,4,12]，基于因子分解的技术 [5,29], 增量确定化 [37]。

最近，一种新的数据驱动方法已在 Manthan [18,19] 中实现，用于 Skolem 综合。Manthan 利用约束采样、机器学习和形式化方法方面的进展，高效地进行 Skolem 综合。Manthan 从 φ 的解中均匀采样。以这些样本为数据，它为每个存在量化变量学习一个候选 Skolem 函数向量，并将其表示为决策树分类器。随后，Manthan 验证所学的候选向量是否为 Skolem 向量。如果候选向量通过验证检查，Manthan 就会返回该 Skolem 向量。否则，Manthan 会利用反例执行 UnsatCore引导的修复，并依靠最大可满足性求解器，尽量减少每个反例中需要修复的候选项数量。

4 概述

本节概述 Manthan3 框架。尽管 Manthan3 与Manthan这一近期提出的Skolem函数综合引擎 [18][19], 在机器学习技术和 SAT/MaxSAT求解器的使用方面有诸多相似之处，但由于 Henkin 依赖关系带来的要求，这两种技术存在关键差异。值得指出的是，处理 Henkin 依赖并非易事；最能说明这一点的或许是：2-QBF 为 Σ_2^P-完全问题，而 DQBF 为 NEXPTIME 完全问题 [35]。

如图 1所示，Manthan3 首先利用约束采样方面的进展生成数据，然后使用这些数据，通过基于机器学习的方法学习一个候选向量 $\boldsymbol{f}$ 。接着，Manthan3 尝试验证候选向量 $\boldsymbol{f}$ 是否为 Henkin 函数向量。如果候选项通过形式化验证检查，Manthan3 就会将其作为有效的 Henkin 向量返回。否则，系统会修复候选向量以满足反例，并重复进行验证检查。请注意，在学习和修复候选项时，Manthan3 需要考虑 Henkin 依赖所施加的限制。

现在，我们将对 Manthan3 的不同组件作高层概述:

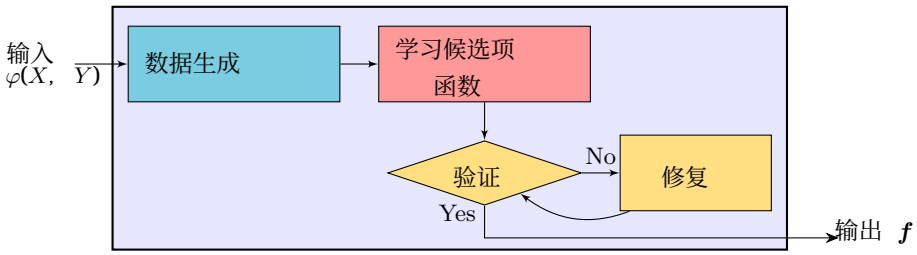


图 1： Manthan3 概览

数据生成： 首先，Manthan3 使用受约束采样器 [20,21]，从规约 φ 的解空间中均匀随机地采样满足赋值。采样得到的满足赋值被视为数据，用于下一阶段训练学习算法，以学习候选函数。

候选学习： Manthan3 为每个存在量化变量 y_i 学习一个二叉决策树分类器，以学习与其对应的候选函数 f_i 。生成样本中 y_i 的赋值被视为标签，而对应亨金依赖 H_i 的赋值则被视为用于学习决策树的特征集。与 y_i 对应的亨金函数 f_i ，是通过对学习所得决策树中从根节点到标签为**1** 的叶节点的所有路径上的标签取析取而计算得出的。

由于亨金依赖， y_i 的特征集必须仅限于 H_i。不过，为了学习出较好的决策树，我们可以将所有满足 $H_j \subset H_i$的 y_j 都纳入特征集。函数 f_j 可以直接在 f_i 中展开，使得 f_i 仅用 H_i来表示。对于 $H_j = H_i$的情况，只要不会造成循环依赖，就允许这样使用 Y 变量；也就是说，如果学习得到的候选项 f_i中出现 y_j ，则不允许将y_i 作为学习候选项 f_j的特征。如果 y_j 出现在 f_i中，我们就说 y_i 依赖于 y_j，记作 $y_i \prec_d y_j$。随着候选函数的学习，Manthan3 会动态发现此类 Y 变量之间必要的变量排序约束。

若函数向量 $\boldsymbol{f}$ 中的 y_j 出现在 f_i 中，且 y_i 未出现在 f_j 中，则该向量有效。若 $\boldsymbol{f}$ 是有效的函数向量，则存在一个定义在 $\{y_1, \ldots, y_m\}$ 上的偏序 $\prec_d$。获得候选向量后，Manthan3 根据在 候选学习 中学到的关于 Y 个变量的部分依赖关系，得到一个有效的线性扩张全序，记作 Order。

验证： 学习到的候选向量未必总是有效的 Henkin 向量。因此，必须验证候选函数。只有当 $\boldsymbol{f}$ 是 Henkin 函数向量时， $\varphi(X, f_1(H_1)..... f_m(H_m))$ 才是重言式。Manthan3 首先对公式 $E(X, Y') = \neg\varphi(X, Y') \wedge (Y' \leftrightarrow \boldsymbol{f})$ 发起 SAT 预言机查询。

如果公式 $E(X,Y')$ 不可满足，Manthan3 将函数向量 $\boldsymbol{f}$ 作为 Henkin 函数向量返回。如果公式 $E(X,Y')$ 可满足，且 δ 是 $E(X,Y')$ 的一个满足赋值，Manthan3 需要确定 $\varphi(X,Y)$ 是否存在一个扩展 X 赋值的命题模型。因此，Manthan3 会对公式 $\varphi(X,Y) \wedge (X \leftrightarrow \delta[X])$ 再进行一次可满足性检查。如果可满足性检查返回不可满足，则相应的 $DQBF$ 公式为 False，且不存在 Henkin 函数向量；因此 Manthan3 会终止。此外，如果 $\varphi(X,Y) \wedge (X \leftrightarrow \delta[X])$ 可满足，且 π 是一个满足赋值，我们就需要修复候选函数向量。注意， $\pi[X]$ 与 $\delta[X]$ 相同， $\pi[Y]$ 是 X 的一个可能扩展赋值，而 $\delta[Y']$ 表示候选函数向量在 $\delta[X]$ 下的输出。现在，我们得到一个反例 σ ，即 $\pi[X] + \pi[Y] + \delta[Y']$。

候选修复： 我们采用反例驱动修复方法来修复候选函数。当 Manthan3 尝试修复反例 σ 时，首先需要从 f_1 到 f_m 个候选项中找出要修复的候选项。Manthan3 借助最大可满足性求解器找出待修复的候选项，并以 $\varphi(X, Y) \wedge (X \leftrightarrow \sigma[X])$ 作为硬约束、以 $(Y \leftrightarrow \sigma[Y'])$ 作为软约束，发起最大可满足性查询。如果最大可满足性求解器返回的解使相应的软约束 $y_i \leftrightarrow \sigma[y_i']$ 不成立，它就会选择函数 f_i 进行修复。确定待修复的候选项后，Manthan3 会利用从不可行性证明中得到的不可满足核，这些证明揭示了候选项未能满足规约的原因，并据此构造修复方案。

现在假设 Manthan3 选择了与变量 y_i 对应的 f_i 作为潜在候选项。Manthan3 构造另一个公式 $G_i(X, Y)$（公式

1) 以找到修复：

$$G_i(X,Y) : \varphi(X,Y) \wedge (H_i \cup \hat{Y} \leftrightarrow \sigma[H_i \cup \hat{Y}]) \wedge (y_i \leftrightarrow \sigma[y_i'])$$
$$\text{where } \hat{Y} \subseteq Y \text{ such that } \forall y_j \in \hat{Y} : H_j \subseteq H_i$$
$$\text{and } \{\mathsf{Order}[index(y_j)] > \mathsf{Order}[index(y_i)]\} \quad (1)$$

简单来说，为了确定是否需要修复 f_i ，我们将规约 $\varphi(X,Y)$ 与一组单子句合取；这些单子句将 y_i 的赋值设为 f_i 当前的输出，并按照反例设置所有依赖关系的赋值。下面说明构造 $G_i(X,Y)$ 的直观思路。公式 $G_i(X,Y)$ 用于回答以下问题：是否可能将 y_i 设为 f_i 的输出（给定其亨金依赖的赋值）？。

上述问题的答案取决于 $G_i(X, Y)$ 是不可满足还是可满足。 $G_i(X, Y)$ 不可满足，表明无法将 y_i 设置为 f_i 的输出，而 $G_i(X, Y)$ 的不可满足核则说明了原因。因此，Manthan3 使用 $G_i(X, Y)$ 的不可满足核来修复候选函数 f_i。具体而言，Manthan3 使用 $G_i(X, Y)$ 的不可满足核中与单子句对应的所有变量来构造修复公式 β；并根据反例 σ 中 y_i' 的赋值，使用 β 来强化或弱化候选项 f_i ，使其满足该反例。

另一方面，如果 $G_i(X, Y)$ 可满足，Manthan3 就会尝试寻找其他候选函数来进行修复。 $G_i(X, Y)$ 可满足，表明在当前亨金依赖的赋值下， y_i 可以取候选函数 f_i 输出的值；然而，要修复反例 σ，我们需要修复另一个候选函数。为此，设 ρ 是 $G_i(X, Y)$ 的一个满足赋值，那么所有满足 $\rho[y_j]$ 与 $\sigma[y_j']$ 不同的 y_j 变量都会被加入待修复潜在候选项的队列中。

修复循环会持续进行，直到 $E(X, Y')$ 不可满足，或 $\varphi(X, Y) \wedge (X \leftrightarrow \delta[X])$ 不可满足为止，其中 δ 是 $E(X, Y')$ 的一个满足赋值。如果 $E(X, Y')$ 不可满足，则我们得到一个 Henkin 函数向量 $\boldsymbol{f}$；如果 $\varphi(X, Y) \wedge (X \leftrightarrow \delta[X])$ 不可满足，则给定的 DQBF 实例为假，并且不存在 Henkin 函数向量。

5 算法细节

Manthan3（算法 1）以一个 DQBF 实例 $\forall X \exists^{H_1} y_1 \ldots \exists^{H_m} y_m \varphi(X, Y)$ 作为输入，并输出一个 Henkin 函数向量 $\boldsymbol{f} := \langle f_1, \ldots, f_m \rangle$。

算法1 假设可以访问以下子程序：

1. GetSamples：该算法以规约为输入，并调用预言机生成规约样本 Σ 。 Σ 中的每个样本都是规约的一个满足赋值。

算法1 Manthan3($\forall X \exists^H Y.\varphi(X,Y)$)

1: $\Sigma \leftarrow$ GetSamples($\varphi(X,\ Y)$)
2: $D \leftarrow \{d_1 = \emptyset \ldots, d_{|Y|} = \emptyset\}$
3: **对** $\langle H_i, H_j \rangle$ **执行**
4: **如果** $H_j \subset H_i$ **，则**
5: $d_j \leftarrow d_j \cup y_i$
6: **当** $y_i \in Y$ **时执行**
7: $f_i,\ D \leftarrow$CandidateHkF($\Sigma,\ \varphi(X,\ Y)\ y_i,\ D$)
8: Order $\leftarrow$ FindOrder(D)
9: **重复**
10: $E(X,\ Y') \leftarrow \neg\varphi(X,\ Y') \wedge (Y' \leftrightarrow \boldsymbol{f})$
11: $ret,\ \delta \leftarrow$ CheckSat($E(X,\ Y')$)
12: **如果** ret =可满足，**则**
13: $res,\ \pi \leftarrow$ CheckSat($\varphi(X,\ Y) \wedge (X \leftrightarrow \delta[X])$)
14: **如果** res =不可满足，**则**
15: **返回** $\forall X \exists^H Y.\varphi(X,\ Y)$ 为 False。
16: $\sigma \leftarrow \pi[X] + \pi[Y] + \delta[Y']$ {σ 是一个反例}
17: $\boldsymbol{f} \leftarrow$ RepairHkF($\varphi(X,\ Y)\ \boldsymbol{f},\ \sigma,$ Order)
18: **直到** ret =不可满足
19: $\boldsymbol{f} \leftarrow$ 替换($\varphi(X,Y)\ \boldsymbol{f}$, Order)
20:**返回** $\boldsymbol{f}$

2. CandidateHkF：该子程序生成与某个存在变量相对应的候选函数。它以规约 φ、生成样本 Σ、我们希望为其学习候选函数的存在变量 y_i，以及用于跟踪 Y 变量之间依赖关系的向量 D 作为输入。CandidateHkF 返回与 y_i相对应的候选函数 f_i，并更新 D 中关于 y_i的依赖关系。算法 2 中详细介绍了 CandidateHkF 过程。3. FindOrder：它以集合 D 为输入，该集合由若干 d_i 组成，其中每个 d_i 都是 Y变量的列表，这些变量可以依赖于 y_i。FindOrder 根据 D中的部分依赖关系，求得一个有效的线性扩张 Order。4. CheckSat：它以规约为输入，并调用 SAT 预言机对该规约进行可满足性检查。它返回可满足性检查的结果，即可满足或不可满足。若结果为可满足，它还会返回该规约的一个满足赋值。5. RepairHkF：该子程序修复当前的候选函数向量，以消除反例。它以规约、候选函数向量、一个反例以及 Order（ Y变量之间依赖关系的线性扩张）作为输入，并返回修复后的候选函数向量。算法 3 详细介绍了 RepairHkF 子程序。

算法1 首先通过调用 GetSamples 子程序 Σ 来（位于第 1 行）生成样本。接下来，算法 1 初始化集合 D （第 2 行），该集合由 d_i组成，其中 d_i 表示依赖于 Y 的变量集合 y_i。第 3-5 行根据其中的子集关系引入变量排序约束

每个 $\langle H_i, H_j \rangle$ 对，也就是说，如果 $H_j \subset H_i$，那么 y_i 可以依赖于 y_j。第7 行对每个 CandidateHkF y_i 变量调用子程序，以学习候选函数f_i。接下来，在第8行，Manthan3 调用 FindOrder 计算Order，即在满足 Y 中所有排序约束的变量之间进行拓扑排序 D。

在第 11 行，CheckSat 检查第 10 行所述公式 $E(X, Y')$ 的可满足性。如果 $E(X, Y')$ 可满足，那么第 13 行的 Manthan3 会再次检查可满足性，以确保命题模型 X 可以扩展为 Y。如果第 13 行的 CheckSat 返回不可满足，那么算法1 会在第 15 行终止，因为不存在 Henkin函数向量；否则，Manthan3 会得到一个需要修复的反例 σ。候选向量 $\boldsymbol{f}$ 根据反例 σ 进入修复迭代（第 17 行），也就是说，子程序 RepairHkF 会修复当前函数向量 $\boldsymbol{f}$，从而修复 σ。只有当 $E(X, Y')$ 不可满足时，Manthan3 才会返回函数向量 $\boldsymbol{f}$。

算法 2 CandidateHkF$(\Sigma, \varphi(X, Y)\ y_i, D)$

```
1:  featset ← H_i
2:  对于 y_j ∈ Y 执行
3:    如果 (H_j ⊆ H_i) ∧ (y_j ∉ (d_i ∪ y_i)) 那么
4:      featset ← featset ∪ y_j
5:  feat, lbl ← Σ_↓featset, Σ_↓y_i
6:  t ←CreateDecisionTree(feat, lbl)
7:  对 n ∈ 叶节点(t) 执行
8:    如果 标签(n) = 1 则
9:      π ← 路径(t, root, n) {从根节点到节点 n 的路径，位于树 t}中
10:     f_i ← f_i ∨ π
11: 对 y_k ∈ f_i 执行
12:   d_k ← d_k ∪ y_i ∪ d_i
13: 返回 f_i, D
```

现在我们详细讨论子程序 CandidateHkF 和 RepairHkF。

算法2 展示了 CandidateHkF子程序。CandidateHkF假设可以访问 CreateDecisionTree，该程序用于构建决策树，t 输入是在一组特征 $featset$上的带标签数据。该程序使用 ID3 算法 [36]，我们使用基尼指数 [36] 作为不纯度度量。

在算法 2 中，第 1 行包含 y_i 在依赖集 H_i 中的特征集 featset。此外，第 3 行将特征扩展为包含所有依赖集 H_j 是 H_i 子集且 y_j 不依赖于 y_i 的变量 y_j，以便决策树也能在此类 y_j 上进行学习。第 5 行从样本 Σ 中选取特征集和值的赋值，并训练一棵决策树。接着，第 7 至 10 行通过对树中所有通向类别标签 1 的路径进行析取，构造一个逻辑公式来表示这棵决策树。在第 12 行，为出现在针对 y_i 的决策树 t 中作为节点的变量 y_k 更新集合 d_k。

算法 3 RepairHkF($\varphi(X,\ Y)$ $\boldsymbol{f}$, σ, Order)

1: $H \leftarrow \varphi(X,\ Y) \wedge (X \leftrightarrow \sigma[X])$; $S \leftarrow (Y \leftrightarrow \sigma[Y'])$
2: $Ind \leftarrow$ FindCandi(H, S)
3: **对** $y_k \in Ind$ **执行**
4: $\hat{Y} \leftarrow \emptyset$
5: **对** $y_j \in Y$ **执行**
6: **如果** $H_j \subseteq H_k \wedge$Order$[index(y_j)] >$ Order$[index(y_k)]$ **那么**
7: $\hat{Y} \leftarrow \hat{Y} \cup y_j$
8: $G_k \leftarrow (y_k \leftrightarrow \sigma[y'_k]) \wedge \varphi(X,Y) \wedge (H_k \leftrightarrow \sigma[H_k]) \wedge (\ \hat{Y} \leftrightarrow \sigma[\hat{Y}])$
9: $ret,\ \rho \leftarrow$ CheckSat(G_k)
10: **如果** $ret = UNSAT$ **那么**
11: $C \leftarrow$FindCore(G_k)
12: $\beta \leftarrow \bigwedge_{l \in C} ite((\sigma[l] = 1)\ l,\ \neg l)$
13: $f_k \leftarrow ite((\sigma[y'_k] = 1)\ f_k \wedge \neg\beta,\ f_k \vee \beta)$
14: **else**
15: **针对** $y_t \in Y \setminus \hat{Y}$ **do**
16: **if** $\rho[y_t] \neq \sigma[y'_t]$ **然后**
17: $Ind \leftarrow Ind.Append(y_t)$
18: $\sigma[y_k] \leftarrow \sigma[y'_k]$
19: **返回** $\boldsymbol{f}$

算法 3表示 RepairHkF 子程序。RepairHkF 假设可以访问以下子程序：

1. FindCandi：它以硬约束和软约束作为输入。它调用 MaxSAT 求解器，对包含硬约束和软约束的规约进行求解，并返回一组变量，这些变量对应于 MaxSAT 求解器为满足规约而舍弃的软约束。2. FindCore：它以一个不可满足公式作为输入，并返回该公式的不可满足核（UnsatCore）。

算法 3 首先尝试使用 FindCandi 查找要修复的潜在候选项。在第 2 行，FindCandi 子程序本质上是调用最大可满足性求解器，将$\varphi(X,Y) \wedge (X \leftrightarrow \sigma[X])$作为硬约束，并将$(Y \leftrightarrow \sigma[Y])$作为软约束，以查找要修复的潜在候选项；它返回一个列表 (Ind)，其中包含 Y 变量，使得与出现在 (Ind)ˆ 中的每个变量对应的候选项都是要修复的潜在候选项。对于每个 $y_k \in$ Ind，第 6 行计算 Y，它是一个 y_j 变量集合，这些变量在 Order 中出现在 y_k 之后，并且对应的H_j 是 H_k的子集。

接下来，算法 3 在第 9 行检查 G_k 公式的可满足性。如果 G_k 不可满足，第 11 行会尝试使用子程序 FindCore 找出 G_k 的不可满足核，第 12 行则利用不可满足核中单子句所对应的文字构造修复公式 β。根据 $\sigma[y'_k]$的值，第 13 行会使用 β 来增强或削弱 f_k 。如果 G_k 可满足且 $\rho \models G_k$，第 15 至 18 行会寻找其他可用于修复的潜在候选项，并将所有满足 $\rho[y_t]$ 与 $\sigma[y'_t]$不相同的 y_t 变量添加到列表 Ind 中。

请注意，在第 8 行，我们在 $G_i(X,Y)$ 中添加约束 $\hat{}\ Y \leftrightarrow \sigma[\hat{}Y]$，其中 $\hat{}Y$是一个由 Y 变量组成的 $\hat{}$ 集合，满足对于 Y 中的所有 y_j， $H_j \subseteq H_i$。固定这类 y_j变量的赋值有助于 Manthan3 为候选项 f_i 合成更好的修复。请看下面的例子。令 $\forall X \exists^{H_1} \exists^{H_2} \varphi(X,Y)$，其中 $\varphi(X,Y) : (y_1 \leftrightarrow x_1 \oplus y_2)$,$H_1 = \{x_1\}$ 和 $H_2 = \{x_1\}$。假设我们需要修复候选项f_1，且 $G_1(X,Y) = (y_1 \leftrightarrow \sigma[y_1']) \wedge \varphi(X,Y) \wedge (x_1 \leftrightarrow \sigma[x_1])$。由于 $G_1(X,Y)$ 不包含导致该反例的 y_2 当前值，因此无法推动 f_1 朝确保 $y_1 \leftrightarrow x_1 \oplus y_2$的方向变化。事实上，在这种情况下，修复公式 β 将为空，因而无法完成修复。

让我们通过一个示例详细讨论 Manthan3 的工作方式。现在我们

x_1	x_2	x_3	y_1	y_2	y_3
0	0	0	1	1	0
0	0	1	1	1	1
1	1	0	0	0	1

图 2： φ的样本（X，Y）

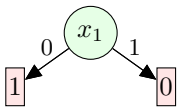


图3： y_1的决策树

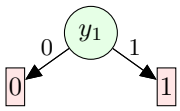


图4： y_2的决策树

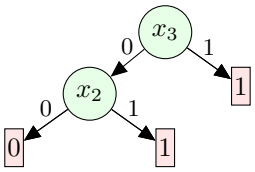


图5： y_3的决策树

通过一个示例说明算法1 。

示例*1*。令 $X = \{x_1, x_2, x_3\}, Y = \{y_1, y_2, y_3\}$ 属于 $\forall X \exists^{H_1} y_1\ \exists^{H_2} y_2 \exists^{H_3} y_3 \varphi(X,\ Y)$，其中 φ $(X,\ Y)$ 是 $(x_1 \vee y_1) \wedge (y_2 \leftrightarrow (y_1 \vee \neg x_2)) \wedge (y_3 \leftrightarrow (x_2 \vee x_3))$，且 $H_1 = \{x_1\}$, $H_2 = \{x_1, x_2\}$，并且 $H_3 = \{x_2, x_3\}$。

数据生成：Manthan3 通过自适应采样生成训练数据。假设采样器生成的数据如图 2所示。

候选学习：由于 $H_1 \subset H_2$，Manthan3 添加了一项依赖约束，规定 y_1 不能依赖于 y_2。Manthan3 现在尝试学习候选项，并对每个CandidateHkF 调用 y_i。由于 $H_3 \not\subseteq H_1$， y_1 不能依赖于y_3，因此 y_1 的特征集 H_1只包含 $\{x_1\}$ 。决策树的构建使用 $\{y_1\}$ 的样本作为特征，并使用 f_1 的样本作为标签。候选函数 f_1 通过对所有以标签为 1 的叶节点结尾的路径取析取来构造。如图3所示， f_1 是 $\neg x_1$。

作为 $H_1 \subset H_2$， y_2 的特征集包含 H_2 和 y_1，但不能将 y_3 作为 $H_3 \not\subseteq H_2$。因此，决策树的构建使用以下样本：

$\{x_1, x_2, y_1\}$ 作为特征，以及 $\{y_2\}$ 作为标签的样本。候选函数 f_1通过对所有以标签 1 的叶节点结尾的路径取析取得到：如图 4 所示， f_2 综合为 y_1。类似地，对于 y_3，特征集为 H_3，并按照图 5 所示构建决策树，使用 x_2、x_3作为特征，并使用 y_3作为标签的样本。我们得到 $f_3 := x_3 \vee (\neg x_3 \wedge x_2)$。在 CandidateHkF 结束时，我们得到 $f_1 := \neg x_1$，$f_2 := y_1$，$f_3 := x_3 \vee (\neg x_3 \wedge x_2)$。假设 FindOrder 返回的全序为 Order $= \{y_3, y_2, y_1\}$。

验证：我们构造 $E(X, Y') = \neg\varphi(X, Y') \wedge (Y' \leftrightarrow \boldsymbol{f})$，结果发现它是可满足的，并令 $\delta = \langle x_1 \leftrightarrow 1, x_2 \leftrightarrow 0, x_3 \leftrightarrow 0, y_1' \leftrightarrow 0, y_2' \leftrightarrow 0, y_3' \leftrightarrow 0\rangle$ 为 $E(X, Y')$ 的一个满足赋值。接下来，Manthan3 检查该赋值是否可以扩展到 Y，以满足规约。它检查 $\varphi(X, Y) \wedge (x_1 \leftrightarrow \delta[x_1]) \wedge (x_2 \leftrightarrow \delta[x_2]) \wedge (x_3 \leftrightarrow \delta[x_3])$ 的可满足性，结果发现它是可满足的；令 π 为该满足赋值， $\pi = \langle x_1 \leftrightarrow 1, x_2 \leftrightarrow 0, x_3 \leftrightarrow 0, y_1 \leftrightarrow 1, y_2 \leftrightarrow 1$、 $y_3 \leftrightarrow 0\rangle$。令 $\sigma = \langle x_1 \leftrightarrow 1, x_2 \leftrightarrow 0, x_3 \leftrightarrow 0, y_1 \leftrightarrow 1, y_2 \leftrightarrow 1, y_3 \leftrightarrow 0, y_1' \leftrightarrow 0$、 $y_2' \leftrightarrow 0$、 $y_3' \leftrightarrow 0\rangle$ 为一个待修复的反例。

候选修复：为了找出待修复的候选项，FindCandi 中将 $\varphi(X, Y) \wedge (x_1 \leftrightarrow \sigma[x_1]) \wedge (x_2 \leftrightarrow \sigma[x_2]) \wedge (x_3 \leftrightarrow \sigma[x_3])$ 作为硬约束，并将 $(y_1 \leftrightarrow \sigma[y_1']) \wedge (y_2 \leftrightarrow \sigma[y_2']) \wedge (y_3 \leftrightarrow \sigma[y_3'])$ 作为软约束，调用最大可满足性求解器。设 FindCandi 返回 $ind = \{y_2\}$。

修复综合从对 $G_2 = \varphi(X, Y) \wedge (x_1 \leftrightarrow \sigma[x_1]) \wedge (x_2 \leftrightarrow \sigma[x_2]) \wedge (y_1 \leftrightarrow \sigma[y_1']) \wedge (y_2 \leftrightarrow \sigma[y_2'])$ 进行可满足性检查开始，其中 f_2。注意，这里我们可以用 y_1 将 G_2 约束为 $H_1 \subset H_2$。该公式不可满足，因此 Manthan3 调用 FindCore，后者返回变量 $\neg x_2$，因为约束 $(x_2 \leftrightarrow \sigma[x_2])$ 和 $(y_2 \leftrightarrow \sigma[y_2'])$ 在 G_2 中无法同时满足。由于对于赋值 σ，候选项 f_2 的输出必须从 0 变为 1，因此通过与 $\neg x_2$ 作析取来修复 f_2，从而得到 $f_2 := y_1 \vee \neg x_2$ 作为新的候选项。

更新后的候选向量 $\boldsymbol{f}$：$\langle f_1 := \neg x_1,\ \ f_2 := y_1 \vee \neg x_2,\ \ f_3 := x_3 \vee (\neg x_3 \wedge x_2)\ \rangle$ 通过了验证检查，也就是说，公式 $E(X, Y')$ 不可满足。因此，Manthan3 返回 $\boldsymbol{f}$ 作为一个 Henkin函数向量。

根据 Henkin 函数的定义，我们知道以下引理成立：

引理 1. *$\boldsymbol{f}$ 是 Henkin函数向量，当且仅当 $\neg\varphi(X, Y) \wedge (Y \leftrightarrow \boldsymbol{f})$ 不可满足。*

Manthan3 仅在 $E(X, Y') : \neg\varphi(X, Y') \wedge (Y' \leftrightarrow \boldsymbol{f})$ 不可满足时才返回函数向量，并且按照构造，每个函数 f_i 都遵循亨金依赖。因此，Manthan3 是可靠的，且返回的函数向量是 Henkin函数向量。

局限性： 对于某些实例，Manthan3 可能无法修复候选向量，因此并非完备。其局限在于，公式 $G(X, Y)$（公式 1）无法识别亨金依赖。

让我们考虑一个例子， ϕ： $\forall X \exists^{H_1} y_1 \exists^{H_2} y_2\ \varphi(X, Y)$，其中 $X = \{x_1, x_2, x_3\}$，$Y = \{y_1, y_2\}$，$\varphi(X, Y) := \neg(y_1 \oplus y_2)$，$H_1 = \{x_1, x_2\}$ 和 $H_2 = \{x_2, x_3\}$。注意， ϕ 为真，Henkin 函数为 $\boldsymbol{f} := \langle f_1(x_1, x_2) := x_2,\ f_2(x_2, x_3) :$

$x_2\rangle$。假设 Manthan3 学到的候选项为 $\boldsymbol{f} := \langle f_1(x_1,\ x_2)\colon x_2,\ \ f_2(x_2,\ x_3)\colon \neg x_2\rangle$。学到的候选项不是 Henkin 函数，因为 $E(X,\ Y')$ 是可满足的。设要修复的反例为 σ 是 $\langle x_1 \leftrightarrow 0,\ x_2 \leftrightarrow 0, x_3 \leftrightarrow 0,\ y_1 \leftrightarrow 0,\ y_2 \leftrightarrow 0,\ y_1' \leftrightarrow 0,\ y_2' \leftrightarrow 1\rangle$。

设要修复的候选项为 y_2，相应的 G_2 公式为 $G_2 := \varphi(X, Y) \wedge (x_2 \leftrightarrow 0) \wedge (x_3 \leftrightarrow 0) \wedge (y_2 \leftrightarrow 1)$。由于 $H_1 \not\subseteq H_2$，公式 G_2 不允许对 y_1施加约束。G_2 结果为可满足，这表明我们应尝试修复 y_1，而不是 y_2；但由于 y_1 也不允许依赖于 y_2，公式G_1 也将是可满足的。因此，Manthan3 无法修复候选项 $\boldsymbol{f}$ 来消除反例 σ。在这种情况下，Manthan3 无法综合 Henkin 函数。因此，Manthan3 并不完备。

6 实验结果

我们实现了 Manthan3[3]，使用 Python 编写，并采用 Open-WBO [34]处理最大可满足性查询，使用 PicoSAT [8] 查找不可满足核，使用 ABC [31] 表示和操作布尔函数，使用 CMSGen 生成所需的样本 [20],UNIQUE [41] 提取唯一确定变量的定义，并使用 Scikit- Learn [2] 学习决策树。

实例： 我们对563个实例进行了广泛比较，这些实例由QBFEval18、19和20 [1],中DQBF赛道的实例集合并而成，涵盖等价性检查问题、控制器综合问题,以及命题可满足性问题的简洁DQBF表示。

测试硬件： 我们的所有实验均在一台高性能计算机集群上进行，每个节点配备一颗具有24个核心和96GB随机存取存储器的E5-2690 v3 CPU，并将每个核心的内存限制设为4GB。所有工具均在单核心上运行，每个基准测试的超时时间为7200秒。

对比工具： 我们将其与先前的最先进技术HQS2 [17] 和Pedant [40]进行了比较。需要注意的是，我们将Manthan3与能够为真 DQBF综合Henkin 函数的工具进行了比较；其余所有DQBF求解器（包括DepQBF [32], DQBDD [42]）都无法综合此类函数。DQBF预处理器HQSpre [45] 由HQS2隐式调用。我们发现,使用预处理器HQSPre时，Pedant的性能会下降；因此，我们采用Pedant未进行预处理时的结果。Manthan3在未使用HQSpre的情况下运行。

评估目标： 众所周知，在 NP 难问题中，不同技术对不同类别实例的适用情况各不相同。实际应用中通常采用组合方法 [11,24,46]。因此，实践中通常关注新技术对现有最先进工具组合的影响。为评估我们的算法对当前算法无法处理的实例所产生的影响，我们聚焦于**虚拟最佳综合器** (VBS)，即由目前已知算法中表现最佳者组成的组合。如果其中至少有一种工具能够

[3] Manthan3 可在以下网址获取：https://github.com/meelgroup/manthan

对给定实例综合出 Henkin 函数，则认为 VBS 能够综合出 Henkin 函数；也就是说，VBS 至少与组合中的每种工具一样强大。VBS 为给定实例综合 Henkin 函数所需的时间，是任一工具为该实例综合出函数所需时间的最小值。

结果：图 6展示了 HQS2 和 Pedant 的 VBS 与 HQS2、Pedant 和 Manthan3 的 VBS 之间的仙人掌图。我们观察到，采用 Manthan3 的 VBS 为 204 个实例合成了函数，而不采用 Manthan3 的 VBS 仅为 178 个实例合成了函数；也就是说，加入 Manthan3 后，VBS 多处理了 26 个实例。在 563 个实例中，至少有一种工具为 204 个实例合成了 Henkin 函数。Manthan3 在 42 个实例上实现了最短的合成时间，其中包括其他工具均无法合成 Henkin 函数的 26 个实例。

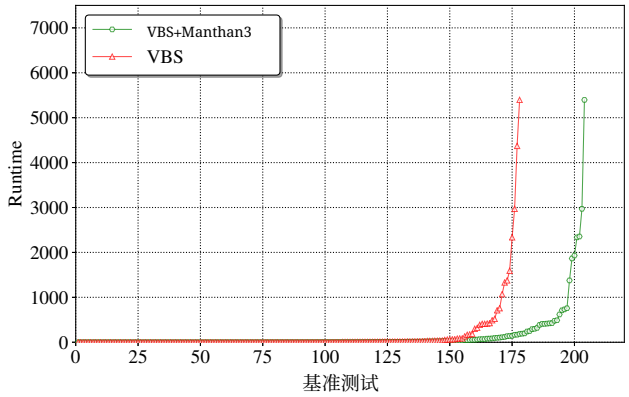


图 6：采用/不采用 Manthan3 时虚拟最佳 Henkin 函数合成器的表现。图中的 VBS 表示 HQS2 和 Pedant 的 VBS。点 $\langle x, y\rangle$ 表示某工具在不超过 y 秒的时间内，为总计 563 个实例中的 x 个实例合成了 Henkin 函数向量。

图 7 表明 Manthan3 的性能与现有工具正交。此外，如图 7中的绿色区域所示，对于 47 个实例，Manthan3 合成 Henkin 函数所用的时间比结合 HQS2 和 Pedant 的 VBS 多不超过 10 秒。

图 9 （以及图 8）分别展示了 Manthan3 与 HQS2（以及 Pedant）的散点图。对成功合成函数的实例进行分布分析可知，这三种工具彼此不可比较。有许多实例只有其中一种工具能够成功，而其他工具均告失败。

总共有 148、138 和 116 个实例，HQS2、Pedant 和 Manthan3 分别能够为这些实例合成 Henkin 函数。此外，有 40 个实例中，Manthan3 能够合成 Henkin 函数，而 HQS2

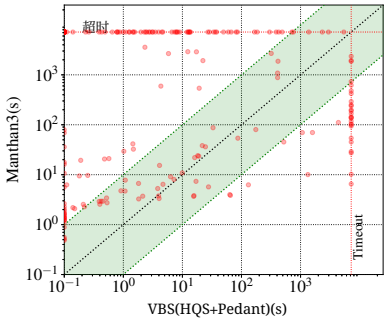


图 7： Manthan3 与 VBS(HQS2+Pedant) 的对比。

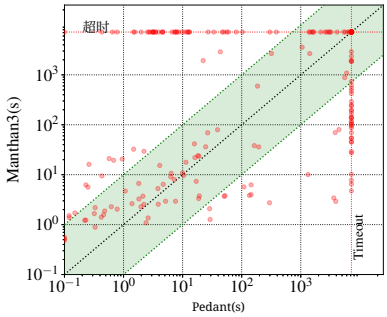


图 8： Manthan3 与 Pedant 的对比

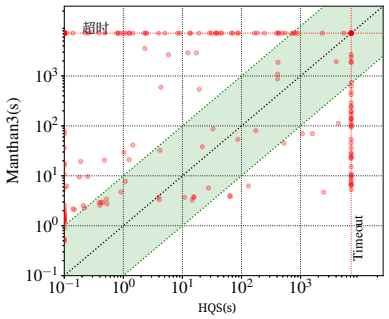


图 9： Manthan3 与 HQS2 的对比。

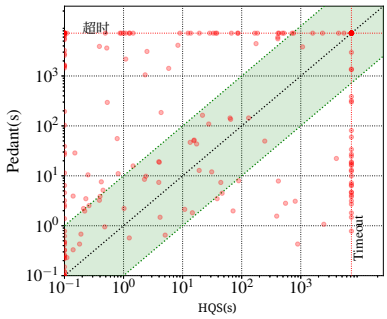


图 10： Pedant 与 HQS2。

点 $\langle x, y\rangle$ 表示，在 $\langle x\rangle$ 轴上的综合器耗时 x 秒，而在 $\langle y\rangle$ 轴上的综合器耗时 y 秒，为一个实例综合 Henkin 函数。

不能。类似地，有 37 个实例中，Pedant 未能合成 Henkin 函数，而 Manthan3 成功合成了。Manthan3 总共有 88 个实例未能合成函数；不过，Pedant 或 HQS2 能够为这些实例合成 Henkin 函数。由于 Manthan3 不完备，在这 88 个实例中，它无法处理其中 49 个，其余实例则超时。

图 10 表明，即使在现有工具 Pedant 和 HQS2 之间，也不存在最好的工具。尽管这两种工具都能为几乎相同数量的实例综合函数，但这些实例属于不同的类别。

结果表明，不同方法适用于不同类别的实例，而 Manthan3 在 Henkin 综合方面实现了突破，能够处理那些最先进的工具都无法综合出 Henkin 函数的实例。

7 结论

Henkin 综合有着广泛的应用，包括电路修复、部分等价性检查和控制器综合。在这项工作中，我们提出了 Henkin 综合器 Manthan3，它建立在机器学习和自动推理领域进展的基础之上。Manthan3 与现有的 Henkin 函数综合方法正交，这表明 Manthan3 所采用的基于机器学习的算法，从根本上不同于当前 Henkin 综合器所使用的算法。我们希望进一步了解这些差异。

致谢。本研究部分由新加坡国家研究基金会通过其卓越研究与技术企业园区（CREATE）计划、NRF 奖学金计划提供支持 [NRF-NRFFAI1-2019-0004]，新加坡教育部第二级资助项目 [MOE-T2EP20121-0011]，以及新加坡教育部第一级资助项目 [R-252-000-B59-114]。计算工作使用了新加坡国家超级计算中心的计算资源 https://www.nscc.sg。

参考资料

1. QBF 求解器评测门户 2020 (2020), http://www.qbflib.org/qbfeval20.php 2. sklearn.tree.decisiontreeclassifier (2021), https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html 3. Akshay, S., Arora, J., Chakraborty, S., Krishna, S., Raghunathan, D., Shah, S.: 布尔函数综合的知识编译。载于：FMCAD 论文集 (2019) 4. Akshay, S., Chakraborty, S., Goel, S., Kulal, S., Shah, S.: 布尔函数综合的难点是什么？载于：CAV 论文集 (2018) 5. Akshay, S., Chakraborty, S., John, A.K., Shah, S.: 迈向并行布尔函数综合。载于：TACAS 论文集 (2017) 6. Balabanov, V., Chiang, H.J.K., Jiang, J.H.R.: Henkin 量词与布尔公式：DQBF 的可认证视角。理论计算机科学论文集 (2014) 7. Balabanov, V., Jiang, J.H.R.: QBF 求值及其应用中的归结证明与 Skolem 函数。载于：CAV 论文集 (2011) 8. Biere, A.: PicoSAT 要点。JSAT 论文集 (2008) 9. Bloem, R., K¨onighofer, R., Seidl, M.: 基于 SAT 的安全规范综合方法。载于：VMCAI 论文集 (2014) 10. Chatterjee, K., Henzinger, T.A., Otop, J., Pavlogiannis, A.: LTL 片段的分布式综合。载于：FMCAD 论文集 (2013) 11. Dudek, J.M., Phan, V.H.N., Vardi, M.Y.: ProCount：采用分级投影连接树的加权投影模型计数。载于：SAT 论文集 (2021) 12. Fried, D., Tabajara, L.M., Vardi, M.Y.: 基于 BDD 的布尔函数综合。载于：CAV 论文集 (2016) 13. Fr¨ohlich, A., Kov´asznai, G., Biere, A.: 用于求解 DQBF 的 DPLL 算法。POS 论文集 (2012)

14. Fr¨ohlich, A., Kov´asznai, G., Biere, A., Veith, H.: idq：基于实例化的 DQBF 求解。载于：SAT 论文集 (2014) 15. Garey, M.R.: NP 完全性理论指南。《计算机与不可解问题》(1979) 16. Gitina, K., Reimer, S., Sauer, M., Wimmer, R., Scholl, C., Becker, B.: 使用依赖量化布尔公式对部分设计进行等价性检查。载于：ICCD 论文集 (2013) 17. Gitina, K., Wimmer, R., Reimer, S., Sauer, M., Scholl, C., Becker, B.: 通过量词消去求解 DQBF。载于：DATE 论文集 (2015) 18. Golia, P., Roy, S., Meel, K.S.: Manthan：一种用于布尔函数综合的数据驱动方法。载于：CAV 论文集 (2020) 19. Golia, P., Slivovsky, F., Roy, S., Meel, K.S.: 工程化高效的布尔函数综合引擎。载于：国际计算机辅助设计会议论文集 (ICCAD) (2021) 20. Golia, P., Soos, M., Chakraborty, S., Meel, K.S.: 设计采样器很容易：测试器带来的益处。载于：FMCAD 论文集 (2021) 21. Gupta, R., Sharma, S., Roy, S., Meel, K.S.: WAPS：加权投影采样。载于：TACAS 论文集 (2019) 22. Henkin, L.: 关于无限长公式与无限方法的一些评论 (1959) 23. Heule, M.J., Seidl, M., Biere, A.: 从 QRAT 证明中高效提取 Skolem 函数。载于：FMCAD 论文集 (2014) 24. Hoos, H.H., Peitl, T., Slivovsky, F., Szeider, S.: 面向电路 QBF 的基于算法组合的算法选择。载于：CP 论文集 (2018) 25. Janota, M.: 基于电路的 QBF 搜索空间剪枝。载于：SAT 论文集。Springer (2018) 26. Janota, M.: 探索通过机器学习实现 QBF 求解中的泛化。载于：AAAI 论文集 (2018) 27. Jiang, J.H.R.: 通过函数复合进行量词消去。载于：CAV 论文集 (2009) 28. Jiang, J.H.R., Kravets, V.N., Lee, N.Z.: 用于组合与时序设计修正的工程变更单。载于：DATE 论文集 (2020) 29. John, A.K., Shah, S., Chakraborty, S., Trivedi, A., Akshay, S.: 因式分解公式的 Skolem 函数。载于：FMCAD 论文集 (2015) 30. Krynicki, M., Mostowski, M.: Henkin 量词。载于：《量词：逻辑、模型与计算》(1995) 31. Logic, B., Group, V.: ABC：一种用于时序综合与验证的系统 (2021), http://www.eecs.berkeley.edu/~alanmi/abc/ 32. Lonsing, F., Biere, A.: DepQBF：依赖感知 QBF 求解器。JSAT 论文集 (2010) 33. Lonsing, F., Egly, U.: DepQBF 6.0：一种超越传统 QCDCL 的基于搜索的 QBF 求解器。载于：CADE 论文集 (2017) 34. Martins, R., Manquinho, V., Lynce, I.: Open-WBO：模块化最大可满足性求解器。载于：SAT 论文集 (2014) 35. Peterson, G., Reif, J., Azhar, S.: 不完全信息多人非合作博弈的下界。《计算机与应用数学》(2001) 36. Quinlan, J.R.: 决策树的归纳。机器学习论文集 (1986) 37. Rabe, M.N.: 用于量词消去与函数综合的增量确定化。载于：CAV 论文集 (2019) 38. Rabe, M.N., Tentrup, L.: CAQE：可认证 QBF 求解器。载于：FMCAD 论文集 (2015)

39. Rabe, M.N., Tentrup, L., Rasmussen, C., Seshia, S.A.: 理解并扩展用于 2QBF 的增量确定化。载于：CAV 会议论文集（2018） 40. Reichl, F.X., Slivovsky, F., Szeider, S.: 通过定义提取实现经认证的 DQBF 求解。载于：可满足性会议论文集（2021） 41. Slivovsky, F.: 基于插值的语义门提取及其在量化布尔公式预处理中的应用。载于：CAV 会议论文集（2020） 42. Sˇıˇc, J., Strejˇcek, J.: DQBDD：一种高效的基于 BDD 的 DQBF 求解器。载于：可满足性会议论文集（2021） 43. Tentrup, L., Rabe, M.N.: DQBF 的子句抽象。载于：可满足性会议论文集（2019） 44. Wimmer, K., Wimmer, R., Scholl, C., Becker, B.: 用于量化布尔公式的 Skolem 函数。载于：ATVA 会议论文集（2016） 45. Wimmer, R., Reimer, S., Marin, P., Becker, B.: Hqspre——一种有效的量化布尔公式和 DQBF 预处理器。载于：TACAS 会议论文集（2017） 46. Xu, L., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Satzilla：面向可满足问题的基于算法组合的算法选择。JAIR 会议论文集（2008）