arXiv:2301.11229v1 [cs.LO] 26 Jan 2023

AutoHyper：HyperLTL 的显式状态模型检测

Raven Beutner(✉) Bernd Finkbeiner

CISPA赫尔姆霍兹信息安全中心，德国
{raven.beutner,finkbeiner}@cispa.de

摘要。 HyperLTL 是一种时序逻辑，可以表达超性质，即关联系统多条执行轨迹的性质。这类性质正变得越来越重要，并自然地出现在信息流控制、鲁棒性、变异测试、路径规划和因果关系检查等领域。迄今为止，完整的 HyperLTL 模型检测工具仅支持无交替公式，也就是只使用全称或只使用存在轨迹量化的公式。涉及量词交替的性质只能以不完整的方式处理，也就是说，即使性质成立，验证仍可能失败。本文介绍 AutoHyper，这是一种基于自动机的显式状态模型检查器，支持完整的 HyperLTL，并能完整处理具有任意量词交替的性质。我们展示了如何将语言包含性检查集成到 HyperLTL 验证中，使 AutoHyper 能够利用一系列现有的包含性检查工具。我们使用来自文献不同领域的大量基准测试评估 AutoHyper，并将其与现有的（不完整）HyperLTL 验证方法进行比较。

1 引言

超性质 [15] 是关联系统多次执行的系统属性。这类属性越来越重要，因为它们自然出现于信息流控制、 [35],鲁棒性 [21], 线性化 [29,30],路径规划 [38], 变异测试 [26],和因果关系检查 [17]等领域。用于表达超性质的一种重要逻辑是 HyperLTL，它通过显式的轨迹量化扩展了线性时序逻辑（LTL） [14]。例如，HyperLTL 可以表达广义非干扰（GNI） [33], ，即系统的高安全级输入不会影响可观察输出。

$$\forall \pi . \forall \pi' . \exists \pi'' . \square \Big(\bigwedge_{a \in H} a_\pi \leftrightarrow a_{\pi''} \Big) \wedge \square \Big(\bigwedge_{a \in L \cup O} a_{\pi'} \leftrightarrow a_{\pi''} \Big) \tag{GNI}$$

这里， H 是高安全级输入的集合， L 是低安全级别输入的集合， O 是低安全级别输出的集合。该公式表示，对于任意轨迹 π， π' 都存在第三条轨迹 π''，它与 π 的高安全级输入一致，并且与 π'的低安全级别输入和输出一致。因此，低安全级别攻击者作出的任何观察，都与每一种可能的高安全级输入相容。

我们关注 HyperLTL 的模型检测（MC）问题，即给定一个（有限状态）系统是否满足给定性质。对于 HyperLTL，量词前缀的结构会直接影响该问题的复杂性。对于无交替公式（即只含单一类型量词的公式），其验证方法已十分成熟，并且可归约为在系统 [3]的自组合上验证轨迹属性。例如，这种归约已在 `MCHyper` [28], 中实现；该工具能够对规模相当大的系统（含有数千个锁存器的电路）进行（无交替）HyperLTL 公式的模型检测。

对于涉及量词交替的性质，验证会更具挑战性（例如上文提到的GNI ）。尽管已有支持完整 HyperLTL 的模型检测算法（见 [14,28]），但这些算法尚未实现。多年来，人们提出了多种实用方法来验证这类性质：Finkbeiner 等人 [28] 和 D'Argenio 等人 [21] 手动将带有量词交替的性质强化为不含量词交替的性质，从而可由`MCHyper`进行检查。Coenen 等人 [18]通过显式指定（由用户提供的）策略，对 $\forall^*\exists^*$ 性质中的存在量化进行实例化（即，先包含任意数量的全称量词，随后包含任意数量的存在量词的性质，例如GNI），从而将问题化简为验证不含量词交替的公式。或者，也可以自动合成用于消解存在量化的策略 [8]。Hsu 等人 [30] 提出了一种适用于 HyperLTL 的有界模型检测（BMC）方法，并在`HyperQube`中实现。更多详情请参见第4节。

尽管所有这些验证工具都能验证（或证伪）有趣的性质，但它们都有同一个根本局限：它们是不完备的。也就是说，对于上述所有工具，我们都能找到它们无法处理的验证实例；这并非因为资源限制，而是因为底层验证算法固有的局限性。此外，这类实例并非罕见，而是在实践中经常遇到。例如，许多用于评估`HyperQube` 的基准测试（由 Hsu 等人 [30]进行）无法通过策略解决存在量化。反过来，Coenen 等人 [18] 验证的许多性质（例如 GNI）无法使用有界模型检测进行验证 [30]。

AutoHyper。 本文介绍了 AutoHyper，这是一款用于 Hyper-LTL 的模型检查器。该工具通过使用自动机补运算迭代消除轨迹量化，从而检查超性质，并将验证归约为自动机 [28] 的空性检查。重要的是，与 MCHyper [28,18] 和 HyperQube [30] 等以往用于 HyperLTL 验证的工具不同，AutoHyper 能够处理任意 HyperLTL 公式（并且对其而言，验证是完备的 ）。使用 AutoHyper 进行模型检测无需人工操作（例如在 MCHyper [18] 中编写显式策略），用户也不必担心给定性质是否能够通过某种方法进行验证。因此，AutoHyper 为 HyperLTL 提供了"一键式"模型检测体验。[1]

[1] AutoHyper 这一名称源于它既基于**自动机**，又是**自动的**（即它是完备的，且无需用户干预）。

为了提升 AutoHyper 的效率，我们从理论上观察到，通常可以避免显式地进行自动机补，而改为对 Büchi 自动机执行语言包含性检查（参见命题 1）。在实践中，这使得 AutoHyper 能够借助多种成熟的语言包含性检查器，包括 spot [25], RABIT [16], BAIT [24], 和 FORKLIFT [23]。

评估。 我们使用 AutoHyper，深入研究了带有量词交替的 HyperLTL 属性模型检测的实际应用。为了评估显式状态模型检测的性能，我们将 AutoHyper 应用于文献中的大量基准测试，并将其与现有的（不完备）工具进行比较。我们发现了一个令人惊讶的现象：至少在目前可用的基准测试上，AutoHyper 实现的显式状态模型检测与 BMC [30] 等符号化方法性能相当（且往往更胜一筹）。这些基准测试来自计算机科学的多个领域，因此，凭借其“一键式”功能、完备性和易用性，AutoHyper 应能成为许多领域的有力补充。

除了将 AutoHyper 用作实用的模型检测工具外，我们还可以将其作为完备的基准，系统地评估现有的（不完备）方法。例如，尽管已知以策略替代存在量化（如 Coenen 等人 [18]所做的那样）是不完备的，但迄今尚不清楚这种不完备性是否经常出现，还是仅仅是罕见现象。我们使用 AutoHyper 获取真实基准，并从有效性（即尽管方法不完备，它仍能验证多少个实例）和效率两方面评估基于策略的验证方法。

结构。 本文其余部分的结构如下。在第 2 节中，我们介绍 HyperLTL。第 3 节回顾基于自动机的验证（简称 ABV），并介绍我们采用语言包含性检查的新方法。第 4 节讨论 HyperLTL 的其他验证方法。在第 6 节中，我们比较不同的后端求解技术，并研究实践中具有多次量词交替的 HyperLTL 模型检查的复杂性；在第 7 节中，我们在一组文献中的基准测试上评估 ABV，并将其与有界模型检查器 HyperQube [30]进行比较；在第 8 节中，我们使用 AutoHyper对基于策略的验证进行详细分析（并与之比较） [18,8]。

2 预备知识

我们固定一个原子命题集合 AP，并定义 $\Sigma := 2^{AP}$。HyperLTL [14]通过对轨迹进行显式量化扩展了线性时序逻辑，从而将其从表达轨迹性质的逻辑提升为表达超性质的逻辑 [15]。设 $\mathcal{V}$ 为一个轨迹变量集合。我们通过以下文法定义 HyperLTL 公式：

$$\psi := a_\pi \mid \neg\psi \mid \psi \wedge \psi \mid \bigcirc\psi \mid \psi \,\mathcal{U}\, \psi$$
$$\varphi := \exists \pi.\, \varphi \mid \forall \pi.\, \varphi \mid \psi$$

其中 $\pi \in \mathcal{V}$ 和 $a \in AP$。

我们假设公式是闭合的，即主体中使用的所有轨迹变量都由某个量词绑定。HyperLTL 的语义是相对于轨迹赋值 Π 来定义的： $\mathcal{V} \rightharpoonup \Sigma^\omega$ 将轨迹变量映射到轨迹。对于 $\pi \in \mathcal{V}$ 和 $t \in \Sigma^\omega$，我们用 $\Pi[\pi \mapsto t]$ 表示将 π 的值更新为 t后得到的轨迹赋值。给定一组轨迹 $\mathbb{T} \subseteq \Sigma^\omega$、一个轨迹赋值 Π和 $i \in \mathbb{N}$，我们定义：

$$
\begin{aligned}
&\Pi, i \models a_\pi \text{ 当且仅当 } a \in \Pi(\pi)(i) \Pi, i \models \neg\psi \text{ 当且仅当 } \Pi, i \not\models \psi \Pi, \\
&i \models \psi_1 \wedge \psi_2 \text{ 当且仅当 } \Pi, i \models \psi_1 \text{ 和 } \Pi, i \models \psi_2 \Pi, i \models \bigcirc \psi \text{ 当且仅当 } \Pi, \\
&i+1 \models \psi \Pi, i \models \psi_1 \mathcal{U} \psi_2 \text{ 当且仅当 } \exists j \geq i.\, \Pi, j \models \psi_2 \text{ 和 } \forall i \leq k < j.\, \Pi, \\
&k \models \psi_1 \Pi \models_{\mathbb{T}} \psi \text{ 当且仅当 } \Pi, 0 \models \psi \Pi \models_{\mathbb{T}} \exists \pi.\, \varphi \text{ 当且仅当} \\
&\exists t \in \mathbb{T}.\, \Pi[\pi \mapsto t] \models_{\mathbb{T}} \varphi \Pi \models_{\mathbb{T}} \forall \pi.\, \varphi \text{ 当且仅当 } \quad \forall t \in \mathbb{T}.\, \Pi[\pi \mapsto t] \models_{\mathbb{T}} \varphi
\end{aligned}
$$

转移系统是一个元组 $\mathcal{T} = (S, S_0, \kappa, L)$，其中 S是状态集合，$S_0 \subseteq S$是初始状态集合， $\kappa \subseteq S \times S$是迁移关系， $L\colon S \to \Sigma$是标记函数。每当 $(s, s') \in \kappa$时，我们记作 $s \xrightarrow{\mathcal{T}} s'$。路径是一个无限序列 $s_0 s_1 s_2 \cdots \in S^\omega$，满足 $s_0 \in S_0$，且对于所有 i，都有 $s_i \xrightarrow{\mathcal{T}} s_{i+1}$。关联的轨迹定义为 $L(s_0)L(s_1)L(s_2) \cdots \in \Sigma^\omega$。我们用轨迹集合$(\mathcal{T}) \subseteq \Sigma^\omega$表示由 $\mathcal{T}$生成的所有轨迹的集合。如果 $\emptyset \models_{Traces(\mathcal{T})} \varphi$，则称 $\mathcal{T}$满足 HyperLTL 属性 φ，记作 $\mathcal{T} \models \varphi$，其中 $\emptyset$ 表示空轨迹赋值。

3 基于自动机的 HyperLTL 模型检测

给定一个系统 $\mathcal{T}$和 HyperLTL 属性 φ，我们希望判定 $\mathcal{T} \models \varphi$。本节回顾用于 HyperLTL 模型检测的基于自动机的方法 [28]。我们还将说明如何将语言包含性检查纳入模型检测流程，以利用大量成熟的语言包含性检查器。

3.1 基于自动机的验证

基于自动机的验证（ABV） [28]的思路是迭代地消去量词，从而将模型检测归约为对自动机进行空性检查。非确定性 Büchi 自动机（NBA）是一个元组 $\mathcal{A} = (Q, Q_0, \delta, F)$，其中$Q$ 是状态的有限集， $Q_0 \subseteq Q$ 是初始状态的集合， $\delta\colon Q \times \Sigma \to 2^Q$ 是转移函数， $F \subseteq Q$ 是接受状态的集合。我们用 $\mathcal{L}(\mathcal{A}) \subseteq \Sigma^\omega$ 表示 $\mathcal{A}$的语言，即所有存在一种运行、使其无限多次访问 F中的状态的无限字（例如，参见 [2]）。对于轨迹 t_1、.....、 $t_n \in \Sigma^\omega$，我们将 zip(t_1、.....、 t_n) $\in (\Sigma^n)^\omega$ 记作逐点积，即 zip(t_1、.....、 t_n)(i) :=($t_1(i)$....., $t_n(i)$）。

设 $\mathcal{T} = (S, S_0, \kappa, L)$ 是一个固定的转移系统，并设 $\dot{\varphi}$ 是某个固定的闭合 HyperLTL 公式（我们用点号表示原始公式，用 φ, φ' 表示 $\dot{\varphi}$的子公式）。对于包含自由轨迹变量 $\pi_1, \ldots, \pi_n$ 的某个子公式 φ，如果对于所有轨迹 $t_1, \ldots, t_n$，都有 $[\pi_1 \mapsto t_1, \ldots, \pi_n \mapsto t_n] \models_{Traces(\mathcal{T})} \varphi$ 当且仅当 $zip(t_1, \ldots, t_n) \in \mathcal{L}(\mathcal{A})$，则称定义在 Σ^n 上的 NBA $\mathcal{A}$ 与 φ $\mathcal{T}$- 等价。也就是说，$\mathcal{A}$ 恰好接受那些轨迹的交织序列；这些轨迹构成了满足 φ的轨迹赋值。

为了检查 $\mathcal{T} \models \dot{\varphi}$，我们归纳地构造一个自动机 $\mathcal{A}_\varphi$，使其与 φ 在 φ 的每个子公式上都是 $\mathcal{T}$ 等价的。对于 $\dot{\varphi}$ 的（无量词）LTL 主体，我们可以通过标准的 LTL 到非确定性 Büchi 自动机的构造[28,2]来构造该自动机。现在考虑某个子公式 $\varphi' = \exists \pi. \varphi$，其中 φ' 包含自由轨迹变量 π_1、……、π_n，因此 φ 包含自由轨迹变量 π_1、……、π_n、π。我们已归纳构造出一个定义在 Σ^{n+1} 上的非确定性 Büchi 自动机 $\mathcal{A}_\varphi = (Q, Q_0, \delta, F)$，它与 φ 是 $\mathcal{T}$ 等价的。我们将定义在 Σ^n 上的自动机 $\mathcal{A}_{\varphi'}$ 定义为 $\mathcal{A}_{\varphi'} := (S \times Q, S_0 \times Q_0, \delta', S \times F)$，其中 δ' 定义为

$$\delta'\Big((s,q), \langle l_1, \ldots, l_n \rangle\Big) := \Big\{ (s', q') \mid s \xrightarrow{\mathcal{T}} s' \wedge q' \in \delta\big(q, \langle l_1, \ldots, l_n, L(s) \rangle\big) \Big\}.$$

非正式地说，$\mathcal{A}_{\varphi'}$ 读取轨迹 $t_1, \ldots, t_n$ 的交织，并猜测一条轨迹 $t \in Traces(\mathcal{T})$，使得 $zip(t_1, \ldots, t_n, t) \in \mathcal{L}(\mathcal{A}_\varphi)$。显然，$\mathcal{A}_{\varphi'}$ 与$\mathcal{T}$等价于 φ'。为了处理全称轨迹量化，我们将公式 $\varphi' = \forall \pi. \varphi$ 视为 $\varphi' = \neg \exists \pi. \neg \varphi$，并将存在量化的构造与自动机补相结合。

按照归纳构造，我们得到一个定义在单元素字母表 Σ^0 上的自动机 $\mathcal{A}_{\dot{\varphi}}$，它与 $\dot{\varphi}$等价于 $\mathcal{T}$。根据 $\mathcal{T}$-等价的定义，$\mathcal{T} \models \dot{\varphi}$ 当且仅当 $\emptyset \models_{Traces(\mathcal{T})} \dot{\varphi}$ 当且仅当 $\mathcal{A}_{\dot{\varphi}}$ 非空（我们可以通过 [20]判定）。

3.2 通过语言包含进行 HyperLTL 模型检查

上述算法要求对 HyperLTL公式中的每次量词交替进行一次补运算。虽然我们无法避免补运算在理论上的代价（见 [35,14]），但在实践中，我们可以将问题简化为一个更易处理的问题：语言包含。

对于系统 $\mathcal{T}$和自然数 $n \in \mathbb{N}$，我们定义 $\mathcal{A}^n_\mathcal{T}$为一个关于$\Sigma^n$ 的非确定性 Büchi 自动机，使得对于任意轨迹 $t_1, \ldots, t_n \in \Sigma^\omega$，都有 $zip(t_1, \ldots, t_n) \in \mathcal{L}(\mathcal{A}^n_\mathcal{T})$ 当且仅当 $t_i \in Traces(\mathcal{T})$ 对每个 $1 \leq i \leq n$都成立。我们可以通过构造 $\mathcal{A}^n_\mathcal{T}$的 n重自组合来构造 $\mathcal{T}$ [3]，再将状态上的标签移到边上，并将所有状态标记为接受状态，从而将其转换为自动机。现在，我们可以阐述语言包含与 HyperLTL 模型检查之间的正式联系（证明见附录A）：

命题 1。设 $\dot{\varphi} = \forall \pi_1 \ldots \forall \pi_n. \varphi$为一个 *HyperLTL*公式（其中 φ 可能包含其他轨迹量词），并设 $\mathcal{A}_\varphi$ 为定义在 Σ^n 上的自动机，该自动机与 φ在 $\mathcal{T}$意义下等价。那么，当且仅当 $\mathcal{T} \models \dot{\varphi}$ $(\mathcal{A}^n_\mathcal{T}) \subseteq \mathcal{L}(\mathcal{A}_\varphi)$。

我们可以利用命题1来避免对最外层量词交替进行补。例如，假设 $\dot{\varphi} = \forall \pi_1. \forall \pi_2. \exists \pi_3. \psi$ 其中 ψ 是无量词的。利用第3.1节中的构造，我们得到一个自动机 $\mathcal{A}_{\exists \pi_3. \psi}$

它与 $\mathcal{T}$等价，其中 $\exists\pi_3.\psi$ （我们可以在 $\mathcal{T}$的尺寸的线性时间内构造 $\mathcal{A}_{\exists\pi_3.\psi}$ ）。根据命题1，于是有 $\mathcal{T} \models \dot{\varphi}$ 当且仅当 $\mathcal{L}\left(\mathcal{A}_{\mathcal{T}}^{2}\right) \subseteq \mathcal{L}\left(\mathcal{A}_{\exists\pi_3.\psi}\right)$。

请注意，补及后续的空性检查是解决（PSPACE完全）语言包含问题的理论最优方法。命题1 因此在渐近意义上并不优于第3.1节中的“标准” ABV。在实践中，通常无需构造显式的补自动机，因为无需进行完整的补运算，就可能得到语言包含或不包含的见证 [25,24,16,23]。因此，命题1与本文工作以及 AutoHyper的性能相关。

4 相关工作与 HyperLTL 验证方法

HyperLTL [14] 是表达超性质研究最广泛的逻辑。计算机科学不同领域中的一系列问题都可以表述为 HyperLTL 模型检测问题，包括（最优）路径规划 [38], 变异测试 [26],线性化 [30], 鲁棒性 [21],信息流控制 [35], 和因果关系检查 [17],，这里只列举了少数几个。因此，任何用于 HyperLTL 的模型检测工具都适用于计算机科学中的诸多学科，并为许多具有挑战性的算法问题提供统一的解决方案。近年来，人们开发了多种用于 HyperLTL 验证的方法（大多并不完备）。下文将对此进行讨论（详见附录B ）。

基于自动机的模型检测。 Finkbeiner 等人 [28] 介绍了第3.1节所述的基于自动机的模型检测方法。对于无交替公式，这些算法对应于系统的自组合构造 [3] ，并已在MCHyper 工具中实现 [28]。 MCHyper 能够处理尺寸相当大的系统（远超显式状态方法的处理能力），但无法处理任何量词交替（这正是 AutoHyper的主要动机）。htltl2mc [14] 是一个用于 HyperLTL 的原型模型检查器$_2$ （用于处理至多包含一次交替的 HyperLTL 片段），构建于 GOAL [37]之上。与 htltl2mc相比， AutoHyper 支持包含任意多次量词交替的性质，并采用具有符号字母表的自动机——这对于处理含有大量原子命题的大型系统至关重要，参见脚注 7）。

基于策略的验证。 Coenen 等人 [18]通过使用显式策略实例化存在量化来 $\forall^*\exists^*$ 验证属性。这种方法——我们称之为基于策略的验证（SBV）——有两种形式：策略可以由用户提供，也可以自动综合。在前一种情况下，模型检测可归结为检查无交替公式，因此能够处理大型系统，但需要用户付出大量工作（因而不是一种“按键即用”的技术）。在后一种情况下，该方法完全自动 [9,8] 运行，但需要代价高昂的策略综合。当用于解析存在量化轨迹的策略只能观察全称量化轨迹的有限前缀时，SBV 是不完备的。尽管可以通过

添加预言变量 [8], 使 SBV 完备，但预言的自动综合目前仅限于非常小的系统和在时序上安全的属性 [5]。我们将在第 8 节研究 SBV 的性能和不完备性。

有界模型检测。 Hsu 等人 [30] 提出了一种用于 HyperLTL 的有界模型检测（BMC）过程。与针对轨迹性质的 BMC 类似 [10], ，系统会展开到固定深度，超过该深度的待处理义务则采用悲观方式（以证明公式成立）或乐观方式（以证明公式不成立）处理。针对轨迹性质的 BMC 可归约为 SAT 求解，而针对超性质的 BMC 则自然归约为 QBF 求解。与有界方法通常的情况一样，HyperLTL 的 BMC 并不完备。例如，当 LTL 主体包含不变式时（例如 GNI 的情况），它无法证明系统满足某个超性质。[2] 我们将在第 7 节中比较 AutoHyper 和 BMC（以 `HyperQube` [30]的形式）。7。

5 AutoHyper：工具概览

`AutoHyper`使用 `F#` 编写，实现了第 3.1 节所述的基于自动机的验证方法，并且可根据用户需要使用第 3.2 节介绍的基于语言包含的归约。 `AutoHyper` 使用 `spot`[25] 进行 LTL 到 NBA 的转换和自动机补。检查语言包含时，`AutoHyper` 使用 `spot` （基于确定化）、`RABIT` [16]（基于大量使用模拟的 Ramsey 方法）、`BAIT` [24], 以及 `FORKLIFT` [23]（后两者均基于良拟序）。`AutoHyper`旨在通过成熟的基于文本的格式（而非专有 API）与外部自动机工具通信，具体采用 `HANOI`[1] 和 `BA` 自动机格式。因此，改进自动机补和包含性检查等基本自动机操作的新工具或更新版工具，都可以轻松集成。内部表示自动机时，我们使用符号字母表（类似于 `spot`）。转移公式以析取范式存储，以便高效执行乘积构造过程中所需的 SAT 检查。

本文中的所有实验均在配备 Intel Core i3（i3-8100B）和 16GB 内存的 Mac Mini 上进行。我们使用了 spot 版本 2.11.1、RABIT 版本 2.4.5、BAIT 提交 369e1a4 和 FORKLIFT 提交 5d519e3。

输入格式。AutoHyper 支持显式状态系统（采用类似 HANOI 的 [1] 输入格式）以及会在内部转换为

[2] 通过对展开深度（也称为完备性阈值）设置界限 [13] ，并在编码中加入循环条件 [10], 可以使针对轨迹性质的有界模型检测达到完备。正如 Hsu 等人 [30], 指出的，要将同样的方法用于超性质，挑战要大得多，目前尚未有解决方案。Hsu 等人 [30]转而提出一种替代的展开语义（他们称之为停机语义），对于在固定步数后终止的程序，这种语义可以缓解完备性不足的问题。这一假设要求很高，对于一般的反应式系统来说往往并不现实。

显式状态表示的符号系统。对符号系统的支持包括 Aiger 电路、以 NuSMV 输入语言的一个片段编写的符号模型 [12],以及一种简单的布尔编程语言 [6]。

随机基准测试。在评估中，我们既使用文献中各种来源的现有实例，也使用随机生成的问题。[3] 我们基于 Erdős–Rényi–Gilbert 模型生成随机转移系统 [27]。给定尺寸 n 和密度参数 $p \in [0,1]$，我们生成一个包含 n 个状态的图，其中任意两个状态 s，s'之间以概率 p存在一条转移 $s \to s'$。为了生成包含 n 条边且期望出度为常数 k的图，我们可以选择 $p = \frac{k}{n}$。此外，我们确保系统是连通的，并且每个状态至少有一条出边。我们通过使用 spot 的 randltl 对 LTL 矩阵进行采样，生成随机 HyperLTL 公式（具有给定的量词前缀）。

6 实践中的 HyperLTL 模型检查复杂度

在讨论文献中的基准测试之前，我们先通过在大量合成（随机）基准测试上评估AutoHyper支持的不同后端包含性检查器（见第6.1节），并对它们进行比较。此外，随机生成基准测试使我们得以探究具有多个量词交替的公式。具有多次交替的性质的模型检测在理论上的难度已得到广泛研究 [14,35]，而我们首次分析这些结果在实践中的表现（见第6.2节）。

6.1 包含性检查器的性能 rs

作为第一组基准测试，我们比较AutoHyper支持的不同后端包含性检查器。在图1中，我们展示了在 10 秒超时时间内，使用spot、BAIT、RABIT和FORKLIFT的包含性检查能够解决多少个实例，并给出超时时间内可解决实例的运行时间中位数。我们发现，就能在 10 秒内完成检查的实例百分比而言，spot明显优于RABIT、BAIT和FORKLIFT。[4] 尽管总体而言，spot 解决的实例最多，但经人工检查发现，也有一些实例只能由RABIT

[3]随机生成实例有两方面的优势。首先，它可以轻松生成大量基准测试。其次，随机生成由多个参数（例如系统规模、转移密度、公式大小等）控制，因此能够全面分析不同参数在实践中对模型检测复杂性的确切影响。[4] 需要指出的是，spot 使用符号字母表上的自动机（即转移由关于 AP 的布尔公式定义）。相比之下，RABIT、BAIT 和 FORKLIFT 仅支持显式字母表（即自动机针对 2^{AP} 中的每个元素各有一个符号）。

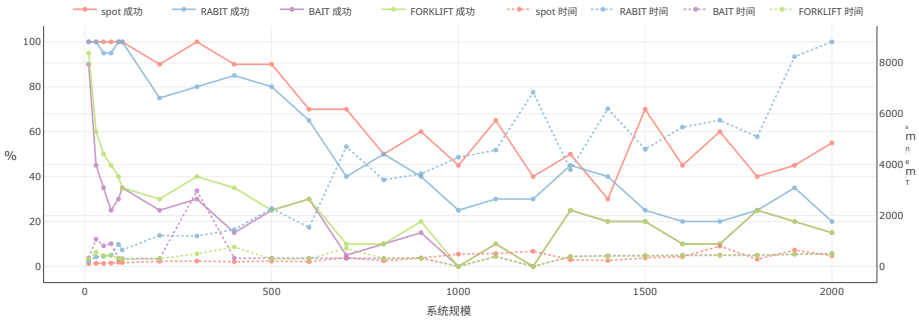


图1：我们在系统规模不同的实例上评估了不同的后端求解器，平均出度为 10，性质规模固定为 20。每种系统规模生成 20 个样本。我们展示了各求解器在 10 秒超时时间内的成功率（左轴），以及已求解实例的运行时间中位数（右轴）。

或 BAIT/FORKLIFT。这说明了为什么 AutoHyper支持多个后端语言包含检查器，它们实现了不同的算法，因此分别擅长处理不同的问题（我们将在第7节验证这一点）。此外，我们的实验表明，HyperLTL 模型检测是具有挑战性的语言包含基准测试的天然来源（见附录 C）。

我们特意将超时时间设为较短的 10 秒，以便在合理的时间内计算（并重现）这些图（计算图 1耗时约 3.5 小时）。如果用户想验证给定实例，且不要求在几秒内得到结果，那么延长求解器的运行时间很可能会进一步提高成功率（另见第 7节的评估）。

6.2 超越 ∀*∃* 的模型检测

借助随机生成的基准测试，我们还可以一窥存在多重量词交替时模型检测的实际复杂性。在理论上，HyperLTL 的模型检测复杂度会随着每次量词交替增加一个指数 [14,35]。借助 AutoHyper，我们首次能够在实践中研究模型检测的复杂性。

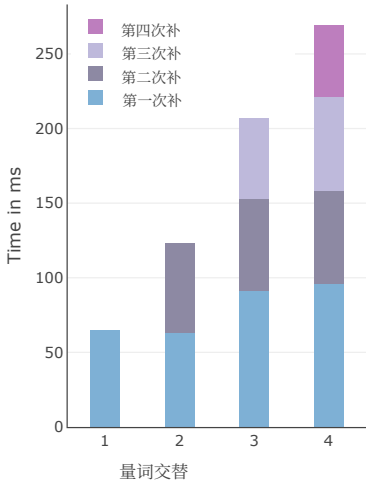


图2：对于量词交替次数不同的性质，我们展示了模型检测过程中自动机补所花费的平均时间。

我们对随机生成的公式进行模型检查，涉及 1 至 4 次量词交替，并根据每次补 spot的开销，展示总运行时间，如图 2 所示（请回想，使用 k 次交替的公式

表 1：我们展示了 AutoHyper 验证GNI 时的运行时间，所使用的布尔程序取自 [6] 和 [7]。我们给出了程序、位宽（ bw ）、中间显式状态表示的尺寸（ Size）以及各个求解器所用的时间。超时时间设置为 60 秒，并用“–”表示。所有情况下性质均成立。时间以秒为单位。

程序	bw	尺寸	t_{spot}	t_{RABIT}	t_{BAIT}	t_{FORKLIFT}
[6],1	1位	17	**0.52**	0.59	0.80	0.61
	3位	65	**0.56**	0.86	-	22.73
	4位	129	**0.99**	5.51	-	-
[6],2	1位	55	**0.53**	0.69	-	5.49
[6],3	1位	20	**0.52**	0.61	3.05	0.98
	3位	80	**0.61**	1.31	-	-
[6],4	1位	29	**0.52**	0.56	0.58	0.57
	3位	113	**0.67**	1.74	-	-

程序	bw	尺寸	t_{spot}	t_{RABIT}	t_{BAIT}	t_{FORKLIFT}
[7],1	1位	5	0.52	**0.56**	0.58	0.57
[7],2	1位	11	**0.51**	0.57	0.72	0.61
	2位	27	**0.52**	0.65	35.7	5.43
	4位	291	**1.46**	-	-	-
[7],3	1位	21	**0.52**	0.60	3.15	1.00
	3位	225	-	**45.2**	-	-
[7],4	1位	25	**0.52**	0.71	12.8	1.63
	3 位	193	**0.98**	-	-	-

使用 ABV 需要进行 k 次自动机补。尽管量词交替次数无疑会影响总运行时间（即各柱的累积高度），但运行时间的增长并不像理论分析所预示的那样，与（非初等的）增长幅度成正比。与理论分析不同（其中第 $k+1$ 次补的开销比第 k 次高出指数级），每次补的开销几乎没有增加（甚至有所下降）。这表明，实际中每轮构造的 $\mathcal{T}$ 等价自动机远小于最坏情况理论分析所预示的规模。因此，验证具有多于一次量词交替的性质，并不像理论所说的那么不可行（至少对于随机生成的测试用例而言）。

7 符号系统上的评估

在本节中，我们使用文献中发现的复杂模型检测问题来检验 AutoHyper。我们的基准测试来自多种来源，包括布尔程序中的非干扰 [6],互斥算法中的对称性 [18], 多线程程序中的非干扰 [36], 不可否认协议中的公平性 [31], 变异测试 [26], 以及路径规划 [38]。

7.1 布尔程序上的 GNI 模型检测

我们使用 AutoHyper验证 GNI ，测试对象是一系列处理高安全级别和低安全级别输入的布尔程序（取自 [6,7]）。表 1展示了使用不同后端求解器时的运行时间结果。我们使用不同位宽测试每个程序，并展示至少一个求解器能够求解的最大位宽（超时时间设为 60 秒）。我们再次注意到，spot 的表现优于其他

表 2：我们在以下基准测试中评估 HyperQube和 AutoHyper： [30]。我们列出系统及其性质（如 [30, 表 2] 中所示）、量词结构（Q^*）、验证结果（Res）（✓ 表示该性质成立， ✗ 表示该性质不成立），以及任一工具的总运行时间（t）。对于HyperQube，我们还列出展开界（k）和展开语义（Sem）。对于AutoHyper，我们还列出中间显式状态空间的尺寸（Size）。时间以秒为单位。

				HyperQube [30]			**AutoHyper**	
系统	**Spec**	Q^*	**Res**	k	**Sem**	t	**Size**	t
Bakery$_3$	φ_{S1}	∃∃	✗	7	pes	**1.9**	167	2.3
面包店$_3$	φ_{S2}	∀∃	✗	12	pes	**2.0**	167	4.2
面包店$_3$	φ_{S3}	∃∀	✗$^!$	20	pes	**2.8**	167	34.6
面包店$_3$	φ_{sym1}	∀∃	✗	10	pes	**1.7**	167	16.2
面包店$_3$	φ_{sym2}	∀∃	✗	10	pes	**1.6**	167	2.9
面包店$_5$	φ_{sym1}	∀∃	✗	10	pes	**17.3**	996	282.1
面包店$_5$	φ_{sym2}	∀∃	✗	10	pes	18.2	996	**18.0**
SNARK-bug1	φ_{lin}	∀∃	✗	26	hpes	618.0	4941	**96.1**
3-线程$_{correct}$	φ_{NI}	∀∃	✓	10	hopt	1.6	64	**1.3**
3-线程$_{incorrect}$	φ_{NI}	∀∃	✗	57	hpes	12.8	368	**7.7**
$NRP : T_{correct}$	φ_{fair}	∃∀	✓	15	hopt	1.3	55	**0.5**
$NRP : T_{incorrect}$	φ_{fair}	∃∀	✓$^!$	15	hopt	1.4	54	**0.8**
Mutant	φ_{mut}	∃∀	✓	8	hopt	1.1	32	**0.8**

包含性检查器；尤其是随着系统尺寸增大，它的扩展性更好。请注意，所有实例中的原子命题数量均为3 ，因此spot对符号字母表的支持对运行时间影响微乎其微。我们强调，表1 中的实例并非都能通过 SBV 验证[18,8] ，除非用户提供固定前瞻。同样，BMC [30] 永远无法验证 GNI。这进一步说明了为什么我们需要完备的模型检测工具（AutoHyper 是首个此类工具）。

7.2 符号系统的显式模型检测

在本节中，我们使用 Hsu 等人 [30]用于评估 HyperQube 的 NuSMV 模型 [12]，评估 AutoHyper 在具有挑战性的符号模型上的表现。

我们验证的性质涵盖范围广泛。例如，我们验证了 Lamport 面包店算法 [32] 不满足各种对称性性质（因为该算法会优先处理票号较小的进程）；我们

检查 SNARK 数据结构上的线性化[5] [29] [22]并发现了一个此前已知的错误；此外，我们采用 Fellner 等人提出的方法生成基于模型的变异测试用例。 [26]。有关基准测试的更多细节，请参阅 [30]。

我们使用 HyperQube 和 AutoHyper检查每个实例，并将结果展示在表 2中。[6] 使用 AutoHyper 时，我们始终使用 spot 的包含性检查器。[7] 对于 HyperQube ，我们采用 [30, 表 2]中列出的展开语义和展开深度。我们观察到，对于大多数实例而言，尽管使用了显式状态方法，因此能够保证完备性（参见第 7.4 节），AutoHyper 的表现仍与 HyperQube 不相上下。对于采用 Lamport 面包店算法的实例，有界模型检测只需展开到很浅的深度，因此求解效率很高；而 AutoHyper 的运行时间主要耗费在 spot 的线性时序逻辑到非确定性 Büchi 自动机的转换上（最多占总时间的 98%）。相反，在大型 SNARK 示例中，AutoHyper 的表现明显更好。

7.3 路径规划的超性质

作为最后一组基准测试，我们采用 Wang 等人提出的、编码为 HyperLTL 的机器人规划问题 [38]。例如，最短路径的综合可以表述为一个 $\exists\forall$ 性质：存在一条通往目标的路径，使得所有其他通往目标的路径都至少需要同样长的时间。Wang 等人 [38] 提出了一种检查所得 HyperLTL 属性的方案：将其编码为一阶逻辑，再由可满足性模理论求解器求解。尽管其性能不及先进的规划工具，HyperLTL 仍能以非常通用的方式表达广泛的问题（最短路径、路径鲁棒性等）。Hsu 等人 [30]发现，HyperQube 中实现的 QBF 编码优于 Wang 等人 [38]提出的基于 SMT 的方法。本节中，我们评估 AutoHyper 在这些规划超性质上的表现，并将其与 HyperQube[8]进行比较。

我们在表 3中展示结果。显然，AutoHyper的表现优于HyperQube，有时甚至快几个数量级。这令人惊讶，因为对于符号化系统上的规划问题（本质上是可达性问题），BMC 等符号化方法本应更具优势。较大的显式状态空间的尺寸.....

[5] 线性化断言，并发数据结构的任何执行都对应于一次顺序执行，这可以自然地表达为一个 $\forall\exists$ 超性质。[6] 对于两个验证实例（Bakery3、φ_{S3}）和（NRP： $T_{incorrect}$、 φ_{fair}），HyperQube 提供了错误的验证结果。为避免将表 2 与 [30, 表 2]进行比较时产生混淆，我们用“！”标记此类实例。特别是，所谓不公平版本的 NRP 协议实际上是公平的。[7] 自动机使用包含最多 18 个字母的符号字母表。因此，转换为显式字母表——RABIT、BAIT 和 FORKLIFT 所要求的形式——是不可行的（这将需要 2^{18} 个符号）。[8] AutoHyper 被设计为一种模型检测工具，即它只检查某个性质是否成立或被违反。然而，正如我们在附录 D 中展示的那样，我们可以利用包含性检查器返回的反例来综合出一个实际计划。

表 3：我们在编码最短路径（φ_{sp}）和鲁棒路径（φ_{rp}）存在性的超性质上评估 HyperQube 和 AutoHyper。我们给出规格（Spec）、网格尺寸（Grid），以及 HyperQube 和 AutoHyper 所用的时间（t）。对于 HyperQube，我们还给出所用的展开深度（k）以及生成的量化布尔公式文件的大小（|QBF|）。对于 AutoHyper，我们还给出生成的显式状态空间的尺寸（尺寸）。时间以秒为单位。超时时间设为 20 分钟，并以“-”表示。

		HyperQube [30]			AutoHyper	
Spec	**Grid**	k	|量化布尔公式|	t	**Size**	t
φ_{sp}	10×10	20	8 MB	4.6	146	**0.68**
	20×20	40	26 MB	168.1	188	**1.50**
	40×40	80	-	-	408	**22.70**
	60×60	120	-	-	404	**88.83**
φ_{rp}	10×10	20	13 MB	4.2	266	**0.59**
	20×20	40	84 MB	22.4	572	**0.78**
	40×40	80	419 MB	265	1212	**1.58**
	60×60	120	-	-	1852	**3.70**

中间量化布尔公式表明，采用更优化的编码（或许针对路径规划）可以提升有界模型检测在此类示例上的性能。

7.4 有界模型检测与显式状态模型检测

有界模型检测在轨迹性质验证方面取得了显著成功，而且通常能够扩展到尺寸远超显式状态方法适用范围的系统 [19]。同样，在无交替超性质的背景下，MCHyper [28] 等符号验证工具（其内部会将问题归约为使用 ABC [11]验证电路）能够验证远超显式状态方法能力范围的系统。相比之下，对于涉及量词交替的超性质模型检测，我们的研究结果有力地表明，应采用显式状态方法（如 AutoHyper 中所实现的方法）：

首先，与符号化方法（例如有界模型检测）相比，显式状态模型检测目前是唯一一种 完备的方法。尽管有界模型检测能够验证或反驳表 2 和 3中的所有性质，但使用当前的有界模型检测编码无法解决许多实例。一个具体例子是，有界模型检测 永远无法 验证其主体包含简单不变式（例如 GNI）的公式，因此也无法验证表 1中的任何实例。因此，显式状态模型检测（如 AutoHyper中所实现的）的最大优势在于，它既是按键式的，又是完备的；也就是说，至少在理论上，它能够验证或反驳所有性质。

其次，AutoHyper的性能似乎 不相上下 ，并且经常优于有界模型检测（甚至达到几个数量级，参见表3）。需要强调的是，尽管在评估 HyperQube 时我们已经固定了展开深度和展开语义，从而为 HyperQube创造了有利条件，但仍然得出了这一结果。[9] 对于轨迹性质，有界模型检测可归约为 SAT 求解；而对于超性质，有界模型检测则可归约为量化布尔公式求解，这是一个难得多的问题，并且工业级工具对它的支持也较少。因此，目前尚不清楚现代 QBF 求解器的进步能否改善超性质有界模型检测的性能，达到 SAT 求解器的进步推动轨迹性质有界模型检测成功的同等程度。我们的研究结果似乎表明，目前量化布尔公式求解通常不如显式的（基于自动机的）求解策略。

8 评估基于策略的验证

到目前为止，我们一直使用 AutoHyper 检查文献中实例所涉及的超性质。在最后这一节中，我们将展示 AutoHyper 也可作为有价值的基准，用于评估不同的（可能不完备的）验证方法。这里我们关注基于策略的验证（SBV），即自动合成一种策略，以解决$\forall^*\exists^*$ HyperLTL 属性 [18,8]中的存在量化。

8.1 基于策略的验证的有效性

众所周知，SBV 并不完备 [18,8]。然而，由于此前缺乏完备的 工具来验证 $\forall^*\exists^*$ 性质，因此无法大规模（即超越人工构造的示例）详细研究 SBV 在实践中的有效性。借助 AutoHyper，我们首次能够对 SBV 进行严格评估。我们使用 [8],的 SBV 实现，它通过将公式转换为确定性奇偶自动机（ DPA ） [34] ，为$\exists$玩家合成策略，并将合成问题表述为奇偶博弈。

我们生成了不同尺寸的随机转移系统和性质，并使用 AutoHyper 计算了真实基准。随后我们进行了 SBV（请记住，SBV 永远无法证明某个性质不成立，也可能无法确立其成立）。我们发现，在生成的实例中，**61.1%** 的情况下性质成立，而 SBV 在**60.4%** 的情况下能够验证该性质。因此，即使不添加预言 [8] 等开销高昂的机制，许多情况下仍可以通过 SBV 成功完成验证。另一方面，我们的结果表明，随机生成会产生一些实例（尽管数量不多）

[9] 在表 2 和 3中，我们针对固定的展开深度 k 和语义执行单次查询。也就是说，我们已经知道要验证满足还是违反，以及证明这一点所需的深度（如 [30]中所示）。在经典有界模型检测循环中，我们会逐步增加展开深度，分别检查满足和违反，因此大约需要执行 $2k$ 次量化布尔公式查询，其中 k 是能够判定满足或违反的最小界限（如果存在这样的界限）。

对于这些问题，SBV 会失败（迄今为止，SBV 失败的示例都需要手工仔细构造）。因此，无法恢复将 SBV 作为默认验证策略，这进一步表明完整模型检测工具的必要性（AutoHyper 是首个此类工具）。

8.2 基于策略的验证效率

在分析了 SBV 的有效性（即有多少实例能够得到验证）之后，我们转而关注 SBV 的效率。从理论上讲，对 ∀*∃* HyperLTL 进行（基于自动机的）模型检测——正如 AutoHyper 中的实现——在规约方面是指数空间完全的，在 PSPACE系统规模方面是完全的 [14,35]。相比之下，SBV 在规约规模方面是2-EXPTIME完全的，但在系统规模方面仅为多项式时间 复杂度 [18]。因此，人们会预期 ABV 在较大规约上表现更好，而 SBV 在较大系统上表现更好（这是实践中更重要的衡量指标）。

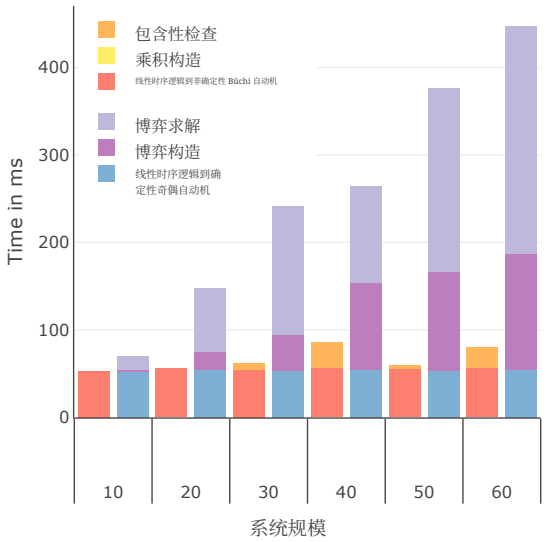


图 3：在性质规模固定为 20、系统规模各异的实例上比较 ABV 和 SBV。对于每种规模，我们生成 100 个随机实例，并对最快的L 个实例取平均值，其中 L是两种方法都能在 5 秒超时时间内解决的实例的最小数量。

然而，在本节中，我们将证明这在实践中并不成立（至少使用 SBV [8] 的当前实现时如此）。我们比较 AutoHyper（ABV）（使用 spot 的包含性检查器）和 SBV 的运行时间。我们将每种方法的运行时间分为三个主要步骤。对于 ABV，这些步骤是 LTL 到 NBA 的转换、乘积自动机的构造以及包含性检查。对于 SBV，这些步骤是 LTL 到 DPA 的转换、博弈的构造以及博弈求解。

我们在图中展示了不同系统规模下的平均成本 3。我们观察到，SBV 的表现不如 ABV；更重要的是，随着系统规模增大，其扩展性较差。这与 ABV 和 SBV 的理论分析相矛盾。运行时间的详细分解表明，表现不佳是由于博弈构造成本高昂，以及求解博弈所需的时间较长。如果比较 ABV 和 SBV 在

性质规模方面的表现（图见附录 E.1），也会得到几乎相同的情况。在这种情况下，结果与理论相符（即 SBV 随规约规模增大而扩展性较差），但我们发现，SBV 的瓶颈并非 LTL 到 DPA 的转换（理论上，其开销呈指数级增长，远高于 ABV 所用的 LTL 到 NBA 的转换），而是构造和求解奇偶博弈。

我们指出，我们使用的 SBV 引擎 [8] 未经优化，且总是构造完整的（可达）博弈图。SBV 性能不佳的原因在于，最坏情况下，博弈的尺寸会随系统尺寸呈二次增长（考虑 $\forall^1\exists^1$ 性质时）。在稠密系统中（即转移较多的系统），这种情况会更加明显：随着转移密度的增加，奇偶博弈的尺寸会趋近其最坏情况尺寸（参见附录E）。相比之下，经过高度优化的包含性检查器（此处为`spot`）似乎几乎能以恒定时间完成包含性检查（尽管理论上其复杂度为指数级）。成熟的语言包含性检查器具备如此高的效率，使得`AutoHyper`能够实现出色的性能，往往优于有界模型检测（BMC）等符号化方法（参见第7节），并进一步彰显了命题1的实际影响。

9 结论

本文介绍了 AutoHyper，这是首个支持任意量词前缀的 HyperLTL 完备模型检查器。我们证明，AutoHyper 能够检查许多涉及量词交替的有趣性质，而且通常优于 BMC 等符号化方法，有时性能甚至相差数个数量级。我们认为，AutoHyper 最大的优势在于其一键式功能与完备性相结合：作为用户，无需担心 AutoHyper 是否适用于某项特定性质（这与 SBV 或 BMC 等工具不同），也无需提供提示（例如在 SBV 中给出显式策略）。

除了在一系列基准测试上评估 AutoHyper 的性能外，我们还使用 AutoHyper 来**(1)** 比较各种后端语言包含性检查器，**(2)** 探索超越一次量词交替的实际验证（理论所暗示的那样，这并非不可行），以及**(3)** 严格评估基于策略的验证在实践中的有效性和效率（与理论分析所暗示的不同，其表现不如基于自动机的方法）。

致谢。 本研究部分由德国科学基金会项目 389792660（清晰系统中心，TRR 248）以及 ERC HYPER 项目（编号 101055412）资助。R. Beutner 作为萨尔布吕肯计算机科学研究生院的一员开展了这项研究。

数据可用性声明

AutoHyper 和所有实验均可在 [4]获取。

参考文献

1. Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Kretínský, J., Müller, D., Parker, D., Strejcek：Hanoi 欧米伽自动机格式。载于：国际计算机辅助验证会议，CAV 2015。计算机科学讲义，第 9206 卷。施普林格（ 2015 10 1007 978 3 319 21690 4_31 ）。https://doi.org/ . / - - - - . Baier, C.,[2] Katoen, J.P.：模型检测原理。麻省理工学院出版社（2008）3. Barthe, G., D'Argenio, P.R., Rezk, T.：通过自组合实现安全信息流。《数学结构与计算机科学》21(6)（2011）。https://doi.org/10 0960129511000193 4 1017. /S . Beutner, R.：AutoHyper：HyperLTL 的显式状态模型检测（2023）。https://doi.org/10 7309986 5 5281. /zenodo. . Beutner, R., Carral, D., Finkbeiner, B., Hofmann, J., Krötzsch, M.：结合函数式规范的超性质判定。载于：ACM/IEEE计算机科学逻辑年会，LICS 2022。ACM（2022）。https://doi.org/10 3533369 6 1145 3531130 . / . . Beutner, R., Finkbeiner, B.：一种用于策略超性质的时序逻辑。载于：并发理论国际会议，CONCUR 2021。信息学论文集，第 203 卷。达格施图尔城堡（2021 24 7 10 4230 2021 ）。https://doi.org/ . /LIPIcs.CONCUR. . . Beutner, R., Finkbeiner, B.：HyperATL*：一种用于多智能体系统中超性质的逻辑。CoRR abs/2203.07283（2022）。https://doi.org/10 07283 8 48550 2203 . /arXiv. . . Beutner, R., Finkbeiner, B.：用于超属性验证的预言变量。载于：IEEE计算机安全基础研讨会，CSF 2022。IEEE（2022）。https://doi.org/10 01797 9 1109 54842 2022 00030, 2206 . /CSF . . https://arxiv.org/abs/ . . Beutner, R., Finkbeiner, B.：超越 k-安全性的超性质软件验证。载于：国际计算机辅助验证会议，CAV 2022。计算机科学讲义，第 13371 2022 10 1007 978 3 031 13185 1_17 卷。施普林格（）。https://doi.org/ . / - - - - . Biere, A., Cimatti, A., Clarke, E.M., Zhu, Y.：无需二元决策图的符号模型检测。载于：系统构造与分析工具和算法国际会议，TACAS 1999。计算机科学讲义，第 1579 卷。施普林格（1999）。https://doi.org/ 10 1007 3 540 49059 0_14 . / - - - . Brayton, R.K., Mishchenko, A.：ABC：一种学术界与工业界级别的验证工具。载于：国际计算机辅助验证会议，CAV 2010。计算机科学讲义，第 6174 2010 10 1007 978 3 642 14295 6_5 卷。施普林格（）。https://doi.org/ . / - - - - . Cimatti, A., Clarke, E.M., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., Sebastiani, R., Tacchella, A.：NuSMV 2：一种用于符号模型检测的开源工具。载于：国际计算机辅助验证会议，CAV 2002，哥本哈根。计算机科学讲义，第 2404 卷。施普林格（2002）。https://doi.org/10.1007/3- 540-45657-0 2913 . Clarke, E.M., Kroening, D., Ouaknine, J., Strichman, O.：有界模型检测的完备性与复杂度。载于：验证、模型检测与抽象解释国际会议，VMCAI 2004。计算机科学讲义，第 2937 卷。施普林格（ 2004 10 1007 978 3 540 24622 0_9 ）。https://doi.org/ . / - - - - . Clarkson, M.R., Finkbeiner, B., Koleini, M., Micinski, K.K., Rabe, M.N., Sánchez, C.：超性质的时序逻辑。载于：安全与信任原则国际会议，POST 2014。计算机科学讲义，第 8414 卷。施普林格（2014）。https://doi.org/ 10 1007 978 3 642 54792 8_15 . / - - - - . Clarkson, M.R., Schneider, F.B.：超性质。载于：IEEE计算机安全基础研讨会，CSF 2008。IEEE（2008）。https://doi.org/10.1109/CSF.2008.7

16. Clemente, L., Mayr, R.：非确定性自动机的高效化简及其在语言包含性测试中的应用。逻辑方法与计算机科学 15(1)（2019）。https://doi.org/10 2019 17 23638 15 1 12 . /LMCS- (:) . Coenen, N., Finkbeiner, B., Frenkel, H., Hahn, C., Metzger, N., Siber, J.：反应式系统中的时序因果关系。载于：国际自动化验证与分析技术研讨会，ATVA 2022。计算机科学讲义，第 13505 卷。施普林格（ 2022 10 1007 978 3 031 19992 9_13 ）。https://doi.org/ . / - - - - . Coenen, N., Finkbeiner, B., Sánchez, C., Tentrup, L.：验证超活性。载于：国际计算机辅助验证会议，CAV 2019。计算机科学讲义，第 11561 2019 10 1007 978 3 030 25540 4_7 卷。施普林格（）。https://doi.org/ . / - - - - . Copty, F., Fix, L., Fraer, R., Giunchiglia, E., Kamhi, G., Tacchella, A., Vardi, M.Y.：有界模型检测在工业环境中的优势。载于：国际计算机辅助验证会议，CAV 2001。计算机科学讲义，第 2102 卷。施普林格（ 2001 10 1007 3 540 44585 4_43 ）。https://doi.org/ . / - - - . Courcoubetis, C., Vardi, M.Y., Wolper, P., Yannakakis, M.：用于验证时序性质的内存高效算法。Formal Methods Syst. Des. 1 00121128 21 2 3 1992 10 1007 (/) ( )。https://doi.org/ . /BF . D' Argenio, P.R., Barthe, G., Biewer, S., Finkbeiner, B., Hermanns, H.：你的软件嗑药了吗？——对被暗中"增强"的程序进行形式化分析。载于：欧洲编程研讨会，ESOP 2017。计算机科学讲义，第 10201 卷。施普林格（ 2017 10 1007 978 3 662 54434 1_4 ）。https://doi.org/ . / - - - - . Doherty, S., Detlefs, D., Groves, L., Flood, C.H., Luchangco, V., Martin, P.A., Moir, M., Shavit, N., Jr., G.L.S.：双字比较并交换并非非阻塞算法设计的灵丹妙药。载于：ACM算法与体系结构并行性年度研讨会，SPAA 2004 1007945 23 2004 10 1145 1007912。ACM（）。https://doi.org/ . / . . Doveri, K., Ganty, P., Mazzocchi, N.：基于 FORQ 的语言包含形式化测试。载于：国际计算机辅助验证会议，CAV 2022。计算机科学讲义，第 13372 2022 10 1007 978 3 031 13188 2_6 卷。施普林格（）。https://doi.org/ . / - - - - . Doveri, K., Ganty, P., Parolini, F., Ranzato, F.：基于良拟序的 Büchi 自动机包含性测试。载于：并发理论国际会议，CONCUR 2021。信息学论文集，第 203 卷。达格施图尔城堡（2021）。https://doi.org/10 3 25 4230 2021 . /LIPIcs.CONCUR. . . Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Aisse, A.G., Schlehuber-Caissier, P., Medioni, T., Martin, A., Dubois, J., Gillard, C., Lauko, H.：从 Spot 2.0 到 Spot 2.10：有哪些新内容？载于：国际计算机辅助验证会议，CAV 2022。计算机科学讲义，第 13372 卷。施普林格（2022）。https://doi.org/ 10 1007 978 3 031 13188 2_9 . / - - - - . Fellner, A., Befrouei, M.T., Weissenbacher, G.：利用超性质进行变异测试。Softw. Syst. Model. 20 1 27 2 2021 10 1007 10270 020 00850 ( ) ( )。https://doi.org/ . /s - - - . Fienberg, S.E.：网络分析统计模型简史与尚待解决的挑战。计算与图形统计学杂志 21(4)（2012）28. Finkbeiner, B., Rabe, M.N., Sánchez, C.：HyperLTL 和 HyperCTL* 模型检测算法。载于：国际计算机辅助验证会议，CAV 2015。计算机科学讲义，第 9206 卷。施普林格（2015）。https://doi.org/10.1007/978-3-319-21690-4 329 . Herlihy, M., Wing, J.M.：线性化：并发对象的一种正确性条件。ACM Trans. Program. Lang. Syst. 12(3)（1990）。https://doi.org/10 78972 30 1145 78969. / . . Hsu, T., Sánchez, C., Bonakdarpour, B.：超性质的有界模型检测。载于：系统验证与分析工具和算法国际会议，TACAS 2021。计算机科学讲义，第 12651 卷。施普林格（2021）。https://doi.org/ . / - - - - .

系统构建与分析，TACAS 2021。计算机科学讲义，第 12651 卷。施普林格（2021）。https://doi.org/ 10 1007 978 3 030 72016 2_6 ./----. Jamroga, W., Mauw, S., Melissen, M.[31]：不可否认协议中的公平性。载于：国际安全与信任管理研讨会，STM 2011。计算机科学讲义，第 7170 2011 10 1007 978 3 642 29963 6_10 卷。施普林格[32]（）。https://doi.org/./----. Lamport, L.：Dijkstra 并发编程问题的一种新解法。ACM 通讯**17**361093 338197410114536108**2**( ) ( )。https://doi.org/./.. McCullough, D.：非干扰与安全性质的可组合性。载于：IEEE 安全与隐私研讨会，SP 1988。IEEE（1988）。https://doi.org/108110 3411091988./SECPRI... Piterman, N.：从非确定性 Büchi 和 Streett 自动机到确定性奇偶自动机。逻辑方法与计算机科学**3**(3)（2007）。https://doi.org/102007 352168335./LMCS-(:). Rabe, M.N.：一种用于信息流控制的时序逻辑方法。博士论文，萨尔大学（2016）36. Smith, G., Volpano, D.M.：多线程命令式语言中的安全信息流。载于：ACM SIGPLAN-SIGACT 编程语言原理研讨会，POPL 1998268975 371998101145268946。ACM（）。https://doi.org/./.. Tsai, M., Tsay, Y., Hwang, Y.：面向博弈、ω 自动机和逻辑的 GOAL。载于：国际计算机辅助验证会议，CAV 2013。计算机科学讲义，第 8044 2013 10 1007 978 3 642 39799 8_62 卷。施普林格[38]（）。https://doi.org/./----. Wang, Y., Nalluri, S., Pajic, M.：机器人学中的超性质：通过 HyperLTL 进行规划。载于：IEEE 国际机器人与自动化会议，ICRA 2020。IEEE（2020）。https://doi.org/10.1109/ICRA40945.2020.9196874

HyperLTL模型检查与语言包含

命题 1。 设 $\dot{\varphi} = \forall \pi_1 \ldots \forall \pi_n . \varphi$ 为一个 *HyperLTL* 公式（其中 φ 可能包含额外的轨迹量词），并设 $\mathcal{A}_\varphi$ 为一个定义在 Σ^n 上的自动机，且该自动机与 φ 满足 $\mathcal{T}$-等价。那么，当且仅当 $\mathcal{T} \models \dot{\varphi}$ （$\mathcal{A}^n_{\mathcal{T}}$） $\subseteq \mathcal{L}$（$\mathcal{A}_\varphi$）时， $\mathcal{L}$。

证明。 对于第一个方向，我们假设 $\mathcal{L}(\mathcal{A}^n_{\mathcal{T}}) \subseteq \mathcal{L}(\mathcal{A}_\varphi)$，并需要证明 $\mathcal{T} \models \dot{\varphi} = \forall \pi_1 \ldots \forall \pi_n . \varphi$。因此，任取 t_1、…、 $t_n \in$ 轨迹($\mathcal{T}$)。根据定义，有 $zip(t_1$、…、 $t_n) \in \mathcal{L}(\mathcal{A}^n_{\mathcal{T}})$，因此根据假设，$zip(t_1$、…、 $t_n) \in \mathcal{L}(\mathcal{A}_\varphi)$。由 $\mathcal{T}$-等价性可知 $\mathcal{A}_\varphi$，这意味着 $[\pi_1 \mapsto t_1, \cdots, \pi_n \mapsto t_n] \models_{Traces(\mathcal{T})} \varphi$，正如所要求的。

对于反向推导，我们假设 $\mathcal{T} \models \dot{\varphi}$，并需要证明 $\mathcal{L}(\mathcal{A}^n_{\mathcal{T}}) \subseteq \mathcal{L}(\mathcal{A}_\varphi)$。令 $t \in \mathcal{L}(\mathcal{A}^n_{\mathcal{T}})$ 为任意对象。根据 $\mathcal{A}_{\mathcal{T}}$ 的定义以及 zip，我们有 $t = zip(t_1, \ldots, t_n)$，其中某些 $t_1, \ldots, t_n \in$ 属于轨迹 ($\mathcal{T}$)。由于 $\mathcal{T} \models \dot{\varphi} = \forall \pi_1 \ldots \forall \pi_n . \varphi$，我们有 $[\pi_1 \mapsto t_1, \cdots, \pi_n \mapsto t_n] \models_{Traces(\mathcal{T})} \varphi$。根据 $\mathcal{T}$-等价性，这意味着 $t = zip(t_1, \ldots, t_n) \in \mathcal{L}(\mathcal{A}_\varphi)$，证毕。□

B 验证方法概述

近年来，人们开发了许多不同的方法来验证 HyperLTL 属性。下文将总结现有方法（可能会重复正文中的信息），并在第 B 节简要讨论它们各自的优点与缺点。

无交替 *HyperLTL* 与手动强化。 不含量词交替的 HyperLTL 属性可以通过检查系统自组合上的轨迹属性来规约 [3]，因此其模型检测复杂度与线性时序逻辑相同（相对于系统尺寸为 NL 完全，相对于公式尺寸为 PSPACE 完全）。这种 HyperLTL 验证方法已在 McHyper 中实现 [28]。通过使用 ABC [11] 作为后端求解器，McHyper 可用于含有数千个锁存器的电路。这远远超出了显式状态模型检测方法的能力范围。

虽然 McHyper 仅支持无交替公式，但 Finkbeiner 等人 [28]和 D'Argenio 等人 [21] 使用它来验证涉及量词交替的性质，方法是将这些性质手动强化为无交替公式。例如，可以通过将存在量化替换为全称量化来强化一个$\forall\exists$性质，从而得到一个 $\forall\forall$ 性质。必须手动论证这种强化的可靠性，且无法自动检查。此外，这种强化针对特定性质，更重要的是，也针对特定模型；因此，该方法不适用于通用 HyperLTL 模型检测。

基于策略的验证。 Coenen 等人 [18] 通过引入显式策略来实现存在量化，从而 $\forall^*\exists^*$ 验证属性。这种方法——我们称之为基于策略的验证（SBV）——有两种形式：策略由用户提供，或由系统综合。在前一种情况下，模型检测可直接简化为检查一个无交替公式（它

	可靠	AA	符号	PB	NL	完全
手动强化 [28,21]	◐	◐	●	○	●	○
SBV[显式策略] [18]	●	○	●	○	●	○
SBV[策略综合] [8,9]	●	○	◐	●	○	○
有界模型检测 [30]	●	●	●	●	○	○
显式自动机绑定验证 [本作品]	●	●	○	●	○	●

图 4：我们列出了现有的 HyperLTL 验证方法及其优缺点。表示“性质”成立，表示其不成立，而表示无法给出明确答案。缩写说明：**AA**= 任意交替，**符号** = 支持符号系统；**一键式** =一键式技术；**NL**=在系统尺寸方面非常高效（NL）的模型检测。

开销较低）。事实上，给定策略的模型检测与检查无交替 HyperLTL 一样复杂（在系统规模上为 NL完全），并且可以应用于规模相当大的系统。另一方面，提供显式策略需要对系统有深入了解。因此，与有界模型检测（例如 HyperQube）或显式状态模型检测（例如 AutoHyper）不同，给定策略的验证并非一键式技术，因此我们没有对其进行评估。在后一种情况下，该方法完全自动化，但需要开销更大的策略综合。理论上，采用自动策略综合的 SBV 在系统尺寸方面比 ABV 更高效：理论上，对 $\forall^*\exists^*$ HyperLTL 进行（基于自动机的）模型检测——如 AutoHyper 中的实现——在规约方面是 指数空间完全的，在系统尺寸方面是 PSPACE完全的 [14,35]。相反，SBV 在规约尺寸方面是 2-EXPTIME完全的，但在系统尺寸方面为 多项式时间 复杂度 [18]。然而，SBV 并不完备，因为用于解析存在量化轨迹的策略只能观察全称量化轨迹的有限前缀（例如，参见 [8]）。虽然近期有人提出用预言作为确保完备性的对策， [8]，但目前不存在高效的预言综合方法。现有的预言综合方法仅适用于具有 ≤ 20 个状态 [8]的系统。

有界模型检测。Hsu 等人 [30]提出了一种用于 HyperLTL 的有界模型检测（BMC）过程。与针对轨迹性质的有界模型检测类似（例如，参见[10]），该公式会展开至固定深度，而超出该深度的待处理义务则采用悲观方式（允许作为公式成立的见证）或乐观方式（允许作为公式不成立的见证）处理。针对轨迹性质的有界模型检测可归约为 SAT 求解，而针对超性质的有界模型检测则自然归约为 QBF 求解。该有界模型检测方法已在HyperQube中实现。与有界方法通常的情况一样，HyperLTL 的有界模型检测并不完备。例如，它永远无法证明系统满足形如 $\mathbb{Q}_1\pi_1.\ldots\mathbb{Q}_n\pi_n.\square\psi$ 的超性质，例如GNI。如果系统在固定步数后终止（即到达一个其唯一出边指向自身的状态），

该状态中的待处理义务便可得到精确评估（因为此后的整个执行过程已确定）。Hsu 等人 [30]利用这一观察提出了所谓的停机语义（分为乐观和悲观两种）。虽然这提高了有界模型检测的准确性（例如，如果系统的所有执行都在 k 步后终止，且展开界为某个 $k' \geq k$，那么停机语义就能验证 GNI），但大多数反应式系统本质上不会终止。表1 中考虑的所有示例都不会终止（反应式程序通常也是如此），因此停机语义并无优势——GNI在这些系统中均无法得到验证。

优点与缺点

图 4展示了各种方法相对优缺点的非正式但直观的概述。正如预期，所有方法都是可靠的（尽管对于手动强化，可靠性论证无法自动检查）。与有界模型检测和 ABV 不同，SBV 仅适用于 $\forall^*\exists^*$ 公式，无法处理任意的量词交替。显式状态模型检测最大的缺点是它不适用于符号系统（例如电路），并且需要预先将其转换为显式系统。尽管如此，本文表明，至少对于现有的基准测试而言，显式模型检测通过内部转换为显式状态表示，性能可与许多符号化方法相当，甚至优于它们。理论上，结合策略综合的 SBV 可通过求解符号奇偶博弈来适用于符号系统，但目前尚无工具支持这一功能（因此，）。手动强化和使用给定策略的 SBV 成本都很低（即，系统尺寸的 NL ），但需要投入大量人工工作，且往往需要领域知识。相比之下，结合策略综合的 SBV、有界模型检测和 ABV 理论上成本更高，但属于“一键式技术”，也就是说，它们无需用户手动输入。从性能来看，有界模型检测处于一个实用的折中位置：虽然效率不如无交替方法，但它可归约为量化布尔公式问题，而理论上该问题的求解速度可快于显式状态模型检测中遇到的自动机理论问题。正如第 7 节所示，这一点在实践中并未体现出来，而且 ABV 的性能往往可与有界模型检测相媲美。最后，研究 ABV（以 AutoHyper 的形式）的最大动因是其完备性；AutoHyper 是第一个能够以可靠且完备的方式验证任意 HyperLTL 属性的工具。

C HyperLTL 模型检查语言包含基准测试

详细比较 BAIT 和 RABIT后发现，在 HyperLTL 模型检测过程中产生的实例似乎明显更有利于 RABIT（见图 1）。这与 [24], 的观察结果形成对比；他们发现，在现有的大多数语言包含基准测试中，BAIT 的表现优于 RABIT。因此，验证 $\forall^*\exists^*$ HyperLTL 是扩展现有自动机包含基准测试集的自然选择（该领域的基准测试仍然稀少，参见例如 [24]）。

由此产生的基准测试似乎涵盖了与现有基准测试集不同的实例（从 BAIT 和 RABIT 在这两类基准测试上的表现截然不同可以看出这一点）。

将计划综合作为 HyperLTL模型检查

我们注意到，Wang 等人研究的路径规划问题 [38], 涉及计划的综合。如果我们希望综合出最短路径，就可以将该问题表述为公式 $\varphi = \exists\pi.\forall\pi'.\psi$ 其中 ψ 表示 π 编码了一条能够到达目标的路径，而所有其他路径 π' 至少需要与 π一样长。对于 π 而言，一个具体的见证轨迹就是该规划问题的最优解。在通过将 φ 规约为 SMT 查询（Wang 等人所采用的方法） [38]或量化布尔公式问题（Hsu 等人所采用的方法）时，我们可以轻松地从编码的可满足模型中提取具体解（该编码或者是 [30]SMT 公式，或者是 $\exists^*\forall^*$ QBF 公式）。使用 $\exists^*\forall^*$ AutoHyper中实现的基于自动机的方法来综合具体路径则不太直接。不过，我们认为，只要所使用的语言包含检查器能够提供反例（大多数求解器都满足这一点，例如RABIT、BAIT和FORKLIFT），我们实际上就可以使用该方法综合出具体解。

在 `AutoHyper`中，我们检查取反后的 $\neg\varphi = \forall\pi.\exists\pi'.\neg\psi$ 公式，因为这样可以利用语言包含性检查器。由于模型满足 φ (所有具有最短路径的模型都会满足 φ)，因此它违反了 $\neg\varphi$，所以根据命题 1，所检查的语言包含关系不成立。在对 t 进行模型检测时，能够证明这一非包含关系的轨迹 $\neg\varphi$ 性质因此包含在系统中，但没有见证轨迹；换言之， t 是 φ中存在量化的见证。因此，能够返回反例的语言包含检查器可用于最优路径规划，并且（如表3所示）优于基于 QBF 的方法。

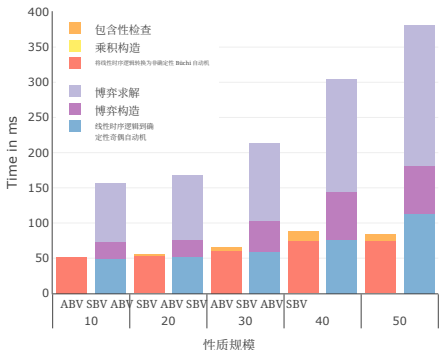
第8节的详细内容

E.1 SBV 的效率——规约大小的影响

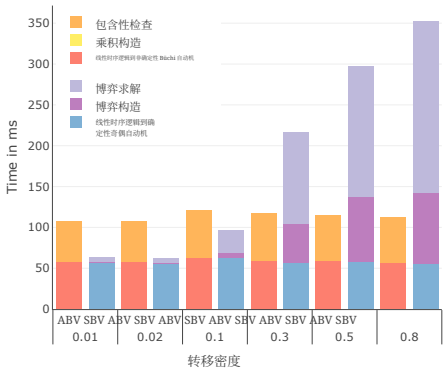
与图 3类似，图 5a比较了 ABV 和 SBV 的运行时间与性质规模之间的关系。尽管这些结果符合理论预测，即 SBV 随规约大小增长时的扩展性不如 ABV，但总运行时间的详细分解表明，性能下降的原因并非理论所指。理论上，限制因素应是成本高得多的 LTL 到 DPA 的转换（其成本呈指数级增长，远高于 ABV 中涉及的 LTL 到 NBA 的转换），但实际的性能膨胀源自奇偶博弈的构造和求解时间。

E.2 SBV 的效率——转移密度的影响

我们进行了与图3 和5a 中类似的实验，但不改变转移系统的尺寸，只改变其密度。图5b 展示了实验结果。



(a) ABV 和 SBV 在不同性质规模下的运行时间。我们将系统规模固定为 30。



(b) ABV 和 SBV 在不同转移密度下的运行时间。我们将系统尺寸固定为 30，性质规模固定为 20。

图 5：我们比较了 ABV 和 SBV 在性质规模（见图 5a）和转移密度（见图 5b）不同的实例上的表现。对于每种规模，我们生成 100 个随机 $\forall^1\exists^1$ 性质。超时时间设为 5 秒。我们对最快的 L 个实例取平均值，其中 L 是 ABV 和 SBV 在 5 秒超时时间内都能求解的实例数的最小值。

我们观察到，随着转移密度增加，SBV 的扩展性很差。这证实了 SBV 性能不佳是由博弈空间的尺寸造成的；随着转移密度增加，博弈空间的尺寸趋近于二次增长（相对于系统的尺寸）。为说明这一点，考虑一个有 n 个状态的系统，其中每个状态都有唯一的后继状态。由此构造的奇偶博弈对该系统进行二重自组合，得到 $\Theta(n)$ 个状态。另一极端是，如果每个状态都转移到所有其他状态，那么生成的博弈的尺寸为 $\Theta(n^2)$，因为所有状态对都会出现在乘积中。