arXiv:2303.14796v1 [cs.LO] 26 Mar 2023

基于自动机的超性质软件模型检测*

Bernd Finkbeiner[1], Hadar Frenkel[1], Jana Hofmann[2]**和 Janine Lohse[3]

[1] CISPA 亥姆霍兹信息安全中心，德国萨尔布吕肯¨Azure Research，微软，英国剑桥 [3] 萨尔兰大学，德国萨尔布吕肯

摘要。我们针对模理论的时序流逻辑（TSL）和超时序流逻辑（HyperTSL）开发了模型检测算法。TSL 在时序逻辑（LTL）的基础上扩展了存储单元、函数和谓词，使其成为一种便捷且富有表达力的逻辑，可用于推理软件及其他具有无限数据域的系统。HyperTSL 进一步扩展了 TSL，用于描述超属性——即关联多个系统执行的性质。因此，HyperTSL 可以表达软件系统中的非干扰等信息流策略。我们为 HyperTSL 增加理论，得到 HyperTSL(T)，并借鉴 LTL 软件验证的方法，为 TSL 和 HyperTSL(T) 构建模型检测算法。由此得到的算法是可靠的，但对于 HyperTSL(T) 的$\forall^*\exists^*$ 片段中的规约而言，必然是不完备的。我们的方法实现了首个针对带量词交替的时序超属性的软件模型检测算法，且不依赖有限状态抽象。

1 引言

超属性 [20] 将轨迹属性 [2] 推广为系统属性，即针对整个系统而非仅针对单个执行轨迹进行推理的属性。超属性涵盖了许多无法表示为轨迹属性的重要属性，例如信息流策略 [20], 、信息敏感性、网络物理系统的鲁棒性，以及分布式计算中的线性化 [11]。对于软件系统，典型的超属性包括程序细化，以及对称性等公平性条件。

为了规范超属性，线性时序逻辑 [48] (LTL) 扩展了轨迹量化，由此产生了超线性时序逻辑 [19] (HyperLTL)。目前已有多种用于 Hyper-LTL [19, 22, 35], 的模型检测算法，但这些算法是为有限状态系统设计的，因此不能直接用于软件。现有用于软件验证时序超属性的算法（例如 [1, 9]），除 [10], 之外，都仅适用于全称超属性，即不包含量词交替的属性。

* 本研究部分由欧洲研究委员会（ERC）HYPER 项目资助（编号：101055412）。
** 本研究工作开展于作者在 CISPA 亥姆霍兹信息安全中心任职期间。

本文提出了用于针对 $\forall^*\exists^*$ 超属性对软件系统进行模型检测的算法。我们的方法与近期提出的 [10]方法互为补充。该方法要求提供系统的有限状态抽象，并基于此既能证明也能证伪 $\forall^*\exists^*$ 超属性。我们不需要抽象，而是提供可靠但必然不完备的近似，以检测规范的反例。

超属性类 $\forall^*\exists^*$ 包含许多重要的超属性，例如程序精化或广义非干扰 [45]。广义非干扰指出，无法通过观察低安全级输出推断出高安全级输入的值。与非干扰不同，它不要求系统具有确定性。广义非干扰可以表示为 $\varphi_{gni} = \forall\pi\exists\pi'.(i_{\pi'} = \lambda \wedge c_\pi = c_{\pi'})$。该公式指出，将高安全级输入 i 的值替换为某个虚拟值 λ 不会改变可观察输出 c。

只有当 i 和 c 的取值范围是有限域时，才能用 HyperLTL 表达上述公式。在软件模型检测中，这确实是一个限制，因为变量的取值范围通常是无限域，例如整数或字符串。为克服这一限制，我们的规约基于超时序流逻辑（HyperTSL） [21]。HyperTSL 使用存储单元以及定义在这些单元上的谓词和更新项，取代 HyperLTL 的原子命题。更新项使用函数描述单元的值如何从前一个步骤变化到当前步骤。因此，这种逻辑特别适合用于规约软件属性。

HyperTSL 最初是为软件系统综合而设计的，因此其中所有谓词和函数都是未解释的。在模型检测的场景中，我们手头有一个具体系统，因此应根据该系统解释函数和谓词。因此，我们引入 Hy-perTSL(T)——带有解释理论的 HyperTSL——作为我们算法的基础。

概述 按照 [39], ，我们将系统表示为一个标记有程序语句的符号自动机。此类自动机的轨迹并非都对应有效的程序执行：例如，轨迹 assert(n_{η} 0)1; $n =$ 0^{ω}[4] ;

(assert ( )) 不可能是程序执行，因为第二个断言总是会失败。这样的轨迹称为不可行轨迹。相反，在可行轨迹中，所有断言理论上都可以成功。作为第一步，我们通过构造一个程序自动机来处理 TSL 模型检测（第 4 节），该自动机的可行接受轨迹对应于违反 TSL 规范的程序执行。为此，我们改造 [26], 的算法；该算法为线性时序逻辑构造这样的自动机，方法是结合给定的程序自动机与取反后的规范的自动机。

然后，我们通过应用自组合这一常用于验证超属性的技术，扩展该算法，以处理不含量词交替的 HyperTSL(T) 公式（第 5.1 节） [5, 6, 29]。

接下来，在第 5.2 节中，我们进一步扩展该算法，以寻找 $\forall^*\exists^*$-HyperTSL(T) 规范的反例（反过来，也可寻找 $\exists^*\forall^*$ 公式的见证）。我们构造一个自动机，对满足公式存在性部分的程序执行组合进行过近似。

[4] 上标 ω 表示程序语句的无限重复。

如果某个程序执行不包含在该过近似中，那么这次执行就是一个反例，证明该程序违反了规范。

更具体地说，对于 HyperTSL(T) 公式 $\forall^m \exists^n \psi$，我们构造 ψ 的自动机与程序自动机的 n 重自组合的乘积。该乘积中的每条可行轨迹都对应于为变量 π_1、...、π_n 选择一组执行，使得 ψ 得以满足。接下来，我们通过移除不可行轨迹来去除（部分）伪见证。我们考虑两类不可行性：k 不可行性，即在 k 个连续时间步内出现的轨迹局部不一致；以及非局部不可行性，它是自动机中某些不可行接受环导致的。在下一步中，我们将自动机投影到全称量化的轨迹上，从而得到满足公式存在部分的轨迹组合的过近似。最后，只需检查该过近似是否包含所有可行轨迹的组合。

最后，在第6节中，我们通过两个示例展示了我们的算法，其中包括广义非干扰。

贡献。我们提出了一种基于自动机的算法，用于对 $\forall^*\exists^*$超属性进行软件模型检测。我们的贡献总结如下。

– 我们扩展了 HyperTSL，使其包含理论，得到适用于模型检测的 HyperTSL 版本。– 我们将 [26] 的方法调整到 TSL(T) 和无交替 HyperTSL(T)，从而为 TSL(T) 和 HyperTSL(T) 首次提出模型检测算法。– 我们进一步扩展该算法，通过可行性分析来反驳 $\forall^*\exists^*$ 超属性并证明 $\exists^*\forall^*$ 超属性。

相关工作 时序流逻辑扩展了线性时序逻辑 [48]，最初是为综合而设计的 [33]。在综合方面，该逻辑已成功用于综合 FPGA 博弈“Syntroids” [37]，，以及综合智能合约 [32]。为推进智能合约综合，文献 [21] 将 TSL 扩展为 HyperTSL。上述工作采用的 TSL 版本将函数和谓词视为未解释的。尽管这一选择非常适合综合，但对于模型检测而言，使用所研究程序的解释更为合理。文献 [31]，为 TSL 加入了理论，并分析了该逻辑的可满足性问题。迄今为止，尚未研究过 TSL 或 HyperTSL 的模型检测（无论是否采用解释理论）。

对于线性时序逻辑，无限状态模型的模型检测问题已得到广泛研究，例如 [13, 16, 24, 26, 36]。我们的工作基于 [26] 提出的自动机式 LTL软件模型检测算法。还有各种用于验证程序上的全称超属性的算法，例如基于类型论的算法 [1, 9]。主要相关工作是 [10]„ 与我们的方法不同，该方法需要依赖谓词抽象，才能针对 $\forall^*\exists^*$ HyperLTL 规范对软件进行模型检测。它们还可以处理

异步超属性，这目前超出了我们的研究范围。另一种用于验证软件上 $\forall\exists$ 超属性的方案是 [50]。该方法使用广义约束 Horn 子句来验证函数式规范。此方法不适用于反应式、非终止程序。最近，还有人提出使用 TLA 的模型检测器（TLA 是一种能够表达软件系统及其属性的逻辑）来验证 $\forall^*\exists^*$ 超属性 [43]。

除了软件模型检测之外，针对各种系统模型和超属性类别的超属性验证也已得到研究。对于 ω-正则属性 [30,35] 、有限状态 Kripke 结构中的异步超属性 [7,12] ，以及定时系统 [41],、实值 [47] 和概率超属性 [3, 27, 28]，均已有模型检测算法（其中一些研究了上述属性的组合）。

2 预备知识

B ¨uchi 自动机是一个元组 $\mathcal{A} = (\Sigma, Q, \delta, q_0, F)$，其中 Σ 是有限字母表；Q 是状态集合； $\delta \subseteq Q \times \Sigma \times Q$ 是转移关系； $q_0 \in Q$ 是初始状态； $F \subseteq Q$ 是接受状态集合。B¨uchi 自动机$\mathcal{A}$ 在字 $\sigma \in \Sigma^\omega$ 上的一次运行，是状态的无限序列 $q_0\ q_1\ q_2 \cdots \in Q^\omega$，满足对所有 $i \in \mathbb{N}$，$(q_i, \sigma_i, q_{i+1}) \in \delta$。如果存在一条在 σ 上的运行，其中有无限多个 $i \in \mathbb{N}$ 满足 $q_i \in F$，则无限字 σ 被 $\mathcal{A}$ 接受。 $\mathcal{A}$ 的语言 $\mathcal{L}(\mathcal{A})$ 是被 $\mathcal{A}$ 接受的字的集合。

2.1 理论模上的时序流逻辑 TSL(T)

时序流逻辑（TSL） [33] 通过将布尔原子命题替换为关于存储单元和输入的谓词，以及用于指定单元值应如何变化的更新项，扩展了线性时序逻辑（LTL） [48]。

我们基于 [31], 的定义（该定义扩展了 [33]的定义），给出理论模上的 TSL 的形式化定义，即 TSL(T)。我们给出的定义源自 [44] ，与 [31]的定义略有不同；更新项的满足关系并非通过语法比较来定义，而是相对于单元和输入的当前值与先前值来定义。此定义适用于模型检测场景，因为在这种场景中给定了一个具体模型。

TSL(T) 基于值集 $\mathbb{V}$ 定义，其中包含 true、false $\in \mathbb{V}$，以及输入集 $\mathbb{I}$ 和存储单元集 $\mathbb{C}$。更新项和谓词根据给定的理论进行解释。理论是一个元组（ $\mathbb{F}$， ε），其中 $\mathbb{F}$ 是函数符号集； $\mathbb{F}_n$ 是元数为 n的函数集； ε : $(\bigcup_{n\in\mathbb{N}} \mathbb{F}_n \times \mathbb{V}^n)$ $\to\mathbb{V}$ 是解释函数，用于计算元数为 n的函数。就本文而言，我们假设每个理论（ $\mathcal{T}_\mathcal{F}$，ε）至少包含{=， ∨， ¬}，并采用它们通常的解释。

函数项 τ_F 由以下语法定义

$$\tau_F ::= c \mid i \mid f(\tau_F,\ \tau_F,\ \ldots\ \tau_F)$$

其中 $c \in \mathbb{C}$、$i \in \mathbb{I}$、$f \in \mathbb{F}$，且 f 中元素的数量与其元数相匹配。赋值 $a : (\mathbb{I} \cup \mathbb{C})$ $\to \mathbb{V}$ 是一个将值赋给输入和单元的函数。

我们用 A 表示所有赋值的集合。给定一个具体赋值，我们可以计算函数项的值。

求值函数 $\eta : \mathcal{T}_{\mathcal{F}} \times \mathsf{A} \to \mathbb{V}$ 定义如下

$$\begin{aligned} \eta(c, a) &= a(c) && \text{for } c \in \mathbb{C} \\ \eta(i, a) &= a(i) && \text{for } i \in \mathbb{I} \\ \eta(f\ (\tau_{F1}, \tau_{F2}, \ldots, \tau_{Fn}), a) &= \varepsilon(f, (\eta(\tau_{F1}), \eta(\tau_{F2}), \ldots, \eta(\tau_{Fn}))) && \text{for } f \in \mathbb{F} \end{aligned}$$

谓词项 τ_P 是一个只求值为 true 或 false 的函数项。我们用 $\mathcal{T}_P$表示所有谓词项的集合。

对于 $c \in \mathbb{C}$ 和 $\tau_F \in \mathcal{T}_{\mathcal{F}}$，$[\![c \leftarrowtail \tau_F]\!]$ 称为更新项。直观地说，更新项 $[\![c \leftarrowtail \tau_F]\!]$ 表示应将 c 更新为 τ_F的值。如果在前一个时间步中，τ_F 的求值结果为 $v \in \mathbb{V}$，那么在当前时间步中，c 应取值为 v。所有更新项的集合为 $\mathcal{T}_U$。TSL 公式按如下方式构造：对于 $c \in \mathbb{C}$、$\tau_P \in \mathcal{T}_P$、$\tau_F \in \mathcal{T}_{\mathcal{F}}$。

$$\varphi ::= \tau_P \mid [\![c \leftarrowtail \tau_F]\!] \mid \neg\varphi \mid \varphi \wedge \varphi \mid \bigcirc\varphi \mid \varphi\,\mathcal{U}\,\varphi$$

常用算子 $\vee$，（“最终”）和（“全局”）可通过以下等式导出 $\varphi \vee \psi = \neg(\neg\varphi \wedge \neg\psi)$ $\Diamond\varphi = \text{真}\ \mathcal{U}\,\varphi$ 和$\varphi = \neg\Diamond\neg\varphi$。 □

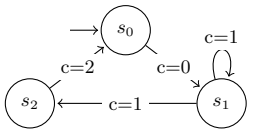
假设初始变量赋值固定为 ζ_{-1}（例如，将所有值设为零）。关于计算 $\zeta \in \mathsf{A}^{\omega}$ 和时间点 t，TSL(T)公式的满足关系定义如下，其中我们将 $\zeta \vDash \varphi$ 定义为 0，$\zeta \vDash \varphi$。

$$\begin{aligned} t, \zeta &\vDash \tau_P && \Leftrightarrow \eta(\tau_P, \zeta_t) = \mathit{true} \\ t, \zeta &\vDash [\![c \leftarrowtail \tau_F]\!] && \Leftrightarrow \eta(\tau_F, \zeta_{t-1}) = \zeta_t(c) \\ t, \zeta &\vDash \neg\varphi && \Leftrightarrow \neg(t, \zeta \vDash \varphi) \\ t, \zeta &\vDash \varphi \wedge \psi && \Leftrightarrow t, \zeta \vDash \varphi \text{ and } t, \zeta \vDash \psi \\ t, \zeta &\vDash \bigcirc\varphi && \Leftrightarrow t+1, \zeta \vDash \varphi \\ t, \zeta &\vDash \varphi\,\mathcal{U}\,\psi && \Leftrightarrow \exists t' \geq t.\ t', \zeta \vDash \psi \text{ and } \forall t \leq t'' < t'.\ t'', \zeta \vDash \varphi \end{aligned}$$

3 模理论 HyperTSL

本节介绍 HyperTSL(T)，即带有理论的 HyperTSL，它使我们能够根据所讨论的程序来解释谓词和函数。在 [21], 中介绍了 HyperTSL 的两个版本：HyperTSL 和 HyperTSL$_{rel}$。前者是 TSL 对超属性的保守扩展，也就是说，谓词只针对单条轨迹进行推理。在 HyperTSL$_{rel}$ 中，谓词可以关联多条轨迹，从而能够表达无限域中的非干扰等属性。本文以 HyperTSL$_{rel}$ 为基础，并进一步允许更新项跨多条轨迹取值。此外，我们还为原本未解释的函数和谓词扩展了基于理论的解释。我们将这种逻辑记为 HyperTSL(T)。

HyperTSL(T) 的语法与 TSL(T) 的语法相同，此外，现在每个单元和输入都分配给一个表示计算的轨迹变量。例如，c_π 现在指的是由轨迹 c_π 表示的计算中的存储单元 c



$\pi := (c = 0)\,(\ c = 1)^\omega,\ \pi' := ((c = 0)\,(\ c = 1)\,(\ c = 2))^\omega$

$\hat{a}_1 : \{c_\pi \mapsto 0, c_{\pi'} \mapsto 0\}, \hat{a}_2 : \{c_\pi \mapsto 1, c_{\pi'} \mapsto 1\}$

$\hat{a}_3 : \{c_\pi \mapsto 1, c_{\pi'} \mapsto 2\}, \hat{a}_4 : \{c_\pi \mapsto 1, c_{\pi'} \mapsto 0\}$

图 1。左：一个程序自动机。右：程序自动机的两条轨迹 π 和 π'。我们将每条轨迹解释为一次计算。同步执行两条轨迹时，每个时间点都有一个对应的超赋值，为 c_π 和 $c_{\pi'}$赋值。右侧展示了前四个时间步的超赋值。它们共同定义了超计算 $\hat{\zeta} := \hat{a}_1(\hat{a}_2\,\hat{a}_3\,\hat{a}_4)^\omega$，与 π 和 π'匹配。

。形式上，设 Π 为一组轨迹变量。我们将超函数项 $\hat{\tau}_F \in \hat{\mathcal{T}}_F$ 定义为一个函数项，其中 $(\ \mathbb{I} \times \Pi)$ 是输入集，$(\ \mathbb{C} \times \Pi)$ 是单元集。

定义 1。超函数项 $\hat{\tau}_F$ 由以下语法定义

$$\hat{\tau_F} ::= c_\pi \mid i_\pi \mid f(\hat{\tau_F},\ \hat{\tau_F},\ \ldots\ \hat{\tau_F})$$

其中 $c_\pi \in \mathbb{C} \times \Pi$、$i_\pi \in \mathbb{I} \times \Pi$、$f \in \mathbb{F}$，且元组 $\hat{}$ 中元素的数量与函数元数相匹配。我们用 $\mathcal{T}_F$ 表示所有超函数项的集合。

$\hat{}$ 类似地，我们将超谓词项 $\hat{\tau}_P \in \mathcal{T}_P$ 定义为取值为 true 或 false 的超函数项；将超赋值 $\hat{\mathrm{A}} = (\mathbb{I} \cup \mathbb{C}) \times \Pi \to \mathbb{V}$ 定义为把每条轨迹中的单元和输入映射到其当前值的函数；将超计算 $\hat{\zeta} \in \hat{\mathrm{A}}^\omega$ 定义为超赋值序列。示例见图 3。

定义 2。设 $c_\pi \in \mathbb{C} \times \Pi$、$\hat{\tau}_P \in \hat{\mathcal{T}}_P$、$\hat{\tau}_F \in \hat{\mathcal{T}}_F$。HyperTSL(T) 公式由以下语法定义：

$$\varphi ::= \psi \mid \forall\pi.\ \varphi \mid \exists\pi.\ \varphi$$
$$\psi ::= \hat{\tau_P} \mid [\![c_\pi \leftarrowtail \hat{\tau_F}]\!] \mid \neg\psi \mid \psi \wedge \psi \mid \bigcirc\psi \mid \psi\,\mathcal{U}\,\psi$$

为了定义 HyperTSL(T) 的语义，我们需要能够将超计算扩展到新的轨迹变量，每个路径量词对应一个变量。设 $\hat{\zeta} \in \hat{\mathrm{A}}^\omega$ 是一个超计算，且 π、$\pi' \in \Pi$、$\zeta \in \mathrm{A}^\omega$ 和 $x \in (\mathbb{I} \cup \mathbb{C})$。我们将使用计算 ζ，以 π 扩展 $\hat{\zeta}$ 定义为：对于 $\pi' \neq \pi$，$\hat{\zeta}[\pi, \zeta](x_{\pi'}) = \hat{\zeta}(x_{\pi'})$；对于 π，$\hat{\zeta}[\pi, \zeta](x_\pi) = \zeta(x_\pi)$。

定义 3。相对于超计算 $\hat{\zeta} \in \hat{\mathrm{A}}^\omega$、计算集合 Z 和时间点 t，HyperTSL(T) 公式的满足关系定义如下：

$$t, Z, \hat{\zeta} \vDash \forall\pi.\ \varphi \quad \Leftrightarrow \forall\zeta \in Z.\ t,\ Z,\ \hat{\zeta}[\pi, \zeta] \vDash \varphi$$
$$t, Z, \hat{\zeta} \vDash \exists\pi.\ \varphi \quad \Leftrightarrow \exists\zeta \in Z.\ t,\ Z,\ \hat{\zeta}[\pi, \zeta] \vDash \varphi$$

不涉及路径量化的情况与第 2.1 节中定义的 TSL(T) 情况类似。我们将 $Z \vDash \varphi$ 定义为 0、Z、$\varnothing^\omega \vDash \varphi$。

4 布希乘积程序与 TSL 模型检测

现在，我们将介绍如何将系统和规范建模为布chi自动机，并将 [26] 的自动机调整至 TSL 的设定。接着，我们介绍适用于 TSL(T) 的模型检测算法。在第 5.2 节中，我们将基于该算法提出一种用于 HyperTSL(T) 模型检测的算法。

我们使用系统的符号表示（例如，参见 [39]），其中转移以程序语句标记，且所有状态均为接受状态。

定义 4。设 $c \in \mathbb{C}$、$\tau_P \in \mathcal{T}_P$ 和 $\tau_F \in \mathcal{T}_{\mathcal{F}}$。我们将（基本）程序语句的集合定义为

$$s_0 ::= assert(\tau_P) \mid c := \tau_F \mid c := *$$
$$s ::= s_0 \mid s; s$$

我们将类型为 s_0 的语句称为基本程序语句，记作 Stmt_0；类型为 s 的语句记作 Stmt。赋值 $c := *$ 表示可以将任意值赋给 c。

程序自动机 $\mathcal{P}$ 是一个带有 $\Sigma = \text{Stmt}$ 的布希自动机，¨即 $\mathcal{P} = (\text{Stmt}\ Q, q_0, \delta, F)$ 且 $\delta \subseteq Q \times \text{Stmt} \times Q$。对系统建模时，我们只需要基本语句，因此有 $\text{Stmt} = \text{Stmt}_0$；并且 $F = Q$，因为所有状态均为接受状态。示意见图 3。

使用程序自动机，可以对 if 语句、while 循环和非确定性选择进行建模。不过，程序自动机的每条轨迹并不都对应一次程序执行。例如，轨迹 (n :=input$_1$); assert ($n > 0$); assert($n < 0$); assert ($true$)$^\omega$并不对应程序执行——第二个断言总是会失败。这样的轨迹称为不可行轨迹。如果一条轨迹对应一次程序执行，且其中所有断言都可能成功，我们就称该轨迹是可行的。下面对此作形式化定义。

定义 5. 如果计算 ζ 在时间点 t与轨迹 $\sigma \in \text{Stmt}_0^\omega$ 匹配，记作 $\zeta \triangleleft_t \sigma$，则满足以下条件:

$$\begin{aligned}
&\text{if } \sigma_t = assert(\tau_P): && \eta(\tau_P, \zeta_{t-1}) = true \quad and \quad \forall c \in \mathbb{C}.\ \zeta_t(c) = \zeta_{t-1}(c)\\
&\text{if } \sigma_t = c := \tau_F: && \eta(\tau_F, \zeta_{t-1}) = \zeta_t(c) \quad and \quad \forall c' \in \mathbb{C}\backslash\{c\}.\ \zeta_t(c') = \zeta_{t-1}(c')\\
&\text{if } \sigma_t = c := *: && \forall c \in \mathbb{C}\backslash\{c\}.\ \zeta_t(c) = \zeta_{t-1}(c)
\end{aligned}$$

其中，ζ_{-1} 是初始赋值。如果计算 ζ 与轨迹 $\sigma \in \text{Stmt}_0^\omega$ 匹配，记作 $\zeta \triangleleft \sigma$，则有 $\forall t \in \mathbb{N}.\ \zeta \triangleleft_t \sigma$。

定义 6. 如果程序自动机 $\mathcal{P}$ 满足关于 Stmt_0 的 TSL(T) 公式φ，则对其所有轨迹 σ，均有 $\forall \zeta \in \text{A}^\omega.\ \zeta \triangleleft \sigma \Rightarrow \zeta \vDash \varphi$。

现在，我们提出一种算法，用于检查程序自动机 $\mathcal{P}$ 是否满足 TSL(T) 公式。该算法改编自 [26]，提出的基于自动机的 LTL 软件模型检测方法，其基本思想是首先将

取反后的规范 φ 转换为自动机 $\mathcal{A}_{\neg\varphi}$，然后将 $\mathcal{A}_{\neg\varphi}$ 与 $\mathcal{P}$ combine 成一个新的自动机，即 Büchi 程序乘积。程序满足该规范，当且仅当 Büchi 程序乘积不接受任何可行轨迹。

在 [26], 中，B¨uchi 程序乘积的构造方式与标准自动机乘积构造类似。为确保结果仍是程序自动机，转移标签不是由 $(s, l) \in \text{Stmt}_0 \times 2^{AP}$ 这样的对组成，而是由程序语句 $(s; \text{assert}(l))$ 组成。于是，B¨uchi 程序乘积的一条可行接受轨迹对应于一个反例，证明该程序违反了规范。下文将讨论如何调整 TSL(T) 的 B¨uchi 程序乘积构造，使得这一性质——可行轨迹对应于反例——对于 TSL(T) 仍然成立。

设 φ 是一个 TSL(T) 规范。为了构造 $\mathcal{A}_{\neg\varphi}$，我们将所有更新项和谓词项都视为原子命题，从而得到一个 LTL 公式 $\neg\varphi_{LTL}$，并将其转换为一个 B¨uchi 自动机。[5] 对于我们的 B¨uchi 程序乘积版本，需要将来自 $\mathcal{P}$ 的转移标签 s 与来自 $\mathcal{A}_{\neg\varphi_{LTL}}$ 的转移标签 l 合并为一条程序语句，使得当且仅当 l 对语句 s 成立时，合并后语句的断言才成功。注意，l 是一个由更新项和谓词项组成的集合。对于更新项 $[\![c \leftarrowtail \tau_F]\!]$，我们不能简单地使用断言来检查它们是否为真，因为我们需要在语句 s 执行之前“保存”τ_F 的值。

我们的设定与 [26] 也不同，因为他们的程序语句不对输入流进行推理。我们通过使用全新的存储单元来模拟输入流的行为，这些存储单元在每个时间步都会被赋予一个新值。接下来，我们定义函数 combine，该函数将一个程序语句 s 与一个转移标签 l 组合成一个新的程序语句，如上所述。

定义 7。令 $\upsilon = \{[\![c_1 \leftarrowtail \tau_{F1}]\!], \ldots, [\![c_n \leftarrowtail \tau_{Fn}]\!]\}$ 为出现在 φ 中的更新项集合，令 ρ 为出现在 φ 中的谓词项集合。令 $l \subseteq (\upsilon \cup \rho)$ 为 $\mathcal{A}_{\neg\varphi}$ 的一个转移标签。令 $(tmp_j)_{j\in\mathbb{N}}$ 为一组全新的单元。令 $\mathbb{I} = \{i_1, \ldots i_m\}$。我们将函数 combine $: \text{Stmt} \times \mathcal{P}(\mathcal{T}_P \cup \mathcal{T}_U) \to \text{Stmt}$ 定义如下。combine(s, l) 的结果由 save values$_l$、s、new inputs、check preds$_l$ 和 check updates$_l$ 中的程序语句组成。因此，我们有：

$$save\_values := tmp_1 := \tau_{F1}; \ \ldots; tmp_n := \tau_{Fn}$$

$$new\_inputs := i_1 := *; \ \ldots \ ; i_m := *$$

$$check\_preds_l := assert\left(\bigwedge_{\tau_P \in l} \tau_P \wedge \bigwedge_{\tau_P \in \rho \setminus l} \neg\tau_P \right)$$

$$check\_updates_l := assert\left(\bigwedge_{[\![c_j \leftarrowtail \tau_{Fj}]\!] \in \upsilon} \begin{cases} c_j = tmp_j & \text{if } [\![c_j \leftarrowtail \tau_{Fj}]\!] \in l \\ c_j \neq tmp_j & \text{else} \end{cases} \right)$$

combine(s, l) := 保存值； s； 新输入； 检查谓词$_l$； 检查更新$_l$

我们可以将此定义扩展为组合轨迹，而不是单个转移标签。这将得到一个函数 combine $: \text{Stmt}^\omega \times \mathcal{P}(\mathcal{T}_P \cup \mathcal{T}_U)^\omega \to \text{Stmt}^\omega$。注意

[5] 关于将 LTL 公式转换为 B¨uchi 自动机的内容，请参见例如 [4,46,49]。

combine 的结果仍然是 Stmt 中的一个程序语句（或一条轨迹 Stmt$^\omega$），它作用于新的单元集合 $\mathbb{C} \cup \mathbb{I} \cup (tmp_j)_{j\in\mathbb{N}}$，我们将其称为 $\mathbb{C}^*$。

例1. 设 $\mathbb{I} = \{i\}$。那么，combine($n := 42, \{[\![n \leftarrowtail n + 7]\!], n > 0\}$) 的结果是 tmp$_0 := n + 7$; $n := 42$; $i := *$; 断言($n > 0$); 断言(n = tmp$_0$)。

由于 combine 会产生组合程序语句，我们现在需要将可行性的定义扩展到所有轨迹。为此，我们定义函数 flatten :Stmt$^\omega \to$ Stmt$_0{}^\omega$，它接收一个程序语句序列，并将其转换为基本程序语句序列，方法是将组合程序语句转换为多个基本程序语句。

定义8. 如果轨迹 $\sigma \in$ Stmt$^\omega$ 与计算 ζ 匹配，则记作 $\zeta \triangleleft \sigma$ if$\zeta \triangleleft$flatten(σ)。如果存在计算 ζ 使得 $\zeta \triangleleft \sigma$，则轨迹 σ 是可行的。

定义9.（组合积）设 $\mathcal{P} = (Stmt, Q, q_0, \delta, Q)$ 是一个程序自动机， $\mathcal{A} = (\mathcal{P}($ $\mathcal{T}_P \cup \mathcal{T}_U)$ $Q', q_0', \delta', F\prime)$ 是一个 B ¨uchi 自动机（例如，自动机 $\mathcal{A}_{\neg\varphi_{LTL}}$）。组合积 $\mathcal{P} \otimes \mathcal{A}$ 是一个自动机$\mathcal{B} = (Stmt, Q \times Q', (q_0, q_0')$ $\delta_B, F_B)$，其中

$$F_B = \{(q, q') \mid q \in Q \wedge q' \in F'\}$$
$$\delta_B = \{((p, q), combine(s, l), (p', q')) \mid (p, s, p') \in \delta \wedge (q, l, q') \in \delta'\}$$

定理 1. 设 $\mathcal{P}$ 是定义在 Stmt$_0$ 上的程序自动机。设 φ 是一个 TSL(T)公式。那么， $\mathcal{P}$ 满足 φ 当且仅当 $\mathcal{P} \otimes \mathcal{A}_{\neg\varphi_{LTL}}$ 没有可行轨迹。

˜证明（概要）。如果 $\zeta \triangleleft \sigma$ 是一个反例，我们可以构造一个计算 ζ，它与 $\mathcal{P} \otimes \mathcal{A}_{\neg\varphi_{LTL}}$ 中对应的组合轨迹相匹配，反之亦然。形式化构造见附录 A.4。

现在，我们可以应用定理 1 来解决模型检测问题：借助 [26] 中的可行性检查作为黑盒，检验 $\mathcal{P}\otimes\mathcal{A}_{\neg\varphi_{LTL}}$ 是否不接受任何可行轨迹。 [26] 的算法基于反例引导的抽象精化（CEGAR [18]）。我们会检查被接受的轨迹是否可行。首先，使用 SMT 求解器检查轨迹的有限前缀。如果这些前缀可行，则使用排序函数综合器检查整条轨迹最终是否会终止。如果轨迹可行，它就构成一个反例。否则，就对自动机进行精化，使其不再包含这个虚假的反例轨迹，然后重复这一过程。更多详情请参见 [26]。SMT 求解器和排序函数综合器的局限性，也限制了程序和 TSL(T)公式中可使用的函数和谓词。

5 HyperTSL(T)模型检测

现在我们来讨论 HyperTSL(T) 的模型检测问题。我们先处理无交替公式，再继续讨论 $\forall^*\exists^*$ 公式。

5.1 无交替 HyperTSL(T)

本节中，我们将自组合技术应用于第 4 节的算法，将其扩展到无交替 HyperTSL(T)。首先，我们定义程序自动机满足 HyperTSL(T) 公式的含义。

定义 10. 设 $\mathcal{P}$ 是一个定义在 Stmt_0，上的程序自动机，φ 是一个 Hyper-TSL(T)公式，并设 $Z = \{\zeta \in \text{A}^\omega \mid \exists \sigma.\ \zeta \triangleleft \sigma$ 且 σ 是 $\mathcal{P}$的一条轨迹。若 $Z \vDash \varphi$，则称 $\mathcal{P}$ 满足 φ。

定义 11. 设 $\mathcal{P} = (\text{Stmt}\ Q, q_0, \delta, Q)$ 是一个程序自动机。$\mathcal{P}$的 n 重自组合为 $\mathcal{P}^n = (\text{Stmt}', Q^n, q_0^n, \delta^n, Q^n)$，其中 Stmt'是定义在输入集 $\mathbb{I} \times \mathit{\Pi}$ 和单元集 $\mathbb{C} \times \mathit{\Pi}$ 上的程序语句，并且其中$Q^n = Q \times \cdots \times Q$, $q_0^n = (q_0, \ldots, q_0)$ 和

$$\begin{aligned}\delta^n = \{&((q_1, \ldots, q_n), ((s_1)_{\pi_1}; \ldots; (s_n)_{\pi_n}), (q_1', \ldots, q_n')) \\ &\mid \forall 1 \le i \le n.\ (q_i, s_i, q_i') \in \delta\}\end{aligned}$$

其中，$(s)_\pi$ 将 s 中使用的每个单元 c 重命名为 c_π，并将每个输入 i 重命名为 i_π。

定理 2. 程序自动机 $\mathcal{P}$ 在 Stmt_0 上满足通用 Hyper-TSL(T) 公式 $\varphi = \forall \pi_1. \ldots \forall \pi_n.\ \psi$，当且仅当 $\mathcal{P}^n \otimes \mathcal{A}_{\neg \psi_{LTL}}$ 没有可行轨迹。

定理 3. 程序自动机 P 在 Stmt_0 上满足存在型 HyperTSL(T) 公式 $\varphi = \exists \pi_1. \ldots \exists \pi_n.\ \psi$，当且仅当 $\mathcal{P}^n \otimes \mathcal{A}_{\psi_{LTL}}$ 有一条可行轨迹。

这些命题的证明与定理 1 的证明类似，详见附录 A.5。

5.2 $\forall^*\exists^*$HyperTSL(T)

本节将介绍一种可靠但必然不完备的算法，用于寻找 $\forall^*\exists^*$ HyperTSL(T) 公式的反例。[6] 此类算法也可以为公式提供见证 $\exists^*\forall^*$。由于 HyperTSL(T) 构建于 HyperLTL 之上，我们结合有限状态 HyperLTL 模型检查的思路 [35] 与第 4 节和第 5.1 节中的算法。

令 $\varphi = \forall^m \exists^n . \psi$。对于 HyperLTL 模型检查，[35] 首先构造一个自动机，其中包含满足 $\psi_\exists := \exists^n . \psi$ 的系统轨迹，然后通过取补来提取针对 $\forall\exists$ 规范的反例。考虑第 4 节中的自动机 $\mathcal{P}^n \otimes \mathcal{A}_{\psi_{LTL}}$，其可行轨迹对应于满足 $\psi_\exists$ 的系统轨迹。如果我们能够移除所有不可行轨迹，就可以应用有限状态 HyperLTL 模型检查的构造方法。不幸的是，一般情况下不可能移除所有不可行部分，因为这样得到的有限状态系统将恰好描述一个无限状态系统。因此，本节的主要思路是从不可行轨迹中移除部分内容

[6] 请注意，由于可行性测试，第 4 节和第 5.1 节中的算法也并不完备。不过，本节所提供算法的不完备性源于公式中的量词交替。

$\mathcal{P}^n \otimes \mathcal{A}_{\psi_{LTL}}$，从而构造出满足 $\psi_\exists$ 的系统轨迹的过近似。若要证伪 φ，反例就是一组不包含在该过近似中的系统轨迹。

我们提出了两种消除不可行性的方法。第一种方法从自动机中移除 k-不可行性，即轨迹中的局部不一致，这种不一致发生在连续 k 个时间步内。在选择 k 时，需要权衡：k 越大，能够识别出的反例就越多，但自动机构造的规模也会呈指数级增长。

第二种方法从自动机中移除不可行的接受环。可能无法将它们全部移除，因此我们会限制迭代次数。我们先给出一个例子，然后详细阐述这两种方法。

例 2。下面的轨迹 t_1 是 3-不可行的，因为无论第二个时间步之前 n 的值是多少，第四个时间步的断言都会失败。

$$t_1 = (n--;\ assert(n >= 0))\ (n := 1;\ assert(n >= 0))\ (n--;\ assert(n >= 0))^\omega$$

相比之下，对于任何 k，轨迹 $t_2 = (n := *)\ (\ n--;$ assert $(\ n >= 0))^\omega$ 都不是 k-不可行的，因为 n 的值总可以足够大，从而通过前 k 个断言。尽管如此，该轨迹仍然不可行，因为 n 不可能永远不减小而又不降到零以下。如果自动机接受了这样的轨迹，那么 $n--;$ assert $(\ n >= 0)$ 就对应一个不可行的接受环。

消除 *k*-不可行性 为了从自动机中消除 k-不可行性，我们构造一个新的程序自动机，用来“记住” $k-1$ 之前的语句。新自动机的状态对应于原自动机中长度为 k 的路径。如果我们可以用 l 扩展 p 所表示的轨迹，使所得轨迹满足 k-可行性，那么就在两个状态 p 和 q 之间添加一条标记为 l 的转移。形式化地，我们得到：

定义 12. 设 $k \in \mathbb{N}$, $\sigma \in$ Stmt$^\omega$。如果存在 $j \in \mathbb{N}$ 使得 $\sigma_j\sigma_{j+1}\cdots\sigma_{j+k-1}$，则称 σ 是 k-不可行的；对于所有可能的初始赋值 ζ_{-1}，assert($true$)$^\omega$ 均不可行。我们也称子序列 $\sigma_j\sigma_{j+1}\cdots\sigma_{j+k-1}$不可行。如果一条轨迹不是 k-不可行的，则称其为 k-可行。[7]

定义 13. 设 $\mathcal{P} =$ (Stmt Q, q_0, δ, F) 是一个程序自动机。设 $k \in \mathbb{N}$。我们将 $\mathcal{P}$ 定义为没有 k-不可行性的 $\mathcal{P}_k =$ (Stmt Q', q_0, δ', F')，其中

$$\begin{aligned} Q' :=& \{(q_1, s_1, q_2 \ldots, s_{k-1}, q_k) \mid (q_1, s_1, q_2) \in \delta \wedge \cdots \wedge (q_{k-1}, s_{k-1}, q_k) \in \delta\} \cup \\ & \{(q_0, s_0, q_1 \ldots, s_{k'-1}, q_{k'}) \mid k' < k-1 \wedge (q_0, s_0, q_1) \in \delta \wedge \ldots \\ & \qquad \wedge (q_{k'-1}, s_{k'-1}, q_{k'}) \in \delta\} \\ \delta' :=& \{((q_1, s_1, q_2 \ldots, s_{k-1}, q_k), s_k, (q_2, s_2, \ldots, q_k, s_k, q_{k+1})) \in Q' \times Stmt \times Q' \\ & \mid s_1 \ldots s_k\ feasible\} \cup \end{aligned}$$

[7] 子序列 $\sigma_j\sigma_{j+1}\ldots\sigma_{j+k-1}$是否是 k-不可行性的见证，可以使用 SMT 求解器进行检查，例如 [14, 15, 17, 25]。

$$\{((q_0, s_0, q_1 \ldots, s_{k'-1}, q_{k'}), s_{k'}, (q_0, s_0, \ldots, q_{k'}, s_{k'}, q_{k'+1})) \in Q' \times Stmt \times Q' \mid k' < k-1 \wedge s_0 \ldots s_{k'} \ feasible\}$$

$$F' := \{(q_1, s_1, q_2 \ldots, s_{k-1}, q_k) \in Q' \mid q_k \in F\} \cup \{(q_0, s_0, q_1 \ldots, s_{k'-1}, q_{k'}) \in Q' \mid k' < k-1 \wedge q_{k'} \in F\}$$

定理 4. $\mathcal{P}_k$ 恰好接受 $\mathcal{P}$ 的 k-可行轨迹。

证明可直接由上述构造得出，详情请参见附录 A.3。

移除不可行接受环 为了移除不可行接受环，我们首先枚举自动机的所有简单环（例如使用 [42]），也包括由自环诱导的环。对于每个至少包含一个接受状态的环 ϱ，我们检验其可行性：首先，使用 SMT 求解器检验 ϱ 是否局部不可行；然后，使用排序函数综合器（例如 [8,23,38]）检验 ϱ^ω 是否不可行。如果我们成功证明其不可行，就使用 [39,40] 中的方法细化模型。下面将形式化这一细化过程。

定义 14. 设 $\mathcal{P} = (\text{Stmt } Q, q_0, \delta, F)$ 是一个程序自动机。设 $\varrho = (q_1, s_1, q_2)(q_2, s_2, q_3) \ldots (q_n, s_n, q_1)$ 是 $\mathcal{P}$ 的一系列转移。若存在一个 $1 \le j \le n$，使得 $q_j \in F$ 且 $(s_1 s_2 \ldots s_{n-1})^\omega$ 对所有可能的初始赋值 ζ_{-1} 都不可行，则称 ϱ 是一个不可行接受环。

定义 15. 令 $\mathcal{P}$ 为一个程序自动机，$C \subseteq (Q \times \text{Stmt} \times Q)^\omega$ 为 $\mathcal{P}$ 的一组不可行接受环。此外，令

$$\varrho = (q_1, s_1, q_2)(q_2, s_2, q_3) \ldots (q_{n-1}, s_{n-1}, q_n) \in C.$$

自动机 $\mathcal{A}_\varrho$ 对应于 ϱ，记作 $\mathcal{A}_\varrho = (\text{Stmt } Q = \{q_0, q_1, \ldots q_n\} \ q_0, \delta, Q \setminus \{q_0\})$，其中

$$\delta = \{(q_0, s, q_0) \mid s \in Stmt\} \cup \{(q_j, s_j, q_{j+1}) \mid 1 \le j < n\} \cup \{(q_0, s_1, q_2), (q_n, s_n, q_1)\}.$$

于是，$\mathcal{A}_\varrho$ 恰好接受以 ϱ^ω 结尾的轨迹，而对前缀没有任何限制。示例见图 2。为了从 $\mathcal{P}$ 中排除 $\mathcal{A}_\varrho$ 的轨迹，我们定义 $\mathcal{P}_C := \mathcal{P} \setminus (\bigcup_{\varrho \in C} \mathcal{A}_\varrho)$。[8] 可以重复此构造，以排除在 $\mathcal{P}_C$ 中新产生的不可行接受环。我们将迭代此过程 k' 次的结果记为 $\mathcal{P}_{C(k')}$。

查找 $\forall^*\exists^*$ HyperTSL(T) 公式的反例 现在考虑一个 HyperTSL(T) 公式 $\varphi = \forall^{1 \cdots m} \exists^{m+1 \cdots n}.\psi$ 和一个程序自动机 $\mathcal{P}$。

[8] 对于两个自动机 $\mathcal{A}_1$、$\mathcal{A}_2$，我们用 $\mathcal{A}_1 \setminus \mathcal{A}_2$ 表示 $\mathcal{A}_1$ 与 $\mathcal{A}_2$ 的补集的交集，其语言为 $\mathcal{L}(\mathcal{A}_1) \setminus \mathcal{L}(\mathcal{A}_2)$。

为了查找反例，我们首先构造组合积 $\mathcal{P}^n \otimes \mathcal{A}_{\psi}$。$\mathcal{P}^n \otimes \mathcal{A}_{\psi}$ 的每条可行接受轨迹都对应于满足 ψ的 n 条可行程序轨迹的组合。接下来，我们消除 k-不可行性，并移除 k'-次的...

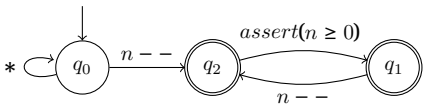


图 2. 针对不可行环 $\varrho = q_1\ n--\ q_2\ q_2\ assert\ n > (,,)(,()^0 q_1)$ 的自动机 $\mathcal{A}^*_{\varrho}$。标签表示每条相关语句对应的一条边。

来自组合积的可行接受环，从而得到自动机 $(\mathcal{P}^n \otimes \mathcal{A}_{\psi})_{k,C(k')}$。使用这个修改后的组合积，我们得到满足规范存在部分的程序执行组合的一个过近似。组合积的每条轨迹都是n 程序执行与谓词/更新项序列的组合。随后，我们从一条可行轨迹中投影出 m 全称量化的程序执行，得到满足公式存在部分的程序执行元组。将此投影应用于 $(\mathcal{P}^n \otimes \mathcal{A}_{\psi})_{k,C(k')}$ 的所有轨迹，可得满足规范存在部分的程序执行的一个过近似。形式化地：

定义16。设 $\mathcal{P}$ 为一个程序自动机，设 $m \leq n \in \mathbb{N}$，并设 $\mathcal{A}_{\psi}$ 为公式 ψ对应的自动机。令 $(\mathcal{P}^n \otimes \mathcal{A})_{k,C(k')} = (\text{Stmt}\ Q, q_0, \delta, F)$。我们定义投影自动机 $(\mathcal{P}^m \otimes \mathcal{A})^{\forall}_{k,C(k')} = (\text{Stmt}\ Q, q_0, \delta^{\forall}, F)$，其中$\delta^{\forall} = \{(q, (s_1; \ldots; s_m)\ q') \mid \exists s_{m+1}, \ldots s_n, l.\, (q, \text{combine}(s_1; \ldots; s_n, l)\ q') \in \delta\}$。记号 $s_1; s_2$ 表示一个语句序列，如定义 4 所述。关于全称投影的更多细节，请参阅文献 [34]。

现在，只需检查这个过近似是否包含 m 所有可行的程序执行元组。若不包含，则可找到一个反例。这归结为检验 $\mathcal{P}^m \backslash (\mathcal{P}^n \otimes \mathcal{A}_{\psi})^{\forall}_{k,C(k')}$ 是否存在可行轨迹。定理 5 说明了我们算法的可靠性。证明见附录 A.6。

定理 5. 设 $\varphi = \forall^{1\cdots m}\exists^{m+1\cdots n}.\psi$ 是一个 HyperTSL(T) 公式。如果自动机 $\mathcal{P}^m \backslash (\mathcal{P}^n \otimes \mathcal{A}_{\psi})^{\forall}_{k,C(k')}$ 存在一条可行轨迹，那么 $\mathcal{P}$ 不满足 φ。

6 算法演示

本节中，我们将第 5.2 节的算法应用于两个简单示例，展示只需消除部分不可行性，就可能足以识别反例。

广义非干扰 回顾公式 $\varphi_{gni} = \forall\pi.\ \exists\pi'.\ (i_{\pi'} = \lambda \wedge c_{\pi} = c_{\pi'})$，该公式在第 1 节中引入，用于描述广义非干扰。我们在φ_{gni} 程序自动机 $\mathcal{P}$ 上进行模型检测，该自动机如图 3（左）所示，并设置 $\lambda = 0$。程序 $\mathcal{P}$ 违反了φ_{gni}，因为对于轨迹 $(\ assert\ (i < 0)\ \ c := 0)^{\omega}$，不存在另一条轨迹使得其上的 c 相等，但 $i = 0$。

$\psi = (i_{\pi'} = 0 \wedge c_{\pi} = c_{\pi'})$的自动机由一个接受状态组成，该状态带有标记为 $\tau_P = (i_{\pi'} = 0 \wedge c_{\pi} = c_{\pi'})$ 的自环。对于这个示例，只需选择 $k = 1$。为了检测 1-不一致性，我们构造 $\mathcal{P}^2$（见图 3 右侧）。然后，$(\mathcal{P}^2 \otimes \mathcal{A}_{\psi})_k$ 是移除了所有 1-不一致转移后的组合积

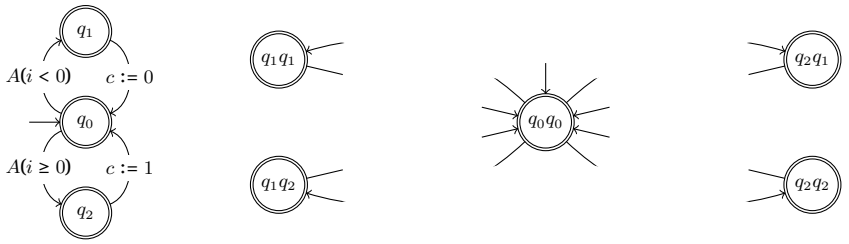


图 3。左：第一个示例中使用的程序自动机 $\mathcal{P}$。右：程序自动机 $\mathcal{P}^2$。为简洁起见，我们用 A 表示 $assert$，并合并连续的断言。

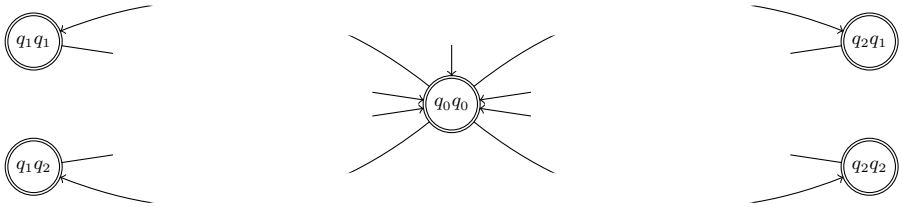


图 5。组合积（ $\mathcal{P}^2 \otimes \mathcal{A}_\psi$）

（组合积见图 5）。

自动机 $(\mathcal{P}^2 \otimes \mathcal{A}_\psi)^\forall_k$ 如图 4 所示。它不包含轨迹 σ = assert ($i < 0$) ($c := 0)^\omega$，而该轨迹是 $\mathcal{P}$ 的一条可行轨迹。因此， σ 是一条可行轨迹，被 $\mathcal{P}\backslash(\mathcal{P}^2 \otimes \mathcal{A}_\psi)^\forall_k$ 接受，并且是一个反例，证明 $\mathcal{P}$ 不满足

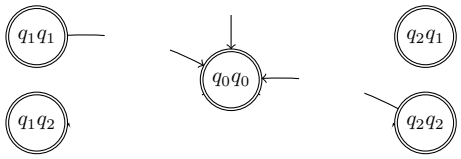


图 4。程序自动机（ $\mathcal{P}^2 \otimes \mathcal{A}_\psi$）$^\forall_k$

广义非干扰——不存在这样的可行轨迹：它在单元 c 的值上相同，但始终具有 $i = 0$。

移除循环的必要性 现在我们给出一个例子，其中移除 k-不可行性并不足够，但移除不可行接受环则会产生一个反例。考虑规范 $\varphi = \forall\pi\exists\pi'.\ (p_\pi \neq p_{\pi'} \wedge n_\pi < n_{\pi'})$ 和图 6 中的程序自动机 $\mathcal{P}_{cy}$。公式 φ 表明，对于每条轨迹 π，都存在另一条轨迹 π'，它在 p 上与 π 不同，但其中n 始终更大。轨迹 π = (n := ∗); (p := ∗); assert($p = 0$); (n − −)$^\omega$ 是 $\mathcal{P}_{cy}$中 φ的一个反例，因为任何在 p 上不同的轨迹 π'都会在每个时间步使其 n 减少 2，因此 $n_{\pi'}$ 最终会低于 n_π。

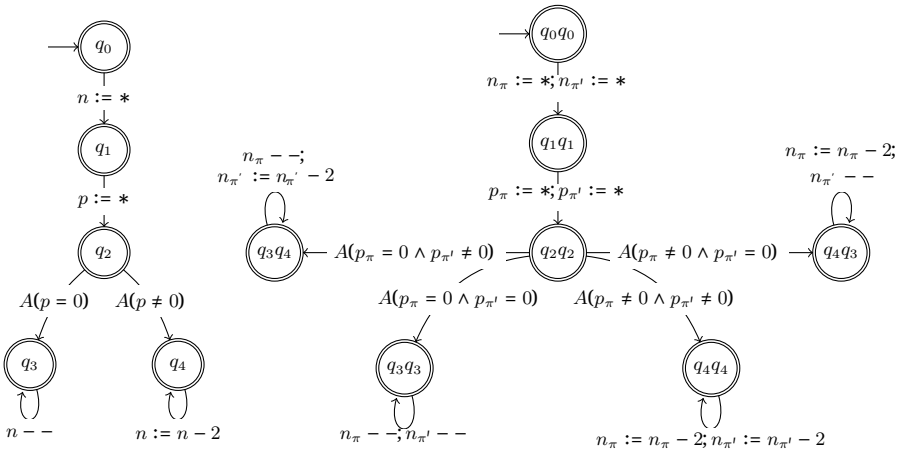


图 6. 左：程序自动机 $\mathcal{P}_{cy}$；右：程序自动机 $\mathcal{P}^2_{cy}$。

自动机 $\mathcal{P}^2_{cy}$ 如图 6 所示。在组合积中，自动机的结构保持不变，并且每个状态都添加了 assert($p_\pi \neq p_{\pi'} \wedge n_\pi < n'_\pi$)。在这里，移除局部 k-不可行性并不足够；假设 $k = 1$。唯一的 1-不可行转移是从 q_2q_2 到 q_3q_3的转移，而这并不能消除反例 π。更大的 k也不起作用，因为组合积中剩余的轨迹对于任何 k 都不是 k 不可行的。

然而， q_3q_4 处的自环是一个不可行接受循环——序列（$n_\pi - -; n_{\pi'} := n_{\pi'} - 2$; 断言 ($n_\pi < n_{\pi'}$)）$^\omega$ 最终必须终止。我们选择 $k' = 1$ 移除所有以该循环结束的轨迹。接下来，我们将自动机投影到全称部分。自动机（ $\mathcal{P}^2 \otimes \mathcal{A}_\psi$）$^{\forall}_{1}$，$C(1)$不接受轨迹 π。但由于 π 位于 $\mathcal{P}$ 中且可行，因此被认定为反例。

7 结论

我们为 HyperTSL 扩展了理论，得到 HyperTSL(T)，并为 TSL(T) 和 HyperTSL(T) 首次提供了无限状态模型检测算法。由于这是首项研究（Hyper）TSL 模型检测的工作，这些也是首批用于（Hyper）TSL 有限状态模型检测的算法。对于 TSL(T)，我们将已知的适用于线性时序逻辑的软件模型检测算法调整到了 TSL(T) 场景。随后，我们利用自组合技术，将该算法推广到 HyperTSL(T) 的交替无关片段。

此外，我们还介绍了一种可靠但必然不完备的算法，用于查找 $\forall^*\exists^*$-HyperTSL(T) 公式的反例（以及证明 $\exists^*\forall^*$ 公式的见证）。该算法能够找到违反广义非干扰等性质的程序执行，而这类性质只有结合全称量词和存在量词才能表达。

寻找 HyperTSL(T) 其他片段的模型检测算法并实现我们的方法，仍有待未来工作完成。

参考文献

1. Alejandro Aguirre、Gilles Barthe、Marco Gaboardi、Deepak Garg 和 Pierre-Yves Strub。高阶程序的关系逻辑。 *Proc. ACM Program. Lang.*, 1 21 1 21 (ICFP): : – :29, 2017。2. B. Alpern 和 F.B. Schneider。定义活性。 *Information Processing Letters*, 第 181–185 页, 1985。3. Shiraj Arora、Ren´e Rydhof Hansen、Kim Guldstrand Larsen、Axel Legay 和 Danny Bøgsted Poulsen。实值信号超属性的统计模型检测。载于 Owolabi Legunsen 和 Grigore Rosu 编, *ModelChecking Software - 28th International Symposium, SPIN 2022, Virtual Event,May 21*, 2022, *Proceedings, Lecture Notes in Computer Science* 第 13255 卷, 第 61–78 页。Springer，2022。4. Tom´as Babiak、Mojm´ır Kret´ınsk´y、Vojtech Reh´ak 和 Jan Strejcek。将线性时序逻辑转换为 b¨uchi 自动机：快速且更具确定性。载于 Cormac Flanagan 和 Bar- bara K¨onig 编, *Tools and Algorithms for the Construction and Analysis of Systems - 18th International Conference, TACAS 2012, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012, Tallinn, Estonia March 24 - April 1 2012. Proceedings* 7214 *Lecture Notes in* , , , 第 卷 , 第 – *Computer Science* 95 109 2012 页。Springer，。5. Gilles Barthe 、 Juan Manuel Crespo 和 C´esar Kunz。超越 2- 安全性：用于关系程序验证的非对称乘积程序。载于 Sergei N. Art¨emov 和 Anil Nerode 编, *Logical Foundations of Computer Science, InternationalSymposium, LFCS 2013, San Diego, CA, USA, January 6-8, 2013. Proceedings, Lecture Notes in Computer Science* 第 7734 卷, 第 29–43 页。Springer，2013。6. Gilles Barthe、Pedro R. D'Argenio 和 Tamara Rezk。通过自组合实现安全信息流。 *Math. Struct. Comput. Sci.*, 21(6):1207–1252, 2011。7. Jan Baumeister、Norine Coenen、Borzoo Bonakdarpour、Bernd Finkbeiner 和 C´esar S´anchez。异步超属性的时序逻辑。载于 Alexandra Silva 和 K. Rustan M. Leino 编, *Computer Aided Verification - 33rd International Conference CAV 2021 Virtual Event July 20-23 2021 Proceedings Part* , , , , , , , 第 卷 , *I* 12759 *Lecture Notes in Computer Science* 694 717 第 – 页。Springer，2021。8. Amir M. Ben-Amram 和 Samir Genaim。整数线性约束循环的线性排序问题。载于 Roberto Giacobazzi 和 Radhia Cousot 编, *The 40th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages POPL '13 Rome Italy - January 23 - 25 2013* 51 62 , , , , , 第 – 页。ACM，2013。9. Nick Benton。静态分析与程序变换的简单关系正确性证明。载于 Neil D. Jones 和 Xavier Leroy 编, *Proceedings of the 31st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2004 Venice Italy January 14-16 2004* 14 25 2004 , , , , , 第 – 页。ACM，。10. Raven Beutner 和 Bernd Finkbeiner。超越 k-安全性的超属性软件验证。载于 Sharon Shoham 和 Yakir Vizel 编, *Computer Aided Verification - 34th International Conference, CAV 2022, Haifa, Israel, August 7-10, 2022, Proceedings, Part I, Lecture Notes in Computer Science* 第 13371 卷，第 341–362 页。Springer，2022。11. Borzoo Bonakdarpour、C´esar S´anchez 和 Gerardo Schneider。结合静态分析与运行时验证监控超属性。载于 Tiziana Mar- garia 和 Bernhard Steffen 编, *Leveraging Applications of Formal Methods,*

*Verification and Validation. Verification - 8th International Symposium, ISoLA*2018, *Limassol, Cyprus, November 5-9*, 2018, *Proceedings, Part II*, *Lecture Notes in Computer Science*第11245卷，第8–27页。Springer，2018年。12. Laura Bozzelli、Adriano Peron 和 C´esar S´anchez。《异步超属性时序逻辑的表达能力与可判定性》。载于 Bartek Klin、Slawomir Lasota 和 Anca Muscholl 编，*33rd International Conference on ConcurrencyTheory, CONCUR 2022, September 12-16*, 2022, *Warsaw, Poland, LIPIcs*第243卷，第27:1–27:16页。Schloss Dagstuhl - Leibniz-Zentrum f¨ur Informatik，2022年。13. Aaron R. Bradley。《无需展开的基于SAT的模型检测》。载于 Ranjit Jhala 和 David A. Schmidt 编，*Verification, Model Checking, and Abstract Interpretation - 12th International Conference VMCAI 2011 Austin TX USA January , , , , , , , 第 卷, 23-25 2011. Proceedings 6538 Lecture Notes in Computer Science* 第70–87页。Springer，2011年。14. Roberto Bruttomesso、Edgar Pek、Natasha S harygina 和 Aliaksei Tsitovich。《openSMT 求解器》。载于 Javier Esparza 和 Rupak Majumdar 编，*Tools and Algo-rithms for the Construction and Analysis of Systems, 16th International Conference TACAS 2010 Held as Part of the Joint European Conferences on Theory and , , , , , , , Practice of Software ETAPS 2010 Paphos Cyprus March 20-28 2010. Proceed- , 第 卷，第 – ings 6015 Lecture Notes in Computer Science 150 153* 页。Springer，2010年。15. J¨urgen Christ、Jochen Hoenicke 和 Alexander Nutz。《SMTInterpol：一种插值 SMT 求解器》。载于 Alastair F. Donaldson 和 David Parker 编，*ModelChecking Software - 19th International Workshop SPIN 2012 Oxford UK July , , , , , , 23-24 2012. Proceedings 7385 Lecture Notes in Computer Science* 第 卷，第248–254页。Springer，2012年。16. Alessandro Cimatti 和 Alberto Griggio。《通过 IC3 进行软件模型检测》。载于 P. Madhusudan 和 Sanjit A. Seshia 编，*Computer Aided Verification - 24th International Conference CAV 2012 Berkeley CA USA July 7-13 2012 Proceed- , , , , , , , 第 卷, ings 7358 Lecture Notes in Computer Science 277 293* 第 – 页。Springer，2012年。17. Alessandro Cimatti、Alberto Griggio、Bastiaan Joost Schaafsma 和 Roberto Se-bastiani。《mathsat5 SMT 求解器》。载于 Nir Piterman 和 Scott A. Smolka 编，*Tools and Algorithms for the Construction and Analysis of Systems - 19th International Conference, TACAS 2013, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2013, Rome, Italy, March 16-24, 2013. Proceedings 7795 Lecture Notes in Computer Science 93 , 第 卷，第 –* 107页。Springer，2013年。18. Edmund M. Clarke、Orna Grumberg、Somesh Jha、Yuan Lu 和 Helmut Veith。《反例引导的抽象精化》。载于 E. Allen Emerson 和 A. Prasad Sistla 编，*Computer Aided Verification, 12th International Confer-ence, CAV 2000, Chicago, IL, USA, July 15-19*, 2000, *Proceedings, Lecture Notes in Computer Science*第1855卷，第154–169页。Springer，2000年。19. Michael R. Clarkson、Bernd Finkbeiner、Masoud Koleini、Kristopher K. Micinski、Markus N. Rabe 和 C´esar S´anchez。《超属性时序逻辑》。载于 Mart´ın Abadi 和 Steve Kremer 编，*Principles of Security and Trust - Third International Conference, POST 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April5-13*, 2014, *Proceedings, Lecture Notes in Computer Science*第8414卷，第265–284页。Springer，2014年。

20. Michael R. Clarkson 和 Fred B. Schneider。《超属性》。 *J. Comput. Secur.*, 18 6 1157 1210, 2010 ( ): – . 21. Norine Coenen、Bernd Finkbeiner、Jana Hofmann 和 Julia Tillman。《基于超属性的智能合约综合》*To appear at the 36th IEEE Computer Security Foundations Symposium (CSF 2023)* 2023 , . 22. Norine Coenen、Bernd Finkbeiner、C´esar S´anchez 和 Leander Tentrup。《超活性验证》。载于 Isil Dillig 和 Serdar Tasiran 编， *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18 2019 Proceedings Part I* 11561 *Lecture Notes in Computer Science* , , , , 第 卷，第 121 139 – 页。Springer，2019。。23. Jakub Daniel、Alessandro Cimatti、Alberto Griggio、Stefano Tonetta 和 Sergio Mover。《通过隐式抽象和良基关系实现无限状态活性到安全性转换》。载于 Swarat Chaudhuri 和 Azadeh Farzan 编， *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23 2016 Proceedings Part I* 9779 *Lecture Notes in Computer Science* , , , , 第 卷，第 271 291 – 页。Springer，2016。。24. Leonardo Mendonca de Moura 和 Nikolaj S. Bjørner。《Z3：一种高效的 SMT 求解器》。载于 C. R. Ramakrishnan 和 Jakob Rehof 编， *Tools and Algorithms for theConstruction and Analysis of Systems, 14th International Conference, TACAS 2008 Held as Part of the Joint European Conferences on Theory and Practice of Software ETAPS 2008 Budapest Hungary March 29-April 6 2008. Proceedings* , , , , , , 第 卷，第 4963 *Lecture Notes in Computer Science* 337 340 – 页。Springer，2008年。26. Daniel Dietsch、Matthias Heizmann、Vincent Langenfeld 和 Andreas Podelski。《模理论公平性：一种新的 LTL 软件模型检测方法》。载于 Daniel Kroening 和 Corina S. Pasareanu 编， *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I, Lecture Notes in Computer Science*第9206卷，第49–66页。Springer，2015年。27. Rayna Dimitrova、Bernd Finkbeiner 和 Hazem Torfah。《马尔可夫决策过程的概率超属性》。载于 Dang Van Hung 和 Oleg Sokolsky 编， *Automated Technology for Verification and Analysis - 18th International Symposium, ATVA 2020, Hanoi, Vietnam, October 19-23, 2020, Proceedings, Lecture Notes in Computer Science*第12302卷，第484–500页。Springer，2020年。28. Oyendrila Dobe、Erika ´Abrah´am、Ezio Bartocci 和 Borzoo Bonakdarpour。《Hyperprob：一种用于概率超属性的模型检测器》。载于 Marieke Huisman、Corina S. Pasareanu 和 Naijun Zhan 编， *Formal Methods - 24th International Symposium, FM 2021, Virtual Event, November 20-26, 2021, Proceedings, Lecture Notes in Computer Science*第13047卷，第657–666页。Springer，2021年。29. Marco Eilers、Peter Müller 和 Samuel Hitz。《模块化乘积程序》。 *ACM Trans. Program. Lang. Syst.* 42 1 3 1 3 37, 2020 , ( ): : – : . 30. Bernd Finkbeiner。《超属性模型检测算法》（特邀论文）。载于 Fritz Henglein、Sharon Shoham 和 Yakir Vizel 编， *Verification, ModelChecking, and Abstract Interpretation - 22nd International Conference, VMCAI*

2021, *Copenhagen, Denmark, January 17-19*, 2021, *Proceedings*, *Lecture Notes in Computer Science*第12597卷，页码3–16。Springer，2021年。31. Bernd Finkbeiner、Philippe Heim 和 Noemi Passing。模理论时序流逻辑。载于 Patricia Bouyer 和 Lutz Schr¨oder 编，*Foundations of Software Science and Computation Structures - 25th International Conference, FOSSACS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software ETAPS 2022 Munich Germany April 2-7 2022 Proceed-* , , , , , , , 卷 , 页 *ings* 13242 *Lecture Notes in Computer Science* 325 346 码 – 。Springer，2022年。32. Bernd Finkbeiner、Jana Hofmann、Florian Kohn 和 Noemi Passing。智能合约控制流的反应式综合。*CoRR*，abs/2205.06039，2022年。33. Bernd Finkbeiner、Felix Klein、Ruzica Piskac 和 Mark Santolucito。时序流逻辑：超越布尔值的综合。载于 Isil Dillig 和 Serdar Tasiran 编，*Computer Aided Verification - 31st International Conference, CAV 2019, New York City NY USA July 15-18 2019 Proceedings Part I* 11561 *Lecture* , , , , , , , 卷 , *Notes in Computer Science* 609 629 2019 页码 – 。Springer，。34. Bernd Finkbeiner 和 Noemi Passing。为活性综合占优策略。载于 Anuj Dawar 和 Venkatesan Guruswami 编，*42nd IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Sci-ence, FSTTCS 2022, December 18-20*, 2022, *IIT Madras, Chennai, India, LIPIcs*第250卷，页码37:1–37:19。Schloss Dagstuhl - Leibniz-Zentrum f¨ur Informatik，2022年。35. Bernd Finkbeiner、Markus N. Rabe 和 C´esar S´anchez。HyperLTL 和 HyperCTL* 的模型检测算法。载于 Daniel Kroening 和 Corina S. Pasareanu 编，*Computer Aided Verification - 27th International Conference, CAV 2015,San Francisco, CA, USA, July 18-24*, 2015, *Proceedings, Part I, Lecture Notes in Computer Science*第9206卷，页码30–48。Springer，2015年。36. Hadar Frenkel、Orna Grumberg 和 Sarai Sheinvald。针对无限数据域上系统与规约模型检测的自动机理论方法。

J. , ( ): – 。37. *Autom. Reason.* 63 4 1077 1101, 2019 Gideon Geier、Philippe Heim、Felix Klein 和 Bernd Finkbeiner。Syntroids：使用时序逻辑规约为现场可编程门阵列综合博弈。载于 Clark W. Barrett 和 Jin Yang 编，*2019 Formal Methods in Computer Aided Design FMCAD* , , , , , , , 页码 – *2019 San Jose CA USA October 22-25 2019* 138 146 2019 。IEEE，。38. Matthias Heizmann、Jochen Hoenicke、Jan Leike 和 Andreas Podelski。线性拉索程序的线性排序。载于 Dang Van Hung 和 Mizuhito Ogawa 编，*Automated Technology for Verification and Analysis 11th International Symposium ATVA 2013 Hanoi Vietnam October 15-18 2013. Proceedings* , , , , , , *Lecture Notes in Computer Science*第8172卷，页码365–380。Springer，2013年。39. Matthias Heizmann、Jochen Hoenicke 和 Andreas Podelski。为自动机爱好者提供的软件模型检测。载于 Natasha Sharygina 和 Helmut Veith 编，*Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg Russia July 13-19* 8044 *Lecture* , , , *Notes in Computer Science* 36 52 2013 *2013. Proceedings*, 卷 , 页码 – 。Springer，。40. Matthias Heizmann、Jochen Hoenicke 和 Andreas Podelski。通过学习终止程序进行终止性分析。载于 Armin Biere 和 Roderick Bloem 编，*Computer Aided Verification - 26th International Conference CAV 2014 Held as* , , , , , , , *Part of the Vienna Summer of Logic VSL 2014 Vienna Austria July 18-22 2014.* , 卷 , 页码 – 。*Proceedings* 8559 *Lecture Notes in Computer Science* 797 813 Springer，2014年。

41. Hsi-Ming Ho、Ruoyu Zhou 和 Timothy M. Jones。定时超属性验证。载于 Johann Gamper、Sophie Pinchinat 和 Guido Sciavicco 编，
26th International Symposium on Temporal Representation and Reasoning TIME , , , , , , 卷 , 页 *2019 October 16-19 2019 Málaga Spain* 147 *LIPIcs* 20 1 20 18 码 : – : 。Schloss Dagstuhl - Leibniz-Zentrum f¨ur Informatik，2019年。42. Donald B. Johnson。求有向图的所有基本回路。 *SIAM J. Comput.* 4 1 77 84, 1975 , ( ): – 。43. Leslie Lamport 和 Fred B. Schneider。使用 TLA 验证超属性。载于
34th IEEE Computer Security Foundations Symposium, CSF 2021, Dubrovnik, Croatia June 21-25 2021 1 16 2021 , , , 页码 – 。IEEE，。44. Benedikt Maderbacher 和 Roderick Bloem。使用抽象细化进行模理论反应式综合。 *CoRR*，abs/2108.00090，2021年。45. Daryl McCullough。非干扰与安全性质的可组合性。载于 *Proceedings of the 1988 IEEE Symposium on Security and Privacy, Oakland, California USA April 18-21 1988* 177 186 1988 , , , , 页码 – 。IEEE计算机学会，。46. Shohei Mochizuki、Masaya Shimakawa、Shigeki Hagihara 和 Naoki Yonezaki。通过非迁移式自动机将线性时序逻辑快速转换为 b¨uchi 自动机。载于 Stephan Merz 和 Jun Pang 编， *Formal Methods and Software Engineering - 16th International Conference on Formal Engineering Methods, ICFEM 2014, Luxembourg Luxembourg November 3-5* 8829 *Lec-* , , , *ture Notes in Computer Science* 364 379 2014 *2014. Proceedings*, 卷 , 页码 – 。Springer，。47. Luan Viet Nguyen、James Kapinski、Xiaoqing Jin、Jyotirmoy V. Deshmukh 和 Taylor T. Johnson。实值信号的超属性。载于 Jean-Pierre Talpin、Patricia Derler 和 Klaus Schneider 编， *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design, MEMOCODE 2017, Vienna, Austria, September 29 - October 02*, 2017，页码104–113。ACM，2017年。48. Amir Pnueli。程序时序逻辑。载于 *18th Annual Symposium on Foundations of Computer Science Providence Rhode Island USA 31 October* - , , , , , 页码 – 。*1 November 1977* 46 57 1977 IEEE计算机学会，。49. Yih-Kuen Tsay 和 Moshe Y. Vardi。从线性时序逻辑到 b¨uchi 自动机：早期而简单的原理。载于 Ernst-R¨udiger Olderog、Bernhard Steffen 和 Wang Yi 编，
Model Checking, Synthesis, and Learning - Essays Dedicated to Bengt Jonsson on The Occasion of His 60th Birthday, *Lecture Notes in Computer Science*第13030卷，页码8–40。Springer，2021年。50. Hiroshi Unno、Tachio Terauchi 和 Eric Koskinen。基于约束的关系验证。载于 Alexandra Silva 和 K. Rustan M. Leino 编， *Computer AidedVerification - 33rd International Conference, CAV 2021, Virtual Event, July 20*-23, 2021, *Proceedings, Part I*, *Lecture Notes in Computer Science*第12759卷，页码742–766。Springer，2021年。

A. 详细的正确性证明

A.1 超线性时序逻辑（HyperLTL）

设 AP 为有限的原子命题集合， Π 为有限的轨迹变量集合。那么，HyperLTL公式由以下语法定义

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

其中 $a \in AP$ 和 $\pi \in \Pi$。

HyperLTL公式的满足关系是相对于将轨迹变量映射 $m : \Pi \to (2^{AP})^\omega$ 到轨迹的映射定义的。为了处理量词，我们需要定义如何将此类映射扩展到新的轨迹变量。定义如下

$$\begin{aligned} &m[\pi \to s](\pi) = s \\ &m[\pi \to s](\pi') = m(\pi') && \text{对于 } \pi \neq \pi' \end{aligned}$$

相对于轨迹集合 $Z \subseteq 2^{AP^\omega}$、轨迹变量到轨迹的映射 $m : \Pi \to 2^{AP^\omega}$ 以及时间点 t，HyperLTL公式的满足关系递归定义如下

$$\begin{aligned} Z, t, m \vDash_{LTL} a_\pi &\iff a \in m(\pi)_t \\ Z, t, m \vDash_{LTL} \neg\varphi &\iff \neg(Z, t, m \vDash_{LTL} \varphi) \\ Z, t, m \vDash_{LTL} \varphi \wedge \psi &\iff Z, t, m \vDash_{LTL} \varphi \wedge Z, t, m \vDash \psi \\ Z, t, m \vDash_{LTL} \bigcirc\varphi &\iff Z, t+1, m \vDash_{LTL} \varphi \\ Z, t, m \vDash_{LTL} \varphi \mathcal{U} \psi &\iff \exists t' \geq t.\ Z, t', m \vDash_{LTL} \psi \wedge \forall t \leq t'' < t'.\ Z, t'', m \vDash_{LTL} \varphi \\ Z, t, m \vDash_{LTL} \forall \pi.\ \varphi &\iff \forall s \in Z.\ m[\pi \to s] \vDash_{LTL} \varphi \\ Z, t, m \vDash_{LTL} \exists \pi.\ \varphi &\iff \exists s \in Z.\ m[\pi \to s] \vDash_{LTL} \varphi \end{aligned}$$

我们将 $Z \vDash_{LTL} \varphi$ 定义为 $Z, 0, \varnothing \vDash_{LTL} \varphi$。

对于一种特殊情况，即只有一个轨迹量词且该量词为全称量词，我们所处的片段就是线性时序逻辑。

A.2 线性时序逻辑与 TSL 的相似性

以下引理阐述了（Hyper）TSL 与线性时序逻辑之间的重要关系。线性时序逻辑的语义是相对于原子命题的子集序列定义的，而 TSL 公式或无量词 HyperTSL公式的语义则是相对于（超）计算定义的。本论文的一个关键观察是，我们可以在两者之间进行“转换”：一个（超）计算定义了一个谓词与更新项子集序列。对于每个时间点，该子集恰好包含当前为真的谓词与更新项。

定义 17。令 $\hat{\zeta} \in \hat{\mathrm{A}}^\omega, \rho \subseteq \hat{\mathcal{T}}_P, \upsilon \subseteq \hat{\mathcal{T}}_U$。我们定义

$$Seq(\hat{\zeta}, \rho, \upsilon)_t = \{\hat{\tau_P} \in \rho \mid t, \varnothing, \hat{\zeta} \vDash \hat{\tau_P}\} \cup \{[\![c \leftarrowtail \hat{\tau_F}]\!] \in \upsilon \mid t, \varnothing, \hat{\zeta} \vDash [\![c \leftarrowtail \tau_F]\!]\}$$

$$Seq(\hat{\zeta}, \rho, \upsilon) = Seq(\hat{\zeta}, \rho, \upsilon)_0\ Seq(\hat{\zeta}, \rho, \upsilon)_1\ Seq(\hat{\zeta}, \rho, \upsilon)_2 \ldots$$

如果上下文已明确给出 ρ 和 υ，我们也会省略这些参数。

引理 1. 令 $t \in \mathbb{N}$。令 φ 为一个不含量词的 HyperTSL 公式。令 $\rho \subseteq \hat{} \hat{} \mathcal{T}_P$, $\upsilon \subseteq \mathcal{T}_U$ 分别为出现在 φ中的谓词项和更新项集合。那么

$$t, Seq(\hat{\zeta}) \vDash_{LTL} \varphi \Leftrightarrow t, \varnothing, \hat{\zeta} \vDash \varphi$$

证明。（引理 1）对 φ进行结构归纳。

– 情形 $\varphi = \hat{} \tau_P$

$$t, Seq(\hat{\zeta}) \vDash_{LTL} \hat{\tau_P} \Leftrightarrow \hat{\tau_P} \in Seq(\hat{\zeta})_t \Leftrightarrow t, \varnothing, \hat{\zeta} \vDash \hat{\tau_P}$$

– 情形 $\varphi = [\![c_\pi \leftarrowtail \hat{} \tau_F]\!]$

$$t, Seq(\hat{\zeta}) \vDash_{LTL} [\![c_\pi \leftarrowtail \hat{\tau_F}]\!] \Leftrightarrow \hat{\tau_P} \in Seq(\hat{\zeta})_t \Leftrightarrow t, \varnothing, \hat{\zeta} \vDash [\![c_\pi \leftarrowtail \hat{\tau_F}]\!]$$

– 情形 $\varphi = \neg\psi$

$$t, Seq(\hat{\zeta}) \vDash_{LTL} \neg\psi \Leftrightarrow \neg(t, Seq(\hat{\zeta}) \vDash_{LTL} \psi) \Leftrightarrow \neg(t, \varnothing, \hat{\zeta} \vDash \psi) \Leftrightarrow t, \varnothing, \hat{\zeta} \vDash \neg\psi$$

– 情形 $\varphi = \psi \wedge \psi'$

$$\begin{aligned} & t, Seq(\hat{\zeta}) \vDash_{LTL} \psi \wedge \psi' \\ \Leftrightarrow \quad & t, Seq(\hat{\zeta}) \vDash_{LTL} \psi \wedge t, Seq(\hat{\zeta}) \vDash_{LTL} \psi' \\ \Leftrightarrow \quad & t, \varnothing, \hat{\zeta} \vDash \psi \wedge t, \varnothing, \hat{\zeta} \vDash \psi' \\ \Leftrightarrow \quad & t, \varnothing, \hat{\zeta} \vDash \psi \wedge \psi' \end{aligned}$$

– 情形 $\varphi = \bigcirc\psi$

$$t, Seq(\hat{\zeta}) \vDash_{LTL} \bigcirc\psi \Leftrightarrow t+1, Seq(\hat{\zeta}) \vDash_{LTL} \psi \Leftrightarrow t+1, \varnothing, \hat{\zeta} \vDash \psi \Leftrightarrow t, \varnothing, \hat{\zeta} \vDash \bigcirc\psi$$

– 情形 $\varphi = \psi \mathcal{U} \psi'$

$$\begin{aligned} & t, Seq(\hat{\zeta}) \vDash_{LTL} \psi \mathcal{U} \psi' \\ \Leftrightarrow \quad & \exists t' \geq t.\ t', Seq(\hat{\zeta}) \vDash_{LTL} \psi' \wedge \forall t \leq t'' < t'.\ t'', Seq(\hat{\zeta}) \vDash_{LTL} \psi \\ \Leftrightarrow \quad & \exists t' \geq t.\ t', Z, \hat{\zeta} \vDash \psi' \wedge \forall t \leq t'' < t'.\ t'', Z, \hat{\zeta} \vDash \psi \\ \Leftrightarrow \quad & t, \varnothing, \hat{\zeta} \vDash \psi \mathcal{U} \psi' \end{aligned}$$

A.3 定理 4 的证明

证明。 $\Rightarrow$设 $q_0, q_1, q_2 \cdots \in Q^\omega$ 是 P 在 k-可行轨迹 σ 上的一次运行。那么，对于每个 $j \in \mathbb{N}$,

$e_j = ((q_j, \sigma_j, q_{j+1} \cdots, \sigma_{j+k-2}, q_{j+k-1})\, \sigma_{j+k-1}, (q_{j+1}, \sigma_{j+1}, \ldots, q_{j+k-1}, \sigma_{j+k-1}, q_{j+k}))$ 是 P_k 的一次转移。此外，对于每个 $k' < k$,

$$e_{k'} = ((q_0, s_0, q_1 \ldots, \sigma_{k'-1}, q_{k'}), \sigma_{k'}, (q_0, \sigma_0, \ldots, q_{k'}, \sigma_{k'}, q_{k'+1}))$$

也是 P_k的一次转移。因此， $q_0, q_1, \ldots$被 P_k 接受。

$\Leftarrow$设 σ 是被 P_k接受的 P 的一条轨迹。那么，存在 Pq_0的状态 $q_1 \ldots$，使得对于每个 $j \in \mathbb{N}$，上述 e_j 都是 P_k的一次转移。因此，根据 P_k 的定义，对于每个 j, $\sigma_j \cdots \sigma_{j+k-1}$ 都是可行的。因此， σ 是 k-可行的。□

A.4 定理 1 的证明

正确性证明的主要思路是构造：给定一个与程序轨迹 σ 相匹配的计算 ζ，构造一个与组合轨迹 combine($\sigma, Seq(\zeta)$) 相匹配的计算，反之亦然（Seq 在定义 17 中定义）。由此可得到所需的可行性证明。为此，我们定义两个 $\widetilde{\ }$ 操作，($-$) 和 ($-$)$|_\sigma$，它们几乎互为逆操作：我们有 (ζ)$|_\sigma$ = $\widetilde{\zeta}$；并且，如果对于某个 X，$\zeta \triangleleft$ combine(σ, X)，那么也有 $\zeta_{|\sigma}$ = ζ。在引理 2 中，我们证明，如果对于某个 X，$\zeta \triangleleft$ combine(σ, X)，那么 $\zeta_{|\sigma} \triangleleft \sigma$。在引理 3 中，我们证明，此时还有 $X = Seq$ ($\zeta_{|\sigma}$)。引理 4 阐述了相反方向：如果 $\zeta \triangleleft \sigma$，那么也有 $\tilde{\zeta} \triangleleft$ combine($\sigma, Seq(\zeta)$)。这三个引理给出了证明算法正确性所需的可行性证明。随后，引理 1 给出了 ζ 违反 TSL 公式与序列 $Seq(\zeta)$ 被 $A_{\neg\varphi}$ 接受之间的等价关系，这对于推理 B¨uchi 程序乘积中是否存在轨迹 combine($\sigma, Seq(\zeta)$) 是必要的。

我们首先定义操作 $\widetilde{\ }$ 和 $\zeta \triangleleft \sigma$。我们需要将此计算扩展为与 combine(σ, $Seq(\zeta)$) 匹配的计算。对于每个时间点 t，我们需要引入与 combine($\sigma_t, Seq(\zeta)_t$) = 匹配的计算步骤

保存值 $Seq(\zeta)_t$；σ_t；新增输入；检查谓词 $Seq(\zeta)_t$；检查更新 $Seq(\zeta)_t$。执行保存值 $Seq(\zeta)_t$ 时，临时变量的值会按照语句 $tmp_j := \tau_{Fj}$ 的要求发生变化。执行实际语句 σ_t 时，计算变为 ζ_t，但仍使用“旧的”输入值，并扩展了时序变量的值。接下来，执行新增输入时，我们逐步将输入值变更为 ζ_t 中的值。然后执行断言，此后计算不再发生变化。

接下来，我们还需要赋值扩展这一概念：对于 $c \neq c'$，我们定义 $a[c \mapsto v](c) = v$ 和 $a[c \mapsto v](c') = a(c')$。

设 $v \subseteq \mathcal{T}_U$ 为公式 φ 中出现的更新项集合，$\rho \subseteq \mathcal{T}_P$ 为其中出现的谓词项集合。

定义18。令 $\mathbb{I} = \{i_1, \ldots i_n\}$ 为输入集，且 $v = \{[\![c_1 \leftarrowtail \tau_{F1}]\!], \ldots, [\![c_m \leftarrowtail \tau_{Fm}]\!]\}$。给定一个计算 ζ，我们将 $\widetilde{\ }$ 适配计算 ζ 定义如下。

$$
\begin{aligned}
a_t^{tmp_1} &:= \zeta_{t-1}[tmp_1 \mapsto \eta(\tau_{F1}, \zeta_{t-1})] \\
a_t^{tmp_j} &:= a^{tmp_{j-1}}[tmp_j \mapsto \eta(\tau_{Fj}, \zeta_{t-1})] && for\ 1 < j \leq m \\
a_t &:= \zeta_t[tmp_1 \mapsto \eta(\tau_{F1}, \zeta_{t-1}), \ldots, tmp_m \mapsto \eta(\tau_{Fm}, \zeta_{t-1}), \\
&\qquad i_1 \mapsto \zeta_{t-1}(i_1),\ \ldots,\ i_n \mapsto \zeta_{t-1}(i_n)] \\
a_t^{i_1} &:= a_t[i_1 \mapsto \zeta_t(i_1)] \\
a_t^{i_j} &:= a^{i_{j-1}}[i_j \mapsto \zeta_t(i_j)] && for\ 1 < j \leq n \\
\widetilde{\zeta_t} &:= a_t^{tmp_1} \ldots a_t^{tmp_m}\ a_t\ a_t^{i_1}\ \ldots\ a_t^{i_n}\ a_t^{i_n}\ a_t^{i_n} \\
\tilde{\zeta} &:= \widetilde{\zeta_0}\ \widetilde{\zeta_1} \ldots
\end{aligned}
$$

请注意，这是适配计算 $\zeta \triangleleft \sigma$ 的唯一可能方式，使得对于任意 X，结果都能与 combine(σ, X) 匹配。

另请注意， $a_t^{i_n} = \zeta_t[tmp_1 \mapsto \eta(\tau_{F1}, \zeta_{t-1}) \ldots, tmp_m \mapsto \eta(\tau_{Fm}, \zeta_{t-1})]$。

我们还可以定义此操作的左逆：将与 combine(σ, X) 匹配的计算约化为与 σ 匹配的计算。

定义 19。设 $\sigma \in \mathrm{Stmt}^\omega, X \in \mathcal{P}(\mathcal{T}_P \cup \mathcal{T}_U)^\omega$ 和 $\zeta \triangleleft \text{combine}(\ \sigma, X)$ 。我们将约化计算 $\zeta_{|\sigma}$ 定义如下。

$$\iota(j) := (|\mathbb{I}| + |v| + 3) \cdot (j+1) - 3$$
$$\zeta_{|\sigma}(U) := (\zeta_{\iota(0)})_{|(\mathbb{I} \cup \mathbb{C})}\ (\zeta_{\iota(1)})_{|(\mathbb{I} \cup \mathbb{C})} \cdots$$

其中 $a_{|(\mathbb{I} \cup \mathbb{C})}$ 表示将赋值的定义域限制为原始输入和单元，从而排除时序变量 tmp_1, $tmp_2 \ldots$

注意，如果 $\zeta \triangleleft \text{combine}(\ \sigma, X)$，那么我们也有 $\zeta = \widetilde{\zeta_{|\sigma}}$，因为这是唯一可能与 combine(σ, X) 匹配且在限制到 σ 后等于 $\zeta_{|\sigma}$ 的计算。

引理 2. 如果 $\zeta \triangleleft \text{combine}(\ \sigma, X)$，那么 $\zeta_{|\sigma} \triangleleft \sigma$。

证明。我们需要证明 $\forall t \in \mathbb{N}.\ \zeta_{|\sigma} \triangleleft_t \sigma$

$\widetilde{\ }$回想一下， $\zeta = \zeta_{|\sigma}$ 并且

$$\widetilde{(\zeta_{|\sigma})_t} = a_t^{tmp_1} \ldots a_t^{tmp_m}\ a_t\ a_t^{i_1}\ \ldots\ a_t^{i_n}\ a_t^{i_n}\ a_t^{i_n}$$

– 情况 $\sigma_t =$ 断言(τ_P)

我们知道 $\zeta \triangleleft_{\iota(t)-|\mathbb{I}|} \text{combine}(\ \sigma, X)$。相应的陈述是 σ_j，因此

$$\eta(\tau_P, \zeta_{\iota(t)-|\mathbb{I}|-1}) = true \quad \wedge \forall c \in \mathbb{C}^*.\ \zeta_{\iota(t)-|\mathbb{I}|}(c) = \zeta_{\iota(t)-|\mathbb{I}|-1}(c)$$

此外，($\zeta_{\iota(t)-|\mathbb{I}|-1}$) $= a_t^{tmp_m}$。这等于 ($\zeta_{|\sigma})_{t-1}$ ，并扩展了临时变量的取值。

由于 τ_P 不包含时序变量，这意味着 $\eta(\tau_P, ((\zeta_{|\sigma})_{t-1})_{|(\mathbb{I} \cup \mathbb{C})})$ 也成立。还需证明

$$\forall c \in \mathbb{C}.\ (\zeta_{\iota(t-1)})_{|(\mathbb{I} \cup \mathbb{C})}(c) = (\zeta_{\iota(t)})_{|(\mathbb{I} \cup \mathbb{C})}(c)$$

这是成立的，因为在 $\zeta_{\iota(t-1)} \ldots \zeta_{\iota(t)-|\mathbb{I}|-1}$ 和在 $\zeta_{\iota(t)-|\mathbb{I}|}, \ldots \zeta_{\iota(t)}$中唯一发生变化的单元都是来自 $\mathbb{C}^* \backslash \mathbb{C}$的单元。– 剩余的两种情况类似。

引理 3. 如果 $\zeta \triangleleft \mathit{combine}(\sigma, X)$，那么 $X = \mathit{Seq}(\zeta_{|\sigma})$。

证明。我们证明 $\forall t.\ X_t = \mathit{Seq}(\zeta_{|\sigma})_t$。我们知道 $\zeta \triangleleft_{\iota(t)+1} \text{combine}(\ \sigma, X)$。对应的陈述是检查谓词$X_t$。设 $h = (\bigwedge_{\tau_P \in X_t} \tau_P \wedge \bigwedge_{\tau_P \in \rho \backslash X_t} \neg \tau_P)$。这意味着

$$\eta(h, \zeta_{\iota(t)+1}) = true \quad \wedge \forall c \in \mathbb{C}^*.\ \zeta_{\iota(t)+1}(c) = \zeta_{\iota(t)}(c)$$

这意味着 $true = \eta((h, \zeta_{\iota(t)})|_{(\mathbb{I}\cup\mathbb{C})}) = \eta(h, (\zeta_{|\sigma})_t)$。因此，对于所有 $\tau_P \in \rho$

$$\tau_P \in Seq(\zeta_{|\sigma})_t \iff t, \zeta_{|\sigma} \vDash \tau_P \iff \eta(\tau_P, \zeta_{\iota(t)}) = true \iff \tau_P \in X_t$$

对于更新项，我们知道 $\zeta \lhd_{\iota(t)+2} \text{combine}(\sigma, X)$。相应的语句是 check updates X_t. Set $h = (\bigwedge_{[\![c_j \leftarrowtail \tau_{Fj}]\!] \in \upsilon} \begin{cases} c_j = tmp_j & [\![c_j \leftarrowtail \tau_{Fj}]\!] \in X_t \\ c_j \neq tmp_j & \text{if else} \end{cases})$ 和之前一样，我们知道 $\eta(h, (\zeta_{|\sigma})_t) = true$。此外，根据定义18，我们知道对于每个 j，$(\zeta_{|\sigma})_t(tmp_j) = \eta(\tau_{Fj}, (\zeta_{|\sigma})_{t-1})$。因此，对于每个 $[\![c_j \leftarrowtail \tau_{Fj}]\!] \in \upsilon$，

$$\begin{aligned} [\![c_j \leftarrowtail \tau_{Fj}]\!] \in Seq(\zeta_{|\sigma})_t &\iff t, \zeta_{|\sigma} \vDash [\![c_j \leftarrowtail \tau_{Fj}]\!] \\ &\iff \eta(\tau_{Fj}, \zeta_{\iota(t-1)}) = \eta(c_j, \zeta_{\iota(t)}) \\ &\iff \eta(c_j = tmp_j, \zeta_{\iota(t)}) = true \\ &\iff [\![c_j \leftarrowtail \tau_{Fj}]\!] \in X_t \end{aligned}$$

引理 4. 若 $\zeta \lhd \sigma$，则 $\tilde{\zeta} \lhd \text{combine}(\sigma, Seq(\zeta))$

证明。我们必须证明，对于所有 t，都有 $\tilde{\zeta} \lhd_t \text{combine}(\sigma, Seq(\zeta))$。显然，除 check preds 或 check updates 类型的时间步外，其他所有时间步均满足这一点，这由 $\tilde{\zeta}$ definition of ζ 得出。

首先考虑检查语句。我们需要证明 $\forall$

$$\eta\left(\left(\bigwedge_{\tau_P \in Seq(\zeta)_t} \tau_P \wedge \bigwedge_{\tau_P \in \rho \setminus Seq(\zeta)_t} \neg\tau_P\right), \tilde{\zeta}_{\iota(t)}\right) = true$$

由于临时变量 $\text{tmp}_1, \text{tmp}_2 \cdots$ 未用于任何 $\tau_P \in \rho$，根据定义18，这等价于

$$\forall \tau_P \in \rho.\ \tau_P \in Seq(\zeta)_t \iff \eta(\tau_P, \zeta_t) = true$$

根据 $Seq(\zeta)_t$ 的定义，这成立。

现在考虑检查更新。我们需要证明 $\forall t, \tilde{\zeta} \lhd_{\iota(t)+2} \text{combine}(\sigma, Seq(\zeta))$。这归结为

$$\eta\left(\left(\bigwedge_{[\![c_j \leftarrowtail \tau_{Fj}]\!] \in \upsilon} \begin{cases} c_j = tmp_j & \text{if } [\![c_j \leftarrowtail \tau_{Fj}]\!] \in Seq(\zeta)_t \\ c_j \neq tmp_j & \text{else} \end{cases}\right), \tilde{\zeta}_{\iota(t)}\right) = true$$

这等价于

$$\forall [\![c_j \leftarrowtail \tau_{Fj}]\!] \in \upsilon.\ \eta(c_j = tmp_j, \tilde{\zeta}_{\iota(t)}) = true \iff [\![c_j \leftarrowtail \tau_{Fj}]\!] \in Seq(\zeta)_t$$

我们知道 $\tilde{\zeta}_{\iota(t)}(\text{tmp}_j) = \eta(\tau_{Fj}, \zeta_{t-1})$。因此，这等价于

$$\forall [\![c_j \leftarrowtail \tau_{Fj}]\!] \in \upsilon.\ \eta(\tau_{Fj}, \zeta_{t-1}) = \zeta_t(c) \iff [\![c_j \leftarrowtail \tau_{Fj}]\!] \in Seq(\zeta)_t$$

根据 $Seq(\zeta)_t$ 的定义，这一点再次成立。

现在，我们已经拥有证明定理 1 所需的全部引理

证明。（定理 1）

⇒ 假设 φ 有一条可行轨迹。因此，它是某个 $\sigma \in \mathcal{L}(P)$ 和 $X \in \mathcal{L}(A_{\neg\varphi})$ 的轨迹 combine(σ, X)。此外，对于某个 $\zeta \in \mathrm{A}^{\omega}$， $\zeta \triangleleft$ combine(σ, X)。根据引理 3，我们知道 $X = Seq(\zeta)$；根据引理 2，我们知道 $\zeta \triangleleft \sigma$。根据 A_{φ} 的正确性，我们知道 $Seq(\zeta) \vDash_{LTL} \neg\varphi$，而根据引理 1，这意味着 $\zeta \vDash \neg\varphi$。因此， ζ 是一个反例，证明了 P不满足 φ。

⇐ 假设 P 不满足 φ。那么，存在一条轨迹 $\sigma \in \mathcal{L}(P)$ 和一次计算 ζ，使得 $\zeta \triangleleft \sigma$ 且 $\zeta \vDash \neg\varphi$。根据引理 1，这意味着$Seq(\zeta) \vDash_{LTL} \neg\varphi$，因此 $Seq(\zeta)$ 被 $A_{\neg\varphi}$ 接受。于是，combine($\sigma, Seq(\zeta)$) 是 $P \otimes A_{\neg\varphi}$ 的一条轨迹。根据引理 4， $\tilde{\zeta} \triangleleft$combine($\sigma, Seq(\zeta)$)，因此这也是一条可行轨迹。

A.5 定理 2 和 3 的证明

由于这两个定理互为对偶，只需给出定理 2 的证明。

证明与定理 1 的证明类似，但现在我们必须处理多条轨迹，因此索引也更多。为求完整起见，我们在此给出证明。

给定 n程序轨迹 $\sigma_{\pi_1}, \dots \sigma_{\pi_n}$，我们定义 $\sigma_j = ((\sigma_{\pi_1})_{\pi_1 j}; (\sigma_{\pi_2})_{\pi_2 j}; \dots (\sigma_{\pi_n})_{\pi_n j})$ ˆ 和 $\sigma = \sigma_1\sigma_2 \dots$。令 $\zeta_{\pi_1} \triangleleft \sigma_{\pi_1} \wedge \dots \wedge \zeta_{\pi_n} \triangleleft \sigma_{\pi_n}$。令 $\zeta = \varnothing[\pi_1, \zeta_{\pi_1}] \dots [\pi_n, \zeta_{\pi_n}]$。这些计算可以扩展为一个与 combine($\sigma, Seq(\hat{\zeta})$) 相匹配的计算。对于每个时间点 t，我们需要引入与 combine(σ_t, X_t) 相匹配的计算步骤： =保存值； σ_t；新输入；检查谓词 X_t；检查更新X_t。执行保存值时，相关临时变量的值会按照语句 $\overline{tmp_j} := \hat{}\tau_{Fj}$ 的要求发生变化。执行实际语句 σ_t ˆ 后，计算变为 ζ_t，但仍使用“旧”输入，并扩展了时序变量的值。接下来，在执行新输入时，我们逐步将输入值改为 ζ_t 中的值。然后执行断言，此后计算不再发生变化。

接下来，我们还需要超赋值扩展的概念：对于 $c \neq c'$，我们定义 $\hat{a}[c \mapsto v](c) = v$ and ˆ $a[c \mapsto v](c') = \hat{a}(c')$。

令 $\upsilon \subseteq \hat{\mathcal{T}}_U$ 为以下更新项集合，且 $\rho \subseteq$ ˆ $\mathcal{T}_P$ 为公式 φ中出现的谓词项。

定义 20. 令 $\mathbb{I} \times \Pi = \{i_1, \dots i_k\}$ 为输入集，且 $\upsilon = \{[\![c_1 \leftarrowtail \hat{} \hat{}\tau_{F1}]\!], \dots, [\![c_m \leftarrowtail \hat{}\tau_{Fm}]\!]\}$。给定计算 $\zeta_{\pi_1} \dots \zeta_{\pi_n}$，令 $\zeta = \varnothing[\pi_1, \sigma_{\pi_1}] \dots [\pi_n, \sigma_{\pi_n}]$. 我们定义适配计算 ($\zeta_{\pi_1}, \dots, \zeta_{\pi_n}$)。

$$
\begin{aligned}
\hat{a}_t^{tmp_1} &:= \hat{\zeta}_{t-1}[tmp_1 \mapsto \eta(\tau_{F1}, \hat{\zeta}_{t-1})] \\
\hat{a}_t^{tmp_j} &:= \hat{a}^{tmp_{j-1}}[tmp_j \mapsto \eta(\tau_{Fj}, \hat{\zeta}_{t-1})] \qquad\qquad for\ 1 < j \leq m \\
\hat{a}_t^{\pi_j} &= (\hat{\zeta}_{t-1}[\pi_1, \zeta_{\pi_1}] \dots [\pi_j, \zeta_{\pi_j}])_t[i_1 \mapsto \zeta_{t-1}(i_1),\ \dots\ , i_k \mapsto \zeta_{t-1}(i_k), \\
&\qquad tmp_1 \mapsto \eta(\tau_{F1}, \hat{\zeta}_{t-1}), \dots, tmp_m \mapsto \eta(\tau_{Fm}, \hat{\zeta}_{t-1})]
\end{aligned}
$$

$$\hat{a}_t^{i_1} := \hat{a}_t^{\pi_n}[i_1 \mapsto \hat{\zeta}_t(i_1)]$$
$$\hat{a}_t^{i_j} := \hat{a}^{i_{j-1}}[i_j \mapsto \hat{\zeta}_t(i_j)] \qquad for\ 1 < j \le k$$
$$(\zeta_{\pi_1}, \widetilde{\ldots, \zeta_{\pi_n}})^t := \hat{a}_t^{tmp_1} \ldots \hat{a}_t^{tmp_n}\ \hat{a}_t^{\pi_1} \ldots \hat{a}_t^{\pi_n}\ \hat{a}_t^{i_1}\ \ldots\ \hat{a}_t^{i_k}\ \hat{a}_t^{i_k}\ \hat{a}_t^{i_k}$$
$$(\zeta_{\pi_1}, \widetilde{\ldots, \zeta_{\pi_n}}) := (\zeta_{\pi_1}, \widetilde{\ldots, \zeta_{\pi_n}})^0\ (\zeta_{\pi_1}, \widetilde{\ldots, \zeta_{\pi_n}})^1 \ldots$$

请注意，这是将计算 $\zeta_{\pi_1} \lhd \sigma_{\pi_1}, \ldots, \zeta_{\pi_n} \lhd \sigma_{\pi_n}$ 扩展为一个可能与 combine(σ, X) 匹配的计算（对任意 X而言）的唯一可能方式。

我们还可以定义此操作的左逆：将匹配 combine(σ, X) 的计算约化为匹配 $\sigma_{\pi_1}, \ldots, \sigma_{\pi_n}$ 的计算，定义如下。

定义 21. 设 $1 \le j \le n, \sigma \in \mathrm{Stmt}^\omega, X \in \mathcal{P}(\hat{\mathcal{T}}_P \cup \hat{\mathcal{T}}_U)^\omega$ 且 $\hat{\zeta} \lhd$combine(σ, X) 。我们将 $(\sigma_{\pi_j})_t$ 在 combine(σ, X) 中的计算步骤索引定义为

$$\iota(t) := (|\mathbb{I} \times \Pi| + |v| + n + 2) \cdot (t + 1) - 3$$

我们将约化计算定义为 $\zeta_{|\pi_j}$。

$$\zeta_{|\pi_j} := (\zeta_{\iota(0)})_{|(\mathbb{I} \cup \mathbb{C})_{\pi_j}}\ (\zeta_{\iota(1)})_{|(\mathbb{I} \cup \mathbb{C})_{\pi_j}} \ldots$$

其中，$\hat{a}_{|(\mathbb{I} \cup \mathbb{C})_{\pi_j}}$ 表示将赋值的定义域限制为标记为 π_j 的单元和输入，从而排除时序变量 $tmp_1, tmp_2 \ldots$ 以及来自其他轨迹的变量。此外，单元和输入会再次重命名：从 c_{π_j} 改为 c，或从 i_{π_j} 改为 i。

注意，如果 $\hat{\zeta} \lhd$ combine(σ, X)，我们也有 $\hat{\zeta}$ 是 ($\hat{\zeta}_{|\pi_1}, \ldots, \hat{\zeta}_{|\pi_n}$) 的适配计算，因为这是唯一能够匹配 combine(σ, X) 且限制到 π_j 后等于 $\zeta_{|\pi_j}$ 的计算。

引理 5. 如果 $\hat{\zeta} \lhd$combine(σ, X) 且 $\sigma = ((\sigma_{\pi_1})_{\pi_1}, \ldots, (\sigma_{\pi_n})_{\pi_n})$，那么对于每个 $1 \le j \le n$，都有 $\hat{\zeta}_{|\pi_j} \lhd \sigma_{\pi_j}$。

证明。我们证明 $\forall t \in \mathbb{N}.\ \zeta_{|\pi_j} \lhd_t \sigma_{\pi_j}$。

回顾一下，$\hat{\zeta}$ 是 ($\hat{\zeta}_{|\pi_1}, \ldots, \hat{\zeta}_{|\pi_n}$) 的适配计算。

– 情形 $\sigma_t =$ 断言(τ_P)^我们知道 $\zeta \lhd_{\iota(t)-|\mathbb{I}\times\Pi|-(n-j)}$ combine(σ, X)，因此

$$\eta(\hat{\tau_P}, \hat{\zeta}_{\iota(t)-|\mathbb{I}\times\Pi|-(n-j)-1}) = true \ \wedge$$
$$\forall c \in \mathbb{C}^*.\ \hat{\zeta}_{\iota(t)-|\mathbb{I}\times\Pi|-(n-j)}(c) = \hat{\zeta}_{\iota(t)-|\mathbb{I}\times\Pi|(n-j)-1}(c)$$

此外，($\hat{\zeta}_{\iota(t)-|\mathbb{I}\times\Pi|-(n-j)-1}$) 在 $j = 0$ 时等于 $\hat{a}_t^{tmp_m}$，否则等于 $\hat{a}_t^{\pi_{j-1}}$；将其限制为来自 π_j 的输入和变量时，两者都等于 $(\zeta_{|\pi_j})_{t-1}$。τ_P 不包含来自其他轨迹的变量或临时变量，因此 $\eta(\tau_P, (\hat{\zeta}_{|\pi_j})_{t-1})$ 也成立。接下来只需证明

$$\forall c \in \mathbb{C}.\ ((\hat{\zeta})_{\iota(t-1)})_{|(\mathbb{I} \cup \mathbb{C})_{\pi_j}}(c) = ((\hat{\zeta})_{\iota(t)})_{|(\mathbb{I} \cup \mathbb{C})_{\pi_j}}(c)$$

^ ^ 这同样成立，因为在 $\zeta_{\iota(t-1)} \ldots \zeta_{\iota(t)-|\mathbb{I}\times\Pi|-(n-j)-1}$ 和 $\hat{\zeta}_{\iota(t)-(n-j)-|\mathbb{I}\times\Pi|}$、……$\hat{\zeta}_{\iota(t)}$ 中发生变化的唯一单元，都是来自 $\mathbb{C}^* \backslash \mathbb{C}$ 的单元或来自其他轨迹的单元。

– 剩下的两种情况与此类似。

引理 6. 若 $\hat{\zeta} \triangleleft combine(\sigma, X)$，则 $X = Seq(\varnothing[\pi_1, \hat{\zeta}_{|\pi_1}] \dots [\pi_n, \hat{\zeta}_{|\pi_n}])$

证明. 设 $\hat{\zeta}' = \varnothing[\pi_1, \hat{\zeta}_{|\pi_1}] \dots [\pi_n, \hat{\zeta}_{|\pi_n}]$。我们证明 $\forall t.\ X_t = Seq(\hat{\zeta}')_t$。我们知道 $\hat{\zeta} \triangleleft_{\iota(t)+1}$ combine(σ, X)。相应的陈述是 $check\_preds_{X_t}$。设$h = (\bigwedge_{\tau_P \in X_t} \hat{\tau}_P \wedge \bigwedge_{\hat{\tau}_P \in \rho \setminus X_t} \neg \tau_P)$。这意味着

$$\eta(h, \hat{\zeta}_{\iota(t)+1}) = true \quad \wedge \forall c \in \mathbb{C}^*.\ \zeta_{\iota(t)+1}(c) = \hat{\zeta}_{\iota(t)}(c)$$

请注意，根据定义 20， $\zeta_{\iota(t)+1}$ 等于

$$(\hat{\zeta}'_{t-1}[\pi_1, \hat{\zeta}'_{\pi_1}] \dots [\pi_j, \hat{\zeta}'_{\pi_n}])_t\ [tmp_1 \mapsto \eta(\hat{\tau_{F1}}, \hat{\zeta}'_{t-1}), \dots, tmp_m \mapsto \eta(\hat{\tau_{Fm}}, \hat{\zeta}'_{t-1})]$$
$$= \hat{\zeta}'_t\ [tmp_1 \mapsto \eta(\hat{\tau_{F1}}, \hat{\zeta}'_{t-1}), \dots, tmp_m \mapsto \eta(\hat{\tau_{Fm}}, \hat{\zeta}'_{t-1})]$$

h 不包含时序变量，因此这意味着 $\eta(h, \hat{\zeta}'_t) = true$。因此，对于所有 $\tau_P \in P$

$$\tau_P \in Seq(\hat{\zeta}')_t \Leftrightarrow t, \hat{\zeta}' \vDash \tau_P \Leftrightarrow \eta(\tau_P, \zeta'_t) = true \Leftrightarrow \tau_P \in X_t$$

最后一个等价关系根据 h的定义成立。

对于更新项，我们知道 $\zeta \triangleleft_{\iota(t)+2}$combine(σ, X)。对应的陈述是 check updatesX_t。令 $h = (\bigwedge_{[\![c_j \leftarrow \hat{\tau}_{Fj}]\!] \in \upsilon} \begin{cases} c_j = & j \\ c_j \neq & j \end{cases} [\![c_j \leftarrow \tau_{Fj}]\!] \in X_t$ tmp if ˆ tmp else) 与之前一样，

我们知道 $\eta(h, \hat{\zeta}'_t) = true$。此外，根据定义 20，我们知道对于每个 j，$\hat{\zeta}_{\iota(t)+1}$(tmp$j$) $= \eta(\hat{\tau}_{Fj}, \hat{\zeta}'_{t-1})$。因此，对于每个 $[\![c_j \leftarrow \hat{\tau}_{Fj}]\!] \in \upsilon$,

$$\begin{aligned} [\![c_j \leftarrow \hat{\tau_{Fj}}]\!] \in Seq(\hat{\zeta}')_t &\Leftrightarrow t, \hat{\zeta}' \vDash [\![c_j \leftarrow \hat{\tau_{Fj}}]\!] \\ &\Leftrightarrow \eta(\hat{\tau_{Fj}}, \hat{\zeta}'_{t-1}) = \eta(c_j, \hat{\zeta}'_t) \\ &\Leftrightarrow \eta(c_j = tmp_j, \hat{\zeta}'_t) = true \\ &\Leftrightarrow [\![c_j \leftarrow \hat{\tau_{Fj}}]\!] \in X_t \end{aligned}$$

最后一个等价关系同样根据 h的定义成立。

引理 7. 如果 $\zeta_{\pi_1} \triangleleft \sigma_{\pi_1} \wedge \dots \wedge \zeta_{\pi_n} \triangleleft \sigma_{\pi_n}$，那么 $\widetilde{\quad}(\zeta_{\pi_1}, \dots, \zeta_{\pi_n}) \triangleleft$combine(σ, $Seq(\hat{\zeta}')$)，其中 $\hat{\zeta}' = \varnothing[\pi_1, \zeta_{\pi_1}] \dots [\pi_n, \zeta_{\pi_n}]$

证明。令 $\hat{\zeta} = \zeta_{\pi_1} \widetilde{\dots}(\ ,\ , \zeta_{\pi_n})$。我们需要证明，对于所有 t，$\hat{\zeta} \triangleleft_t$combine($\sigma, Seq(\hat{\zeta}')$)。根据 ζ 的定义，除检查谓词或检查更新 ˆ 这两类时间步外，其余时间步均显然成立。

首先考虑检查谓词。- 我们需要证明 $\forall t$, $\hat{\zeta} \triangleleft_{\iota(t)+1}$combine($\sigma, Seq(\hat{\zeta}')$) 。归根结底,

$$\eta\left(\left(\bigwedge_{\hat{\tau_P} \in Seq(\hat{\zeta}')_t} \hat{\tau_P} \wedge \bigwedge_{\hat{\tau_P} \in \rho \setminus Seq(\hat{\zeta}')_t} \neg \hat{\tau_P}\right), \hat{\zeta}_{\iota(t)+1}\right) = true$$

ˆ请回顾，根据定义 20， $\zeta_{\iota(t)+1}$ 等于

$$(\hat{\zeta}'_{t-1}[\pi_1, \hat{\zeta}'_{\pi_1}] \dots [\pi_j, \hat{\zeta}'_{\pi_n}])_t \; [tmp_1 \mapsto \eta(\tau_{\hat{F}1}, \hat{\zeta}'_{t-1}), \dots, tmp_m \mapsto \eta(\tau_{\hat{F}m}, \hat{\zeta}_{t-1})]$$
$$= \hat{\zeta}'_t \; [tmp_1 \mapsto \eta(\tau_{\hat{F}1}, \hat{\zeta}'_{t-1}), \dots, tmp_m \mapsto \eta(\tau_{\hat{F}m}, \hat{\zeta}_{t-1})]$$

因此，由于临时变量未在 ˆτ_P 中使用，这等价于

$$\forall \hat{\tau_P} \in \rho.\ \hat{\tau_P} \in Seq(\zeta')_t \iff \eta((\hat{\tau_P}), \zeta'_t) = true$$

根据 $Seq(\zeta)_t$ 的定义，这成立。

现在考虑检查更新。- 我们需要证明 $\forall t, \hat{\zeta} \triangleleft_{\iota(t)+2}$combine($\sigma, Seq(\hat{\zeta}')$)。这归结为

$$\eta\left(\left(\bigwedge_{[\![c_j \leftarrowtail \tau_{\hat{F}j}]\!] \in \upsilon} \begin{cases} c_j = tmp_j & \text{if } [\![c_j \leftarrowtail \tau_{\hat{F}j}]\!] \in Seq(\hat{\zeta}')_t \\ c_j \neq tmp_j & \text{else} \end{cases}\right), \hat{\zeta}_{\iota(t)+2}\right) = true$$

这又等价于

$$\forall [\![c_j \leftarrowtail \tau_{Fj}]\!] \in \upsilon.\ \eta(c_j = tmp_j, \hat{\zeta}_{\iota(t)+2}) = true \iff [\![c_j \leftarrowtail \tau_{Fj}]\!] \in Seq(\hat{\zeta}')_t$$

我们知道 ˆ $\zeta_{\iota(t)+2}$(tmp$_j$) = ˆζ'_{t-1}(τ_{Fj})。因此，这等价于

$$\forall [\![c_j \leftarrowtail \tau_{\hat{F}j}]\!] \in \upsilon.\ \eta(\tau_{\hat{F}j}, \hat{\zeta}'_{t-1}) = \hat{\zeta}'_t(c) \iff [\![c_j \leftarrowtail \tau_{\hat{F}j}]\!] \in Seq(\hat{\zeta}')_t$$

根据 Seq(ˆζ')$_t$的定义，这再次成立。

现在，我们已经具备了证明定理 2 所需的所有引理

证明。（定理 2）

⇒ 假设 $P^n \otimes A_{\neg\varphi}$ 有一条可行轨迹。那么，这是一条轨迹 combine(σ, X)，其中 $\sigma \in \mathcal{L}$(P^n)和 $X \in \mathcal{L}(A_{\neg\varphi})$。我们知道 σ_t =((σ_{π_1})$_{\pi_1 t}$; . . . ; (σ_{π_n})$_{\pi_n t}$)。此外，对于某个 ˆ $\zeta \in$ ˆ A$^\omega$，有 ˆ $\zeta \triangleleft$combine(σ, X)。根据引理 5，我们知道 ˆ$\zeta_{|\pi_1} \triangleleft \sigma_{\pi_1} \wedge \dots \wedge$ ˆ$\zeta_{|\pi_n} \triangleleft \sigma_{\pi_n}$。令 ˆ$\zeta'$ = ∅[π_1, ˆ$\zeta_{|\pi_1}$] . . . [π_n, ˆ$\zeta_{|\pi_n}$]。根据引理 6，我们知道 $X = Seq$(ˆζ')。根据 A_ψ 的正确性，这意味着 Seq(ˆζ') $\vDash_{LTL} \neg\varphi$，而根据引理 1，这意味着 ˆ$\zeta' \vDash \neg\varphi$。因此，ˆ $\zeta_{|\pi_1} \dots$ ˆ$\zeta_{|\pi_n}$是可行的反例轨迹，证明 $\forall \pi_1. \dots \forall \pi_n.\ \psi$ 不成立。

⇐ 假设 P不满足 φ。那么，存在轨迹 $\sigma_{\pi_1}, \dots \sigma_{\pi_n} \in \mathcal{L}(P)$ 和计算 $\zeta_{\pi_1}, \dots \zeta_{\pi_n}$，使得 $\zeta_{\pi_1} \triangleleft \sigma_{\pi_1} \wedge \dots \wedge \zeta_{\pi_n} \triangleleft \sigma_{\pi_n}$ 且 ˆζ' = ∅[π_1, ζ_{π_1}] . . . [π_n, ζ_{π_n}] $\vDash \neg\varphi$。根据引理 1，这意味着 Seq(ˆζ') $\vDash_{LTL} \neg\varphi$，因此 Seq(ˆζ') 被 $A_{\neg\varphi}$ 接受。令 σ_t =((σ_{π_1})$_{\pi_1 t}$; . . . ; (σ_{π_n})$_{\pi_n t}$) 和 $\sigma = \sigma_0 \sigma_1 \dots$。那么，combine($\sigma, Seq(\zeta)$) 是 $P \otimes A_{\neg\varphi}$ 的一条轨迹。根据引理 7，$\overline{\quad}(\zeta_{\pi_1}, \dots, \zeta_{\pi_n}) \triangleleft$combine($\sigma, Seq(\zeta')$)，因此这也是一条可行轨迹。

A.6 定理 5 的证明

现在我们证明定理 5。为此，我们需要以下引理：回忆一下，对于程序执行 σ_π，(σ_π)$_\pi$ 表示将 σ_π 中的每个单元 c 重命名为 c_π，并将每个输入 i 重命名为 i_π。

引理 8. 设 $\sigma \in \mathrm{Stmt}^{\omega}$ 可行，且 $\sigma_t = (((\sigma_{\pi_1})_{\pi_1})_t, \ldots, ((\sigma_{\pi_n})_{\pi_n})_t)$，其中 $\sigma_{\pi_1}, \cdots \sigma_{\pi_n}$。那么 $\sigma_{\pi_1}, \ldots, \sigma_{\pi_n}$ 也都可行。

ˆˆ 证明。由于 σ 可行，我们知道，对于某个 ζ，$\zeta \triangleleft \sigma$。对于所有 $1 \leq j \leq n$，我们定义 ζ_{π_j} 如下：

$$(\zeta_{\pi_j})_t = (\hat{\zeta}_{t \cdot n + j - 1})_{|(\mathbb{I} \cup \mathbb{C})_{\pi_j}}$$
$$\zeta_{\pi_j} = (\zeta_{\pi_j})_0 \, (\zeta_{\pi_j})_1 \ldots$$

其中，$\hat{a}_{|(\mathbb{I} \cup \mathbb{C})_{\pi_j}}$ 与之前一样，表示将赋值的定义域限制为标记为 π_j 的单元和输入，从而排除其他轨迹中的变量。此外，这些单元和输入再次从 c_{π_j} 重命名为 c，或从 i_{π_j} 重命名为 i。$t \cdot m + j - 1$ 是 σ 中 $(\sigma_{\pi_j})_t$ 的索引。

我们证明，对于所有时间点 t，$\zeta_{\pi_j} \triangleleft_t \sigma_{\pi_j}$

– 情况 $(\zeta_{\pi_j})_t$ = 断言 (τ_P)

ˆ我们知道 $\zeta \triangleleft_{t \cdot m + j - 1} \sigma$，因此

$$\eta(\hat{\tau_P}, \hat{\zeta}_{t \cdot m + j - 1}) = true \wedge \forall c \in \mathbb{C} \times \Pi.\ \hat{\zeta}_{t \cdot m + j - 1}(c) = \hat{\zeta}_{t \cdot m + j - 2}.$$

此外，$\eta(\tau_P, (\zeta_{\pi_j})_t)$ 也成立，因为 τ_P 不包含来自其他轨迹的变量。接下来只需证明

$$\forall c \in \mathbb{C}.\ ((\zeta_{\pi_j})_t)(c) = (\zeta_{\pi_j})_{t-1}(c)$$

ˆˆ 这也成立，因为 $\zeta_{(t-1) \cdot m + j - 1}, \ldots \zeta_{t \cdot m + j - 2}$ 中唯一被更改的单元是来自其他轨迹的单元。– 剩下的两种情况类似。

现在我们证明定理 5

证明。假设 $P^m \backslash (P^n \otimes A_\psi)^{\forall}_{k, C(k')}$ 有一条可行轨迹 σ，且满足 ˆ $\zeta \triangleleft \sigma$。根据引理 8，这意味着 $\zeta_{\pi_1} \triangleleft \sigma_{\pi_1} \wedge \cdots \wedge \zeta_{\pi_m} \triangleleft \sigma_{\pi_m}$。只需证明 $\varnothing[\pi_1, \zeta_1] \ldots [\pi_m, \zeta_m] \nvDash \exists \pi_{m+1}. \ldots \exists \pi_n. \psi$，因为这意味着 $\zeta_1, \ldots \zeta_m$ 是一个反例，证明 P 不满足 φ。

反证法。假设 $\varnothing[\pi_1, \zeta_1] \ldots [\pi_m, \zeta_m] \vDash \exists \pi_{m+1}. \ldots \exists \pi_n. \psi$。那么，存在轨迹 $\sigma_{\pi_{m+1}}, \ldots \sigma_{\pi_n}$ 和计算 $\zeta_{\pi_{m+1}} \ldots \zeta_{\pi_n}$，使得 $\zeta_{\pi_{m+1}} \triangleleft \sigma_{\pi_{m+1}} \wedge \cdots \wedge \zeta_{\pi_n} \triangleleft \sigma_{\pi_n}$ 且 ˆ $\zeta' = \varnothing[\pi_1, \zeta_{\pi_1}] \ldots [\pi_n, \zeta_{\pi_n}] \vDash \psi$。令 $\sigma'_t = ((\sigma_{\pi_1})_{\pi_1 t}; \ldots; (\sigma_{\pi_n})_{\pi_n t})$ 且 $\sigma' = \sigma'_0 \, \sigma'_1 \ldots$。现在，根据引理 1 以及 A_ψ 的正确性，我们知道 $Seq(\hat{\zeta}')$ 被 A_ψ 接受，因此 combine$(\sigma', Seq(\hat{\zeta}'))$ 被 $P^n \otimes A_\psi$ 接受。此外，根据引理 6，我们知道 combine$(\sigma', Seq(\hat{\zeta}'))$ 也是可行的，因此它也是 k-可行的，进而被 $(P^n \otimes A_\psi)_k$ 接受。此外，它不以不可行循环结束，因此也被 $(P^n \otimes A_\psi)_{k, C(k')}$ 接受。但这意味着 σ 被 $(P^n \otimes A_\psi)^{\forall}_{k, C(k')}$ 接受，因此不被 $P^m \backslash (P^n \otimes A_\psi)^{\forall}_{k, C(k')}$ 接受。矛盾。