arXiv:2301.10485v1 [cs.GT] 25 Jan 2023

少量提示下的 LTL反应式综合

Mrudula Balachander ⋆、Emmanuel Filiot ⋆⋆ 和 Jean-Fraņcois Raskin

布鲁塞尔自由大学，比利时布鲁塞尔 ⋆⋆⋆

摘要。 我们研究一种 Mealy 机综合问题的变体：综合出的机器必须能够针对环境的所有可能行为（包括敌对行为）执行 LTL 规范。在本文研究的变体中，用户提供待设计系统的高层 LTL 规范 φ，以及系统必须产生的一组执行示例 E 。我们的综合算法分两个阶段运行。首先，我们通过对自动机学习算法进行专门扩展，泛化示例 E 中采取的决策。此阶段在保留 φ可实现性的同时，对用户提供的示例 E 进行泛化。其次，算法将学习阶段得到的（通常不完整的）Mealy 机转化为能够实现 φ的完整 Mealy 机。示例用于引导综合过程。我们给出一个完备性结果，表明只需一组规模较小（多项式级）的示例，我们的过程就能学习任何能够实现 φ 的 Mealy 机 M 。我们还证明，我们的问题是经典 LTL 综合问题（即当 $E = \emptyset$时）的推广，其最坏情况下的复杂度与之相同。从 E 中学习所增加的成本，在 E 的规模以及能够实现 φ的解的符号表示规模方面，也仅为多项式级。当综合算法 Acacia-Bonzai 求解普通 LTL 综合问题时，会计算出这一符号表示。最后，我们通过一组示例展示了该方法的实际价值。

1 引言

反应式系统众所周知难以设计，甚至难以正确规约 [1,15]。因此，形式化方法应运而生，成为帮助设计人员构建正确反应式系统的有用工具。例如，模型检测要求设计人员提供一个模型，以 Mealy 机$\mathcal{M}$的形式描述系统对其环境所产生事件的响应，并同时描述必须满足的核心正确性性质 。这些性质以逻辑形式表达，通常写成 LTL公式 φ_{CORE}。随后，算法会判定 $\mathcal{M} \models \varphi_{\mathsf{CORE}}$ 是否成立，也就是说，系统在其环境中的所有执行是否都满足该规约。自动反应式综合则更进一步：它旨在自动生成

⋆ Mrudula Balachander：Mrudula Balachander 是 F.R.S-FNRS 的研究员。⋆⋆ Emmanuel Filiot：Emmanuel Filiot 是 F.R.S-FNRS 的高级研究助理。
⋆⋆⋆ 本研究部分获得科学研究基金会（F.R.S.- FNRS）MIS项目F451019F及ASP-Fellowship资助的支持。

一个模型，该模型根据对“做什么”的高层描述生成，而非描述“如何”去做。因此，用户只需提供一份 LTL 规范 φ，算法便会自动生成一台 Mealy 机 $\mathcal{M}$，使得 $\mathcal{M} \models \varphi$ 在 φ 可实现时成立。遗憾的是，大多数情况下，仅提供核心正确性性质 φ_{CORE} 并不足以获得一台在实际中有用的 Mealy 机 $\mathcal{M}$，下文将对此加以说明。

示例*1*。 [从 φ_{CORE} - 互斥问题的综合] 我们来考虑经典的互斥问题。对于这个问题最简单的形式，我们需要设计一个仲裁器，它接收来自两个进程的请求，这两个进程由受环境控制的两个原子命题 r_1 和 r_2 建模；仲裁器还要授权访问临界区，临界区由受系统控制的两个原子命题 g_1和 g_2 建模。核心正确性性质（即要实现什么）包括：(i) 互斥访问，即绝不会同时向两个进程授予访问权限；(ii) 公平性，即已请求访问的进程最终能够访问临界区。这些互斥问题（ME）的核心正确性规约可以很容易地用线性时序逻辑表示如下：$\varphi^{\mathsf{ME}}_{\mathsf{CORE}} \equiv \square(\neg g_1 \vee \neg g_2) \wedge \square(r_1 \rightarrow \Diamond g_1) \wedge \square(r_2 \rightarrow \Diamond g_2)$。事实上，无论如何 $\mathcal{M}$ 实现互斥，例如采用 Peterson、Dedekker、Backery 等算法，我们都会对该公式进行模型检验，以验证核心正确性性质。不幸的是，如果我们将 $\varphi^{\mathsf{ME}}_{\mathsf{CORE}}$ 提交给由 Acacia-Bonzai [10]、 BoSy [19] 或 Strix [27]等工具实现的线性时序逻辑综合过程，就会得到 $\mathcal{M}$ 中 1-(左)所示的解（这三个工具返回的都是这个解）。虽然该解完全正确，也满足规约 $\varphi^{\mathsf{ME}}_{\mathsf{CORE}}$，但它忽略了来自环境的输入，并以轮询方式向临界区授予访问权限。可以说，它未必算得上是互斥问题的高效解法。这说明，仅提供问题的核心正确性规约，也就是只说明要实现什么，会限制综合算法解决设计问题的能力。要为互斥问题生成实用的解，还需要提供更多指导。

现在的主要问题是：我们应该如何指定这些额外的性质？显然，如果我们想使用“纯粹的”LTL 综合算法，就别无选择：我们需要用额外的低层属性来强化规约 $\varphi^{\mathsf{ME}}_{\mathsf{CORE}}$ $\varphi^{\mathsf{ME}}_{\mathsf{LOW}}$。让我们回到运行示例。

示例*2*。 [综合自 $\varphi^{\mathsf{ME}}_{\mathsf{CORE}}$ 和 $\varphi^{\mathsf{ME}}_{\mathsf{LOW}}$] 为了避免出现未经请求的授权的解决方案，我们需要强化核心规约。Strix在线演示网站建议向 $\varphi^{\mathsf{ME}}_{\mathsf{CORE}}$ 添加以下3个线性时序逻辑公式 $\varphi^{\mathsf{ME}}_{\mathsf{LOW}}$ $\varphi^{\mathsf{ME}}_{\mathsf{CORE}}$（见 Full arbitrer$n = 2$，网址：https://meyerphi.github.io/strix-demo/）：(1) $\bigwedge_{i\in\{1,2\}} \square((g_i \wedge \square\neg r_i) \rightarrow \Diamond\neg g_i)$，(2) $\bigwedge_{i\in\{1,2\}} \square(g_i \wedge \bigcirc(\neg r_i \wedge \neg g_i) \rightarrow \bigcirc(r_i \mathrm{R} \neg g_i))$，以及 (3)$\bigwedge_{i\in\{1,2\}}(r_i \mathrm{R} \neg g_i)$。现在，虽然规约 $\varphi^{\mathsf{ME}}_{\mathsf{CORE}} \wedge \varphi^{\mathsf{ME}}_{\mathsf{LOW}}$ 能让 Strix 为我们提供更好的解决方案，但它比所需的复杂得多（它有9个状态，可见附录C），而且显然不像是互斥问题的最优解。例如，图 1（右）中的模型可以说更为自然。我们如何才能得到这个模型，而不必将它编码进

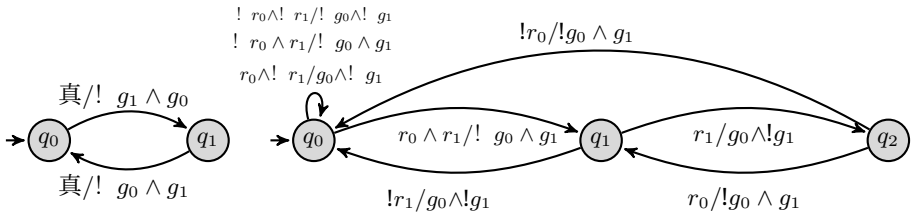


图 1：（左）Strix 针对高层规约说明 $\varphi^{\mathsf{ME}}_{LOW}$ 给出的互斥问题解决方案。边标记的形式为 φ/ψ，其中 φ 是关于输入原子命题（由环境控制的布尔变量）的布尔公式，而 ψ 是输出命题集合（由系统控制的布尔变量）上的文字的极大一致合取。（右）一种自然的解决方案，也就是我们手写时会给出的方案；对于同一规约说明，结合两个简单示例，我们的学习与综合算法会自动生成该方案。

线性时序逻辑规范中？这样做会大大削弱一开始使用综合过程的意义。

一般而言，高层属性关注安全性，需要在所有实现上进行验证。相比之下，低层属性更多涉及特定实现，也就是说，它们更多地描述预期行为，并关注实现的效率。此时，质疑线性时序逻辑作为低层属性的规范语言是否恰当，并将其用作引导综合过程实现 φ_{CORE} 的相关解决方案，是合理的。本文介绍了一种引导综合生成有用解决方案以实现 φ_{CORE} 的替代方法：我们提出使用执行示例来展示预期解决方案的行为。随后，我们将搜索范围限制在能够泛化这些示例的解决方案上。在需求工程中，示例或执行场景被视为一种合适的工具，可用于获取有关复杂系统的需求 [14]。对于反应式系统设计，示例尤其适用，因为它们通常比完整解决方案甚至部分解决方案更容易编写。这是因为在编写示例时，用户可以同时控制输入和输出，从而避开反应式系统设计的主要难题：必须应对所有可能的环境输入。我们将在运行示例中对此进行说明。

例*3.* [根据 $\varphi^{\mathsf{ME}}_{\mathsf{CORE}}$ 和示例进行综合] 让我们保留 $\varphi^{\mathsf{ME}}_{\mathsf{CORE}}$ 作为线性时序逻辑规格，仅考虑它，并考虑以下简单的执行前缀，以说明互斥问题的解决方案应如何运行：

(1) $\{!r_1, !r_2\}.\{!g_1, !g_2\}\#\{r_1,!r_2\}.\{g_1,!g_2\}\#\{!r_1, r_2\}.\{!g_1, g_2\}$(2) $\{r_1, r_2\}.\{g_1,!g_2\}\#\{!r_1, !r_2\}.\{!g_1, g_2\}$

这些轨迹前缀规定了对典型的固定有限输入序列的响应：(1) 如果最初没有请求，则不授予任何访问权限（注意，这排除了已有的轮询解）；如果进程 1 发出请求，随后

进程 2 发出请求，则先授予进程 1 访问权限，之后再授予进程 2；(2)如果两个进程同时发出请求，则先授予进程 1 访问权限，之后再授予进程 2。给定这两条简单轨迹以及 φ_{CORE}，我们的算法生成了图 1（右）所示的解。可以说，这个解现在既简单又自然。

贡献首先，我们提出一种综合算法 SynthLearn：给定一个 LTL 规范 φ_{CORE} 和一个由执行前缀组成的有限集 E，若存在一台 Mealy 机 $\mathcal{M}$，使得 $\mathcal{M} \models \varphi_{\mathsf{CORE}}$（即 $\mathcal{M}$ 实现了 φ_{CORE}），且 $E \subseteq$Prefix($L(\mathcal{M})$)（即 $\mathcal{M}$ 与 E中的示例相兼容），则返回该机器。否则返回不可实现。此外，我们要求 SynthLearn 能够泛化 E所展示的决策。这一可学习性要求通常在自动机学习中通过完备性准则形式化；我们在此将其调整如下：对于所有规约 φ_{CORE}，以及所有满足 $\mathcal{M} \models \varphi_{\mathsf{CORE}}$的 Mealy 机 $\mathcal{M}$，都存在一个较小的示例集 E（其大小关于 $|\mathcal{M}|$为多项式），使得 L(SynthLearn(φ_{CORE}, E)) $= L(\mathcal{M})$。我们针对安全规约在定理 4 中证明了这一完备性结果，并在第4节通过归约到安全规约，将其扩展到ω-正则规约和 LTL 规约。

其次，我们证明 SynthLearn 的最坏情况执行时间为 2Ex-pTime（定理 7）；而这在最坏情况下是最优的，因为已知普通线性时序逻辑综合问题（当 $E = \emptyset$时）已经是 2ExpTime-Complete [29]。SynthLearn 首先将用户提供的示例进行泛化，同时保持 φ_{CORE} 的可实现性。这种泛化会生成一台可能存在缺失转移的 Mealy 机（称为 preMealy 机）。随后，将这台 preMealy 机扩展为一台（完整的）Mealy 机，使其能够针对环境的所有行为实现 φ_{CORE}。在补全阶段，SynthLearn会尽可能复用从示例中泛化得到的决策。泛化阶段对于充分利用示例至关重要。对 $\varphi_{\mathsf{CORE}} \wedge \varphi_E$ 运行经典综合算法（其中 φ_E 是 E 的线性时序逻辑编码），通常会得到更复杂的机器；这些机器无法泛化沿着 E 中示例所作出的决策。虽然 Synth-Learn 的总体复杂度为 2ExpTime，且是最优的，但我们证明，若采用精心选择的符号表示，则其复杂度仅关于 E 的大小以及一组实现 φ_{CORE} 的 Mealy 机呈多项式增长，见定理 6。这种符号表示采用函数反链的形式，并且在实践中往往十分紧凑 [21]。当 Acacia-Bonzai 求解 φ_{CORE} 的普通线性时序逻辑综合问题时，默认会计算这种表示。因此，在保持可实现性的同时泛化示例，只会带来边际的多项式代价。我们已将综合算法实现为一个原型，该原型使用 Acacia-Bonzai 计算符号反链表示。我们报告了在若干示例上取得的结果。

相关工作 研究人员一直倡导使用执行场景来引出规约，尤其是在需求工程领域，参见例如 [14,16] 及其中的参考文献。在 [30],中，学习技术被用于将示例转化为能够对其进行泛化的线性时序逻辑公式。这些方法与我们的工作互为补充，因为它们可用于获得高层规格说明 φ_{CORE}。

在非空洞综合中 [7], , 会自动向 LTL 规范添加示例，以迫使综合过程生成符合 [25]所指含义、非空洞的解。这些示例直接根据 LTL 规范的语法生成，且不能由用户提出。因此，我们的方法与这种方法相互独立且互为补充。事实上，我们可以使用非空洞方法自动生成的示例，并请用户判断这些示例是否符合期望。我们的方法更加灵活，是半自动且以用户为中心的：用户可以提供任何自己喜欢的示例，因此能够更灵活地引导综合过程，得到用户认为有意义的解。此外，我们的综合过程基于学习算法，而 [7] 中的算法基于约束求解，并且无法像我们的算法那样提供泛化保证（参见定理 4）。

在正式规约中补充用户提供的额外信息，是语法引导综合 框架（SyGuS [3]）的核心；例如，该框架已在程序草图 [33]中实现：在 SyGuS 中，规约是一个逻辑公式，而候选程序受用户提供的语法限制，以缩小并引导搜索。搜索通过依赖于学习 [34] 的反例引导归纳综合技术（CEGIS）完成。与我们的方法不同，示例并非由用户提供，而是通过针对规约对候选程序进行模型检测而自动生成。这些技术与我们的方法也相互独立：SyGuS 面向由可判定背景理论上的表达式在语法上定义的程序，并且高度依赖 SAT/SMT 求解器。利用示例合成程序（示例编程）的做法，例如已在电子表格字符串处理程序这一背景下得到探索，其基础是学习 [32], ；这也是人工智能领域当前的趋势（例如参见 [28] 及其中的引文）。不过，这种方法仅依赖示例，而不依赖逻辑规约。

[4] 探讨了使用形式化规范和场景来合成分布式协议。他们的方法也分为两个阶段：首先根据场景构建一个不完整的机器，然后将其转换为完整机器。但他们的工作与我们的工作有两个重要区别。首先，他们的第一阶段不依赖学习技术，也不尝试对给定的示例进行泛化。其次，在他们的设定中，所有动作都是可控的，不存在对抗环境，因此他们解决的是可满足性问题，而不是像我们这样解决可实现性问题。因此，他们的问题在计算上不如我们解决的问题复杂：对于线性时序逻辑规约， Pspace 与 2ExpTime。

本文研究的综合问题是对 LTL 综合问题的扩展。该问题的现代解决方案采用避免 Safra 构造的自动机构造，这种方法最初由 [26], 提出，经 [31,20], 简化，并于近期由 [18] 进一步发展。已有高效的无 Safra 构造实现，例如参见 [8,19,27,17]。此前有多项研究提出了替代方法，以提升综合算法所能提供的解决方案质量。一种与本文提出的方法正交且互补的热门研究方向是

通过在形式化规范中加入定量方面来扩展规约，例如参见 [5,9,24,2], ，并且只综合最优解决方案。

我们算法的第一阶段受到基于状态合并算法（如 RPNI [23,22]）的自动机学习技术启发。要生成能够保留 φ_{CORE} 可实现性的部分解，必须谨慎地修改这些学习算法。在此背景下，要证明补全阶段的完备性和终止性，都需要格外谨慎。

2 反应式综合问题的预备知识

词、语言与自动机 字母表是由有限个符号构成的集合。字母表 Σ 上的词 u（分别为 ω-词）是由 Σ 中的符号组成的有限序列（分别为无限序列）。我们用 ϵ 表示空词，并用 $|u| \in \mathbb{N} \cup \{\infty\}$ 表示 u 的长度。特别地，$|\epsilon| = 0$。对于 $1 \leq i \leq j \leq |u|$，我们令 $u[i{:}j]$ 表示从位置 i 到位置 j（包括两端）的 u 子串，并用 $u[i]$ 代替 $u[i{:}i]$。字母表 Σ 上的有限词（分别为 ω-词）集合记作 Σ^*（分别为 Σ^ω）。我们令 $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$。给定两个词 $u \in \Sigma^*$ 和 $v \in \Sigma^\infty$，若存在某个 $w \in \Sigma^\infty$ 使得 $v = uw$，则 u 是 v 的前缀，记作 $u \preceq v$。v 的前缀集合记作 $\mathsf{Prefs}(v)$。假设字母表 Σ 上有一个线性序 $<_\Sigma$，有限词按长度字典序 $\preceq_{ll}$ 排列：若 $|u| < |v|$，或者 $|u| = |v|$ 且 $u = p\sigma_1 u'$，其中 $v = p\sigma_2 v'$，则 $u \preceq_{ll} v$，这里 p、u'、$v' \in \Sigma^*$，且 $\sigma_1 <_\Sigma \sigma_2$。本文中，凡是提及字母表上词的序 $\preceq_{ll}$，都默认该字母表上存在任意一个线性序。字母表 Σ 上的语言（分别为 ω-语言）是 $L \subseteq \Sigma^*$（分别为 $L \subseteq \Sigma^\omega$）的子集。

在本文中，我们固定两个字母表 $\mathcal{I}$ 和 $\mathcal{O}$，其元素分别称为输入和输出。给定一个词 $u \in (\mathcal{I}\mathcal{O})^\infty$，我们令 $\mathsf{in}(u) \in \mathcal{I}^\infty$ 表示从 u中删除所有 $\mathcal{O}$-符号后得到的词。我们类似地定义 $\mathsf{out}(u)$，并将这两个函数自然地扩展到语言上。

关于 ω-词的自动机 奇偶自动机是一个元组 $\mathcal{A} = (Q, Q_{\mathsf{init}}, \Sigma, \delta, d)$，其中 Q 是一个有限非空状态集，$Q_{\mathsf{init}} \subseteq Q$ 是初始状态集，Σ是一个有限非空字母表，$\delta\colon Q \times \Sigma \to 2^Q \setminus \{\emptyset\}$ 是转移函数，$d\colon Q \to \mathbb{N}$ 是奇偶函数。当且仅当 $|Q_{\mathsf{init}}| = 1$且对所有 $q \in Q$都有 $|\delta(q,\sigma)| = 1$ 时，自动机 $\mathcal{A}$ 是确定性的。转移函数自然地扩展为函数 $\mathsf{Post}^*\colon Q \times \Sigma^* \to 2^Q \setminus \{\emptyset\}$，归纳定义如下：对所有 $q \in Q$ 以及所有 $(u,\sigma) \in \Sigma^* \times \Sigma$，$\mathsf{Post}^*(q,\epsilon) = \{q\}$；$\mathsf{Post}^*(q,u\sigma) = \bigcup_{q' \in \mathsf{Post}^*(q,u)} \delta(q',\sigma)$。

$\mathcal{A}$在一个 ω-词 $w = w_0 w_1 \ldots$上的运行，是一个状态$r = q_0 q_1 \ldots$的无限序列，满足 $q_0 \in Q_{\mathsf{init}}$，并且对所有 $i \in \mathbb{N}$，$q_{i+1} \in \delta(q_i, w_i)$。如果运行 r无限次访问到的最小颜色为偶数，即 $\mathsf{liminf}(d(q_i))_{i \geq 0}$ 为偶数，则称其为接受运行。若 $\mathsf{dom}(d) = \{0,1\}$（颜色为 1 的状态称为接受状态），则称 $\mathcal{A}$ 为 B ¨uchi 自动机；若 $\mathsf{dom}(d) = \{1,2\}$，则称其为余 B ¨uchi 自动机；若¨为一个 B ¨uchi 自动机，且颜色为 1 的状态集合（称为不安全状态，记作 Q_{usf}）构成一个陷阱，即对所有 $q \in Q_{\mathsf{usf}}$及所有 $\sigma \in \Sigma$，都有 $\delta(q,\sigma) \subseteq Q_{\mathsf{usf}}$，则称其为安全自动机；若$\{0,1\}$为$\{0,1\}$-着色，且颜色为 0 的状态集合构成一个陷阱，则称其为可达性自动机。

最后，我们考察非确定性的存在解释和普适解释，由此得到两种不同的 ω-词语言概念：在存在（分别为普适）解释下，若存在一个在 w 上的运行 r，使得 r 是接受运行（分别地，对 w上的所有运行 r， r都是接受运行），则词 $w \in \Sigma^\omega$ 属于 $\mathcal{A}$的语言。我们分别用$L^\exists(\mathcal{A})$和 $L^\forall(\mathcal{A})$表示这两种解释定义的语言。注意，若 $\mathcal{A}$ 是确定性的，则存在解释和普适解释等价，因此我们将 $L(\mathcal{A}) = L^\exists(\mathcal{A})$记作 $L^\forall(\mathcal{A})$。有时，对于一个确定性自动机 $\mathcal{A}$，我们会将初始状态改为状态 $q \in Q$，并用 $\mathcal{A}[q]$ 表示初始状态固定为单元素集合 $\{q\}$的确定性自动机 $\mathcal{A}$。

对于$co\text{-}B\ddot{u}chi$ 自动机，我们还定义了一种更强的接受条件，称为 K-co-B¨uchi。该条件要求，对于 $K \in \mathbb{N}$，一次运行只有在访问标记为 1 的状态不超过K次时才被接受。形式化地说，若运行 $r = q_0 q_1 \dots q_n \dots$满足 $|\{i \geq 0 \mid d(q_i) = 1\}| \leq K$，则它是满足 K-co-B¨uchi 接受条件的接受运行。在普适解释下，由 $\mathcal{A}$ 针对 K-co-B¨uchi 接受条件定义的语言记作 $L_K^\forall(\mathcal{A})$。请注意，该语言是安全性语言，因为如果词 $p \in \Sigma^*$ 的某个前缀满足以下条件： $\mathcal{A}$ 在p 上有一个运行前缀，该前缀访问了标记为颜色 1 的状态超过 K 次，那么 p 的所有可能延伸 $w \in \Sigma^\omega$ 都会被 $\mathcal{A}$拒绝。

（预）*Mealy*机给定一个从集合 X 到集合 Y的（偏）函数 f，我们用 dom(f) 表示其定义域，即满足 $f(x)$有定义的元素 $x \in X$ 所组成的集合。输入字母表为 $\mathcal{I}$、输出字母表为$\mathcal{O}$的***preMealy*** 机 $\mathcal{M}$ 是一个三元组$(M, m_{\mathsf{init}}, \Delta)$，其中 M是非空状态集， $m_{\mathsf{init}} \in M$是初始状态， $\Delta\colon Q \times \mathcal{I} \to \mathcal{O} \times M$是一个偏函数。若一对$(m, \mathsf{i})$不属于 dom($\Delta$)，则它是 $\mathcal{M}$ 中的一个空缺。一台***Mealy*** 机是满足 Δ为全函数（即 dom(Δ) $= M \times \mathcal{I}$）的 preMealy 机。

我们根据 preMealy 机 $\mathcal{M} = (M, m_{\mathsf{init}}, \Delta)$ 所定义的、由 $\mathcal{I} \cup \mathcal{O}$ 上的有限字和无限字构成的语言，定义它的两种语义。首先，我们定义两个（可能为偏函数） $\mathsf{Post}_\mathcal{M}\colon M \times \mathcal{I} \to M$ 和 $\mathsf{Out}_\mathcal{M}\colon M \times \mathcal{I} \to \mathcal{O}$，使得对于所有满足 $\Delta(m, \mathsf{i})$ 有定义的 $(m, \mathsf{i}) \in M \times \mathcal{I}$，都有 $\Delta(m, \mathsf{i}) = (\mathsf{Post}_\mathcal{M}(m, \mathsf{i}), \mathsf{Out}_\mathcal{M}(m, \mathsf{i}))$。我们自然地将这两个函数扩展到任意输入序列 $u \in \mathcal{I}^+$，分别记为 $\mathsf{Post}^*_\mathcal{M}$ 和 $\mathsf{Out}^*_\mathcal{M}$。特别地，对于 $u \in \mathcal{I}^+$，$\mathsf{Post}^*_\mathcal{M}(m, u)$ 是 $\mathcal{M}$ 从 m 开始读取 u 后到达的状态，而 $\mathsf{Out}^*_\mathcal{M}(m, u)$ 是 $\mathcal{M}$ 读取 u 时产生的 $\mathcal{O}$ 中的最后一个输出。当 $\mathcal{M}$ 的含义可从上下文中明确时，省略下标 $\mathcal{M}$。现在， $\mathcal{M}$ 接受的、属于 $(\mathcal{I}\mathcal{O})^*$ 的有限字的语言 $L(\mathcal{M})$ 定义为： $L(\mathcal{M}) = \{\mathsf{i}_1\mathsf{o}_1 \dots \mathsf{i}_n\mathsf{o}_n \mid \forall 1 \leq j \leq n$，其中 $\mathsf{Post}^*_\mathcal{M}(m_{\mathsf{init}}, \mathsf{i}_1 \dots \mathsf{i}_j)$ 有定义，且 $\mathsf{o}_j = \mathsf{Out}^*_\mathcal{M}(m_{\mathsf{init}}, \mathsf{i}_1 \dots \mathsf{i}_j)\}$。 $\mathcal{M}$ 接受的无限字语言 $L_\omega(\mathcal{M})$ 是 $L(\mathcal{M})$ 的拓扑闭包： $L_\omega(\mathcal{M}) = \{w \in (\mathcal{I}\mathcal{O})^\omega \mid \mathsf{Prefs}(w) \cap (\mathcal{I}\mathcal{O})^* \subseteq L(\mathcal{M})\}$。

反应式综合问题 规约是一种语言 $\mathcal{S} \subseteq (\mathcal{I}\mathcal{O})^\omega$。反应式综合问题 （简称综合问题）是指：给定一个规约 $\mathcal{S}$，构造一台 Mealy 机 $\mathcal{M}$，使得 $L_\omega(\mathcal{M}) \subseteq \mathcal{S}$（若存在）。这样的机器 $\mathcal{M}$ 称为实现 该规约 $\mathcal{S}$，也写作

$\mathcal{M} \models \mathcal{S}$。如果存在某台 Mealy 机 $\mathcal{M}$ 能够实现规约 $\mathcal{S}$，我们也称 $\mathcal{S}$ 是可实现 的。由此得到的判定问题称为可实现性问题。

众所周知，如果 $\mathcal{S}$ 是 ω-正则的（可由奇偶自动机[35]识别），则可实现性问题是可判定的 [1]，而且可以有效地构造出一个实现该规约的 Mealy 机。若 $\mathcal{S}$ 是以 LTL公式 [29]给出的，则可实现性问题是 2ExpTime-Complete；若 $\mathcal{S}$ 是以普适 coB¨uchi 自动机给出的，则 ExpTime-Complete 。

定理 1 ([6])。 **</style}规约 若以普适 coBuchi 自动机给出，则其可实现性问题是。此外，如果 可实现且 有 个状态，那么 可由一个具有)**个状态的 *Mealy* 机实现。

我们将这一结果推广到以下可实现性问题，并先对其作非正式描述。给定一个规约 $\mathcal{S}$ 和一台 preMealy 机 $\mathcal{P}$，目标是判断能否将 $\mathcal{P}$ 补全为一台实现 $\mathcal{S}$的 Mealy 机。现在我们正式定义这个问题。给定两台 preMealy 机$\mathcal{P}_1$、$\mathcal{P}_2$，若 $\mathcal{P}_1$ 按照以下意义是 $\mathcal{P}_2$ 的子图，我们记作 $\mathcal{P}_1 \preceq \mathcal{P}_2$：存在一个从 $\mathcal{P}_1$ 的状态到 $\mathcal{P}_2$ 的状态的单射映射 Φ，该映射保持初始状态（s_0 是 $\mathcal{P}_1$ 的初始状态，当且仅当 $\Phi(s_0)$ 是 $\mathcal{P}_2$的初始状态）以及转移（$\Delta_{\mathcal{P}_1}(p,\mathrm{i}) = (\mathrm{o}\ q)$，当且仅当 $\Delta_{\mathcal{P}_2}(\Phi(p),\mathrm{i}) = (\mathrm{o}\ \Phi(q))$）。因此，$L(\mathcal{P}_1) \subseteq L(\mathcal{P}_2)$ 且 $L_\omega(\mathcal{P}_1) \subseteq L_\omega(\mathcal{P}_2)$。给定一台 preMealy 机$\mathcal{P}$，如果存在一台 Mealy 机$\mathcal{M}$ 满足 $\mathcal{P} \preceq \mathcal{M}$ 且 $\mathcal{M}$ 实现 $\mathcal{S}$，我们就称规约 $\mathcal{S}$ 是 $\mathcal{P}$-可实现的。注意，如果 $\mathcal{P}$ 是一台（完全）Mealy 机，那么 $\mathcal{S}$ 是 $\mathcal{P}$-可实现的，当且仅当 $\mathcal{P}$ 实现 $\mathcal{S}$。

定理 2. 给定一个定义规约 $\mathcal{S} = L^\forall(\mathcal{A})$ 的普适 *co-B¨uchi* 自动机 $\mathcal{A}$ （有 n 个状态），以及一个有 m 个状态和 n_h 个空缺的 ***preMealy*** 机 $\mathcal{P}$，判定 $\mathcal{S}$ 是否 $\mathcal{P}$-可实现是 ExpTime-困难的，且属于 ExpTime （在 n 中，且关于 m 为多项式时间）。此外，若 $\mathcal{S}$ 是 $\mathcal{P}$-可实现的，则它可由一个有 $m + n_h 2^{O(nlog_2 n}$ 个状态的 ***Mealy*** 机 $\mathcal{P}$-实现。即使 $\mathcal{P}$ 只有两个状态且 $\mathcal{A}$ 是确定性可达自动机，该问题仍然是困难的。

将证明从附录移至此处。在证明定理 2 之前，我们先指出，$\mathcal{P}$-可实现性问题是经典可实现性问题的推广，因为后者等价于 $\mathcal{P}_0$-可实现性问题，其中 $\mathcal{P}_0$ 是一个仅由单个状态（即初始状态）组成且没有任何转移的 preMealy 机。因此，我们沿用定理 1 的ExpTime 下界。不过，我们证明了 $\mathcal{P}$-可实现性问题本质上更难：事实上，我们表明，即使 $\mathcal{P}$ 是固定的 preMealy 机，且 $\mathcal{S}$ 由确定性可达自动机给出， ExpTime 困难性仍然成立。这与经典可实现性问题形成对比：对于以确定性可达自动机给出的规约，判定其是否可实现属于 PTime [11]。我们的规约与示例综合算法大量依赖于对不同 preMealy 机 $\mathcal{P}$反复调用 $\mathcal{P}$-可实现性检查器。不过，我们在第 4 节中表明，若预先计算规约自动机的一些可实现配置的符号表示（最坏情况下需指数时间，但在实践中较为紧凑），那么所有这些调用都可以在该表示上的多项式时间内完成。

定理 2 的证明。我们首先证明上界。令 $Q_{\mathcal{P}}$ 为 $\mathcal{P}$的状态集合， $\Delta_{\mathcal{P}}$ 为其转移函数， p_0 为其初始状态。对于任意 $p \in Q_{\mathcal{P}}$，我们将其左语言Left_p 定义为

$$\mathsf{Left}_p = \{u \in (I.O)^* \mid \mathsf{Post}^*_{\mathcal{P}}(p_0, u) = p\}$$

那么， $\mathcal{P}$-可实现性可由以下性质刻画：

断言。 $\mathcal{S}$是 $\mathcal{P}$-可实现的，当且仅当 $L_\omega(\mathcal{P}) \subseteq \mathcal{S}$，并且对于 P的每个空缺 $h = (p,$ i)，都存在 $\mathsf{o}_h \in \mathcal{O}$ 和一台 Mealy 机 $\mathcal{M}_h$，使得对于所有 $u \in \mathsf{Left}_p$， $\mathcal{M}_h$ 都实现$(u\mathsf{io}_h)^{-1}\mathcal{S}$。[1]

断言的证明。对于“若”这一方向，我们证明 $\mathcal{P}$ 可以扩展为一台 Mealy 机 $\mathcal{M}$，该机器按如下方式 $\mathcal{P}$-实现 $\mathcal{S}$： $\mathcal{M}$ 由所有 $\mathcal{P}$的空缺 h 对应的 $\mathcal{P}$ 作不交并组成，并与 Mealy 机 $\mathcal{M}_h$合并，再添加转移 $\Delta_{\mathcal{M}}(h) = (\mathsf{o}_h, \mathsf{init}_h)$，其中 init_h 是 $\mathcal{M}_h$的初始状态。显然，$\mathcal{P}$ 是 $\mathcal{M}$的子图。我们证明 $\mathcal{M}$ 实现 $\mathcal{S}$。令 $w \in L_\omega(\mathcal{M})$。假设 $w \notin \mathcal{S}$ 成立，并由此导出矛盾。由于 $L_\omega(\mathcal{P}) \subseteq L^\forall(\mathcal{A})$，因此 $w \notin L_\omega(\mathcal{P})$。这意味着 $\mathcal{M}$ 在 w 上的执行必然会经过 $\mathcal{P}$的一个空缺 $h = (p,$ i)。因此， w 可以分解为 $w = u\mathsf{io}_h v$，其中 u 是 w 满足 $u \in \mathsf{Left}_p$的最长前缀。由于 $w \notin \mathcal{S}$，我们得到 $v \notin (u\mathsf{io}_h)^{-1}\mathcal{S}$。根据 $\mathcal{M}$的定义，我们有 $v \in L_\omega(\mathcal{M}_h)$，因此 $\mathcal{M}_h$ 并未实现$(u\mathsf{io}_h)^{-1}\mathcal{S}$，这就产生了矛盾。

反过来，假设存在某个 Mealy 机 $\mathcal{M}$，使得 $\mathcal{S}$ 可由其实现为 $\mathcal{P}$。由于$L_\omega(\mathcal{P}) \subseteq L_\omega(\mathcal{M})$ 且 $L_\omega(\mathcal{M}) \subseteq \mathcal{S}$，我们得到 $L_\omega(\mathcal{P}) \subseteq \mathcal{S}$。现在，考虑一个空缺$h = (p,$ i)。由于 $\mathcal{P}$ 是 $\mathcal{M}$的子图， p 是 $\mathcal{M}$ 的状态，并且 $\Delta_{\mathcal{M}}$ 是完全的，因此存在 $\mathsf{o}_h \in \mathcal{O}$，使得对于 $\mathcal{M}$的某个状态 p'， $\Delta_{\mathcal{M}}(h) = (\mathsf{o}_h, p')$。考虑机器 $\mathcal{M}_{p'}$，它与 $\mathcal{M}$ 完全相同，只是其初始状态为 p'： $\mathcal{M}_{p'}$ 是一个 Mealy 机，对所有 $u \in \mathsf{Left}_p$，它都实现 $(u\mathsf{io}_h)^{-1}\mathcal{S}$。事实上，令 $v \in L_\omega(\mathcal{M}_{p'})$。根据 $\mathcal{M}_p$的定义，我们有 $u\mathsf{io}_h v \in L_\omega(\mathcal{M}) \subseteq \mathcal{S}$。因此，$v \in (u\mathsf{io}_h)^{-1}\mathcal{S}$。. □

剩下需要证明的是，断言的刻画可以在ExpTime中判定。首先，判定 $L_\omega(\mathcal{P}) \subseteq L^\forall(\mathcal{A}) = \mathcal{S}$ 是否成立，是一个标准的自动机包含问题。事实上， $\mathcal{P}$ is 可以视为一个确定性的 B¨uchi 自动机，其中所有状态都是接受状态；而 $\mathcal{A}$ is 是一个普适的 co-B¨uchi 自动机，可以在线性时间内将其补集化为一个非确定性的 B¨uchi 自动机 $\mathcal{B}$。接着，只需检验 $L_\omega(\mathcal{P}) \cap L^\exists(\mathcal{B}) = \varnothing$。这可以在关于两台机器大小的 PTime 内完成。因此，检验 $L_\omega(\mathcal{P}) \subseteq L^\forall(\mathcal{A}) = \mathcal{S}$是否成立，可以在 PTime内完成。

现在，我们要判定该刻画的第二部分。注意，给定一个空缺 $h = (p,$ i) 和 $\mathsf{o}_h \in \mathcal{O}$，存在一台 Mealy 机 $\mathcal{M}_h$，使得对于所有 $u \in \mathsf{Left}_p$， $\mathcal{M}_h$ 实现 $(u\mathsf{io}_h)^{-1}\mathcal{S}$，当且仅当规约 $\bigcap_{u \in \mathsf{Left}_p}(u\mathsf{io}_h)^{-1}\mathcal{S}$ 是可实现的。给定 h 和 o_h，我们在线性时间内构造一个普适的 co-B¨uchi 自动机，用于识别 $\bigcap_{u \in \mathsf{Left}_p}(u\mathsf{io}_h)^{-1}\mathcal{S}$。首先，我们计算状态集合

$$R_p^{\mathcal{A},\mathcal{P}} = \{q \in Q_{\mathcal{A}} \mid \exists u \in (I.O)^*, \mathsf{Post}^*_{\mathcal{P}}(p_0, u) = p \wedge \mathsf{Post}^*_{\mathcal{A}}(q_0, u\mathsf{io}_h) = q\}$$

[1] 对于字母表 Σ、集合 $A \subseteq \Sigma^\omega$ 和 $u \in \Sigma^*$、 $u^{-1}A = \{v \in \Sigma^\omega \mid uv \in A\}$。

这可以在多项式时间内完成。接着，我们定义普适 co-B¨uchi 自动机，记作 $\mathcal{A}_p$，它恰好是 $\mathcal{A}$，其中初始状态集合设为 $R^{\mathcal{A},\ p}{}^{\mathcal{P}}$。我们有 $L^{\forall}(\mathcal{A}_p) = \bigcap_{u\in左_{\mathcal{P}}}(u\mathsf{io}_h)^{-1}\mathcal{S}$，然后使用定理 1 来判定：在ExpTime 中，关于 $\mathcal{A}_p$ 的大小（其大小与 $\mathcal{A}$ 的大小呈线性关系），$L^{\forall}(\mathcal{A}[p])$ 是否可实现。

如果 $\mathcal{S}$ 是 $\mathcal{P}$-可实现的，那么根据断言证明中构造的机器 $\mathcal{M}$，它也是 $\mathcal{P}$-可实现的。对于 $\mathcal{P}$ 的每个空缺 $h = (p, \mathsf{i})$，根据定理 1，机器 $\mathcal{M}_h$ 的大小可界定为 $2^{O(nlog_2 n_)}$，其中 n 是 $\mathcal{A}_p$ 的状态数，这恰好等于 $\mathcal{A}$ 的状态数。因此，如果 $\mathcal{P}$ 有 n_h 个空缺，$\mathcal{S}$ 就能由一台有 $m + n_h 2^{O(nlog_2 n_)}$ 个状态的 Mealy 机实现为 $\mathcal{P}$-可实现。

对于下界，我们将问题归约为判定由确定性自顶向下树自动机定义的 n 个有限树语言的交集是否非空。已知这个问题是 ExpTime-c [13]。这使我们能够证明，即使规约由确定性可达性自动机给出，$\mathcal{P}$-可实现性仍有相应的下界。这与普通可实现性形成对比：对于这一类规格，普通可实现性可在 PTime 内求解 [12]。直观而言，$\mathcal{P}$-可实现性比可实现性更难的根本原因在于，$\mathcal{P}$ 对解施加了严格约束。具体来说，它要求实现 $\mathcal{S}$ 的系统 $\mathcal{P}$ 在任何能够到达 $\mathcal{P}$ 同一状态的前缀之后，都表现一致。这就是为什么在上面的 ExpTime 解法中，需要检查形如 $\bigcap_{u\in \mathsf{Left}_{\mathcal{P}}}(u\mathsf{io}_h)^{-1}\mathcal{S}$ 的规约交集是否可实现；这比尝试实现单体规约更难。

现在我们给出获得下界的详细证明。它将以下 ExpTime-c 问题 [13] 归约为：给定 n 确定性自顶向下树自动机 $(\mathcal{T}_i)_{i=1}^{n}$，判断 $\bigcap_{i=1}^{n} L(\mathcal{T}_i) \neq \varnothing$。其主要思想已经体现在受限问题中，其中 $\mathcal{T}_i$ 是确定有限自动机，已知该问题为 PSpace-c，因此我们先介绍这一情形。设 $(\mathcal{D}_i = (Q_i, in_i, F_i, \delta_i)_{i=1}^{n})$ 是某个字母表 Σ 上的 n 确定有限自动机。令 $\mathcal{I} = \{\mathsf{i}_1、\ldots\ldots、\mathsf{i}_n\}$，并令 $\mathcal{O} = \Sigma \cup \{\mathsf{skip}、\mathsf{exit}\}$。对于所有 $j \in \{1, \ldots, n\}$，令 $\mathcal{I} \otimes L(\mathcal{D}_j)$ 表示形如 $\lambda_1\sigma_1\lambda_2\sigma_2 \ldots \lambda_k\sigma_k \in (\mathcal{I}\mathcal{O})^*$ 且满足 $\sigma_1 \ldots \sigma_k \in L(\mathcal{D}_j)$ 的词的集合。考虑以下规约：

$$\mathcal{S} = \bigcup_{j=1}^{n} \{\mathsf{i}_j.\mathsf{skip}.u.\mathsf{i}.\mathsf{exit}.x \mid u \in \mathcal{I} \otimes L(\mathcal{D}_j), \mathsf{i} \in \mathcal{I}, x \in (\mathcal{I}\mathcal{O})^{\omega}\}$$

我们还定义如下具有两个状态的 preMealy 机 $\mathcal{P}$：从其初始状态 m_0 出发，每当它读取 i_j（对于任意 $j = 1, \ldots, n$）时，就输出 skip 并转移到它的第二个状态 m，该状态是一个空缺。

我们证明：

1. $\mathcal{S}$ 可由一个多项式规模的确定性可达自动机 $\mathcal{A}_{\mathcal{S}}$ 识别。2. $\mathcal{S}$ 是 $\mathcal{P}$-可实现的，当且仅当 $\bigcap_{i=1}^{n} L(\mathcal{D}_i) \neq \varnothing$。

首先，注意到 $\mathcal{S}$ 可由一个多项式规模的确定性可达自动机 $\mathcal{A}_{\mathcal{S}}$ 识别。直观地说，每个自动机 $\mathcal{D}_i$ 都经过修改，使得来自 $\mathcal{I}$ 的任意输入符号都可以在两个输出字母之间读取，因此

它识别 $\mathcal{I}\otimes L(\mathcal{D}_i)$。我们将修改后的自动机记作 $\mathcal{I}\otimes\mathcal{D}_i$，并假设所有自动机 $\mathcal{I}\otimes\mathcal{D}_i$ 的状态集合两两不相交。从它唯一的初始状态$\mathcal{A}_{\mathcal{S}}$出发，对于所有 $j=1$、...、n，都可以读取由两个符号组成的序列 $\mathsf{i}_j\cdot$，跳过并转到 $\mathcal{I}\otimes\mathcal{D}_j$的初始状态。此外，我们添加一个状态 q_{reach}，它是唯一的接受状态（也就是说，它的颜色为 0，而其他任何状态的颜色均为 1）。从 q_{reach}出发，任意序列都是接受的（它是一个陷阱）。最后，对于 $\mathcal{D}_j$的所有接受状态 q_f 以及所有输入 $\mathsf{i}\in\mathcal{I}$，我们令 $\mathcal{A}_{\mathcal{S}}$ 在读取 i。时从状态 q_f转移到 q_{reach}。

对于第二个断言，其证明背后的主要直观思路是：对于任何可能的初始输入，$\mathcal{P}$都会转移到同一个状态 m，而 $\mathcal{A}_{\mathcal{S}}$ 则会转移到不同的状态。因此，$\mathcal{P}$ 确保无论初始输入是什么i_j，之后都应采用相同的策略；另一方面，$\mathcal{S}$ 的定义则取决于初始输入。形式化地说，假设 $\mathcal{M}$ 是一个 Mealy 机 $\mathcal{P}$，实现了 $\mathcal{S}$。那么，由于 $\mathcal{P}$ 是 $\mathcal{M}$的子图，$\mathcal{M}$ 的语言必然具有以下形式

$$L_\omega(\mathcal{M})=\mathcal{I}.\mathsf{skip}.L' \qquad (1)$$

对于某个 L'，使得 $\mathcal{I}.\mathsf{skip}.L'\subseteq\mathcal{S}$。令 $w\in L_\omega(\mathcal{M})$。它必然具有以下形式$w=\mathsf{i}_j\cdot\mathsf{skip}.u.\mathsf{i}$。$\mathsf{exit}.x$ 对于某个 $j=1,\dots,n$，$u\in(\mathcal{I}\otimes L(\mathcal{D}_j))$，$\mathsf{i}\in\mathcal{I}$且 $x\in(\mathcal{I}\mathcal{O})^\omega$。由（1）可得，对于任何其他 $j'\neq j$，$w'=\mathsf{i}_{j'}\cdot\mathsf{skip}.u.\mathsf{i}$。$\mathsf{exit}.x\in L_\omega(\mathcal{M})$，因此，$u\in(\mathcal{I}\otimes L(\mathcal{D}_{j'})$。所以，$\bigcap_{i=1}^n L(\mathcal{D}_j)\neq\varnothing$。

反向命题的证明类似：如果 $v\in\bigcap_{i=1}^n L(\mathcal{D}_j)$，那么要实现 $\mathcal{P}$- $\mathcal{S}$，系统只需先执行 skip，再执行 v，然后永远执行 exit。这个策略很容易用扩展了 $\mathcal{P}$的 Mealy 机来描述。

这证明了 PSPACE-困难性。将后一种归约扩展到确定性自顶向下树自动机（在有限二进制 Σ-树上）是标准做法：环境在树中选择方向 $\{1,2\}$，而系统选择标记。我们像之前一样令 $\mathcal{I}=\{\mathsf{i}_1,\dots,\mathsf{i}_j\}\cup\{1,2\}$和 $\mathcal{O}=\Sigma\cup\{\mathsf{exit},\mathsf{skip}\}$。规约 $\mathcal{S}$ 修改如下：$\mathcal{S}=\bigcup_{j=1}^n\mathcal{S}_j$ 其中每个 $\mathcal{S}_j$ 都是形如 $\mathsf{i}_j\cdot\mathsf{skip}.u.\mathsf{i}.\mathsf{exit}.x$的词的集合，并且存在一棵有限二进制树$t\in L(\mathcal{T}_j)$，使得 u 是 t的一条根到叶分支；也就是说，$u=d_1\sigma_1\dots d_k\sigma_k$ 其中每个 $d_i\in\{1,2\}$ 都是一个方向，而每个 σ_i 都是由根到节点路径 $d_1\dots d_i$标识的 t 节点的标记。preMealy 机 $\mathcal{P}$与之前相同，并且很容易看出，新的规约 $\mathcal{S}$ 可由一个多项式规模的确定性可达性自动机定义：这是因为树自动机是确定性自顶向下的，而且确定性自顶向下树自动机的路径语言是正则的，可由多项式规模的确定有限自动机识别 [13]。

下面简要说明该构造的正确性。如果 $t\in\bigcap_{i=1}^n L(\mathcal{T}_i)$，那么$\mathcal{P}$ 可以扩展为一个充满的 Mealy 机，它在状态 m 之后会精确模仿 t 的结构：状态对应于 t 中的路径；当收到一个新的方向作为输入时，它会输出沿该方向到达的 t 的标记。反之，如果当前路径是 t 的叶节点，那么无论 Mealy 机今后收到什么输入，它都会永远输出 exit。该机器保证能够实现该规约，因为无论初始输入是什么，由环境的选择所诱导的树中的所有分支都被所有树自动机接受。

反过来，如果存在一个 Mealy 机 $\mathcal{M}$，它扩展了 $\mathcal{P}$ 并实现该规约，那么无论初始输入是什么，它之后都会采用相同的策略。于是，可以根据环境所做的选择（方向，这些选择描述树中的路径）以及系统所做的选择（这些选择对应于由这些路径标识的节点的标记），重建一棵被所有树自动机接受的树。由于所有可能结果最终都必须输出 exit，以这种方式构造的树保证是有限的。□

3 从安全规约和示例进行综合

本节介绍我们用于从示例和安全规约综合 Mealy 机的学习框架。第 4 节将说明如何将其推广到任意 ω-正则规约，并通过归约至安全规约来解决。这是一个两阶段算法，概述如下：(1) 在保持规约可实现性的同时，尽可能泛化示例，并输出一台 preMealy 机；(2) 将这台 preMealy 机补全为一台完整的 Mealy 机。

3.1 阶段 1：泛化示例

该阶段通过尽可能泛化示例，同时保持规约的可实现性，充分利用这些示例。如果存在这样的 preMealy 机，该阶段便会输出一台与示例一致且能够实现规约的 preMealy 机。这是一种类 RPNI 学习算法 [23,22]，其中包含用于保持规约可实现性的特定测试。

此阶段的第一步是构建一台树形 preMealy 机，其接受的语言恰好是给定示例集 E 的所有前缀 Prefs(E)，称为前缀树接受器（PTA）。形式化地，我们将 PTA 定义如下：

前缀树接受器集合 $E \subseteq (\mathcal{I}\mathcal{O})^*$（不一定有限）是一致的，当且仅当对于所有 $e \in \mathsf{Prefs}(E) \cap (\mathcal{I}\mathcal{O})^*\mathcal{I}$，都存在唯一的输出 $\mathsf{o}_E(e) \in \mathcal{O}$，使得 $e.\mathsf{o}_E(e) \in \mathsf{Prefs}(E)$。当 E 一致且有限时，我们可以规范地将其关联到一台树形 preMealy 机，记作 PTA(E)，使得L(PTA(E)) $= \mathsf{Prefs}(E) \cap (\mathcal{I}\mathcal{O})^*$，定义如下：

$$\mathsf{PTA}(E) = (\mathsf{Prefs}(E) \cap (\mathcal{I}\mathcal{O})^*, \epsilon, (e, \mathsf{i}) \mapsto (\mathsf{o}_E(e\mathsf{i}), e\mathsf{i}\mathsf{o}_E(e\mathsf{i})))$$

例 *4*。令 $\mathcal{I} = \{\mathsf{i}, \mathsf{i}'\}$ 和 $\mathcal{O} = \{\mathsf{o}, \mathsf{o}'\}$，并考虑 $E_0 = \{\mathsf{i}'\mathsf{o}, \mathsf{ioioi}'\mathsf{o}'\}$。那么 E_0 是一致的，PTA(E_0) 如图 2 左侧所示。为简洁起见，我们将其状态记为 0，……，4，其中 $0 = \epsilon$、$1 = \mathsf{i}'\mathsf{o}$、$2 = \mathsf{io}$、$3 = \mathsf{ioio}$ 和 $4 = \mathsf{ioioi}'\mathsf{o}'$。

在该阶段的下一步中，算法会尝试合并 PTA 中尽可能多的状态。用于选择与给定状态合并的另一个状态的策略是算法的一个参数，称为 <style id='1'>合并策略</style>{v3}。

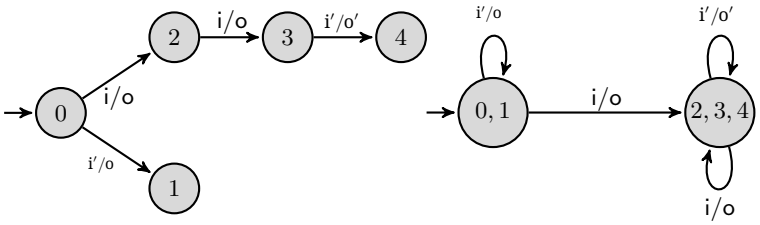


图 2：例 4 中的 preMealy 机 PTA({i′o，ioioi′o′})，以及按照例 6 所述，由划分 $\{\{0,1\},\{2,3,4\}\}$ 诱导的等价关系取商所得的机器。

形式上，合并策略 σ_G 定义在由 4-tuples $(\mathcal{M}, m, E, X)$ 构成的集合上，其中 $\mathcal{M}$ 是一个 preMealy 机，m 是 $\mathcal{M}$的一个状态，E 是一个示例集，X 是 $\mathcal{M}$ 的状态子集（即要与 m 合并的候选状态），并返回X的一个状态，即 $\sigma_G(\mathcal{M}, m, E, X) \in X$。形式化定义如下：

状态合并 现在，我们定义 RPNI 中经典的、适用于 Mealy 机的状态合并操作。若等价关系 $\sim$ 定义在 M 上，则称其为 $\mathcal{M}$ 的同余，如果对于所有 $x \sim x'$ 和 $\mathsf{i} \in \mathcal{I}$，当 $\Delta_{\mathcal{M}}(x, \mathsf{i})$ 和 $\Delta_{\mathcal{M}}(x', \mathsf{i})$ 均有定义时，$\mathrm{Post}_{\mathcal{M}}(x, \mathsf{i}) \sim \mathrm{Post}_{\mathcal{M}}(x', \mathsf{i})$。若对于 $\mathcal{M}$ 还满足 $\mathrm{Out}_{\mathcal{M}}(x, \mathsf{i}) = \mathrm{Out}_{\mathcal{M}}(x', \mathsf{i})$，则称其为 Mealy 同余。当上下文中 $\mathcal{M}$ 的含义明确时，我们就简称为同余和 Mealy 同余。如果 $\sim$ 是一个 Mealy 同余，那么以下 preMealy 机（称为 $\mathcal{M}$ 关于 $\sim$的商）是定义良好的 preMealy 机（它不依赖于代表元的选择）：$\mathcal{M}/_{\sim} = (M/_{\sim}, [m_{\mathsf{init}}], ([s], \mathsf{i}) \mapsto (\mathrm{Out}(s, \mathsf{i}), [\mathrm{Post}(s, \mathsf{i})]))$。在此定义中，$[s]$ 表示 s 关于 $\sim$的等价类，并且我们选取一个代表元 s，使得 $\Delta(s, \mathsf{i})$ 有定义。如果不存在这样的代表元，则转移在 i 上未定义。

阶段 1 的伪代码见算法 1。这里我们给出一个运行示例，以更好地说明该算法的工作方式。首先，算法会检查示例集合 E 是否一致[2]；如果一致，则根据定理 2，进一步检查 PTA(E) 是否可以补全为一个实现给定规约$\mathcal{S}$的 Mealy 机。

例 *5.* [从 $\varphi_{\mathsf{CORE}}^{\mathsf{ME}}$ 和示例] 进行综合。考虑例3中描述的经典互斥问题，其线性时序逻辑规范为$\varphi_{\mathsf{CORE}}^{\mathsf{ME}}$，执行前缀为：

(1) $\{!r_1, !r_2\}.\{!g_1, !g_2\}\#\{r_1, !r_2\}.\{g_1, !g_2\}\#\{!r_1, r_2\}.\{!g_1, g_2\}$(2) $\{r_1, r_2\}.\{g_1, !g_2\}\#\{!r_1, !r_2\}.\{!g_1, g_2\}$我们首先构建图 3 所示的 PTA，然后检查 $\varphi_{\mathsf{CORE}}^{\mathsf{ME}}$ 是否 PTA − 可实现。

[2] E 当输出由前缀唯一确定时，该集合是一致的。形式化地说，对于所有前缀 $u \in \mathrm{Prefs}(E) \cap (\mathcal{I}\mathcal{O})^*\mathcal{I}$，都存在唯一的输出 $\mathsf{o} \in \mathcal{O}$，使得 $u\mathsf{o} \in \mathrm{Prefs}(E)$。

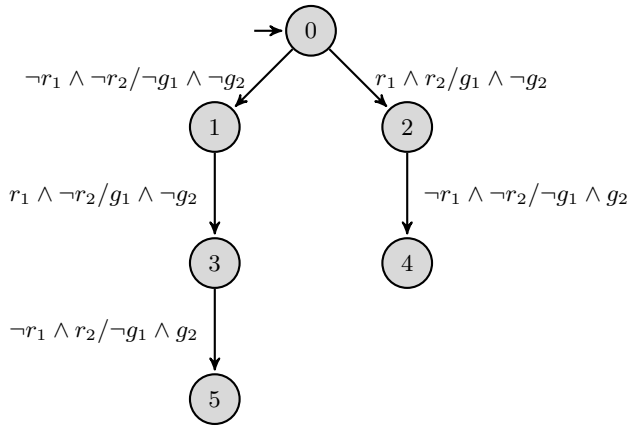
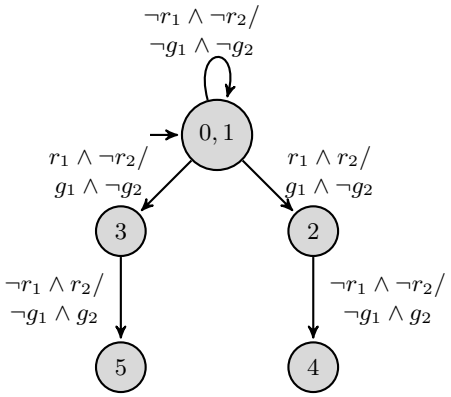


图 3：例 5 的 preMealy 机 PTA。在这里，我们发现 $\varphi^{\mathsf{ME}}_{\mathsf{CORE}}$ 是 PTA 可实现的

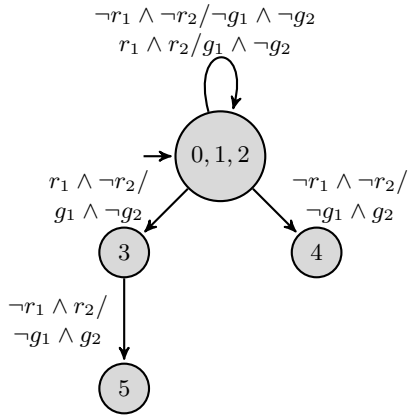
如果是这样，那么算法会将 E 的所有前缀作为示例集，并进入一个循环，在该循环中，通过合并若干等价类，反复逐步粗化 PTA(E) 状态上的某个同余关系 $\sim$ 。同余关系 $\sim$ 初始为最细等价关系。粗化按特定顺序进行：示例（即 PTA(E) 的状态）按长度优先的字典序排列。进入循环处理示例 e 时，算法会在第 5 行计算所有状态，即循环中已处理过的所有示例 e' （$e' \prec_{ll} e$），并筛选出当前等价类可与 e 的等价类合并的状态（谓词 Mergeable(PTA(E) $\sim$, e, e')）。状态合并是自动机学习算法中的标准操作，直观而言，就是合并 e 的 $\sim$ 等价类和 e' 的 $\sim$ 等价类，并将这次合并传播到 e 和 e' 的后代，而不会导致任何冲突。在第 6 行，算法会对前一个集合进行筛选，只保留与 e 合并后能够补全为实现 $\mathcal{S}$ 的 Mealy 机的 preMealy 机（同样依据定理 2）。筛选后，如果仍有多个可合并候选项，则使用合并策略 σ_G 选出其中一个，然后通过等价类合并操作（MergeClass(PTA(E) $\sim$, e, e')）进一步粗化该等价关系。最后，算法返回 PTA(E) 按计算得到的 Mealy 同余关系取商所得的结果。顺带一提，当 $\mathcal{S}$ 为普适的，即 $\mathcal{S} =(\mathcal{IO})^\omega$ 时，任意 Mealy 机都能实现它，因此第 6 行不会筛除任何可合并候选项。所以，当 $\mathcal{S}$ 为普适时，算法 1 可视为一种用于学习 preMealy 机的 RPNI 变体。

例 *5*（续）：从 $\varphi^{\mathsf{ME}}_{\mathsf{CORE}}$ 和示例 我们注意到，图 3 中 PTA 的每个状态m 都是 $\sim$-类 e，其中 e 是满足$\Delta(q_{\mathsf{init}}, e) = m$的最短前缀。然后我们检查 $\varphi^{\mathsf{ME}}_{\mathsf{CORE}}$ 是否 PTA $-$ 可实现[3]，发现结果为

[3] 请参阅 *Checking* PTA $-$可实现性 *of a specification* S 第4节



(a) 我们首先合并 preMealy 机 PTA 的状态 0 和 1，即合并等价类 $[\epsilon]$ 和 $[\{\neg r_1 \wedge \neg r_2\}\{\neg g_1 \wedge \neg g_2\}]$ and 。随后检查 PTA – 的可实现性，结果为真。所得机器如下所示。



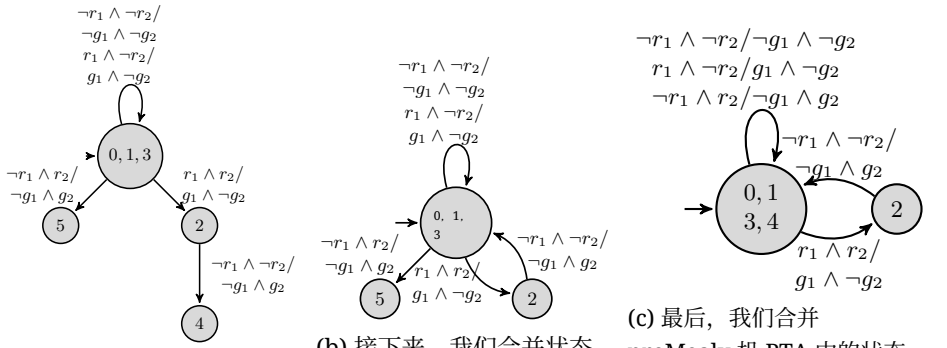
(b) 接着，我们合并 preMealy 机 PTA 的状态$\{0, 1\}$ and 2，即合并等价类 $[\epsilon]$ 和$[r_1 \wedge r_2/g_1 \wedge \neg g_2]$。随后检查 PTA – 的可实现性，结果为假。我们通过观察轨迹 $(r_1 \wedge r_2/g_1 \wedge \neg g_2)^\omega$ 不满足线性时序逻辑子公式 $G(r_2 \implies Fg_2)$ 来验证这一点。因此，合并未成功，并被撤销。

图 4：例 5 中 preMealy 机 PTA 的合并阶段。

确实如此。我们注意到，每个状态都按长度字典序标记。接着，我们按照上述顺序开始合并状态，如图 4 和图 5 所示。

3.2 阶段 2：将 preMealy 机完备化为 Mealy 机

由于泛化阶段只构造 PTA 并尝试合并其中的状态，因此它可能不会返回（完备的）Mealy 机。换言之，它返回的机器可能仍包含一些空缺（缺失转换）。第二阶段的目标是在实现规约的同时，将这些空缺补全为 Mealy 机。更具体地说，当从某个状态 m 出发、输入为 $\mathsf{i} \in \mathcal{I}$的转移未定义时，算法必须选择一个输出符号 $\mathsf{o} \in \mathcal{O}$ 和一个要转移到的状态 m'；该状态可以是已有状态，也可以是要创建的新状态（在这种情况下，我们用 $m' = \text{fresh}$ 表示 m' 是一个新状态）。在我们的实现中，如果可以复用泛化阶段创建的状态 m'，则优先选择它而非其他状态，以便利用示例。不过，我们现在描述的补全阶段算法并不依赖于任何特定的状态选择策略。因此，它由一个补全策略 σ_C参数化，该策略定义在所有三元组 $(\mathcal{M}, m, \mathsf{i} X)$ 上，其中 $\mathcal{M}$ 是一个状态集为



(a) 接下来，我们合并 pre-Mealy 机 PTA 的状态 $\{0, 1\}$ 和 3，即合并等价类 $[\epsilon]$ 和 $[\{\neg r_1 \wedge \neg r_2\}\{\neg g_1 \wedge \neg g_2\}\ \{r_1 \wedge \neg r_2\}\{g_1 \wedge \neg g_2\}]$ #。然后，我们检查 PTA –的可实现性，结果为真。所得机器如下所示。

(b) 接下来，我们合并状态 $\{0, 1, 3\}$ 和 pre-中的 4 Mealy 机 PTA，即，我们合并等价类 $[\epsilon]$ 和 $[\{r_1 \wedge r_2\}\{g_1 \wedge \neg g_2\}\#\{\neg r_1 \wedge \neg r_2\}\{\neg g_1 \wedge g_2\}]$。然后我们检查 PTA –的可实现性结果表明其可实现。由此得到的 机器如图所示。

(c) 最后，我们合并 preMealy 机 PTA 中的状态 $\{0, 1, 3, 4\}$ 和 5，即合并等价类 $[\epsilon]$ 和 $[\{\neg r_1 \wedge \neg r_2\}\{\neg g_1 \wedge \neg g_2\}\#\{r_1 \wedge \neg r_2\}\{g_1 \wedge \neg g_2\}]\ \{\neg r_1 \wedge r_2\}\{\neg g_1 \wedge g_2\}]$ #。随后，我们检查 PTA –的可实现性，结果为真。由此得到的机器如图所示。

图 5：例 5（续）中 preMealy 机 PTA 的合并阶段。

M，(m，i) 是 $\mathcal{M}$的一个空缺，而 $X \subseteq \mathcal{O} \times (M \cup \{\text{fresh}\})$ 是候选对 (o m') 的列表。它返回 X中的一个元素，即 $\sigma_C(\mathcal{M},\ m,\ \text{i}\ X) \in X$。

除了 σ_C之外，补全算法还将一个 preMealy 机 $\mathcal{M}_0$ 和一个规约 $\mathcal{S}$作为输入，并输出一个 Mealy 机；若该机器存在，则它 $\mathcal{M}_0$-实现$\mathcal{S}$。伪代码见算法 2。首先，它检查 $\mathcal{S}$是否 $\mathcal{M}_0$可实现，否则返回 UNREAL。然后，它不断补全 $\mathcal{M}_0$ 中的空缺。输出/状态候选项列表的计算在第 5 行的循环中进行。注意，**for**循环遍历 $M \cup \{\text{fresh}()\}$，其中 fresh() 是一个过程，返回一个不属于 M的新状态。该算法维持这样一个不变式：在**while**循环的每次迭代中，$\mathcal{S}$ 都是 $\mathcal{M}$可实现的；这一点得益于第 7 行基于定理 2 的检查。因此，候选项列表必定非空。从这些候选项中选出一个，并在第 10 行相应地将 (m,i) 上的转移添加到 $\mathcal{M}$ 。

3.3 从规约和示例进行综合的两阶段算法

用于安全规约和示例的两阶段综合算法，称为 SynthSafe(E，$\mathcal{S}$，σ_G，σ_C)，工作方式如下：它以示例集 E、以确定性安全自动机形式给出的规约 $\mathcal{S}$，以及泛化策略和补全策略 σ_G，σ_C 分别作为输入。若存在满足 $E \subseteq L(\mathcal{M})$ 的 Mealy 机 $\mathcal{M}$ ，使其实现 $\mathcal{S}$ ，则返回该机器。第一步，它调用 Gen(E，$\mathcal{S}$，σ_G)。如果调用返回 UNREAL，那么 SynthSafe 也返回

算法 1： GEN(E,$\mathcal{S}$,σ_G) – 泛化算法

输入： 有限示例集 $E \subseteq (\mathcal{I}.\mathcal{O})^*$，以及给定规约 $\mathcal{S} \subseteq (\mathcal{I}.\mathcal{O})^\omega$ 作为确定性安全自动机，以及合并策略 σ_G

输出： 一个 preMealy 机 $\mathcal{M}$，使得 $E \subseteq L(\mathcal{M})$ 且 $\mathcal{S}$ 是 $\mathcal{M}$-可实现的，若其存在，否则为 UNREAL。

如果 E 不一致，或 $\mathcal{S}$ 不是 PTA(E)-可实现的 **则返回** UNREAL
$E \leftarrow$ 偏好设置(E) $\cap (\mathcal{I}\mathcal{O})^*$;
$\sim \leftarrow \{(e, e) \mid e \in E\}$; // $\sim = diag_E$
对于 $e \in E$ *in length-lexicographic order* $\preceq_{ll}$ **执行**
  $mergeCand \leftarrow \{e' \mid$ 可合并(PTA(E) $\sim, e, e') \wedge e' \prec_{ll} e\}$
  $mergeCand \leftarrow \{e' \in mergeCand \mid \mathcal{S}$ 是合并状态(PTA(E) $\sim e, e') - realizable\}$
  如果 $mergeCand \neq \varnothing$ **那么**
    $e' \leftarrow \sigma_G(\mathcal{M}, e, mergeCand)$
    $\sim \leftarrow$ 合并类(PTA(E) $\sim, e, e'$)
返回 $\mathsf{PTA}(E)/_\sim$

UNREAL。否则，调用 GEN 会返回一个 preMealy 机 $\mathcal{M}_0$。第二步，SYNTHSAFE 调用 COMP($\mathcal{M}_0$, $\mathcal{S}$, σ_C)。如果此次调用返回 UNREAL，那么 SYNTHSAFE 也会如此；否则，SYNTHSAFE 返回由 COMP 计算出的 Mealy 机。SYNTHSAFE 的伪代码见算法 3。

对于某些补全策略，补全过程可能无法终止。这是因为补全策略可能会不断选择形如 (o m') 的状态对，其中 m' 是一个新状态。然而，我们证明，对于*惰性*补全策略，该过程总是终止的。如果补全策略 σ_C 倾向于选择已有状态，则称其为*惰性*；形式上，这意味着如果 $X \setminus (\mathcal{O} \times \{新\}) \neq \varnothing$，那么 $\sigma_C(\mathcal{M}, m,$ i $X) \notin \mathcal{O} \times \{新\}$。第一个定理确立了惰性补全策略下算法的正确性和终止性（我们假设函数 σ_G 和 σ_C 的最坏情况运行时间相对于其输入规模为指数时间）。

定理 3（终止性和正确性）。对于所有有限的示例集 $E \subseteq (\mathcal{I}.\mathcal{O})^*$，所有以确定性安全自动机 $\mathcal{A}$ 表示且具有 n 个状态的规约 $\mathcal{S} \subseteq (\mathcal{I}.\mathcal{O})^\omega$，所有合并策略 σ_G 以及所有补全策略 σ_C，如果 SYNTHSAFE($E, \mathcal{S}, \sigma_G, \sigma_C$) 终止，则会返回一个 *Mealy* 机 $\mathcal{M}$，使得 $E \subseteq L(\mathcal{M})$，且 $\mathcal{M}$ 实现 $\mathcal{S}$（若其存在）；否则返回 *UNREAL*。此外，如果 σ_C 是惰性的，则 SYNTHSAFE($E, \mathcal{S}, \sigma_G, \sigma_C$) 会在最坏情况下于指数时间内终止（关于 E 的规模为多项式时间，关于 n 为指数时间）。

后一条定理的证明基于若干关于泛化阶段和补全阶段的结果，详见附录 D.9。直观而言，复杂度主要取决于检查 $\mathcal{P}$-可实现性（定理 2）的复杂度，以及补全过程的终止时间；我们证明，后者在 n 中最坏情况下为指数级。假设规约是一个

[4] E 的大小是 E的示例长度之和。

算法 2： COMP($\mathcal{M}_0$,$\mathcal{S}$,σ_C)：preMealy 机补全算法

输入： 一个 preMealy 机 $\mathcal{M}_0 = (M,\ m_{\text{init}},\ \Delta)$，一个规约 $\mathcal{S} \subseteq (\mathcal{I}.\mathcal{O})^*$ 以确定性安全自动机的形式给出，一个补全策略 σ_C

输出： 一个（完备的）Mealy 机 $\mathcal{M}$，使得 $\mathcal{S}$ 是 $\mathcal{M}_0$-可实现的，否则为 UNREAL。

如果 $\mathcal{S}$ 不可 $\mathcal{M}_0$实现 **则返回** UNREAL
$\mathcal{M} \leftarrow \mathcal{M}_0$
当 *there exists a hole* $(m,\text{i}) \in M \times \mathcal{I}$ **时**
  $candidates \leftarrow \varnothing$
  对于(o m') $\in \mathcal{O} \times (M \cup \{$新建()$\})$ **执行**
    // fresh() 表示一个不在 M中的新状态
    $\mathcal{M}_{\text{o},\ m'} \leftarrow (M \cup \{m'\},\ m_{\text{init}},\ \Delta \cup \{(m,\ \text{i}) \mapsto (\text{o}\ m')\})$
    如果 $\mathcal{S}$ *is* $\mathcal{M}_{\text{o},m'}$*-realizable* **那么**
      $candidates \leftarrow candidates \cup \{(\text{o}\ m')\}$
  (o m') $\leftarrow \sigma_C(\mathcal{M},\ m,\ \text{i}\ candidates)$
  $(M,\ \Delta) \leftarrow (M \cup \{m'\},\ \Delta \cup \{(m,\ \text{i}) \mapsto (\text{o}\ m')\})$
  $\mathcal{M} \leftarrow (M, m_{\text{init}}, \Delta)$
返回 $\mathcal{M}$

确定性安全自动机 $\mathcal{A}$，这一假设用于证明补全算法的终止性。直观而言，对于迄今构造出的 preMealy 机 $\mathcal{M}$ 中的任意状态 m，我们将 $\mathcal{A}$ 的状态子集 Q_m 与之关联；当读取能够到达 m 的前缀时，该子集中的状态可在 $\mathcal{A}$ 中到达。我们证明，当向 $\mathcal{M}$ 和 $Q_{m'} \subseteq Q_m$ 添加一条通向新状态 m' 的转移时，可以改用 m，而不必使用 m'（附录 D.6 中的引理 11）。这是可行的，因为此类子集足以概括 $\mathcal{A}$ 在无限后缀上的行为，原因在于它是一个安全条件。我们还证明，随着向 $\mathcal{M}$ 添加更多转移，子集 Q_m 具有某种单调性，从而可以用由 $\subseteq$-反链组成的最长链的长度来界定终止时间；在最坏情况下，该长度相对于 $\mathcal{A}$ 的状态数呈指数级增长（附录 D.6 中的引理 13 ）。

如果对于所有满足 $L(\mathcal{T}) = L(\mathcal{M})$ 的 Mealy 机 $\mathcal{M}$，Mealy 机 $\mathcal{T}$ 的状态数都不少于 $\mathcal{T}$ 的状态数，那么 $\mathcal{T}$ 就是最小的。下一个结果在附录 D.10 中证明，它指出，对于实现规约 $\mathcal{S}$ 的任意最小 Mealy 机，我们的综合算法都能将其返回，从而提供具有代表性的示例。

定理 4 (Mealy 完备性)。对于所有表示为确定性安全自动机的规格 $\mathcal{S} \subseteq (\mathcal{I}.\mathcal{O})^\omega$，以及所有实现 $\mathcal{S}$ 的最小 ***Mealy*** 机 $\mathcal{M}$，都存在一个有限示例集 $E \subseteq (\mathcal{I}.\mathcal{O})^*$，其大小关于 $\mathcal{M}$ 的大小呈多项式级；使得对于所有泛化策略 σ_G、补全策略 σ_C，以及所有满足 $E \subseteq E' \subseteq L(\mathcal{M})$ 的示例集 E'，$\textsc{SynthSafe}(E', \mathcal{S}, \sigma_G, \sigma_C) = \mathcal{M}$。

定理 4 的表述中给出的多项式上界更确切地说如下：E的基数为 $O(m + n^2)$，其中 n是 $\mathcal{M}$ 的状态数，而 m 是其转移数。此外，每个示例$e \in E$ 的长度为 $O(n^2)$。更多详情请参见注释 1。

算法 3： SynthSafe(E,$\mathcal{S}$,σ_G,σ_C) – 根据规约和示例进行综合的算法 输入：规约 $\mathcal{S} \subseteq (\Sigma.\mathcal{O})^*$，以确定性安全自动机的形式给出；有限示例集 $E \subseteq (\Sigma.\mathcal{O})^*$；泛化策略和补全策略 σ_G, σ_C输出：Mealy 机 $\mathcal{M}$，使得 $E \subseteq L(\mathcal{M})$ 且 $\mathcal{M}$ 在其存在时可实现 $\mathcal{S}$，否则返回 UNREAL。1 如果 $\textsc{Gen}(E, \mathcal{S}, \sigma_G) \neq \mathit{UNREAL}$ 则

2 $\mathcal{M}_0 \leftarrow \textsc{Gen}(E, \mathcal{S}, \sigma_G)$ // 根据 σ_G 返回一个泛化示例集的 preMealy 机，并使得 $\mathcal{S}$ 可$\mathcal{M}_0$实现 3 否则 4 返回 UNREAL 5 如果 $\textsc{Comp}(\mathcal{M}_0, \mathcal{S}, \sigma_C) \neq \mathit{UNREAL}$ 则6 $\mathcal{M} \leftarrow \textsc{Comp}(\mathcal{M}_0, \mathcal{S}, \sigma_C)$ // 根据 σ_C通过创建新状态或重用状态来补全 $\mathcal{M}_0$ 7 返回 $\mathcal{M}$8 否则 9 返回 UNREAL

注释 *1.* 我们在此给出特征样本 $E_{\mathcal{T}}$ 的大小上界。令 n 和 m 分别表示 $\mathcal{T}$ 的状态数和转移数。那么，对于所有状态 t，s_t 的长度至多为 $n-1$；因此，对于所有满足 $\Delta(p)$ 已定义的 $p = (t, \mathrm{i})$，$e_p = f^{\mathcal{T}}_{\mathrm{i}\Phi}(s_t)$ 的长度至多为 $2n$。对于两个不同的状态 $t \neq t'$ 和 $d_{t,t'}$ 的长度至多为 n^2。因此，$v_{t,t'}$ 的长度至多为 $2(n+n^2)$。至多有 m 个词 e_p 和 n^2 个词 $v_{t,t'}$。因此，总体而言，$E_{\mathcal{T}}$ 的基数上界为 $m+n^2$，其大小上界为 $mn + 2(n^3+n^4)$。

4 从 ω-正则规格和示例进行综合

在第4节中，我们现在考虑将规约 $\mathcal{S}$ 表示为普适 coB¨uchi 自动机的情况。我们考虑这一类规约，因为它对于 ω-正则语言是完备的，并且允许紧凑的符号表示。本节后续部分将考虑 LTL 规约的情况。

以普适 *coB* ¨uchi 自动机 </style}我们针对 ω-正则规约的解决方案，依赖于将问题归约为第 3 节讨论的安全情形。该方案基于先前的研究，这些研究为 ω-正则反应式综合开发了所谓的无 Safra 算法 [26,31,20]。其主要思路是将自动机的接受条件从 coB¨uchi 加强为 K-coB¨uchi，这是一种安全接受条件。当 K 足够大时，该方法对于普通合成问题（不含示例）是完备的（最坏情况下，K 相对于自动机状态数呈指数级增长，参见例如 [20]）。此外，它还支持增量式综合算法：如果

由带有 k-coB¨üchi 接受条件的自动机定义的规约是可实现的，那么对于 $k \leq K$，由采用 K-coB¨üchi 接受条件定义的规约也可实现。这里，由于我们也将示例纳入考虑，因此需要对这些结果稍作调整。

定理 5. 给定一个具有 n 个状态、用于定义规约 $\mathcal{S} = L^{\forall}(\mathcal{A})$ 的普适 *co-B¨uchi* 自动机 $\mathcal{A}$，以及一个具有 m 个状态的 *preMealy* 机 $\mathcal{P}$，对于 $\mathcal{S}$ $\mathcal{P}$ $\mathcal{S}' = L_K^{\forall}$ A $\mathcal{P}$ $K =$，当且仅当 () 对) 可实现 $nm|\mathcal{I}|2^{\mathbf{O}(n \log_2 n}$ 时，$\mathcal{S}$ $\mathcal{P}$ $\mathcal{S}' = L_K^{\forall}$ A $\mathcal{P}$ $K =$ 才是可实现的。 $nm|\mathcal{I}|2^{\mathbf{O}(n \log_2 n}$

证明。根据定理 2，给定一个具有 n 个状态、用于定义规约 $\mathcal{S}$ 的普适 co-B¨uchi 自动机 $\mathcal{A}$，以及一个具有 m 个状态和 n_h 个空缺的 preMealy 机 $\mathcal{P}$，$\mathcal{S}$ 是 $\mathcal{P}$- 可实现的，当且仅当它可由一个具有 $m + n_h 2^{O(nlog_2 n)}$ 个状态的 Mealy 机 $\mathcal{P}$ 实现。设 $\mathcal{M}$ 是这样一个 Mealy 机。证明的其余部分依赖于以下引理：

引理 1 ([20]**).** 设 $\mathcal{A}$ 是一个具有 α 个状态的普适 *coB¨uchi* 自动机，$\mathcal{M}$ 是一个具有 β 个状态的 *Mealy* 机，则对于 $k = \alpha \times \beta$，$L_\omega(\mathcal{M}) \subseteq L^{\forall}(\mathcal{A})$ 当且仅当 $L_\omega(\mathcal{M}) \subseteq L_k^{\forall}(\mathcal{A})$。

因此，我们得到 $\mathcal{M}$ $L_K^{\forall}$ $\mathcal{A}$ $K = n \times$ $m + n_h 2^{O(nlog_2 n}$ $\leq$ 对于 $nm|\mathcal{I}|2^{O(nlog_2 n}$ $L_k^{\forall}$ $\mathcal{A}$ k ())) 实现 ( )。反过来，对于任意，任何实现 ( ) 的机器也实现 $L^{\forall}(\mathcal{A})$。 □

下面的引理可直接得出：

引理 2。对于所有 *co-B¨uchi* 自动机 $\mathcal{A}$、所有 *preMealy* 机 $\mathcal{P}$ 以及所有$k_1 \leq k_2$，我们有 $L_{k_1}^{\forall}(\mathcal{A}) \subseteq L_{k_2}^{\forall}(\mathcal{A})$，因此，若 $L_{k_1}^{\forall}(\mathcal{A})$ 是 $\mathcal{P}$-可实现的，则$L_{k_2}^{\forall}(\mathcal{A})$ 是 $\mathcal{P}$-可实现的。此外，对于所有 $k \geq 0$，若 $\mathcal{S}' = L_k^{\forall}(A)$ 是 $\mathcal{P}$-可实现的，则 $\mathcal{S} = L^{\forall}(\mathcal{A})$ 是 $\mathcal{P}$-可实现的。

将后两个结果应用于集合 E 中大小为 m 的示例的 $\mathcal{P} =$ PTA(E) 后，我们可以设计一种算法，从由含有 n 个状态的普适 coB¨uchi 自动机 $\mathcal{A}$ 定义的规约中综合 Mealy 机，并且满足 E：该算法针对不断增大的 k 值，在安全规范 $L_k^{\forall}(\mathcal{A})$ 和 E 上调用 SynthSafe，直到得出肯定结论，或达到上限 $K = 2^{\mathbf{O}(mn_1 \log_2 mn)+1}$。后一种情况下，它会返回 UNREAL。不过，要正确应用 SynthSafe，$L_k^{\forall}(\mathcal{A})$ 必须由确定性安全自动机表示。这是可行的，因为 k-coB¨uchi 自动机是可确定化的 [20]。

确定化 k-co-B¨uchi 自动机 $\mathcal{A}$ 的确定化依赖于对子集构造的一种简单推广：在读取无限词时，除了记住运行前缀可到达的状态集之外，该构造还会统计到达给定状态 q 的运行前缀访问过标记为颜色 1 的状态的最大次数（请记住，运行至多访问 k 个这样的状态才能被接受）。确定性自动机的状态是所谓的计数函数，对于 co-B¨uchi 自动机 $\mathcal{A} = (Q, q_{\mathsf{init}}, \Sigma, \delta, d)$ 和 $k \in \mathbb{N}$，形式化定义为函数集

记作 $CF(\mathcal{A}, k)$，其中函数 $f\colon Q \to \{-1, 0, 1, \ldots, k, k+1\}$。若对于某个状态 q，$f(q) = -1$，这意味着 q 处于非活动状态（当前前缀上没有运行可以从 $\mathcal{A}$ 到达 q）。初始计数函数 f_{init} 将所有颜色为 1 的初始状态映射为 1，将所有颜色为 0 的初始状态映射为 0，并将所有其他状态映射为 -1。我们用 $\mathcal{D}(\mathcal{A}, k)$ 表示 $=(Q^{\mathcal{D}} = CF(\mathcal{A}, k)\ q^{\mathcal{D}}_{\mathsf{init}} = f_{\mathsf{init}}, \Sigma, \delta^{\mathcal{D}}, Q^{\mathcal{D}}_{\mathsf{usf}})$，即通过该确定化过程得到的确定性自动机。下面给出其形式化描述：

定义1（在 $CF(\mathcal{A}, k)$）下的确定化。 设 $\mathcal{A} = (Q, q_{\mathsf{init}}, \Sigma, \delta, d)$ 是一个 *co-Büchi* 自动机，且 $k \in \mathbb{N}$。我们将确定性安全自动机 $\mathcal{D}(\mathcal{A}, k) = (Q^{\mathcal{D}}, q^{\mathcal{D}}_{\mathsf{init}}, \Sigma, \delta^{\mathcal{D}}, Q^{\mathcal{D}}_{\mathsf{usf}})$ 与这一对 $(\mathcal{A}, k)$ 相关联，其中：

1. $Q^{\mathcal{D}} = CF(\mathcal{A}, k)$ 是 $\mathcal{A}$ 的 k-计数函数集合。2. $q^{\mathcal{D}}_{\mathsf{init}} = f_0$，其中对于所有 $q \neq q_{\mathsf{init}}$，$f_0(q) = -1$；对于 $q = q_{\mathsf{init}}$，$f_0(q) = 0$，且 $d(q_{\mathsf{init}}) = 2$；对于 $q = q_{\mathsf{init}}$，$f_0(q) = 1$，且 $d(q_{\mathsf{init}}) = 1$。简而言之，被赋值为 -1 的状态均处于非活动状态。初始时，只有 $q = q_{\mathsf{init}}$ 处于活动状态。如果它被标记为颜色 1，则其计数器等于 1；否则等于 0。
3. 对于所有 $f \in CF(\mathcal{A}, k)$ 以及 $\sigma \in \Sigma$，转移函数 $\delta^{\mathcal{D}}$ 定义如下：$\delta^{\mathcal{D}}(f_1, \sigma) = f_2$ 其中对于所有 $q \in Q$，$f_2(q) =$

$$\min\left(\left(\max_{q' \in Q : f_1(q') \geq 0 \wedge q \in \delta(q', \sigma)} f_1(q')\right) + x, k+1\right),\ \textit{with } x = 1 \textit{ if } d(q) = 1, \textit{ and } x = 0 \textit{ if } d(q) = 2.$$

4. 不安全计数函数的集合定义[5] 为 $Q^{\mathcal{D}}_{\mathsf{usf}} = \{f \mid \exists q \in Q \cdot f(q) = k+1\}$。

由 $\mathcal{D}(\mathcal{A}, k)$ 定义的语言是满足以下条件的无限字集合：$w \in \Sigma^\omega$ $\mathcal{D}(\mathcal{A}, k)$ 在 w 上的唯一运行从不访问状态（计数函数）f，使得 $d(f) = 1$。这个（安全）无限字语言记作 $L(\mathcal{D}(\mathcal{A}, k))$。$\mathcal{D}(\mathcal{A}, k)$ 的大小有上界 $k^{\mathbf{O}(|\mathcal{A}|)}$。

引理3（$\mathcal{D}(\mathcal{A}, k)$ 正确性，[20]）。 对于所有普适 *co-Büchi* 自动机 $\mathcal{A}$，对于所有 $k \in \mathbb{N}$，$L^{\forall}_k(\mathcal{A}) = L(\mathcal{D}(\mathcal{A}, k))$。

现在我们可以用伪代码给出算法 SynthLearn，即算法 4。

复杂度考量与上界改进 由于自动机 $\mathcal{D}(\mathcal{A}, k)$ 在最坏情况下的大小相对于自动机 $\mathcal{A}$ 呈指数级增长，直接应用定理 3 会得到一个双指数时间过程。这一复杂度源于以下事实：如定理 2 所示，$\mathcal{P}$-可实现性问题相对于确定性自动机的大小属于指数时间；此外，补全过程的终止性在最坏情况下也需要相对于确定性自动机的大小呈指数级增长。

我们证明，可以改进每次调用 SynthSafe 的复杂度，并获得最优的最坏情况（单）指数级复杂度。我们提供了一种算法，用于检查规约 $\mathcal{S} = L^{\forall}_k(\mathcal{A})$ 的 $\mathcal{P}$-可实现性，其运行时间为

[5] 很容易验证，$Q^{\mathcal{D}}_{\mathsf{usf}}$ 是所需的陷阱。

算法 4： SynthLearn**(**E**,**$\mathcal{A}$**,**σ_G**,**σ_C**) – 通过归约至安全性的** ω**-正则规约和示例综合算法 输入：一个具有** n **个状态的普适 co-B¨uchi 自动机** $\mathcal{A}$ **、有限示例集** $E \subseteq (\mathcal{I}.\mathcal{O})^*$ **、一个泛化策略** σ_G **和一个补全策略** σ_C**。输出：一个实现** $L^{\forall}(\mathcal{A})$ **且满足** $E \subseteq L(\mathcal{M})$ **的 Mealy 机** $\mathcal{M}$ **；如果该机器不存在，则输出 UNREAL。**

$K \leftarrow nm|\mathcal{I}|2^{\mathbf{O}(n \log_2 n)}$; $k \leftarrow 0$; // m 是 E2 的大小 **while** $k \leq K$ **do 3 if** SynthSafe$(E, \mathcal{D}(\mathcal{A}, k)\ \sigma_C, \sigma_G) \neq UNREAL$ **then 4 return** SynthSafe$(E, \mathcal{D}(\mathcal{A}, k)\ \sigma_C, \sigma_G)$5 $k \leftarrow k + 1$; **6 return** $UNREAL$



关于 $\mathcal{A}$ 的规模呈单指数级，关于 k 和 $\mathcal{P}$ 的规模呈多项式级。其次，我们对补全算法的终止性进行了更精细的复杂度分析，结果表明其最坏情况下的运行时间关于 $|\mathcal{A}|$ 呈指数级。这两项改进使总体复杂度达到 SynthLearn，其关于规约 $\mathcal{A}$ 的规模呈指数级，关于示例集 $|E|$ 呈多项式级。可以证明，这是最优的最坏情况复杂度，因为对于 $E = \emptyset$，该问题已经是 ExpTime-Complete。

接下来，我们解释第一个改进，即终止性的上界。

我们为综合算法第一阶段结束时由过程 Gen 输出的 preMealy 机确定了完成所需迭代次数的上界（在规约由 k-coB¨uchi 自动机 $\mathcal{A}$ 给出且可实现的情况下）。为获得所需的指数级上界，我们依赖于计数函数反链链的最大长度，这些计数函数按如下方式赋予偏序：令 $A \in \mathcal{AC}_{\preceq}(CF(\mathcal{A},\ k))$ 和 $B \in \mathcal{AC}_{\preceq}(CF(\mathcal{A},\ k))$，则当且仅当 $\forall f \in A \cdot \exists g \in B \cdot f \preceq g$时，$A \trianglelefteq_{\mathsf{CF}} B$。这些链的长度有如下上界$k^{\mathbf{O}(n)}$:

引理 4。任意 $\lhd_{\mathsf{CF}}$-链在 $(\mathcal{AC}_{\preceq}(CF(\mathcal{A}, k)) \trianglelefteq_{\mathsf{CF}})$ 中的长度至多为 $k^{\mathbf{O}(n)}$，其中 n 是 $\mathcal{A}$中的状态数。

证明。与引理 12 的证明相同，对于一个由计数函数组成的反链 $X = \{f_1, \ldots, f_n\}$，我们将其关于 $\preceq$ 的向下闭包定义为 $\downarrow X$。于是，给定另一个反链 Y，可得 $X \lhd_{\mathsf{CF}} Y$ 当且仅当 $\downarrow X \subsetneq \downarrow Y$。因此，$\lhd_{\mathsf{CF}}$-链的最大长度不超过计数函数的数量，即 $k^{\mathbf{O}(n)}$。 □ □

检查 $\mathcal{P}$规约 $\mathcal{S} = L_k^{\forall}(\mathcal{A})$ 的可实现性。为了降低复杂度，我们利用确定性自动机$\mathcal{D}(\mathcal{A},\ k)$中存在的一些结构。首先，对于由 $f_1 \preceq f_2$ 定义的偏序 $\preceq$，计数函数集 $CF(\mathcal{A},\ k)$构成一个完备格；若对所有状态 q，$f_1(q) \leq f_2(q)$，则满足该偏序。我们将 $f_1 \bigsqcup f_2$ 记为 f_1，f_2的最小上界，并将 $W_k^{\mathcal{A}}$记为满足规约 $L(\mathcal{D}(\mathcal{A},\ k)[f])$可实现的计数函数 f集合（即

由 $\mathcal{D}(\mathcal{A},\ k)$定义且初始状态为 f的规约）。已知 $W_k^{\mathcal{A}}$关于 $\preceq$ [20]是向下封闭的，因为对于所有 $f_1 \preceq f_2$，任何实现$L(\mathcal{D}(\mathcal{A},\ k)[f_2])$的机器，也会实现 $L(\mathcal{D}(\mathcal{A},\ k)[f_1])$。因此，可以用其关于 $\preceq$的极大元素构成的反链 $\lceil W_k^{\mathcal{A}} \rceil$ 紧凑地表示 $W_k^{\mathcal{A}}$。现在，第一个改进来自以下结果：

引理 *5.* 给定一个 ***preMealy*** $\mathcal{P} = (M, m_0, \Delta)$、一个 ***co-B ¨uchi*** 自动机 $\mathcal{A}$，以及$k \in \mathbb{N}$。对于所有状态 $m \in M$，令 $F^*(m) = \bigsqcup\{f \mid \exists u \in (\mathcal{I}\mathcal{O})^* \cdot \mathit{Post}^*_{\mathcal{P}}(m_0, u) = m \wedge \mathit{Post}_{\mathcal{D}}(f_0, u) = f\}$。那么，$L(\mathcal{D}(\mathcal{A}, k))$ 是 $\mathcal{P}$-可实现的，当且仅当不存在$m \in M$使得 $F^*(m) \notin W_k^{\mathcal{A}}$。

很容易证明，算子 F^* 可以在多项式时间内计算。因此，后一个引理表明，存在一个关于 $|\mathcal{P}|$、$|\mathcal{A}|$、$k \in \mathbb{N}$ 和 $\lceil W_k^{\mathcal{A}} \rceil$ 的大小的多项式时间算法，用于检查 $L^\forall(\mathcal{A})$ 的 $\mathcal{P}$-可实现性。形式化细节见附录 E.1。

我们在本小节最后总结综合算法在由普适 co-B¨uchi 自动机定义的 ω-正则规约上的行为。

定理 6. 给定一个普适 ***coB ¨uchi***自动机 $\mathcal{A}$ 和一个示例集 E，综合算法 SynthLearn 会在存在解时返回一个 ***Mealy*** 机 $\mathcal{M}$，使得 $E \subseteq L(\mathcal{M})$ 且 $L_\omega(\mathcal{M}) \subseteq L^\forall(\mathcal{A})$；其最坏情况下的运行时间关于 $\mathcal{A}$ 的规模为指数级，关于 E的规模为多项式级。否则，它会返回 ***UNREAL***。

注意，算法 4 调用算法 1，而算法 1 本身会调用检查$\mathcal{P}$-可实现性的过程；检查 $\mathcal{P}$-可实现性需要多项式时间，因为我们只需计算不动点并检查其是否安全。

以 ***LTL*** 公式给出的规约 现在我们可以将算法 4 应用于由 LTL 公式 φ给出的规约。事实上，得益于上一小节的结果，要为 LTL 规约提供一种算法，我们只需将 φ 转换为普适 co-B¨uchi 自动机。这可以根据下面的引理完成。众所周知（见 [26]），给定一个定义在两组原子命题 $P_{\mathcal{I}}$ 和 $P_{\mathcal{O}}$上的 LTL 公式 φ，我们可以在指数时间内构造一个普适 co-B¨uchi 自动机 $\mathcal{A}_\varphi$，使得 $L^\forall(\mathcal{A}_\varphi) = [\varphi]$，即 $\mathcal{A}$ 恰好识别满足 φ的词集 $w \in (2^{P_{\mathcal{I}}}2^{P_{\mathcal{O}}})^\omega$。由此我们得到以下定理，它给出了针对示例集$E$ 和 LTL 公式 φ的综合算法的复杂度。该复杂度在最坏情况下可证明是最优的，因为判定 $[\varphi]$ 是否能由 $E = \emptyset$实现（即普通的 LTL 可实现性问题）本身就已经是 2ExpTime-Complete [29]。

定理 7。 给定一个 ***LTL***公式 φ 和一个示例集 E，综合算法 SynthLearn 会返回一个 ***Mealy*** 机 $\mathcal{M}$，使得 $E \subseteq L(\mathcal{M})$ 且$L_\omega(\mathcal{M}) \subseteq [\varphi]$（如果它存在），其最坏情况下的运行时间关于 φ的大小为双指数时间，关于 E的大小为多项式时间。否则，它会返回 ***UNREAL***。

5 实现与案例研究

我们使用 Python 在一个原型工具中实现了上一节的算法 SynthLearn，并使用工具 Acacia-Bonzai [10] 来操作

计数函数反链。我们首先说明用于定义状态合并和补全策略的启发式方法，然后通过一个案例研究展示我们的实现如何运行，该案例研究旨在综合一个电梯控制器。有兴趣的读者可以在附录 A 中找到其他案例研究，包括一个电动自行车控制器，以及互斥的两种变体。

5.1 合并与补全策略

为了实现前几节中的算法，我们需要确定策略，以便在泛化阶段从候选项中选择可能的合并方案，并在完备化阶段选择可能的输出。我们实现的策略如下。

首先，我们考虑一个合并策略 σ_G ，它定义在四元组 $(\mathcal{M}, m, E, X)$ 上，其中 $\mathcal{M}$ 是一个 preMealy 机， m 是 $\mathcal{M}$ 的一个状态， E 是一个示例集， X 是 $\mathcal{M}$ 中允许进行合并的状态子集；该策略返回 X 中一个具有以下性质的状态。给定一个示例 e ，它在当前 preMealy 机中转移到状态 m ，以及在算法 1 第 7 行计算得到的候选合并项集合$\{m_1, m_2, \ldots, m_k\}$，我们将当前 preMealy 机上不动点 F^* 计算出的计数函数与每个状态 m_i 相关联。随后，我们的合并策略会选择集合中一个标记有 $\preceq$-极小元的状态 m_i。直观而言，优先选择极小计数函数，能够尽可能保留示例 e之后可能出现的行为集。事实上，根据引理 15，我们知道，如果 $f_1 \preceq f_2$，那么 $L(\mathcal{D}(\mathcal{A}, k)[f_2]) \subseteq L(\mathcal{D}(\mathcal{A}, k)[f_1])$。

其次，我们考虑一种完备化策略 σ_C ，它是一个定义在所有三元组 $(\mathcal{M}, m, \mathrm{i}\ X)$ 上的函数，其中 $\mathcal{M}$ 是当前的 preMealy 机，其状态集为 M；(m, i) 是 $\mathcal{M}$中的一个空缺；而 $X \subseteq \mathcal{O} \times (M \cup \{\text{新鲜}\})$ 是一个候选对列表 $(\mathrm{o}\ m')$。它返回 X中的一个元素，即 $\sigma_C(\mathcal{M}, m, \mathrm{i}\ X) \in X$ ，并且具有以下性质。请注意，为确保终止，完备化策略 σ_C 必须是惰性的，即如果 $X \setminus (\mathcal{O} \times \{\text{新鲜}\}) \neq \varnothing$，那么 $\sigma_C(\mathcal{M}, m, \mathrm{i}\ X) \notin \mathcal{O} \times \{\text{新鲜}\}$。然后，在可能的候选项集合 $\{(\mathrm{o}1, m_1), (\mathrm{o}2, m_2) \ldots, (\mathrm{o}_k, m_k)\}$中，我们再次优先选择与 F^* 在当前 preMealy 机上计算出的$\preceq$-最小计数函数相关联的状态。

原型中实现的合并与补全策略 我们的工具实现了一种合并 策略 σ_G ：给定一个示例 e ，它在当前 preMealy 机中会到达状态 m，以及根据算法 1 第 7 行计算出的一组合并候选状态 $\{m_1, m_2, \ldots, m_k\}$，我们选择具有 $\preceq$-最小计数函数 $F^*(m_i)$ 的状态 m_i，其定义见引理 5。直观而言，优先选择最小计数函数，可以尽可能保留示例 e之后仍可能出现的行为集合。

我们的工具还实现了一种补全策略 σ_C：对于 preMealy 机 $\mathcal{M}$ 的每个空缺（m, i），从候选对列表中选取一个元素，该元素同样优先选择与 $\preceq$-最小计数函数相关联的状态。

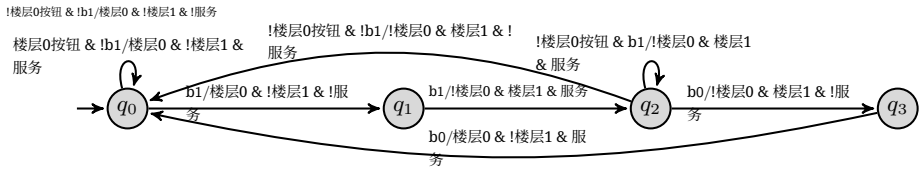


图6：在不含示例的电梯规约上，我们的工具返回的机器。这里， $q0$表示需要时服务楼层0的状态， $q1$ 表示楼层1按钮待处理的状态， $q2$表示服务楼层1的状态， $q3$表示楼层0按钮待处理的状态。

5.2 案例研究

电梯控制器示例 我们将说明如何使用我们的工具为双层电梯系统构造一个合适的控制器。

考虑两层楼，便足以说明更一般的电梯所面临的大多数主要难题。控制器的输入由两个原子命题 b0 和 b1 表示；当用户按下0层（分别为1层）的按钮时，它们为真。输出由原子命题 f0 和 f1 表示；当电梯位于0层（分别为1层）时，它们为真；ser 则在电梯服务当前楼层时为真（即电梯门打开）。此控制器应确保以下核心性质：

1. **功能保证：** 每当按下楼层0的按钮（分别为楼层1的按钮）时，电梯最终必须服务 楼层0（分别为楼层1）：

```
G(b0 -> F (f0 & ser)) & G(b1 -> F (f1 & ser))
```

2. **安全保证：** 电梯始终恰好位于一层：G(f0<->!f1) 3. **安全保证：** 电梯门打开时，电梯不能在两层之间转移：G((f0 & ser) -> X(!f1)) & G((f1 & ser) -> X(!f0)) 4. **初始状态：** 电梯初始时应位于楼层0：f0

此外，我们作出如下**假设**：每当按下楼层0（或楼层1）的按钮时，该按钮必须保持按下状态，直到电梯服务该楼层为止，即 G(b0 -> (b0 W (f0 & ser))) & G(b1 -> (b1 W (f1 & ser)))。

在深入介绍这个示例的细节之前，我们先说明在此示例中使用工具所采用的方法。我们首先只提供上述电梯的高层规格说明 φ_{CORE} 。然后从工具获得第一个 Mealy 机。接着，我们检查该机器，找出我们不满意的行为前缀，并为其提供更好的备选决策。然后，我们在 φ_{CORE} 上运行工具，并输入已确定的示例，从而得到一台新机器；我们以此方式继续迭代，直到对综合得到的米利机感到满意为止。

现在我们来详细说明。当工具接收到这份规约且没有任何示例时，会生成图6所示的机器。该解会使控制器在0层和1层之间切换，有时并无必要。例如，考虑轨迹 s # {!b0 & !b1}{!f0 & f1 & !ser} # {!b0 & !b1}{f0 & !f1 & !ser}，其中我们令 s = {!b0 & b1}{f0 & !f1 & !ser}

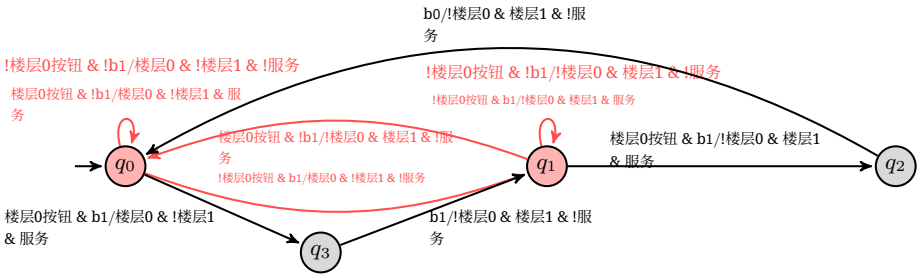


图7：在包含附加示例的电梯规约上，我们的工具返回的 Mealy 机。将示例泛化后、完备化前得到的 preMealy 机以红色突出显示。生成该机器耗时 3.10 秒。

{!b0 & b1}{!f0 & f1 & ser}。在这里，我们注意到，转移会返回到状态 q_0，此时电梯位于0层；而在服务完1层后，电梯本可以继续停留在1层。上述方法使我们能够识别出以下三个示例：

1. 第1条轨迹表明，在服务楼层1后，由于 b0 为假，电梯必须停留在楼层1：s # {!b0 & !b1}{!f0 & f1 & !ser} # {!b0 & !b1}{!f0 & f1 & !ser}2. 第2条轨迹表明，由于 b1 为假，电梯必须停留在楼层0：{!b0 & !b1}{f0 & !f1 & !ser} # {!b0 & !b1}{f0 & !f1 & !ser}3. 第3条轨迹确保在 s 之后，服务楼层1后，服务楼层0时不会出现不必要的延迟：s # {b0 & !b1}{!f0 & f1 & !ser} #{b0 & !b1}{f0 & !f1 & ser}

有了这些附加示例，我们的工具生成了图 7 中的机器，它对这些示例进行了泛化，并确保电梯现在只会在必要时移动。例如，第一条轨迹的末尾被泛化为状态 q_1 上的一个循环，确保除非按下楼层0按钮，否则电梯不会从 1 层前往 0 层。值得注意的是，这里提供的示例数量远小于定理 4 证明的理论（多项式）上界。

6 结论

本文介绍了少量提示合成这一问题。合成问题的这一变体允许用户利用高质量解的预期执行示例来引导综合。现有的综合工具在仅提供高层规格说明时，可能无法给出自然的解；而提供完整规约又有悖于综合的根本目标，因此我们认为，我们的算法在实践中具有更大的应用潜力。

在理论方面，我们详细研究了新综合过程需要解决的问题的计算复杂度。我们证明了该算法是完备的，也就是说，任何实现规约 φ 的 Mealy 机 $\mathcal{M}$ 都可以由我们的算法根据 φ

以及一个足够丰富的示例集 E得到，其大小相对于 $\mathcal{M}$的规模有多项式界。在实践方面，我们已将算法实现为一个原型工具，通过定制的状态合并学习算法扩展了 Acacia-Bonzai [10] 。我们证明，仅需少量示例，就能仅凭高层线性时序逻辑规格生成高质量的机器。该工具目前尚未完全优化。虽然这足以证明我们方法的价值，但我们仍将着力提升实现的效率。

作为未来工作，我们将考虑扩展用户界面，以便通过工具交互式且简洁地指定解决方案输出的（反）例集合。同一方向上，另一个有趣的未来方向是处理参数化示例（例如，将楼层数作为参数的电梯）。这需要提供一种简洁的语法来定义参数化示例，并为这种情形设计高效的综合算法。我们还将考虑构造负例的可能性，因为我们的理论结果可以直接扩展到这种情形，而且将其集成到实现中应该很容易。

参考文献

1. Abadi, M., Lamport, L., Wolper, P.：反应式系统的可实现与不可实现规约。载于：自动机、语言与编程，第16届国际研讨会，ICALP89，意大利斯特雷萨，1989年7月11日至15日，会议论文集。计算机科学讲义，第372卷，第1–17页。Springer（1989） 2. Almagor, S., Kupferman, O., Velner, Y.：在硬布尔约束下最小化期望成本及其在定量综合中的应用。载于：第27届并发理论国际会议，CONCUR 2016，2016年8月23日至26日，加拿大魁北克市。LIPIcs，第59卷，第9:1–9:15页。Schloss Dagstuhl - Leibniz-Zentrum f¨ur Informatik（2016） 3. Alur, R., Bod´ık, R., Dallal, E., Fisman, D., Garg, P., Juniwal, G., Kress-Gazit, H., Madhusudan, P., Martin, M.M.K., Raghothaman, M., Saha, S., Seshia, S.A., Singh, R., Solar-Lezama, A., Torlak, E., Udupa, A.：语法引导综合。载于：可信软件系统工程，第1–25页（2015） 4. Alur, R., Martin, M.M.K., Raghothaman, M., Stergiou, C., Tripakis, S., Udupa, A.：根据场景和需求综合有限状态协议。载于：硬件与软件：验证与测试——第10届国际海法验证会议，HVC 2014，以色列海法，2014年11月18日至20日，会议论文集。计算机科学讲义，第8855卷，第75–91页。Springer（2014） 5. Bloem, R., Chatterjee, K., Henzinger, T.A., Jobstmann, B.：通过定量目标提高综合质量。载于：计算机辅助验证，第21届国际会议，CAV 2009，法国格勒诺布尔，2009年6月26日至7月2日，会议论文集。计算机科学讲义，第5643卷，第140–156页。Springer（2009） 6. Bloem, R., Chatterjee, K., Jobstmann, B.：图博弈与反应式综合。载于：模型检测手册，第921–962页。Springer（2018） 7. Bloem, R., Chockler, H., Ebrahimi, M., Strichman, O.：综合非空洞系统。载于：Bouajjani, A., Monniaux, D.（编），验证、模型检测与抽象解释，第55–72页。施普林格国际出版，Cham（2017）

8. Bohy, A., Bruy`ere, V., Filiot, E., Jin, N., Raskin, J.：Acacia+，一种用于线性时序逻辑综合的工具。载于：计算机辅助验证——第24届国际会议，CAV 2012，美国加利福尼亚州伯克利，2012年7月7日至13日，会议论文集。计算机科学讲义，第7358卷，第652–657页。Springer（2012） 9. Bruy`ere, V., Filiot, E., Randour, M., Raskin, J.：在保证下满足期望：超越定量博弈中的最坏情况综合。《信息与计算》**254**，259–295（2017）。https://doi.org/10.1016/j.ic.2016.10.011, https://doi.org/10. 1016 2016 10 011 / 10 j.ic. . . . Cadilhac, M., Perez, G.A.：Acacia-bonsai：基于下闭集的线性时序逻辑可实现性的现代实现。CoRR **abs/2204.06079** （2022）。https://doi.org/10. 48550 06079 11 2204 06079, 10 48550 2204**28**1 /arXiv. . https://doi.org/ . /arXiv. . . Chandra, A.K., Kozen, D., Stockmeyer, L.J.：交替。《ACM 期刊》（），114–133（1981）。https://doi.org/10.1145/322234.322243, https://doi.org/10.1145/ 322234.322243 12. Clarke, E.M., Henzinger, T.A., Veith, H., Bloem, R.（编）：模型检测手册。Springer（2018）。https://doi.org/10.1007/978-3-319-10575-8, https://doi.org/10 8 13 1007 978 3 319 10575 . / - - - - . Comon, H., Dauchet, M., Gilleron, R., Jacquemard, F., Lugiez, D., L¨oding, C., Tison, S., Tommasi, M.：树自动机技术与应用（2007） 14. Damas, C., Lambeau, B., van Lamsweerde, A.：场景、目标与状态机：模型综合的双赢伙伴关系。载于：第14届ACM SIGSOFT软件工程基础国际研讨会会议论文集，FSE 2006，美国俄勒冈州波特兰，2006年11月5日至11日，第197–207页。ACM（2006） 15. D'Ippolito, N., Braberman, V.A., Piterman, N., Uchitel, S.：综合用于活性目标的非异常基于事件的控制器。《ACM软件工程方法论汇刊》**22**(1)，9:1–9:36（2013）。https://doi.org/10.1145/2430536.2430543, https://doi.org/10 2430543 16 1145 2430536 . / . . Dupont, P., Lambeau, B., Damas, C., van Lamsweerde, A.：QSM 算法及其在软件行为模型归纳中的应用。《应用人工智能》**22**(1&2)，77–115（2008）。https://doi.org/10.1080/08839510701853200, https://doi.org/10 08839510701853200 17 1080. / . Duret-Lutz, A., Renault, E., Colange, M., Renkin, F., Gbaguidi, A., 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？CoRR **abs/2206.11366** （2022）。https://doi.org/10 48550 2206 11366, 10 48550 2206 11366 . / 18 arXiv. . https://doi.org/ . /arXiv. . . Esparza, J., Kretınsky, J., Raskin, J., Sickert, S.：从线性时序逻辑和极限确定性 Büchi 自动机到确定性奇偶自动机。载于：系统构造与分析工具和算法——第23届国际会议，TACAS 2017，作为欧洲软件理论与实践联合会议 ETAPS 2017 的一部分举行，瑞典乌普萨拉，2017年4月22日至29日，会议论文集，第一部分。计算机科学讲义，第10205卷，第426–442页（2017） 19. Faymonville, P., Finkbeiner, B., Tentrup, L.：BoSy：有界综合实验框架。载于：Majumdar, R., Kuncak, V.（编），计算机辅助验证——第29届国际会议，CAV 2017，德国海德堡，2017年7月24日至28日，会议论文集，第二部分。计算机科学讲义，第10427卷，第325–332页。Springer（2017）。https://doi.org/10.1007/978-3-319-63390-9_ 17, 10 1007 978 3 319 63390 9_17 https://doi.org/ . / - - - - . 20 Filiot, E., Jin, N., Raskin, J.：线性时序逻辑可实现性的反链算法。载于：计算机辅助验证，第21届国际会议，CAV 2009，法国格勒诺布尔，

法国，2009年6月26日至7月2日。会议论文集。计算机科学讲义，第5643卷，第263–277页。Springer（2009）21. Filiot, E., Jin, N., Raskin, J.: 线性时序逻辑综合的反链与组合算法。形式化方法与系统设计 **39**(3), 261–296（2011）。https://doi.org/ 10 3 22 1007 10703 011 0115 3, 10 1007 10703 011 0115 . /s - - - https://doi.org/ . /s - - - . Giantamidis, G., Tripakis, S., Basagiannis, S.: 从输入输出轨迹中学习摩尔机。《国际软件工具与技术转移杂志》**23**(1), 1–29（2021）23. Heinz, J., de la Higuera, C., van Zaanen, M.: 计算语言学语法推断。《人类语言技术综合讲座》，Morgan & Claypool 出版社（2015）。https: //doi.org/10.2200/S00643ED1V01Y201504HLT028, https://doi.org/10.2200/ S00643ED1V01Y201504HLT028 24. Kupferman, O.: 关于高质量综合。载于：计算机科学——理论与应用——第11届俄罗斯国际计算机科学研讨会 CSR 2016，俄罗斯圣彼得堡，2016年6月9日至13日，会议论文集。计算机科学讲义，第9691卷，第1–15页。Springer（2016）25. Kupferman, O., Vardi, M.Y.: 时序模型检测中的空洞性检测。载于：Pierre, L., Kropf, T.（编）《硬件设计与验证方法》。第82–98页。Springer Berlin Heidelberg，德国柏林、海德堡（1999）26. Kupferman, O., Vardi, M.Y.: 无 Safra 树决策过程。载于：第46届IEEE计算机科学基础年会（FOCS 2005），2005年10月23日至25日，美国宾夕法尼亚州匹兹堡，会议论文集。第531–542页。IEEE计算机学会（2005）27. Meyer, P.J., Sickert, S., Luttenberger, M.: Strix：显式反应式综合强势回归！载于：计算机辅助验证——第30届国际会议 CAV 2018，作为联合逻辑会议 FloC 2018 的一部分举行，英国牛津，2018年7月14日至17日，会议论文集，第一部分。计算机科学讲义，第10981卷，第578–586页。Springer（2018）28. Natarajan, N., Simmons, D., Datha, N., Jain, P., Gulwani, S.: 实时从少量示例中学习自然程序。载于：Chaudhuri, K., Sugiyama, M.（编）第22届国际人工智能与统计会议，AISTATS 2019，2019年4月16日至18日，日本冲绳那霸。机器学习研究论文集，第89卷，第1714–1722页。PMLR（2019），http://proceedings. mlr.press/v89/natarajan19a.html 29. Pnueli, A., Rosner, R.: 关于异步反应式模块的综合。载于：《自动机、语言与编程》，第16届国际研讨会 ICALP89，意大利斯特雷萨，1989年7月11日至15日，会议论文集。计算机科学讲义，第372卷，第652–671页。Springer（1989）30. Raha, R., Roy, R., Fijalkow, N., Neider, D.: 用于学习线性时序逻辑片段的可扩展随时算法。载于：Fisman, D., Rosu, G.（编）《系统构造与分析工具和算法》——第28届国际会议 TACAS 2022，作为欧洲软件理论与实践联合会议 ETAPS 2022 的一部分举行，德国慕尼黑，2022年4月2日至7日，会议论文集，第一部分。计算机科学讲义，第13243卷，第263–280页。Springer（2022）。 https://doi.org/10.1007/978-3-030-99524-9_14, https: //doi.org/ 10 1007 978 3 030 99524 9_14 . / - - - - . Schewe, S., Finkbeiner, 31 B.: 有界综合。载于：《自动化技术验证与分析》，第5届国际研讨会 ATVA 2007，日本东京，2007年10月22日至25日，会议论文集。计算机科学讲义，第4762卷，第474–488页。Springer（2007）

32. Singh, R., Gulwani, S.: 利用示例转换电子表格数据类型。载于：第43届ACM SIGPLAN-SIGACT编程语言原理研讨会会议论文集，POPL 2016，美国佛罗里达州圣彼得堡，2016年1月20日至22日。第343–356页（2016）。 https://doi.org/10.1145/2837614.2837668, https://doi.org/10 2837668 33 1145 2837614 . / . . Solar-Lezama, A.: 程序草图。STTT15(5-6), 475–495（2013）， https:// doi.org/ 10 1007 10009 012 0249 7 . /s - - - . Solar-Lezama, A., Tancau, L., Bodık, R., Seshia, S.A., Saraswat, V.A.: 有限程序的组合式草图。载于：Shen, J.P., Martonosi, M.（编）第12届编程语言与操作系统架构支持国际会议会议论文集，ASPLOS 2006，美国加利福尼亚州圣何塞，2006年10月21日至25日。第404–415页。ACM（2006）。https://doi.org/10.1145/1168857. 1168907, 1168907 35 10 1145 1168857 https://doi.org/ . / . . Thomas, W.: 无限对象上的自动机。载于：《理论计算机科学手册》，B卷：形式模型与语义（1991）

A 其他示例

在本附录中，我们为感兴趣的读者提供了另外三个示例。

A.1 电动自行车示例

这里，我们旨在为电动自行车的控制器综合出一台 Mealy 机，该控制器负责调节制动系统和电助力。其输入为以下原子命题：brake，当骑行者握下手刹时为真；full，当电池传感器指示电池已充满时为真；speedy，当自行车速度超过 25 千米/小时时为真。其输出为以下原子命题：rim，在轮圈刹车启动时设为真；recharge，在电机刹车启动并为电池充电时为真；assist，在电机为骑行者提供助力时为真。该控制器应确保以下性质：

1. 电池充满时，不能为其充电：

   G(full -> !recharge)

2. 当骑行者不制动时，制动系统均不启动：

   G(!制动 ->(!rim & !充电))

3. 只要自行车速度超过 25 千米/小时，助力就会停用：G(speedy ->!assist) 4. 如果骑行者连续至少三个周期制动，那么两套制动系统中的一套应保持启动，直到骑行者不再制动：G((brake & X brake & XX brake) -> XX((recharge | rim) W !brake)) 5. 如果骑行者连续至少三个周期不制动，那么除非速度超过限值，否则助力应保持启动，直到她再次制动 G((!brake & X(!brake) & XX(!brake)) -> ((XX((!speedy -> assist) W brake))))

6. 只要电机状态发生变化，就应至少保持一个周期的空闲状态：

   G(充电 -> X(!助力)) & G(助力 -> X(!充电))

7. 助力和制动互斥：

   G((充电 | rim) -> !助力)

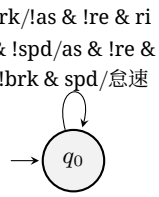


图8：基于电动自行车规约，由我们的工具（不含附加示例）和 Strix 得到的初步机器。

将此规约直接提供给我们的工具或 Strix，而不提供任何示例，我们得到的解永远不会给电池充电。它只有一个状态，并在该状态上循环，标记为[6] brk/!as & !re & ri、!brk & !spd/as & !re & !ri，以及 !brk & spd/idle。图 8 展示了该解。为了获得更好的机器，又不必正式规定电池究竟应在何时充电或不充电，我们向工具提供接下来的两个简单场景（它们是针对第一台机器的反例）：

1. 第一个场景描述了一条轨迹：当骑行者启用手刹（brk 变为 true），且电池未充满（ful 为 false）时，电机刹车会被启用，以便在制动的同时为电池充电（re 被设为 true）：

   {!刹车,速度}{!加速,!倒车,!右转} # {刹车,!满电}{!加速,倒车,!右转} # {刹车,!满电}{!加速,倒车,!右转}

2. 第二种情景描述了一条轨迹：当骑行者拉起手刹（brk 变为 true），且电池充满电（ful 为 true）时，轮圈刹车会被激活（ri 被设为 true）：

   {brk,ful}{!as,!re,ri}

请注意，与输出不同，这两个示例中的输入并不完整：第一个示例没有说明在第一步 ful 是否为 true。同样，记法 {brk,!ful} 也没有说明 spd 是否为 true。这里的集合表示法是工具提供的一种语法，可用于指定非极大集合。因此，第一个轨迹实际上对应 8 个示例，因为有 3 个可能的布尔符号（brk、spd 和 ful）；第二个轨迹则对应 2 个示例。这让用户可以简洁地一次表示多个示例，并专注于提供给工具作为输入的相关布尔信号。

有了这些额外信息，我们的工具输出了图9中的机器，该机器会在制动时以及任何可能的时候为电池充电。在这里，我们注意到，第一条轨迹已被泛化为：每当

[6] 为便于阅读，我们将：– 输出边标记 !as &!re & !ri 替换为术语 idle– 将标记助力替换为 as，将充电替换为 re，将 rim 替换为 ri，将制动替换为 br，并将充满替换为 ful。

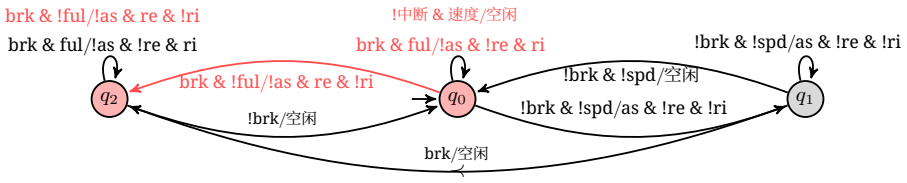


图9：我们的工具根据用户提供的示例生成的电动自行车规约对应的 Mealy 机，如前文所述。红色部分突出显示了在对示例进行泛化后、完备化前得到的 preMealy 机。请注意，我们的算法会尽可能复用泛化阶段创建的状态，因此在本例中，完备化阶段只额外创建了一个状态。

骑行者启动手刹且电池未充满时，控制器就会使用电机刹车并为电池充电。同样，第二条轨迹已被泛化为：每当骑行者启动手刹且电池已充满时，控制器就会使用轮圈刹车。

A.2 带优先级进程的互斥（优先级仲裁器）

此处，我们旨在为三个进程 0、1 和 m 合成一台用于互斥的 Mealy 机，其中进程 m 具有优先级。这在某些情况下会很有用：我们希望让一个进程优先于其他进程，即设置一个主控 进程。以下是该系统的高层规格说明：

1. G(请求 0 ->_F 授予 0) 2. G(请求 1 -> F 授予 1) 3. G(请求 m -> 授予 m) 4. G((!授予 0 ∧-!授予 1)|(!授予- 0 ∧ !授予 m)|(!授予 m ∧ !授予 1))

上述公式对应于典型的互斥规约，并额外增加了一项约束：必须立即授权主控 进程的请求。

将这些规格交给 Strix 后，会得到图 10 所示的机器（request 和 grant 分别缩写为 r 和 g）。

我们注意到，非优先级进程的请求没有被纳入考虑，因此这并不是一个可行的解决方案。我们的工具给出的机器如图 11 所示，情况稍好一些，但仍存在会产生未经请求的授权的轨迹。一个这样的轨迹示例为 {rm & !r0 & r1}{!g0 & !g1 & gm} # {!rm & !r0 & !r1}{g0 & !g1 & gm}，其中进程 1 获得了访问授权，但从未提出请求。

为获得令人满意的解，我们还提供以下执行场景。我们先从符合这种模式的轨迹开始：第一步有两个请求，第二步没有请求。我们通过在第一步授予一个进程、在第二步授予另一个进程来解决这条轨迹：

1. {!rm & r0 & r1}{g0 & !g1 & !gm} # {!rm & !r0 & !r1}

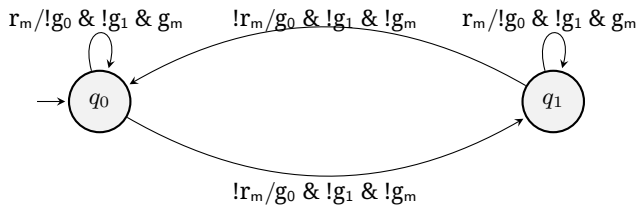


图 10：Strix 返回的优先级仲裁器的 Mealy 机

{!g0 & g1 & !gm} 2.{rm & !r0 & r1}{!g0 & !g1 & gm} # {!rm & !r0 & !r1} {!g0 & g1 & !gm}

现在，我们来处理一个三个进程同时收到请求的轨迹示例：

1.{rm & r0 & r1}{!g0 & !g1 & gm} # {!rm & !r0 & !r1} {g0 & !g1 & !gm}

最后，我们来考察同时请求 rm 和 r0 的轨迹：

1.{rm & r0 & !r1}{!g0 & !g1 & gm} # {!rm & r0 & r1}{g0 & !g1 & !gm} # {!rm & !r0 & !r1}{!g0 & g1 & !gm} 2.{rm & r0 & !r1}{!g0 & !g1 & gm} # {!rm & !r0 & !r1}{g0 & !g1 & !gm} # {!rm & r0 & !r1}{g0 & !g1 & !gm} 3.{rm & r0 & !r1}{!g0 & !g1 & gm} # {!rm & !r0 & r1}{g0 & !g1 & !gm} # {!rm & !r0 & !r1}{!g0 & g1 & !gm}

随后，我们得到图 12 中的机器。我们注意到，没有虚假授权，且进程 0 和进程 1 的请求顺序得到记录并受到遵守。这确保了非优先级进程之间的公平性。

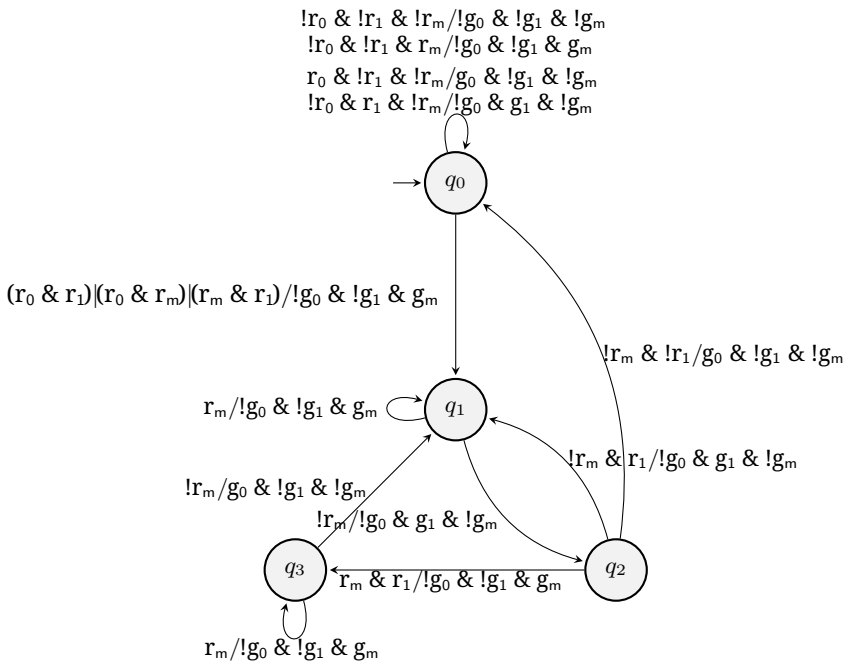


图11：我们的工具在优先仲裁器规约上（不含附加示例）生成的初步机器

A.3 无后续授权的互斥

这里，我们旨在综合一个用于互斥的 Mealy 机，并附加一项性质：实现绝不会连续两次授予权限。这在某些情况下可能很有用：我们希望让获得授权的进程稍作休息，以便执行其他指令。以下是该系统的高层规约：

1. G(请求 0 ->_F grant 0) 2. G(请求 1 -> F grant 1) 3. G(!grant 0 | !grant 1) 4. G(grant 0 -> X (!grant 1 & !grant 0)) 5. G(grant 1 -> X (!grant 1 & !grant 0))

前三个公式对应一个典型的互斥问题，最后两个则规定不能连续发生两次授权。

将这些高层规约提供给 Strix 后，会得到图 13 所示的机器（request 和 grant 分别简写为 r 和 g）。与引言中的示例一样，我们可以看到，Strix 提出的解决方案没有考虑请求，因此不能被视为解决我们问题的有效方案。未提供示例时，我们的工具会生成类似的机器。

为了获得令人满意的解决方案，我们另外提供以下执行场景。首先，让

s ={!r_0 & !r_1}{!g_0& !g_1} # {r_0 & !r_1}{g_0 & !g_1}

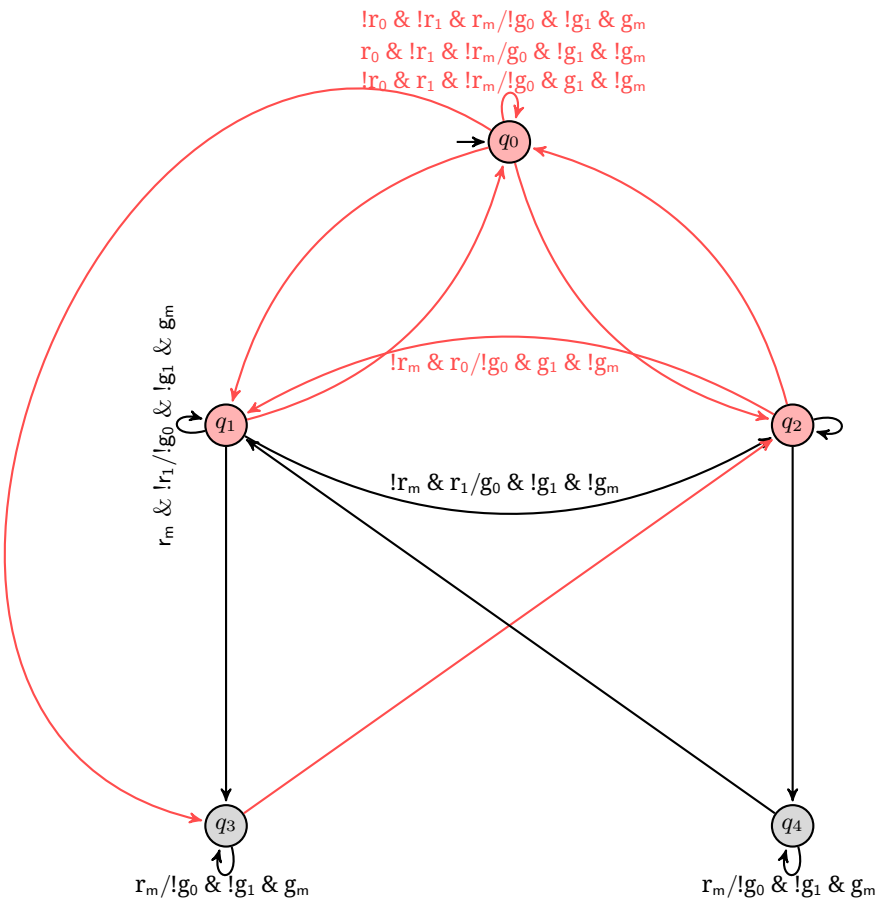


图12：在包含附加示例的优先级仲裁器规约上，我们的工具返回的 Mealy 机。对示例进行泛化后、完备化前得到的 preMealy 机以红色突出显示。在状态 $q0$ 中，没有待处理请求。在状态 $q1$ 中，r_0 处于待处理状态；相应地，在状态 $q2$ 中，r_1 处于待处理状态。在状态 $q3$ 和 $q4$ 中，请求 r_0 和 r_1 均处于待处理状态，但授予请求的顺序很重要。在 $q3$ 中，我们先授予 r_0，然后授予 r_1；在 $q4$ 中，我们先授予 r_1，然后授予 r_0。

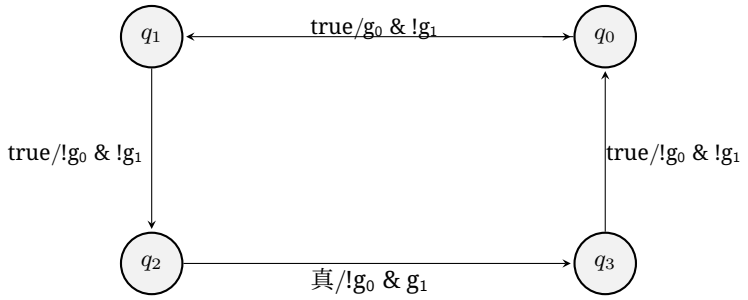


图 13：Strix 返回的无后续授权 Mealy 机

然后，我们通过以下五种不同的延续补全这个简单场景，这些延续展示了高效解决方案对该问题的相关响应：

– 此情境要求在 s 之后发出两个请求时优先处理 r 1 （其中进程 0 已获得授权）：

```
s # {!r0}{!g_0 & !g_1} # {r_0 & r_1}{!g_0 & g_1}
```

– 即使 r 0 已在 s 中获得授权，如果在 s 之后首先发出请求的是进程 0，那么它应优先于另一个进程：

```
s # {r_0 & !r_1}{!g_0 & !g_1} # {true}{g_0 & !g_1}
```

– 然而，如果两个进程在 s 之后同时发出请求，则进程 1 优先：

```
s # {r_0 & r_1}{!g_0 & !g_1} # {!r_0 & !r_1}{!g_0 & g_1}
```

最后，我们得到图 14 中的机器，这是一个自然的解决方案。下面描述这台机器的状态。状态 q_0 表示当前没有待处理的请求，并且可以在下一步执行授权。在状态 q_1 中，没有待处理的请求，但可以在下一步执行授权。在状态 q_2 中， r_1 处于待处理状态，并且可以在下一步进行授权。类似地，在状态 q_3 中，r_0 处于待处理状态，并且可以在下一步执行授权。在状态 q_4 中，两个进程的请求都处于待处理状态，并且可以在下一步进行授权，依此类推。

B 线性时序逻辑语法与语义

为使内容完整自足，我们在此定义线性时序逻辑（LTL）的语法与语义。

给定一组原子命题 P，LTL 的公式按以下语法构造：

$$\varphi := \mathsf{true} \mid p \mid \neg\varphi \mid \varphi_1 \vee \varphi_2 \mid \mathbf{X}\varphi \mid \varphi_1 \mathsf{U} \varphi_2$$

其中， $p \in P$ 是一个原子命题， φ、 φ_1 和 φ_2 是 LTL 公式，“**X**”是下一步 算子，“U”是直到算子。

LTL 公式沿无限字 $w \in (2^P)^\omega$ 的真值按以下方式递归定义：

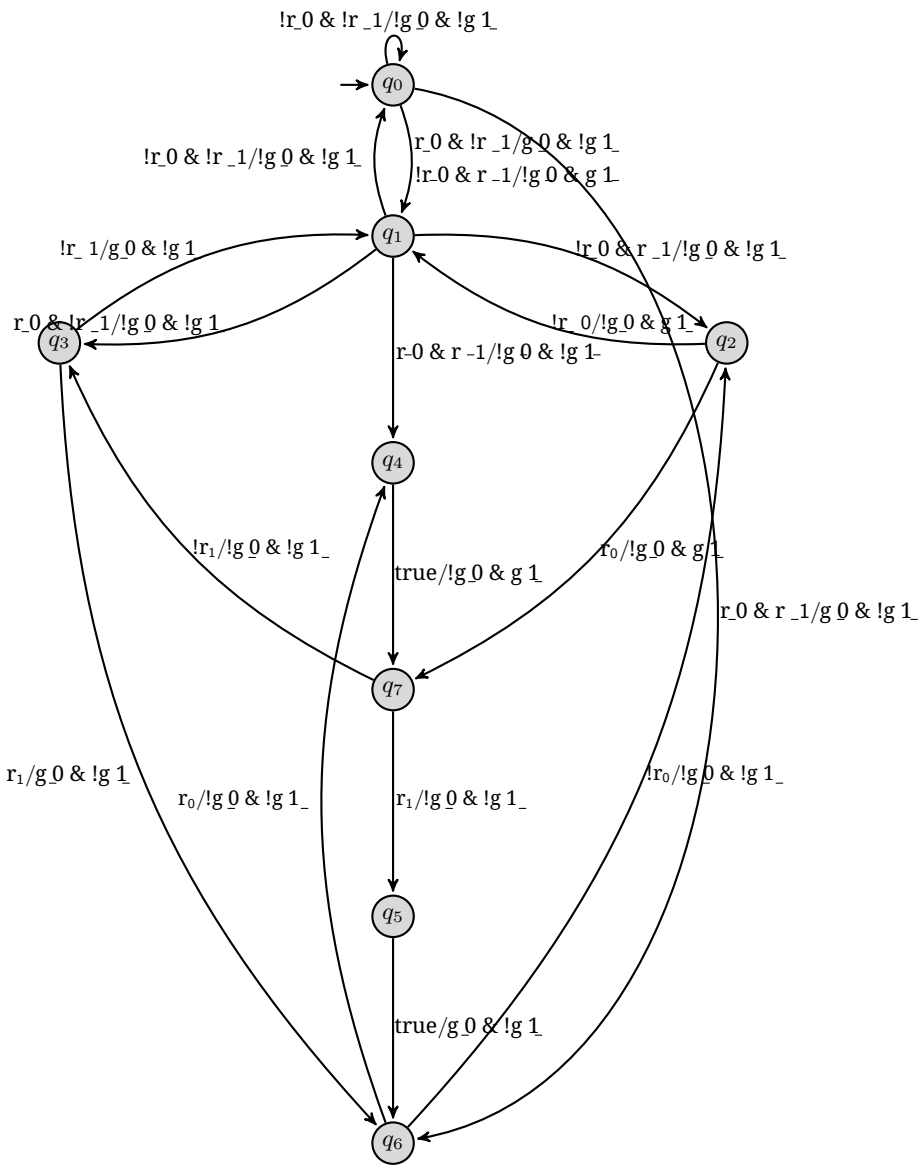


图 14：我们的工具生成的、用于描述不连续授权互斥规约的 Mealy 机。

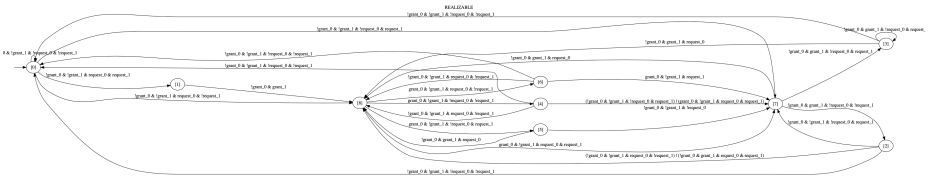


图 15：Strix 对充满仲裁器规约的输出 $n = 2$。

– $w \models$真– $w \models p$ 当且仅当 $p \in w[0]$– $w \models \neg\varphi$ 当且仅当 $w \not\models \varphi$– $w \models \varphi_1 \vee \varphi_2$ 当且仅当 $w \models \varphi_1$ 或 $w \models \varphi_2$– $w \models \mathrm{X}\varphi$ 当且仅当 $w[1\ldots] \models \varphi$ ($w[1\ldots]$ 表示 w 的后缀，该后缀不包含首字母 $w[0]$)– $w \models \varphi_1 \mathrm{U} \varphi_2$ 当且仅当存在 $i \geq 0$，使得 $w[i\ldots] \models \varphi_2$，并且对于所有 j、$0 \leq j < i$、$w[j\ldots] \models \varphi_1$

我们还考虑以下缩写：

– ”最终”：F$\varphi \equiv$真Uφ– ”总是”：G$\varphi \equiv \neg$ F$\neg\varphi$– ”弱直到”： $\varphi_1 \mathrm{W} \varphi_2 \equiv \varphi_1 \mathrm{U} \varphi_2 \vee \mathrm{G}$ φ_1– ”释放”： $\varphi_1 \mathrm{R} \varphi_2 \equiv \neg(\neg\varphi_1 \mathrm{U} \neg\varphi_2)$

C Strix 在充满仲裁器示例上的输出 $n = 2$

图 15 展示了 Strix 对引言中完整互斥规约的输出。

D 第 3 节的详细内容与证明

D.1（pre）-Mealy 机的补充记法

定义 *2*（记法 $f_{\mathrm{i}\Phi}$）。给定一个 *preMealy* 机 $\mathcal{M} = (M, m_{\mathrm{init}}, \Delta)$，我们定义（可能是部分的）函数 $f_{\mathrm{i}\Phi} \colon M \times \mathcal{I}^* \to (\mathcal{I}\mathcal{O})^*$，其定义为 $f_{\mathrm{i}\Phi}(m, i_1 \ldots i_k) = i_1 o_1 \ldots i_k o_k$，满足对于 $1 \leq j \leq k$，$\mathit{Post}_{\mathcal{M}}(m, i_1 \ldots i_j)$ 有定义，且 $o_j = \mathit{Out}_{\mathcal{M}}(m, i_1 \ldots i_j)$。对于所有 $u \in \mathcal{I}^*$，我们用 $f_{\mathrm{i}\Phi}(u)$ 代替 $f_{\mathrm{i}\Phi}(m_{\mathrm{init}}, u)$。注意，$\mathcal{M}$ 接受的语言满足 $L(\mathcal{M}) = \{f_{\mathrm{i}\Phi}(u) \in (\mathcal{I}\mathcal{O})^* \mid f_{\mathrm{i}\Phi}(u)$ 有定义$\}$。

注 *2.* Mealy 机是不含空缺的 preMealy 机。如果 $\mathcal{M}$ 是 Mealy 机，那么 $f_{\mathrm{i}\Phi}^{\mathcal{M}}$的定义域为 $M \times \mathcal{I}^*$。

D.2 泛化阶段：前缀树接受器与状态合并

例 *6.* 考虑 preMealy 机 PTA(E_0)，其中 E_0 已在例 4 中定义。由状态划分 $\{\{0, 2, 3\}, \{1, 4\}\}$ 定义的等价关系 $\sim_0$ 是 PTA(E_0) 的同余，但不是 Mealy 同余，因为我们有 $0 \sim_0 3$，但对于输入 i′，Out$_{\mathrm{PTA}(E_0)}$(0,i′) = o $\neq$ o′ =Out $_{\mathrm{PTA}(E_0)}$(3, i′)。

然而，由状态划分 $\{\{0,1\},\{2,3,4\}\}$ 定义的等价关系 $\sim_0'$ 是 PTA(E_0) 的 Mealy 同余。商 PTA(E_0)/$_{\sim_0'}$ 如图 2 右侧所示，它是一个（完全）Mealy 机。其语言 L 由正则表达式 (i′ + i)o(io +i′o′)* 表示。注意 $E_0 \subset L$。

下述引理指出，将 preMealy 机 $\mathcal{M}$ 按照 Mealy 同余 $\mathcal{M}$ 取商后，得到的仍是一个 preMealy 机，并且它是 $\mathcal{M}$ 的泛化。

引理 6. 如果 $\sim$ 是 $\mathcal{M}$ 的 *Mealy* 同余，那么 $\mathcal{M}/_{\sim}$ 是一个 *preMealy* 机，使得 $L(\mathcal{M}) \subseteq L(\mathcal{M}/_{\sim})$。

证明。这是因为，对于输入序列 v，$\mathcal{M}$ 的任意执行都对应于 $\mathcal{M}/_{\sim}$ 的唯一执行（其状态序列是 $\mathcal{M}$ 中各状态在 v 下对应的等价类构成的序列），并且产生相同的输出。

□ □

对于 preMealy 机 $\mathcal{M}$ 的状态上的等价关系 $\sim$，非同余点是三元组 $(x, x', \mathrm{i}) \in M \times M \times \mathcal{I}$，满足 $x \sim x'$，Post(x, i) 和 Post(x', i) 均有定义，但 Post(x, i) $\not\sim$ Post(x', i)。给定一个非同余点 $p = (x, x', \mathrm{i})$，可以更新等价关系 $\sim$，以消除 p，得到一个更粗的等价关系 $U(\sim, p)$，其对所有 $y, y' \in M$ 均有定义，如下：$yU(\sim, p)y'$ 当 $y \sim y'$，或者 Post(x, i) 和 Post(x', i)，或者 Post(x, i) 和 Post(x', i)。

例 7. 例如，考虑例 4 中定义的 preMealy 机 PTA(E_0)，以及由状态划分 $\{\{0,2\},\{1\},\{3\},\{4\}\}$诱导的等价关系 $\sim_1$。那么，$p_1 = (0, 2, \mathrm{i})$ 是一个非同余点，因为 Post$_{\mathrm{PTA}(E_0)}(0, \mathrm{i}) = 2 \not\sim_1 3 =$ Post$_{\mathrm{PTA}(E_0)}(2, \mathrm{i})$。那么，$\sim_2 = U(\sim_1, p_1)$ 由划分 $\{\{0,2,3\},\{1\},\{4\}\}$ 诱导。接着，$p_2 = (0, 3, \mathrm{i}')$ 对于 $\sim_2$ 是一个非同余点，而 $U(\sim_2, p_2)$ 恰好是例 6 中定义的 $\sim_0$，它是一个同余，因此不包含任何非同余点。

我们用 $\sim \sqsubseteq \sim'$ 表示等价关系 $\sim$ 比某个等价关系 $\sim'$ 更细。以下命题可在附录 D.4 中（很容易）证明：

命题 1。 $U(\sim, p)$ 是一个等价关系，使得 $\sim \sqsubseteq U(\sim, p)$。

给定 M 上的等价关系 $\sim$，现在我们要定义一个过程，不断删除非同余点，也就是不断迭代应用函数 U，直到不再存在非同余点。因此，所得的等价关系是 $\mathcal{M}$ 的同余。我们在引理 7 中证明，删除这些点的顺序无关紧要。我们通过 选择函数 这一概念将其形式化：选择函数 ch 是一个函数，对于 preMealy 机状态上的任意等价关系，它都会输出一个非同余点（如果存在），否则未定义。给定等价关系 $\sim$、选择函数 ch 和两个状态 m，m'，我们将 $\sim^{m,\ ch}{}^{m'}$ 记作序列 $(\sim^{m,\ m',\ n})_{n \geq 0}$ 的不动点，其中对于所有 x，$x' \in M$，$x \sim^{m,\ m',\ 0} x'$，若 $x \sim x'$，或 $x \sim m$ 且 $x' \sim m'$，或 $x \sim m'$ 且 $x' \sim m$

（用等价类表示时，$\sim^{m}$，$_{m'}$，^{0} 会合并 $[m]_{\sim}$ 和 $[m']_{\sim}$）。对于所有 $n>0$，$\sim^{m}$ $\sim^{m}$ $\sim^{m}$ $\sim^{m}$，$_{m'}$，$^{n}=U$（,$_{m'}$，$^{n-1}$，ch(M, ,$_{m'}$，$^{n-1}$)），若 ch(M, ,$_{m'}$，$^{n-1}$) 有定义；否则 $\sim^{m}$，$_{m'}$，$^{n}=\sim^{m}$，$_{m'}$，$^{n-1}$。

例 8。这一收敛序列已经在例 7 中展示过：对于某个特定的选择函数，它先选取 p_1 = (0, 2, i)，然后选取 p_2 =(0, 3, i')，并从由划分 $\{\{s\} \mid s = 0$、.....、$4\}$ 诱导的等价关系 $\sim$ 开始。随后，$\sim^{0}$、2、$^{0}=\sim_{1}$、$\sim^{0}$、2、$^{1}=U(\sim_{1}$、$p_1)$ $=\sim_{2}$以及 $\sim^{0}$、2、$^{3}=U(\sim_{2}p_2)$ $=\sim_{0}$，后者是不动点序列的不动点。请注意，如果采用不同的顺序，先取 p_2，然后取 p_1，我们将得到相同的不动点：$U(\sim_{1}$、$p_2)$ 由划分 $\{\{0,2\},\{3\},\{1,4\}\}$ 诱导，而 $U(U(\sim_{1}$、$p_1))$ 由划分 $\{\{0,2,3\},\{1,4\}\}$ 诱导，因此，$U(U(\sim_{1}$、$p_1)$ $p_2)$ $=U(U(\sim_{1}$、$p_2)$ $p_1)$。这一观察可以推广，正如下述引理所示（证明见附录 D.5）。

引理 7。对于任意等价关系 $\sim$ 和选择函数 ch_1, ch_2，有 $\sim^{m,\ m'}_{ch_1}=\sim^{m,\ m'}_{ch_2}$。

根据前一个引理，给定前 Mealy 机 $\mathcal{M}$ 的状态上的等价关系 $\sim$，对于任意选择函数ch，我们可以定义 $\sim^{m,m'}$，使得 $\sim^{m,\ m'}_{ch}$。注意，$\sim^{m,m'}$ 是 $\mathcal{M}$ 的同余，因为它不再包含非同余点，但它不一定是 $\mathcal{M}$的 Mealy 同余。如果状态 m, m' 属于 $\mathcal{M}$，且 $\sim^{m,m'}$ 是 $\mathcal{M}$的 Mealy 同余，我们就称这两个状态为 $\sim$可合并 。如果 m 和 m' 是$diag_{\mathcal{M}}$-可合并的，我们就称它们是可合并 的，其中 $diag_M$ 是 M上的最细等价关系，即$diag_M = \{(m, m) \mid m \in M\}$。

例*9*。例如，在例 4 中，状态 0 和 2 不可合并，因为 $diag^{0,\ 2}_{E_0}=\sim_{0}$ 不是 Mealy 同余，如例 7 所示。不过，2 和 3 可合并，3 和 4 可合并，1 和 4 也可合并。

我们用 Mergeable($\mathcal{M}$, $\sim$, m, m')（分别）表示谓词：当且仅当 m 和 m' 是 $\sim$-可合并（分别为可合并）时，该谓词成立。当 m 和 m' 是 $\sim$-可合并时，我们令 MergeClass($\mathcal{M}$, $\sim$, m, m') $=\sim^{m,\ m'}$，并令 MergeStates($\mathcal{M}$, $\sim$, m, m') $=\mathcal{M}/_{\sim^{m,\ m'}}$。

引理 *8*。对于任意 *preMealy* 机 $\mathcal{M}$、$\mathcal{M}$ 的状态上的任意等价关系 $\sim$，以及任意 $\sim$-可合并状态 m、m'，有 $\sim \sqsubseteq$ *MergeClass(*$\mathcal{M}, \sim, m, m'$*)*，因此 *L(*$\mathcal{M}$*)* $\subseteq$ *L (MergeStates(*$\mathcal{M}, \sim, m, m'$*))*。此外，若 $m \not\sim m'$，则 $\sim \sqsubset$ *MergeClass(*$\mathcal{M}, \sim, m, m'$*)*，因此 *MergeStates(*$\mathcal{M}, \sim, m, m'$*)* 的状态数严格少于 $\mathcal{M}/_{\sim}$。

证明。根据引理 6 和 $\sim$-可合并状态的定义可立即得出。 □ □

D.3 算法GEN（算法1）的终止性和正确性

我们首先证明，算法 GEN 确实泛化了示例，同时保持规约的可实现性。

引理 *9.* 对于任意合并策略 σ_G、任意有限示例集 E 以及任意以确定性安全自动机 $\mathcal{A}$给出的规约 $\mathcal{S}$，若 $\textsc{Gen}(E, \mathcal{S}, \sigma_G) \neq UNREAL$，则 $\textsc{Gen}(E, \mathcal{S}, \sigma_G)$ 是一个 ***preMealy*** 机 $\mathcal{M}$，满足 $\mathcal{S}$是 $\mathcal{M}$-可实现的，且 $E \subseteq L(\mathcal{M})$。若 $\textsc{Gen}(E, \mathcal{S}, \sigma_G) = UNREAL$，则不存在这样的 ***preMealy*** 机。此外， $\textsc{Gen}(E, \mathcal{S}, \sigma_G)$ 的终止时间关于 E 的大小[7] 是多项式级的，关于 $\mathcal{A}$的状态数 n 是指数级的。

证明。 假设 $\textsc{Gen}(E, \mathcal{S}, \sigma_G) \neq UNREAL$。那么 E必然是一致的，并且$\mathcal{S}$是 PTA(E)-可实现的。根据引理 8，第 4 行的循环会为 PTA(E)计算出越来越粗的 Mealy 同余。因此，若将算法退出循环后计算得到的关系记为 $\sim_E$，则有 $diag_E \sqsubseteq \sim_E$，进而有 $L(\text{PTA}(E)/_{diag_E}) = L(\text{PTA}(E)) = E \subseteq L(\text{PTA}(E)/_{\sim_E})$。此外，第 6 行还确保，对于循环迭代期间计算出的所有等价关系 $\sim$， $\mathcal{S}$ 都是 PTA(E)/$_\sim$-可实现的，特别是对于 $\sim_E$也是如此。

现在，假设 $\textsc{Gen}(E, \mathcal{S}, \sigma_G) = UNREAL$，那么要么 E不一致，此时显然不存在满足 $E \subseteq L(\mathcal{M})$ 的 preMealy 机 $\mathcal{M}$；要么 $\mathcal{S}$ 不是 PTA(E)-可实现的。对于第二种情况，假设存在一个 preMealy 机 $\mathcal{M}$，使得 $E \subseteq L(\mathcal{M})$，且 $\mathcal{S}$ 可由某个机器$\mathcal{P}$实现为 $\mathcal{M}$-可实现的；下面导出矛盾。我们证明， $\mathcal{S}$ 可由一个 Mealy 机 $\mathcal{P}'$ 实现为 PTA(E)-可实现的，该 Mealy 机由 $\mathcal{P}$ 与 PTA(E)的同步积得到： $\mathcal{P}'$ 的一个状态要么是 $\mathcal{P}$ 的状态 p，要么是一个对 (p, e)，其中 e 是 PTA(E)的一个状态。积的初始状态为 (p_0, ϵ)，其中 p_0 是 $\mathcal{P}$的初始状态。对于状态 (p, e) 和输入 i $\in \mathcal{I}$，若对于某个 o $\in \mathcal{O}$， $\text{Out}_{\mathcal{P}}(p, \text{i}) = \text{Out}^{\text{PTA}(E)}(e, \text{i}) =$ o，则积转移到 (o, ($\text{Post}_{\mathcal{P}}(p, \text{i})$, $\text{Post}^{\text{PTA}(E)}$))；否则，转移到 $\Delta_{\mathcal{P}}(p, \text{i})$。这是正确的，因为 $L(\text{PTA}(E)) = \text{Prefs}(E) \subseteq L(\mathcal{P})$。此外，根据积的定义，PTA($E$) 是 $\mathcal{P}'$ 的子图（在状态重命名意义下）。

关于复杂度，循环会被访问 |Prefs(E)| 次。第 5 行需要多项式时间（对于 preMealy 机中任意两个状态，计算 $\sim^{m,m'}$ 的时间为多项式时间）。根据 2，第 6 行的每次可实现性测试所需时间关于 preMealy 机的状态数为多项式时间，关于 n 为指数级。这里，preMealy 机的状态数小于 E 的大小。总的来说，这便得到了所声称的复杂度。 □ □

D.4 命题 1 的证明

证明。 对于所有 $y \in M$, $y \sim y$，因此 $y\ U(\sim, p)\ y$。根据定义，对称性也显而易见。下面证明传递性。如果 $y\ U(\sim, p)\ y'$ 且 $y'\ U(\sim, p)\ y''$，那么有以下几种情况：

1. $y \sim y'$和 $y' \sim y''$：因此 $y \sim y''$，所以 $y\ U(\sim, p)\ y''$。2. $y \sim y'$和 $y' \sim \text{Post}(x, \text{i})$ 以及 $y'' \sim \text{Post}(x', \text{i})$：因此 $y \sim \text{Post}(x, \text{i})$ 和$y'' \sim \text{Post}(y, \text{i})$，所以 $y\ U(\sim, p)\ y''$。3. $y \sim y'$和 $y'' \sim \text{Post}(x, \text{i})$ 以及 $y' \sim \text{Post}(x', \text{i})$：因此 $y'' \sim \text{Post}(x, \text{i})$ 和$y \sim \text{Post}(x', \text{i})$，所以 $y\ U(\sim, p)\ y''$。

[7] E 的大小定义为 Prefs(E) 的基数

4. $y \sim \text{Post}(x, \text{i})$ 和 $y' \sim \text{Post}(x', \text{i})$，以及 $y' \sim y''$：因此 $y'' \sim \text{Post}(x', \text{i})$，所以 $y\ U(\sim, p)\ y''$。5. $y \sim \text{Post}(x, \text{i})$、 $y' \sim \text{Post}(x', \text{i})$ 和 $y' \sim \text{Post}(x, \text{i})$，以及 $y'' \sim \text{Post}(x', \text{i})$：因此，$y \sim y'$ 和 $y' \sim y''$，这意味着 $y \sim y''$，所以 $y\ U(\sim, p)\ y''$。6. $y \sim \text{Post}(x, \text{i})$、 $y' \sim \text{Post}(x', \text{i})$、 $y'' \sim \text{Post}(x, \text{i})$ 和 $y' \sim \text{Post}(x', \text{i})$：直接得到 $y \sim y''$，所以 $y\ U(\sim, p)\ y''$。其余情形与已证明的情形对称，只需将 y 替换为 y''，并将 y'' 替换为 y。

□
□

D.5 引理7的证明

证明。我们定义等价关系之间的关系 $\rightarrow$ 如下：若存在某个非同余点 p（如果存在），使得 $\sim' = U(\sim, p)$，则有 $\sim \rightarrow \sim'$。我们证明$\rightarrow$ 是局部合流的，根据纽曼引理，这意味着 $\rightarrow$ 是全局合流的。换言之，$\rightarrow$ 的自反传递闭包 $\rightarrow^*$ 满足：只要 $\sim \rightarrow^* \sim_1$ 且 $\sim \rightarrow^* \sim_2$，就存在 $\sim'$ 使得 $\sim_1 \rightarrow^* \sim'$且 $\sim_2 \rightarrow^* \sim'$。局部合流性的证明并不困难，但技术性较强，因为需要考虑许多情况。主要思路是证明：如果 p 和 p'是 $\sim$的两个不同非同余点，那么 p' 是 $U(\sim, p)$的一个非同余点，且 p'是 $U(\sim, p')$的一个非同余点；然后证明$U(U(\sim, p)\ p')\ = U(U(\sim, p')\ p)$。

形式化地，设 $p = (x, y, \text{i})$ $p = (x', y', \text{i}')$ 是关于 $\sim$的两个不同的非同余点。令 $\sim_p = U(\sim, p)$ 且 $\sim_{p'} = U(\sim, p')$。因此，$\sim \rightarrow \sim_p$且 $\sim \rightarrow \sim_{p'}$。假设$\sim_p \neq \sim_{p'}$。我们证明 p' 是 $\sim_p$ 的一个非同余点（对称地，p是 $\sim_{p'}$的一个非同余点）。假设 p' 不是 $\sim_p$ 的非同余点。那么，$\text{Post}_{\mathcal{M}}(x', \text{i}) \sim_p \text{Post}_{\mathcal{M}}(y', \text{i})$。我们还知道，$\text{Post}_{\mathcal{M}}(x', \text{i}) \not\sim \text{Post}_{\mathcal{M}}(y', \text{i})$，因为 p' 是 $\sim$的一个非同余点。根据 $\sim_p$的定义，以下两个对称情形可能发生：

1. $\text{Post}_{\mathcal{M}}(x', \text{i}') \sim \text{Post}_{\mathcal{M}}(x, \text{i})$ 且 $\text{Post}_{\mathcal{M}}(y', \text{i}') \sim \text{Post}_{\mathcal{M}}(y, \text{i})$，或者
2. $\text{Post}_{\mathcal{M}}(x', \text{i}') \sim \text{Post}_{\mathcal{M}}(y, \text{i})$ 且 $\text{Post}_{\mathcal{M}}(y', \text{i}') \sim \text{Post}_{\mathcal{M}}(x, \text{i})$。

我们证明这两种情况都会推出$\sim_p = \sim_{p'}$，这与假设矛盾。考虑第一种情况。那么，

$u \sim_p v$ 当且仅当

$u \sim v$，或者 $u \sim \text{Post}_{\mathcal{M}}(x, \text{i})$ 且 $v \sim \text{Post}_{\mathcal{M}}(y, \text{i})$，或者 $v \sim \text{Post}_{\mathcal{M}}(x, \text{i})$ 且$u \sim \text{Post}_{\mathcal{M}}(y, \text{i})$ 当且仅当

$u \sim v$，或者 $u \sim \text{Post}_{\mathcal{M}}(x', \text{i}')$ 且 $v \sim \text{Post}_{\mathcal{M}}(y', \text{i}')$，或者 $v \sim \text{Post}_{\mathcal{M}}(x', \text{i}')$ 且$u \sim \text{Post}_{\mathcal{M}}(y', \text{i}')$ 当且仅当

$u \sim_{p'} v$。

第二种情况是对称的。我们刚刚证明了 p' 是$\sim_p$ 的一个非同余点；根据对称性，p 是$\sim_{p'}$的一个非同余点。最后，我们证明 $U(\sim_p, p') = U(\sim_{p'}, p)$，从而得出 $\rightarrow$是局部合流的。我们证明$U(\sim_p, p')$比 $U(\sim_{p'}, p)$更细，反方向的证明则完全对称。假设 $u\ U(\sim_p, p')\ v$，下面证明 $u\ U(\sim_{p'}, p)\ v$。因此，存在以下几种情况之一：

1. $u \sim_p v$，2. $u \sim_p \text{Post}_{\mathcal{M}}(x', \text{i}')$ 且 $v \sim_p \text{Post}_{\mathcal{M}}(y', ')$，3. $u \sim_p \text{Post}_{\mathcal{M}}(y', \text{i}')$ 且 $v \sim_p \text{Post}_{\mathcal{M}}(x', ')$。

L　　让我们考虑前两种情况（第三种情况与第二种情况对称）：1. $u \sim_p v$ 意味着以下三种情况之一成立：

(a) $u \sim v$：那么$u \sim_{p'} v$，且 $uU(\sim_{p'}, p)v$。(b) $u \sim \text{Post}_{\mathcal{M}}(x, \text{i})$且 $v \sim \text{Post}_{\mathcal{M}}(y, \text{i})$：那么$u \sim_{p'} \text{Post}_{\mathcal{M}}(x, \text{i})$且 $v \sim_{p'} \text{Post}_{\mathcal{M}}(y, \text{i})$，因此 $u\ U(\sim_{p'}, p)\ v$。(c) $u \sim \text{Post}_{\mathcal{M}}(y, \text{i})$且 $v \sim \text{Post}_{\mathcal{M}}(x, \text{i})$：此情况与前一种情况对称。

2. $u \sim_p \text{Post}_{\mathcal{M}}(x', \text{i}')$且 $v \sim_p \text{Post}_{\mathcal{M}}(y', \text{i}')$意味着以下情况之一成立：(a) $u \sim \text{Post}_{\mathcal{M}}(x', \text{i}')$且 $v \sim \text{Post}_{\mathcal{M}}(y', \text{i}')$：因此，$u \sim_{p'} v$，进而 $u\ U(\sim_{p'} p)\ v$。(b) $u \sim \text{Post}_{\mathcal{M}}(x', \text{i}')$且 $v \sim \text{Post}_{\mathcal{M}}(x, \text{i})$并且$\text{Post}_{\mathcal{M}}(y', \text{i}') \sim \text{Post}_{\mathcal{M}}(y, \text{i})$：因此 $v \sim_{p'} \text{Post}_{\mathcal{M}}(x, i)$且$\text{Post}_{\mathcal{M}}(y', \text{i}') \sim_{p'} \text{Post}_{\mathcal{M}}(y, \text{i})$。根据$\sim_{p'}$的定义，我们还有 $\text{Post}_{\mathcal{M}}(y', \text{i}') \sim_{p'} \text{Post}_{\mathcal{M}}(x', \text{i}')$。由于 $u \sim \text{Post}_{\mathcal{M}}(x', \text{i}')$，我们有 $u \sim_{p'} \text{Post}_{\mathcal{M}}(x', \text{i}')$；根据后一个陈述，我们得到 $u \sim_{p'} \text{Post}_{\mathcal{M}}(y', \text{i}')$，进而 $u \sim_{p'} \text{Post}_{\mathcal{M}}(y, \text{i})$。由此可得 $u\ U(\sim_{p'}, p)\ v$。(c) $u \sim \text{Post}_{\mathcal{M}}(x', \text{i}')$且 $v \sim \text{Post}_{\mathcal{M}}(y, \text{i})$并且 $\text{Post}_{\mathcal{M}}(y', \text{i}') \sim \text{Post}_{\mathcal{M}}(x, \text{i})$：将 y 替换为 x 并将 x 替换为 y后，此情况与后一种情况对称。(d) $u \sim \text{Post}_{\mathcal{M}}(x, \text{i})$并且$\text{Post}_{\mathcal{M}}(x', \text{i}') \sim \text{Post}_{\mathcal{M}}(y, \text{i})$且 $v \sim \text{Post}_{\mathcal{M}}(y', \text{i}')$：根据$\sim_{p'}$的定义，我们有$\text{Post}_{\mathcal{M}}(x', \text{i}') \sim_{p'} \text{Post}_{\mathcal{M}}(y', \text{i}')$，由此得到 $v \sim \text{Post}_{\mathcal{M}}(y, \text{i})$，进而 $v \sim_{p'} \text{Post}_{\mathcal{M}}(y, \text{i})$。由 $u \sim \text{Post}_{\mathcal{M}}(x, \text{i})$可得 $u \sim_{p'} \text{Post}_{\mathcal{M}}(x, \text{i})$。因此，$u\ U(\sim_{p'}, p)\ v$。(e) $u \sim \text{Post}_{\mathcal{M}}(x, \text{i})$并且$\text{Post}_{\mathcal{M}}(x', \text{i}') \sim \text{Post}_{\mathcal{M}}(y, \text{i})$且 $v \sim \text{Post}_{\mathcal{M}}(x, \text{i})$并且$\text{Post}_{\mathcal{M}}(y', \text{i}') \sim \text{Post}_{\mathcal{M}}(y, \text{i})$：我们立即得到 $u \sim v$，因此 $u \sim_{p'} v$，进而 $u\ U(\sim_{p'}, p)\ v$。(f) $u \sim \text{Post}_{\mathcal{M}}(x, \text{i})$并且$\text{Post}_{\mathcal{M}}(x', \text{i}') \sim \text{Post}_{\mathcal{M}}(y, \text{i})$且 $v \sim \text{Post}_{\mathcal{M}}(y, \text{i})$并且$\text{Post}_{\mathcal{M}}(y', \text{i}') \sim \text{Post}_{\mathcal{M}}(x, \text{i})$：由 $u \sim \text{Post}_{\mathcal{M}}(x, \text{i})$可得 $u \sim_{p'} \text{Post}_{\mathcal{M}}(x, \text{i})$；由 $v \sim \text{Post}_{\mathcal{M}}(y, \text{i})$可得 $v \sim_{p'} \text{Post}_{\mathcal{M}}(y, \text{i})$，因此$u\ U(\sim_{p'}, p)\ v$。(g) $u \sim \text{Post}_{\mathcal{M}}(y, \text{i})$并且$\text{Post}_{\mathcal{M}}(x', \text{i}') \sim \text{Post}_{\mathcal{M}}(x, \text{i})$且 $v \sim \text{Post}_{\mathcal{M}}(y', \text{i}')$：交换 x 和 y后，此情况与情况（d）对称。(h) $u \sim \text{Post}_{\mathcal{M}}(y, \text{i})$并且$\text{Post}_{\mathcal{M}}(x', \text{i}') \sim \text{Post}_{\mathcal{M}}(x, \text{i})$且 $v \sim \text{Post}_{\mathcal{M}}(x, \text{i})$并且$\text{Post}_{\mathcal{M}}(y', \text{i}') \sim \text{Post}_{\mathcal{M}}(y, \text{i})$：交换 x和 y后，此情况与情况（e）对称。(i) $u \sim \text{Post}_{\mathcal{M}}(y, \text{i})$并且$\text{Post}_{\mathcal{M}}(x', \text{i}') \sim \text{Post}_{\mathcal{M}}(x, \text{i})$且 $v \sim \text{Post}_{\mathcal{M}}(y, \text{i})$并且$\text{Post}_{\mathcal{M}}(y', \text{i}') \sim \text{Post}_{\mathcal{M}}(x, \text{i})$：交换 x 和 y 后，此情况与情况（f）对称。

□　　□

D.6 完备化阶段的详细信息和结果

本节的目标是证明 $\textsc{Comp}(\mathcal{M}_0, \mathcal{S}, \sigma_C)$ 的正确性和终止性。对于某些补全策略，补全过程可能无法终止

。这是因为补全策略可能会一直选择形如 (o m') 的对，其中 m' 是一个新状态。不过，我们将证明，对于第 3.3 节定义的 惰性 补全策略，过程总是会终止。回顾一下，惰性策略总是优先选择已有状态。我们首先证明正确性。

引理 10. 如果算法 $\textsc{Comp}(\mathcal{M}_0, \mathcal{S}, \sigma_C)$ 终止并返回 *Mealy* 机 $\mathcal{M}$，那么 $\mathcal{S}$ 由 $\mathcal{M}$ 实现为 $\mathcal{M}_0$，即 $\mathcal{S}$ 可由 $\mathcal{M}$ 实现，且 $\mathcal{M}_0$是 $\mathcal{M}$ ($\mathcal{M}_0 \preceq \mathcal{M}$) 的子图。

证明。令 $\mathcal{M}_i$ 为经过第 i 次迭代后计算得到的机器。正如之前所述，第 1 行和第 7 行的检查确保了如下不变式：每个 $\mathcal{M}_i$ $\mathcal{M}_0$ 都实现 $\mathcal{S}$。对于第 1 行的检查，这一点显而易见。由于算法的第 $i+1$ 次迭代将机器 $\mathcal{M}_i$ 补全为机器 $\mathcal{M}_{i+1}$，因此我们得到 $\mathcal{M}_0 \preceq \mathcal{M}_1 \preceq \mathcal{M}_2 \ldots$。所以，如果 $\mathcal{S}$ 可由 $\mathcal{M}_i$ 实现，那么它也可由 $\mathcal{M}_i$ 实现为 $\mathcal{M}_0$，因此第 1 行的检查保证所有机器 $\mathcal{M}_i$ $\mathcal{M}_0$ 都实现 $\mathcal{S}$。此外，第 9 行的候选项列表保证非空。事实上，根据不变式，$\mathcal{M}_i$ 始终可以补全为实现 $\mathcal{S}$ 的 Mealy 机。因此，第 9 行的选择是明确有定义的。于是，如果算法返回一台机器，该机器必然是 Mealy 机，因为它没有空缺；此外，它 $\mathcal{M}_0$实现 $\mathcal{S}$。 □ □

接下来，我们的目标是证明惰性策略的终止性。以下技术引理是证明终止性的关键引理。它给出了一个充分条件，满足该条件时，preMealy 机的某个状态可以被重复用于补全一个空缺。我们需要引入一些记号。给定一个 preMealy 机 $\mathcal{M} = (M, m_0, \Delta)$ 和一个确定性奇偶自动机 $A = (Q, q_0, \mathcal{I} \cup \mathcal{O}, \delta_A, d)$，对于所有 $m \in M$，我们记 $R^{A,\mathcal{M}}_m \subseteq Q$ （或者简称 R_m，如果 A 和 $\mathcal{M}$ 的含义在上下文中明确），表示 A 中从其初始状态出发，读取那些在 $\mathcal{M}$ 中读取后会到达状态 m 的词时能够到达的所有状态。形式化地，

$$R^{A,\mathcal{M}}_m = \{\mathsf{Post}^*_A(q_0, u) \mid u \in (\mathcal{I}\mathcal{O})^*, \mathsf{Post}^*_{\mathcal{M}}(m_0, u) = m\}$$

给定一个子集 $Q' \subseteq Q$ 和一些输入 $u \in (\mathcal{I}\mathcal{O})^*$，我们令 $\mathsf{Post}^*_A(Q, u) = \{\mathsf{Post}^*_A(q, u) \mid q \in Q'\}$。

引理 *11*。设 $\mathcal{S}$ 是一个由（完全[8]）确定性安全自动机 $A = (Q, Q_{usf}, q_0, \delta_A)$ 给出的安全规范。设 $\mathcal{M} = (M, m_0, \delta)$ 是一个 *preMealy*-机器，且 $\mathcal{S}$ 是 $\mathcal{M}$-可实现的。设 (m, i) 是 $\mathcal{M}$ 的一个空缺（如果存在）。对于所有满足 $\in \mathcal{O}$ 且 $m' \in M$ 的 *o*，若 $\mathit{Post}^*_A(R_m, io) \subseteq R_{m'}$，则 $\mathcal{S}$ 对于$\mathcal{M}' = (M, m_0, \delta \cup \{(m,i) \mapsto (o\ m')\})$ 是 $\mathcal{M}'$-可实现的。

证明。我们沿用引理陈述中的记号。设 $\mathcal{M}_t = (M_t, m_0, \Delta_t)$ 是一台实现 $\mathcal{S}$ 且满足 $\mathcal{M}$ 是子图的 Mealy 机

[8] 本文中的自动机根据定义都是完全的，即对于任意状态和任意输入，总是存在一条转移；但我们在此特别强调这一点，因为这是该陈述成立的必要条件。

$\mathcal{M}_t$ 的子图，即 $\mathcal{M} \preceq \mathcal{M}_t$。下标 t 表示 $\mathcal{M}_t$ 的转移函数是全函数。不失一般性，我们假设 $\mathcal{M}_t$ 具有一种特殊形式：离开 $M \subseteq M_t$ 后，就再也不会访问它。形式化地说，对于所有 $u \in \mathcal{I}^*$，如果 $\mathsf{Post}^*_{\mathcal{M}_t}(m_0, u) \notin M$，那么对于所有 $v \in \mathcal{I}^*$，$\mathsf{Post}^*_{\mathcal{M}_t}(m_0, uv) \notin M$。使用额外的一位内存即可修改机器 $\mathcal{M}_t$，使其满足这一假设。

令 $\mathsf{o} = \mathsf{Out}^{\mathcal{M}_t}(m,\mathsf{i})$，并令 $\mathcal{M}' = (M, m_0, \delta' := \delta \cup \{(m,\mathsf{i}) \mapsto (\mathsf{o}\ m')\})$。我们通过重定向从 (m,i) 到 $(\mathsf{o}\ m')$ 的转移，将 $\mathcal{M}_t$ 改造成 Mealy 机 $\mathcal{M}'_t$。形式化地，$\mathcal{M}'_t = (M_t, m_0, \Delta'_t)$，其中

$$\Delta'_t = (\Delta_t \setminus \{(m,\mathsf{i}) \to \Delta_t(m,\mathsf{i})\}) \cup \{(m,\mathsf{i}) \mapsto (\mathsf{o}, m')\}$$

显然，$\mathcal{M}'$ 是 $\mathcal{M}'_t$ 的子图，因为 $\mathcal{M}$ 是 $\mathcal{M}_t$的子图。剩下要证明的是，$\mathcal{M}'_t$ 实现了 $\mathcal{S}$。

假设情况并非如此，并由此导出矛盾。换言之，存在 $w \in L_\omega(\mathcal{M}'_t)$（$u$ 是一个有限词），使得 $w \notin L(A)$。令$u \in (\mathcal{IO})^*$ 为 w的最短不安全前缀，即 w 的一个前缀，使得 A 在 u 上的执行所经过的状态均为安全状态，唯独最后一个状态除外。我们根据 u 在 $\mathcal{M}'_t$ 中的执行以及对新转移 $(m,\mathsf{i}) \mapsto (\mathsf{o}\ m')$（记作 t_{new}）的访问，将其分解。根据我们对 $\mathcal{M}_t$形式的假设，一旦离开$\mathcal{M}$，便不会再访问它。根据 $\mathcal{M}'_t$ 的定义，因此一旦离开 $\mathcal{M}'$，便不会再访问它。因此，$\mathcal{M}'_t$ 在 u上的执行会多次（至少一次）访问 t_{new}，同时保持在 $\mathcal{M}$ 中；在最后一次访问 t_{new}之后，执行会继续由 $\mathcal{M}_t$ 在 u的剩余后缀上进行，并最终到达一个不安全状态。形式化地，存在 $u_1, \ldots, u_k \in (\mathcal{IO})^*$ 和$p_1, p'_1, \ldots, p_{k-1}, p'_{k-1}, p_k \in Q$，使得 $u = u_1 \mathsf{io} u_2 \mathsf{io} \ldots \mathsf{io}\ u_k$，并且

$$\begin{array}{l} q_0 \xrightarrow{u_1}_A\ p_1 \xrightarrow{\mathsf{io}}_A p'_1 \xrightarrow{u_2}_{\mathsf{assist}} p_2 \cdots p'_{k-1} \xrightarrow{u_k} \quad p_k \in Q_{usf} \\ m_0 \xrightarrow{u_1}_{\mathcal{M}} m \xrightarrow{\mathsf{i|o}} m' \xrightarrow{u_2}_{\mathcal{M}} \quad m \ldots m' \quad \xrightarrow{u_k}_{\mathcal{M}_t} m'' \end{array}$$

我们证明，对于所有 $1 \le j \le k-1$，存在 $x_j \in (\mathcal{IO})^*$，使得 $\mathsf{Post}_{\mathcal{M}}(m_0, x_j) = m'$ 且 $\mathsf{Post}_A(p_0, x_j) = p'_j$。我们对 j 使用归纳法进行证明。对于 j =1，注意到我们有 $p_1 \in R^A{}_{,m}{}^{\mathcal{M}}$，因此 $p'_1 \in \mathsf{Post}_A(R^A{}_{,m}{}^{\mathcal{M}}, \mathsf{io})$。所以，$p'_1 \in R^A{}_{,m'}{}^{\mathcal{M}}$。这意味着存在 $x_1 \in (\mathcal{IO})^*$ 满足该断言。假设该命题在秩 $j-1$ 时成立。那么，存在 x_{j-1}，使得 $\mathsf{Post}_{\mathcal{M}}(m_0, x_{j-1}) = m'$ 且 $\mathsf{Post}_A(p_0, x_{j-1}) = p'_{j-1}$。因此，$\mathsf{Post}_{\mathcal{M}}(m_0, x_{j-1}u_j) = m$ 且 $\mathsf{Post}_A(p_0, x_{j-1}u_j) = p_j$。这意味着 $p_j \in R^A{}_{,m}{}^{\mathcal{M}}$，因此 $p'_j \in \mathsf{Post}_A(R^A{}_{,m}{}^{\mathcal{M}}, \mathsf{io})$，从而存在 $x_j \in (\mathcal{IO})^*$，在秩 j 时满足该断言。

现在，考虑词 $x_{k-1}u_k$：$\mathsf{Post}_A(p_0,\ x_{k-1}u_k) = p_k \in Q_{usf}$ 和 $\mathsf{Post}^{\mathcal{M}_t}(m_0,\ x_{k-1}u_k) = m''$。这与 $\mathcal{M}_t$ 实现了 $L(A)$ 相矛盾。 □ □

我们现在可以证明算法 2 的终止性了。为了确定复杂度，我们需要引入一些关于子集的链和反链的概念。设 Y是一个基数为 n的有限集。众所周知，集合 2^Y按包含关系构成偏序。因此，2^Y中的元素的反链是满足以下条件的集合 $\mathcal{Y} \subseteq 2^Y$：对于所有 $Y_1,\ Y_2 \in \mathcal{Y}$, Y_1 和 Y_2 在 $\subseteq$下不可比较。我们将 $\mathcal{AC}_{\subseteq}(Y)$记作

Y上的所有 $\subseteq$-反链的集合。集合 $\mathcal{AC}_{\subseteq}(Y)$可以按照记作 $\trianglelefteq$的偏序进行排序：对于所有 $\mathcal{Y}_1$，$\mathcal{Y}_2 \in \mathcal{AC}_{\subseteq}(Y)$，若对于所有 $Y_1 \in \mathcal{Y}_1$，都存在 $Y_2 \in \mathcal{Y}_2$ 使得 $Y_1 \subseteq Y_2$。众所周知，$(\mathcal{AC}_{\subseteq}(Y)\ \trianglelefteq)$是一个格。$(\mathcal{AC}_{\subseteq}(Y)\ \trianglelefteq)$中的链是一个序列 $\mathcal{Y}_1 \lhd \mathcal{Y}_2 \lhd \cdots \lhd \mathcal{Y}_m$（注意所有关系都是严格的）。以下引理是界定终止时间的关键：

引理 12。设 Y是一个基数为 n 的集合。$(\mathcal{AC}_{\subseteq}(Y)\ \trianglelefteq)$ 中的任意 $\lhd$-链长度至多为 2^n。

证明。给定一个反链 $\mathcal{Y} \in \mathcal{AC}_{\subseteq}(Y)$，令 $\downarrow \mathcal{Y}$ 为 $\mathcal{Y}$ 的向下闭包，即 $\downarrow \mathcal{Y} = \{X \subseteq Y \mid \exists X' \in \mathcal{Y}, X \subseteq X'\}$。现在注意，对于所有$\mathcal{Y}_1$，$\mathcal{Y}_2 \in \mathcal{AC}_{\subseteq}(Y)$，都有 $\mathcal{Y}_1 \lhd \mathcal{Y}_2$ 当且仅当 $\downarrow \mathcal{Y}_1 \subsetneq \downarrow \mathcal{Y}_2$。确实，对于“仅当”方向，假设 $\mathcal{Y}_1 \lhd \mathcal{Y}_2$，并取 $X_1 \in \downarrow\mathcal{Y}_1$。因此，存在 $Y_1 \in \mathcal{Y}_1$ 使得 $X_1 \subseteq Y_1$。所以，存在 $Y_2 \in \mathcal{Y}_2$ 使得 $Y_1 \subseteq Y_2$，这意味着 $X_1 \in \downarrow\mathcal{Y}_2$。反过来，假设 $\downarrow \mathcal{Y}_1 \subsetneq \downarrow \mathcal{Y}_2$。令 $Y_1 \in \mathcal{Y}_1$。那么，$Y_1 \in \downarrow\mathcal{Y}_1$，所以 $Y_1 \in \downarrow\mathcal{Y}_2$，这意味着存在 $Y_2 \in \mathcal{Y}_2$ 使得 $Y_1 \subseteq Y_2$。这一观察表明，反链的任意 $\lhd$-链的长度至多为 2^Y 的子集的极大$\subsetneq$-链的长度，而后者至多为 $2^{|Y|}$。 □ □

现在我们证明终止性（假设补全策略 σ_C 可在指数时间内计算）。

引理 13。若 σ_C是惰性的，则 $\textsc{Comp}(\mathcal{M}_0, S, \sigma_C)$ 的终止时间关于$|\mathcal{I}|$和 $\mathcal{M}_0$中的空缺数量为多项式级，关于定义 $\mathcal{S}$的确定性安全自动机的状态数以及 $\mathcal{M}_0$的状态数为指数级。

证明。设 $A = (Q, q_0, \delta, d)$ 是定义该规约的确定性奇偶自动机。设 $\mathcal{M}_0 = (M_0, m_0, \Delta_0)$ 是一个 preMealy 机，使得 $\mathcal{S}$ 是$\mathcal{M}_0$-可实现的（否则算法会在第 1 行终止）。假设算法不终止，并且对于所有 $i \geq 1$，令 $\mathcal{M}_i = (M_i, m_0, \Delta_i)$ 为 while 循环第 i 次迭代时计算出的前 Mealy 机。对于所有 $i \geq 0$和 $m \in M_i$，我们将 R^i_m 定义为 $R^{A,m}_{\mathcal{M}_i}$的简写。由于 m 是 $\mathcal{M}_i$ 的一个状态，且$\mathcal{M}_i$ 是 $\mathcal{M}_{i+1}$ 的一个子图，因此在 $\mathcal{M}_i$ 中到达 m 的词集包含于在 $\mathcal{M}_{i+1}$中到达 m 的词集。因此我们得到以下单调性：$R^i_m \subseteq R^{i+1}_m$。

令 $\mathcal{X}_i = \{R^i_m \mid m \in M_i\}$。用 $\lceil \mathcal{X}_i \rceil$ 表示 $\mathcal{X}_i$ 中关于包含关系的极大元素。根据单调性，我们得到序列 $(\lceil \mathcal{X}_i \rceil)_{i \geq 0}$最终稳定：存在 α，使得对所有 $i \geq \alpha$，都有 $\lceil \mathcal{X}_i \rceil = \lceil \mathcal{X}_{i+1} \rceil$。

考虑一个针对 $j \geq \alpha$ 的机器 $\mathcal{M}_j$，以及算法在第 j 次迭代时完成的 $\mathcal{M}_j$ 中的空缺 (m, i)，即对于某个 o，有 $\Delta_{j+1}(m, \mathsf{i}) = (\mathsf{o}\ m') \in \mathcal{O}$ and $m' \in M_{j+1}$。我们断言 $m' \in M_j$，也就是说，算法复用了 $\mathcal{M}_j$ 中某个已有的状态。确实，考虑集合 $R^j_m\mathsf{,i,o} = \mathrm{Post}^*_A(R^j_m\mathsf{,io})$。我们有 $R^j_m\mathsf{,i,o} \subseteq R^{j+1}_{m'}$，因为从 (m, i) 到 $(\mathsf{o}\ m')$ 的转移为 $\mathcal{M}_{j+1}$。显然，存在 $X \in \lceil \mathcal{X}_{j+1} \rceil$ 使得 $R^j_m\mathsf{,i,o} \subseteq R^{j+1}_{m'} \subseteq X$，并且由于 $\lceil \mathcal{X}_j \rceil = \lceil \mathcal{X}_{j+1} \rceil$, $X \notin \mathcal{X}_j$。根据 $\mathcal{X}_j$的定义，对于某个 $m'' \in M_j$，有 $X = R^j_{m''}$。因此，$R^j_m\mathsf{,i,o} \subseteq R^j_{m''}$。令 $\mathcal{M}' = (M_j, m_0, \delta_i \cup \{(m, \mathsf{i}) \mapsto (\mathsf{o}\ m'')\}$。根据引理 11，$\mathcal{S}$ 是 $\mathcal{M}'$-可实现的，

因此，在第 j 次迭代时，第 9 行的集合 $candidates$ 包含对 (o m'')。由于选择策略是惰性的，如果 $m' \notin M_j$，它会优先选择 m''。因此，$m' \in M_j$。

我们刚刚证明了，对于所有$j \geq \alpha$， $\mathcal{M}_{j+1}$ 的空缺数严格比 $\mathcal{M}_j$少一个。这意味着，如果 $\mathcal{M}_\alpha$ 有 k 个空缺，那么 $\mathcal{M}_{\alpha+k}$ 就没有空缺，算法也就终止了，这与前提矛盾。

现在，我们来界定终止前 **while**循环的迭代次数。这里只给出主要思路。对于算法的每次迭代 j，我们令 H_j 表示 $\mathcal{M}_j$的空缺数。以下断言已在附录 D.7 中证明：断言。 对于所有 $j<\alpha$，要么 $H_{j+1} = H_j - 1$ 且 $\lceil \mathcal{X}_j \rceil \trianglelefteq \lceil \mathcal{X}_{j+1} \rceil$，要么 $H_{j+1} = H_j + |\mathcal{I}| - 1$ 且 $\lceil \mathcal{X}_j \rceil \triangleleft \lceil \mathcal{X}_{j+1} \rceil$。

将该断言的第一种情况称为减少步骤，另一种情况称为增加步骤。根据引理 12，增加步骤的最大次数受限于 $2^{|Q|}$。这也使我们能够界定减少步骤的次数。附录 D.8 中详述的简单计算表明，终止前的迭代次数受限于 $2^{|Q|} + k_0 + (2^{|Q|} + 1)(|\mathcal{I}| - 1) \in O(k_0 + |\mathcal{I}|.2^{|Q|})$。

因此， Comp终止所需的总体时间复杂度，是由上述表达式界定的迭代次数乘以**while**循环中每次内部计算的时间复杂度。后者主要取决于检查$\mathcal{M}_{o,\ m'}$-可实现性的复杂度。根据定理 2，该复杂度关于$\mathcal{M}_{o,\ m'}$ 的状态数是多项式级的，关于 A的状态数是指数级的。 □ □

D.7 引理 13 中所用断言的证明

我们先来证明这个断言：

断言 在每次迭代 $j < \alpha$中，要么 $H_{j+1} = H_j - 1$ 且 $\lceil \mathcal{X}_j \rceil \trianglelefteq \lceil \mathcal{X}_{j+1} \rceil$，要么 $H_{j+1} = H_j + |\mathcal{I}| - 1$ 且 $\lceil \mathcal{X}_j \rceil \triangleleft \lceil \mathcal{X}_{j+1} \rceil$。

证明。 根据单调性，我们总有 $\lceil \mathcal{X}_j \rceil \trianglelefteq \lceil \mathcal{X}_{j+1} \rceil$。现在，假设在步骤 j，可以重用 $\mathcal{M}_j$ 中的某些状态来完成所选空缺，那么 $H_{j+1} = H_j - 1$。

现在假设在步骤 j 中创建了一个新状态，并令 $h = (m,\mathsf{i})$ 为选定进行完备化的空缺。显然， $H_{j+1} = H_j + |\mathcal{I}| - 1$，因为添加一个新状态会创建 $|\mathcal{I}|$ 个空缺。由于没有状态可以被重复使用，根据引理 11 可知，对于所有 $\mathsf{o} \in \mathcal{O}$ 和所有 $m' \in M_j$，$\mathsf{Post}^*_A(R^j_m,\mathsf{io}) \not\subseteq R^j_{m'}$。因此，特别地，对于所有 $X \in \lceil \mathcal{X}_j \rceil$，$\mathsf{Post}^*_A(R^j_m,\mathsf{io}) \not\subseteq X$。令 $(m,\mathsf{i}) \mapsto (\mathsf{o}\ f)$ 为添加到 $\mathcal{M}_j$ 的新转移，其中 $f \notin M_j$。注意， $R^{j+1}_f = \mathsf{Post}^*_A(R^j_m,\mathsf{io})$。因此，对于所有 $X \in \lceil \mathcal{X}_j \rceil$，$R^{j+1}_f \not\subseteq X$。由于 $R^{j+1}_f \in \mathcal{X}_{j+1}$，存在 $Y \in \lceil \mathcal{X}_{j+1} \rceil$ 使得 $R^{j+1}_f \subseteq Y$。必然有 $Y \notin \lceil \mathcal{X}_{j+1} \rceil$，从而证明了 $\lceil \mathcal{X}_j \rceil \neq \lceil \mathcal{X}_{j+1} \rceil$。因此，$\lceil \mathcal{X}_j \rceil \triangleleft \lceil \mathcal{X}_{j+1} \rceil$。 □ □

D.8 引理 13 证明中给出的上界证明

下面的证明详细说明了为界定终止前 **while** 循环的迭代次数所进行的计算，记该次数为 β。

证明。 令 $j_1 < j_2 < \cdots < j_t \leq \alpha$ 为与增加步骤相对应的迭代。我们还令 $j_{t+1} = \beta$ 和 $j_0 = 0$。对于所有 $0 \leq \ell \leq t+1$，令 x_ℓ 为

机器 $\mathcal{M}_{j_\ell}$ 中空缺的数量。特别地， $x_{t+1} = 0$。于是，我们得到如下关系：

$$x_0 = k_0 \qquad x_{\ell+1} = x_\ell - (j_{\ell+1} - j_\ell - 1) + |\mathcal{I}| - 1$$

事实上，在两次增加步骤 j_ℓ 和 $j_{\ell+1}$ 之间，有 $j_{\ell+1} - j_\ell - 1$ 个减少步骤，而增加步骤 $j_{\ell+1}$ 会添加 $|\mathcal{I}| - 1$ 个空缺。

显然， β 受增加步骤的最大数量与减少步骤的最大数量之和限制。后者对应于

$$\sum_{\ell=0}^{t}(j_{\ell+1} - j_\ell - 1) = \sum_{\ell=0}^{t}(x_\ell - x_{\ell+1} + |\mathcal{I}| - 1)$$

$$= x_0 - x_{t+1} + (t+1)(|\mathcal{I}| - 1) = k_0 + (t+1)(|\mathcal{I}| - 1)$$

此外， t 受 $2^{|Q|}$ 限制，因此

$$\beta \leq 2^{|Q|} + k_0 + (2^{|Q|} + 1)(|\mathcal{I}| - 1) \in O(k_0 + |\mathcal{I}|.2^{|Q|})$$

□ □

D.9 定理 3 的证明

该定理的证明基于附录 D 中证明的一系列引理。下面概述一下本附录的结构：

1. 附录 D.2 对算法 GEN 中使用的概念作出正式定义：前缀树接受器、状态合并，尤其是可合并类及其合并结果的概念，并给出示例。引理 8 陈述了关于合并的性质。2. 附录 D.3 中的引理 9 证明了泛化阶段（算法 GEN）的正确性，并分析了其终止时间。3. 附录 D.6 确立了补全阶段的正确性（引理 10）。随后，分析了采用惰性补全策略时的复杂度（引理 11、12 和 13）。

现在我们已经具备了证明定理 3 所需的一切条件。首先，我们证明该命题中的正确性部分。假设 SYNTHSAFE($E, \mathcal{S}, \sigma_G, \sigma_C$) 终止，那么有两种情况：

1. SYNTHSAFE($E, \mathcal{S}, \sigma_G, \sigma_C$) = UNREAL，接着要么 GEN($E, \mathcal{S}, \sigma_G$) = UNREAL，并由引理 9 得到结果；要么 GEN($E, \mathcal{S}, \sigma_G$) 返回某个 preMealy 机 $\mathcal{M}_0$，使得 $\mathcal{S}$ 不是 $\mathcal{M}_0$-可实现的。这种情况不可能发生：根据引理 9， $\mathcal{S}$ 必然是 $\mathcal{M}_0$-可实现的。2. SYNTHSAFE($E, \mathcal{S}, \sigma_G, \sigma_C$) 返回一台 Mealy 机 $\mathcal{M}$。令 $\mathcal{M}_0$ 为 GEN($E, \mathcal{S}, \sigma_G$) 返回的 preMealy 机。那么，根据引理 9， $\mathcal{S}$ 是 $\mathcal{M}_0$-可实现的，且 $E \subseteq L(\mathcal{M}_0)$。由引理 10 可知， $\mathcal{S}$ 可由 $\mathcal{M}$ 实现为 $\mathcal{M}_0$-可实现。因此， $E \subseteq L(\mathcal{M})$，且 $\mathcal{S}$ 可由 $\mathcal{M}$ 实现。

现在，我们证明惰性策略情形下的终止性及其复杂度。首先，根据引理 9，GEN(E，$\mathcal{S}$，σ_G) 总是会终止，运行时间关于 E 的大小为多项式级，关于 n $\mathcal{A}$的状态数为指数级。如果GEN(E，$\mathcal{S}$，σ_G) $\neq UNREAL$，则它会输出一台通过合并 PTA(E) 的状态得到的 preMealy 机 $\mathcal{M}_0$，因此其状态数小于 E的大小。根据引理 13，COMP($\mathcal{M}_0$，$\mathcal{S}$，σ_c) 会终止，运行时间关于 $|\mathcal{I}|$ 和 $\mathcal{M}_0$ 的空缺数（该数有 $|E|$这一上界）为多项式级，关于 n为指数级。这就得到了所声称的复杂度。

D.10 定理 4 的证明

证明。 我们证明， SYNTHLEARN 的泛化阶段已经完备，也就是说，给定一个精心选择的示例集，它就会返回 $\mathcal{M}$。因此，补全阶段也会立即返回 $\mathcal{M}$ ，因为 $\mathcal{M}$中没有空缺。引理 14 陈述了泛化阶段的 Mealy 完备性结果。

□ □

下面的结果表明，对于给定规约的任意最小 Mealy 机，只要输入一个包含特征示例集的示例集，且该特征示例集的规模相对于机器大小为多项式规模，就能学习得到该 Mealy 机。

引理 14。 对于所有以确定性安全自动机形式给出的规约 $\mathcal{S}$ ，以及所有实现 $\mathcal{S}$的最小 *Mealy* 机 $\mathcal{T}$，都存在一个多项式规模（相对于 $\mathcal{T}$的状态数和转移数）的 $E_{\mathcal{T}} \subseteq (\mathcal{I}\mathcal{O})^*$，使得对于所有合并策略 σ_G 以及所有满足 $E_{\mathcal{T}} \subseteq E \subseteq L(\mathcal{T})$ 的有限集 E，$GEN(E, \mathcal{S}, \sigma_G) = \mathcal{T}$。

证明。我们首先定义特征样本，并概述证明思路。然后给出更正式的细节。令 $\mathcal{T} = (T, t_0, \Delta_{\mathcal{T}})$，对于所有$t \in T$，令 $s_t \in \mathcal{I}^*$ 为到达 t的 $\preceq_{ll}$-最短词，即满足 $\text{Post}^*_{\mathcal{T}}(t_0, s_t) = t$。注意， $s_{t_0} = \epsilon$。由于 $\mathcal{T}$是最短的，对于任意两个满足$t \neq t'$的状态 t, t'，存在唯一的 $\preceq_{ll}$-最短词 $d_{t,t'} \in \mathcal{I}^+$，可区分 t 和 t'，也就是说，当 $\mathcal{T}$分别从 t 和 t' 出发读取 $d_{t,t'}$时，产生的输出序列不同。形式化地说，这意味着如果 $d_{t,t'} = \mathrm{i}_1 \dots \mathrm{i}_n$，则存在 $1 \leq j \leq n$，使得 $\text{Out}_{\mathcal{T}}(t, \mathrm{i}_1 \dots \mathrm{i}_j) \neq \text{Out}_{\mathcal{T}}(t', \mathrm{i}_1 \dots \mathrm{i}_j)$。注意，$d_{t,t'} = d_{t',t}$。现在定义 $E_{\mathcal{T}}$（特征样本）。对于任意一对 $p = (t, \mathrm{i})$，只要 $\Delta_{\mathcal{T}}(t, \mathrm{i})$ 有定义，就令 $e_p = f^{\mathcal{T}}_{\mathbb{i}\mathbb{o}}(s_t \mathrm{i})$（符号 $f_{\mathbb{i}\mathbb{o}}$已在附录 D.1 中定义）。换言之，机器的每个转移都对应一个示例。现在，我们还定义一些示例，以防止 PTA 中的某些状态被合并。对于所有 $t \neq t' \in T$，我们定义示例 $v_{t,t'} = f^{\mathcal{T}}_{\mathbb{i}\mathbb{o}}(s_t d_{t,t'})$，最后令

$$E_{\mathcal{T}} = \{v_{t,t'} \mid t, t' \in T, t \neq t'\} \cup \{e_p \mid p \in T \times \mathcal{I}, \Delta_{\mathcal{T}}(p) \text{ is defined}\}$$

设 E 是一个有限集，使得 $E_{\mathcal{T}} \subseteq E \subseteq L(\mathcal{T})$。为了证明，对于任意规约 $\mathcal{S}$ 和策略 σ_G，GEN(E，$\mathcal{S}$，σ_G) $= \mathcal{T}$（在状态重命名意义下），我们证明以下不变式：算法第 e 次迭代计算出的等价关系 $\sim_e$ 比等价关系 $\sim_{\mathcal{T}}$ 更粗；后者将 PTA(E) 中的两个状态（即 E 的两个示例）视为等价，当且仅当它们在 $\mathcal{T}$ 中到达相同状态。此外，将这两个等价关系限制在所有示例上时，它们相同

$e' \preceq e$。为证明这一结果，我们证明，当且仅当 $e_1 \sim_{\mathcal{T}} e_2$ 时，一个 $\sim_e$ 等价类 $[e_1]$ 可以与另一个 $\sim_e$ 等价类 $[e_2]$ 合并。

现在给出形式证明。下文中，我们假设 E 是前缀闭的，即 $E = \mathrm{Prefs}(E) \cap (\mathcal{IO})^*$。这是不失一般性的，因为算法会在第 2 行首先计算 E 的前缀闭包。我们先证明一个有用的断言。

对于所有 $e \in E$，令 $\Phi(e) \in T$，使得 $\mathrm{Post}^*_{\mathcal{T}}(t_0, e) = \Phi(e)$。给定 e，$e' \in E$，若 $\Phi(e) = \Phi(e')$，则称 e 和 e' 是 $\mathcal{T}$-等价的，记作 $e \sim_T e'$。以下断言表明，$\sim_T$ 是 PTA(E) 的 Mealy 同余，并且将后者对 $\sim_T$ 取商后，恰好得到 $\mathcal{T}$（相差一个状态重命名）。

断言 *1* $\sim_T$ 是 PTA(E) 的 Mealy 同余，并且 $\mathcal{T} = \mathrm{PTA}(E)/_{\sim_T}$（相差一个状态重命名）。

我们给出一些证明该断言的直观思路。详细证明见附录 D.11。由于 $E \subseteq L(\mathcal{T})$，可以证明 $\Phi(\mathrm{Post}^{\mathrm{PTA}(E)}(e, \mathrm{i})) = \mathrm{Post}_{\mathcal{T}}(\Phi(e), \mathrm{i})$（如果存在 $\mathrm{o} \in \mathcal{O}$ 使得 $e\mathrm{io} \in E$；否则 $\mathrm{Post}^{\mathrm{PTA}(E)}(e, \mathrm{i})$ 未定义）。这意味着 $\sim_T$ 是一个同余。类似地，我们还可以得到 $\mathrm{Out}^{\mathrm{PTA}(E)}(e, \mathrm{i}) = \mathrm{Out}_{\mathcal{T}}(\Phi(e), \mathrm{i})$（如果有定义），这意味着 $\sim_T$是 PTA(E) 的 Mealy 同余。为了证明 PTA(E) 关于 $\sim_T$ 的商是 $\mathcal{T}$，我们首先利用这样一个事实：E 中包含 $\mathcal{T}$每个状态的一个样例，因此 $\sim_T$ 的等价类数量与 $\mathcal{T}$的状态数量相同。我们已经看到，当转移有定义时，$\mathrm{PTA}(E)/_{\sim_T}$ 的一次转移产生的输出与 $\mathcal{T}$产生的输出一致，因为 $\sim_T$ 是一个 Mealy 同余。我们证明，对于 $\sim_T$ 的所有等价类和所有输入，$\mathrm{PTA}(E)/_{\sim_T}$ 的转移都有定义，因为 E 中包含 $\mathcal{T}$每条转移的一个样例 e_p。

现在，让我们回到引理的证明。对于所有 $e \in E$，令 $\sim_e$ 为循环在第 4 行经过第 e 次迭代后计算得到的 Mealy 同余。我们证明，对于所有 e，$\sim_T$ 比任意 $\sim_e$ 都更粗，并且将 $\sim_T$ 限制在 e 的 $\preceq_{ll}$-向下闭包上时，它与 $\sim_e$ 相等。这足以得出结论：我们的算法返回 $\mathcal{T}$（至多相差状态重命名）。形式化地，给定 $e \in E$，我们令$\downarrow e = \{e' \in (\mathcal{IO})^* \mid e' \preceq_{ll} e\}$。注意，$\downarrow e \subseteq E$。我们证明以下两个不变式：$\sim_e$ 总是比 $\sim_T$ 更细，并且将 $\sim_T$ 和 $\sim_e$限制在 $\downarrow e$上时，两者相等。

– 不变式 1 对于所有 $e \in E$，$\sim_e \sqsubseteq \sim_T$。**– 不变式 2** 对于所有 e，$\sim_e \cap (\downarrow e)^2 = \sim_T \cap (\downarrow e)^2$。

在证明这些不变式之前，我们先证明**不变式 2** 蕴含引理中的陈述。事实上，令 $e^* = \max_{\preceq_{ll}}(E)$。于是，$\downarrow e^* = E$，因此 $\sim_{e^*} \cap (\downarrow e^*)^2 = \sim_{e^*} = \sim_T$。所以，算法返回的机器是 $\mathrm{PTA}(E)/_{\sim_T}$，由断言 *1* 可得所需结果。

我们将借助一个有用的断言：如果 $\sim$ 比 $\sim_T$更细，且 $x \sim_T y$，那么合并 x 的 $\sim$-等价类与 y 的 $\sim$-等价类，会得到一个比 $\sim_T$更细的等价关系。直观地说，这是因为 $[x]_\sim$ 和 $[y]_\sim$ 是同一个 $\sim_T$-等价类的子集，而在计算 $\sim^{x,y}$ 时递归进行的任何合并也会保持这一性质。详细证明见附录 D.11。

断言 *2*：对于所有 x，$y \in E$，如果 $x \sim_T y$且 $\sim \sqsubseteq \sim_T$，那么 $\sim^{x,\,y} \sqsubseteq \sim_T$。

接下来只需证明 **不变式 1** 和**不变式 2**。我们通过归纳法一起证明它们。

初始化 初始步骤 $e = \epsilon$ 很简单。事实上，$\sim_\epsilon = diag_E$，因此 $\sim_\epsilon \sqsubseteq \sim_T$。此外，$\downarrow \epsilon = \{\epsilon\}$，所以，$\sim_\epsilon \cap (\downarrow \epsilon)^2 = \{(\epsilon, \epsilon)\} = \sim_T \cap (\downarrow \epsilon)^2$。

归纳步骤 现在证明经过一次迭代后，不变量仍然成立。假设它们对 $e \in E$ 成立，并证明对于 $f \in E$也成立，其中 f 是 e 在 E 的 llex 序中的直接后继。在给出形式细节之前，先说明一些直观理解。直观而言，我们证明：当且仅当 $f \sim_T y$时，可以合并 $[f]_{\sim_e}$ 与某个满足 $y \preceq_{ll} e$的 $[y]_{\sim_e}$。为证明"仅当"方向，我们利用这样一个事实：当 $f \not\sim_T y$时，合并它们的 $\sim_e$等价类会产生一个不是 Mealy 同余的同余，因为 $\Phi(f)$ 和 $\Phi(y)$ 可由 $d_{\Phi(f)、\Phi(y)}$区分。

现在进行形式证明。我们根据第 5 行的 $mergeCand$ 是否为空，分为两种情况。如果为空，则证明对于所有 $y \preceq_{ll} e$，$\Phi(y) \neq \Phi(f)$。如果非空，则证明任意 $y \in mergeCand$ 都满足 $\Phi(y) = \Phi(f)$。

– 情况 1：$mergeCand = \varnothing$。在这种情况下，不存在合并，因此 $\sim_e = \sim_f$。根据归纳假设，立即可得**不变式 1**。要证明**不变式 2**，我们需要证明对于所有 $x, y \in \downarrow f$，若 $x \sim_T y$，则 $x \sim_f y$，即 $x \sim_e y$。当 $x, y \in \downarrow e$ 时，根据归纳假设可知该结论成立。若 $x = y = f$，则根据 $\sim_e$ 的自反性，该结论成立。剩下的情况是 $x = f$ 和 $y \preceq_{ll} e$。我们证明这种情况实际上不可能发生，因为否则 Mergeable(PTA(E) $\sim_e, f, y$) 将成立（从而 $mergeCand \neq \varnothing$）。因此，假设 $\Phi(f) = \Phi(y)$，并证明 $\sim^f_{,e\ y}$ 是 PTA(E) 的 Mealy 同余。

根据断言 2，对于所有 $\alpha \sim^f_{,e\ y} \beta$, $\Phi(\alpha) = \Phi(\beta)$，都有结论。现在，假设存在 αio$_1 \in E$和 βio$_2 \in E$，那么，o$_1$ = Out$_\mathcal{T}$($\Phi(\alpha)$,i) =Out$_\mathcal{T}$($\Phi(\beta)$,i) = o$_2$（因为 $E \subseteq L(\mathcal{T})$）。这意味着 $\sim^f_{,e\ y}$是 PTA(E) 的 Mealy-同余，因此 $mergeCand \neq \varnothing$，矛盾。

– 情形 2：$mergeCand \neq \varnothing$。在这种情况下，$f$ 与某个 $y \preceq_{ll} e$之间发生了合并。因此 $\sim_f = \sim^f_{,e\ y}$。

我们先证明不变式 2。需要证明，对于所有满足 $\alpha \sim_T \beta$ 的 $\alpha, \beta \preceq_{ll} f$，都有 $\alpha \sim_f \beta$。如果 $\alpha, \beta \preceq_{ll} e$，那么根据归纳假设，$\alpha \sim_e \beta$，又因为 $\sim_e \sqsubseteq \sim_f$，可得 $\alpha \sim_f \beta$。如果 $\alpha = \beta = f$，则由自反性即可得证。因此，假设 $\alpha = f$ 且 $\beta \preceq_{ll} e$。我们有 $f \sim_T \beta$ 和 $y \sim_T f$，所以$y \sim_T \beta$；又因为 $y, \beta \preceq_{ll} e$，根据归纳假设，$y \sim_e \beta$。直观地说，由于 $\sim_f$ 合并了 f 的 $\sim_e$-等价类与 y 的等价类，并传播了这次合并，因此可得 $\beta \sim_f f \sim_f y$。形式化地，根据 $\sim^f$ 的定义，$_{e\ y\ 0}$，可得 $y \sim^f_{,e\ y\ 0} \beta$。此外，$f \sim^f_{,e\ y\ 0} y$，因此 $f \sim^f_{,e\ y\ 0} \beta$。根据命题 1，$U$ 关于序 $\sqsubseteq$递增，因此，$\sim^f_{,e\ y\ 0} \sqsubseteq \sim_f$，进而 $f \sim_f \beta$，不变式 2 得证。

现在来证明 **不变式 1**。首先，如果 $\Phi(f) = \Phi(y)$，那么根据断言 2，可知对所有 $\alpha \sim_f \beta$, $\Phi(\alpha) = \Phi(\beta)$，即 $\alpha \sim_T \beta$。这表明 $\sim_f \sqsubseteq \sim_T$ **(不变式 1)**。因此，接下来只需证明 $\Phi(f) = \Phi(y)$。假设 $\Phi(f) \neq \Phi(y)$。我们证明 $y \notin mergeCand$，这就产生矛盾；换言之，$\sim^f_{,e\ y}$不可能是 PTA(E) 的 Mealy 同余。注意，$s_{\Phi(y)}$ 是在 $\mathcal{T}$ 中到达 $\Phi(y)$ 的最短词，而 $s_{\Phi(f)}$ 是在 $\mathcal{T}$ 中到达 $\Phi(f)$ 的最短词。因此$s_{\Phi(y)} \sim_T y$ 和 $s_{\Phi(f)} \sim_T f$。由于 $s_{\Phi(f)} \preceq_{ll} y \preceq_{ll} e$ 和 $s_{\Phi(f)} \preceq_{ll} f$，根据 **不变式 2**

可知 $s_{\Phi(f} \sim_f f$ 和 $^{s_{\Phi(y)}} \sim_f {}^{y}$。此外，根据 $\sim_f$ 的定义， $y \sim_f y$。因此，$s_{\Phi(f)} \sim_f s_{\Phi(y)}$。考虑区分 $\Phi(f)$ 和 $\Phi(y)$ 的输入词 $d_{\Phi(f),\Phi(y)}$，并将其分解为 di。根据 E 的定义， $f^{\mathcal{T}}_{\mathrm{i}\circledcirc}(s_{\Phi(f)}d\mathrm{i}) \in E$ 且 $f^{\mathcal{T}}_{\mathrm{i}\circledcirc}(s_{\Phi(y)}d\mathrm{i}) \in E$（记号 $f_{\mathrm{i}\circledcirc}$ 已在附录 D.1 中定义）。存在 $\mathrm{o}_1 \neq \mathrm{o}_2 \in \mathcal{O}$，使得 $f^{\mathcal{T}}_{\mathrm{i}\circledcirc}(s_{\Phi(f)}d\mathrm{i}) = f^{\mathcal{T}}_{\mathrm{i}\circledcirc}(s_{\Phi(f)}d)\mathrm{io}_1 = f^{\mathcal{T}}_{\mathrm{i}\circledcirc}\ s_{\Phi(y)}d\ _2 \sim_f f^{\mathcal{T}}_{\mathrm{i}\circledcirc}\ s_{\Phi(f)}d\ \sim_f f^{\mathcal{T}}_{\mathrm{i}\circledcirc}\ s_{\Phi(y)}d$，且 $f^{\mathcal{T}}_{\mathrm{i}\circledcirc}(s_{\Phi(y)}d\mathrm{i})(\ )\mathrm{io}_2$。由于 是同余，可得 $(\ )(\ )$，但 $\mathrm{Out}^{\mathsf{PTA}(E)}(f^{\mathcal{T}}_{\mathrm{i}\circledcirc}(s_{\Phi(f)}d), \mathrm{i}) = \mathrm{o}_1 \neq \mathrm{o}_2 = \mathrm{Out}^{\mathsf{PTA}(E)}(f^{\mathcal{T}}_{\mathrm{i}\circledcirc}(s_{\Phi(y)}d), \mathrm{i})$。这表明 $\sim_f$ 不是 PTA(E) 的 Mealy 同余，因此 y 和 f 不能合并。因此， $\Phi(f) = \Phi(y)$，证毕。

请注意，该证明不依赖于任何特定规约 $\mathcal{S}$，也不依赖于任何合并策略 σ_G。

□

D.11 引理 14 证明中各断言的证明

断言 1 的证明 首先，我们证明 $\sim_T$ 是一个同余。设 $e \sim_T e'$，且 $\mathrm{i} \in \mathcal{I}$。假设 $\Delta_{\mathsf{PTA}(E)}(e, \mathrm{i})$ 和 $\Delta_{\mathsf{PTA}(E)}(e', \mathrm{i})$ 均有定义，即，对于某些 o、$\mathrm{o}' \in \mathcal{O}$，有 $e\mathrm{io} \in E$，且 $e'\mathrm{io}' \in E$。于是，我们得到 $\Phi(e\mathrm{i}) = \mathrm{Post}_{\mathcal{T}}(\Phi(e), \mathrm{i}) = \mathrm{Post}_{\mathcal{T}}(\Phi(e'), \mathrm{i}) = \Phi(e'\mathrm{i})$。换言之， $e\mathrm{i} \sim_T e'\mathrm{i}$。下面证明 $\mathrm{o} = \mathrm{o}'$。显然，$\mathrm{o} = \mathrm{Out}_{\mathcal{T}}(\Phi(e), \mathrm{i})$，因为 $E = L(\mathrm{PTA}(E)) \subseteq L(\mathcal{T})$。同理，$\mathrm{o}' = \mathrm{Out}_{\mathcal{T}}(\Phi(e'), \mathrm{i})$。因此，由于 $\Phi(e) = \Phi(e')$，可得 $\mathrm{o} = \mathrm{o}'$。显然， $\sim_T$至多有 $|T|$ 个等价类。由于对于所有状态 $t \in T$， E 都包含一个示例 s_t，使得 $\Phi(s_t) = t$，因此 $\sim_T$ 至少有 $|T|$ 个等价类。至此，我们已证明，$\sim_T$ 是 PTA(E) 的一个 Mealy 同余，其等价类数量与 $\mathcal{T}$ 的状态数量相同。对于 $t \in T$，令 $\Phi^{-1}(t) = \{e \in E \mid \Phi(e) = t\}$。请注意，对于某个代表元 $e \in E$，有 $\Phi^{-1}(t) = [e]$。最后，我们证明 $\mathcal{T}$ 与 $\mathrm{PTA}(E)/_{\sim_T}$ 在状态重命名 Φ^{-1} 下相等。首先， $\mathcal{T}$ 的初始状态映射到 $[\epsilon]$，而后者是 $\mathrm{PTA}(E)/_{\sim_T}$ 的初始状态。下面证明 Φ^{-1} 保持 $\mathcal{T}$ 的转移。设 t、i、o t'，使得 $\Delta_{\mathcal{T}}(t, \mathrm{i}) = (\mathrm{o}\ t')$。我们证明 $\Delta_{\mathsf{PTA}(E)/_{\sim_T}}(\Phi^{-1}(t), \mathrm{i}) = (\mathrm{o}\ \Phi^{-1}(t'))$。首先， $s_t \in \Phi^{-1}(t)$，因为$\Phi(s_t) = t$。此外，根据 E 的定义， $e_{t,\mathrm{i}} = f^{\mathcal{T}}_{\mathrm{i}\circledcirc}(s_t\mathrm{i}) = f^{\mathcal{T}}_{\mathrm{i}\circledcirc}(s_t)\mathrm{io} \in E$，因此$\Delta_{\mathsf{PTA}(E)}(f^{\mathcal{T}}_{\mathrm{i}\circledcirc}(s_t), \mathrm{i}) = (\mathrm{o}\ f^{\mathcal{T}}_{\mathrm{i}\circledcirc}(s_t\mathrm{i}))$。此外， $\Phi(s_t\mathrm{i}) = t'$。由此可得 $\Delta_{\mathsf{PTA}(E)/_{\sim_T}}(\Phi^{-1}(t), \mathrm{i}) = (\mathrm{o}\ \Phi^{-1}(t'))$。反向结论则由以下事实得出： $\sim_T$ 是 PTA(E) 的 Mealy 同余，且 PTA(E) 选取的所有输出都与 $\mathcal{T}$ 一致，即 $L(\mathrm{PTA}(E)) = E \subseteq L(\mathcal{T})$。断言 1 证明完毕。

□

断言 2 的证明 根据 $\sim^x$ 的定义，^{y}，存在一些非同余点$p_1, \ldots, p_n \in E \times E \times \mathcal{I}$，使得

$$\sim^{x,y} = U(U(\ldots(U(\sim^{x,y,0}, p_1), \ldots), p_{n-1}), p_n)$$

令 $\sim^0 = \sim^{x,\ y,\ 0}$ 和 $\sim^j = U(\sim^{j-1},\ p_j)$，其中 $1 \leq j \leq n$。我们对 j 进行归纳法，证明 $\sim^j \sqsubseteq \sim_T$。

在秩 $j = 0$， $\alpha \sim^0 \beta$ 意味着以下情况之一：(*i*) $\alpha \sim \beta$；或 (*ii*) $\alpha \sim x$ 且 $\beta \sim y$；或 (*iii*) $\beta \sim x$ 且 $\alpha \sim y$。对于情况 (*i*)，根据假设， $\sim \sqsubseteq \sim_T$，因此得证。对于情况 (*ii*)，根据假设， $x \sim_T y$，并且由于 $\sim \sqsubseteq \sim_T$， $\alpha \sim_T x$ 且 $y \sim_T \beta$。因此，$\alpha \sim_T \beta$。情况 (*iii*) 与 (*ii*) 对称。

在秩 $j > 0$，如果 $p_j = (z_1, z_2, \mathrm{i})$，那么 $\alpha \sim^j \beta$ 意味着以下情况之一：(i) $\alpha \sim^{j-1} \beta$；或 (ii) $\alpha \sim^{j-1} \mathrm{Post}^{\mathrm{PTA}(E)}(z_1, \mathrm{i})$ 且 $\beta \sim^{j-1} \mathrm{Post}^{\mathrm{PTA}(E)}(z_2, \mathrm{i})$；或 (iii) 交换 α 和 β 后得到的 (ii) 的对称情形。对于情况 (i)，由归纳假设可得该结论。对于情况 (ii)，由归纳假设可得 $\Phi(\alpha) = \Phi(\mathrm{Post}^{\mathrm{PTA}(E)}(z_1, \mathrm{i}))$，以及 $\Phi(\beta) = \Phi(\mathrm{Post}^{\mathrm{PTA}(E)}(z_2, \mathrm{i}))$。根据非同余点的定义，我们还有 $z_1 \sim^{j-1} z_2$，因此 $\Phi(z_1) = \Phi(z_2)$。现在，$\Phi(\mathrm{Post}^{\mathrm{PTA}(E)}(z_1, \mathrm{i})) = \mathrm{Post}_{\mathcal{T}}(\Phi(z_1), \mathrm{i}) = \mathrm{Post}_{\mathcal{T}}(\Phi(z_2), \mathrm{i}') = \Phi(\mathrm{Post}^{\mathrm{PTA}(E)}(z_2, \mathrm{i}))$，由此可得 $\Phi(\alpha) = \Phi(\beta)$。情况 (iii) 与 (ii) 对称。断言 2 的证明结束。 □

E 第4节的细节和结果

E.1 优化规约 $\mathcal{P}$-可实现性检查：$\mathcal{D}(\mathcal{A}, k)$

.

如论文正文所述，直接应用定理 2 来检查 $\mathcal{D}(\mathcal{A}, k)$ 是否 $\mathcal{P}$-可实现，会得到双重指数级上界。我们转而证明，利用 $\mathcal{D}(\mathcal{A}, k)$ 的结构可以节省一个指数级，其总结如下定理：

定理 8. 给定一个具有 n 个状态的普适 *co-Büchi* 自动机 $\mathcal{A}$，以及定义安全规约 $S = L_k^{\forall}(\mathcal{A}) = L(\mathcal{D}(\mathcal{A}, k))$ 的 $k \in \mathbb{N}$，还有一个具有 m 个状态和 n_h 个空缺的 *preMealy* 机 $\mathcal{P}$，判断 S 是否 $\mathcal{P}$-可实现是 ExpTime-Complete。

在该定理中，假设 k 以二进制形式给出。为证明上界，我们利用这样一个事实：$\mathcal{D}(\mathcal{A}, k)$ 的状态集构成一个完备格，并具有若干有趣的性质。

定义 *3*（计数函数格）。对于任意 *co-Büchi* 自动机 $\mathcal{A}$ 和 $k \in \mathbb{N}$，定义 $\preceq \subseteq CF(\mathcal{A}, k) \times CF(\mathcal{A}, k)$ 当且仅当对所有 $q \in Q$，都有 $f_1 \preceq f_2$ $f_1(q) \leq f_2(q)$。集合 $(CF(\mathcal{A}, k), \preceq)$ 构成一个完备格，其极小元为 $-1 = \langle -1, -1, \ldots, -1 \rangle$，它表示将值 -1 赋给每个状态 $q \in Q$ 的函数；该格的最小上界算子 $\sqcup$ 定义如下：对于所有 $q \in Q$，$f_1 \sqcup f_2 = f$ 满足 $f(q) = \max(f_1(q), f_2(q))$。此上界算子可推广到任意有限计数函数集 $\mathcal{F} = \{f_1, f_2, \ldots, f_n\} \subseteq CF(\mathcal{A}, k)$，而 $\mathcal{F}$ 的最小上界记作 $\bigsqcup \mathcal{F}$。

$\mathcal{D}(\mathcal{A}, k)$中结构的本质体现在以下一系列结果中。

引理 *15.* 对于所有 *co-Büchi* 自动机 $\mathcal{A}$、所有 $k \in \mathbb{N}$ 以及所有计数函数 $f_1, f_2 \in CF(\mathcal{A}, k)$，只要满足 $f_1 \preceq f_2$，就有 $L(\mathcal{D}(\mathcal{A}, k)[f_2]) \subseteq L(\mathcal{D}(\mathcal{A}, k)[f_1])$；因此，如果 $L(\mathcal{D}(\mathcal{A}, k)[f_2])$ 可实现，那么 $L(\mathcal{D}(\mathcal{A}, k)[f_1])$ 也可实现。

证明。由于计数函数记录了到目前为止访问接受状态的次数，因此从 f_2 开始比从 f_1开始受到更多约束。所以，从 f_2 接受的任何词也会从 f_1接受。 □ □

引理 16（[20]）。对于所有 *co-B¨uchi* 自动机 $\mathcal{A}$、所有 $k \in \mathbb{N}$ 和所有计数函数 $f \in CF(\mathcal{A}, k)$，判断 $L_k^{\forall}(\mathcal{A}[f])$ 是否可实现是 ExpTime-Complete。

推论 1. 对于所有 *co-B¨uchi* 自动机 $\mathcal{A}$，对于所有 $k \in \mathbb{N}$，计数函数集 $W_k^{\mathcal{A}} = \{f \in CF(\mathcal{A}, k) \mid L(\mathcal{D}(\mathcal{A}, k)[f]) \text{ 可实现}\}$，且是 $\preceq$-向下封闭的，并可由其极大元素的 $\preceq$-反链 $\lceil W_k^{\mathcal{A}} \rceil$ 表示于$W_k^{\mathcal{A}}$中。该极大元素集可在关于 $\mathcal{A}$ 的大小和 k的二进制编码长度的指数时间内计算出来。

现在我们提出一个引理，本节稍后将借助它改进求解普适 coB¨uchi 规约的 $\mathcal{P}$-可实现性问题的算法上界。

引理 17. 对于所有 *co-B¨uchi* 自动机 $\mathcal{A}$，对于所有 $k \in \mathbb{N}$，对于所有计数函数集 $\mathcal{F} = \{f_1, f_2, \ldots, f_n\} \subseteq CF(\mathcal{A}, k)$：

$$L(\mathcal{D}(\mathcal{A}, k)[\bigsqcup \mathcal{F}] = \bigcap_{f \in \mathcal{F}} L(\mathcal{D}(\mathcal{A}, k)[f]).$$

证明。由于 $CF(\mathcal{A}, k)$ 是有限的，只需证明对于所有 $f_1, f_2 \in CF(\mathcal{A}, k)$，都有 $L(\mathcal{D}(\mathcal{A}, k)[f_1 \sqcup f_2] = L(\mathcal{D}(\mathcal{A}, k)[f_1]) \cap L(\mathcal{D}(\mathcal{A}, k)[f_1])$。为证明这一性质，我们考虑一个词 $w \in (\mathcal{IO})^\omega$ 以及从 f_1,f_2 和 $f_1 \sqcup f_2$ 出发、在 w 上的运行。我们分别将这些运行记为 r_1、r_2 和 r_1,2。令$r_1 = g_0 g_1 \ldots g_n \ldots$满足 $g_0 = f_1$，$r_2 = h_0 h_1 \ldots h_n \ldots$满足 $h_0 = f_2$，并令r_1满足 $_2 = l_0 l_1 \ldots l_n \ldots$和 $l_0 = f_1 \sqcup f_2$。根据 $\delta^{\mathcal{D}}$的定义，利用归纳法很容易证明，对于所有位置 $i \geq 0$ 以及所有 $q \in \mathcal{A}$，都有 $l_i(q) = \max(g_i(q), h_i(q))$，因此 $l_i = f_i \sqcup g_i$。显然，r_1,2 接受，当且仅当 r_1 和 r_2 都接受。这是因为，对于 $l = f \sqcup g$，我们有，对于 $q \in Q$：$l(q) = k+1$ 当且仅当 $f(q) = k+1$ 或 $g(q) = k+1$。

□ □

定理 *8* 的证明 现在我们可以证明该命题了。给定一个 preMealy $\mathcal{P} = (M, m_0, \Delta)$、co-B¨uchi 自动机 $\mathcal{A} = (Q, q_{\text{init}}, \Sigma, \delta, d)$ 和 $k \in \mathbb{N}$，我们可以根据推论 1 在指数时间内计算出 $\preceq$-反链 $\lceil W_k^{\mathcal{A}} \rceil \subseteq CF(\mathcal{A}, k)$。接下来，为了判断 $\mathcal{P}$ 能否补全为一个实现 $L(\mathcal{D}(\mathcal{A}, k))$ 的（充满） Mealy 机，我们通过函数 $F^*: M \to CF(\mathcal{A}, k)$ 为 $\mathcal{P}$ 中的所有 $m \in M$定义状态标记，具体如下：

$$F^*(m) = \bigsqcup \{f \mid \exists u \in (\mathcal{IO})^* \cdot \text{Post}_{\mathcal{P}}^*(m_0, u) = m \wedge \delta^{\mathcal{D}}(f_0, u) = f\}$$

现在我们的目标是证明 F^* 可以在多项式时间内计算。为此，我们首先定义如下函数序列（$F_j: M \to CF(\mathcal{A}, k)$)$_{j \in \mathbb{N}}$：
– 对于 $\mathcal{P}$ 的所有状态 $m \in M$，令 $F_0(m) = f_0$，当 $m = m_0$ 时成立；否则令 $F_0(m) = -1$。– 对于 $j > 0$，对于 $\mathcal{P}$ 的所有状态 $m \in M$，令 $F_j(m) = \bigsqcup_{(m_1,\text{i},\text{o}\ m)|\Delta(m_1,\text{i})=(\text{o}\ m)} \delta^{\mathcal{D}}(F_{j-1}(m_1),(\text{i},\text{o}))$，其中 $\delta^{\mathcal{D}}$ 是 $\mathcal{D}(\mathcal{A}, k)$ 的转移函数。下面的引理将这一函数序列的性质形式化。

引理 18. 序列 $(F_j\colon M \to CF(\mathcal{A}, k))_{j\in\mathbb{N}}$ 满足：

1. 序列至多经过 $|M| \times |Q| \times (k+1)$ 步后稳定。注意，序列 $(F_j)_{j\in\mathbb{N}}$ 稳定时所对应的函数。*2.* 每次迭代的计算时间有上界 $\mathbf{O}(|M|^2 \times |Q|)$。*3.* 对所有 $m \in M$，$F^* = G^*$。

因此，存在一个关于 $|\mathcal{P}|$，$|\mathcal{A}|$，$k \in \mathbb{N}$ 和 $\lceil W_k^{\mathcal{A}} \rceil$ 的大小的多项式时间算法，用于检查 $L^{\forall}(\mathcal{A})$ 的 $\mathcal{P}$-可实现性。

证明。对于第 (1) 点，我们首先注意到，对于 $\mathcal{M}$ 的每个状态 m，计数函数序列 $(F_j(\cdot))_{j\in\mathbb{N}}$ 会在 $|\mathcal{M}| \times |Q| \times (k+1)$ 后稳定。事实上，对于所有$j \geq 0$以及所有 $m \in M$，都有 $F_j(m) \preceq F_{j+1}(m)$。由于计数函数格中的链 $CF(\mathcal{A}, k)$ 的长度至多为 $|Q| \times (k+1)$，因此每个 m 至多更新这么多次。于是，由 F_j 定义的 $|M|$ 状态标签稳定前的迭代总数至多为$|M| \times |Q| \times (k+1)$。

对于第（2）点，我们注意到，计数函数 $F_j(m)$是通过对至多 $|M|$ 个计数函数应用 $\mathcal{D}(\mathcal{A}, k)$的转移函数，并取所得结果的最小上界来计算的；这些计数函数由 F_{j-1}定义。需要注意的是，为此我们不必构造整个自动机$\mathcal{D}(\mathcal{A}, k)$，因为我们可以根据定义1(3)按需计算转移。因此，更新一个状态 m 的复杂度为$\mathbf{O}(|M| \times |Q|)$，而更新所有状态的总体复杂度为$\mathbf{O}(|M|^2 \times |Q|)$。

对于（3），我们根据 Δ 和 $\delta^{\mathcal{D}}$的定义，通过归纳法证明，对于所有 $j \geq 0$ 和 $m \in M$：

$$F_j(m) = \bigsqcup \{f \mid 0 \leq i \leq j \wedge \exists u \in (\mathcal{I}\mathcal{O})^i \cdot \mathsf{Post}^*(m_0, u) = m \wedge \delta^{\mathcal{D}}(f_0, u) = f\}.$$

□ □

现在我们证明，$L(\mathcal{D}(\mathcal{A}, k))$是 $\mathcal{P}$-可实现的，当且仅当不存在 $m \in M$使得 $F^*(m) \notin W_k^{\mathcal{A}}$。按照定理 1的证明，我们知道，$L(\mathcal{D}(\mathcal{A}, k))$是 $\mathcal{P}$-可实现的，当且仅当 $L_\omega(\mathcal{P}) \subseteq L(\mathcal{D}(\mathcal{A}, k))$，并且对于 $\mathcal{P}$的每个空缺$h = (p, \mathsf{i})$，都存在$\mathsf{o}_h \in \mathcal{O}$和一台 Mealy 机 $\mathcal{M}_h$，使得对于所有$u \in \mathsf{Left}_p$，$\mathcal{M}_h$ 实现$(u\mathsf{i}\mathsf{o}_h)^{-1}L(\mathcal{D}(\mathcal{A}, k))$。

首先，检查 $L_\omega(\mathcal{P}) \subseteq L(\mathcal{D}(\mathcal{A},\ k))$，可以通过验证$F^*(m)(q) \neq k+1$对所有 $m \in M$是否成立来完成。其次，检查 $\mathcal{M}_h$是否存在，等价于检查$\bigcap_{u\mathsf{i}\in \mathsf{Left}_p} (u\mathsf{i})^{-1}L(\mathcal{D}(\mathcal{A},\ k))$ 是否可实现。根据引理 17 和第 (3) 点，这又等价于检查 $L(\mathcal{D}(\mathcal{A},\ k)[F^*(p)])$ 是否可实现，而这等价于检查 $F^*(p) \in W_k^{\mathcal{A}}$ 是否成立。两项检查都可以在关于 $\mathcal{A}$ 的大小和 $\lceil W_k^{\mathcal{A}} \rceil$的大小的多项式时间内完成。

我们已经证明，给定一个 preMealy $\mathcal{P}$、co-B¨uchi 自动机 $\mathcal{A}$, $k \in \mathbb{N}$ 以及 $\preceq$-反链 $\lceil W_k^{\mathcal{A}} \rceil$，我们可以在这些输入的规模所对应的多项式时间内计算函数序列 $(F_j\colon M \to CF(\mathcal{A},\ k))_{j\in\mathbb{N}}$，从而判断（根据第 (4) 点）规约 $L(\mathcal{D}(\mathcal{A},\ k))$ 是否可 $\mathcal{P}$ 实现。根据推论 1，反链 $\lceil W_k^{\mathcal{A}} \rceil$ 可以在与 $\mathcal{A}$ 的规模相对应的指数时间内计算。

以及 k 的编码。由此我们证明了定理 8 的上界。下界直接推论自定理 2 和定理 5：定理 2 证明了对于任何以普适 coB¨uchi 自动机 $\mathcal{A}$ 给出的规约，下界都成立；定理 5 则将 $L^{\forall}(\mathcal{A})$ 的 $\mathcal{P}$ 可实现性归约为 $L_k^{\forall}(\mathcal{A})$ 的 $\mathcal{P}$ 可实现性，其中 k 的状态数关于 $\mathcal{A}$ 的状态数呈指数级增长。由于 k 采用二进制表示，其规模仍为多项式，因此该归约是多项式的。定理 8 的证明至此结束。

□