arXiv:2601.14078v1 [cs.FL] 20 Jan 2026

从树到类树：异步自动机的分布式与综合

Mathieu Lehaut[1 [0000−0002−6205−0682]⋆]、
Anca Muscholl[2 [0000−0002−8214−204X]⋆⋆†] 和 Nir
Piterman[3 [0000−0002−8242−5357]⋆⋆⋆†]

[1] 华沙大学，波兰[2] LaBRI，波尔多大学，法国[3] 哥德堡大学和查尔姆斯理工大学，哥德堡，瑞典

摘要。 我们重新审视受限设置下 Zielonka 异步自动机的分布式构造与综合。首先，我们为进程架构为树状的异步自动机展示了一种简单的二次分布式构造。如果架构存在一个底层生成树，且通信仅限于树上的局部通信，则称该架构为树状。该二次分布结果推广了已知的树架构构造，并改进了针对三角化依赖字母表的较早指数级构造。最后，我们研究分布式控制器综合问题，并证明其对于树状架构是可判定的。这一结果将可判定性边界从树架构扩展到了树状架构，同时保持相同的Tower$_d(n)$复杂度界，其中 n 是系统的大小， $d \geq 0$ 是进程树的深度。

关键词： 分布式综合，Zielonka 自动机，语言分布

1 引言

我们考虑一种具有固定通信结构的分布式系统规范形式——Zielonka 的 异步自动机。这是一种广为人知的模型，支持在固定通信拓扑下进行分布式综合，其理论基础是 Mazurkiewicz 迹 [20]。在 Zielonka 自动机中，进程连接到特定的信道集合，每个进程都有自己的信道子集

⋆本研究得到瑞典研究理事会（VR）项目（编号：2020-04963）和国家科学中心（NCN）资助项目 2021/41/B/ST6/00535 的支持。⋆⋆ 本研究得到法国国家研究署 ANR-23-CE48-0005 资助项目 PaVeDyS 的支持。⋆⋆⋆ 本研究得到瑞典研究理事会（VR）项目（编号：2020-04963）以及由克努特与爱丽丝·瓦伦堡基金会资助的瓦伦堡人工智能、自主系统与软件计划（WASP）的支持。

† 本研究的部分工作是在访问西蒙斯计算理论研究所期间完成的。

。进程通过在它们所连接的信道上进行同步来通信。在通信过程中，所有参与的进程共享其局部状态，然后根据这些彼此共享的信息转移到新状态。此外，该模型采用会合式通信：只有当所有参与者（连接到该信道的进程）都同意时，通信才会发生。当两个信道没有共同进程时，它们上的通信可以并行执行，因此该模型是异步的。Zielonka 关于正则语言分布的开创性结果凸显了这一模型的重要性。给定一个通信架构——即哪个进程连接到哪些信道——以及一种符合该架构异步性的正则语言，总是可以将该语言分布到一个异步自动机中 [26]。

Zielonka 的结果是分布式综合中一个突出而罕见的例子，但计算代价高昂。从一个由具有 n 个状态的自动机接受的正则语言，以及一个包含 p 个进程的拓扑出发，可以构造一个大小为 $O(4^{p^4}n^{p^2})$ 的等价异步自动机 [11]。请注意，指数增长仅由进程数量（它是输入的一部分）导致；然而，正如 [11] 中的下界所示，这种增长确实不可避免。这一构造相当复杂，即便历经近 40 年，也仍未被广泛或轻易地理解。它促使人们开展了大量研究，探讨如何理解、解释并改进这一构造（例如 [21,13,11,2]）。最近，有人提出了一种构造异步自动机的替代方法，使用命题动态逻辑的偏序变体 [1]。

一般 Zielonka 结果的复杂性促使研究人员转而研究分布可能更容易实现的简单情形。一个值得注意的例子是 Krishna 和 Muscholl [16] 的构造，他们表明，当通信架构被限制为一棵树时——每个通道至多由两个进程监听，且不存在环——每个进程所需的状态数量至多是原顺序自动机状态数量的平方。Muscholl 和 Diekert [6] 则针对三角化依赖字母表给出了一个更简单的构造，不过其规模关于顺序自动机的状态数量呈指数增长。

异步自动机模型还因其在控制领域中的应用而引人注目，这类似于 Ramadge 和 Wonham 对离散事件系统的监督控制 [24,18]。对于通过共享变量进行同步通信的系统，早期研究已经确立，分布式控制是不可判定的 [23],，只有在非常受限的通信架构下才是可判定的，而且复杂度极高 [17,8]。相比之下，对于基于使用 Zielonka 自动机的异步通信的分布式控制，研究人员发现了更复杂的架构，在这些架构下问题仍然是可判定的。值得注意的是，研究表明，当通信被限制在一棵树上时，ω-正则获胜条件下的控制是可判定的 [22]（关于局部可达性条件，另见 [12]）。他们的构造在树的深度 d 上具有 Tower(d) 复杂度，但适用于一种在同步共享变量世界中不可判定的架构。类似地，对于一致且连接良好的架构，该问题再次变得可判定，但这次需要指数空间 [9]。一般而言，基于 Zielonka 自动机的分布式控制，也称为具有

因果记忆的异步博弈，相当于所谓的佩特里博弈 [3]。然而，尽管早先人们曾希望这些一般架构具有可判定的控制问题，但最近已经证明，一般的分布式综合是不可判定的 [14]。

最近，Hausmann 等人研究了可重配置异步自动机模型中的分布 [15]。他们的模型通过允许运行时更改进程进行通信的信道集合，推广了异步自动机。他们证明，Krishna 和 Muscholl [16] 的简单分布式构造同样适用于一种更一般的架构，他们称之为 树状架构。也就是说，只要通信具有潜在的生成树，并且通信在树上是局部的，就允许进程更加自由地进行通信。本文在异步自动机的背景下介绍这一构造，并去除与重配置相关的复杂记法。尽管为了遵循树的结构，他们的构造在进程数量上呈指数级增长，但当通信架构固定时，便不再需要这样做，此时就会得到 Krishna 和 Muscholl 的二次构造。此外，我们还表明，该构造也改进了 Diekert 和 Muscholl 此前针对三角化依赖图的一个已知指数级构造 [6],，因为事实证明，三角化依赖图等价于树状架构。最后，我们重新考察了树形架构上的分布式综合结果 [22] ，并表明这一构造同样可以推广到树状架构的情形。

为方便起见，本手稿电子版中的技术术语和记号均通过超链接指向其定义（参见https://ctan.org/pkg/knowledge）。

2 预备知识

2.1 确定有限自动机

在字母表 Σ 上的确定有限自动机（DFA）记作 $\mathcal{A} = (\Sigma, S, \Delta, s_0, F)$，其中 S 是有限状态集合，$\Delta\colon\ S \times \Sigma \to S$ 是部分转移函数，$s_0 \in S$ 是初始状态，$F \subseteq S$ 是接受状态集合。给定一个词 $w = a_0 \cdots a_{n-1}$，A 在 w 上的初始运行是 $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \ldots \xrightarrow{a_{n-1}} s_n$ 的一条路径，即对于每个 $0 \leq i < n$，都有 $s_{i+1} = \Delta(s_i, a_i)$。如果 $s_n \in F$，则初始运行是接受的，此时 w 被 $\mathcal{A}$ 接受。$\mathcal{A}$ 的语言记作 $\mathcal{L}(\mathcal{A})$，是被 $\mathcal{A}$ 接受的词的集合。通常我们会滥用记号，用 $\Delta(s, u)$ 表示从状态 s' 在词 u 上到达的状态 s （或简称 $s \xrightarrow{u} s'$）。

独立关系是一种对称且自反的关系 $I \subseteq \Sigma \times \Sigma$。如果可以从 u 开始，反复交换两个相邻的独立字母，最终得到 v，则称两个词 u、$v \in \Sigma^*$ 是 I-不可区分的，记作 $u \sim_I v$。也就是说，$\sim_I$ 是关系 $\{(uabv, ubav) \mid (a, b) \in I, u, v \in \Sigma^*\}$ 的传递闭包。我们用 $[u]_I$ 表示词 $u \in \Sigma^*$ 的 I-等价类。这也称为 Mazurkiewicz迹 [20]。设 $\mathcal{A} = (\Sigma, S, \Delta, s_0, F)$ 是一个定义在 Σ 上的确定性自动机。若对于所有独立字母对

$(a, b) \in I$ 和所有状态 $s \in S$，都有 $\Delta(s, ab) = \Delta(s, ba)$。如果 $\mathcal{A}$ 具有这一性质，那么词 u 被 $\mathcal{A}$ 接受，当且仅当 $[u]_I$ 中的所有词都被接受。

2.2 异步自动机

通信架构 $\mathbb{C}$: $\Sigma \to 2^{\mathbb{P}}$ 为每个字母关联读取该字母的进程子集。我们令 $\mathbb{C}^{-1}(p) = \{a \in \Sigma \mid p \in \mathbb{C}(a)\}$。分布式字母表是 $(\Sigma, \mathbb{C})$，其中 $\mathbb{C}$ 是一种通信架构。它通过以下方式定义独立关系 $I(\mathbb{C}) \subseteq \Sigma \times \Sigma$：当且仅当 $\mathbb{C}(a) \cap \mathbb{C}(b) = \emptyset$时，$a(b) \in I(\mathbb{C})$。独立关系的补集是依赖关系 $D(\mathbb{C}) = \Sigma \times \Sigma \setminus I(\mathbb{C})$。容易看出，$(a, b) \in D(\mathbb{C})$ 当且仅当 $\mathbb{C}(a) \cap \mathbb{C}(b) \neq \emptyset$。依赖关系$D$ 定义一个图 $G_D = (\Sigma, D)$，其中 Σ 是节点集合，D 是边集合。若对于每个 $a \in \Sigma$，都有 $|\mathbb{C}(a)| \leq 2$，则称 $\mathbb{C}$ 为二元关系。二元关系 $\mathbb{C}$定义一棵树：若图 $(\mathbb{P}, E)$ 是一棵树，其中 $(p, q) \in E$ 当且仅当 $\mathbb{C}(a) = \{p, q\}$ 且对某个 $a \in \Sigma$有 $p \neq q$。对于任意进程 $p \in \mathbb{P}$，令 $\Sigma_p = \{a \in \Sigma \mid \{p\} = \mathbb{C}(a)\}$为 p局部动作的集合，也就是仅涉及 p的动作集合。

一种异步自动机（简称：异步自动机） [26]定义在分布式字母表$(\Sigma, \mathbb{C})$和进程 $\mathbb{P}$ 上，是一个元组 $\mathcal{B} = ((S_p)_{p\in\mathbb{P}}, (s_p^0)_{p\in\mathbb{P}}, (\delta_a)_{a\in\Sigma}, Acc)$，满足：– S_p 是进程 p的有限状态集，$s_p^0 \in S_p$ 是其初始状态；– $\delta_a : \prod_{p\in\mathbb{C}(a)} S_p \to \prod_{p\in\mathbb{C}(a)} S_p$ 是字母 a 的部分转移函数，它只取决于 $\mathbb{C}(a)$中的进程状态，并会改变这些状态；– $Acc \subseteq \prod_{p\in\mathbb{P}} S_p$ 是接受状态集。

$\mathcal{B}$ 的一个全局状态是$\mathbf{s} = (s_p)_{p\in\mathbb{P}}$，它给出每个进程的状态。对于全局状态$\mathbf{s}$ 和子集 $P \subseteq \mathbb{P}$，我们用$\mathbf{s} \downarrow_P = (s_p)_{p\in P}$ 表示$\mathbf{s}$ 中由 P中的进程状态组成的部分。

$\mathcal{B}$ 在词 $a_1 a_2 \dots a_n$ 上的初始运行，是乘积（全局）自动机中从初始状态 $\mathbf{s}_0 = (s_p^0)_{p\in\mathbb{P}}$出发的一条路径 $\mathbf{s}_0 \xrightarrow{a_1} \mathbf{s}_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} \mathbf{s}_n$：对于所有$0 < i \leq n$，$\mathbf{s}_i \in \prod_{p\in\mathbb{P}} S_p$，$a_i \in \Sigma$，满足 $\mathbf{s}_i \downarrow_{\mathbb{C}(a_i)} = \delta_{a_i}(\mathbf{s}_{i-1} \downarrow_{\mathbb{C}(a_i)})$ 且 $\mathbf{s}_i \downarrow_{\mathbb{P}\setminus\mathbb{C}(a_i)} = \mathbf{s}_{i-1} \downarrow_{\mathbb{P}\setminus\mathbb{C}(a_i)}$。如果 $\mathbf{s}_n$ 属于 Acc，则该初始运行是接受运行。如果存在这样的接受运行，则词 $a_1 a_2 \dots a_n$ 被 $\mathcal{B}$接受（注意，尽管自动机是确定性的，但在某些词上可能不存在运行）。 $\mathcal{B}$的语言记作 $\mathcal{L}(\mathcal{B})$，是被 $\mathcal{B}$接受的词的集合。令 $\mathrm{Runs}(\mathcal{B})$ 表示 $\mathcal{B}$的运行集合，令 $\mathrm{Runs}_p(\mathcal{B})$ 表示它们在状态的 p分量上的投影。

我们称语言 $\mathcal{L} \subseteq \Sigma^*$ 在（Σ，$\mathbb{C}$）上是分布式识别的，如果存在一个异步自动机 $\mathcal{B}$，使得 $\mathcal{L}$（$\mathcal{B}$） $= \mathcal{L}$。

定理 1（[26,11,16]）。给定一个 I（$\mathbb{C}$）-菱形确定性自动机 $\mathcal{A}$，存在一个异步自动机 $\mathcal{B}$，它分布式识别 $\mathcal{A}$的语言。一般而言，如果 $\mathcal{A}$ 有 n 个状态，那么 $\mathcal{B}$ 的每个进程都有 O（$4^{|\mathbb{P}|^4} n^{|\mathbb{P}|^2}$）个状态。如果 $\mathbb{C}$ 诱导出一棵树，那么 $\mathcal{B}$ 的每个进程都有 O（n^2）个状态。

一个异步自动机 $\mathcal{B}$ （记作 $|\mathcal{B}|$）的大小定义为 $\max_{p\in\mathbb{P}} |S_p|$。这种标准的大小概念考虑了进程的极大局部存储器（状态空间），因为异步自动机是一个分布式设备。

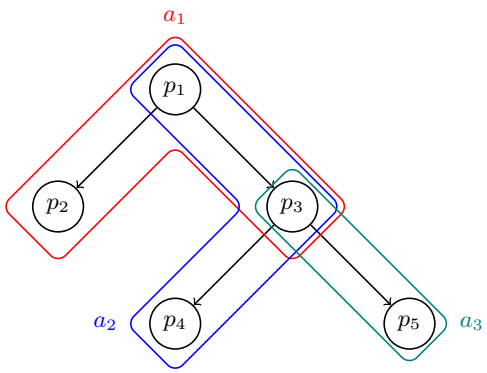


图 1. 一种树状通信架构：黑色边构成树，而通信架构则用不同颜色组表示每个字母。

2.3 树状通信架构

和之前一样，设 $\mathbb{P}$ 是一个有限进程集合，Σ 是一个有限字母表。如果存在一棵树能够覆盖使两个以上进程同步的字母，那么该通信架构就是树状的。形式化地说，有如下结论。

设 $|\mathbb{P}| = n$。$\mathbb{P}$ 上的一棵树是一个 $T = (\mathbb{P}, r, E)$，其中 $\mathbb{P}$ 是节点集合，$r \in \mathbb{P}$ 是根节点，而 $E \subseteq \mathbb{P}^2$ 是边集合，且该集合无环并且连通。

定义 1. 称 $\mathfrak{A} = (\mathbb{C}, T)$，其中 $\mathbb{C}$ 是一个通信架构，且 T 是一棵树，称其为树状通信架构*(*简称：*TCA)*，当且仅当

1. 对所有字母 $a \in \Sigma$，集合 $\mathbb{C}(a)$ 在 T 中连通；也就是说，若 $p, q \in \mathbb{C}(a)$，则从 p 到 q 的路径上所有进程也都属于 $\mathbb{C}(a)$；*2.* 若 $(p, q) \in E$，则存在一个字母 $a \in \Sigma$，使得 $p, q \in \mathbb{C}(a)$。

条件 1 确保给定字母上的通信始终在树中是“局部的”。条件 2 确保这棵树不由互不连通的部分组成。如果条件 2 不成立，我们称 $\mathfrak{A}$ 是森林状的。

例 *1.* 固定进程集合 $\mathbb{P} = \{p_1, \ldots, p_5\}$ 和字母集合 $\Sigma = \{a_1, a_2, a_3\}$。则 $\mathfrak{A} = (($ $\mathbb{C}(a_1)$ $\mathbb{C}(a_2)$ $\mathbb{C}(a_3))$ $T)$，如图 1 所示，是一个以 p_1 为根节点的树状通信架构：

1. 每个字母都在 T 中连通。2. T 中的所有边都至少由一个字母覆盖。

父节点、子节点、后代节点、叶节点、邻居节点和子树的概念均按通常方式定义。

3 树状架构的异步自动机

Hausmann 等人 [15] 证明，当可重配置异步自动机通过树状架构进行通信时，可以对其进行分布。可重配置

异步自动机会在运行过程中改变其通信架构。因此，它们比异步自动机更为通用。在此，我们介绍其在固定树状架构下的构造。等价地，我们证明，Krishna 和 Muscholl 针对树架构的构造 [16] 经过扩展并采用更复杂的转移后，也适用于树状架构。

我们将独立性的定义扩展到序列和字母集合。首先，若对每个$a_1 \in C_1$和 $a_2 \in C_2$，都有$(a_1, a_2) \in I(\mathbb{C})$，则称 C_1与 $C_2 \subseteq \Sigma$独立（记作$(C_1, C_2) \in I(\mathbb{C})$）。等价地，$C_1 \times C_2 \subseteq I(\mathbb{C})$。若存在字母集合$C_1$、$C_2 \subseteq \Sigma$，使得$w_1 \in C_1^*$、$w_2 \in C_2^*$，且$(C_1, C_2) \in I(\mathbb{C})$，则称 w_1与 $w_2 \in \Sigma^*$ 独立（记作$(w_1, w_2) \in I(\mathbb{C})$）。

固定一个确定有限自动机 $\mathcal{A} = (\Sigma, S, \Delta, s_0, F)$和 $\mathfrak{A} = (\mathbb{C}, T)$，使得 $\mathcal{A}$ 是 $I(\mathbb{C})$-菱形。若 w_1 和 w_2 是两个 $I(\mathbb{C})$-独立词，则对于任何由 w_1 和 w_2交错而成的 w，都有 $\Delta(s, w_1 w_2) = \Delta(s, w_2 w_1) = \Delta(s, w)$。对此可对 w的长度进行归纳证明。

我们现在研究需要哪些信息才能计算 $\Delta(s, w_1 w_2)$，而无需记住 w_1 和 w_2 本身。

固定 $\mathcal{A}$ 的一个状态 s，并设 w_1 和 w_2 是两个 $I(\mathbb{C})$-独立词。令$s_1 = \Delta(s, w_1)$、$s_2 = \Delta(s, w_2)$，并令 C_1、C_2 为满足 $w_1 \in C_1^*$、$w_2 \in C_2^*$ 和 $(C_1, C_2) \in I(\mathbb{C})$ 的集合。下一个引理表明，只需知道 s、s_1、s_2 和 C_2（而不必知道确切的词 w_1、w_2），就能计算状态 $\Delta(s, w_1 w_2)$。

引理 1（[5]）。设 $\mathcal{A}$ 是一个 $I(\mathbb{C})$-菱形确定有限自动机。存在一个（部分定义的）函数 $Diam : S^3 \times 2^\Sigma \to S$，使得若 s、s_1、w_1、s_2、w_2、C_2 均如上所述有定义，则 $Diam(s, s_1, s_2, C_2) = \Delta(s, w_1 w_2)$。

证明。令 C_1' 为满足 $(C_1', C_2) \in I(\mathbb{C})$ 的极大字母集合。由此可得 $C_1 \subseteq C_1'$。任取词 w_1' 和 w_2'，满足 $w_1' \in C_1'^*$、$w_2' \in C_2^*$，且对于 $i \in \{1, 2\}$，有 $s_i = \Delta(s, w_i) = \Delta(s, w_i')$。下面证明 $\Delta(s, w_1' w_2') = \Delta(s, w_1 w_2)$。

首先，根据菱形性质和假设，我们有 $s' := \Delta(s, w_1 w_2) = \Delta(s, w_2 w_1) = \Delta(s_2, w_1)$。我们还知道 $\Delta(s, w_2') = s_2$，因此 $s' = \Delta(s, w_2' w_1)$。接着，再次利用菱形性质：$s' = \Delta(s, w_2' w_1) = \Delta(s, w_1 w_2')$。最后，再次利用假设得出：$s' = \Delta(s, w_1 w_2') = \Delta(s, w_1' w_2')$。

最后，我们可以将 $\mathrm{Diam}(s, s_1, s_2, C_2)$ 设为状态 $\Delta(s, w_1' w_2')$，其中 $w_1' \in C_1'^*, w_2' \in C_2^*$。□

在 [16], 中，Diam函数已用于TCA $\mathfrak{A} = (\mathbb{C}, T)$ 和二元 $\mathbb{C}$及 $I(\mathbb{C})$-菱形确定有限自动机 $\mathcal{A}$，方式如下。对于父节点与其一个子节点之间的一次通信，设 s 为父节点和子节点都知晓的 $\mathcal{A}$ 最近状态，s_1 并设 s_2 分别为父节点和子节点知晓的最近状态，且 C_2 为定义域在以该子节点为根的子树中的字母集合。然后，可以通过计算 $\mathrm{Diam}(s, s_1, s_2, C_2)$，合并通信后父节点和子节点掌握的信息。由于下文将展示非二元 $\mathbb{C}$的一般情形，因此不再提供更多细节。

在我们的树状设定中，通信可以涉及两个以上的参与者。为此，我们自然地将 Diam 函数扩展为作用于子树。设 T 是一棵树，每个节点都标记有一对状态（s，t）和一个字母集合 $C^{\downarrow}$；直观来说，s 是该节点处的进程与其父节点都知晓的最近状态，t 是该进程知晓的最近状态，而 $C^{\downarrow}$ 是该节点为根的子树中可找到、且独立于与父节点共享的字母集合。设 r是 T的根，标记为（s_r，t_r，$C_r^{\downarrow}$），并假设 r有 k个子节点，分别标记为（s_i，t_i，$C_i^{\downarrow}$），且对应于树 T_i （对于 $i \in \{1,\dots,k\}$）。然后，我们按如下方式递归定义函数 TDiam：

$$\mathrm{TDiam}(T) = \begin{cases} t_r & \text{if } k = 0, \\ \mathrm{Diam}(s_k, \mathrm{TDiam}(T \setminus T_k), \mathrm{TDiam}(T_k), C_k^{\downarrow}) & \text{otherwise.} \end{cases}$$

注意，状态p函数从 [16] 等价于$\mathrm{TDiam}(T_p)$ 其中 T_p是以 p为根的树。

考虑一个字母 $a \in \Sigma$，其中 $\mathbb{C}(a) = \{p_1,\dots,p_k\}$。我们将 a 的树记作 T_a，定义为 T 的子树，其节点集合恰好为 $\mathbb{C}(a)$；根据 TCA 的定义，这确实是一棵树。现在考虑第二个字母 $b \neq a$ 以及某个 p，使得 $p \in \mathbb{C}(a)$，但 $p \notin \mathbb{C}(b)$。同样根据 TCA 的定义，$\mathbb{C}(b)$ 占据了 T 中的一个连通区域，且该区域不包含 p。因此，它必定位于 p“上方”或“下方”：前者意味着从 p 到该区域的唯一路径经过 p 的父节点；后者意味着该路径经过 p 的某个子节点。我们将满足第二种情况的字母集合定义为 $C_p^{\downarrow}$。现在假设每个 $p_i \in \mathbb{C}(a)$ 都关联着 $\mathcal{A}$ 的一对状态（s_i，t_i）。然后，我们用（s_i，t_i，$C_{p_i}^{\downarrow}$）标记 T_a 的每个节点 p_i，并定义状态 s_a =TDiam(T_a)。直观而言，这意味着我们结合 T_a 中所有节点的信息，来计算最新的状态 s_a。

现在可以陈述我们的构造了。我们通过定义 $\mathcal{B} = ((S_p)_{p\in\mathbb{P}}, (s_p^0)_{p\in\mathbb{P}}, (\delta_a)_{a\in\Sigma},$ 接受集合$)$，来分配 $\mathcal{A} = (\Sigma, S, \Delta, s_0, F)$。具体而言，对于每个 $p \in \mathbb{P}$，我们定义集合 S_p及其初始状态 s_p^0，为每个 $a \in \Sigma$定义转移函数 δ_a，并定义接受集合 接受集合。对于每个 $p \in \mathbb{P}$，我们设定 $S_p = S \times S$和 $s_p^0 = (s_0, s_0)$。

对于 $a \in \Sigma$，我们现在描述转移 $\delta_a : \prod_{p\in\mathbb{C}(a)} S_p \to \prod_{p\in\mathbb{C}(a)} S_p$。该转移按照正确的顺序，对所有其他进程所持有的状态计算菱形闭包。基于这一计算，每个进程都会更新其状态。

令 $\mathbb{C}(a) = \{p_1,\dots,p_k\}$，并对所有 $1 \le i \le k$，令（s_i，t_i）为 p_i的状态。令 $s_a = \mathrm{TDiam}(T_a)$ 如前所述，并令 $s' = \Delta(s_a, a)$。我们设定

$$\delta_a((s_1,t_1),\dots,(s_k,t_k)) = ((u_1,v_1),\dots,(u_k,v_k))$$

其中，进程 p_i 的新状态是 $u_i = \textsc{is-root}_a^{p_i} ? s_i : s'$，并且 $v_i = s'$ 与

$$\textsc{is-root}_a^{p} ? s : t = \begin{cases} s & \text{if } p \text{ is the root of } T_a, \\ t & \text{otherwise.} \end{cases}$$

如果 $\Delta(s_a, a)$ 未定义，那么 δ_a 也未定义。

现在我们定义 Acc。考虑一个全局状态 $\mathsf{s} = ((s_1, t_1) \dots (s_n, t_n))$。回顾一下，TCA 为 $\mathfrak{A} = (\mathbb{C}, T)$。对于树 T，考虑将每个节点 i 在 T 中标记为一对 (s_i, t_i)，并考虑在先前定义的意义下，节点 i“下方”出现的字母集合 $C_i^{\downarrow}$。当且仅当 $\mathrm{TDiam}(T) \in F$ 时，我们定义 $\mathsf{s} \in \mathrm{Acc}$。

下面给出我们的主要结果。

定理 2. 若 $\mathcal{A}$ 是 $\mathbb{C}$-菱形 且 $\mathbb{C}$ 是 树状，则 $\mathcal{B}$ 以分布式方式识别 $\mathcal{L}(\mathcal{A})$。 $\mathcal{B}$ 的大小为 $O(n^2)$，其中 n 是 $\mathcal{A}$的状态数。

3.1 正确性证明

本节专门用于证明定理 2。

首先，完全独立于构造过程，我们需要定义一个（或一组）进程的视图，正如 [16]，中所述，然后证明一个关于 TDiam 与这些视图之间关系的有用性质。设 $\mathcal{A}$ 是一个 $\mathbb{C}$-菱形确定有限自动机，$w \in \Sigma^*$ 是一个词，$\rho = s_0 \xrightarrow{a_1} \dots \xrightarrow{a_k} s_k$ 是 $\mathcal{A}$ 在 $w = a_1 \dots a_k$ 上的初始运行，而 $X \subseteq \mathbb{P}$ 是一组进程。由于 X 中的进程不一定参与 w 中的每次通信，因此 ρ 的某些部分可能在 X 不知情的情况下发生，而仅凭自身，X 中的进程可能无法知道在 ρ 结束时到达的状态。我们递归地定义 X 在 w 上的视图；直观而言，它对应于 w 的子词，X 中的进程可以通过共享动作看到它，并记作 $\mathrm{view}_X(w)$：

$$\mathrm{view}_X(\varepsilon) = \varepsilon$$

$$\mathrm{view}_X(w \cdot a_k) = \begin{cases} \mathrm{view}_{X \cup \mathbb{C}(a_k)}(w) \cdot a_k & \text{if } X \cap \mathbb{C}(a_k) \neq \emptyset, \\ \mathrm{view}_X(w) & \text{otherwise.} \end{cases}$$

我们计算 X 中进程的因果过去：首先，从至少有一个 X 中进程参与的最后一次通信开始。然后，视图 $_X(w)$ 通过共享动作，向后计算 X 中进程所看到的 w 子序列。滥用一下记号，我们将任意树 $T = (\mathbb{P}, r, E)$ 与其进程集合 $\mathbb{P}$ 等同，并用 $\mathrm{view}_T(w)$ 代替 $\mathrm{view}_{\mathbb{P}}(w)$。我们还定义这些进程能够计算出的 $\mathcal{A}$ 的状态为 $\sigma_X(w) = \Delta(s_0, \mathrm{view}_X(w))$。此外，由于 $\mathcal{A}$ 是 $\mathbb{C}$-菱形的，我们可以推断，对于任意字母 a，$\Delta(s_k, a)$ 有定义，当且仅当 $\Delta(s_0, \mathrm{view}^{\mathbb{C}(a)}(w)a)$ 有定义。也就是说，带有字母 a 的通信能否发生，不能取决于那些正在监听 a 的进程视图之外发生的通信。

设 $p \in \mathbb{P}$ 是 T 中父节点为 q 的某个进程。p 在 w 上的共享父视图，记作 $\mathrm{view}_p^{\uparrow}(w)$，定义为 $\mathrm{view}^{\{p\}}(w')$，其中 w' 是 w 的最短前缀，包含所有满足 $\{p, q\} \subseteq \mathbb{C}(b)$ 的字母 b 的所有出现。换言之，这是从 p 的视角所见的动作序列，直到最后一次涉及 p 及其父节点的通信。这不一定与 $\mathrm{view}^{\{p\}}(w)$ 相同，因为 p 可能在 w' 结束后与其子节点通信。出于类似原因，$\mathrm{view}_p^{\uparrow}(w)$ 不一定与 $\mathrm{view}^{\{q\}}(w)$ 相同。如果 p 是 T 的根节点，则对所有 w，令 $\mathrm{view}_p^{\uparrow}(w) = \varepsilon$。最后，我们

令 $\sigma_p^{\uparrow}(w) = \Delta(s_0, \text{视图}_p^{\uparrow}(w))$ 为在 $\mathcal{A}$ 读取该序列后到达的状态。

该Diam 函数及其扩展TDiam与系统独立部分的视图配合良好。具体而言，我们证明：如果 T 中的每个节点都正确标记，那么 TDiam(T) 就能正确计算出 T中每个进程对应的视图。

引理 2。对于所有词 w 以及 T 的所有子树 T'，其中每个节点 p 的T'都标记为 $(\sigma_p^{\uparrow}(w)\ \sigma_{\{p\}}(w)\ C_p^{\downarrow})$，有TDiam$(T') = \sigma_{T'}(w)$。

证明。对 T'的结构进行归纳，方法与 [16]中相同。

如果 T'是一个由$(s_p, t_p, C_p^{\downarrow})$标记的单节点 p，那么根据假设，TDiam(T') $= t_p = \sigma_{\{p\}}(w)$。否则，假设 T'由一个根节点 p构成，该根节点标记为$(s_p, t_p, C_p^{\downarrow})$，并有 k个子节点 $p_1, \ldots, p_k$，它们分别标记为$(s_1, t_1, C_1^{\downarrow}) \ldots, (s_k, t_k, C_k^{\downarrow})$，并分别连接到子树 $T_1, \ldots, T_k$。假设该性质在这些子树上成立，也就是说，对于所有 $i \le k$，都有 TDiam(T_i) $= \sigma_{T_i}(w)$。于是，令 T_i'表示 T' 的子树，该子树由 p、其前 i 个子节点及其各自的后代构成，我们递归地证明

$$(P_i) \qquad \text{TDiam}(T_i') = \sigma_{T_i'}(w)$$

对于初始步骤 $i = 0$，根据假设，TDiam(T_0') $= t_p = \sigma_{\{p\}}(w)$。现在假设对于 $1 \le i \le k$，(P_{i-1}) 成立。根据构造

$$\text{TDiam}(T_i') = \text{Diam}(s_i, \text{TDiam}(T_{i-1}'), \text{TDiam}(T_i), C_i^{\downarrow}).$$

由 (P_{i-1}) 可知 TDiam(T_{i-1}') $= \sigma_{T_{i-1}'}(w)$，根据假设，我们有 $s_i = \sigma_{p_i}^{\uparrow}(w)$ 且 TDiam(T_i) $= \sigma_{T_i}(w)$。

令 $w_0 = \text{view}_{p_i}^{\uparrow}(w)$、$w_i = \text{view}^{T_i}(w)$ 和 $w_{i-1} = \text{view}^{T_{i-1}'}(w)$。根据定义，$w_0$ 是 w_i 和 w_{i-1}的前缀。因此，令 $w_i = w_0 \cdot w_i'$ 和 $w_{i-1} = w_0 \cdot w_{i-1}'$。由于 w_0 包含涉及 p_i 及其父节点 p的最后一次通信，因此很容易看出 w_i' 和 w_{i-1}' 相互独立；否则，w_0 就不会是它们的最长公共前缀。还要注意，view$^{T_i'}(w) = w_0 v$，其中 v 是 w_i'和 w_{i-1}'的一种交错。此外，w_i' 的所有动作都在 $C_{p_i}^{\downarrow}$ 中，否则它们会涉及 p。根据 TDiam 的定义，我们得到 TDiam(T_i') $=$Diam$(s_i, \sigma_{T_{i-1}'}(w)\ \sigma_{T_i}(w)\ C_i^{\downarrow})$，其中 $s_i = \sigma_{p_i}^{\uparrow}(w) = \Delta(s_0, w_0)$，$\sigma_{T_{i-1}'}(w) = \Delta(s_i, w_{i-1}')$，且 $\sigma_{T_i}(w) = \Delta(s_i, w_i')$。因此，由引理 1，TDiam($T_i'$) $= \sigma_{T_i'}(w)$。

我们已经证明，对于所有 i，(P_i) 都成立，因此对于 $i = k$ 也成立；又由于 $T' = T_k'$，TDiam(T') $= \sigma_{T'}(w)$。 □

注意，这条引理意味着，如果 T' 是完整树 T 且被正确标记，那么 TDiam(T) $= \sigma_{\mathbb{P}}(w) = \Delta(s_0, w)$。我们将利用这一性质证明构造的正确性。

我们来定义在一次运行过程中应始终满足的不变式。令 $w \in \Sigma^*$。第一个不变式指出，当且仅当 $\mathcal{B}$ 在 w 上存在一次运行 ρ' 时，$\mathcal{A}$ 在同一个词 w 上也存在一次运行ρ。

($\text{I}_{\text{DEF}}[w]$)：ρ 在 w 上的运行 $\Leftrightarrow$ 已定义；ρ' 在 w 上的运行已定义。

现在，如果两次运行都未定义，那么 w 显然会被 $\mathcal{A}$ 和 $\mathcal{B}$同时拒绝，无需再作证明。因此，对于其余不变式，我们假设 ρ和 ρ'都已定义，并分别以状态 s和 $\mathbf{s} = ((s_1, t_1) \dots, (s_n, t_n))$ 结束。接着，我们建立一些不变式，将 ρ 的实际状态与每个进程在其局部状态中保存的一对状态联系起来，这与上一节给出的直觉相对应。

$$(\mathtt{I}^{\uparrow}_S[w]): \quad \forall p \in \mathbb{P}.\, s_p = \sigma^{\uparrow}_p(w) \qquad (\mathtt{I}_S[w]): \quad \forall p \in \mathbb{P}.\, t_p = \sigma_{\{p\}}(w)$$

这两个不变式与引理 2结合使用，是证明$\mathcal{A}$ 和 $\mathcal{B}$ 的语言等价的关键。现在我们证明这些不变式归纳地成立。

不变式的初始化。 所有不变式在空字 $w = \varepsilon$ 上都成立。($\mathtt{I}_{\mathrm{DEF}}[\varepsilon]$) 是显然的。($\mathtt{I}^{\uparrow}_S[\varepsilon]$)、($\mathtt{I}_S[\varepsilon]$) 可由初始状态 $\mathbf{s}_0$ 的定义轻易推出。

转移后的不变式。 现在考虑形如 $w' = w \cdot a$ 的词，其中运行 ρ 和 ρ' 都定义到 w，并分别以状态 s 和 $\mathbf{s}$ 结束。假设所有不变式在 w 中成立。我们证明它们在 w' 中仍然成立。

对于 ($\mathtt{I}_{\mathrm{DEF}}[w']$)，利用 ($\mathtt{I}^{\uparrow}_S[w]$)、($\mathtt{I}_S[w]$) 和引理 2，我们有 $s' := \mathrm{TDiam}(T_a) = \sigma_{\mathbb{C}(a)}(w)$。因此，当且仅当 $\Delta(s', a)$ 有定义时，$\Delta(s, a)$ 才有定义。根据 $\mathcal{B}$ 的定义，从 $\mathbf{s}$ 存在一个 a 转移，当且仅当 $\Delta(s', a)$ 有定义，当且仅当 $\Delta(s, a)$ 有定义，当且仅当 $\mathcal{A}$ 中存在一个 a 转移。因此 ($\mathtt{I}_{\mathrm{DEF}}[w']$) 成立。

现在假设两个运行都对 w' 有定义，并令 $s \xrightarrow{a} s'$ 属于 $\mathcal{A}$，且 $\mathbf{s} \xrightarrow{a} \mathbf{s}'$ 属于 $\mathcal{B}$。令 $p \in \mathbb{P}$，其在 $\mathbf{s}$ 中的状态为 (s_p, t_p)，在 $\mathbf{s}'$ 中的状态为 (s'_p, t'_p)。如果 p 未参与 a 上的通信，那么其状态保持不变，根据定义，$\mathrm{view}^{\uparrow}_p(w') = \mathrm{view}^{\uparrow}_p(w)$ 且 $\mathrm{view}^{\{p\}}(w') = \mathrm{view}^{\{p\}}(w)$，因此 ($\mathtt{I}^{\uparrow}_S[w']$) 和 ($\mathtt{I}_S[w']$) 可分别由 ($\mathtt{I}^{\uparrow}_S[w]$) 和 ($\mathtt{I}_S[w]$) 直接得到。

现在假设 p 参与了这次通信。我们先证明 ($\mathtt{I}_S[w']$)。根据定义，$\mathrm{view}^{\{p\}}(w') = \mathrm{view}^{\mathbb{C}(a)}(w) \cdot a$。同样，结合 ($\mathtt{I}_S[w]$)、($\mathtt{I}^{\uparrow}_S[w]$) 和引理 2，可得 $\sigma_{\mathbb{C}(a)}(w) = \mathrm{TDiam}(T_a) = s'$。因此 $\sigma_{\{p\}}(w') = \Delta(s', a) = t'_p$，且 ($\mathtt{I}_S[w']$) 成立。对于 ($\mathtt{I}^{\uparrow}_S[w']$)，分两种情况。首先，假设 p 在 T 中的父节点未参与 a 上的通信（这可能是因为 p 是根节点，没有父节点；也可能是因为其父节点没有监听 a）。在这种情况下，根据定义，$\mathrm{view}^{\uparrow}_p(w') = \mathrm{view}^{\uparrow}_p(w)$，且根据 ($\mathtt{I}^{\uparrow}_S[w]$)，$\sigma^{\uparrow}_p(w) = s_p$。此外，$p$ 必定是 T_a 的根节点。因此 $s'_p = \text{IS-ROOT}^p_a ? s_p : \Delta(s_a, a) = s_p$，满足 ($\mathtt{I}^{\uparrow}_S[w']$)。其次，假设现在 p 的父节点（记为 q）参与了 a 上的通信。那么根据定义，$\mathrm{view}^{\uparrow}_p(w') = \mathrm{view}^{\{p\}}(w') = \mathrm{view}^{\mathbb{C}(a)}(w) \cdot a$。因此，按照与证明 ($\mathtt{I}_S[w']$) 相同的方法，$\sigma^{\uparrow}_p(w') = \Delta(s', a)$。于是，$s'_p = \text{IS-ROOT}^p_a ? s_p : \Delta(s', a) = \Delta(s', a)$，从而得到 ($\mathtt{I}^{\uparrow}_S[w']$)。

证明结论。我们已证明，对于任意词w，这些不变式都成立。我们用它们来证明 $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$。设 $w \in \Sigma^*$ 为一个词，那么 w 被 $\mathcal{A}$ 接受，当且仅当 $\mathcal{A}$ 在 w 上的一次运行最终到达状态 $s \in F$。根据 ($\mathtt{I}_{\mathrm{DEF}}[w]$)，$\mathcal{A}$ 在 w 上的运行有定义，当且仅当 $\mathcal{B}$ 在 w 上的运行有定义。如果两次运行都未定义，那么 w 既不属于 $\mathcal{L}(\mathcal{A})$，也不属于 $\mathcal{L}(\mathcal{B})$。否则，根据 ($\mathtt{I}^{\uparrow}_S[w]$)、($\mathtt{I}_S[w]$),

以及引理2，我们有 TDiam$(T) = $ 视图$_{\mathbb{P}}(w) = s$。于是 $\mathcal{B}$ 接受 w 当且仅当 TDiam$(T) = s \in F$ 当且仅当 $\mathcal{A}$ 接受 w。因此 $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$，由此得证定理2。

3.2 弦依赖字母表

Zielonka 的通用分布式构造之所以复杂，是因为需要存储关于其他进程所掌握信息的相关信息，以及一种时间戳机制，以便将这些信息片段组合起来 [26]。Diekert 和 Muscholl 已证明，如果 Σ 中字母之间的依赖关系是一个弦图，就可以避免这种复杂的信息结构 [6,7]。

我们注意到，树状架构为 Diekert 和 Muscholl 的构造提供了新的视角，并带来了一种异步自动机构造，其效率有指数级提升。

考虑一个图 $G = (V, E)$。顶点序列 $v_1, \ldots, v_n$ 是一个环，当且仅当对每个 i，都有 $(v_i, v_{i+1}) \in E$ 且 $(v_n, v_1) \in E$。如果对每个满足 $n > 3$ 的环 $v_1, \ldots, v_n$，都存在一对 $(v_i, v_j) \in E$，使得 $1 < |j - i| < n - 1$，那么图 $G = (V, E)$ 就是弦图。也就是说，该环必须含有一条弦。

考虑一个架构 $\mathbb{C}$，并令 $D = D(\mathbb{C})$。事实证明，当且仅当依赖图是树状的时，架构 $G = (\Sigma, D)$是弦图且连通。由此可知，如果依赖图是弦图但不连通，那么其架构就是森林状的。

引理 3 ([10])。架构 $\mathbb{C}$ 是树状的，当且仅当 $G_D = (\Sigma, D)$ 是弦图且连通。

用 Gavril 的术语来说，给定一棵树，树的子树所构成的每个图中，若边对应于子树之间的非空交集，则该图都是弦图。反过来，每个弦图都可以表示为某棵恰当的树的子树交集图。连通性部分很容易补充。

首先要注意，对于非连通依赖图，异步自动机的构造很容易归约为对于连通依赖图的构造。为此，假设 Σ 是两个两两独立的字母表Σ_1, Σ_2的不交并，因此对于任意 $a \in \Sigma_1, b \in \Sigma_2$，都有 $(a, b) \notin D$。考虑一个定义在 Σ 上的 I-菱形确定有限自动机 $\mathcal{A}$以及一个输入词 $w \in \Sigma^*$。我们可以利用菱形性质（引理 1），根据 $\mathcal{A}$ 分别在输入 w 对 Σ_1 和 Σ_2的投影 w_1 和 w_2 上到达的状态，推知 $\mathcal{A}$ 在 $w \sim_I w_1 w_2$上到达的状态。更正式地说，假设我们构造了一个定义在字母表 Σ_1 上的异步自动机 $\mathcal{B}_1$，以及一个定义在字母表 Σ_2上的异步自动机$\mathcal{B}_2$。对于每个 $\mathcal{B}_1, \mathcal{B}_2$，我们可以假设，在相应输入上到达的全局状态能够确定 $\mathcal{A}$ 在该词上到达的状态。通过并行组合 $\mathcal{B}_1$，$\mathcal{B}_2$，我们得到一个异步自动机 $\mathcal{B}$（因为它们没有共同的字母）。$\mathcal{B}$ 在 $w \sim_I w_1 w_2$ 上到达的全局状态能够确定 $\mathcal{A}$的两个状态$s_1 = \Delta(s_0, w_1)$ $s_2 = \Delta(s_0, w_2)$，而 Diam$(s_0, s_1, s_2, \Sigma_2)$ 则指出哪些全局状态是接受状态。

我们在定理2 中改进的结果将在下一个定理中陈述。其大小呈指数级，是因为该构造依赖于转移幺半群。

定理 3（[5,6]）。给定一个 I-菱形确定有限自动机 $\mathcal{A}$，其定义在一个三角化依赖图 $G = (\Sigma, D)$ 上，可以构造一个异步自动机 $\mathcal{S}$，使其识别相同的语言，且所需规模为 $|\mathcal{S}| = n!$，其中 $n = |\mathcal{A}|$。

4 分布式综合

现在，我们将二叉树架构上的异步自动机控制问题的可判定性结果 [22] 推广到树状架构。我们首先回顾相关定义，并将其调整为适合本文的记法。

4.1 控制器与控制问题

固定一个分布式字母表（Σ, $\mathbb{C}$）。与 [24]中的设定相同， Σ 被划分为系统（可控）动作 Σ_{sys} 和环境（不可控）动作 Σ_{env}。请记住， Σ_p 表示 p-局部动作，类似地， Σ_p^{env}也如此定义。与 [22] 中一样，我们要求所有通信动作（即至少由两个进程共享的动作）均不可控；换言之，所有可控动作都是局部动作：$\Sigma_{sys} \subseteq \cup_{p\in\mathbb{P}}\Sigma_p$。这并不构成限制，因为如 [22]中的命题 6 所示，可将通信动作也可控的一般设定归约到这一设定。此外，每个状态都应至少有一个可控（因而是局部）动作。当然，这也不是限制，因为必要时总可以添加局部的可控自环。这一假设会使构造稍微简单一些。

在控制问题中，通常使用局部接受条件，而不是全局接受条件，因为采用局部条件的系统更容易控制。也就是说，我们将*AA*定义中的异步自动机部分替换为一个集合 $\{Acc_p\}_{p\in\mathbb{P}}$。每个$Acc_p$ 的形式为（F_p, Ω_p），其中 $F_p \subseteq S_p$ 且 $\Omega_p : S_p \to \mathbb{N}$ 一个局部奇偶函数。其思想是，有限运行被 p 接受，当且仅当它终止于 F_p中的某个状态；无限运行被 p 接受，当且仅当它满足 p对应的（极大）奇偶条件。于是，运行被 $\mathcal{A}$ 接受，当且仅当它被所有进程 p接受。

我们称在 w 上的运行是极大的，如果无法将 w 分解为$w = u \cdot v$，且不存在 $a \in \Sigma$ 满足 $\mathbb{C}(a) \cap \mathbb{C}(v) = \emptyset$，使得在 $u \cdot a \cdot v$ 上的运行有定义。通俗地说，这意味着我们无法通过某个动作来延长 w 上的运行；该动作涉及的进程在 w中都只执行有限次动作。

现在我们定义控制器的概念。本节余下部分中，固定一个异步自动机 $\mathcal{A} = ((S_p)_{p\in\mathbb{P}}, (s_p^0)_{p\in\mathbb{P}}, (\delta_a)_{a\in\Sigma}, (Acc_p)_{p\in\mathbb{P}})$。为简单起见，我们不采用控制器的一般定义，而直接讨论针对 $\mathcal{A}$在 [22], 的覆盖控制器，并简称为控制器。文献 [22] 的引理 10 表明，覆盖控制器的控制问题等价于一般控制器的控制问题，因此我们只关注前者。

控制器的直观概念是：它是一台机器（由一个不带接受条件的异步自动机描述），引导系统选择哪些可控

动作，以确保只有被 $\mathcal{A}$接受的运行才会发生。另一方面，由于环境动作是不可控的，控制器不应能够限制它们的发生。为此，控制器可以通过添加额外信息来丰富 $\mathcal{A}$的状态，并利用这些信息选择从该状态出发可行的可控动作子集，只启用该子集中的动作。

形式上，控制器是 $\mathcal{A}$ 本身就是一个异步自动机 $\mathcal{C} = ((C_p)_{p\in\mathbb{P}}, (c_p^0)_{p\in\mathbb{P}}, (\Delta_a)_{a\in\Sigma})$，并配有一个投影 π，它将每个状态映射到 $c_p \in C_p$ 的 $\mathcal{C}$ 一个状态 $s_p \in S_p$，并且满足以下要求：

– 对所有满足 $\mathbb{C}(a) = \{p_1, \ldots, p_k\}$ 的 $a \in \Sigma$，以及 $\mathbf{c}, \mathbf{c}' \in C_{p_1} \times \cdots \times C_{p_k}$，若 $\Delta_a(\mathbf{c}) = \mathbf{c}'$，则有 $\delta_a(\pi(\mathbf{c})) = \pi(\mathbf{c}')$（其中 π 逐分量应用），– 对所有 $a \in \Sigma_{env}$，若 $\delta_a(\pi(\mathbf{c}))$ 有定义，则 $\Delta_a(\mathbf{c})$ 也有定义，– 对所有 $p \in \mathbb{P}$，$\pi(c_p^0) = s_p^0$。

得益于投影 π，控制器 $\mathcal{C}$ 继承了 $\mathcal{A}$的接受条件。若控制器 $\mathcal{C}$ 对于 $\mathcal{A}$ 是获胜的，$\mathcal{A}$ 的每个极大运行 $\mathcal{C}$ 都是接受的。

该控制问题针对给定的输入 $\mathcal{A}$，询问是否存在一个获胜控制器 $\mathcal{C}$，用于 $\mathcal{A}$（如果存在，则明确构造该控制器）。

例 *2*。考虑这样一种服务器-客户端设置：一个进程（服务器）与多个进程（客户端）通信。服务器向所有客户端广播，告知它希望解决一项任务。每个客户端独立开始处理所请求的任务，并在完成后向服务器报告。服务器收到两个子节点发回的答案后，会再次广播，通知所有客户端停止工作。

形式化地，令 $\mathbb{P} = \{p, c_1, \ldots, c_n\}$ 表示包含服务器 p 和客户端 $c_1, \ldots, c_n$ 的集合。我们有两个动作 t_1, t_2，分别表示对两种不同任务的请求，$\mathbb{C}(t_1) = \mathbb{C}(t_2) = \mathbb{P}$。每个进程 c_i 有两个局部动作 p_1^i, p_2^i 来推进这些任务，$\mathbb{C}(p_1^i) = \mathbb{C}(p_2^i) = \{c_i\}$。此外，还有一个共享动作 e_i，用于向服务器报告任务已结束，$\mathbb{C}(e_i) = \{c_i, p\}$。最后，还有一个动作 r，可以向所有客户端广播，在任务成功完成后重置它们的状态，$\mathbb{C}(r) = \mathbb{P}$。该架构呈树状，服务器 p 位于根节点，所有客户端 c_i 都是 p的子节点。图 2 展示了服务器和客户端的自动机。对于接受条件，我们要求服务器无限次访问其初始状态，并且每个客户端都无限次执行动作 e_i。为简化表述，不接受有限运行（客户端不会崩溃）。

现在来讨论控制问题：我们称所有局部动作（即动作 p_k^i）都是可控的，而根据定义，所有非局部动作都是不可控的。一个朴素的控制器只在每个进程上启用正确的进展动作，例如 p_1^i 在收到 t_1 广播后，但它并不是获胜控制器。事实上，由于环境决定哪些进程可以执行其 e_i，它可以为每个任务都选择相同的两个进程。这样一来，那些未被选中的进程永远轮不到执行e_i，因此不会处于接受状态。不过，我们仍然可以通过轮询的方式构造一个获胜控制器。为此，只需在每个 c_i的状态中加入一个计数器，记录从开始以来已请求的任务数对n取模后的值，因此我们有 $C_{c_i} = S_{c_i} \times \{0, \ldots, n-1\}$，而投影 π就是

到第一个分量上的投影。如果计数值为 $k-1$，那么进程 c_k 和 c_{k+1} 的控制器会被设定为只启用正确的进展动作，而其他进程则不启用任何动作。此后，环境别无选择，只能执行相应的 e_k 和 e_{k+1}，以使运行继续进行。接着，服务器发送重置广播，请求下一个任务，k 递增，每个进程依次执行自己的 e_i 动作，因此这个控制器是获胜控制器。

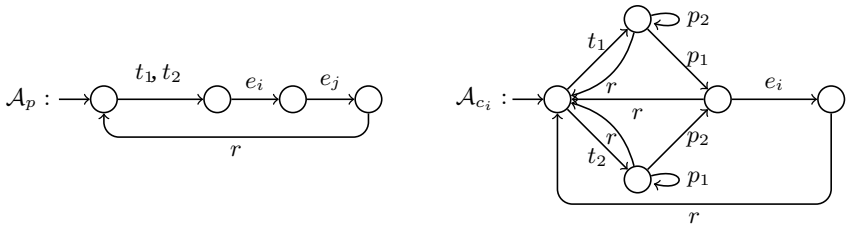


图 2. 用于分配和执行任务的服务器-客户端协议。

4.2 由父节点模拟叶进程

在 [22], 中，研究表明，在一个所有信道均为二元的树形架构中，叶进程可以由其父节点模拟，以便针对局部奇偶条件进行控制。具体做法是将叶进程的所有局部动作抽象为一个有限摘要；此后，叶进程与父节点之间的任何同步都可以由父节点在本地模拟。这种模拟使我们能够推导出控制问题的复杂度上界，该上界为关于树深度的指数塔。

我们在此证明，[22] 中的证明可以推广到树状架构。关键观察在于，TCA中，由于连续性条件，任何涉及叶节点与其他进程之间的通信必然也涉及该叶节点的父节点。因此，如果某个进程在进行某次转移时依赖于即将移除的叶节点的状态，那么移除之后它仍能获取该信息，因为模拟该叶节点的父节点会提供这一信息。

在本小节余下部分中，我们考虑一个异步自动机 $\mathcal{A}$ 和一个 TCA，其中 $\mathbb{P}$ 是进程集合，ℓ 是一个叶进程，而 p 是 ℓ 的父节点。第一步是确保进程之间能够“频繁”通信。这样，我们就能有限地描述叶进程 ℓ 在与 ℓ及其他进程共享的连续两个动作之间发生的情况。

定义 2。 一个异步自动机 $\mathcal{A}$ 是 ℓ-短的，如果存在某个界 $B \in \mathbb{N}$，使得在 ℓ 的每次局部 $\mathcal{A}$-运行中，连续执行 ℓ 个局部动作的数量最多为 B。

引理4（定理 10，[22]）。对于每个异步自动机 $\mathcal{A}$，我们都可以构造一个 ℓ-短自动机 $\mathcal{A}^{\circledS}$，使得 $\mathcal{A}$ 可控，当且仅当 $\mathcal{A}^{\circledS}$ 可控。任何 $q \neq \ell$ 的状态空间都不会从 $\mathcal{A}$ 变为 $\mathcal{A}^{\circledS}$；ℓ 在 $\mathcal{A}_\ell^{\circledS}$ 中的状态是 $\mathcal{A}_\ell$的简单局部路径。

尽管引理 4 的证明相当复杂，但它并不依赖通信架构是一棵树。其核心思想是，每个奇偶博弈都等价于一个有限环博弈。

因此，要证明叶进程可以由其父节点模拟（出于控制的目的），只需处理 $\mathcal{A}^{\circledS}$，特别是处理一个ℓ-短的 异步自动机。根据引理 4，我们知道界 B 可以设置为 ℓ-状态在 $\mathcal{A}$中的数量，因此 $B = |S_\ell|$ 即为 S_ℓ 在 ℓ 中的状态集合 $\mathcal{A}$。

现在我们可以证明，给定 $\mathcal{A}$，我们能够构造一个异步自动机 $\mathcal{A}^{\setminus \ell}$，其进程集合为 $\mathbb{P} \setminus \{\ell\}$，并且 $\mathcal{A}$ 是可控的当且仅当$\mathcal{A}^{\setminus \ell}$ 是可控的。此外，对于每个 $p' \neq p$，都有 $\mathcal{A}_{p'}^{\setminus \ell} = \mathcal{A}_{p'}$，且 $\mathcal{A}_p^{\setminus \ell}$的大小与 $|\mathcal{A}_p| \cdot |\Sigma_p^{sys}| \cdot exp(|\mathcal{A}_\ell|)$ 成正比。

令 $\mathcal{A} = ((S_p)_{p\in\mathbb{P}}, (s_p^0)_{p\in\mathbb{P}}, (\delta_a)_{a\in\Sigma}, (\mathrm{Acc}p)_{p\in\mathbb{P}})$，其中 $\ell \in \mathbb{P}$ 是一片叶子，$p \in \mathbb{P}$ 是它的父节点，并假设根据引理4，$\mathcal{A}$ 是 ℓ-短的。利用 $\mathbb{P}^{\setminus \ell} = \mathbb{P} \setminus \{\ell\}$，我们构造 $\mathcal{A}^{\setminus \ell} = ((S_p^{\setminus \ell})_{p\in\mathbb{P}^{\setminus \ell}}, (s_p^{0\setminus \ell})_{p\in\mathbb{P}^{\setminus \ell}}, (\delta_a^{\setminus \ell})_{a\in\Sigma^{\setminus \ell}}, (\mathrm{Acc}_p^{\setminus \ell})_{p\in\mathbb{P}^{\setminus \ell}})$。新的字母表 $\Sigma^{\setminus \ell}$ 将在稍后定义。对于所有进程 $q \neq p$，都有 $S_q^{\setminus \ell} = S_q$、$s_q^{0\setminus \ell} = s_q^0$，且 $\mathrm{Acc}_q^{\setminus \ell} = \mathrm{Acc}_q$。对于所有满足 $\mathbb{C}(a) \cap \{\ell, p\} = \emptyset$ 的动作 $a \in \Sigma$，都有 $\delta_a^{\setminus \ell} = \delta_a$。也就是说，所有与 p 或 ℓ 均无关的分量保持不变。

我们来定义 $S_p^{\setminus \ell}$。为此，我们首先定义 ℓ-局部策略。记住，$\mathcal{A}$ 是 ℓ-短的，因此令 B 为相应的界。还要记住，Σ_ℓ 是 ℓ-局部动作的集合。为便于表述，当 a_1、……、$a_n \in \Sigma_\ell$ 以及 s、s'都是 ℓ的状态时，我们用 $s \xrightarrow{a_1 \cdots a_n} s'$ 代指 $s' = \delta_{a_n}(\ldots(\delta_{a_1}(s)))$。若存在状态 s'使得 $s \xrightarrow{a_1 \cdots a_n} s'$，则记作 $s \xrightarrow{a_1 \cdots a_n}$。从给定状态 $s_\ell \in S_\ell$ 出发的 ℓ-局部策略，是一个偏函数 $f : (\Sigma_\ell)^{\leq B} \to \Sigma_\ell^{sys}$，满足：每当 $f(w) = a$ 时，$s_\ell \xrightarrow{w \cdot a}$ 在 $\mathcal{A}_\ell$中有定义。若 $s_\ell \xrightarrow{a} s'_\ell$，且 f是从 s_ℓ出发的 ℓ-局部策略，则 $f_{|a}$ 是从 s'_ℓ 出发、定义为 $f_{|a}(w) = f(a \cdot w)$ 的 ℓ-局部策略。定义了这一点后，父节点 p的 $S_p^{\setminus \ell}$中的一个状态属于以下 3 种类型之一：

$$(s_p, s_\ell), (s_p, s_\ell, f), (s_p, a, s_\ell, f)$$

其中 $s_p \in S_p$，$s_\ell \in S_\ell$ 分别是 p 和 ℓ 的模拟状态，f 是从 ℓ 出发的一个 - 局部策略，s_ℓ，而 $a \in \Sigma_p^{sys}$ 是父节点 p 的一个可控局部动作。由于 $\mathcal{A}$ 是 ℓ-短的，ℓ-局部策略可以简单地看作 $\mathcal{A}_\ell$中不重复的路径，因此总共只有$exp(|\mathcal{A}_\ell|)$个。最后，我们令 $s_p^{0\setminus \ell} = (s_p^0,\ s_\ell^0$ 为 p 的初始状态。

现在，我们先精确定义新字母表 $\Sigma^{\setminus \ell}$，再定义新的转移。此前所有 ℓ 局部的动作现在都改为不可控的 p 局部动作，无论它们在 Σ 中是否可控。同样，此前所有 p 局部的动作现在在 $\Sigma^{\setminus \ell}$ 中也不可控，但对于每个

p局部动作 $a \in \Sigma_p^{sys}$，若其可控，我们就添加一个新动作$ch(a)$ 于 $\Sigma^{\setminus \ell}$，该动作同样是 p局部且可控的。我们还为每个$ch(f)$ 添加新的动作，这些动作是p局部且可控的，对每个 ℓ局部策略 f均如此。任何包含 ℓ 的非局部动作，在 Σ 中仍然保留于 $\Sigma^{\setminus \ell}$，但其定义域中会移除 ℓ。在 Σ中其余所有动作，即局部于某个不属于 $\{p, \ell\}$ 的进程的动作，或与 ℓ不相交的非局部动作，均保持不变。

现在定义涉及 p 的动作上的转移。首先，对于形如（s_p， s_ℓ）的 p 状态，只有一种可用的转移类型

$$(s_p, s_\ell) \xrightarrow{ch(f)} (s_p, s_\ell, f) \text{ for all } \ell\text{-local strategies } f \text{ from } s_\ell$$

其中系统为 ℓ 固定一个局部策略。接下来，从这些状态出发，同样只有一种可能的转移类型

$$(s_p, s_\ell, f) \xrightarrow{ch(a)} (s_p, a, s_\ell, f) \text{ for all } a \in \Sigma_p^{sys} \text{ enabled from } s_p$$

其中系统选择一个 p 局部可控（在 Σ_p 中）动作[4]，，前提是存在一条从 s_p 出发、执行该动作的转移。随后这里会有多个选择。这两个动作

$$(s_p, a, s_\ell, f) \xrightarrow{b_p} (s'_p, s_\ell, f) \text{ if } (s_p, s'_p) \in \delta_{b_p} \text{ and } b_p = a \text{ or } b_p \in \Sigma_p^{env}$$
$$(s_p, a, s_\ell, f) \xrightarrow{b_\ell} (s_p, a, s'_\ell, f_{|b_\ell}) \text{ if } (s_\ell, s'_\ell) \in \delta_{b_\ell} \text{ and } b_\ell \text{ is enabled by } f$$

是 p 局部动作（均不可控），它们可以通过其预先选定的（先前的）可控动作 a 推进 p，通过任意局部不可控动作 b_p 推进 p，或通过预先选定策略 f 所启用的任意动作推进 ℓ，即任意不可控的 ℓ 局部动作，或由 f 在 ε 上输出的那个可控动作。随后还会出现非局部动作，其中 p 与其他进程通信。

$$((s_p, a, s_\ell, f), s_{q_1}, \ldots, s_{q_k}) \xrightarrow{b_{p+}} ((s'_p, s_\ell, f), s'_{q_1}, \ldots, s'_{q_k}) \text{ if}$$
$$p \in \mathbb{C}(b_{p+}), \ell \notin \mathbb{C}(b_{p+}), ((s_p, s_{q_1}, \ldots, s_{q_k}), (s'_p, s'_{q_1}, \ldots, s'_{q_k})) \in \delta_{b_{p+}}$$
$$((s_p, a, s_\ell, f), s_{q_1}, \ldots, s_{q_k}) \xrightarrow{b_{p\ell+}} ((s'_p, s'_\ell), s'_{q_1}, \ldots, s'_{q_k}) \text{ if}$$
$$\ell, p \in \mathbb{C}(b_{p\ell+}), ((s_p, s_\ell, s_{q_1}, \ldots, s_{q_k}), (s'_p, s'_\ell, s'_{q_1}, \ldots, s'_{q_k})) \in \delta_{b_{p\ell+}}$$

第一种适用于 ℓ 未参与通信的情况，因此，当 s_p 更新为新状态且 a的选择被重置时， s_ℓ 和 f 保持不变。第二种发生在 ℓ 参与通信时，此时会读取并更新 s_p 和 s_ℓ，并重置 a 和 f 的选择。请注意， $k = 0$ 涵盖了仅涉及 p 和 ℓ 的通信发生的情况。此外，由于我们假设架构是树状的，因此通信不可能在未包含 p 的情况下涉及 ℓ 和其他进程，这就是为何未涵盖这种情况。请注意，这是与原始构造的唯一重大区别

[4] 这里用到了这样一个假设：每个状态中至少有一个可控动作处于启用状态。

构造自 [22], ，这也是正确性证明大体相似的原因。

$\mathcal{A}^{\setminus\ell}$的最后一部分是接受条件 $\mathrm{Acc}_p^{\setminus\ell} = (F_p^{\setminus\ell}, \Omega_p^{\setminus\ell})$。对于 $F_p^{\setminus\ell}$而言，这很简单：任何形式为 $(s_p, s_\ell) \in F_p \times F_\ell$的状态，以及任何满足 $s_p \in F_p$ 的 (s_p, s_ℓ, f)、(s_p, a, s_ℓ, f)，其中从 s_ℓ 出发、遵循 f 的任何可能的极大延续最终都会到达 F_ℓ中的某个状态，都属于该集合。定义 $\Omega_p^{\setminus\ell}$则更加棘手。首先， p 的无限运行在限制到状态的s_p 分量时，必须满足 Ω_p 的奇偶条件。接下来分两种情况。如果在 s_ℓ 分量上的投影是无限的并且满足 Ω_ℓ ，则该运行必须被接受，这相当于两个奇偶条件的合取。可以将其转换为单个奇偶条件，我们将在第4.3节讨论其复杂度。否则，我们要求对于所到达的最后一个 f 和 s_ℓ （唯一确定为运行中被无限次看到的分量），从 s_ℓ 出发、遵循 f 的任何可能延续最终都会到达 F_ℓ中的某个状态。这相当于单个奇偶条件，即 p上的奇偶条件。

注意，我们没有定义在 ℓ 上的运行是无限的，而在 p 上的运行是有限时会发生什么，因为这与 ℓ-短假设不相容。

现在， $\mathcal{A}^{\setminus\ell}$ 已经定义，我们可以证明以下定理了（证明见附录）。

定理 4. 对于每个 ℓ-短 异步自动机 $\mathcal{A}$ ，若其具有局部接受条件，则存在一个 获胜控制器 适用于 $\mathcal{A}$ ，当且仅当存在一个获胜控制器适用于 $\mathcal{A}^{\setminus\ell}$。

证明。 $\boxed{\Rightarrow}$ 令 $\mathcal{C} = ((C_p)_{p\in\mathbb{P}}, (c_p^0)_{p\in\mathbb{P}}, (\Delta_a)_{a\in\Sigma})$ 为 $\mathcal{A}$ 的一个获胜控制器，其关联投影为 π。令 $c \in C_\ell$，我们从 $\pi(c)$ 构造一个 ℓ-局部策略，称为 f_c，如下所示。令 $c \xrightarrow{a_1} c_1 \xrightarrow{a_2} \dots \xrightarrow{a_k} c_k$ 为 $\mathcal{C}_\ell$ 中一个满足 $a_1, \dots, a_k \in \Sigma_\ell$ 的运行，并令 a 为从 c_k 启用的一个可控动作（如果有多个，则任取一个）。接着，我们定义 $f_c(a_1 \dots a_k) = a$。由于 $\mathcal{A}$ 是 ℓ-短的， f_c 是一个定义良好的 ℓ-局部策略。

我们为 $\mathcal{A}^{\setminus\ell}$ 构造控制器 $\mathcal{C}^{\setminus\ell} = ((C_p^{\setminus\ell})_{p\in\mathbb{P}^{\setminus\ell}}, (c_p^{0\setminus\ell})_{p\in\mathbb{P}^{\setminus\ell}}, (\Delta_a^{\setminus\ell})_{a\in\Sigma^{\setminus\ell}})$。与 p 无关的分量保持不变：对于所有 $q \neq p$， $C_q^{\setminus\ell} = C_q$, $c_q^{0\setminus\ell} = c_q^0$；对于所有满足 $p \notin \mathbb{C}(a)$ 的 $a \in \Sigma^{\setminus\ell}$ ， $\Delta_a^{\setminus\ell} = \Delta_a$ 。 $C_p^{\setminus\ell}$ 中的状态形式为 (c_p, c_ℓ) 、 (c_p, c_ℓ, f) 或 (c_p, a, c_ℓ, f)，其中 $c_p \in C_p$、 $c_\ell \in C_\ell$、 $a \in \Sigma_p$ 和 f 是来自 $\pi(c_\ell)$ 的 ℓ-局部策略。其初始状态为 $c_p^{0\setminus\ell} = (c_p^0, c_\ell^0)$。关联投影 $\pi^{\setminus\ell}$ 从 π 导出： $\pi^{\setminus\ell}(c_p, c_\ell) = (\pi(c_p)\ \pi(c_\ell))$，依此类推。

最后，我们描述与分量 p 相关的转移。从状态 (c_p, c_ℓ) 可执行的唯一可控动作是动作 $ch(f)$，其中 $f = f_{c_\ell}$ 是从 C_ℓ 中提取的 ℓ局部策略，如上文所定义，并转移到状态 (c_p, c_ℓ, f)。从 (c_p, c_ℓ, f) 可执行的唯一可控动作是动作 $ch(a)$，其中 $a \in \Sigma_p^{sys}$是 c_p 中启用的一个动作，并转移到状态 (c_p, a, c_ℓ, f)。如果存在多个这样的 a，则唯一地选取其中一个。如果不存在，则任意指定一个

选定的动作[5] $a_0 \in \Sigma_p$。所有其他动作均不可控，并遵循 $\mathcal{A}^{\setminus \ell}$ 的结构：

- $(c_p, a, c_\ell, f) \xrightarrow{b_p} (c'_p, c_\ell, f)$ if $c_p \xrightarrow{b_p} c'_p$ and either $b_p = a$ or $b_p \in \Sigma_p^{env}$ executes either the previously picked controllable a or any uncontrollable p-local action,
- $(c_p, a, c_\ell, f) \xrightarrow{b_\ell} (c_p, a, c'_\ell, f_{|b_\ell})$ if $c_\ell \xrightarrow{b_\ell} c'_\ell$ and either $b_\ell = f(\varepsilon)$ or $b_\ell \in \Sigma_\ell^{env}$ executes an enabled ℓ-local action,
- $((c_p, a, c_\ell, f), c_{q_1}, \ldots, c_{q_k}) \xrightarrow{b_{p+}} ((c'_p, c_\ell, f), c'_{q_1}, \ldots, c'_{q_k})$ if $\Delta_{b_{p+}}(c_p, c_{q_1}, \ldots, c_{q_k}) = (c'_p, c'_{q_1}, \ldots, c'_{q_k})$ is a communication between p and other processes not including ℓ,
- $((c_p, a, c_\ell, f), c_{q_1}, \ldots, c_{q_k}) \xrightarrow{b_{p\ell+}} ((c'_p, c'_\ell), c'_{q_1}, \ldots, c'_{q_k})$ if $\Delta_{b_{p\ell+}}(c_p, c_\ell, c_{q_1}, \ldots, c_{q_k}) = (c'_p, c'_\ell, c'_{q_1}, \ldots, c'_{q_k})$ is a communication between p and other processes including ℓ.

很容易看出，$\mathcal{C}^{\setminus \ell}$ 确实是 $\mathcal{A}^{\setminus \ell}$ 的控制器，并且 $\pi^{\setminus \ell}$ 满足所需条件，因为根据定义，$\mathcal{C}^{\setminus \ell}$ 模仿了 $\mathcal{A}^{\setminus \ell}$的结构。

最后，我们证明 $\mathcal{C}^{\setminus \ell}$ 对 $\mathcal{A}^{\setminus \ell}$是必胜的。设 $\rho^{\setminus \ell}$ 是 $\mathcal{C}^{\setminus \ell}$的一个极大运行。我们通过从 $\rho^{\setminus \ell}$ 中移除带有形如$ch(f)$和$ch(a)$的动作的转移、在状态中隐藏额外的分量 a 和 f，并将分量 c_p 和 c_ℓ解耦，由此构造出相应的 $\mathcal{C}$ 上的运行 ρ。可以通过归纳法证明，ρ 确实是 $\mathcal{C}$的一个运行。

当对 ℓ 的投影是有限的，并以 c_ℓ, f结束时，ρ 可能不是极大的。在这种情况下，我们根据 f为 ℓ取任意可能的局部延续。因此，我们可以假设运行 ρ 是极大的。

我们证明 $\rho^{\setminus \ell}$ 是接受的。我们知道 ρ是极大的，且 $\mathcal{C}$ 是获胜的，因此所有分量都是接受的。这直接意味着，对于所有 $q \neq p$，在分量 q 上的运行$\rho^{\setminus \ell}$ 是接受的，无论它是有限还是无限。只剩下分量p。如果在 p上的运行是有限的，它会以形如 (c_p, a, c_ℓ, f) 的状态结束。这意味着在 ρ中，分量 p 和 ℓ 分别以状态 c_p 和 c_ℓ结束。首先，我们注意到 $\pi(c_p) \in F_p$。由于 $\mathcal{C}_\ell$ 也是获胜的，因此按照 f，从 c_ℓ出发的 ℓ 的所有极大延续都会到达 $\pi^{-1}(F_\ell)$。因此，$\pi^{\setminus \ell}(c_p, a, c_\ell, f) \in F_p^{\setminus \ell}$。如果在$p$ 上的运行是无限的，那么要么有无限多个动作影响 ℓ，要么没有。对于前一种情况，在 ρ 中，p 和 ℓ 上的运行都是无限的，并分别属于 Ω_p和 Ω_ℓ。于是，根据定义，在 $\rho^{\setminus \ell}$ 中，p上的运行满足 $\Omega_p^{\setminus \ell}$。对于后一种情况，在 ρ 中，p 上的运行是无限的，且属于 Ω_p；但对于 ℓ，运行是有限的，并以某个状态 $c_\ell \in \pi^{-1}(F_\ell)$结束，且没有可能的延续，否则就会与极大性假设矛盾。于是 $\Omega_p^{\setminus \ell}$成立，本方向的证明至此结束。

[5] 这有助于简化转移的定义，避免区分情况。在这种情况下，唯一可能的后续操作要么是 p, ℓ 的不可控动作，要么是应用策略 f。

$\boxed{\Leftarrow}$令 $\mathcal{C}^{\backslash\ell} = ((C_p^{\backslash\ell})_{p\in\mathbb{P}\backslash\ell}, (c_p^{0\backslash\ell})_{p\in\mathbb{P}\backslash\ell}, (\Delta_a^{\backslash\ell})_{a\in\Sigma\backslash\ell})$ 为 $\mathcal{A}^{\backslash\ell}$ 的一个获胜控制器，其关联投影为 $\pi^{\backslash\ell}$。我们要构造一个用于 $\mathcal{A}$的获胜控制器 $\mathcal{C}$。

我们先来说明 $\mathcal{C}$ 的直观思路。可以假设 $\mathcal{C}^{\backslash\ell}$ 每次最多只启用一个可控动作（这种控制器限制更多，因此如果它之前能够获胜，那么现在仍然能够获胜）。由于投影形式为 $(s_p,\ s_\ell)$ 或 $(s_p,\ s_\ell,\ f)$ 的状态，其所有出边转移都带有可控动作，这意味着从这些状态出发，必定存在一条路径到达形式为 $(s_p,\ a,\ s_\ell,\ f)$ 的真实状态。我们将忽略前述中间状态，只关注真实状态。目标是将分量 $C_p^{\backslash\ell}$ 解耦为 C_p 和 C_ℓ。它们各自保存一份 $C_p^{\backslash\ell}$ 状态的副本，而 C_ℓ 还会额外保存自上次与 p 通信以来执行的 ℓ 局部动作序列。p 和 ℓ 将分别遵循 a 和 f 的选择，独立推进。当它们之间（也可能与其他进程之间）发生通信时，它们会结合 p 的状态和 ℓ 的动作序列，恢复在 $C_p^{\backslash\ell}$ 中本应到达的实际状态，然后相应地更新。这之所以可行，是因为树状结构使得 p 和 ℓ 的推进彼此独立，因此我们可以按顺序将它们组合起来，得到与任何其他顺序相同的结果。

形式化地，我们将分量 p 的真实状态定义为集合

$$TS = \{c_p \in C_p^{\backslash\ell} \mid \pi^{\backslash\ell}(c_p) \text{ is of the form } (s_p, a, s_\ell, f)\}$$

对于任意状态 $c_p \notin TS$，我们令 $ts(c_p)$ 表示从 c_p 经由一次 $ch(a)$ 转移，或两次 $ch(f)$、$ch(a)$ 转移可达的唯一真实状态。

我们为 $\mathcal{A}$ 构造控制器 $\mathcal{C} = ((C_p)_{p\in\mathbb{P}}, (c_p^0)_{p\in\mathbb{P}}, (\Delta_a)_{a\in\Sigma})$。与之前一样，所有与 p 和 ℓ 无关的分量都与 $\mathcal{C}^{\backslash\ell}$ 中的相同。p 的状态为 $C_p = \mathsf{TS}$，而 ℓ 的状态为 $C_\ell = \mathsf{TS} \times \Sigma_\ell^{\leq B}$。初始状态分别为 $\mathsf{ts}(c_p^{0\backslash\ell})$ 和 $(\mathsf{ts}(c_p^{0\backslash\ell})\ \varepsilon)$。投影 π 定义如下：令 $c \in \mathsf{TS}$ 且 $\pi^{\backslash\ell}(c) = (s_p, a, s_\ell, f)$，则对于 p，我们设 $\pi(c) = s_p$；对于 ℓ，我们设 $\pi((c, w)) = s'_\ell$，其中 $s_\ell \xrightarrow{w} s'_\ell$。涉及 p 或 ℓ 的转移共有四种不同类型。

– 对于 p-局部转移 $b \in \Sigma_p$，我们有 $\Delta_b(c_p) = \mathsf{ts}(\Delta_b^{\backslash\ell}(c_p))$，也就是说，我们按照 $\mathcal{C}_p^{\backslash\ell}$中的方式推进，但跳过非真实状态。– 对于 ℓ-局部转移 $b \in \Sigma_\ell$，如果 $c_p \xrightarrow{wb}$ 属于 $C_p^{\backslash\ell}$，则有 $\Delta_b((c_p, w)) = (c_p, wb)$，也就是说，我们只记录状态第二个分量中执行的动作。注意，由于将 $\mathcal{C}^{\backslash\ell}$ 的状态投影到 $\mathcal{A}^{\backslash\ell}$的状态上，并且由于 f的 B-短策略，因此 $|wb| \leq B$。– 设 b涉及 p和 $q_1, \ldots, q_k$，但不涉及 ℓ，并假设 $\Delta_b^{\backslash\ell}(c_p, c_{q_1}, \ldots, c_{q_k}) = (c'_p, c'_{q_1}, \ldots, c'_{q_k})$。于是我们定义 $\Delta_b(c_p, c_{q_1}, \ldots, c_{q_k}) = (\mathsf{ts}(c'_p)\ c'_{q_1}, \ldots, c'_{q_k})$，也就是说，与第一种情况相同。– 设 b 涉及 p、ℓ和 $q_1, \ldots, q_k$，并假设 $\Delta_b^{\backslash\ell}(c_p^3, c_{q_1}, \ldots, c_{q_k}) = (c'_p, c'_{q_1}, \ldots, c'_{q_k})$。还假设 $w \in \Sigma_\ell^*$ 是一个由局部 ℓ-动作组成的序列，且 $c_p^1 \xrightarrow{w} c_p^3$ 属于 $C_p^{\backslash\ell}$。

我们在 $\mathcal{C}$中定义 $\Delta_b((c_p^2,\ w)\ c_p^1,\ c_{q_1},\ \ldots,\ c_{q_k}) = ((ts(c'_p)\ \varepsilon),\ ts(c'_p)\ c'_{q_1},\ \ldots,\ c'_{q_k})$。也就是说，首先将 p 分量中到达的状态 c_p^1 （从 c_p^2 出发）与在 ℓ 中累积的序列 w 结合，得到实际状态 $c_p^3 \in C_p^{\setminus \ell}$，然后执行 $\mathcal{C}^{\setminus \ell}$ 中的 b-转移。从新状态 c'_p 计算新的真实状态，并将 p 和 ℓ 的状态设为新的真实状态，同时重置 ℓ 分量中的 ℓ-局部转移序列。

通过检查每种类型的转移所需满足的条件，可以证明 $\mathcal{C}$ 确实是 $\mathcal{A}$ 的控制器。前面三种类型不会带来问题，这仅仅是因为 $\mathcal{C}$ 的定义方式如此。对于第四种类型，则依赖于这样的观察：在通信之间，p 分量和 ℓ 分量彼此独立地推进。因此，可以按照上文所述的方式将它们的推进结合起来，从而重新获得 $\mathcal{C}^{\setminus \ell}$ 的一次实际运行。

给定 $\mathcal{C}$ 中的任意运行，都可以构造一个等价的 良序运行，使得在涉及 p 和 ℓ 的两个动作之间，我们将所有 ℓ 的局部动作推迟到末尾。这意味着我们只需关注 $\mathcal{C}$ 的良序运行。然后，对于 $\mathcal{C}$ 的任意良序运行 ρ，只需在每两个真实状态之间（如有需要）添加（唯一确定的）缺失动作 $ch(a)$ 和 $ch(f)$，即可将其转换为序列 $\rho^{\setminus \ell}$。通过对转移进行分类讨论，可以根据 ρ 的长度利用归纳法得到以下结论：

1. $\rho^{\setminus \ell}$ 是 $\mathcal{C}^{\setminus \ell}$的一次定义良好的运行；2. 对于进程 $q \notin \{p, \ell\}$， q 在 ρ 结束时的状态与 $\rho^{\setminus \ell}$中的相同；3. 对于进程 p 和 ℓ，我们有： p（分别为 ℓ）状态的 π 投影在 ρ 结束时，与 p 状态的 $\pi^{\setminus \ell}$ 投影中的 p （分别为 ℓ）分量相同；4. 如果 ρ 是极大的，则 $\rho^{\setminus \ell}$ 也是极大的。

结合这些性质以及 $C^{\setminus \ell}$ 是 $\mathcal{A}^{\setminus \ell}$的获胜控制器这一事实，可得 $\mathcal{C}$ 是 $\mathcal{A}$的获胜控制器，从而证明得证。

4.3 在树状架构上求解控制问题

定理4 将 [22]从二元通道上的叶节点消除过程推广到了树状架构。反复执行此步骤，可将分布式控制问题化简为只涉及单个进程的问题，即通常的奇偶博弈。在 [22]中论证了所得奇偶博弈的大小为 $\mathrm{Tower}_d(n)$，其中 n 是异步自动机的大小，而 $d \geq 0$ 是进程树的深度。在 [12]中给出了控制问题的匹配下界。对于下界，我们将在下面基于以下引理给出详细论证：

引理 5。 给定 $n+1$ 奇偶条件 p_0 和 p_1、……、p_n，存在一个确定性奇偶自动机，其具有 $O\left((p_0+p)\cdot {}^{p_1\cdot(p_0+1)^p}(\Pi_{i>0}(p_i!))\right)$ 个状态和 p_0+p 个优先级，其中 $p=\sum_{i>1} p_i$读取由 $n+1$ 个优先级组成的元组，并且当且仅当所有 $n+1$奇偶条件都被接受时接受。

证明。 我们使用索引出现记录构造的一个变体 [25,19] ，将 $n+1$ 个奇偶条件转换为一个条件。首先，我们使用索引出现记录，将条件 p_1、……、p_n 转换为一个具有 p 个优先级的奇偶条件。这是索引出现记录的直接应用，只需注意每个奇偶条件内部的顺序始终保持不变。因此，我们保留 p 个索引的一个排列，但要求属于同一个奇偶条件的所有索引始终按递减顺序排列。因此，这样的排列共有 $\frac{p!}{(\prod_{i>0}(p_i!))}$ 个。接下来需要保留 p_0 中的元素在 p 个索引的序列中的相应顺序。由于 p_0 中的元素本身按递减顺序排列，只需记录它们中有多少个出现在 p 个索引序列中的每个元素之前。因此，将选项数量乘以 $(p_0+1)^p$。于是，$\frac{p!\cdot(p_0+1)^p}{(\prod_{i>0}(p_i!))}$ 个可能的值表示 p_0+p 个索引的所有可能序列，其中每个奇偶条件的索引自身按递减顺序排列。所创建的奇偶自动机的每次转移都需要标示 p_0+p 与 1 之间的一个优先级。为此，我们记录在从一个p_0+p 序列转移到其后继序列的过程中，被访问的最左索引是什么。这使状态总数达到引理中所述的数量。注意， p_0 不会出现在指数中。

□

我们注意到，求解获胜条件为奇偶条件合取的博弈，其复杂度已知为 co-NP 完全 [4]。因此，不可能像前一个引理中描述的那样，将其转换为奇偶博弈的过程大幅简化。

我们还注意到，前一个构造的上界 [22] 假设接受条件不属于输入的一部分。

推论 1. 树状架构上的异步自动机控制问题是可判定的，且复杂度为 d-指数时间，其中 d 是树的深度。

证明。我们迭代应用定理 4。选择一个父节点 p，其叶节点为 ℓ_1、……、ℓ_n。对每个叶节点 ℓ_1、……、ℓ_n应用该定理。最终得到一个等价可实现控制问题，其中 ℓ_1、……、ℓ_n 缺失，而 p 被替换为 $p^{\setminus \ell_1 \quad \ell_n}$ $\prod_{i>0} exp\,|S_{\ell_i}|$ $p^{\setminus \ell_1 \quad \ell_n}$ $|S_p|\cdot|\Sigma|^n$ ·、……。 $p^{\setminus \ell_1 \ldots \ell_n}$ ……的状态数与（ ）成正比。此外，……的无穷条件是 $n+1$ 个奇偶条件的合取。根据引理 5，如果 p 和ℓ_1、……、ℓ_n 的奇偶优先级数分别为 p_0、、……、p_n，那么将 $p^{\setminus \ell_1}$、……、$^{\ell_n}$ 的状态数乘以 $O\left(\frac{(p_0+p)\cdot {}^{p}!\cdot(p_0+1)^{p}}{(\prod_{i>0}(p_i!))}\right)$，便可得到具有 $\sum_{i\geq 0} p_i$个优先级的奇偶条件。

由此可知，通过逐层消去叶节点，我们得到一个异步自动机，其大小为高度为 d的指数塔，其中 d 是被消去的树的深度，其奇偶指标则是树中所有进程的奇偶指标之和。

当整棵树简化为单个进程时，我们得到一个针对单个进程的控制问题，推论也就成立。 □

指数时间 d复杂度上界在推论 1 中给出，而 [12] 的一个结果达到了这一界限，该结果展示了如何将一个 d空间有界图灵机的接受问题归约为一个作用于深度为 控制问题，其运行于树架构上，深度为$O(d)$（尽管并不完全 d，因此仍存在一定差距）。

5 结论

我们重新探讨了树状进程架构下 Zielonka 异步自动机的更简单构造，以及控制问题。我们证明，Zielonka 自动机的二次构造可以从树推广到树状架构；同时还展示了这一方法如何改进此前针对三角化依赖字母表的异步自动机指数级构造。最后，我们证明分布式控制器综合同样可以顺利推广到树状架构。

尽管 Zielonka 模型中的因果同步看起来可能与真实系统相距甚远，但它代表了处理通信的第一步。通信与 Zielonka 模型中的会合机制具有不同的特点，例如，由于接收方比发送方掌握更多信息，因此通信缺少 Zielonka 同步所具有的对称性。与此同时，因果依赖（例如 Zielonka 模型中产生的因果依赖）十分重要，并且可以通过实际的通信形式（如本地广播）加以近似。在这种背景下，我们相信，相较于树形架构，树状进程架构可能为未来涉及通信的应用提供更大的潜力。事实上，树状架构允许真正的多方通信，因为信道不受限于二元连接。这样便增加了通知受通信影响的整个“区域”的要求，这表明其可能适用于交互与物理位置相关联的场景。在未来的工作中，我们正在实现一种通信基础设施，以支持 Zielonka 类型同步的一个宽松版本。我们的框架确保动作按照与理论上的（Zielonka）模型相同的偏序发生。目前，该通信框架执行的是手工建模的异步自动机。然而，如果通信架构是树状的，我们就可以基于与本文类似的思想，使用中央制定计划的自动化分布。在理论层面，我们计划研究哪些中央计划能够被分布到具有有界通信信道的树状通信框架中。

参考文献

1. Bharat Adsul、Paul Gastin、Shantanu Kulkarni 和 Pascal Weil。《关于 Mazurkiewicz 迹上的表达完备的局部过去时命题动态逻辑及其应用》。载于 Pawel Sobocinski、Ugo Dal Lago 和 Javier Esparza 编，*Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science LICS 2024 Tallinn Estonia July 8-11 2024* 2 1 2 13 2024 , , , , , , 页码：–：。ACM，。doi:10.1145/3661814.3662110。 2. S. Akshay、Ionut Dinca、Blaise Genest 和 Alin Stefanescu。《实现现实的异步自动机》。载于 *FSTTCS'13*，LIPIcs，第 213–224 页。Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik，2013。 3. Raven Beutner、Bernd Finkbeiner 和 Jesko Hecking-Harbusch，《为分布式综合转换异步博弈》。载于 *International Conference on Concurrency Theory (CONCUR'19)* 140 *LIPIcs* 26 1 26 16 ，卷 of，第：–：页。Schloss Dagstuhl - Leibniz-Zentrum f¨ur Informatik，2019。 4. Krishnendu Chatterjee、Thomas A. Henzinger 和 Nir Piterman。《广义奇偶博弈》。载于 Helmut Seidl 编，*Foundations of Software Science andComputational Structures, 10th International Conference, FOSSACS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Soft-ware, ETAPS 2007, Braga, Portugal, March 24-April 1,* 2007，*Proceedings, Lecture Notes in Computer Science* 的第 4423 卷，第 153–167 页。Springer，2007。doi:10.1007/978-3-540-71389-0_12。 5. Robert Cori、Yves M´etivier 和 Wieslaw Zielonka。《异步映射与异步元胞自动机》。*Inf. Comput.*，106(2):159–202，1993。URL：https://doi.org/10.1006/inco.1993.1052，doi:10.1006/INCO.1993.1052。 6. Volker Diekert 和 Anca Muscholl。《关于 M´etivier 为三角剖分图构造异步自动机的注记》。*Fundam. Informaticae* 25 3 241 246, , （）：–。1996 10 3233 1996 253402 doi:。/FI- -。 7. Volker Diekert 和 Grzegorz Rozenberg 编。*The Book of Traces*。World Scientific，1995。doi:10.1142/2563。 8. Bernd Finkbeiner 和 Sven Schewe。《一致分布式综合》。载于 *IEEE Symposium on Logic in Computer Science (LICS' 05)* 321 330, 2005 *20th Annual* ，第–页。doi:10.1109/LICS.2005.53。 9. Paul Gastin、Nathalie Sznajder 和 Marc Zeitoun。《面向良连接架构的分布式综合》。*Formal Methods Syst. Des.*，34(3):215–237，2009。URL：https://doi.org/10.1007/s10703-008-0064-7，doi:10.1007/ S10703-008-0064-7。 10. F˘anic˘a Gavril。《树中子树的交图恰好是弦图》。*Journal of Combinatorial Theory, Series B*，16 1 47 56，1974（）：–。URL：https://www.sciencedirect.com/science/article/pii/009589567490094X，doi:10.1016/0095-8956(74)90094-X。 11. Blaise Genest、Hugo Gimbert、Anca Muscholl 和 Igor Walukiewicz。《确定性异步自动机的最优 Zielonka 型构造》。载于 *Automata, Languages and Programming ICALP 2010* 6199 *Lecture Notes in Com-* , , *Automata puter Science* 52 63 2010 10 1007 978 3 642 14162 1 _5 的卷，第 – 页。Springer，。doi：。/ - - - - \。 12. Blaise Genest 、 Hugo Gimbert 、 Anca Muscholl 和 Igor Walukiewicz。《树架构上的异步博弈》。载于 Fedor V. Fomin、Rusins Freivalds、Marta Z. Kwiatkowska 和 David Peleg 编，*Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12,* 2013，*Proceedings, Part II, Lecture Notes in Computer Science* 的第 7966 卷，第 275–286 页。Springer，2013。doi:10.1007/978-3-642-39212-2_26。

13. Blaise Genest 和 Anca Muscholl。构造指数大小的确定性 Zielonka 自动机。载于 *Automata, Languages and Programming, ICALP 2006*，第 565–576 页。Springer，2006。14. Hugo Gimbert。具有因果记忆的分布式异步博弈不可判定。*Logical Methods in Computer Science*，第 18 卷，第 3 期，2022年9月。doi:10.46298/lmcs-18(3:30)2022。15. Daniel Hausmann、Mathieu Lehaut 和 Nir Piterman。保持树状通信拓扑的重配置语言分布。载于 *Automated Technology for Verification and Analysis - 22nd International Symposium, ATVA 2024, Kyoto, Japan, October 21-24, 2024, Proceedings*，《计算机科学讲义》。Springer，2024。16. Siddharth Krishna 和 Anca Muscholl。具有无环通信结构的 Zielonka 自动机的二次构造。 *Theor. Comput. Sci.*，503：109–114，2013。doi:10.1016/j.tcs.2013.07.015。17. Orna Kupferman 和 Moshe Y. Vardi。综合分布式系统。载于 *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*，LICS '01，第 389 页，美国，2001。IEEE计算机学会。18. Stephane Lafortune、Karen Rudie 和 Stavros Tripakis。Ramadge-Wonham 监督控制理论三十年：回顾与未来展望 [ ] *IEEE Control Systems Magazine* 38 4 111 112, 2018 10 1109 2018 2830083 会议报告 。，( )：– 。doi： 。/MCS。 。 。19. Christoph L¨oding。自动机变换方法：复杂度及其与二阶逻辑的联系。硕士论文，基尔克里斯蒂安-阿尔布雷希特大学，德国基尔，1999。可访问：https://www.lics.rwth-aachen.de/ global/show_document.asp?id=aaaaaaaaabcqdty。20. Antoni Mazurkiewicz。并发程序模式及其解释。*DAIMI Report Series*，(78)，1977。21. Madhavan Mukund 和 Milind A. Sohoni。在分布式系统中追踪最新传闻。 *Distributed Comput.*，10(3)：137–148，1997。doi:10.1007/S004460050031。22. Anca Muscholl 和 Igor Walukiewicz。面向无环架构的分布式综合。 *arXiv preprint arXiv:1402.3314*，2014。会议版本：FST&TCS 2014。23. A. Pnueli 和 R. Rosner。分布式反应式系统难以综合。载于 *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*，第 746–757 页，第 2 卷，1990。doi:10.1109/FSCS.1990.89597。24. Peter Ramadge 和 Walter Wonham。一类离散事件系统的监督控制。*SIAM Journal on Control and Optimization*，25：206–230，1987年1月。doi:10.1137/0325013。25. Shmuel Safra。具有强公平性接受条件的 ω自动机的指数级确定化（扩展摘要）。载于 S. Rao Kosaraju、Mike Fellows、Avi Wigderson 和 John A. Ellis 编， *Proceedings of the 24th Annual ACM Symposium on Theory of Computing May 4-6 1992 Victoria British , , , , , Columbia Canada* 275 282 1992 10 1145 129712 129739 第 – 页。ACM，。doi： 。/ 。 。26. Wieslaw Zielonka。有限异步自动机札记。 *RAIRO Theor.*，( )：– *Informatics Appl.* 21 2 99 135, 1987 10 1051 。URL：https://doi.org/ 。/ita/1987210200991。