基于电路的分布式计算的描述复杂性

Veeti Ahvonen[1], 、Damian Heiman[1], 、Lauri Hella[1], 和 Antti Kuusisto[1,2]

[1]坦佩雷大学，芬兰[2]赫尔辛基大学，芬兰

电子邮件：firstname.lastname@tuni.fi

arXiv:2303.04735v1 [cs.LO] 8 Mar 2023

摘要

我们研究一种现实场景下的分布式算法：分布式消息传递通过电路实现。我们证明，在这一设定中，模态替换演算 MSC 能够刻画电路的表达能力。电路与 MSC-程序之间的转换在两个方向上都是线性的。此外，我们还证明，基于 Cole-Vishkin 的着色算法可以通过对数大小的程序来描述。

关键词： 分布式计算、描述复杂性、逻辑、图着色

1 引言

分布式计算研究网络中的计算，并与包括元胞自动机和神经网络在内的多个研究领域直接相关。本文研究基于电路的分布式系统。分布式系统是一个标记有向图（允许自循环），其中节点通过相互发送消息进行通信。在每个通信轮次中，节点向其邻居发送消息，并根据（1）自身先前的状态和（2）从邻居处收到的消息更新其状态。

分布式计算的描述复杂性研究始于 [7], ，该文通过模态逻辑刻画了常数时间分布式算法的类。 [8], 取消了常数时间这一假设，并证明有限消息传递自动机 （FMPA）的表达能力可由模态替换演算 MSC刻画；MSC是在模态逻辑基础上扩展了 Datalog 风格规则的演算。论文 [7]和 [8]没有考虑标识符，也就是类似于 IP 地址的 ID 号码。

本文研究带有标识符场景下基于电路的分布式计算。每个节点都运行同一个电路 C 的副本。在每个通信轮次中，节点将当前位串 s 发送给其邻居，并通过将 s 和邻居发送的字符串 s_1、.....、s_m 输入 C（令 s' 为 C 的输出），更新为新的字符串 s' 。这是一个符合现实的分布式计算模型，同时也考虑了局部计算——即电路的计算步骤。在分布式计算中，通常只有通信步骤计入开销。由于我们研究的是分布式系统，因此将这些电路称为消息传递电路，或 MPC，尽管严格来说它们只是普通电路。

我们证明了这种基于电路的模型与逻辑 MSC 完全对应。不同于以往关于分布式计算描述复杂性的研究，我们采用电路风格的范式，在这种范式中，算法由一个分配函数 F 指定，该函数生成

在最简单的情况下，对于每个输入 $n \in \mathbb{Z}_+$，都会生成一个电路 $F(n)$，该电路作用于所有大小为 n 的分布式系统（即标记有向图或克里普克模型）。作为主要结果之一，我们证明 MSC 逻辑程序与常数扇入消息传递电路之间可以相互转换，且大小仅有线性 增长。因此，我们可以交替使用电路分配函数和 MSC 程序分配函数。相关的形式化表述如下，其中 Π 表示所考虑的命题符号集合（包括用于标识符位的命题符号）， Δ 是图的度数上界。

定理 5.5。 给定一个大小为 m 、针对(Π, Δ)的MPC ，我们可以构造一个等价的 Π-程序，其MSC为。对于MPC的扇入有常数上界 c 的情形，该程序的大小为 $\mathcal{O}(m)$。

定理 5.6。给定 Π、 Δ 以及一个大小为 m、其MSC 为 Π-程序，我们可以构造一个等价的 MPC ，针对(Π, Δ) ，其大小为 $\mathcal{O}(\Delta m + |\Pi|)$。

我们尤其关注这样一种可行情形： $F(n)$ 是一个大小为$\mathcal{O}(\log n)$的电路。根据上述结果，我们可以证明，对于常数 Δ 和常数扇入上界，如果存在一个生成对数规模电路的分配函数，那么也存在一个生成对数规模程序的分配函数，反之亦然。我们通过证明，对于度数上界为 Δ 的图，可以生成大小为 $\mathcal{O}(\log n)$、能够计算 $(\Delta+1)$-着色的程序，并采用 $Cole\text{-}Vishkin$ [5] style 方法，从而应用这一结果；这也意味着电路方面存在类似的结果。

总体而言，基于电路的方法非常适合研究局部计算与消息传递之间的相互作用。尽管这类作用十分重要，但分布式计算研究对其关注相对较少。我们给出了一系列相关结果。

相关工作。 如前所述，分布式计算的描述复杂性研究主要始于 [7], ，该文通过模态逻辑刻画了一系列相关复杂性类。例如，文中证明，分级模态逻辑刻画了 MB(1) 类。该类包含这样一些问题：算法可在常数时间内求解，其识别能力足以区分传入消息的多重集，但无法做到更多。该文还借助逻辑来区分部分所研究的类别。 [8], 取消了常数时间限制，并表明有限分布式消息传递自动机（FMPA）与模态替换演算 MSC 相对应；本文也研究这一逻辑。 [11], 拓展了关于 MSC 的研究，证明 MSC 与同步自动机相对应，而模态逻辑中 μ 片段的 μ-演算则同样刻画了异步分布式自动机。

此前已有研究从逻辑的角度探讨带标识符的分布式计算 [3]。论文 [3] 通过对使用标识符的某一类算法进行统一逻辑刻画来处理标识符，而我们的工作基于电路式范式，公式和电路根据模型规模给出。因此，这两种方法显然无法以一种独特明确的方式进行比较。尽管如此，我们的工作与 [3] 之间的一个简单区别是，我们将标识符逐比特地视为具体的位串。因此，我们可以表达例如“当前标识符有一个比特为 1”，而不意味着当前节点不可能拥有系统中最小的标识符。这是因为我们无法保证当前图（或分布式系统）中的标识符集合是什么；而在有向图中，我们甚至无法遍历整个图来查明这一点。另一方面， [3] 中的逻辑可以表达例如“当前节点拥有最大的标识符”，而我们无法做到这一点。当然，借助非一致公式分配函数，电路式范式甚至可以指定不可计算性质。

与本文最接近的工作是 [8], ，该文通过 MSC 给出了前文提到的有限消息传递自动机刻画。该论文并未采用电路式范式。此外，我们无法将电路转换为有限消息传递自动机，再

使用 [8], 中的转换方法，因为这会导致大小呈指数级膨胀。此外，反向转换在 [8]中不是多项式时间的。此外，该论文没有讨论标识符或 Cole-Vishkin 算法，而且其研究基于将性质直接与单个公式关联的范式，而非我们采用的电路式方法。关于其他相关且非常及时的工作， [1] 研究了图神经网络（即 GNNs），并确立了聚合-组合GNNs 与分级模态逻辑之间的对应关系。关于 GNNs 与逻辑的其他相关工作，参见例如 [6]。关于分布式计算模型逻辑刻画的更多工作，我们提及这些学位论文 [9], [12]。

2 预备知识

我们用 $\mathbb{Z}_+$ 表示正整数集合。对于每个 $n \in \mathbb{Z}_+$，我们用 $[n]$ 表示集合 $\{1, \ldots, n\}$，并用 $[n]_0$ 表示集合 $\{0, \ldots, n\}$。令 PROP 为一个可数无限的命题符号集合。我们假设 PROP 划分为两个无限集合 PROP_0 和 PROP_1, ，直观上，PROP_0 包含普通命题符号，而 PROP_1 由专用于编码 ID 号码的特殊命题符号组成。我们用 $\Pi \subseteq$PROP 表示命题符号的有限集合。Π_0 （分别为 Π_1）表示 Π 中包含普通（分别为特殊）命题的子集。集合 PROP 配备了线性序 $<^{\text{PROP}}$，该线性序也在任意集合 $S \subseteq$PROP 上诱导出线性序 $<^S$。

设 Π 为一个命题符号集合。定义在 Π 上的**克里普克模型** 是一个结构 (W, R, V)，其非空论域为 W，**可达关系** 为 $R \subseteq W \times W$，**赋值函数** $V\colon \Pi \to \mathcal{P}(W)$ 为每个 $p \in \Pi$ 指定一个节点集合 $V(p)$，在这些节点中，p 被视为真。**带指定点的克里普克模型** 是一个对 (M, w)，其中 M 是一个克里普克模型， w 是 M 论域中的一个节点。我们用 succ(w) 表示集合 $\{\, v \in W \mid (w, v) \in R \,\}$。

与 [7, 8], 中一样，我们用克里普克模型来刻画分布式系统。边 $(w, u) \in R$ 通过可达关系 R 将节点 w 连接到 u，这意味着 w 可以看到由 u发送的消息。因此，我们采用 [7, 8] 的约定，即消息传递的方向与 R 的边相反。另一种做法是考虑仅包含逆模态的模态逻辑，也就是说，基于逆可达关系 R^{-1} 的模态逻辑。

设 $k \in \mathbb{N}$，并考虑一个由 k 比特字符串 $\overline{b}_j$ 组成的无限序列 $S = (\overline{b}_j)_{j \in \mathbb{N}}$。设 $A \subseteq [k]$ 和 $P \subseteq [k]$ 为子集，分别称为注意位和打印位（严格来说，也称为位位置）。令 $(\overline{a}_j)_{j \in \mathbb{N}}$ 和 $(\overline{p}_j)_{j \in \mathbb{N}}$ 为字符串 S 中子串所构成的相应序列，也就是说，$(\overline{a}_j)_{j \in \mathbb{N}}$ 记录位置属于 A 的子串；$(\overline{p}_j)_{j \in \mathbb{N}}$ 也类似。令 $(\overline{r}_j)_{j \in \mathbb{N}}$ 为位置属于 $A \cup P$ 的子串序列。我们称 S 在第 n 轮接受，当且仅当 $\overline{a}_n$ 中至少有一个比特为 1，且对于每个 $m < n$，相应的 $\overline{a}_m$ 中所有比特均为 0。此时 S 也输出 $\overline{p}_n$。更准确地说， S 关于 (k, A, P) 在第 n 轮接受，而 $\overline{p}_n$ 是 S 关于 (k, A, P) 的输出。序列 $(\overline{r}_j)_{j \in \mathbb{N}}$ 是关于 (k, A, P) 的指定序列，而向量 $\overline{r}_j$ 是第 j 轮的指定字符串。

2.1 逻辑

对于一组命题符号 Π，ML(Π)-**公式** 的集合由以下文法给出

$$\varphi ::= \top \mid p \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \Diamond\varphi$$

其中 $p \in \Pi$ 和 $\top$ 是逻辑常量符号。此类公式 φ在带指定点的克里普克模型 (M, w) 中的真值按通常方式定义：$(M, w) \models p \Leftrightarrow w \in V(p)$，且对于某个 $v \in W$，$(M, w) \models \Diamond\varphi \Leftrightarrow (M, v) \models \varphi$，使得 $(w, v) \in R$。$\top$、$\neg$、$\wedge$ 的语义与通常情况相同。

现在，固定一个模式变量集合 VAR $:= \{ V_i \mid i \in \mathbb{N} \}$。我们主要使用元变量 X、Y、Z 等来表示 VAR 中的符号。集合 VAR 配有一个线性序 $<^{\mathrm{VAR}}$，它在任意 $\mathcal{T} \subseteq$ VAR 上诱导出相应的线性序 $<^{\mathcal{T}}$。给定集合 $\mathcal{T} \subseteq$ VAR 和集合 $\Pi \subseteq$ PROP，$(\Pi, \mathcal{T})$ **模式** 的**模态替换演算**（或 MSC）是由以下文法生成的集合

$$\varphi ::= \top \mid p \mid V_i \mid \neg\varphi \mid (\varphi \wedge \varphi) \mid \Diamond\varphi$$

其中 $p \in \Pi$ 和 $V_i \in \mathcal{T}$。MSC（在 Π 上）的一个 **终止子句** 是形如$V_i(0) :\!- \varphi$的字符串，其中 $V_i \in$ VAR 且 $\varphi \in$ ML(Π)。MSC（在 Π 上）的一个 **迭代子句** 是形如 $V_i :\!- \psi$ 的字符串，其中 $V_i \in$ VAR，且对于某个集合 $\mathcal{T} \subseteq$ VAR，ψ 是一个（$\Pi, \mathcal{T}$）-模式。在终止子句 $V_i(0) :\!- \varphi$中，符号 V_i 是该子句的**头谓词**，而 φ 是该子句的**主体**。类似地，V_i 是迭代子句 $V_i :\!- \psi$的头谓词，而 ψ 是其主体。

设 $\mathcal{T} = \{Y_1, \ldots, Y_k\} \subseteq$ VAR 是一个有限、非空的集合，其中包含互不相同的模式变量。MSC 的一个（$\Pi, \mathcal{T}$）**-程序** Λ 由两个列表组成

$$\begin{array}{ll} Y_1(0) :\!- \varphi_1 & Y_1 :\!- \psi_1 \\ \vdots & \vdots \\ Y_k(0) :\!- \varphi_k & Y_k :\!- \psi_k \end{array}$$

这些列表由子句（或规则）组成。第一个列表包含 k 个在 Π 上的终止子句，第二个列表包含 k 个迭代子句，其主体是（$\Pi, \mathcal{T}$）-模式。为了完整定义 Λ，我们还固定两个集合 $\mathcal{P} \subseteq \mathcal{T}$ 和 $\mathcal{A} \subseteq \mathcal{T}$，分别由 Λ 的打印谓词和注意谓词组成。集合 $\mathcal{P} \cup \mathcal{A}$ 是 Λ 的指定谓词集合。若 Λ 是某个 $\mathcal{T} \subseteq$ VAR 对应的（$\Pi, \mathcal{T}$）-程序，我们就称 Λ 为 Π-程序。Λ 的头谓词集合记作 HEAD(Λ)。对于每个变量 $Y_i \in$ HEAD(Λ)，我们用 Y_i^0表示终止子句 $Y_i(0) :\!- \varphi_i$的主体。递归地，假设我们已为每个$Y_i \in$ HEAD(Λ)定义了一个 ML(Π)-公式 Y_i^n。令 φ_j 表示 Y_j 的迭代子句的主体。公式 Y_j^{n+1}由将 φ_j 中的每个 Y_i 替换为 Y_i^n得到。于是，Y_i^n是 Y_i 的第 n 个迭代公式。假设我们已固定一个（$\Pi, \mathcal{T}$）-程序 Λ；如果 φ 是一个（$\Pi, \mathcal{T}$）-模式，那么我们用 φ^{n+1} 表示从模式 φ 出发，同时将每个 $Y_i \in$ HEAD(Λ)替换为 Y_i^n后得到的 ML(Π)-公式。现在，设（M, w）是一个带标记的 Π-模型，且 Λ 是一个 MSC 的 Π-程序。我们定义（M, w）$\models \Lambda$，当且仅当存在某个 n 以及 Λ 的某个注意谓词 Y，使得（M, w）$\models Y^n$。在第 3 节中，我们还将使用打印谓词为 MSC 定义输出条件。

假设 $p_1, \ldots, p_\ell$ 按照顺序 $<^{\mathrm{PROP}}$ 枚举了 Π 中所有指定命题。对于 Kripke 模型 M 中的每个节点 w，我们令 ID(w) 表示 w 的标识符，也就是一个 $|\Pi_1|$ 位串；当且仅当 $(M, w) \models p_i$ 时，ID(w) 的第 i 位为 1。模型 M 是一个带标识符的 Kripke 模型，即对于 M 中任意两个不同节点 w 和 w'，均有 ID(w) $\neq$ ID(w')。我们令 $\mathcal{K}(\Pi, \Delta)$ 表示 Π 上所有带标识符的有限 Kripke 模型 (W, R, V) 所构成的类，其中每个节点的出度至多为 $\Delta \in \mathbb{N}$。对于节点 w，设 $s_1, \ldots, s_d$ 按字典序排列，依次为 succ(w) 中各成员的标识符。若节点 $v \in \in$ succ(w) 是 w 的第 i 个邻居，当且仅当 ID(v) $= s_i$。与局部标识符类似，若 $p_1, \ldots, p_m$ 按照顺序 $<^{\mathrm{PROP}}$ 枚举了 Π 中的所有命题，那么，Π 上的 Kripke 模型 M 中节点 w 的局部输入是一个 m 位串 t；当且仅当 $(M, w) \models p_i$ 时，t 的第 i 位为 1。现在，设 Λ 是一个 MSC 的 $(\Pi, \mathcal{T})$-程序，ψ 是一个 $(\Pi, \mathcal{T})$-模式。我们令 md(ψ) 表示 ψ 的模态深度（即 ψ 中菱形算子 $\Diamond$ 的最大嵌套深度）。我们令 mdt(Λ)（分别地，mdi(Λ)）表示 Λ 的终止子句（分别地，迭代子句）主体的最大模态深度。我们用 SUBS(Λ) 表示 Λ 的所有子模式的集合，其中包括头谓词和迭代

子句及终止子句的主体。若 S 是模式的集合，则 SUBS(S) 是 S 中所有模式的全部子模式所构成的集合。

2.2 电路与分布式计算

布尔电路是一个有向无环图，其中每个入度非零的节点都标记为符号 $\wedge$、$\vee$、$\neg$ 之一。电路的节点称为门。门 u 的入度称为其扇入，门 u 的出度称为其扇出。电路的输入门恰好是扇入为零的门；这些门不标记为 $\wedge$、$\vee$、$\neg$。输出门是扇出为零的门；我们允许电路有多个输出门。每个标记为 $\neg$ 的门，其扇入均为 1。电路 C 的大小 $|C|$ 是其中门的数量。电路 C 的深度 $d(C)$ 是从输入门到输出门的最长路径长度（边数）。电路 C 中门 G 的高度 $h(G)$ 是从输入门到该门 G 的最长路径长度。因此，输入门的高度为零。电路的输入门和输出门都按线性顺序排列。一个有 n 个输入门和 k 个输出门的电路计算一个类型为 $\{0,1\}^n \to \{0,1\}^k$ 的函数。其计算方式很自然，类似于对应于 $\wedge$、$\vee$、$\neg$ 的布尔运算符；形式化定义见例如 [10] 。电路的输出是由输出门的输出比特确定的二进制字符串。请注意，标记为 $\wedge$、$\vee$ 的门可以具有任意扇入（包括 0）。扇入为零的 $\wedge$ 门始终输出 1，因此对应于 $\top$ 符号。扇入为零的 $\vee$ 门始终输出 0，因此对应于 $\bot$ 符号。

根据一个布尔公式，通过考察其逆树表示，即可轻松定义相应的电路，即边指向相反方向的树表示。在逆树表示中，如果存在一条从 w 指向 v 的边，则节点 v 是 w 的**父节点**。于是，w 是 v 的**子节点**。注意，输入门没有任何子节点，而输出门没有父节点。w 的**后代节点**定义如下：w 的每个子节点都是 w 的后代节点，并且 w 的后代节点的每个子节点也都是 w 的后代节点。

定义 2.1。设 Π 为一组命题，且 $\Delta \in \mathbb{N}$。(Π, Δ) 的**电路**是一个电路 C，它规定了一个函数

$$f : \{0,1\}^{|\Pi|+k(\Delta+1)} \to \{0,1\}^k$$

对于某个 $k \in \mathbb{N}$。数值 k 称为 C 的**状态长度**。电路 C 还与集合 $A \subseteq [k]$ 和 $P \subseteq [k]$ 相关联，它们分别由**注意位**和**打印位**组成。为方便起见，我们也将 (Π, Δ) 的电路 C 称为 (Π, Δ) 的**消息传递电路**（或 MPC）。

集合 $A \cup P$称为电路的**指定比特** 集合。如果 C是**适用于**带有标识符的克里普克模型 M 的电路，那么 C 就是针对 (Π, Δ) 的消息传递电路，其中 Π恰好是由 M解释的命题符号集合，并且 Δ 至少等于 M中节点的最大出度。带有 $|\Pi_1| = m$、针对 (Π, Δ) 的电路 C称为**用于** m 标识符**比特的电路**。我们用 $\mathrm{CIRC}(\Pi_0, \Delta)$ 表示所有满足以下条件的电路 C 构成的集合：存在某个 Π，且 $\Pi \cap$属于 $\mathrm{PROP}_0 = \Pi_0$，使得电路 C是针对 (Π, Δ) 的电路。需要强调的是，严格来说，在指定 MPC 时，我们应始终（连同电路一起）指定集合 Π、Δ、注意位和打印位，以及输入门和输出门的顺序。

在给出克里普克模型 $M \in \mathcal{K}(\Pi, \Delta)$ 中带有电路 C for (Π, Δ) 的分布式计算的形式化定义之前，我们先非正式地描述这一过程。每个节点 u of M都会运行电路 C 的一个副本。节点 u 与一个局部输入相关联，其定义如下。假设 p_1、.....、$p_{|\Pi|}$ 按照顺序 $<^{\mathrm{PROP}}$枚举 Π 中的所有命题。u 处的一个$local\ input$ 是一个 $|\Pi|$ 比特的二进制字符串 $\overline{s}$，满足：$\overline{s}$ 的第 i 个比特为 1，当且仅当 $u \in V(p_i)$。在

计算开始时，u处的电路读取字符串 $\overline{s}\cdot 0^{\ell}$，其中 u 处的局部输入为 $\overline{s}$，而 $\ell = k(\Delta+1)$，因此 0^{ℓ} 只是输入 C 中不对应命题符号的那一部分。随后，电路进入一个$state$，即 C 的 k 比特输出字符串。令 $s(0,u)$ 表示该字符串；我们称其为节点 u处**state in communication round0**。现在，递归地假设对于每个节点 u，我们已知通信轮次 $n\in\mathbb{N}$ 中的状态 $s(n,u)$。那么，u 在轮次 $n+1$ 中的状态 $s(n+1,u)$ 按如下方式计算。

1. 在每个节点 u，电路都会向节点 w 发送 $s(n,u)$，使得 $R(w,u)$。请注意，消息的流向与 R-边的方向相反。2. u 处的电路将其状态更新为 $s(n+1,u)$，即电路以 $\overline{s}\cdot\overline{s}_0\cdots\overline{s}_\Delta$ 为输入时产生的 k 比特字符串，其中 $\overline{s}\cdot\overline{s}_0\cdots\overline{s}_\Delta$ 是按如下方式指定的 $i\in\{0,\ldots,\Delta\}$ 个 k 比特字符串 s_i 的拼接。

- 字符串 $\overline{s}$ 是 u 处的局部输入。
- 字符串 $\overline{s}_0$ 是状态 $s(n,u)$。
- 令 $i\in\{1,\ldots,m\}$，其中 $m\le\Delta$ 是 u 的出度。那么，$\overline{s}_i$ 是 u 的第 i 个邻居 v_i 的状态 $s(n,v_i)$。
- 对于 $i>m$，有 $\overline{s}_i = 0^k$。

接下来，我们形式化地定义 MPC 的计算。对于 (Π,Δ) 的 MPC C（状态长度为 k）以及克里普克模型 $M=(W,R,V)\in\mathcal{K}(\Pi,\Delta)$，共同定义了一个同步分布式系统，该系统执行一个由 ω 个轮次组成的序列，定义如下。每一轮 $n\in\mathbb{N}$ 定义一个**全局配置** $f_n\colon W\to\{0,1\}^k$。令 $\overline{t}_w$ 表示与 w 处为真的命题集合相对应的二进制字符串（即局部输入）。第 0 轮的配置是函数 f_0，使得 $f_0(w)$ 是 C 以 $\overline{t}_w\cdot 0^{k(\Delta+1)}$ 为输入所产生的 k 比特二进制字符串。递归地，假设我们已经定义了 f_n。令 v_1、.....、$v_m\in\mathsf{succ}(w)$ 为 w $(m\le\Delta)$ 的邻居，并按其标识符顺序排列。令 $\overline{s}_w$ 为以下 k 比特二进制字符串的拼接$\overline{t}_w\cdot\overline{s}_0\cdots\overline{s}_\Delta$，使得

1. $\overline{s}_0 = f_n(w)$,
2. $\overline{s}_i = f_n(v_i)$ for each $i\in\{1,\ldots,m\}$,
3. $\overline{s}_j = 0^k$ for $j\in\{m+1,\ldots,\Delta\}$.

那么，$f_{n+1}(w)$ 就是 C 在输入 $\overline{s}_w$下的输出字符串。现在，考虑序列 $(f_n(w))_{n\in\mathbb{N}}$ 在 w处产生的 C k 比特字符串。假设序列 $(f_n(w))_{n\in\mathbb{N}}$ 相对于 (k,A,P) 在第 n 轮接受（分别为输出 $\overline{p}$）。那么，w **接受**（分别为 **输出** $\overline{p}$）的轮次为 n。注意，在第 n 轮之后，w 处的电路仍会继续执行。

给定一个克里普克模型 $M=(W,R,V)$，**解标记**是一个将节点映射到字符串的函数 $W\to\{0,1\}^*$。这些字符串表示分布式计算中各节点的输出。例如，我们可以用对应于“是”和“否”的字符串标记节点。M 的**部分解标记**是一个从 W 到 $\{0,1\}^*$ 的偏函数，也就是说，它是某个 $U\subseteq W$ 上的类型为 $U\to\{0,1\}^*$ 的函数。部分解标记允许 W 中某些节点出现“发散计算”。电路 C 在模型 $M=(W,R,V)$ 上的**全局输出**是一个函数 $g\colon U\to\{0,1\}^*$，满足

1. $U\subseteq W$,
2. 对所有 $w\in U$，电路 C 在某一轮次 n 中输出 $g(w)$，并且
3. C 不会针对任何 $v\in W\setminus U$产生输出。

现在，固定一个有限的命题符号集合 $\Pi_0 \subseteq \mathrm{PROP}_0$ 。直观而言，这些是模型中的“实际”命题，而标识符命题集合会随着模型规模的增大而增长。令 $\mathcal{M}(\Pi_0)$ 表示所有带有标识符且具有命题符号集合 Π、满足 $\Pi \cap \mathrm{PROP}_0 = \Pi_0$ 的有限克里普克模型 M 所构成的类。因此， Π_0 对于 $\mathcal{M}(\Pi_0)$ 中的所有模型都相同，但标识符的符号会有所不同。考虑一个子类 $\mathcal{M} \subseteq \mathcal{M}(\Pi_0)$。现在，定义在 $\mathcal{M}$ 上的分布式计算**问题**是一个映射 p ，其定义域为 $\mathcal{M}$ ，并为每个输入 M 关联一个（可能无限的）部分解标记集合 $p(M)$，对应于 M。集合 $p(M)$ 表示问题 p 在 M 上的可接受答案集合。许多图问题（例如着色）自然涉及这样一组答案标记。

对于 $\Delta \in \mathbb{N}$，我们用 $\mathcal{M}(\Pi_0, \Delta)$ 表示将 $\mathcal{M}(\Pi_0)$ 限制到最大出度为 Δ 的模型后得到的模型。$\mathcal{M}(\Pi_0, \Delta)$ 的电路序列是一个函数 $F\colon \mathbb{Z}_+ \to \mathrm{CIRC}(\Pi_0, \Delta)$，满足 $F(n)$ 是一个用于 $\lceil \log n \rceil$ 个标识符位的电路。现在，如果对于定义域大小为 n 的每个 $M \in \mathcal{M}(\Pi_0, \Delta)$， $F(n)$ 的全局输出都属于 $p(M)$，那么 F 就能解决 p 关于 $\mathcal{M}(\Pi_0, \Delta)$ 的问题。令 $c \in \mathbb{N}$。我们将 $\mathsf{DCC}^{c}_{\Delta}[\log n]$ 定义为这样一类分布式计算问题：它们可由一个电路序列 F 解决，其中电路的最大扇入为 c，且 $F(n)$ 的大小为 $\mathcal{O}(\log n)$。与之相关的 LogSpace 一致类要求每个 F 都能在 LogSpace 内计算。DCC 是 “ 通过电路进行分布式计算”（distributed computing by circuits）的缩写。

3 扩展 MSC

这里我们定义 MSC 的一些扩展，主要用作后续证明中的工具。设 Π 为一组命题， $\mathcal{T}$为一组模式变量。令 $\Delta \in \mathbb{N}$。在多模态 MSC（或 MMSC）中，我们用算子 $\Diamond_1$、.....、 $\Diamond_\Delta$取代 $\Diamond$，其余语法与 MSC 相同。模式 $\Diamond_i \varphi$ 只询问 φ 在第 i个邻居处是否为真。更形式化地说，若 (M, w) 是一个带有标识符的带指定点的克里普克模型，则 $(M, w) \models \Diamond_i \varphi \Leftrightarrow (M, v_i) \models \varphi$，使得 $(w, v_i) \in R$，且 v_i 是 w的第 i个邻居。注意，若 w 的出度小于 i，则 $\Diamond_i \varphi$ 在 w处为假。MMSC 的 (Π, Δ)-程序与 MSC 的 Π-程序完全相同，但允许使用的算子是 $\Diamond_1$、.....、 $\Diamond_\Delta$，而不是 $\Diamond$。MMSC 的 Π-程序Λ 对于任意 $\Delta \geq d$都是一个 (Π, Δ)-程序，其中 d 是 Λ中任意菱形里的最大子索引。我们也为 MMSC 程序固定打印谓词和注意谓词。请注意，MMSC 并非通常意义上的逻辑，因为算子 $\Diamond_i$ 需要有关定义标识符的谓词的信息。可以通过签名变更，并将注意范围限制为关系的出度至多为一的多模态模型来解决这一问题。这种做法会稍显繁琐，而本文采用的当前方法已足够。

接下来，我们定义带有条件规则的 MSC （或 CMSC）。在这里，我们允许将“if-else”规则作为迭代子句。令 $\varphi_1, \ldots, \varphi_n$和 $\psi_1, \ldots, \psi_n$以及 χ为基本 MSC 的 $(\Pi, \mathcal{T})$-模式。**条件迭代子句** 是形如以下形式的规则

$$X :-_{\varphi_1, \ldots, \varphi_n} \psi_1; \ldots; \psi_n; \chi.$$

模式 φ_i 是头谓词 X 的**条件**，而模式 ψ_i 是相关的**结果**。最后一个模式 χ 称为**备份**。注意，当 $n = 0$时，我们得到一个标准 MSC 子句。CMSC 的 Π-**程序**与 MSC 的完全相同，但允许使用条件迭代子句。因此，CMSC 的程序 Λ 由 k个终止子句、 $k' \leq k$ 个条件迭代子句以及针对某个 $k \in \mathbb{Z}_+$的 $k - k'$ 个标准迭代子句组成。同样，我们还指定一些模式变量集作为打印谓词和注意谓词。

为了定义语义，我们将像在 MSC 中那样，指定每个头谓词的第 n 个迭代公式。非正式地说，我们总是使用第一个（从左往右）成立的条件 φ_i，并据此将相应的结果 ψ_i 作为规则的主体来求值。如果没有任何

条件成立，那么我们就使用备份。设 Λ 为 CMSC 的一个 Π-程序。首先，我们令头谓词 $Y_i \in \mathrm{HEAD}(\Lambda)$ 的第零个迭代子句 Y_i^0 为 Y_i 的终止子句。递归地，假设我们已为每个 $Y_i \in \mathrm{HEAD}(\Lambda)$ 定义了一个 ML(Π) 公式 Y_i^n。现在，考虑规则

$$Y_i :-_{\varphi_1,\dots,\varphi_m} \psi_1; \dots; \psi_m; \chi.$$

令 φ_j^{n+1}为通过将条件 φ_j中的每个模式变量 Y_k 替换为 Y_k^n而得到的公式。公式 χ^{n+1} 和 ψ_k^{n+1} 以类似方式得到。于是，公式 Y_i^{n+1}为

$$\bigvee_{k\le m} \Big(\Big(\bigwedge_{j<k} \neg\varphi_j^{n+1} \Big) \wedge \varphi_k^{n+1} \wedge \psi_k^{n+1} \Big) \ \vee \ \Big(\Big(\bigwedge_{j\le m} \neg\varphi_j^{n+1} \Big) \wedge \chi^{n+1} \Big).$$

通常，备用模式 χ 就是规则的头谓词 X。这意味着，如果没有任何条件成立，头谓词的真值就不会改变。若公式 φ_k^n 在第 $n \ge 1$ 轮的节点 w 处为真，且同一规则中所有“更早”的条件 φ_j^n（即 $j < k$）均不为真，则称条件 φ_k 在第 $n \ge 1$ 轮的 w 处是**hot（活跃）**的。否则，备份为 hot。若某条规则的一个条件公式为 hot，则称条件迭代子句（或相应的头谓词）在节点 w 的第 $n \ge 1$ 轮是**active（激活）**的。

最后，我们大致将**消息传递**MSC（或 MPMSC）定义为带有条件规则的多模态 MSC。**(Π, Δ)-程序**与 MMSC 中带有条件规则的 (Π, Δ)-程序完全相同，但需满足以下限制。

1. 终止子句的模态深度以及规则条件的模态深度均为零。
2. 标准迭代子句的结果、备份和主体的模态深度均至多为一。

与 MMSC 中一样，不允许使用算子 $\Diamond$。MPMSC 的 Π-程序的定义与 MMSC 的 Π-程序类似。因此，对于某个 $k \in Z_+$，MPMSC 的程序包含 k 终止子句、 $k' \le k$条件迭代子句和 $k - k'$ 标准迭代子句。我们还固定注意谓词集和打印谓词集。语义的定义与 CMSC 中相同，但要注意现在使用菱形算子 $\Diamond_i$。MPMSC 程序的非终止子句若包含至少一个菱形算子，则称为**通信子句**。若通信子句在第 $n \in \mathbb{Z}_+$ 轮的 w 时满足以下条件之一，则称其为**广播**。

1. 条件 φ_i 在 w 时是热的，且相应的结果包含一个菱形算子。
2. 备份在 w 时是热的，且包含一个菱形算子。
3. 该规则不是条件规则，但包含一个菱形算子。

接下来，我们将定义所有 MSC 变体（包括标准 MSC）中程序的接受条件和输出条件。这些接受条件将与之前给出的标准 MSC 条件一致。设 Λ 是一个程序， $\mathcal{A}$ 和 $\mathcal{P}$ 分别是注意谓词集和打印谓词集。设 $Y_1, \dots, Y_k$ 按照 $<^{\mathrm{VAR}}$ 的顺序枚举 Λ 中的头谓词。设 $M = (W, R, V)$ 是一个克里普克模型。每个轮次 $n \in \mathbb{N}$ 定义一个全局配置 g_n: $W \to \{0,1\}^k$，如下所示。第 n 轮的配置是函数 g_n，使得当且仅当 $(M, w) \models Y_i^n$ 时，$g_n(w)$ 的第 i 个比特为 1。若序列 $(g_n(w))_{n\in\mathbb{N}}$ 在第 n 轮相对于 $(k, \mathcal{A}, \mathcal{P})$ 接受（分别为输出 $\overline{p}$），则称节点 w 在第 n 轮接受（分别为输出 $\overline{p}$）。于是， n 是 Λ 在 w 处的输出轮次。若节点 w 在某个轮次 n 接受，我们记作 $(M, w) \models \Lambda$。对于消息传递 MSC 的程序 Λ 和模型 M，全局通信轮次是指这样一个计算轮次 n： M 的至少一个节点中，至少有一个通信子句正在广播。若程序 Λ 在 w 处于全局通信时间 m 输出 $\overline{p}$，则

Λ 在 w 处的输出轮次为 n，且 $m \leq n$ 是集合 $\{0, \ldots, n\}$ 中轮次的全局通信轮次数。

现在，令 $\mathcal{L}$表示我们所有 MSC 变体的所有程序的集合。令 $\mathcal{C}$ 表示所有 MPC 的集合。对于每个 $\Lambda \in \mathcal{L}$，我们称克里普克模型 M 适用于 Λ，如果M 解释（至少）所有出现在 Λ 中的命题符号。对于一个关于 (Π, Δ) 的消息传递电路，我们称 M 适用于该电路，如果 M 所解释的命题符号集合恰好是 Π，且 M 的最大出度至多为 Δ。现在，令x 和 y 为 $\mathcal{C} \cup \mathcal{L}$中的任意成员。我们称 x 和 y（接受）等价，如果对于每个同时适用于 x 和 y 的克里普克模型 M，以及模型中的每个节点 w，x 和 y 在 w 产生相同的输出，或者二者均不在 w产生任何输出。我们称 x 和 y 强等价，如果对于每个适用于 x 和 y 的 M，以及模型中的每个节点w 和每个轮次 n，对象 x 和 y 在 w产生相同的指定字符串$\overline{r}_n$。我们还为 MPMSC 和 MPC 定义一种特殊的等价概念。我们称 MPMSC 程序 Λ 与电路 C 强通信等价，如果对于每个同时适用于 Λ 和 C 的M，以及模型中的每个节点 w，该电路的指定序列S 恰好等于该程序的指定字符串序列 $(\overline{r}_j)_{j \in \{0\} \cup G}$，其中 $G \subseteq \mathbb{Z}_+$ 是该程序的全局通信轮次集合 n。最后，任意 MSC 变体程序的长度是命题符号、头谓词以及算子 $\top$、 $\neg$、 $\wedge$、 $\Diamond$、 $\Diamond_i$ 出现次数的总和。程序 Λ 的模态深度 $\mathsf{md}(\Lambda)$ 是其规则体（迭代和终止）的最大模态深度。

4 将 MPMSC 与消息传递电路联系起来

为了获得所需的描述性刻画，我们首先将 MPC 转换为 MPMSC。

4.1 从 MPC 到 MPMSC

为了最终将 MPC 转换为 MPMSC，我们将首先展示如何用 MSC 的无菱形算子程序模拟标准布尔电路的求值。设 C 是一个深度为 d、具有 ℓ 个输入门和 k 个输出门的电路。令 L 表示 MSC 的任意一种变体。固定模式变量 I_1、 $\ldots I_\ell$ 和 O_1，.....，O_k，这里给出的两个序列均按顺序 $<^{\mathrm{VAR}}$ 排列。考虑一个具有以下性质的 L 程序 Λ。

1. Λ 的模式变量集合包含（至少）变量 I_1、 $\ldots I_\ell$、 O_1、.....、O_k。2. 该程序不含菱形算子（$\Diamond$ 或 $\Diamond_i$），也不含命题符号。3. 每个模式变量 X 的终止子句为 $X(\mathbf{0}) :\!-\ \bot$。

令 $P\colon \{\bot, \top\}^\ell \to \{\bot, \top\}^k$为如下定义的函数。对于每个输入 $(x_1, \ldots, x_\ell) \in \{\bot, \top\}^\ell$到 P，将 Λ修改为一个新程序 $\Lambda(x_1, \ldots, x_\ell)$，方法是将每个终止子句$I_i(\mathbf{0}) :\!-\ \bot$改为 $I_i(\mathbf{0}) :\!-\ x_i$。令 $(y_1, \ldots y_k) \in \{\bot, \top\}^k$为第 d个迭代公式 $O_1^d, \ldots, O_k^d$的真值元组，其中回顾一下，d是我们的电路 C的深度。于是我们定义 $P(x_1, \ldots, x_\ell) := (y_1, \ldots, y_k)$。现在，如果以这种方式定义的 P 与 C计算出的函数相同，那么 Λ **模拟** 电路 C（关于 $I_1, \ldots, I_\ell$和 $O_1, \ldots, O_k$）。

引理 4.1。 对于每个大小为 m 且具有常数扇入的 C 电路，都存在一个大小为 $\mathcal{O}(m)$ 的L 程序，可以模拟 C，其中 L 是 MSC 的任意变体。

证明。 首先假设 d 的 C 深度至少为 1。我们通过使用恒等门（即扇入为 1 的 $\wedge$-门），对 C 进行修改，使每个输出门的高度恰好为 d。然后，我们为所得电路 C'的每个门定义一个模式变量。这些变量

分别为输入门的 $I_1, \ldots, I_\ell$，以及输出门的 $O_1, \ldots, O_k$。设 X为 C的一个 $\wedge$-门 G 对应的模式变量。我们定义相应的终止子句$X(0) :- \bot$和迭代子句 $X :- Y_1 \wedge \cdots \wedge Y_j$，其中 $Y_1, \ldots, Y_j$是与 G相连的门所对应的变量。类似地，对于析取门 G对应的变量 X'，我们定义规则 $X'(0) :- \bot$和 $X' :- Y'_1 \vee \cdots \vee Y'_j$，其中 $Y'_1, \ldots, Y'_j$是与 G相连的门所对应的变量。对于否定，我们定义 $X''(0) :- \bot$ 和 $X'' :- \neg Y$，其中 Y 是相连门所对应的变量。我们将与输入门相关的头谓词I_i 的终止子句设为 $I_i(0) :- \bot$。这个选择无关紧要，因为在检查程序是否模拟电路时，我们会修改终止规则以匹配输入字符串。对应的迭代子句为 $I_i :- I_i$。

最后，在极端情况下，若 C的深度为 0，因此每个输入门也是输出门，我们定义该程序的头谓词序列为 $(I_1, \ldots, I_\ell) = (O_1, \ldots, O_k)$，并使每个头谓词 $I_i = O_i$ 对应的（终止和迭代）子句为 $I_i :- \bot$。

□

定理 4.2. 给定一个大小为 m、用于(Π, Δ) 的MPC，我们可以构造一个与之通信等价的(Π, Δ)-*MPMSC* 程序。假设MPC的扇入有一个常数上界 c，则程序的大小与电路的大小呈线性关系。

证明。设 C 是一个用于 (Π, Δ)、状态长度为 k 的 MPC。我们先非正式地说明程序 Λ_C 将如何处理电路 C。程序 Λ_C 使用 k 个头谓词来模拟电路的状态。我们将使用引理 4.1 构造程序，而算子 $\Diamond_i$ 将用于模拟接收邻居发送的消息。程序 Λ_C 按照由 $d+1$ 轮组成的周期重复计算，其中 $d = d(C)$ 是 C 的深度。模拟接收邻居的消息需要一轮，其余的 d 轮用于模拟电路的求值。

接下来我们给出形式化证明。首先，我们定义一个长度为 $d(C)+1$的简单时钟，用于为程序正确计时。该时钟由头谓词$T_0, T_1, \ldots, T_{d(C)}$ 以及以下规则组成： $T_0(0) :- \bot$, $T_0 :- T_{d(C)}$, $T_1(0) :- \top$, $T_1 :- T_0$; 对于 $i \in [d(C) - 1]$, 有 $T_{i+1}(0) :- \bot$ 和 $T_{i+1} :- T_i$。每一轮中，恰有一个头谓词 T_i 为真，其余均为假。在第 0 轮中，唯一为真的谓词是 T_1; 在第 $i \in [d(C) - 1]$轮中，唯一为真的谓词是 T_{i+1}。经过 $d(C)$ 轮后， T_0 为真，下一轮时钟便会重新开始。

设 Γ_C 是一个程序，用于模拟引理 4.1 的证明中所述电路 C 的内部求值过程。我们将按如下方式改写 Γ_C 的一些迭代子句。若 X_G 是 Γ_C 中对应于非输入门 G 的头谓词，则将相应的迭代子句 $X_G :- \varphi$ 改写为 $X_G :^{\frown_{T_{h(G)}}} \varphi; X_G$，其中 $h(G)$ 是门 G 的高度。

对于每个 $\ell \in [|\Pi|]$，令 I^Π_ℓ 表示 Γ_C 的一个头谓词，它对应于 C 中读取命题 p_ℓ 真值的输入门。对于每个 $i \in [k]$ 和 $j \in [\Delta]_0$，令 $I_{(i,j)}$ 表示 Γ_C 的一个头谓词，它对应于 C 中读取第 i 个邻居的状态串的输入门。“邻居 0”指本节点。接下来，我们将改写头谓词对应于输入门的子句。对于每个 $i \in [k]$，令 O_i 表示 Γ_C 的一个头谓词，它对应于 C 的第 i 个输出门。I^Π_i 的终止（分别为迭代）子句改写为 $I^\Pi_i(0) :- p_i$ （分别为 $I^\Pi_i :- _{T_0} p_i; I^\Pi_i$）。若 $j \neq 0$，则每个 $I_{(i,j)}$ 的终止（分别为迭代）子句改写为 $I_{(i,j)}(0) :- \bot$ （分别为 $I_{(i,j)} :-_{T_0} \Diamond_j O_i; I_{(i,j)}$）。每个 $I_{(i,0)}$ 的终止（分别为迭代）子句改写为 $I_{(i,0)}(0) :- \bot$ （分别为 $I_{(i,0)} :-_{T_0} O_i; I_{(i,0)}$）。

Λ_C 的注意力谓词和打印谓词定义如下。设 C 中注意力（分别为打印）位位置的集合为 $A \subseteq [k]$ （分别为 $P \subseteq [k]$）。注意力（分别为,

prin Λ_C的 t）谓词恰好是头谓词 O_j，其中 $j \in A$（分别为 $j \in P$）。

我们分析 Λ_C 的工作原理。程序以周期性方式执行，每个周期包含$d(C) + 1$ 轮次。在第 0 轮，程序 Λ_C 读取命题符号，并使用头谓词 I_i^{Π}记录局部输入；这些谓词的真值在其余计算过程中保持不变。此外，T_1 在第 0 轮变为真。在第 1 轮，对应于高度为一的门的头谓词处于活动状态，因此会更新。（注意，输入门对应的谓词 $I_{(i,j)}$ 处于非活动状态，因为 T_0 为假。它们在第 1 轮仍为假，因为它们在第 0 轮变为假，而备份不会影响真值。）从高度一开始，执行随后继续处理对应高度为二的门的谓词，依此类推。在第 $d(C)$轮，对应输出门的头谓词 O_i 处于活动状态。如果注意谓词为真，程序也会打印。在第 $d(C) + 1$轮，谓词 T_0为真。因此，输入门谓词 $I_{(i,j)}$ 处于活动状态，周期再次开始，它们使用菱形算子 $\Diamond_i$进行更新。由此得到的真值对应于输入给电路的一个输入串。接着，我们在第 $d(C) + 2$轮模拟高度为一的门，并在后续轮次中继续模拟，直至高度为 $d(C)$ 的门，完成 Λ_C执行的第二个周期。之后的周期与此类似。因此，我们的程序 Λ_C 以周期性方式模拟电路 C 。

程序 Λ_C 与 C等价且通信等价。 Λ_C 中的通信子句是同步的，即每个节点都在相同轮次进行广播。这是因为在每个节点上模拟电路所需的时间相同。因此， Λ_C 的输出和全局通信时间与 C相匹配。根据引理 4.1，由于 C的扇入为常数，转换在 C 的大小上显然是线性的。 □

4.2 从 MPMSC 到 MPC

将 MPMSC 程序转换为电路或许更容易。所构造的 MPC 的状态串实质上存储了程序所用的头谓词和命题符号的值，并通过模拟程序子句来计算新的状态串。我们先给出以下引理，说明如何去除条件规则。证明见附录，其基本思路是将条件子句中的条件表示在标准子句中。非平凡之处在于如何保持转换的线性。

引理 4.3. 给定一个 Π-程序（属于CMSC），我们可以构造一个与之强等价的 Π-程序（属于MSC），其大小与CMSC-程序的大小呈线性关系，并且终止子句和迭代子句的最大模态深度均与原程序相同。

回顾前一个引理的证明，可以很容易地得到 MPMSC 的相应结果，因为 MPMSC 中的终止子句的模态深度总是零。

推论 4.4. 给定一个大小为 m 的 Π-程序（属于MPMSC），我们可以构造一个与之强等价的 Π-程序（属于MMSC），其大小为 $\mathcal{O}(m)$，且迭代子句的最大模态深度与原程序相同，终止子句的模态深度为零。构造出的程序中的所有菱形算子也都出现在原程序中。

令 $d \in \mathbb{Z}_+$。令 x 为命题符号或头谓词。若满足以下条件，则称符号 x 在 MPMSC 程序 Λ 中是 d- **无处不在**的。

1. 符号 x 出现在某个迭代子句的主体中，且不在任何菱形的作用域内。2. 对于每个 $i \in [d]$，符号 x 都会出现在 $\Diamond_i$作用域内某个迭代子句的主体中。

如果以下条件的析取成立，则命题符号 p 在 Λ 中是**弱 d-全显现的**。

1. 命题 p 不会出现在 Λ的任何迭代子句中。

2. 命题符号 p 在 Λ中是 d-全程出现的。

在给出从 MPMSC 程序到 MPC 的翻译之前，我们先证明以下引理。

引理 4.5. 给定一个 MPMSC 的 Π-程序 Λ 和 $d \in \mathbb{Z}_+$，我们可以构造一个与之强等价的 MPMSC Π-程序，使其满足以下条件。

1. Π 中的每个命题符号都会出现在该程序中。

2. Λ 中的每个命题符号都是弱 d-全程出现的。

3. Λ的每个头谓词都是 d-全显的。

4. 程序的大小为 $\mathcal{O}\left(d\left|\Lambda\right| + \left|\Pi\right|\right)$ 。

证明。 对于每个未出现在 Λ中的 $p \in \Pi$ ，我们通过将终止子句的某个主体 ψ替换为 $\psi \wedge (p \vee \neg p)$，将 p 添加到程序中。因此，最终得到一个每个 $p \in \Pi$ 都会出现的程序。

对于 Λ中每个不是 d-全显的头谓词 Y，我们通过将非条件迭代子句的某个模式 ψ（或条件迭代子句的备份）替换为

$$\psi \wedge (Y \vee \neg Y) \wedge \bigwedge_{i \in [d]} \Diamond_i (Y \vee \neg Y).$$

设 q 是 Λ 中一个在 Λ里不是弱 d-全显的命题符号。如果 q 确实出现在程序的某个迭代子句中，那么我们就像处理头谓词那样，使它成为 d-全显的。如果 q 没有出现在程序的任何迭代子句中，那么我们不做任何处理。

现在，我们的程序已经准备就绪。显然，它与 Λ 强等价，并且大小为 e

$$\mathcal{O}((d+1)\left|\Lambda\right| + \left|\Pi\right|) = \mathcal{O}(d\left|\Lambda\right| + \left|\Pi\right|),$$

正如所需。 □

接下来，我们证明另一个引理，它将作为把 MPMSC 程序转换为 MPC 程序的工具。该引理可用于用一个电路模拟两个电路。

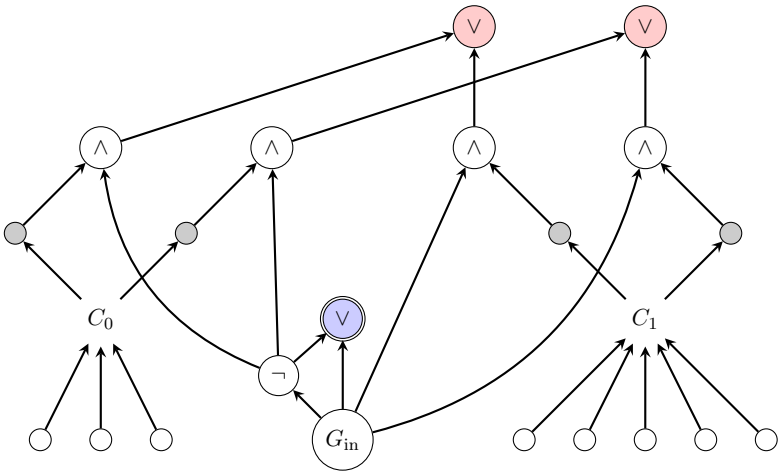
引理 4.6。 设 C_0 和 C_1 是具有 k_0 和 k_1 个输入门且输出门数量相同的电路。根据 C_0 和 C_1，我们可以构造一个具有 $k_0 + k_1 + 1$个输入门的电路 C，满足以下性质。设 $\overline{s}_0$ 是一个 k_0比特输入串， $\overline{s}_1$ 是一个 k_1比特输入串。设 $\overline{s} = \overline{s}_0 b \overline{s}_1$，其中 $b \in \{0,1\}$。

1. 如果 $b = 0$，且 $\overline{t}_0$ 是 C_0 在输入 $\overline{s}_0$时的输出，那么 C 在输入 $\overline{s}$时输出 $1\,\overline{t}_0$ 。

2. 如果 $b = 1$，且 $\overline{t}_1$ 是 C_1 在输入 $\overline{s}_1$时的输出，那么 C 在输入 $\overline{s}$时输出 $1\,\overline{t}_1$ 。

3. C 的前 k_0 个输入门等价于 C_0 的输入门，而 C 的后 k_1个输入门等价于 C_1的输入门。

证明。 在给出形式化构造之前，我们先给出一个示例（如下图所示）： C_0有 3 个输入门，C_1 有 5 个输入门，且二者都有 2 个输出门（以灰色绘制）。在图中，符号 C_0 和 C_1 分别对应这些电路；为了说明构造过程，图中仅绘制了输入门和输出门。蓝色的 $\vee$-门（以双圆绘制）是输出门，它输出比特 1，该比特是 C的输出字符串的开头。图像顶部的红色 $\vee$-门是新的输出门：如果 C_0 接收 0 作为输入，它们就模拟 C_0 的输出；否则，红色的 $\vee$-门模拟 C_1的输出。输入门 G_{in}负责将输入 $\overline{s}_0 b \overline{s}_1$ 中的比特 b 传递给 C。



为了构造 C，我们首先为新输入门 G_{in} 构造电路 C'，使得C' 无论接收什么 G_{in} 作为输入，输出都为 1。这个电路可以像示例中那样，使用一个¬-门和一个 $\vee$-门来构造；后者是双圆圈中的蓝色输出门。我们使用 C' 在 C的输出开头输出比特 1，并使用 G_{in} 和¬-门作为 C 的标志，以指示应将哪个电路（C_0 或 C_1）用作输出。

电路 C 的其余部分按如下方式构造。我们将 C_0 和 C_1 复制到 C，并将电路 C' 也添加到 C。电路 C 的输入按如下顺序排列： C_0 的输入在左侧， C' 的输入在中间，C_1 的输入在右侧。接下来，我们使用 G_{in} 和 C' 的 ¬-门，按如下方式连接电路 C_0 和 C_1 的输出门。如果G_0 是电路 C_0的第 i个输出门，并且 G_1 是电路 C_1 的第 i个输出门（回想一下，电路的输出门数量相同），那么我们可以构造一个新输出门：当且仅当 G_{in} 接收 0 作为输入时，该门输出 G_0 的值；否则，它输出 G_1的值。具体构造如下。

1. 对于 C_0 的每个输出门，我们引入一个新的 $\wedge$门，然后将 C_0 的每个输出门与这些 $\wedge$门一一对应地连接起来。我们还为 C_1 的每个输出门类似地引入一个新的 $\wedge$门，并以类似的方式将它们连接起来。

2. 我们将 C' 的 $\neg$门连接到 C_0 的新 $\wedge$门。

3. 我们将 G_{in} 连接到 C_1 的新 $\wedge$门。

4. 最后，我们引入新的 $\vee$门 $G_1, \ldots, G_p$，每个门对应 C_0 的一个输出。然后，我们将 C_0 的新 $\wedge$门与这些 $\vee$门一一对应地连接起来。最后，我们以类似的方式将 C_1 的新 $\wedge$门连接到门 $G_1, \ldots, G_p$（见图），确保正确模拟 C_0 和 C_1 的输出字符串顺序。

现在，很明显，我们的电路 C 按陈述中的要求运行。 □

现在我们可以证明以下结论。

定理 4.7。 给定 Π、 Δ 和一个大小为 m 的 Π-程序 MPMSC ， 我们可以构造一个与其强等价的 MPC ， 其对应于(Π, Δ) ， 大小为 $\mathcal{O}(\Delta m + |\Pi|)$。

证明。 下面给出证明思路。首先，我们将 MPMSC 程序转换为一个强等价的 MPMSC 程序，其中命题符号和头谓词保持不变（引理 4.5）。然后，将该程序转换为一个强等价的 MMSC 程序（推论 4.4）。基于该程序，我们构造一个 MPC，其状态串存储头谓词和命题符号的真值。该电路基本上是直接根据子句的逆树表示构造的。菱形作用域内的头谓词和命题符号将对应于用于接收相邻节点发送的比特的输入门。此外，不在菱形作用域内的头谓词和命题将对应于用于本节点的输入门。在通信轮次 0 中，电路使用由终止子句构造的子电路；在后续轮次中，使用由迭代子句构造的子电路，这可以通过应用引理 4.6 来实现。

固定 $\Delta \in \mathbb{Z}_+$ 和一个命题集合 Π。设 Λ 为一个大小为 m 的 MPMSC Π-程序。首先，我们将 Λ 转换为一个强等价的、大小为 $\mathcal{O}(\Delta m+|\Pi|)$ 的 MPMSC Π-程序 Λ'，其中命题符号和头谓词按照引理 4.5 的陈述保持不变。根据推论 4.4，对于 Λ' ，存在一个强等价的、大小为$\mathcal{O}(|\Lambda'|) = \mathcal{O}(\Delta m + |\Pi|)$ 的 MMSC Π-程序 Γ ，使得 Γ 的终止子句的模态深度为 0，且迭代子句的模态深度至多为 1。

令 $\Pi' \subseteq \Pi$ 为 Π 中出现在程序某个迭代子句中的命题集合，并令 k 为 Γ中头谓词的数量。假设 $X_1, \ldots, X_k$ 是 Γ的头谓词，其顺序为 $<^{\mathrm{VAR}}$。我们假设 $p_1, \ldots, p_{|\Pi|}$ 按顺序 $<^{\mathrm{PROP}}$ 枚举了 Π 中的命题符号。我们将从 Γ 构造一个消息传递电路 C_Γ，用于 (Π, Δ)，其状态长度为 $|\Pi'|+1+k$。 C_Γ 的状态串中最后的 k 个比特将在每一轮 n编码 X_i^n的真值。最前面的 $|\Pi'|$ 个比特将编码命题的真值——因此这些值在计算过程中是静态或恒定的（为便于处理菱形算子，将使用这些比特）。额外的那个比特表示我们是否已经求值了 Γ的终止子句，同时也表示节点是否已从其邻居接收到消息。由于具有恒定的真值， Π' 中的命题很容易处理：对于每个对应命题符号的输入门 G_{in} ，我们添加一个新的 $\wedge$-门，并将 G_{in} 连接到它，以获得恒等变换。构造的其余部分则需要更多工作。直观地说，头谓词 X_i 的比特由两个子电路更新：第一轮由对应于 X_i 终止子句的子电路更新，后续轮次则由对应于迭代子句的子电路更新。

为构造 C_Γ，我们首先说明如何将 Γ的终止子句转换为相应的电路。随后，这些电路将组合起来形成电路 C_0 ，之后它将被接入并成为最终电路 C_Γ的一部分。现在，令 X_i (0) :– φ_i 为 Γ的一个终止子句。注意，由此可知 φ_i 不含菱形算子。让我们为公式 φ_i构造一个电路 $C_{(i,0)}$ 。对于每个 $j \in [\,|\Pi|\,]$，令 $I^{\Pi}_{(j,0)}$ 表示一个输入门，其作用——直观地说——是在电路的本节点读取符号 p_j 的真值。我们将 φ_i 转换为其逆树表示。然后，对于每个 j，将树中命题符号 p_j 的每次出现替换为输入门 $I^{\Pi}_{(j,0)}$。现在，公式 φ_i 对应的电路 $C_{(i,0)}$ 已经就绪。输出门是扇出为零的门。我们将所有 i对应的电路 $C_{(i,0)}$ 组合成一个电路 C_0 ，使它们共享输入门；也就是说，将同一个门 $I^{\Pi}_{(j,0)}$ 的所有重复实例合并为一个输入门。 C_0 的输出节点按显而易见的顺序排列为电路 $C_{(i,0)}$ 的输出节点，使得若$j' < j''$，则 $C_{(j',0)}$ 排在 $C_{(j'',0)}$ 之前。

接下来，我们将为 Γ 的每个迭代子句构造一个电路。这些电路将 n

合并为一个电路 C_1，最终连接起来，成为 C_Γ 的一部分。对应于迭代子句的电路，其输入对应于通过菱形算子 $\Diamond_i$ 访问的邻居的消息。现在，设 $X_{k'} :- \psi_{k'}$ 是 Γ 的一个迭代子句。我们为模式 $\psi_{k'}$ 构造一个电路 $C_{(k',1)}$。设 $j \in [\Delta]$ 和 $i \in [k]$。令 $I_{(i,j)}$ 表示 $C_{(k',1)}$ 的一个输入门，其作用直观上是读取 X_i 在第 j 个邻居处的真值。类似地，我们用 $I_{(i,0)}$ 表示读取本节点处 X_i 的输入门。令 $I_{(0,j)}$ 表示一个输入门，其作用直观上是指示本节点是否收到了来自第 j 个邻居的消息。此外，我们用 $I^{\Pi'}_{(i,j)}$ 表示一个输入门，用于读取符号 $p_i \in \Pi'$ 在邻居 j 处的值；请注意，“邻居 0”指本节点。现在，我们首先将 $\psi_{k'}$ 转换为其逆树表示。然后，按如下方式将树中的命题和头谓词替换为输入门。首先，如果头谓词 X_i（分别地，命题符号 p_i）是树中某个 $\Diamond_j$-节点的后代节点，那么我们使用 $I_{(i,j)}$（分别地，$I^{\Pi'}_{(i,j)}$）作为替换用的输入门。（请注意，后代节点不一定是该 $\Diamond_j$-节点的子节点。）如果头谓词（分别地，命题符号）不是菱形节点的后代节点，则使用 $I_{(i,0)}$（分别地，$I^{\Pi'}_{(i,0)}$）作为替换用的输入门。替换所有头谓词和命题后，我们用一个 $\wedge$-门替换 $\Diamond_j$-节点，将每个菱形节点的父节点直接连接到其子节点，并将 $I_{(0,j)}$ 也连接到这个替换用的 $\wedge$-门。现在，如果 $I_{(0,j)}$ 的输入为 0，那么替换用的 $\wedge$-门也会输出 0。至此，用于 $\psi_{k'}$ 的电路 $C_{(k',1)}$ 已构造完成。输出是扇出为零的门。为了构造 C_1，我们将所有 k' 对应的电路 $C_{(k',1)}$ 合并，并确保对于所有 i 和 j，重复出现的输入门 $I_{(i,j)}$ 合并为一个输入门。同样，通过合并输入门来去除重复的 $I^{\Pi}_{(i,j)}$。$C_{(j,1)}$ 的输出门是电路 $C_{(k',1)}$ 的输出门，排列顺序应使得当 $k'' < k'''$ 时，$C_{(k'',1)}$ 的输出排在 $C_{(k''',1)}$ 的输出之前。

电路 C_0 现在模拟程序的初始化计算轮次，而 C_1模拟其余轮次。接下来，我们将把 C_0 和 C_1 合并为一个电路 C_Γ，其中 C_0 计算第一轮，C_1 计算之后的所有轮次。这可以通过对 C_0 和 C_1应用引理 4.6 来构造。电路 C_Γ 使用输入门 $I^{\Pi}_{(i,0)}$、$I^{\Pi'}_{(i',j)}$、$I_{(i'',j)}$ 以及一个由引理 4.6 添加的输入门（总计有 $|\Pi|+(|\Pi'|+1+k)(\Delta+1)$ 个输入门）。这些输入门的顺序显而易见。注意位和打印位分别是 Γ中注意谓词和打印谓词对应的索引。现在，C_Γ 会从其邻居那里接收到非零消息，因为引理 4.6 在该电路的输出前添加了一个 1。这确保了如果不存在第 j个邻居，那么替代 $\Diamond_j$节点的 $\wedge$-门会按预期输出 0。此外，正如我们在证明开头提到的，我们有一些属于 Π' 的比特，它们在每个求值轮次中都应保持不变。这些比特的输入门记为 $I^{\Pi'}_{(i,0)}$。此时，我们尚未为属于 Π'的比特定义相应的输出门，因此我们将按如下方式处理 C_Γ。对于每个输入门 $I^{\Pi'}_{(i,0)}$，都可以轻易构造一个恒等电路，将输入作为 $I^{\Pi'}_{(i,0)}$ 的输出。这些恒等电路的输出门按显而易见的顺序添加到 C_Γ 的输出之前。

电路 C_Γ 已构造出来，用于模拟程序 Γ；通过直接但繁琐的归纳法可以证明，该电路与程序强等价。由于电路的构造基于对程序子句的逆树表示进行简单修改，因此很容易看出，电路的大小为 $\mathcal{O}(\Delta m + |\Pi|)$。

□

5 将标准 MSC 与消息传递电路相联系

5.1 在程序中使用时钟

我们的下一个目标是定义一个程序，用来模拟一个带有分针和秒针的时钟。分针程序处理长度为 ℓ 的二进制字符串，而秒针则协助更新分针。分针从字符串 0^ℓ 开始，并按规范字典序遍历所有 ℓ 位字符串。要更新分针，我们使用秒针，如下所示：从右侧开始寻找字符串中第一个零。我们将这个零——或它在字符串中的位置——称为翻转点。翻转点左侧的比特保持不变，其余比特（包括翻转点）都翻转。例如，111011 从右数第三个比特是翻转点，字符串更新为 111100。到达字符串 1^ℓ 后，分针从 0^ℓ 重新开始。

我们将使用模式变量 $M_1, \ldots, M_\ell$ 来编码分针的各位比特，其中M_1 记录最右位， M_2 记录从右数第二位，以此类推。秒针使用头谓词 $S_1, \ldots, S_\ell$ 和 S_{changing}来定义。考虑一个示例（见图），其中 $\ell = 4$。最左列给出计算轮次，接下来是分针的位串（M_4, M_3, M_2, M_1）；中间是秒针的字符串（S_4, S_3, S_2, S_1）；最右列给出变量 S_{changing}的值。

	M_4	M_3	M_2	M_1	S_4	S_3	S_2	S_1	S_{changing}
0.	0	0	0	0	0	0	0	0	0
1.	0	0	0	1	0	0	0	0	1
2.	0	0	0	1	0	0	0	1	1
3.	0	0	0	1	0	0	0	1	0
4.	0	0	1	0	0	0	0	0	1
5.	0	0	1	0	0	0	0	0	0
6.	0	0	1	1	0	0	0	0	1
7.	0	0	1	1	0	0	0	1	1
8.	0	0	1	1	0	0	1	1	1
9.	0	0	1	1	0	0	1	1	0
10.	0	1	0	0	0	0	0	0	1
⋮			⋮				⋮		⋮
k.	0	1	1	1	0	0	0	0	1
$k+1$.	0	1	1	1	0	0	0	1	1
$k+2$.	0	1	1	1	0	0	1	1	1
$k+3$.	0	1	1	1	0	1	1	1	1
$k+4$.	0	1	1	1	0	1	1	1	0
$k+5$.	1	0	0	0	0	0	0	0	1
⋮			⋮				⋮		⋮

现在，第 0 行是一个特殊情况，因为第 0 轮是初始化轮。此后，计算以垂直轮次块的形式进行，各块之间由虚线分隔：首先是第 1–3 轮的块，然后是第 4–5 轮的块，再然后是第 6–9 轮的块，依此类推。这些块的高度各不相同。(M_4, M_3, M_2, M_1) 对应的字符串表示分针。我们注意到，在每个块内，分针保持不变；切换到下一个块时，分针则按标准字典序方式递增。变量 S_{changing} 在每个块内都为 1，但块的最后几行除外（图中已突出显示）。因此，S_{changing} 表示我们应该开始一个新块。(S_4, S_3, S_2, S_1) 对应的字符串始终属于 0^i1^j 类型，并且其中比特 1 的数量逐渐增加，直到到达倒数第二行。这体现了这样一个想法：(S_4, S_3, S_2, S_1) 从右向左复制 (M_4, M_3, M_2, M_1) 对应的字符串，直到到达翻转点，也就是 (M_4, M_3, M_2, M_1) 对应的字符串中从右数第一个 0。

现在，很容易将这种情况从 $\ell = 4$ 推广到一般情况。我们还需要一个始终比基本时钟快一步的时钟。我们将这个前向时钟 定义如下。对于基本时钟的每个头谓词 M_i, S_i, S_{changing}，我们定义一个新符号$M'_i, S'_i, S'_{\text{changing}}$。前向时钟只使用这些新符号。迭代子句通过复制相应的规则并将谓词替换为新符号来获得。前向时钟的终止子句也通过复制基本时钟的终止子句以类似方式获得，但 M'_1 和 S'_{changing} 对应的子句除外，我们将它们设为 $M'_1(0) :- \top$ 和 $S'_{\text{changing}}(0) :- \top$。这意味着前向时钟在其他方面与基本时钟相同，但它从字符串 $0^{\ell-1}1$ 开始，而不是从 0^{ℓ} 开始。注意，前向时钟开始时 S_{changing} 为真，否则在第 1 轮中，前向时钟的分针对应的字符串将是 0^{ℓ}。类似地，也可以定义一个以 $0^{\ell-2}10$开头的双重前向时钟 。

现在我们正式定义这个程序。令 Π为命题的有限集合，且 $\ell := |\Pi_1|$。对于每个 $i \in [\ell]$，令 $S_i(0) :- \bot$。此外，令 $S_1 :-_{S_{\text{changing}}} M_1; \bot$ 和 $S_i :^{\frown}_{S_{\text{changing}}} S_{i-1} \wedge M_i; \bot$，其中 $i \in \{2, \ldots, \ell\}$。为定义 S_{changing}的规则，请注意， S_{changing} 的作用是在 $(S_\ell, \ldots, S_1)$ 的值重复时（即与上一轮对应的字符串相同），恰好在这些轮次变为假。由于头谓词 S_i（对于 $i \neq 1$）按照规则主体 $S_{i-1} \wedge M_i$（或者对于 $i = 1$，按照 M_1）进行更新，因此我们可以通过确保对于某个 $i \neq 1$， S_i 和 $S_{i-1} \wedge M_i$ 不等价，或者确保 M_1 和 S_1 不等价，来保证 $(S_\ell, \ldots, S_1)$ 当前轮和上一轮的字符串不同。这可以通过迭代规则强制实现

$$S_{\text{changing}} :-_{S_{\text{changing}}} \neg(S_1 \leftrightarrow M_1) \vee \bigvee_{1<i\leq\ell} \neg\left(S_i \leftrightarrow (S_{i-1} \wedge M_i)\right); \top.$$

对应的终止规则为 $S_{\text{changing}}(0) :- \bot$。

然后，我们用头谓词 $M_1, \ldots, M_\ell$ 固定分针。对于所有 $i \in [\ell]$，令 $M_i(0) :- \bot$。我们还令 $M_1 :-_{S_{\text{changing}}} M_1; \neg M_1$，并且对于 $i \in \{2, \ldots, \ell\}$，定义$M_i :-_{S_{\text{changing}}} M_i; \psi_{\text{change } i}$，其中

$$\psi_{\text{change } i} := (S_i \wedge \neg M_i) \vee (S_{i-1} \wedge \neg S_i \wedge \neg M_i) \vee (\neg S_{i-1} \wedge \neg S_i \wedge M_i).$$

$\psi_{\text{change } i}$ 的第一个析取项处理翻转点右侧的值；第二个析取项将翻转点改为一；最后一个析取项则保持翻转点左侧的比特不变。

现在，我们已经为 $(M_\ell, \ldots, M_1)$ 中位串的循环定义了一个时钟。相对于谓词 M_i，该时钟的大小是线性的。

5.2 模拟多模态菱形算子

要在 MSC 中模拟 MPMSC（或 MMSC），我们需要仅用 $\Diamond$ 来模拟每个 $\Diamond_i$ 。为此，我们将使用标识符和时钟。其思路是按照标识符给定的顺序，逐个扫描邻居。为了使转换后的大小保持线性，不同的菱形算子 $\Diamond_i$ 将在不同轮次中被“读取”。我们注意到，可以加快读取菱形算子的速度，特别是在常数出度模型中，但下文的方法已足以满足当前研究的需要。我们先通过一个示例来说明模拟的直观思路。在该示例中，我们假设 $\Pi_1 = \{p_1, p_2, p_3\}$ 和 $\ell = |\Pi_1| = 3$，并考虑一个程序，其中菱形算子的最大子索引为 $I = 3$。来看一个有三个邻居的节点，其标识符分别为 000、010 和 111。考虑以下数组。

	X_{ID}			M_3	M_2	M_1	M'_3	M'_2	M'_1	X_{reset}	$X_{\text{not same}}$	N_1	N_2	N_3
0.	$\neg p_3$	$\neg p_2$	$\neg p_1$	0	0	0	0	0	1	0	1	0	0	0
1.	$\neg p_3$	$\neg p_2$	p_1	0	0	1	0	0	1	0	1	1	0	0
2.	$\neg p_3$	$\neg p_2$	p_1	0	0	1	0	0	1	0	0	1	0	0
3.	$\neg p_3$	$\neg p_2$	p_1	0	0	1	0	1	0	0	0	1	0	0
4.	$\neg p_3$	p_2	$\neg p_1$	0	1	0	0	1	0	0	1	1	0	0
5.	$\neg p_3$	p_2	$\neg p_1$	0	1	0	0	1	1	0	0	1	1	0
6.	$\neg p_3$	p_2	p_1	0	1	1	0	1	1	0	1	1	1	0
	⋮				⋮			⋮		⋮	⋮		⋮	
k.	p_3	p_2	$\neg p_1$	1	1	0	1	1	1	0	0	1	1	0
$k+1$.	p_3	p_2	p_1	1	1	1	1	1	1	0	1	1	1	0
$k+2$.	p_3	p_2	p_1	1	1	1	1	1	1	0	0	1	1	1
$k+3$.	p_3	p_2	p_1	1	1	1	1	1	1	0	0	1	1	1
$k+4$.	p_3	p_2	p_1	1	1	1	1	1	1	0	0	1	1	1
$k+5$.	p_3	p_2	p_1	1	1	1	0	0	0	1	0	1	1	1
$k+6$.	$\neg p_3$	$\neg p_2$	$\neg p_1$	0	0	0	0	0	0	0	1	0	0	0
$k+7$.	$\neg p_3$	$\neg p_2$	$\neg p_1$	0	0	0	0	0	1	0	0	1	0	0
$k+8$.	$\neg p_3$	$\neg p_2$	p_1	0	0	1	0	0	1	0	1	1	0	0
	⋮				⋮			⋮		⋮	⋮		⋮	

虚线为时钟 (M_3, M_2, M_1) 划分出若干块，使得位串同样从一个块到下一个块递增。这里只显示分针谓词。元组 (M'_3, M'_2, M'_1) 编码了一个领先一步运行的前向时钟。直观来说，X_{reset} 是一个标志，用于指示何时将标志 N_i（稍后解释）重置为零。形式上，X_{reset} 当且仅当 $(M_3, M_2, M_1) = (1, 1, 1)$ 且 $(M'_3, M'_2, M'_1) = (0, 0, 0)$ 时为真；这种情况出现在基本时钟对应字符串 111 的最后一轮。标志 $X_{\text{not same}}$ 在基本时钟和前向时钟至少有一位不同时，恰好在每轮之后为真，也就是说，(M_3, M_2, M_1) 和 (M'_3, M'_2, M'_1) 对应的字符串不同。（为技术上的便利，$X_{\text{not same}} = 1$ 在第 0 轮为真。）X_{not} 本身的用途是帮助我们以如下方式处理谓词 N_i。首先，我们恰好有三个谓词 N_1、N_2、N_3，因为菱形算子的最大子索引为 $I = 3$。谓词 N_i 在满足以下条件的轮次 n 变为真：字符串 (M_3, M_2, M_1) 恰好在第 i 个相邻节点的第 $n-1$ 轮与其标识符匹配。例如，我们的第二个邻居的标识符为 010，而在第 4 轮，$(M_3, M_2, M_1) = (0, 1, 0)$，因此 N_2 在第 5 轮变为真。谓词 N_i 会一直保持为真，直到在紧接着 X_{reset} 为真的那一轮之后的轮次中被重置。谓词 X_{ID} 恰好在 (M_3, M_2, M_1) 编码当前节点标识符的轮次为真；也就是说，如果该标识符为 $b_3b_2b_1 \in \{0,1\}^*$，则有 $(M_3, M_2, M_1) = (b_3, b_2, b_1)$，这意味着 p_3、p_2、p_1 的真值分别为 b_3、b_2、b_1。图中的前三列编码了这些向量

$$(\varphi_3, \varphi_2, \varphi_1) \in \{p_3, \neg p_3\} \times \{p_2, \neg p_2\} \times \{p_1, \neg p_1\}$$

满足以下性质：在第 n 轮中，变量 X_{ID} 等价于合取 $\varphi_3 \wedge \varphi_2 \wedge \varphi_1$。例如，在第六轮中，$(M_3, M_2, M_1) = (0, 1, 1)$，而 $(\varphi_3, \varphi_2, \varphi_1) = (\neg p_3, p_2, p_1)$。使用 $\Diamond$ 模拟菱形算子 $\Diamond_i$ 的一个关键直觉，是通过 X_{ID} 在邻居节点处不断变化的真值，来扫描其标识符；这些节点是邻居节点。

对于固定的 Π_1 和 $I \in \mathbb{Z}_+$，我们在一般情形下对程序作形式化说明。令 $\ell = |\Pi_1|$。我们为头谓词 $(M_\ell, \ldots, M_1)$ 和 $(N_I, \ldots, N_1)$ 构造一个程序。假设 $p_1, \ldots, p_\ell$ 按顺序 $<^{\mathrm{PROP}}$ 枚举了 Π_1 中的命题。我们用头谓词的元组 $(M_\ell, \ldots, M_1)$ 来定义时钟，用 $(M'_\ell, \ldots, M'_1)$ 来定义前向时钟。出于技术上的便利，我们甚至用 $(M''_\ell, \ldots, M''_1)$ 来定义双重前向时钟。头谓词 X_{reset} 的程序为 $X_{\mathrm{reset}}(0) \coloneq \bot$，$X_{\mathrm{reset}} \coloneq \bigwedge_{i \le \ell} \neg M''_i \wedge \bigwedge_{i \le \ell} M'_i$. 该

头谓词 $X_{\text{not 相同}}$的程序为 $X_{\text{not 相同}}(0) :- \top$， $X_{\text{not 相同}} :- \bigvee_{i\le\ell} \neg(M_i \leftrightarrow M_i')$。 X_{ID}的程序为 $X_{\text{ID}}(0) :- \bigwedge_{i\le\ell} \neg p_i$, $X_{\text{ID}} :- \bigwedge_{i\le\ell}(M_i' \leftrightarrow p_i)$。

对于每个 $i \in [I]$， N_i 的规则为 $N_i(0) :- \bot$ 和

$$N_i :-_{X_{\text{reset}},\ \neg N_i \wedge N_{i-1} \wedge X_{\text{not same}}} \bot; \Diamond X_{\text{ID}}; N_i$$

其中，当 $i-1=0$时， N_{i-1} 会被直接删除。第一个条件 X_{reset} 显然成立。第二个条件 $\neg N_i \wedge N_{i-1} \wedge X_{\text{not 相同}}$ 会检查我们是否已经找到邻居$i-1$，即 N_{i-1}是否为真（注意，变量 N_0不存在，因此无需对 N_1进行此类检查）。此外，该条件还会检查我们是否尚未找到邻居 i，也就是说，N_i 为假，并且时钟已在上一轮次更新，即 $X_{\text{not same}}$为真。这确保在一个块内，恰好只有一个标志 N_i 可以变为真。

菱形 $\Diamond_i$ 宏 关于 φ 且相对于 $(N_i, N_{i-1}, X_{\text{ID}})$ 的模式为 $\Diamond_i\varphi := (\neg N_i \wedge N_{i-1} \wedge \Diamond(\varphi \wedge X_{\text{ID}}))$。注意，在该 $\Diamond_1$ 宏中，省略了 N_0（这意味着上文中的N_{i-1} 被删除）。非正式地说， $\Diamond_i$ 宏只在计算的特定时刻才“可靠”，因此必须谨慎使用。附录中给出的以下引理的证明展示了这一点。此外，后续结果（定理 5.2）可由引理 4.3 立即推出。

引理 5.1. 给定 Π以及一个大小为 m 的 Π-程序（MPMSC），其中菱形的最大子索引为 I，我们可以构造一个等价的、大小为 $\mathcal{O}(I+|\Pi_1|+m)$ 的 Π-程序（CMSC）。计算时间为 $(2^{|\Pi_1|})$ 倍的 MPMSC 程序计算时间的 $\mathcal{O}$。

定理 5.2. 给定 Π 以及一个大小为 m 的 Π-程序（MPMSC），其中菱形中的最大子索引为 I，我们可以构造一个等价的、大小为 $\mathcal{O}(I+|\Pi_1|+m)$ 的 Π-程序（MSC）。计算时间为) 倍的 MPMSC 程序计算时间的 $2^{\mathcal{O}(|\Pi_1|}$。

5.3 MSC 的 A范式

MSC[1] 程序是指终止子句的模态深度为零（迭代子句的模态深度至多为一）的 MSC 程序。我们从以下引理开始。该证明见附录，其思路是：(1) 通过适当使用条件 MSC，将终止子句纳入迭代子句；(2) 将 CMSC 转换为 MSC。

引理 5.3。对于每个 Π-程序 Λ（属于 MSC），都存在一个等价的 Π-程序（属于MSC），其终止子句的模态深度为零。该程序的大小与 $|\Lambda|$ 成线性关系。其计算时间与 Λ 的计算时间成线性关系。

接下来，我们证明可以将迭代子句的模态深度降至一。证明见附录，其依据是一个简单策略：为原程序中每个类型为 $\Diamond\psi$ 的子模式使用新的头谓词$X_{\Diamond\psi}$。运用这一策略，并结合引理 5.3，我们可以证明以下范式结果。

定理 5.4。对于每个 Π-程序 Λ（属于 MSC），都存在一个等价的 Π-程序（属于MSC）[1]。该MSC[1]-程序的大小为 $\mathcal{O}(|\Lambda|)$，且该程序的计算时间为 $\mathcal{O}(max(1,\text{md}(\Lambda)))$ 倍的 Λ 计算时间。

5.4 等价性与时间损耗

现在我们可以将 MSC 与 MPC 联系起来。在第 4.1 节中，我们证明了定理 4.2，说明可以将 MPC 转换为通信等价的 MPMSC 程序，且其大小与 MPC 的大小呈线性关系。另一方面，定理 5.2 表明，我们可以将任意 MPMSC 程序转换为等价的 MSC 程序。因此，我们得到如下结论。

定理 5.5. 给定一个MPC ，其参数为(Π, Δ)，我们可以构造一个等价的 Π 程序，其计算模型为MSC。对于MPC中扇入的常数上界 c ，该程序的大小与电路的大小呈线性关系。其计算时间为$(d + 2^{|\Pi_1|})$ 倍MPC的计算时间，其中 d 是MPC的深度。

定理 4.7 表明，我们可以将 MPMSC 程序转换为强等价的 MPC。定理 5.4 展示了如何将 MSC-程序转换为强等价的 MSC[1] 程序，这意味着，将 MSC-程序转换为 MPMSC 程序时，可以做到不会使程序大小过度膨胀。这些结果推出了如下推论。

定理 5.6。 给定 Π、 Δ 以及一个大小为 m 的 MSC Π-程序，存在一个对于 (Π, Δ) 等价的、大小为 $\mathcal{O}(\Delta m + |\Pi|)$ 的MPC 。其计算时间是 $(\max(1, d))$ 倍于该MSC-程序的计算时间，其中 d 是该MSC-程序的模态深度。

根据上述结果，我们注意到，$\mathsf{DCC}^{c}_{\Delta}[\log n]$ 中的问题也可以用 MSC-程序序列来描述。

最后，我们注意到，定理 4.2 也是我们的主要结果之一。它提醒我们，通信时间确实是一个不同于计算时间的概念。

我们注意到，除了标准的分布式计算之外，还可以使用 MSC 进行建模，例如神经网络。实数可以通过使用头谓词来编码浮点数，自然地进行建模。标识符（以及一些更轻量的方案）可用于追踪消息的发送来源，而偏置则可以自然地编码到命题符号中。激活函数可以进行近似，当然，标准的算术计算也可以。我们将此留待未来工作。

6 基于 Cole & Vishkin 的着色

6.1 Cole-Vishkin 算法概要

一个**图** 是一个有序对 $G = (N, E)$，其中 N 是一个**节点**集合，而 E 是一个**边** 集合$(v, u) \in N \times N$。边关系是对称的（如果 $(v, u) \in E$，那么 $(u, v) \in E$），且是自反的（对于所有 $v \in N$，$(v, v) \notin E$）。如果 $(v, u) \in E$，我们就称 v 和 u 互为**邻居** 。

图 G 的**定向** 是一个有序对 (N, E')，其中对于每一对 $(v, u) \in E$，要么 $(v, u) \in E'$，要么 $(u, v) \in E'$，但不能两者兼有。如果 $(v, u) \in E'$，则我们称 u 为v 的**父节点** ，称 v 为 u的**子节点** 。如果 $v_1 = v_k$且 $(v_1, v_2) \ldots, (v_{k-1}, v_k) \in E'$，则称元组 $(v_1, \ldots, v_k)$，$k > 1$为一个**有向环** 。

图 G 的**森林分解** 是一个有序对集合 $\mathcal{F}$，其中的有序对为 $F_i = (N, E_i)$，且存在 G的一个定向 (N, E')，使其不含有向环，其中 $\bigcup E_i = E'$，并且对于所有 $i \neq j$， $E_i \cap E_j = \emptyset$ 。我们称每个 $F_i \in \mathcal{F}$ 为一个**定向森林** （ G的）。

我们记 $\log(k) := \lfloor \log_2(k) \rfloor + 1$，因此 $\log(k)$ 是 k 的二进制位数。考虑一个图 G，其节点数为 n，节点的最大度为 Δ。我们介绍一种源自 [2]，第 36–37 页的颜色约简算法，其中每个节点一开始都拥有集合 $\{1, \ldots, n\}$ 中的一个唯一标识符（ID）；这就是它的初始颜色。该算法在 $3^{\Delta} + \log^*(n) + \mathcal{O}(1)$ 个通信轮次内，生成一个 $(\Delta + 1)$-着色。

利用标识符，每个节点首先定向其邻接边，使每条边都指向标识符更大的节点；由此得到图G的一个定向。此外，每个节点根据相连节点的标识符，按升序为自己的所有出边标记为 $1, \ldots, \Delta$。按照标签将边分成不同的集合后，我们得到 Δ 个定向森林的森林分解：$\mathcal{F} = (F_1, \ldots, F_\Delta)$。

在每个定向森林中，每个节点运行 Cole-Vishkin 算法 [4, 5]（或 CV 算法）。节点首先将自己的颜色与父节点的颜色进行比较。颜色按通常方式编码为二进制。它会找出与父节点颜色不同的最右位；假设这是第 i 位。如果该节点的第 i 位是 $b \in \{0,1\}$，那么其新颜色的第一位也为 b。新颜色的其余位是数字 i 的二进制表示中的各位。根节点没有父节点，因此改为将自己的颜色与颜色 0 进行比较。

重复运行该算法后，每个节点都能够减少其当前颜色在每个定向森林中的索引编号。如果某个森林中节点的最高颜色长度为 k，那么在下一轮通信后，新的颜色长度至多为 $\log(k)+1$。经过 $\log^*(n)+\mathcal{O}(1)$ 轮通信后，每个节点在每个森林中的颜色长度都至多为 3。由于未使用颜色 0，因此这在每个森林中都是一种 7-着色。

此后，只需经过 $\mathcal{O}(1)$ 轮通信，就可以用一种称为“shift-down”的技术，将每个 7-着色进一步归约为 3-着色。这些着色可以拼接成整个图的 3^Δ-着色，随后用一种称为“基本颜色归约”的技术，在少于 3^Δ 轮通信内将其归约为 $(\Delta+1)$-着色。

下一节将概述一个 MPMSC 程序，用于模拟该着色算法的开头部分（完整细节见附录）。定义此程序时，打印谓词集、注意谓词集以及指定谓词集均相同。我们的记法与之前略有不同。此前，头谓词用带索引的单个斜体字母表示（例如 X_i）。在接下来的程序中，头谓词不再用斜体书写，并且可以包含多个符号（例如 $\mathrm{HP1}_i$）。为方便起见，公式也可以采用相同的规则；某个记号表示变量还是公式，会在引入时予以说明。

定义 6.1。设 Λ 是一个 MPMSC 的 (Π,Δ)-程序，其中对于每个克里普克模型 $(W,R,V) \in \mathcal{K}(\Pi,\Delta)$，每个节点 $w \in W$ 都会在某一轮次 $n \in \mathbb{N}$ 输出 $\overline{p}_w$。如果对于每个克里普克模型 $(W,R,V) \in \mathcal{K}(\Pi,\Delta)$ 中的每个节点 $w \in W$，都存在某个 $a \in [k-1]_0$ 和 $b \in \mathbb{N}$ 使得 $\overline{p}_w = 0^a 1 0^b$，我们就说 **$\Lambda$定义了一个 k-着色**。如果在每个克里普克模型 $(W,R,V) \in \mathcal{K}(\Pi,\Delta)$ 中，对每个 $(w,v) \in R$，都有$\overline{p}_w \neq \overline{p}_v$，则这种着色是**适当的**。

6.2 用程序模拟 CV 算法

设 $G=(N,E)$ 是一个具有唯一标识符的图，其节点数为 n，节点的最大度为 Δ。其**对应的**带有标识符的克里普克模型在 $\Pi=\{p_1,\dots,p_{\log(n)}\}$ 上为模型 (W,R,V)，其中 $W=N$、$R=E$，并且当且仅当 v 的标识符的第 i 位比特为 1 时，$v \in V(p_i)$。注意，$(W,R,V) \in \mathcal{K}(\Pi,\Delta)$。

程序首先模拟将图分解为定向森林的过程。为此，每个节点都需要知道自己的哪些邻居具有比自己更大的标识符。这通过逐个比较标识符的比特来完成，需要 $\ell=\log(n)$ 次迭代。我们用变量 T_1、……、$\mathrm{T}_{\ell+2}$来统计这些迭代。每个变量初始值均为 false。变量 T_1 在第一次迭代后永久变为 true，T_2 在第二次迭代后变为 true，依此类推。

为了与邻居比较标识符，我们为节点标识符的各个比特定义变量 I_1、……、I_ℓ，并为其第 δ 个邻居的标识符比特定义变量 I_1^δ、……、$\mathrm{I}_\ell^\delta$$(1 \le \delta \le \Delta)$。在终止子句中，每个 I_i 都取命题 p_i 的真值。第一次迭代中，变量 I_i 保持不变，而变量 I_i^δ 取公式 $\Diamond_\delta \mathrm{I}_i$ 的真值。在接下来的 ℓ 次迭代中，变量轮流传递真值，即 $\mathrm{I}_i^{(\delta)}$ 取 $\mathrm{I}_{i-1}^{(\delta)}$ 的真值。

经过一轮迭代后，变量 $\mathrm{I}_\ell^{(\delta)}$ 中存储的是节点的（第 δ 个邻居的）标识符最左侧的比特。再经过一轮迭代后，该变量存储第二个比特，以此类推。我们通过找出区分两个标识符的第一个比特，来判断节点的标识符是否大于其邻居的标识符；比特为 1 的节点具有更大的标识符。我们定义变量 DIF^δ $(1 \le \delta \le \Delta)$，用于

比较变量 I_ℓ 和 I_ℓ^δ 中的比特；如果真值（即比特）不同，DIF^δ 就为真。

最后，我们定义变量 HIGH^δ 和 LOW^δ $(1 \le \delta \le \Delta)$。$\mathrm{DIF}^\delta$ 变为真之后，如果某个节点的第 δ 个邻居的标识符比它更大，则 HIGH^δ 变为真；否则 LOW^δ 变为真。经过 $\ell+2$ 次迭代后，每个节点中的每个 δ 要么满足 HIGH^δ，要么满足 LOW^δ。

现在可以定义一个长度为 $\mathcal{O}(\Delta)$ 的公式 $\langle\delta\rangle\varphi$：当且仅当节点 v 的父节点（位于定向森林 F_δ 中）满足 φ 时，该公式在节点 v 中为真。该公式猜测索引 i（如果存在），使得 $\Diamond_i\varphi$ 为真，并且 i 是 LOW^i 为真的第 δ 个索引。

接下来，我们扩展程序，以便在每个定向森林中模拟 CV 算法。我们逐比特比较节点的颜色，并在颜色比较之间安排通信轮次。为此，我们使用第 5.1 节中的时钟及其相关变量。时钟的字符串包含 $\log(\ell) = \log\log(n)$ 个比特。在时钟周期结束时的一个唯一时刻将用于触发程序此阶段中的全局通信轮次。

为了比较节点与邻居的颜色，我们定义变量 B_i^δ $(1 \le \delta \le \Delta)$，表示森林 F_δ 中节点颜色各比特的值；并定义变量 P_i^δ $(1 \le \delta \le \Delta)$，表示其父节点颜色各比特的值，初始值取自标识符。每个变量 B_i^δ 和 P_i^δ 都会循环遍历森林F_δ中颜色的各个比特，也就是说，B_i^δ 接收 B_{i+1}^δ的真值，而 P_i^δ 接收 P_{i+1}^δ的真值。变量每隔一个“分钟”旋转一次。分针始终指示当前 B_1^δ 和 P_1^δ 中各比特在二进制表示中的原始位置。在全局通信轮次中，变量 B_i^δ 更新为下一个颜色 N_i^δ（见下文），变量 P_i^δ 则根据公式 $\langle\delta\rangle\mathrm{N}_i^\delta$更新为父节点的下一个颜色。对于根节点，这些公式始终为假，这意味着每个根节点都会将自己的颜色与颜色 0 进行比较，正如第 6.1 节所述。

我们使用与之前相同的变量 DIF^δ，比较变量 B_1^δ 和 P_1^δ 中子节点与父节点颜色的比特。最后，我们定义变量 N_i^δ $(1 \le \delta \le \Delta)$，用于计算每个时钟轮次中节点的下一个颜色。当 DIF^δ 变为真时，节点的不同位会从 B_1^δ 复制到 N_1^δ，其位置的比特则从分针变量 M_i 复制到变量 N_{i+1}^δ。这些值会一直保留到下一个全局通信轮次，届时它们将用于更新变量 B_i^δ 和 P_i^δ。

CV 算法需要 $L = \log^*(n) + 3$ 个通信轮次。我们用由变量 H_1、…、H_L 组成的时针来计数；经过 $\log^*(n)+2$ 个“小时”（即若干时钟周期）后，H_L 变为真；再过一个“小时”，所有变量的迭代便会停止。

对于引理 6.2，我们定义与每个节点颜色对应的 7^Δ “指定”谓词：将 Δ 森林中的颜色按顺序拼接时，把所有比特排成一行。现在，以下引理成立；将程序扩展以采用 shift-down 和基本颜色约简技术后，接下来的定理也成立（证明见附录）：

引理 6.2。存在一个 MPMSC 公式，具有以下性质。

1. 它定义了一个合法的 7^Δ着色。*2.* 忽略指定谓词后，它有 $\mathcal{O}(\Delta\log(n))$ 个头部，大小为 $\mathcal{O}(\Delta^2\log(n))$。*3.* 所需的迭代次数为 $\mathcal{O}(\log(n)\log\log(n)\log^*(n))$。*4.* 全局通信轮次恰好为 $\log^*(n)+4$。

定理 6.3。存在一个 MPMSC 公式，具有以下性质。

1. 它定义了一个合法的（$\Delta+1$）着色。*2.* 它有 $\mathcal{O}$（$\Delta\log$（n））$+\mathcal{O}$（3^Δ）个头部，且其大小为 $\mathcal{O}$（$\Delta^2\log$（n））$+\mathcal{O}$（3^Δ）。

3. 所需的迭代次数为 $\mathcal{O}$（log（n） log log（n） log*（n））$+\mathcal{O}$（$\Delta 3^{\Delta}$）。

4. 全局通信轮次恰好为 $\log^*(n) + 3^{\Delta} - \Delta + 11$。

程序的全局通信轮次与算法的通信轮次相同；如果 Δ 是常数，则程序的大小为 $\mathcal{O}(\log(n))$。

致谢。 Veeti Ahvonen 获得了芬兰科学与文学学院 Vilho、Yrjö 和 Kalle Väisälä 基金会的资助。Antti Kuusisto 还获得了芬兰科学院“计算逻辑理论”项目的资助，项目资助编号为 324435、328987、352419、352420、352419、353027。Antti Kuusisto 还获得了芬兰科学院“通过逻辑解释人工智能”（XAILOG）项目的资助，项目资助编号为 345612。

参考文献

[1] Pablo Barceló、Egor V. Kostylev、Mikaël Monet、Jorge Pérez、Juan L. Reutter 和 Juan Pablo Silva。图神经网络的逻辑表达能力。载于第八届国际学习表征会议，*ICLR 2020*，埃塞俄比亚亚的斯亚贝巴，*2020*年*4*月*26*日至*30*日。OpenReview.net，2020年。[2] Leonid Barenboim 和 Michael Elkin。分布式图着色。分布式计算理论综合讲座，第11卷，2013年。[3] Benedikt Bollig、Patricia Bouyer 和 Fabian Reiter。寄存器中的标识符——用逻辑描述网络算法。*CoRR*，abs/1811.08197，2018年。[4] Richard Cole 和 Uzi Vishkin。具有最优并行链表排名应用的确定性抛硬币。信息与控制，70(1)：32–53，1986年。[5] Andrew Goldberg、Serge Plotkin 和 Gregory Shannon。稀疏图中的并行对称性破缺。载于第十九届 *ACM* 计算理论年度研讨会论文集，第315–324页，1987年。[6] Martin Grohe。图神经网络的逻辑。载于第*36*届 *ACM/IEEE* 计算机科学中的逻辑年度研讨会，*LICS 2021*，意大利罗马，*2021* 年 *6* 月 *29* 日至*7*月*2*日，第1–17页。IEEE，2021年。[7] Lauri Hella、Matti Järvisalo、Antti Kuusisto、Juhana Laurinharju、Tuomo Lempiäinen、Kerkko Luosto、Jukka Suomela 和 Jonni Virtema。分布式计算的弱模型及其与模态逻辑的联系。分布式计算，28(1)：31–53，2015年。[8] Antti Kuusisto。模态逻辑与分布式消息传递自动机。载于计算机科学逻辑会议 *2013*（*CSL 2013*），第23卷，莱布尼茨国际信息学会议录（*LIPIcs*），第452–468页，2013年。[9] Tuomo Lempiäinen。分布式计算中的逻辑与复杂性。博士论文，阿尔托大学，2019年。[10] Leonid Libkin。有限模型论基础。《理论计算机科学丛书：EATCS系列》。施普林格，2004年。[11] Fabian Reiter。异步分布式自动机：模态 μ 片段的一种刻画。载于Ioannis Chatzigiannakis、Piotr Indyk、Fabian Kuhn 和 Anca Muscholl（编），第*44*届国际自动机、语言与编程研讨会，*ICALP 2017*，第80卷，*LIPIcs*，第100:1–100:14页。达格施图尔城堡—莱布尼茨信息学中心，2017年。

[12] Fabian Reiter。分布式自动机与逻辑。（分布式自动机与逻辑）。博士论文，索邦巴黎西岱大学，法国，2017年。

7 附录 A

引理 4.3 的证明：

证明。 设 Λ 是一个包含以下条件迭代子句的程序

$$\begin{aligned} X_1 &:-_{\varphi_{1,1},\ldots,\varphi_{1,m_1}} \psi_{1,1};\ldots;\psi_{1,m_1};\chi_1 \\ &\vdots \\ X_k &:-_{\varphi_{k,1},\ldots,\varphi_{k,m_k}} \psi_{k,1};\ldots;\psi_{k,m_k};\chi_k, \end{aligned}$$

其中，$\varphi_{i,\ j}$ 是第 i 条规则的第 j 个条件，而$\psi_{i,\ j}$ 是相应的结果。所有其余迭代子句均为非条件迭代子句。我们如下定义一个与之强等价的 MSC Π-程序 Λ' 。我们保留所有终止子句和非条件迭代子句，不作修改。条件迭代子句将被修改。指定谓词将按 Λ 中的方式选取。对于每个条件子句的头谓词 X_i ，新的迭代子句将是一个与该条件子句效果相同的非条件规则。通过递归指定模式 $\theta_{(i,1)}$、.....、$\theta_{(i,m_i)}$ ，为每个 X_i 定义该非条件规则。直观而言， $\theta_{(i,1)}$ 将涵盖这样一种情况： X_i的真值由备份或最后一个条件-结果对决定。模式 $\theta_{(i,2)}$ 将处理倒数第二个条件 - 结果对， $\theta_{(i,3)}$ 处理倒数第二个之前的条件 - 结果对，以此类推。最后一个模式 $\theta_{(i,m_i)}$ 处理第一个条件。

为了定义 $\theta_{(i,1)}$、.....、$\theta_{(i,m_i)}$，我们首先令

$$\theta_{(i,1)} \coloneqq (\varphi_{i,m_i} \wedge \psi_{i,m_i}) \vee (\neg\varphi_{i,m_i} \wedge \chi_i).$$

假设我们已经为 $j < m_i$ 定义了 $\theta_{(i,j)}$ ，令

$$\theta_{(i,j+1)} \coloneqq (\varphi_{i,m_i-j} \wedge \psi_{i,m_i-j}) \vee (\neg\varphi_{i,m_i-j} \wedge \theta_{(i,j)}).$$

头谓词 X_i 的迭代子句为 $X_i :- \theta_{(i,m_i)}$。

现在我们已经定义了 Λ'。不难证明， Λ' 与 Λ 强等价，并且其大小相对于 Λ 呈线性，同时终止子句和迭代子句的最大模态深度相同。

□

引理 5.1 的证明：

证明。为了正确模拟 MPMSC 程序，我们必须为一些子程序计时。直观而言，我们将为程序中出现的每个形如 $\Diamond_i\psi$ 的子模式定义一个头谓词，并在原程序中用相应的头谓词替换这些子模式。为这些新头谓词设定规则的计时方式，使得$\Diamond_i\psi$ 的规则恰好在扫描第 i 个邻居时激活。此外，我们调整所有相关规则，使其等待新规则扫描完所有可能的标识符，从而正确检查带有菱形算子的条件。

令 Π 为一个有限命题集， Λ 为 MPMSC 的 Π-程序，并回顾一下：终止子句的模态深度为零，而迭代子句的模态深度至多为一。假设出现在 Λ 中的菱形算子的最大子索引为 I。我们将从 Λ 构造一个大小为 $\mathcal{O}(I + |\Pi_1| + |\Lambda|)$ 的 CMSC 程序 Γ。首先，我们加入一份副本：

Λ，并将其置于 Γ 中。随后，按照第5.1节和第5.2节中的方案，我们添加一个用于模拟多模态菱形算子的程序，其头谓词为 $(M_{|\Pi_1|}, \ldots, M_1)$ 和 $(N_I, \ldots, N_1)$（请注意，我们甚至使用了与上文附录中相同的符号），并包含上述附录各节“时钟的形式化程序”和“模拟多模态菱形的形式化程序。”中的所有规则。与此同时，我们还按照第5.2节中的说明（更形式化地说，按照上文附录中的说明），将 MPMSC 的每个子模式$\Diamond_i\psi$ 替换为 $\Diamond_i\psi$ 宏。从此以后，所有带有 $\Diamond_i$ 的模式均指这些新宏。

我们按如下方式计时 Γ。首先，我们按照下文所述，将 Γ 的一些子模式分离到新的子程序中。对于每个 $i \in [I]$，令 $S_i \subseteq$ SUBS(Γ) 表示 SUBS(Γ) 中形如 $\Diamond_i\psi$ 的模式集合。注意，Γ 的终止子句的模态深度为零，因此 $\Diamond_i$ 只会出现在迭代子句中。我们令 $S_i := \{\Diamond_i\psi_1, \ldots, \Diamond_i\psi_{k_i}\}$，其中 $k_i := |S_i|$。对于每个 $i \in [I]$ 和 $j \in [k_i]$，我们向 Γ 添加一个新的头谓词 $X_{(i,j)}$，并加入以下规则

$$X_{(i,j)}(0) :- \bot \quad X_{(i,j)} :-_{\neg N_i} \Diamond_i\psi_j;\ X_{(i,j)},$$

其中 N_i 在第 5.2 节中有所描述（并在上文附录中给出了更形式化的说明）。$X_{(i,j)}$ 的规则会在每一轮“测试”模式 $\Diamond_i\psi_j$，直到邻居 i 被扫描的那一轮，并且（重要的是）包括该轮。起初，我们可能会得到关于 $X_{(i,j)}$ 的“错误”真值，但在实际扫描邻居 i 的那一轮，这些真值便会固定下来。

接下来，我们将程序 Γ 中的每个 $\Diamond_i\psi_j \in S_i$ 替换为相应的模式变量 $X_{(i,j)}$。现在，我们已经为每个 $\Diamond_i$ 计时，但仍然需要使用模拟多模态菱形的程序中的 X_{reset}，按照如下方式为程序 Γ 的其他部分计时。设 $X :- \psi$ 是 Γ 的一个非条件迭代子句，该子句最初取自 Λ，即该规则不属于时钟或其他辅助程序。我们通过向该子句添加一个新条件来处理它，使迭代子句 $X :- \psi$ 转换为 $X :-_{X_{\text{reset}}} \psi;\ X$。向该子句添加条件 X_{reset} 可以防止菱形 $\Diamond_i$ 宏被误用，因为该宏仅在特定时间可靠。现在，设$Y :-_{\varphi_1,\ldots,\varphi_k} \psi'_1;\ \ldots;\ \psi'_k;\ \chi$ 是 Γ 的一个条件迭代子句，该子句最初取自 Λ。我们类似地向其中添加一个新条件，使该规则转换为$Y :-_{\neg X_{\text{reset}},\ \varphi_1,\ldots,\varphi_k} Y;\ \psi'_1;\ \ldots;\ \psi'_k;\ \chi$. 由于条件 X_{reset} 恰好在基本时钟重置之前为真，我们可以确定，所有本应变为真的头谓词 N_i 确实已经在当前块或更早的某个块中变为真。注意，对于 $\Diamond_i$ 的菱形宏而言，在 N_i 变为真之前的任何块中，该宏都不可靠；此外，即使在 N_i 变为真的那个块中，第一轮也不可靠。

我们的 CMSC 程序现在已经准备就绪。其大小显然为 $\mathcal{O}(I + |\Pi_1| + |\Lambda|)$。下面简要说明程序 Γ如何模拟 HEAD(Λ) 中的头谓词。Γ 中 HEAD(Λ) 的终止子句与 Λ 中的保持不变，因此它们的起始轮次与 Λ 中的完全相同。接下来，我们描述 Λ 在第零轮之后的轮次如何由 Γ模拟，并指出，模拟单轮当然需要持续好几轮。在一个模拟周期中——即模拟 Λ 的一轮——时钟会遍历所有可能的标识符，并以 X_{reset} 为真作为该周期的结束。在周期内，我们为 Γ的头谓词 $X_{(i,j)}$ 所添加的迭代子句会配合时钟和其他辅助程序来模拟每个 $\Diamond_i$。当 $\Diamond_i$ 可靠时，新的头谓词 $X_{(i,j)}$ 便会激活，并存储 $\Diamond_i\psi_j$的真值。当时钟扫描完所有可能的标识符后，X_{reset} 变为真；随后，HEAD(Λ) 中 Γ 的每个原始头谓词都会激活，其运行方式与 Λ 中相应的规则完全相同。同时，标志 N_i 会重置。接着，一个新的周期便会开始。较容易更严格地证明，Γ 与 Λ在（接受性）方面等价。

每一轮中的“时间损失”可以从菱形算子$\Diamond_i$的时钟定义看出。延迟取决于标识符的长度，因为我们总是必须检查所有

标识符。模拟一轮 Λ 需要 $\mathcal{O}(2^{|\Pi_1|})$ 轮次。 □

引理 5.3 的证明：

证明。 设 Λ 是一个 MSC 的 Π-程序。如果 $\mathsf{mdt}(\Lambda) = 0$，则问题已解决，因此我们可以假设 $\mathsf{mdt}(\Lambda) > 0$。设 $X_1, \ldots, X_q$ 是 Λ的头谓词，$\varphi_1, \ldots, \varphi_q$ 是终止子句的主体，而 $\psi_1, \ldots, \psi_q$ 是迭代子句的主体。我们按如下方式定义一个条件式 MSC 程序Λ'。我们添加一个新的模式变量 I，使得 $I(0) :- \bot, I :- \top$，以指示当前计算的是迭代步骤还是终止步骤。我们将每个头谓词 X_i 替换为头谓词 X_{ψ_i}，并设定 $X_{\psi_i}(0) :- \bot$ 和 $X_{\psi_i} :-_I \psi_i; \varphi_i$。在 Λ' 中选择注意力谓词（分别为打印谓词），使得 X_{ψ_i} 是 Λ' 中的注意力谓词（分别为打印谓词），当且仅当 X_i 是 Λ 中的注意力谓词（分别为打印谓词）。

不难证明，对于每个指向的 Π-模型 (M, w)，$(M, w) \models X_{\psi_i}^{n+1} \iff (M, w) \models X_i^n$ 均成立，其中 $n \geq 0$ 和 $i = 1, \ldots, q$。此外，$|\Lambda'| = \mathcal{O}(|\Lambda|)$。$\Lambda'$ 的计算时间与 Λ 的计算时间呈线性关系。

□

定理 5.4 的证明：

证明。 该变换基于用新的模式变量为 MSC-程序的子模式标记。首先，我们简要地非正式描述证明思路。我们定义一个时钟，使得我们可以按程序求值步骤的周期进行处理，每个周期的长度等于原程序的最大模态深度。新程序逐步模拟原程序，使用模态深度不断增加的求值公式，直到达到最大模态深度。

设 Γ 是一个 MSC 的 Π-程序。根据引理 5.3，存在一个等价的 MSC 的 Π-程序Λ，其 $\mathsf{mdt}(\Lambda) = 0$。如果 $\mathsf{mdi}(\Lambda) \leq 1$，则结论成立，因此我们假设 $\mathsf{mdi}(\Lambda) > 1$。设 X_1、……、X_q 是 Λ 的头谓词，设 φ_1、……、φ_q（分别为 ψ_1、……、ψ_q）是对应终止（分别为迭代）子句的主体。设 $S_\Diamond$ 是模式 ψ_1、……、ψ_q 中形如 $\Diamond\psi$ 的子模式集合。设 $S_i = \{\, \Diamond\psi \in S_\Diamond \mid \mathsf{md}(\Diamond\psi) = i \,\}$ 对每个 $i = 1$、……、$\mathsf{mdi}(\Lambda) - 1$ 都成立，也就是说，我们按模态深度将 $S_\Diamond$ 划分为若干子集。令$k_i := |S_i|$，并令 $\theta_{(i,1)}$、……、$\theta_{(i,k_i)}$ 表示 S_i 中的模式，也就是说，现在 $S_i = \{\theta_{(i,1)}, \ldots, \theta_{(i,k_i)}\}$。现在我们可以将 Λ 转换为所需的程序 Λ'。

我们首先定义一个时钟，使其运行 $\mathsf{mdi}(\Lambda)$ 轮次，然后从头开始重复。该时钟的程序由头谓词 T_1、……、$T_{\mathrm{mdi}(\Lambda)}$ 以及以下规则组成：$T_1(0) :- \top, T_1 :- T_{\mathrm{mdi}(\Lambda)}$；对于 $i \in [\mathsf{mdi}(\Lambda) - 1]$，规则为 $T_{i+1}(0) :- \bot$ 和$T_{i+1} :- T_i$。

接下来，我们将每次处理一个模态深度，构造用于求值原始程序的规则。完成构造后，我们将给出一个具体示例，这或许能帮助读者在阅读下文的规则技术规范时更容易理解。

现在，对于 Λ 的每个头谓词 X_i 以及相应终止子句的主体 φ_i，我们重新定义头谓词 X_{X_i}，并定义相应的终止子句 $X_{X_i}(0) :-\varphi_i$。这些内容将加入 Λ'。相应的迭代子句稍后再定义。在此之前，我们先为子模式 $\theta_{(i,j)}$ 定义一些辅助规则。

我们向 Λ' 添加一个新的头谓词 $X_{\theta(1,j)}$，并添加规则 $X_{\theta(1,j)}(0) :- \bot$ 和 $X_{\theta(1,j)} :-_{T_1} \theta^*_{(1,j)}$；$X_{\theta(1,j)}$ 对每个 $j \in [k_0]$ 均如此，其中 $\theta^*_{(1,j)}$ 是通过将 $\theta_{(1,j)}$ 中的每个头谓词 X_i 替换为 X_{X_i} 而得到的模式。对于 $i \in \{2, \ldots, \mathsf{mdi}(\Lambda) - 1\}$，我们向 Λ' 添加一个新的头谓词 $X_{\theta(i,j)}$，并添加规则 $X_{\theta(i,j)}(0) :- \bot$ 和 $X_{\theta_{(i,j)}} :-_{T_i} \theta^*_{(i,j)}$；$X_{\theta(i,j)}$，其中模式 $\theta^*_{(i,j)}$ 定义如下。我们首先将 $\theta_{(i,j)}$ 中的每个 $\theta_{(i-1,j')}$ 替换为 $X_{\theta_{(i-1,j')}}$，也就是说，将 $\theta_{(i,j)}$ 的每个模态深度为 $i-1$ 的子模式替换为相应的头谓词。接着，在得到的模式中，我们将每个模态深度为 $i-2$ 的子模式替换为相应的头谓词。然后，我们继续进行替换

模态深度为 $i-3$ 的子模式，依此类推，直到将每个可能的子模式都替换为相应的头谓词。最后，我们将每个头谓词 X_i 替换为 X_{X_i}。最终得到模式 $\theta^*_{(i,j)}$，其模态深度为一。

接下来，我们为 Λ' 中的头谓词 X_{X_i} 定义迭代子句。对于每个 $i \in [k]$，我们定义子句 $X_{X_i} :-_{T_{\mathrm{mdi}(\Lambda)}} \psi_i^*$；$X_{X_i}$，其中模式 ψ_i^* 是通过采用与模式 $\theta_{(i,j)}$ 类似的策略，从 ψ_i 构造得到的：首先，将 ψ_i 的每个模态深度为 $\mathsf{mdi}(\Lambda) - 1$ 的子模式 $\theta \in S_{\mathrm{mdi}(\Lambda)-1}$ 替换为相应的头谓词X_θ，然后对模态深度为 $\mathsf{mdi}(\Lambda) - 2$ 的子模式进行类似处理，依此类推。最后，将每个头谓词 X_i 替换为 X_{X_i}。

作为最后一步，选择 Λ' 的注意力谓词（分别为打印谓词），使得 X_{X_i} 是 Λ' 中的注意力谓词（分别为打印谓词），当且仅当 X_i 是 Λ 中的注意力谓词（分别为打印谓词）。

我们给出一个关于该构造的简单示例。考虑 Λ 是该程序的情形 m

$$X(0) :- p \qquad X :- \Diamond\Diamond(\Diamond\Diamond X \wedge \Diamond X).$$

根据上述构造，程序 Λ' 除了前文所述的时钟规则外，还包含终止子句和迭代子句

$$\begin{array}{ll}
X_X(0) :- p & X_X :-_{T_4} \Diamond X_{\Diamond(\Diamond\Diamond X \wedge \Diamond X)}; X_X \\
X_{\Diamond X}(0) :- \bot & X_{\Diamond X} :-_{T_1} \Diamond X_X; X_{\Diamond X}. \\
X_{\Diamond\Diamond X}(0) :- \bot & X_{\Diamond\Diamond X} :-_{T_2} \Diamond X_{\Diamond X}; X_{\Diamond\Diamond X} \\
X_{\Diamond(\Diamond\Diamond X \wedge \Diamond X)}(0) :- \bot & X_{\Diamond(\Diamond\Diamond X \wedge \Diamond X)} :-_{T_3} \Diamond(X_{\Diamond\Diamond X} \wedge X_{\Diamond X}); X_{\Diamond(\Diamond\Diamond X \wedge \Diamond X)}
\end{array}$$

通过归纳法可以直接证明，以下条件成立。

1. 对于每个带指点的 Π-模型 (M, w)，对于每个 n，都有 $(M, w) \models \theta^n_{(i,j)} \Leftrightarrow (M, w) \models X^{\mathrm{mdi}(\Lambda)n+i}_{\theta_{(i,j)}}$。

2. 对于任意 $n \in \mathbb{N}$，以及任意满足 $\mathsf{mdi}(\Lambda)n + i < m < \mathsf{mdi}(\Lambda)(n+1) + i$ 的计算轮次 $m \in \mathbb{N}$，变量 $X_{\theta_{(i,j)}}$ 在轮次 m 中的解释与在轮次 $\mathsf{mdi}(\Lambda)n + i$ 中相同。

因此，以下条件成立。

1. 对于每个带指点的 Π-模型 (M, w)，对于每个 n，都有 $(M, w) \models X^n_i \Leftrightarrow (M, w) \models X^{\mathrm{mdi}(\Lambda)n}_{X_i}$。2. 对于任意 $n \in \mathbb{N}$，以及任意满足 $\mathsf{mdi}(\Lambda)n < m < \mathsf{mdi}(\Lambda)(n+1)$ 的计算轮次 $m \in \mathbb{N}$，变量 X_{X_i} 在轮次 m 中的解释与在轮次 $\mathsf{mdi}(\Lambda)n$ 中相同。

因此，翻译后的程序等价于原始程序 Λ，因此也等价于 Γ。显然，Λ' 的大小为 $\mathcal{O}(|\Lambda|) = \mathcal{O}(|\Gamma|)$，因为 Λ 的子模式数量与 $|\Lambda|$呈线性关系。计算时间是 $\mathcal{O}(max(1, \mathsf{md}(\Lambda)))$ 倍的 Γ的计算时间。

□

8 附录 B

8.1 森林分解

程序首先模拟将图分解为定向森林的过程。仅知道邻居的顺序还不够；节点还需要知道

邻居的标识符比节点本身的标识符更小还是更大。为此，需要逐位比较颜色的比特，这需要 ℓ 次迭代。我们用变量 T_1、.....、$\mathrm{T}_{\ell+2}$ 来统计这些迭代（除非另有说明，否则每个变量的终止子句为 $\bot$）：

$$\mathrm{T}_1 :- \top, \quad \mathrm{T}_2 :- \mathrm{T}_1, \quad \ldots \quad \mathrm{T}_\ell :- \mathrm{T}_{\ell-1}, \quad \mathrm{T}_{\ell+1} :- \mathrm{T}_\ell, \quad \mathrm{T}_{\ell+2} :- \mathrm{T}_{\ell+1}.$$

每个变量初始为假（$\mathrm{T}_i(0) :- \bot$）。变量 T_1 在第一轮迭代后变为真，变量 T_2 在第二轮迭代后变为真，依此类推。

为了比较一个节点的标识符与其邻居的标识符，我们首先定义变量符号 I_i，它们包含节点标识符的各个比特：

$$\mathrm{I}_1(0) :- p_1, \qquad \mathrm{I}_2(0) :- p_2, \qquad \ldots, \quad \mathrm{I}_\ell(0) :- p_\ell.$$

$$\mathrm{I}_1 :-_{\mathrm{T}_{\ell+1},\mathrm{T}_1} \mathrm{I}_1; \mathrm{I}_\ell; \mathrm{I}_1, \quad \mathrm{I}_2 :-_{\mathrm{T}_{\ell+1},\mathrm{T}_1} \mathrm{I}_2; \mathrm{I}_1; \mathrm{I}_2, \quad \ldots, \quad \mathrm{I}_\ell :-_{\mathrm{T}_{\ell+1},\mathrm{T}_1} \mathrm{I}_\ell; \mathrm{I}_{\ell-1}; \mathrm{I}_\ell.$$

在终止子句中，每个变量 I_i 的真值均为命题 p_i的真值。在迭代子句中，经过第一轮迭代后，这些变量会依次循环遍历节点标识符的每个比特。当所有 ℓ 个比特都访问过第 ℓ个位置后，标志 $\mathrm{T}_{\ell+1}$ 会停止循环，使每个变量保留其原始值（与终止子句中的值相同）。

我们还定义变量 I_i^δ $(1 \le \delta \le \Delta)$，用于保存节点第 δ个邻居的标识符比特：

$$\mathrm{I}_1^\delta :-_{T_{\ell+1},T_1} \mathrm{I}_1^\delta; \mathrm{I}_\ell^\delta; \Diamond_\delta \mathrm{I}_1, \quad \mathrm{I}_2^\delta :-_{T_{\ell+1},T_1} \mathrm{I}_2^\delta; \mathrm{I}_1^\delta; \Diamond_\delta \mathrm{I}_2, \quad \ldots, \quad \mathrm{I}_\ell^\delta :-_{T_{\ell+1},T_1} \mathrm{I}_\ell^\delta; \mathrm{I}_{\ell-1}^\delta; \Diamond_\delta \mathrm{I}_\ell.$$

在第一轮迭代中，变量 I_i^δ 继承第 δ个邻居的标识符比特；这是第一轮全局通信轮次。此后，这些变量便开始像变量 I_i一样循环轮转，并且也会同时停止轮转。

经过一轮迭代后，变量 I_ℓ 中保存了节点标识符最左侧的比特，变量 I_ℓ^δ中保存了其邻居标识符最左侧的比特。经过第二轮迭代后，这些变量中保存的将是第二个比特，以此类推。我们通过找到区分这些标识符的第一个比特来判断哪个节点的标识符更大；比特为 1 的节点拥有更大的标识符。

接下来，我们定义变量 DIF^δ $(1 \le \delta \le \Delta)$，用于比较标识符的比特：

$$\mathrm{DIF}^\delta :-_{\mathrm{T}_1} \neg \left(\mathrm{I}_\ell \leftrightarrow \mathrm{I}_\ell^\delta \right); \bot.$$

当且仅当节点 v 的颜色比特与其第 δ 个邻居的颜色比特在位置 ℓ 上不匹配时，变量 DIF^δ 才为真。第一轮迭代中不进行比较。

我们利用 DIF^δ 首次变为真的那一轮，定义变量 HIGH^δ 和 LOW^δ $(1 \le \delta \le \Delta)$，以告知节点哪些邻居拥有更大的标识符、哪些邻居拥有更小的标识符，因为这完全取决于我们找到的第一个不同的比特：

$$\mathrm{HIGH}^\delta :-_{\mathrm{HIGH}^\delta,\mathrm{LOW}^\delta,\mathrm{DIF}^\delta} \top; \bot; \mathrm{I}_1; \bot, \quad \mathrm{LOW}^\delta :-_{\mathrm{LOW}^\delta,\mathrm{HIGH}^\delta,\mathrm{DIF}^\delta} \top; \bot; \neg \mathrm{I}_1; \bot.$$

如果一个节点的标识符大于其第 δ 个邻居的标识符，则变量 HIGH^δ 变为 true。同样，如果其标识符较小，则变量 LOW^δ 变为 true。一旦在变量 I_ℓ 中找到不同位，我们就会在下一次迭代中将它们的值从变量 I_1 复制过来。经过 $\ell + 2$ 轮次后，对于每个节点中的每个 δ，HIGH^δ 或 LOW^δ 至少有一个为 true。为便于给出下一个定义，我们还定义 $\mathrm{HIGH}^0 := \top$。

现在可以定义如下公式：

$$\langle \delta \rangle \varphi := \bigvee_{\delta \le i \le \Delta} \left(\mathrm{HIGH}^{i-\delta} \wedge \mathrm{LOW}^{i-\delta+1} \wedge \Diamond_i \varphi \right).$$

公式 $\langle \delta \rangle \varphi$ 在节点 v 中为真，当且仅当 φ 在定向森林 F_δ 中节点 v 的父节点处为真。最后，我们定义公式 $\mathrm{END1} :=\mathrm{T}_{\ell+2}$，用于结束第一阶段。

8.2 Cole-Vishkin

我们将目前为止的程序扩展为在每个定向森林中模拟 CV 算法。我们逐比特比较节点的颜色，并在颜色比较之间插入通信轮次。为此，我们使用第 5.1 节中定义的时钟，并用 END1 标记时钟开始计时的时刻。这意味着，在此阶段开始之前，时钟变量始终保持其终端真值。我们使用与之前相同的变量符号：S_1、. . . 、$\mathrm{S}^{\log(\ell)}$、M_1、. . . 、$\mathrm{M}^{\log(\ell)}$ 以及 $\mathrm{S}_{\text{changing}}$。为方便起见，我们定义 $\mathrm{S}_{\text{changing}}(0) :- \top$；这只会使时钟延迟一次迭代。我们还将公式 $\mathrm{S}_{\text{on}} \coloneqq \mathrm{S}_{\text{changing}}$ 和 $\mathrm{S}_{\text{off}} \coloneqq \neg \mathrm{S}_{\text{on}}$ 定义为便于使用的简写。时钟的字符串包含 $\log(\ell) = \log\log(n)$ 个比特，因为分针始终指向颜色的 $\log(n)$ 个比特之一，并将其位置再次编码为二进制。

我们需要一个额外的标志，用来标记全局通信轮次发生的时刻：

$$\mathrm{CR} :-_{\mathrm{END1}} \bigwedge_{i=1}^{\log(\ell)} \neg \mathrm{M}_i \wedge \mathrm{S}_{\text{on}}; \bot.$$

变量 CR 利用时钟周期刚开始时的一个独特时刻来标记全局通信轮次。这个独特时刻是分针全为零的两次迭代中的第一次。在第二次这样的迭代中，公式 S_{on} 不再成立。也可以借助第 5.2 节定义的变量 X_{reset} 来定义这个独特时刻，但我们并不需要其他用途的必要前向时钟。

为了比较节点与其邻居的颜色，我们定义变量 $\mathrm{B}_i^\delta (1 \leq \delta \leq \Delta)$，表示森林 F_δ中节点颜色的各个比特，从其标识符开始。我们还定义变量 P_i^δ $(1 \leq \delta \leq \Delta)$，表示森林$F_\delta$中节点的父节点颜色的各个比特：

$$\begin{array}{llll}
\mathrm{B}_1^\delta :-_{\mathrm{CR},\mathrm{S}_{\text{off}}} \mathrm{N}_1^\delta; \mathrm{B}_2^\delta; \mathrm{B}_1^\delta, & \mathrm{B}_2^\delta :-_{\mathrm{CR},\mathrm{S}_{\text{off}}} \mathrm{N}_2^\delta; \mathrm{B}_3^\delta; \mathrm{B}_2^\delta, & \ldots, & \mathrm{B}_\ell^\delta :-_{\mathrm{CR},\mathrm{S}_{\text{off}}} \mathrm{N}_\ell^\delta; \mathrm{B}_1^\delta; \mathrm{B}_\ell^\delta. \\
\mathrm{P}_1^\delta :-_{\mathrm{CR},\mathrm{S}_{\text{off}}} \langle\delta\rangle \mathrm{N}_1^\delta; \mathrm{P}_2^\delta; \mathrm{P}_1^\delta, & \mathrm{P}_2^\delta :-_{\mathrm{CR},\mathrm{S}_{\text{off}}} \langle\delta\rangle \mathrm{N}_2^\delta; \mathrm{P}_3^\delta; \mathrm{P}_2^\delta, & \ldots, & \mathrm{P}_\ell^\delta :-_{\mathrm{CR},\mathrm{S}_{\text{off}}} \langle\delta\rangle \mathrm{N}_\ell^\delta; \mathrm{P}_1^\delta; \mathrm{P}_\ell^\delta.
\end{array}$$

每个变量 B_i^δ 都依次循环遍历森林 F_δ中节点颜色的各个比特。每个变量 P_i^δ 也以同样的方式遍历其父节点颜色的各个比特。变量每过一个“分钟”就旋转一次，由条件 S_{off}标记。分针始终以二进制形式指示当前位于 B_1^δ 和 P_1^δ 中的比特原来的位置。在全局通信轮次期间，变量 B_i^δ 会更新为下一个颜色 N_i^δ （见下文），而变量 P_i^δ 会接收森林 F_δ中该节点父节点的新颜色。对于根节点，公式 $\langle\delta\rangle \mathrm{N}_i^\delta$ 始终不成立，这意味着根节点将自己的颜色与颜色 0 进行比较，正如第 6.1 节所述。

我们使用与之前相同的变量 DIF^δ，并使用标志 CR 来防止假阳性，以比较子节点和父节点颜色的比特。我们比较位置 1 上的比特，而不是像之前那样比较 ℓ，因为 CV 算法要寻找最右侧不同位：

$$\mathrm{DIF}^\delta :-_{\mathrm{CR},\mathrm{END1},\mathrm{T}_1} \bot; \neg \left(\mathrm{B}_1^\delta \leftrightarrow \mathrm{P}_1^\delta\right); \neg \left(\mathrm{I}_\ell \leftrightarrow \mathrm{I}_\ell^\delta\right); \bot.$$

我们还定义变量 GET^δ $(1 \leq \delta \leq \Delta)$：找到第一个不同的比特后，这些变量会变为 true，直到在全局通信轮次中被重置：

$$\mathrm{GET}^\delta :-_{\mathrm{CR},\mathrm{GET}^\delta} \bot; \top; \mathrm{DIF}^\delta.$$

现在可以定义变量 N_i^δ $(1 \leq \delta \leq \Delta)$，用于计算下一个颜色

即节点在时钟每一轮中的颜色：

$$\begin{aligned}
&\mathrm{N}_1^\delta :-_{\mathrm{GET}^\delta,\mathrm{DIF}^\delta} \mathrm{N}_1^\delta; \mathrm{B}_1^\delta; p_1,\\
&\mathrm{N}_2^\delta :-_{\mathrm{GET}^\delta,\mathrm{DIF}^\delta} \mathrm{N}_2^\delta; \mathrm{M}_1; p_2,\\
&\vdots\\
&\mathrm{N}_\ell^\delta :-_{\mathrm{GET}^\delta,\mathrm{DIF}^\delta} \mathrm{N}_\ell^\delta; \mathrm{M}_{\ell-1}; p_\ell.
\end{aligned}$$

默认情况下，这些变量存储节点的标识符，并在第一个全局通信轮次中由变量 B_i^δ 和 P_i^δ 使用。条件 DIF^δ 将第一个不同的比特的真值复制到变量 N_1^δ，并将其位置的比特从分针复制到其余变量 N_i^δ。条件 GET^δ 会保留计算出的颜色，直到时钟进入下一轮，从而确保只考虑第一个不同的比特。

CV 算法需要 $L = \log^*(n) + 3$ 个通信轮次。我们使用由变量 H_1、.....、H_L组成的时针来计数；经过 $\log^*(n)+2$ 个“小时”后，H_L 变为真：

$$\mathrm{H}_1 :-_{\mathrm{H}_1 \vee \mathrm{CR}} \top; \bot, \quad \mathrm{H}_2 :-_{\mathrm{H}_2 \vee (\mathrm{H}_1 \wedge \mathrm{CR})} \top; \bot, \quad \ldots, \quad \mathrm{H}_L :-_{\mathrm{H}_L \vee (\mathrm{H}_{L-1} \wedge \mathrm{CR})} \top; \bot.$$

再过一个小时后，使用以下变量停止时钟：

$$\mathrm{STOP} :- \mathrm{H}_L \wedge \left(\bigwedge_{i=1}^{\log(\ell)} \neg \mathrm{M}_i \wedge \mathrm{S}_{\mathrm{on}} \right)$$

括号中包含的公式与 CR 中使用的公式相同。因此，该变量会在最后一个“小时”与 CR 同时开启，并可用于停止变量 P_i^δ 的迭代，以避免不必要的全局通信轮次。再经过一次迭代后，使用标志END2 :–STOP 停止所有其他先前提到的变量。

为了证明引理 6.2，我们定义 8^Δ “指定”谓词。这些谓词对应于每个节点的颜色，即将其 Δ 来自森林的颜色串接起来所得的结果：

$$\begin{aligned}
&\mathrm{CLR}_0 :-_{\mathrm{END2}} \bigwedge_{\delta=1}^{\Delta} \left(\neg \mathrm{B}_3^\delta \wedge \neg \mathrm{B}_2^\delta \wedge \neg \mathrm{B}_1^\delta \right); \bot,\\
&\vdots\\
&\mathrm{CLR}_{8^\Delta - 1} :-_{\mathrm{END2}} \bigwedge_{\delta=1}^{\Delta} \left(\mathrm{B}_3^\delta \wedge \mathrm{B}_2^\delta \wedge \mathrm{B}_1^\delta \right); \bot.
\end{aligned}$$

这些颜色只是由比特拼接而成的。我们可以从程序中删除变量 CLR_0, CLR_8,CLR_{16}, . . .，因为它们对应的拼接结果中，某个定向森林里的节点颜色为 0，而这是不可能的。如果将剩余变量 CLR_i 视为指定谓词，那么引理 6.2 现在成立：

► **引理 6.2**。存在一个MPMSC 公式，具有以下性质。

1. 它定义了一个合法的 7^Δ-着色。2. 忽略指定谓词后，它有 $\mathcal{O}(\Delta \log(n))$ 个头部，大小为 $\mathcal{O}(\Delta^2 \log(n))$。3. 所需的迭代次数为 $\mathcal{O}(\log(n) \log\log(n) \log^*(n))$。4. 恰有$\log^*(n) + 4$ 个全局通信轮次。

证明。 1) 我们有 7^{Δ} 个指定（注意和打印）谓词：CLR_0、.....、$\mathrm{CLR}_{8^{\Delta}-1}$，不包括变量

$$\mathrm{CLR}_0, \mathrm{CLR}_8, \mathrm{CLR}_{16}, \ldots.$$

由于 B_1^{δ}、B_2^{δ} 和 B_3^{δ} $(1 \leq \delta \leq \Delta)$ 的真值的每种可能组合都对应于单个变量 CLR_i，因此打印谓词彼此互斥，这意味着在给定节点中至多只有一个打印谓词为真。每个省略的谓词都对应于一种不可能的真值组合，这意味着在每个节点中必有一个打印谓词变为真。这意味着每个节点输出一个 7^{Δ} 比特的位串，其中恰好有一个比特为 1。因此，该程序定义了一种 7^{Δ}-着色。

如果两个邻居共享相同的输出，那么它们对于变量 B_1^{δ}、B_2^{δ} 和 B_3^{δ} $(1 \leq \delta \leq \Delta)$ 的真值组合就会相同。到此时，每个节点中的变量 B_i^{δ} $(i > 3, 1 \leq \delta \leq \Delta)$ 都已经变为假，这意味着在每个森林中，这两个邻居会共享相同的颜色。这是不可能的，因为存在某个森林 F_{δ}，其中一个邻居是另一个邻居的父节点，而该程序确保父节点和子节点永远不会共享相同的颜色。因此，没有两个邻居会共享相同的输出，并且该着色是合法的。

2) 让我们来计算头部的数量及其长度：

- 我们有 $\mathcal{O}(1)$ 个大小为 $\mathcal{O}(\log\log(n))$ 的变量 $\mathrm{S}_{\text{变化}}$、CR 和 STOP。
- 我们有 $\mathcal{O}(\Delta)\mathcal{O}(\log(n))$ 个变量 T_i、I_i、I_i^{δ}、DIF^{δ}、HIGH^{δ}、LOW^{δ}、S_i、M_i、B_i^{δ}、P_i^{δ}、GET^{δ}、N_i^{δ}、H_i 和 END2。

将头部数量相加，得到

$$\mathcal{O}(1) + \mathcal{O}(\Delta)\mathcal{O}(\log(n)) = \mathcal{O}(\Delta \log(n)).$$

将头部数量乘以各自的大小，得到

$$\begin{aligned} \mathcal{O}(1)\mathcal{O}(\log\log(n)) + \mathcal{O}(\Delta)\mathcal{O}(\log(n))\mathcal{O}(\Delta) &= \mathcal{O}(\log\log(n)) + \mathcal{O}(\Delta^2 \log(n)) \\ &= \mathcal{O}(\Delta^2 \log(n)). \end{aligned}$$

3) 算法的第一阶段会持续 $\mathcal{O}(\log(n))$ 次迭代，直到变量 END1 变为真。之后，时钟开始计时，并持续 $\mathcal{O}(\log^*(n))$ 轮次。

时钟的每一轮都包含相同数量的迭代。对于分针的每个位置，我们有 $\mathcal{O}(\log\log(n))$ 次迭代：第一次迭代中，分针更新到新位置；至多有 $\log\log(n)$ 次迭代，秒针在其中统计分针最右侧的 1 的个数；另有一次迭代，秒针停止移动。分针总共有 $\log(n)$ 个位置，时钟运行 $\mathcal{O}(\log^*(n))$ 个“小时”。这远远超过第一阶段的迭代次数，因此时间复杂度为 $\mathcal{O}(\log(n)\log\log(n)\log^*(n))$。

4) 变量 I_i^{δ} 和 P_i^{δ} 是仅有的包含菱形算子的变量。这些菱形算子在第一迭代轮次以及变量 CR 为真的每一轮中激活。后者每个“小时”发生一次，而时针将“小时”数限制为 $\log^*(n) + 3$。因此，恰好有 $\log^*(n) + 4$ 个全局通信轮次。 □

第一个全局通信轮次用于接收邻居的标识符以进行比较；此后，剩余的 $\log^*(n) + 3$ 个全局通信轮次用于计算 CV，这与 CV 算法的通信轮次数相同。

8.3 shift-down

算法的下移阶段从前一阶段结束时的 7着色继续进行。它通过在每个定向森林中重复以下三个步骤来实现：

1. 每个节点继承其父节点的颜色（根节点将颜色改为 1 或 2，且与之前的颜色不同）。这样一来，同一节点的所有子节点都具有相同的颜色。每个节点都会保存之前的颜色以供比较。2. 每个节点都会收到一条消息，其中包含其父节点的颜色。该颜色也会保存下来以供比较。3. 每个节点将自己的颜色（在步骤 1 中继承）、其子节点的颜色（在步骤 1 中保存）以及其父节点的颜色（在步骤 2 中保存）进行比较。如果它的颜色最大，就将其改为 1、2 或 3 中未被其子节点或父节点使用的最小颜色。

每重复执行这些步骤一次，每个定向森林中的最高颜色至少降低 1。重复 4 次后，每个定向森林中的 7-着色都会降为 3-着色。使用该算法无法实现最优的 2着色。

我们扩展之前的程序以纳入下移技术，同时省略前一阶段的指定谓词。我们用三个变量标记当前的下移步骤：

$$\mathrm{Z}_1 :\!-_{\mathrm{Z}_1 \vee \mathrm{Z}_2, \mathrm{END2}} \bot; \top; \bot, \quad \mathrm{Z}_2 :\!-_{\mathrm{Z}_1} \top; \bot, \quad \mathrm{Z}_3 :\!-_{\mathrm{Z}_2} \top; \bot.$$

在前一阶段结束并激活 Z_1 之前，三个变量都为假。这些变量轮流开启和关闭；在任一迭代轮次中，恰好只有一个变量为真。值为真的变量中，索引较低者指示程序在下一次迭代中执行的步骤。

这些变量只需循环 4 轮。为了终止这一阶段，我们像在第一阶段中那样定义另一组计时器变量：

$$\mathrm{T}'_1 :\!-_{\mathrm{T}'_1 \vee \mathrm{Z}_3} \top; \bot, \quad \mathrm{T}'_2 :\!-_{\mathrm{T}'_2 \vee (\mathrm{T}'_1 \wedge \mathrm{Z}_3)} \top; \bot, \quad \mathrm{T}'_3 :\!-_{\mathrm{T}'_3 \vee (\mathrm{T}'_2 \wedge \mathrm{Z}_3)} \top; \bot, \quad \mathrm{T}'_4 :\!-_{\mathrm{T}'_4 \vee (\mathrm{T}'_3 \wedge \mathrm{Z}_3)} \top; \bot.$$

第一个循环结束后，变量 T'_1 变为真；第二个循环结束后，T'_2 变为真，以此类推。每个变量一旦激活，就会一直保持为真。我们还定义公式 END3：$=\mathrm{T}'_4$，用于标记下移阶段的结束。

鉴于我们已将每个森林中的颜色数量减少到仅 7 种，直接引用各森林中节点颜色的变量 C_i^δ 会很方便；我们稍后会定义这些变量。我们还使用变量 Cc_i^δ $(1 \le \delta \le \Delta)$，用于存储节点先前的颜色，该颜色会在步骤 1 中传递给其子节点；以及变量 Cp_i^δ $(1 \le \delta \le \Delta)$，用于存储其父节点的颜色。上标指向定向森林的标签，下标指向颜色。后两组变量很容易定义：

$$\begin{aligned} &\mathrm{Cc}_1^\delta :\!-_{\mathrm{Z}_1} \mathrm{C}_1^\delta; \mathrm{Cc}_1^\delta, && \ldots, \quad \mathrm{Cc}_7^\delta :\!-_{\mathrm{Z}_1} \mathrm{C}_7^\delta; \mathrm{Cc}_7^\delta. \\ &\mathrm{Cp}_1^\delta :\!-_{\mathrm{Z}_2} \langle\delta\rangle \mathrm{C}_1^\delta; \mathrm{Cp}_1^\delta, && \ldots, \quad \mathrm{Cp}_7^\delta :\!-_{\mathrm{Z}_2} \langle\delta\rangle \mathrm{C}_7^\delta; \mathrm{Cp}_7^\delta. \end{aligned}$$

对于变量 C_i^δ，我们需要考虑在步骤 1 中以不同方式更改根节点的颜色。为此，我们定义额外的变量 R^δ $(1 \le \delta \le \Delta)$，用于标记每个定向森林的根节点：

$$\mathrm{R}^\delta :\!-_{\mathrm{R}^\delta, \mathrm{CR}} \top; \neg\langle\delta\rangle\top; \bot.$$

只有当节点在森林 F_δ 中没有父节点时，才会被标记为该森林中的根节点；也就是说，菱形算子 $\langle\delta\rangle$ 不会指向任何位置。由于其他变量在前一阶段 CR 为

真时已经使用了菱形算子，因此这不会增加任何全局通信轮次，也不需要额外的标志。

对于步骤 3，我们使用一些实用公式来简化问题。我们定义公式 G^δ $(1 \leq \delta \leq \Delta)$；当森林 F_δ 中某个节点的颜色高于其子节点和父节点时，这些公式为真：

$$\mathrm{G}^\delta \coloneqq \mathrm{Z}_3 \wedge \bigvee_{i=3}^{7} \left(\mathrm{C}_i^\delta \wedge \neg \bigvee_{j=i+1}^{7} \left(\mathrm{Cc}_j^\delta \vee \mathrm{Cp}_j^\delta \right) \right).$$

这些公式表明，程序处于步骤 3，该节点在森林 F_δ 中的颜色介于 3 和 7 之间，并且其父节点和子节点都没有更高的颜色。

我们还定义公式 L_i^δ $(1 \leq \delta \leq \Delta)$。为真的公式中较低的索引告诉我们，在森林 F_δ 中，颜色 1、2 和 3 中哪一种是节点可以切换到的最低可用颜色：

$$\mathrm{L}_1^\delta \coloneqq \neg \left(\mathrm{Cc}_1^\delta \vee \mathrm{Cp}_1^\delta \right), \quad \mathrm{L}_2^\delta \coloneqq \neg \mathrm{L}_1^\delta \wedge \neg \left(\mathrm{Cc}_2^\delta \vee \mathrm{Cp}_2^\delta \right), \quad \mathrm{L}_3^\delta \coloneqq \neg \mathrm{L}_1^\delta \wedge \neg \mathrm{L}_2^\delta.$$

现在我们可以定义变量 C_i^δ $(1 \leq \delta \leq \Delta)$ 了：

$$\begin{array}{lllllll}
\mathrm{C}_1^\delta \coloneq_{\mathrm{G}^\delta, \mathrm{Z}_3 \vee \mathrm{Z}_2, \mathrm{R}^\delta \wedge \mathrm{Z}_1, \mathrm{Z}_1, \mathrm{END2}} & \mathrm{L}_1^\delta; & \mathrm{C}_1^\delta; & \neg \mathrm{C}_1^\delta; & \langle\delta\rangle \mathrm{C}_1^\delta; & \neg \mathrm{B}_3^\delta \wedge \neg \mathrm{B}_2^\delta \wedge \mathrm{B}_1^\delta; & \bot, \\
\mathrm{C}_2^\delta \coloneq_{\mathrm{G}^\delta, \mathrm{Z}_3 \vee \mathrm{Z}_2, \mathrm{R}^\delta \wedge \mathrm{Z}_1, \mathrm{Z}_1, \mathrm{END2}} & \mathrm{L}_2^\delta; & \mathrm{C}_2^\delta; & \mathrm{C}_1^\delta; & \langle\delta\rangle \mathrm{C}_2^\delta; & \neg \mathrm{B}_3^\delta \wedge \mathrm{B}_2^\delta \wedge \neg \mathrm{B}_1^\delta; & \bot, \\
\mathrm{C}_3^\delta \coloneq_{\mathrm{G}^\delta, \mathrm{Z}_3 \vee \mathrm{Z}_2, \mathrm{Z}_1, \mathrm{END2}} & \mathrm{L}_3^\delta; & \mathrm{C}_3^\delta; & & \langle\delta\rangle \mathrm{C}_3^\delta; & \neg \mathrm{B}_3^\delta \wedge \mathrm{B}_2^\delta \wedge \mathrm{B}_1^\delta; & \bot, \\
\mathrm{C}_4^\delta \coloneq_{\mathrm{G}^\delta, \mathrm{Z}_3 \vee \mathrm{Z}_2, \mathrm{Z}_1, \mathrm{END2}} & \bot; & \mathrm{C}_4^\delta; & & \langle\delta\rangle \mathrm{C}_4^\delta; & \mathrm{B}_3^\delta \wedge \neg \mathrm{B}_2^\delta \wedge \neg \mathrm{B}_1^\delta; & \bot, \\
\vdots & & & & & & \\
\mathrm{C}_7^\delta \coloneq_{\mathrm{G}^\delta, \mathrm{Z}_3 \vee \mathrm{Z}_2, \mathrm{Z}_1, \mathrm{END2}} & \bot; & \mathrm{C}_7^\delta; & & \langle\delta\rangle \mathrm{C}_7^\delta; & \mathrm{B}_3^\delta \wedge \mathrm{B}_2^\delta \wedge \mathrm{B}_1^\delta; & \bot.
\end{array}$$

上标表示定向森林的标签，下标表示节点在该森林中的颜色。从左到右，我们先考虑步骤 3：如果节点的颜色高于其子节点和父节点，那么最低可用变量 C_1^δ、C_2^δ 或 C_3^δ 会变为真。如果节点的颜色较低，或者我们处于步骤 2，则变量不会更新。如果我们处于步骤 1 且该节点是根节点，那么只有变量 C_1^δ 和 C_2^δ 会交替变化（根节点对应的 $\langle\delta\rangle$-公式始终为假）。如果该节点在步骤 1 中不是根节点，那么它会通过菱形 $\langle\delta\rangle$ 继承其父节点的颜色。变量激活后，会根据前一阶段最终颜色的比特配置来组合生成。

最后，我们可以使用标志 END3 作为条件，在步骤 1–3 循环 4 次后，停止本节中所有变量的更新。在公式 END3 变为真的前一次迭代中，变量 Z_3 为真，这意味着颜色变量 C_i^δ 会在当前循环结束后，于下一次迭代中被 END3 停止。

我们再次定义 4^Δ “指定的”谓词，以便证明一个有用的引理。它们表示定向森林 F_δ中某个节点的最终颜色：

$$\mathrm{CLR}_0 \coloneq_{\mathrm{END3}} \bigwedge_{\delta=1}^{\Delta} \left(\neg \mathrm{C}_3^\delta \wedge \neg \mathrm{C}_2^\delta \wedge \neg \mathrm{C}_1^\delta \right); \bot,$$

$$\vdots$$

$$\mathrm{CLR}_{4^\Delta - 1} \coloneq_{\mathrm{END3}} \bigwedge_{\delta=1}^{\Delta} \left(\mathrm{C}_3^\delta \wedge \neg \mathrm{C}_2^\delta \wedge \neg \mathrm{C}_1^\delta \right); \bot.$$

这些颜色由各森林专属的颜色构成，就像它们是比特对一样。因此，我们可以再次从程序中移除变量 CLR_0,CLR_4,CLR_8, . . .,

因为颜色 0 在任何森林中都不会用到。如果我们将剩余变量 C_i^δ 视为指定谓词，则以下引理成立：

引理 8.1。 存在一个具有以下性质的MPMSC 公式。

1. 它定义了一种合法的 3^Δ着色。*2.* 忽略指定谓词后，它有 $\mathcal{O}$（$\Delta\log$（n）个头部，并且大小为 （$\Delta^2\log$（n）。*3.* 所需的迭代次数为 $\mathcal{O}$（$\log$（n） $\log\log$（n） $\log^*$（n）。*4.* 恰好有 $\log^*$（n） $+12$ 个全局通信轮次。

证明。 1）我们有 3^Δ 个指定（attention 和 print）谓词：CLR_0、.....、$\mathsf{CLR}_{4^\Delta-1}$，减去变量

$$\mathrm{CLR}_0, \mathrm{CLR}_4, \mathrm{CLR}_8, \ldots.$$

每个指定谓词分别对应于每个$1 \leq \delta \leq \Delta$的一个真变量 C_1^δ、C_2^δ 或 C_3^δ ，这意味着它们再次彼此互斥，并且在给定节点中只能有一个为真。对于每个 $1 \leq \delta \leq \Delta$，变量 C_i^δ （$1 \leq i \leq 7$）彼此互斥，其中一个也必须为真。此外，变量 C_i^δ （$i > 3$）在此时均为假。这意味着每个节点中至少有一个指定谓词必须变为真。因此，每个节点输出一个 3^Δ比特位串，其中恰好有一个比特为 1，并且该程序定义了一种 3^Δ着色。

如果两个邻居具有相同的输出字符串，那么对于所有变量 C_i^δ $(1 \leq i \leq 7, 1 \leq \delta \leq \Delta)$，它们将具有相同的真值。这意味着这些邻居在每个定向森林中都具有相同的颜色。这再次是不可能的，因为存在某个定向森林 F_δ，其中一个邻居是另一个的父节点，而程序确保父节点和子节点绝不会具有相同的颜色。因此，没有两个邻居具有相同的输出，且该着色是正确的。

2）让我们计算头部的数量及其长度。根据引理 6.2，我们此前有 $\mathcal{O}(\Delta\log(n))$ 个头部。它们的大小为 $\mathcal{O}(\Delta^2\log(n))$。让我们计算新的头部及其大小：

- We have $\mathcal{O}(\Delta)$ variables Z_i, T'_i, Cc_i^δ, Cp_i^δ, R^δ and C_i^δ of size $\mathcal{O}(\Delta)$.

将这些头部与程序之前的部分相加，我们得到

$$\mathcal{O}(\Delta\log(n)) + \mathcal{O}(\Delta) = \mathcal{O}(\Delta\log(n)).$$

将这些大小相加，我们得到

$$\mathcal{O}(\Delta^2\log(n)) + \mathcal{O}(\Delta)\mathcal{O}(\Delta) = \mathcal{O}(\Delta^2\log(n)) + \mathcal{O}(\Delta^2) = \mathcal{O}(\Delta^2\log(n)).$$

3）根据引理 6.2，之前的迭代次数为 $\mathcal{O}(\log(n)\log\log(n)\log^*(n))$。下移阶段的迭代次数等于步数乘以循环次数。由于有 $\mathcal{O}(1)$ 步和 $\mathcal{O}(1)$ 个循环，这意味着额外增加 $\mathcal{O}(1)$ 次迭代。迭代总次数仍为

$$\mathcal{O}(\log(n)\log\log(n)\log^*(n)) + \mathcal{O}(1) = \mathcal{O}(\log(n)\log\log(n)\log^*(n)).$$

4）前一阶段恰好有 $\log^*(n) + 4$ 个全局通信轮次。在这一阶段，只有在步骤 1 和 2 中才会使用菱形算子。由于这些步骤会循环执行 4 次，因此额外增加了 $4 \cdot 2 = 8$ 个全局通信轮次，这与该算法相符。于是，全局通信轮次的总数为

$$\log^*(n) + 4 + 8 = \log^*(n) + 12.$$

□

8.4 基本颜色归约

在下一阶段，我们将前一阶段得到的 3-着色组合起来，得到整个图的着色：把它们并列放置，森林 F_1 中节点的颜色成为其颜色最右侧的两个比特，森林 F_2 中节点的颜色成为接下来的两个比特，以此类推，直到森林 F_Δ 中节点的颜色成为最左侧的两个比特。这样便得到整个图的一个合法的 (3^Δ)-着色。

最后，我们使用一种称为基本颜色归约的技术，将其转化为一个 $(\Delta+1)$-着色。每一轮中，节点都会将自己的颜色发送给所有邻居。之后，颜色高于其邻居的节点会将颜色改为集合 $[\Delta+1]$ 中最低的可用颜色；这些颜色中至少有一种可用，因为每个节点最多有 Δ 个邻居。我们重复此算法，直到节点的最高颜色为 $\Delta+1$，这最多需要 $7^\Delta-(\Delta+1)$ 个通信轮次。

我们需要 2Δ 个变量来存储新颜色的比特。对于每个 δ，我们定义两个变量，用来存储节点在森林 F_δ 中颜色的比特。这些比特合在一起将定义节点在图中的完整颜色：

$$\begin{aligned}&\mathrm{B}_1 :-_{\mathrm{END3}} \mathrm{C}_1^1 \vee \mathrm{C}_3^1; \bot, \quad \mathrm{B}_3 :-_{\mathrm{END3}} \mathrm{C}_1^2 \vee \mathrm{C}_3^2; \bot, \quad \ldots, \quad \mathrm{B}_{2\Delta-1} :-_{\mathrm{END3}} \mathrm{C}_1^\Delta \vee \mathrm{C}_3^\Delta; \bot,\\ &\mathrm{B}_2 :-_{\mathrm{END3}} \mathrm{C}_2^1 \vee \mathrm{C}_3^1; \bot, \quad \mathrm{B}_4 :-_{\mathrm{END3}} \mathrm{C}_2^2 \vee \mathrm{C}_3^2; \bot, \quad \ldots, \quad \mathrm{B}_{2\Delta} :-_{\mathrm{END3}} \mathrm{C}_2^\Delta \vee \mathrm{C}_3^\Delta; \bot.\end{aligned}$$

变量 $\mathrm{B}_{2\delta-1}$ 指节点在森林 F_δ 中颜色的最右位，而 $\mathrm{B}_{2\delta}$ 指最左位。

在算法的最后一个阶段，每个节点都会将自己的颜色与邻居的颜色进行比较。我们已经为节点自身颜色的各个比特定义了变量，但还必须为邻居颜色的所有比特定义变量。与第 8.1 节和第 8.2 节一样，在将每个节点的颜色与其邻居的颜色进行比较时，比特会轮流移动。为此，我们需要 2Δ 个计时器变量。比较完比特后，节点会更改颜色并再次进行比较。每次都需要重置计时器，因此我们使用标志 CR' 来重置它们（稍后会定义该标志）：

$$\mathrm{T}_1^1 :-_{\mathrm{CR}',\mathrm{T}_1^1} \top; \top; \bot, \quad \mathrm{T}_2^1 :-_{\mathrm{CR}',\mathrm{T}_1^1} \bot; \top; \bot, \quad \ldots, \quad \mathrm{T}_{2\Delta}^1 :-_{\mathrm{CR}',\mathrm{T}_{2\Delta-1}^1} \bot; \top; \bot.$$

此外，我们还要再增加 $\Delta+1$ 个计时器变量，用于一个阶段，在该阶段中，我们确定节点可以切换到的最低可用颜色，因此计时器变量总数为 $3\Delta+1$ 个：

$$\mathrm{T}_1^2 :-_{\mathrm{CR}',\mathrm{T}_{2\Delta}^1} \bot; \top; \bot, \quad \mathrm{T}_2^2 :-_{\mathrm{CR}',\mathrm{T}_1^2} \bot; \top; \bot, \quad \ldots, \quad \mathrm{T}_{\Delta+1}^2 :-_{\mathrm{CR}',\mathrm{T}_\Delta^2} \bot; \top; \bot.$$

除了这个两部分计时器外，我们还需要第二个计时器来统计第一个计时器的循环次数。图中节点的最大颜色在每次循环中至少减少一个，因此我们需要 $t=3^\Delta-\Delta$ 个变量；经过 $t-1$ 次循环后，每个节点的颜色都会降至至多 $\Delta+1$。定义如下：

$$\mathrm{LP}_1 :-_{\mathrm{LP}_1,\mathrm{CR}'} \top; \top; \bot, \quad \mathrm{LP}_2 :-_{\mathrm{LP}_2,\mathrm{CC}} \top; \mathrm{LP}_1; \bot, \quad \ldots, \quad \mathrm{LP}_t :-_{\mathrm{LP}_t,\mathrm{CC}} \top; \mathrm{LP}_{t-1}; \bot.$$

较低索引指的是计数器当前所在的循环。一旦 $\mathrm{END4} := \mathrm{LP}_t$ 变为真，程序就可以停止。我们将使用此标志来停止所有变量，但为节省空间，定义中省略了它。

我们定义两个标志：CC 表示节点改变颜色的轮次，CR' 表示紧随其后的全局通信轮次。

$$\mathrm{CC} :-_{\mathrm{CR}'\vee\mathrm{CC},\mathrm{T}_{\Delta+1}^2} \bot; \top; \bot, \quad \mathrm{CR}' :-_{\mathrm{CC},\mathrm{CR}'\vee\mathrm{T}_1^1,\mathrm{END3}} \top; \bot; \top; \bot.$$

条件 END3 触发第一次全局通信轮次；该条件将不再被重新检查。此后，计时器变量开始计时。等它们全部开启后，

节点改变颜色时，变量 CC 会在一个轮次内变为 true。随后，变量 CR′ 也会在一个轮次内变为 true，标记全局通信轮次。之后，计数器重置，我们重新开始。

我们修改所有位变量 B_i 的定义，使其在计数器的第一部分处于激活状态时轮转：

$$\begin{aligned}
&\mathrm{B}_1 :-_{\mathrm{T}_1^2,\mathrm{T}_1^1,\mathrm{END3}}\ \mathrm{B}_1;\mathrm{B}_{2\Delta};\mathrm{C}_1^1 \vee \mathrm{C}_3^1;\bot,\\
&\mathrm{B}_2 :-_{\mathrm{T}_1^2,\mathrm{T}_1^1,\mathrm{END3}}\ \mathrm{B}_2;\mathrm{B}_1;\mathrm{C}_2^1 \vee \mathrm{C}_3^1;\bot,\\
&\vdots\\
&\mathrm{B}_{2\Delta} :-_{\mathrm{T}_1^2,\mathrm{T}_1^1,\mathrm{END3}}\ \mathrm{B}_{2\Delta};\mathrm{B}_{2\Delta-1};\mathrm{C}_2^{\Delta} \vee \mathrm{C}_3^{\Delta};\bot.
\end{aligned}$$

这些变量恰好轮转 2Δ 次迭代的真值，然后停在其原始值。接下来的 $\Delta+1$ 次迭代中，我们搜索最小的可用颜色，此时这些变量保持不变。稍后我们还会再修改一次这些定义，以展示节点如何更改颜色。

我们还需要变量 B'^{δ}_i $(1 \le \delta \le \Delta)$ 来表示每个邻居颜色的比特：

$$\begin{aligned}
&\mathrm{B}'^{\delta}_1 :-_{\mathrm{CR}',\mathrm{T}_1^2,\mathrm{T}_1^1}\ \Diamond_\delta \mathrm{B}_1;\mathrm{B}'^{\delta}_1;\mathrm{B}'^{\delta}_{2\Delta};\bot,\\
&\mathrm{B}'^{\delta}_2 :-_{\mathrm{CR}',\mathrm{T}_1^2,\mathrm{T}_1^1}\ \Diamond_\delta \mathrm{B}_2;\mathrm{B}'^{\delta}_2;\mathrm{B}'^{\delta}_1;\bot,\\
&\vdots\\
&\mathrm{B}'^{\delta}_{2\Delta} :-_{\mathrm{CR}',\mathrm{T}_1^2,\mathrm{T}_1^1}\ \Diamond_\delta \mathrm{B}_{2\Delta};\mathrm{B}'^{\delta}_{2\Delta};\mathrm{B}'^{\delta}_{2\Delta-1};\bot.
\end{aligned}$$

这些比特的轮转方式与之前相同。在全局通信轮次中，这些变量接收邻居更新后颜色的比特。

接下来，我们定义变量 DIF'^{δ} $(1 \le \delta \le \Delta)$，用于比较一个节点与其邻居的比特。我们比较的是颜色的大小，因此必须从比较最左侧的比特开始：

$$\mathrm{DIF}'^{\delta} :-_{\mathrm{CR}',\mathrm{T}_1^1}\ \bot;\neg\left(\mathrm{B}_{2\Delta} \leftrightarrow \mathrm{B}'^{\delta}_{2\Delta}\right);\bot.$$

我们在第 8.2 节定义过一个类似的变量。同样，我们定义变量 HIGH'^{δ} 和 LOW'^{δ} ($1 \le \delta \le \Delta$)，用来存储哪个节点的颜色更高这一信息：

$$\begin{aligned}
&\mathrm{HIGH}'^{\delta} :-_{\mathrm{CR}',\mathrm{HIGH}'^{\delta},\mathrm{LOW}'^{\delta},\mathrm{DIF}'^{\delta}}\ \bot;\top;\bot;\mathrm{B}_1;\bot,\\
&\mathrm{LOW}'^{\delta} :-_{\mathrm{CR}',\mathrm{LOW}'^{\delta},\mathrm{HIGH}'^{\delta},\mathrm{DIF}'^{\delta}}\ \bot;\top;\bot;\neg\mathrm{B}_1;\bot.
\end{aligned}$$

如果某个节点的颜色高于其第 δ 个邻居，变量 HIGH'^{δ} 就为真；如果颜色较低，变量 LOW'^{δ} 就为真。真值根据另一端（索引 1）的比特计算，而不是根据进行比较的那一端（索引 2Δ）计算，因为在两者之间的单轮迭代中，这些变量的真值已旋转到另一侧。

与第 8.3 节一样，我们定义一个公式，用来判断某个节点的颜色是否高于其所有邻居：

$$\mathrm{G} :-_{\mathrm{CC},\mathrm{T}^2_{\Delta+1}}\ \bot;\bigwedge_{\delta=1}^{\Delta}\mathrm{HIGH}'^{\delta};\bot.$$

如果某个节点的颜色比它所有邻居的颜色都高，就需要将颜色改为一个邻居都没有使用的最低颜色。为此，我们定义 $\Delta+1$

用于表示节点应改为该颜色的变量：

$$
\begin{aligned}
&\mathrm{L}_1 :-_{\mathrm{T}_1^2} \bigwedge_{\delta=1}^{\Delta} \neg \left(\bigwedge_{i=2}^{\log(\Delta+1)} \neg \mathrm{B}'^{\delta}_i \wedge \mathrm{B}'^{\delta}_1 \right); \bot,\\
&\mathrm{L}_2 :-_{\mathrm{T}_2^2} \bigwedge_{\delta=1}^{\Delta} \neg \left(\bigwedge_{i=3}^{\log(\Delta+1)} \neg \mathrm{B}'^{\delta}_i \wedge \mathrm{B}'^{\delta}_2 \wedge \neg \mathrm{B}'^{\delta}_1 \right) \wedge \neg \mathrm{L}_1; \bot,\\
&\mathrm{L}_3 :-_{\mathrm{T}_3^2} \bigwedge_{\delta=1}^{\Delta} \neg \left(\bigwedge_{i=3}^{\log(\Delta+1)} \neg \mathrm{B}'^{\delta}_i \wedge \mathrm{B}'^{\delta}_2 \wedge \mathrm{B}'^{\delta}_1 \right) \wedge \bigwedge_{\delta=1}^{2} \neg \mathrm{L}_\delta; \bot,\\
&\vdots
\end{aligned}
$$

在对应的计时器变量 T_i^2 变为真的同一迭代轮次中，每个变量 L_i 都会计算其较低索引处的颜色是否为最低可用颜色。在右侧合取中，它们检查所有较低颜色是否都已被占用；在左侧合取中，它们利用比特检查该颜色是否可用。经过 $\Delta+1$ 次迭代后，所有变量都已计算出真值，并且只有一个变量变为真。

最后，我们修订变量 B_i 的定义，以涵盖节点改变颜色的情况（我们用 $\ell' = \log(\Delta+1)$、$f(x) = \lceil \frac{x}{2} \rceil$ 和 $g(x) = x - 2f(x) + 2$ 表示；函数 f 和 g 用于固定某些索引）：

$$
\begin{aligned}
&\mathrm{B}_1 :-_{\mathrm{G},\mathrm{T}_1^2,\mathrm{T}_1^1,\mathrm{END3}} \bigvee_{i=1}^{2^{\ell'-1}} \mathrm{L}_{2i-1}; \mathrm{B}_1; \mathrm{B}_{2\Delta}; \mathrm{C}_1^1 \vee \mathrm{C}_3^1; \bot,\\
&\mathrm{B}_2 :-_{\mathrm{G},\mathrm{T}_1^2,\mathrm{T}_1^1,\mathrm{END3}} \bigvee_{j=1}^{2} \bigvee_{i=1}^{2^{\ell'-2}} \mathrm{L}_{4i-j}; \mathrm{B}_2; \mathrm{B}_1; \mathrm{C}_2^1 \vee \mathrm{C}_3^1; \bot,\\
&\vdots\\
&\mathrm{B}_{\ell'} :-_{\mathrm{G},\mathrm{T}_1^2,\mathrm{T}_1^1,\mathrm{END3}} \bigvee_{j=1}^{2^{\ell'-1}} \mathrm{L}_{2^{\ell'}-j}; \mathrm{B}_{\ell'}; \mathrm{B}_{\ell'-1}; \mathrm{C}_{g(\ell')}^{f(\ell')} \vee \mathrm{C}_3^{f(\ell')}; \bot,\\
&\mathrm{B}_{\ell'+1} :-_{\mathrm{G},\mathrm{T}_1^2,\mathrm{T}_1^1,\mathrm{END3}} \bot; \mathrm{B}_{\ell'+1}; \mathrm{B}_{\ell'}; \mathrm{C}_{g(\ell'+1)}^{f(\ell'+1)} \vee \mathrm{C}_3^{f(\ell'+1)}; \bot,\\
&\vdots\\
&\mathrm{B}_{2\Delta} :-_{\mathrm{G},\mathrm{T}_1^2,\mathrm{T}_1^1,\mathrm{END3}} \bot; \mathrm{B}_{2\Delta}; \mathrm{B}_{2\Delta-1}; \mathrm{C}_2^{\Delta} \vee \mathrm{C}_3^{\Delta}; \bot.
\end{aligned}
$$

在节点改变颜色的轮次中，如果某个节点没有最高颜色，其比特保持不变。如果它具有最高颜色，那么它会变为介于 1 和$\Delta+1$之间的某种颜色。这意味着从 $\ell'+1$开始的比特变为 0，相应的变量也变为假。其余 ℓ' 个变量的真值根据节点变成的颜色计算：

- 最右位为 1 的颜色有 $[1], [3], [5], \ldots$.
- 右起第二位为 1 的颜色有 $[2,3], [6,7], [10,11], \ldots$.
- 右起第三位为 1 的颜色是 $[4,5,6,7]$, $[12,13,14,15]$,$[20,21,22,23]$、.....。

• 右起第 k 位为 1 的颜色是颜色 $2^k i - j$，其中 $i \in \mathbb{Z}_+$ 且$j \in [k]$。

最后，我们为每种可能的最终颜色定义一个 $\Delta + 1$ 指定谓词：

$$\begin{aligned}
\mathrm{CLR}_1 &:-_{\mathrm{END4}} \bigwedge_{i=2}^{\log(\Delta+1)} \neg \mathrm{B}_i \wedge \mathrm{B}_1; \bot, \\
\mathrm{CLR}_2 &:-_{\mathrm{END4}} \bigwedge_{i=3}^{\log(\Delta+1)} \neg \mathrm{B}_i \wedge \mathrm{B}_2 \wedge \neg \mathrm{B}_1; \bot, \\
\mathrm{CLR}_3 &:-_{\mathrm{END4}} \bigwedge_{i=3}^{\log(\Delta+1)} \neg \mathrm{B}_i \wedge \mathrm{B}_2 \wedge \mathrm{B}_1; \bot, \\
&\vdots
\end{aligned}$$

有了这些指定谓词，定理 6.3 现在成立：

► **定理 6.3**。存在一个MPMSC 公式，具有以下性质。

1. 它定义了一种合法的（$\Delta + 1$）着色。*2.* 它有 $\mathcal{O}(\Delta \log(n)) + \mathcal{O}(3^\Delta)$ 个头部，大小为 $\mathcal{O}(\Delta^2 \log(n)) + \mathcal{O}(3^\Delta)$。*3.* 所需的迭代次数为 $\mathcal{O}(\log(n) \log\log(n) \log^*(n)) + \mathcal{O}(\Delta 3^\Delta)$。*4.* 恰有 $\log^*(n) + 10 + 3^\Delta - \Delta$ 个全局通信轮次。

证明。 1) 我们有 $\Delta + 1$ 个指定谓词（注意和打印谓词）：$\mathrm{CLR}_1, \ldots, \mathrm{CLR}_{\Delta+1}$。它们分别对应变量 $\mathrm{B}_1, \ldots, \mathrm{B}_{\ell'}$ 的不同组合。这使得打印谓词两两互斥，也就是说，在给定节点中至多一个打印谓词为真。每个节点中至少有一个变量 B_i 为真；而当节点输出时，变量 $\mathrm{B}_{\ell'+1}$ 及其后续变量在每个节点中都已变为假，这意味着每个节点中至少有一个打印谓词变为真。因此，每个节点输出一个（$\Delta+1$）位串，其中恰有一位为 1，而该程序定义了一种（$\Delta+1$）着色。

如果两个邻居的输出相同，那么它们对变量 $\mathrm{B}_1, \ldots, \mathrm{B}_{\ell'}$ 的真值也相同。它们对变量 $\mathrm{B}_{\ell'+1}$ 及之后的变量的真值也相同，因为这些变量在每个节点中都已变为假。这意味着这些邻居会拥有相同的颜色，而这与程序的规定相矛盾。因此，任意两个邻居的输出都不相同，着色是合法的。

2) 根据引理 8.1，我们已有 $\mathcal{O}(\Delta \log(n))$ 个头部，程序的大小为 $\mathcal{O}(\Delta^2 \log(n))$。现在把新增的头部及其大小加进去。

• 我们有 $\mathcal{O}(\Delta)$ 个变量 T^i_j, CC, CR′, B'^{δ}_i, DIF'^{δ}, HIGH'^{δ}, LOW'^{δ}, G, L_i, B_i 和 CLR_i，大小为 $\mathcal{O}(\Delta^2)$。

• 我们有 $\mathcal{O}(3^\Delta)$ 个变量 LP_i，大小为 $\mathcal{O}(1)$。

将这些头部与程序先前的部分合并，得到

$$\mathcal{O}(\Delta \log(n)) + \mathcal{O}(\Delta) + \mathcal{O}(3^\Delta) = \mathcal{O}(\Delta \log(n)) + \mathcal{O}(3^\Delta).$$

将头部乘以其大小，得到

$$\begin{aligned}
&\mathcal{O}(\Delta^2 \log(n)) + \mathcal{O}(\Delta)\mathcal{O}(\Delta^2) + \mathcal{O}(3^\Delta)\mathcal{O}(1) \\
&= \mathcal{O}(\Delta^2 \log(n)) + \mathcal{O}(\Delta^3) + \mathcal{O}(3^\Delta) \\
&= \mathcal{O}(\Delta^2 \log(n)) + \mathcal{O}(3^\Delta).
\end{aligned}$$

3) 前一阶段耗费了 $\mathcal{O}(\log(n)\log\log(n)\log^*(n))$ 次迭代。基本颜色归约阶段需要近 3^Δ 次重复，每次重复包含若干次 Δ 迭代。迭代总数为

$$\mathcal{O}(\log(n)\log\log(n)\log^*(n)) + \mathcal{O}(\Delta)\mathcal{O}(3^\Delta) = \mathcal{O}(\log(n)\log\log(n)\log^*(n)) + \mathcal{O}(\Delta 3^\Delta).$$

4) 程序中先前需要菱形的迭代次数为 $\log^*(n) + 12$。在此阶段，每当变量 CR′ 为真时，就会使用一个菱形；这在每次循环中发生一次。循环次数为 $3^\Delta - (\Delta + 1)$，与算法的通信轮次相同。这意味着全局通信轮次总数为 $\log^*(n) + 3^\Delta - \Delta + 11$。

□

如果假设 Δ 是常数，那么头部数量和程序大小为 $\mathcal{O}(\log(n))$，所需的迭代次数为 $\mathcal{O}(\log(n)\log\log(n)\log^*(n))$，全局通信轮次为 $\log^*(n) + \mathcal{O}(1)$。