arXiv:2603.13058v1 [cs.DS] 13 Mar 2026

对 SLP 压缩字符串上的 MSO 查询求值进行动态直接（按排名）访问

Martín Muñoz 阿尔图瓦大学，法国国家科学研究中心，UMR 8188，朗斯计算机科学研究中心（CRIL），F-62300，法国

摘要

我们提出一种算法：给定索引 t，可生成字符串上某个 MSO 查询按字典序排列的第 t 个答案。该算法需要线性时间预处理，并构建一种数据结构，使每次此类调用都能在对数时间内得到答案。随后，我们展示如何将该算法扩展到由直线程序（SLP）压缩的字符串，同样只需对字符串的（压缩编码）进行线性时间预处理，并能在对数时间内直接访问原始字符串。最后，我们进一步扩展该算法，允许在直接访问数据结构完成处理后对 SLP 进行复杂编辑，并在对数时间内将这些编辑反映到数据结构中。为此，我们改造了 Schmid 和 Schweikardt（PODS 2022）提出的文档编辑框架。与近期关于字符串上 MSO 查询动态直接访问的结果（Bourhis 等人，ICDT 2025）相比，本工作将访问过程的时间复杂度提升了一个对数因子，并将结果扩展到了直线程序。

2012 年 ACM 学科分类计算理论 → 逻辑与数据库；计算理论 → 数据管理的数据结构与算法；计算理论 →自动机扩展

关键词与短语直接访问、单子二阶逻辑、增量求值、文本压缩、压缩文本上的算法

1 引言

基于单子二阶逻辑（MSO）公式构建的查询求值，是数据库理论和形式语言算法学中一项研究充分的任务。其原因之一是，它可以作为通向其他形式体系的桥梁。具体而言，根据 Büchi-Elgot-Trakhtenbrot 定理 [9], ，众所周知，单词上的 MSO 等价于有限状态自动机和正则表达式。这种等价关系也扩展到树：树上的 MSO 查询等价于树正则语言和可见下推自动机。这项任务的适用范围还不止于此：正则表达式的表述意味着 MSO 求值可用于字符串处理；树的表述意味着它可用于 XML 处理；尤为相关的是，MSO 可用于处理正则跨度器的求值 [14, 7, 23]。

基本的求值任务是：给定一个 MSO 公式 φ和一个数据字符串 w， $w \models \varphi$是否成立？（将 w 视为通常意义上的有序单子结构）。这一任务的自然扩展是改为通过带有自由变量 $\varphi(x_1$、 ..... 、 $x_k)$ 的公式进行查询，并计算满足 $w \models \varphi(\mu(x_1)$..... 、 $\mu(x_k))$ 的赋值 μ。

自然地，通过这一扩展，答案集可能会比输入实例大得多。为了解决这一问题，大量研究探讨了如何在不一次性生成所有答案的情况下提取答案。在 MSO 求值方面，有研究探索了逐个枚举这些答案的任务 [3, 12, 1]；也有研究关注排名枚举，即必须按顺序生成答案 [7]；还有直接访问，即按照任意索引 [4, 17]访问答案集。

我们识别出文献中两项共用同一名称的不同任务。第一项是随机访问：对于答案集合 S，目标是构建一个介于1与 $|S|$ 之间的索引，使得每个索引都从该集合中产生不同的答案。第二项是排序访问：任务相同

但答案上存在一种顺序，且索引遵循该顺序。文献中的命名存在差异，这两项任务也都被命名为直接访问 [11, 8, 10, 5]。

我们强调这一事实：正如 [11], 所指出的，直接随机访问可以模拟均匀采样和枚举；而直接排序访问则涵盖分位数查询（例如答案的中位数）、排名枚举和 Top-k 求答。

除了求值任务外，一些研究还涉足增量求值领域。换言之，我们允许在数据针对查询处理完毕后对其进行更改。对于 MSO 查询，这方面已经在枚举 [2, 19, 21, 30, 24]中得到了广泛研究。然而，据我们所知，只有一项工作 [6] 研究了可编辑字符串上 MSO 的直接访问。

本文研究了一种在 MSO 查询上实现高效直接排序访问的算法。此外，我们还考虑了一个更复杂的设置：字符串可能已被压缩为具有单元素语言 [28]的文法。这种形式的编码通常称为直线程序（SLP） [18]；它们具有良好的压缩界，并具备吸引人的算法性质 [5, 25, 26, 27, 20]。关于 SLP 压缩字符串（以及树）上的 MSO 查询求值，已有研究从枚举 [29, 21, 30, 24], 角度进行了探讨；但我们认为，本文是首项从直接访问角度研究这一问题的工作。

背景 本工作改进了 [6] 中的一项结果：一种用于字符串上 MSO 查询的动态直接访问算法。该结果在输入字符串的长度上实现了对数平方访问和对数更新。在本工作中，我们将对数平方访问降低为单个对数因子，同时保持对数更新。此外，我们还将结果扩展到 SLP 压缩字符串；由于数据结构的构造，这一扩展自然地呈现出来。我们借鉴了该工作的多种概念和见解。

论文结构 在第2节中，我们介绍一些已知概念及其简单扩展。在第3节中，我们给出本文的主要结果之一，即用于字符串上 MSO 查询的直接访问算法。在第4节中，我们将这一结果扩展到直线程序；在第5节中，我们说明如何处理编辑。最后，我们在第6节中作简要总结。

由于篇幅所限，我们将至少一个证明放在附录中。对于我们认为可直接推出的结论，将不附证明。

2 预备知识

令 $\mathbb{N} = \{0, 1, ...\}$。给定集合 A, B，我们用记号 B^A 表示函数 $f\colon A \to B$的集合。特别地，对于集合 A，我们用 $\mathbb{N}^{A\times A}$ 表示以对 $(a_1, a_2) \in A \times A$为索引的自然数矩阵。我们还用记号 I_A 表示 $A \times A$ 单位矩阵；即，当 $a_1 = a_2$时，$I_A(a_1, a_2) = 1$；当 $a_1 \neq a_2$时，$I_A(a_1, a_2) = 0$。

我们使用 $[1, n] = \{1, \dots, n\}$。我们用符号 $<$表示 $\mathbb{N}$上的标准序。如果集合 A 在某个序 $\preccurlyeq$下是全序的，我们将用“切片”表示一个子集 $B \subseteq A$，它可以由两个元素 $a_l, a_r \in A$ 描述，使得 $B = \{a \mid a_l \preccurlyeq a \preccurlyeq a_r\}$。

我们用 $\omega < 3$ 表示二进制矩阵乘法的指数 [31]。

字符串与映射给定一个有限字母表 Σ，Σ 上的字符串（或简称字符串）是一个序列 $w = a_1 \dots a_n \in \Sigma^*$，其中 Σ^* 表示由Σ中的元素组成的有限序列的集合。对于这样的 w，我们将其长度定义为 n，记作 $|w|$。长度为零的字符串记为 ε，对于 $\Sigma^* \setminus \{\varepsilon\}$，我们记作 Σ^+。对于字符串 w_1 和 w_2，我们用 $w_1 \cdot w_2$表示它们的拼接。对于字符串 $w = a_1 \dots a_n$，任意给定 $1 \leq i \leq j \leq n$，我们使用记号 $w[i, j] = a_i \dots a_j$，并且 $w[i] = w[i, i] = a_i$。

固定一个 n。给定一个有限变量集 $X = \{x_1, \ldots, x_k\}$，从 X 到 w的映射是一个函数 $\mu : X \to [1, n]$。或者，我们也可以将映射定义为形如 $\{x_1 \mapsto s_1, \ldots, x_k \mapsto s_k\}$ 的集合，其中 $s_1 \ldots, s_k \in [1, n]$。部分映射是映射的任意子集。部分映射也可以理解为定义域为 X的部分函数，其中全函数就是映射。固定 X 上的一个顺序 $\prec$，使得 $x_1 \prec \cdots \prec x_k$。我们将此顺序扩展到映射上如下：设 $\mu_1, \mu_2 : X \to [1, n]$ 是两个映射，则 $\mu_1 \prec \mu_2$ 当且仅当 $\mu_1(x_1) = \mu_2(x_1) \ldots, \mu_1(x_{i-1}) = \mu_2(x_{i-1})$，并且对于某个 $i \in [1, k]$，有 $\mu_1(x_i) \prec \mu_2(x_i)$。

给定一个 $x \in X$ 和 $\alpha \subseteq [1, n]$，我们将 $x \Mapsto \alpha$ 称为一个映射规则。如果对于每条规则 $x \Mapsto \alpha \in \tau$ 都满足 $\mu(x) \in \alpha$，则称一个（部分）映射 $\mu : X \to [1, n]$ 遵循一组映射规则 τ。当 α 是单例集合时，我们可以简化记法：不写作$x \Mapsto \{s\}$，而写作 $x \mapsto s$。对于固定的 X，如果映射规则提及每个$x \in X$，则称其为函数式。如果每个映射都遵循 τ_1 当且仅当其遵循 τ_2，则两组映射规则 τ_1、τ_2 称为等价的。注意，对于每组映射规则，都存在一组与之等价的函数式映射规则。

变量集自动机给定一个固定的变量集合 X，变量集自动机是一个元组$\mathcal{A} = (\Sigma, Q, \Delta, I, F)$，其中 Σ是一个有限字母表，Q 是一个状态集合 $\Delta \subseteq Q \times (\Sigma \times 2^X) \times Q$ 是转移集合，而 I、$F \subseteq Q$ 分别是初始状态和终止状态的集合。一个运行 ρ $\mathcal{A}$ 在某个字符串 $w = a_1 \ldots a_n$ 上的运行是一系列

$$\rho = q_0 \xrightarrow{a_1, S_1} q_1 \xrightarrow{a_2, S_2} \cdots \xrightarrow{a_n, S_n} q_n$$

其中 $q_0 \in I$和 $(q_{i-1}, (a_i, S_i), q_i) \in \Delta$ 对每个 $i \in [1, n]$均成立。我们称 ρ是有效的，如果$S_1 \cup \cdots \cup S_n = X$，并且对每个 $i, j \in [1, n]$，都有 $S_i \cap S_j = \emptyset$。换言之，$S_1, \ldots, S_n$ 构成 X 的一个划分。对于有效的 ρ，我们提取映射 μ_ρ，其定义为 $\mu_\rho(x) = i$，其中 S_i 是包含 x的唯一集合。我们还称 ρ 是接受的，如果 $q_n \in F$。变量集自动机的语义由一个函数给出

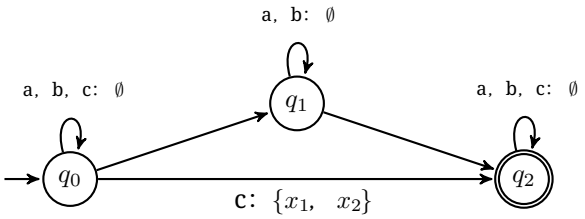
$$[\![\mathcal{A}]\!](w) := \{\mu_\rho \mid \rho \text{ 是} \mathcal{A} \text{ 在 } w\text{上的一个有效接受运行}\}.$$

给定 $w = a_1 \ldots a_n$和 $l, r \in [1, n]$，其中 $l \leq r$，我们称 ρ'是一个在 w 上的 (l, r)-部分运行，如果存在如上所述在 w 上的运行 ρ，且 $\rho' = q_{l-1} \xrightarrow{a_l, S_l} q_l \xrightarrow{a_{l+1}, S_{l+1}} \cdots \xrightarrow{a_r, S_r} q_r$。从一个 (l, r)-部分运行 ρ'中，我们提取部分映射 $\mu_{\rho'} := \{x \mapsto i \mid \rho' \text{ 具有一个 } S_i \ni x\}$。

与 [6], 中一样，我们讨论变量集自动机的两项限制：如果 $\mathcal{A}$ 的每个接受运行必然都是有效的，就称 $\mathcal{A}$ 为函数式；如果对于任意两个满足 $\mu_{\rho_1} = \mu_{\rho_2}$ 的运行 ρ_1 和 ρ_2，都有 $\rho_1 = \rho_2$，就称 $\mathcal{A}$ 为无歧义。注意，对于每个变量集自动机，都存在一个与之等价的函数式无歧义自动机 [13, 22]。下文中，我们只关注同时满足这两项限制的变量集自动机。

我们将 $[\![\mathrm{A}]\!](w)\tau$ 定义为：$= \{\mu \in [\![\mathrm{A}]\!](w) \mid \mu \text{ 满足 } \tau\}$，并用 $\sharp[\![\mathrm{A}]\!](w)\tau$ 表示其大小。一般而言，我们采用的映射限制 τ 会使 $[\![\mathrm{A}]\!](w)\tau$ 成为 $[\![\mathrm{A}]\!](w)$ 的一个切片。也就是说，$[\![\mathrm{A}]\!](w)\tau$ 中的元素按照顺序 $\prec$ 在 $[\![\mathrm{A}]\!](w)$ 中是连续的。

MSO 与变量集自动机。与 [6], 的思路相同，我们将结果表述为字符串上的单子二阶逻辑查询求值：对于字符串上的形如 $\varphi(x_1, \ldots, x_k)$ 的 MSO 公式（其开放的一阶变量为 $x_1, \ldots, x_k$），以及一个被解释为结构的字符串 $w \in \Sigma^*$，我们将 $[\![\varphi]\!](w) := \{\mu : \{x_1, \ldots, x_k\} \to [1, |w|] \mid \varphi(\mu(x_1) \ldots, \mu(x_k))\}$ 定义为在 φ 和 w 上求值所得的答案集。本文中的结果均以变量集自动机来表述，因为对于任意字符串上的 MSO 公式 $\varphi(x_1, \ldots, x_k)$，都存在一个针对 $X = \{x_1, \ldots, x_k\}$ 的变量集自动机 A，使得 $[\![\varphi(x_1, \ldots, x_k)]\!] = [\![\mathrm{A}]\!]$。



x_1	x_2
1	2
1	4
1	6
3	4
3	6
5	6
7	7
8	9

图1 （左）运行示例$\mathcal{A}^{\text{ex}}$ 由自动机所示，以及（右）列表 o f 字符串 w_0 =abababcab 的 $[\![\mathcal{A}^{\text{ex}}]\!](w_0)$ 中的映射。

直线程序。上下文无关文法是一个元组 $G = (N, \Sigma, R, S_0)$，其中 N 是一个非空的非终结符集合，Σ 是有限字母表，$S_0 \in N$ 是开始符号，且 $R \subseteq N \times (N \cup \Sigma)^+$ 是规则集合。按照惯例，规则 $(A, \beta) \in R$ 将写作 $A \to \beta$，并将 Σ 和 N 分别称为终结符号集和非终结符号集。若上下文无关文法 $\mathfrak{S} = (N, \Sigma, R, S_0)$ 满足以下条件：R 是从 N 到 $(N \cup \Sigma)^+$ 的全函数，且有向图 $(N, \{(A, B) \mid (A, \beta) \in R$，其中 B 出现在 β 中 $\})$ 是无环的，则它是一个直线程序（SLP）。对于每个 $A \in N$，令 $R(A)$ 为满足 $(A, \beta) \in R$ 的唯一 $\beta \in (N \cup \Sigma)^+$；对于每个 $a \in \Sigma$，令 $R(a) = a$。我们将 R 递归地扩展为一个态射 $R^* : (N \cup \Sigma)^* \to \Sigma^*$，使得当 u 是 Σ 上的字符串时，$R^*(u) = u$；而 $R^*(\alpha_1 \dots \alpha_n) = R^*\big(R(\alpha_1) \cdot \ldots \cdot R(\alpha_n)\big)$，其中对每个 $i \le n$，$\alpha_i \in (N \cup \Sigma)$。根据我们对 SLP 的定义，$R^*(A)$ 属于 Σ^+，并且对每个 $A \in N$ 都有唯一确定的值。于是，我们将由 S 编码的字符串定义为 $\mathrm{str}(S) = R^*(S_0)$。

若上下文无关文法 $G = (N, \Sigma, R, S_0)$ 中的规则 R 要么具有 $A \to BC$ 的形式，要么具有 $A \to a$ 的形式，其中 $A, B, C \in N$，或者为 $a \in \Sigma$，则称该文法处于乔姆斯基范式。

我们将直线程序 $\mathfrak{S} = (N, \Sigma, R, S_0)$ 的大小定义为 $|\mathfrak{S}| = \sum_{A \in N} |R(A)|$。

例。设 $\mathfrak{S} = (N, \Sigma, R, S_0)$ 是一个直线程序，其中 $N = \{S_0, A, B, C, D, S_{\mathsf{a}}, S_{\mathsf{b}}, S_{\mathsf{c}}\}$，$\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$，且 $R = \{S_0 \to AB, A \to DD, B \to CD, C \to DS_{\mathsf{c}}, D \to S_{\mathsf{a}}S_{\mathsf{b}}, S_{\mathsf{a}} \to \mathsf{a}\ \ S_{\mathsf{b}} \to \mathsf{b}\ \ S_{\mathsf{c}} \to \mathsf{c}\}$。于是有 $\mathrm{str}(D) = \mathsf{ab}$，$\mathrm{str}(C) = \mathsf{abc}$，$\mathrm{str}(B) = \mathsf{abcab}$，$\mathrm{str}(A) = \mathsf{abab}$，且 $\mathrm{str}(\mathfrak{S}) = \mathrm{str}(S_0) = \mathsf{ababababcab}$，即 $\mathfrak{S}$ 所表示的字符串。我们在图 4 中展示 R 的有向无环图（在直线程序主算法的语境下）。

机器模型 处理直线程序 $\mathfrak{S}$时，它所表示的字符串 w 的长度可能为 $2^{|\mathfrak{S}|}$。特别是，仅表示其中的一个索引 w 就可能需要关于 $|\mathfrak{S}|$呈线性数量的比特。

我们通过定义函数 κ 来处理这一问题，使得 $\kappa(T)$描述值和结果均至多为 T的数之间进行一次算术运算的代价。我们将处理表示 $[\![\mathcal{A}]\!](w)$中索引的数，因此我们可能处理的最大值为 $T \le |w|^{|X|}$。我们简单地假设所执行的每次运算的代价为 $\kappa(|w|^{|X|})$，并在给出的界中省略这一因子；在此说明，该因子应被视为隐含的。

注意，对于字符串 w，$\kappa(|w|)$ 可以假定为常数，这符合使用大小为 $\log n$ 的常数开销寄存器这一典型假设。然而，这意味着界中包含一个额外的 $|X|$ 因子，而这一点实际上已在 [6] 中得到妥善处理。尽管如此，我们仍选择将该因子隐含处理。

2.1 运行示例

我们先介绍一个示例，用它来说明该模型的语义，之后还会用它解释直接访问算法。我们定义一个无歧义函数式变量集自动机 $\mathcal{A}^{\mathsf{re}}$，如图1（左）所示，并在字符串$w_0 =$ abababcab 上对其进行求值。我们可以看到，$\mathcal{A}^{\mathsf{re}}$ 在 w_0 上的接受运行（这些运行给出了 $[\![\mathcal{A}^{\mathsf{re}}]\!](w_0)$ 中的前三个元素）为：

$$\rho_1 = q_0 \xrightarrow{\mathsf{a},\{x_1\}} q_1 \xrightarrow{\mathsf{b},\{x_2\}} q_2 \xrightarrow{\mathsf{a},\emptyset} q_2 \xrightarrow{\mathsf{b},\emptyset} q_2 \xrightarrow{\mathsf{a},\emptyset} q_2 \xrightarrow{\mathsf{b},\emptyset} q_2 \xrightarrow{\mathsf{c},\emptyset} q_2 \xrightarrow{\mathsf{a},\emptyset} q_2 \xrightarrow{\mathsf{b},\emptyset} q_2,$$
$$\rho_2 = q_0 \xrightarrow{\mathsf{a},\{x_1\}} q_1 \xrightarrow{\mathsf{b},\emptyset} q_1 \xrightarrow{\mathsf{a},\emptyset} q_1 \xrightarrow{\mathsf{b},\{x_2\}} q_2 \xrightarrow{\mathsf{a},\emptyset} q_2 \xrightarrow{\mathsf{b},\emptyset} q_2 \xrightarrow{\mathsf{c},\emptyset} q_2 \xrightarrow{\mathsf{a},\emptyset} q_2 \xrightarrow{\mathsf{b},\emptyset} q_2,$$
$$\rho_3 = q_0 \xrightarrow{\mathsf{a},\{x_1\}} q_1 \xrightarrow{\mathsf{b},\emptyset} q_1 \xrightarrow{\mathsf{a},\emptyset} q_1 \xrightarrow{\mathsf{b},\emptyset} q_1 \xrightarrow{\mathsf{a},\emptyset} q_1 \xrightarrow{\mathsf{b},\{x_2\}} q_2 \xrightarrow{\mathsf{c},\emptyset} q_2 \xrightarrow{\mathsf{a},\emptyset} q_2 \xrightarrow{\mathsf{b},\emptyset} q_2,$$

因此，$\mu_{\rho_1} = \{x_1 \mapsto 1, x_2 \mapsto 2\}$、$\mu_{\rho_2} = \{x_1 \mapsto 1, x_2 \mapsto 4\}$ 和 $\mu_{\rho_3} = \{x_1 \mapsto 1, x_2 \mapsto 6\}$。完整的映射集合 $[\![\mathcal{A}^{\mathsf{re}}]\!](w_0)$ 可见图1（右）。

3 字符串上的单子二阶逻辑查询直接访问

本节将专门用于证明本文的第一个主要结果。

▶**定理1**。 固定一个变量集合 X。设 A 是一个无歧义*vset*自动机，且$w \in \Sigma^*$。

1. 对于 X 上的固定顺序 $\prec$，经过 $O(|Q|^\omega \cdot |X| \cdot |w|)$ 预处理后，我们可以在 $O(|Q|^\omega \cdot |X|^2 \cdot \log(|w|))$ 时间内直接（排名）访问集合 $[\![A]\!](w)$。*2.* 如果在访问阶段将 $\prec$ 作为输入给出，则该问题需要 $O(|Q|^\omega \cdot 2^{|X|} \cdot |w|)$ 的预处理时间，且直接（排名）访问的界限不变。

解决方案结构。首先，我们将介绍一个通用过程，用于检索集合 $[\![\mathcal{A}]\!](w)$ 的第 t 个答案；该过程假设我们能够根据 $\mathcal{A}$ 和 w 计算某些数据。接着，我们定义一个矩阵集合，用以描述所需的逻辑信息，从而能够检索这些数据。然后，我们将介绍一种数据结构，用于存储这些矩阵的精简版本。最后，我们将介绍实际的访问阶段，该阶段使用缓存策略，在同一遍对数时间扫描中遍历搜索空间并计算相关数据。

在本节中，我们将重点证明第 (1) 项，即顺序 $\prec$在 X上是全局固定的。我们假设 $X = \{x_1, \ldots, x_k\}$ 且 $x_1 \prec \ldots \prec x_k$。本节末尾，我们将说明如何调整该解法以处理第 (2) 项。

3.1 访问阶段模板

令 t 为查询索引，并令 μ^* 为集合 $[\![\mathcal{A}]\!](w)$ 中的第 t 个映射。访问阶段包括依次逐个计算值 $\mu^*(x_1) \ldots, \mu^*(x_k)$。

访问阶段见算法 1。基本思路是通过二分搜索找到 $\mu^*(x_1)$；然后通过假设值 $\mu^*(x_1)$ 固定不变的二分搜索找到 $\mu^*(x_2)$；接着通过假设 $\mu^*(x_1)$ 和 $\mu^*(x_2)$ 固定不变的二分搜索找到 $\mu^*(x_3)$；依此类推。第一次二分搜索通过将 $\mu^*(x_1)$ 表示为满足以下条件的值 s_1 来查找它

$$\sharp[\![\mathcal{A}]\!](w)\langle x_1 \Mapsto [1, s_1 - 1]\rangle < t \leq \sharp[\![\mathcal{A}]\!](w)\langle x_1 \Mapsto [1, s_1]\rangle.$$

在该算法中，我们假设可以访问一个函数 $\mathsf{mid} : [1, n] \times [1, n] \to [1, n]$，它为每个范围 (l, r) 指定一个中点 $m := \mathsf{mid}(l, r)$；目前，我们只假设该函数在访问阶段开始时已固定，且 $l \leq m < r$。

算法 1 用于在 $[\![\mathcal{A}]\!](w)$ 中检索第 t 个元素的简化算法。

过程 访问($\mathcal{A}$ w, t)
  声明 $s_1, \ldots, s_k$
  对 $i \in [1, k]$ **执行**
    $(l,\ r) \leftarrow (1,\ n)$
    当 $l < r$ **时执行**
      $m \leftarrow \mathsf{mid}(l, r)$
      如果 $t \leq \sharp[\![A]\!](w)\langle x_1 \mapsto s_1, \ldots, x_{i-1} \mapsto s_{i-1}, x_i \Mapsto [1, m]\rangle$ **那么**
        $(l,\ r) \leftarrow (l,\ m)$
      else
        $(l, r) \leftarrow (m + 1, r)$
    $s_i \leftarrow l$
    $t \leftarrow t - \sharp[\![A]\!](w)\langle x_1 \mapsto s_1, \ldots, x_{i-1} \mapsto s_{i-1}, x_i \Mapsto [1, s_i - 1]\rangle$
  返回 $\{x_1 \mapsto s_1, \ldots, x_k \mapsto s_k\}$

算法中的二分搜索会迭代一个范围 (l, r)，该范围初始为 $(1, n)$，并始终保证包含目标值 s。作为不变量，我们维持不等式$\sharp[\![\mathsf{A}]\!](w)x_1 \Mapsto [1, l] < t \leq \sharp[\![\mathsf{A}]\!](w)x_1 \Mapsto [1, r]$。每次迭代都会决定将 (l, r) 缩小到其前半部分 (l, m)，还是后半部分 $(m + 1, r)$，其中 $m = \mathsf{mid}(l, r)$；这通过统计 $[\![\mathsf{A}]\!](w)$ 中有多少个答案映射 x_1 到范围 $(1, m)$ 来实现。当范围大小为 1（即 $l = r$）时，s_1 只有一个可能的取值，因此我们将其赋为 $\mu^*(x_1)$。然后，从 t 中减去映射 x_1 到小于 s_1 的值的答案数量，并继续查找$\mu^*(x_2)$。从 t 中减去该值后，我们便可以聚焦于 $[\![\mathsf{A}]\!](w)$ 中假定 x_1 映射到 s_1 的切片。其余部分的计算方式相同。

在我们的运行示例中，可以在图1（右侧）中看到 $[\![\mathsf{A}^{\mathsf{ex}}]\!](w_0)$ 的完整映射列表。如果访问索引为 $t = 5$，那么我们会找到 $s_1 = 3$，因为 $\sharp[\![\mathsf{A}^{\mathsf{ex}}]\!](w_0)x_1 \Mapsto [1, 3 - 1] = 3 < 5$ 且 $\sharp[\![\mathsf{A}^{\mathsf{ex}}]\!](w_0)x_1 \Mapsto [1, 3] = 5 \leq 5$。接着，在下一次迭代之前，我们需要从 t 中减去值 $\sharp[\![\mathsf{A}^{\mathsf{ex}}]\!](w_0)x_1 \Mapsto [1, 3 - 1] = 3$，并使用 $t = 2$ 进行二分搜索。这是因为搜索空间已缩减为仅包含以 $x_1 \mapsto 3$ 开头的映射。最后，我们找到 $s_2 = 6$，因为 $\sharp[\![\mathsf{A}^{\mathsf{ex}}]\!](w_0)x_1 \mapsto 3, x_2 \Mapsto [1, 6 - 1] = 1 < 2$且 $\sharp[\![\mathsf{A}^{\mathsf{ex}}]\!](w_0)x_1 \mapsto 3, x_2 \Mapsto [1, 6] = 2 \leq 2$，因此 $\mu^* = \{x_1 \mapsto 3, x_2 \mapsto 6\}$，

►**引理 2.** 算法 *1* 在 $[\![\mathcal{A}]\!](w)$ 中找到第 t 个答案。

通用算法带给我们如下启示：$[\![\mathcal{A}]\!](w)$ 中相关的切片，是由针对 $x_1, \ldots, x_{i-1}$ 的固定赋值，以及将 x_i 映射到某个 m 的范围 $[1, m]$ 内某处的一条规则所定义的。这对我们的算法至关重要。

该过程的运行时间取决于 mid 的选择，以及计算这些相关切片大小所需的时间。我们将留待稍后分析。

相关文献。 这一思路是将变量的一个前缀固定为具体值，然后计算相应切片的大小 $[\![\mathcal{A}]\!](w)$，它往往是直接访问 [8, 10] 的核心子程序。我们认为，从概念上讲，我们方案最重要的改进是在核心子程序中加入“将前缀中的下一个变量映射到一个范围”这一规则。

算法2 对 $[\![\mathcal{A}]\!](w)$ 进行直接访问的预处理阶段。

```
过程 预处理(A w)
    声明 T_1, ....., T_k
    对于 i ∈ [0,k], 执行
        构建(T_i, 1, n)
过程 构建(T_i, l, r)
    变量位于 T_i⟨l,r⟩ ← {x_{i+1}, .., x_k}
    如果 l = r 则
        a ← w[l]
        T_i⟨l,r⟩ :: M ← Δ^{x_{i+1}, ..., x_k}_a
        返回 M
    else
        m ← mid(l, r)
        M_1 ← 构建(T_i, l, m)
        M_2 ← 构建(T_i, m+1, r)
        T_i⟨l,r⟩ :: M ← M_1 · M_2
        返回 M
```

3.2 $\mathcal{M}$矩阵

为了弥合 $[\![\mathcal{A}]\!](w)$ 的相关切片与我们将要介绍的数据结构之间的差距，我们在 $\mathbb{N}^{Q\times Q}$ 中引入了一个有用的矩阵集合。

首先，我们为映射规则及其集合添加一些新特性。我们称映射规则 $x \Mapsto \alpha$ *inside* $[l,r]$ 当且仅当 $\alpha \subseteq [l,r]$。此外，我们将复合函数集 τ_1, τ_2（由映射规则组成），并定义运算 $\tau_1 \cdot \tau_2 := \{ x \Mapsto \alpha \cup \beta \mid x \Mapsto \alpha \in \tau_1$ 和 $x \Mapsto \beta \in \tau_2\}$。注意，如果 τ_1 是包含于 $[l,m]$ 内的映射规则集合，并且 τ_2 是包含于 $[m+1,r]$ 内的映射规则集合，那么 $\tau_1 \cdot \tau_2$ 是包含于 $[l,r]$ 内的映射规则集合。

对于 $1 \le l \le r \le n$，以及包含于 $[l,r]$ 内的映射规则集合 τ，我们在 $\mathbb{N}^{Q\times Q}$ 中定义该矩阵：

$$\mathcal{M}\langle l,r:\ \tau\rangle(p,q) := |\{\rho \mid \rho \text{是从 } p \text{ 到 } q\text{、满足 } \tau \text{ 的 } \mathcal{A} \text{ 在 } w \text{ 上的 } (l,r)\text{-部分运行。}\}|$$

这些矩阵背后的原理是根据三个标准，对构成 $[\![\mathcal{A}]\!](w)$ 中映射的部分映射进行分组：部分映射所来源于的 $[1,n]$ 内的范围、 X 中变量可以出现于 $[1,n]$ 中的位置，以及界定各自部分运行的状态。

在我们的运行示例中，我们突出显示了运行 ρ_1, ρ_2, ρ_3 ，它们分别对应 $[\![\mathsf{A}^{\mathsf{ex}}]\!](w_0)$ 中的前三个映射。让我们特别来看一个 M 矩阵：令 $\tau_{x_1=1} := \{x_1 \mapsto 1\}$，并令 $(l,r) = (1,5)$。于是我们发现 $\mathbf{M}^{1,5}_{\tau_{x_1=1}}(q_0,q_1) = 1$，因为在 w_0 上存在一个从 q_0 到 q_1 的 $(1,5)$-部分运行（可由 ρ_3 得到）；此外，$\mathbf{M}^{1,5}_{\tau_1}(q_0,q_2) = 2$，因为它统计了由 ρ_1 和 ρ_2 得到的 $(1,5)$-部分运行。

这些矩阵让我们能够通过一次组合概括大量计数逻辑：

▶ **引理3.** 设 $1 \le l \le m < r \le n$，设 τ_L 为包含在 $[l,m]$中的函数式映射规则集，且设 τ_R 为包含在 $[m+1,r]$中的函数式映射规则集。那么：

$$\mathcal{M}\langle l,r:\tau_L\cdot\tau_R\rangle = \mathcal{M}\langle l,m:\tau_L\rangle \cdot \mathcal{M}\langle m+1,r:\tau_R\rangle$$

最后，我们注意到，对于任意τ，只要$\mathcal{A}$是函数式且无歧义的，等式 $\sharp[\![\mathcal{A}]\!](w)\langle\tau\rangle = \sum_{p\in I,\ q\in F}\mathcal{M}\langle 1,n:\ \tau\rangle(p,q)$ 成立。

3.3 预处理阶段

现在介绍该算法所用的数据结构，其核心思想是有策略地将部分$\mathcal{M}$ 矩阵显式表示。接下来，我们确定这些矩阵，并说明为何

$\Delta^{\{x_1,x_2\}}_{\mathsf{a}}$	q_0	q_1	q_2
q_0	1	1	0
q_1	0	1	0
q_2	0	0	1

$\Delta^{\{x_1,x_2\}}_{\mathsf{b}}$	q_0	q_1	q_2
q_0	1	0	0
q_1	0	1	1
q_2	0	0	1

$\Delta^{\{x_1,x_2\}}_{\mathsf{c}}$	q_0	q_1	q_2
q_0	1	0	1
q_1	0	0	0
q_2	0	0	1

$\Delta^{\{x_2\}}_{\mathsf{a}}$	q_0	q_1	q_2
q_0	1	**0**	0
q_1	0	1	0
q_2	0	0	1

$\Delta^{\{x_2\}}_{\mathsf{b}}$	q_0	q_1	q_2
q_0	1	0	0
q_1	0	1	1
q_2	0	0	1

$\Delta^{\{x_2\}}_{\mathsf{c}}$	q_0	q_1	q_2
q_0	1	0	**0**
q_1	0	0	0
q_2	0	0	1

$\Delta^{\emptyset}_{\mathsf{a}}$	q_0	q_1	q_2
q_0	1	0	0
q_1	0	1	0
q_2	0	0	1

$\Delta^{\emptyset}_{\mathsf{b}}$	q_0	q_1	q_2
q_0	1	0	0
q_1	0	1	**0**
q_2	0	0	1

$\Delta^{\emptyset}_{\mathsf{c}}$	q_0	q_1	q_2
q_0	1	0	0
q_1	0	0	0
q_2	0	0	1

图 2 将$\mathcal{A}^{\mathsf{ex}}$ 限制到不同集合 $Y \subseteq X$后得到的转移矩阵。蓝色粗体数值标出了某个矩阵与其上方矩阵之间唯一的差异

它们足以计算访问阶段中相关的 $[\![\mathcal{A}]\!](w)$所有切片的大小。

我们定义一组$\{\mathbb{T}_Y\}_{Y\subseteq X}$ 二叉树，每棵树都有 $2\cdot n-1$ 个节点，并具有以下递归结构（所有树的结构都相同）：对于一个 $\mathbb{T}\in\{\mathbb{T}_Y\}_{Y\subseteq X}$，

- $\mathbb{T}$ 中的每个节点都由一对 l,r 编号，使得 $1\le l\le r\le n$；$\mathbb{T}$ 的根节点编号为 $\mathbb{T}1,n$；
- 每个满足 $l<r$ 的节点 $\mathbb{T}l,r$ 都有一个左子节点 $\mathbb{T}l,m$ 和一个右子节点 $\mathbb{T}m+1,r$，其
- 中$m:=\mathrm{mid}(l,r)$；并且当 $l=r$ 时，节点 $\mathbb{T}l,r$ 是一个叶节点。

再次假设存在一个固定函数 mid，它为 (l,r) 指定一个中点 m to，使得 $l\le m<r$。如果 (l,r) 可以通过使用该 mid 执行上述递归过程推导得出，我们就称其为有效下标。对 mid 的唯一要求是强平衡：使用这一 mid 计算出树 $\mathbb{T}$ 后，对于每个节点 $\mathbb{T}l,r$，l,m 的高度 h_1 与 $m+1,r$ 的高度 h_2 必须满足 $h_1-h_2\in\{-1,0,1\}$。我们选择针对一般函数 mid 描述该算法，以便在直线程序情形（第 4 节）中重用其逻辑。对于字符串，只需定义 $\mathrm{mid}(l,r)=\lfloor\frac{l+r}{2}\rfloor$。

特别是，存储在 $\mathbb{T}_Y$ 中的信息可以通过对 $\mathcal{A}$ 的转移集施加限制来描述。对于 $Y\subseteq X$ 和 $a\in\Sigma$，定义 $\Delta_a^Y:=\{(p,\ (a,\ S)\ q)\in\Delta\mid S\subseteq Y$ 对于某个 $p,\ q\in Q\}$。我们也将此记法扩展为表示该集合的邻接矩阵。在图 2 中，我们列出了自动机 $\mathcal{A}^{\mathsf{ex}}$ 在 Y 值 $\{x_1,\ x_2\}$、$\{x_2\}$ 和 $\emptyset$ 下对应的这些矩阵。

现在来描述每个 $\mathbb{T}_Y$ 中各节点所存储的数据。除了子节点（通过使用索引 $\langle l,\ m\rangle$ 和 $\langle m+1,\ r\rangle$计算 $m\leftarrow\mathrm{mid}(l,\ r)$ 来访问）之外，每个 $\mathbb{T}_Y\langle l,\ r\rangle$ 还包含一个矩阵 $M\in\mathbb{N}^{Q\times Q}$，记作 $\mathbb{T}_Y\langle l,\ r\rangle::M$，以及一个变量集合，其初始化为 Y，并在算法中以“vars in $\mathbb{T}_Y\langle l,\ r\rangle$”表示。

存储在 $\mathbb{T}_Y\langle l,r\rangle$ 中的矩阵 M 递归定义如下：如果 $l<r$，则令其左子节点和右子节点中的矩阵分别为 M_1 和 M_2，并赋值为 $M\leftarrow M_1\cdot M_2$；否则，如果 $l=r$，则将 M 赋值为 Δ_a^Y，其中 $a=w[l]$。对于运行示例中的树，我们在图3中列出了其中两棵。

该过程在算法2中有所描述，不过存在一个细微之处。为了证明第（1）项

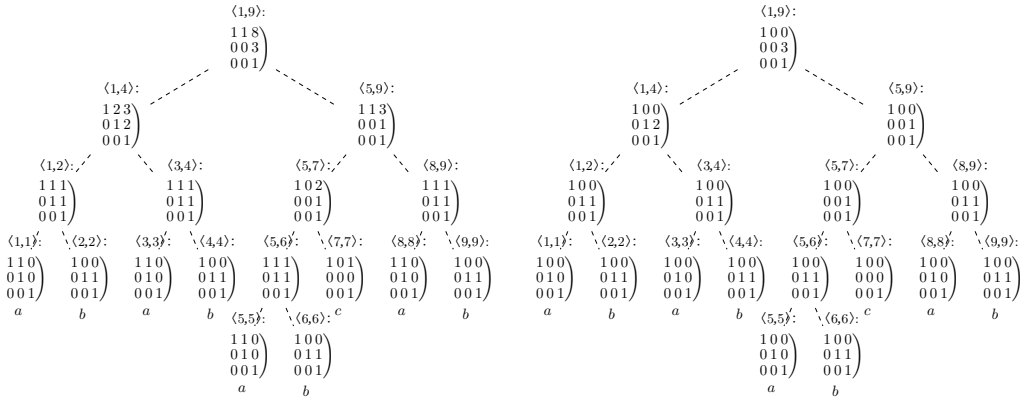


图3 由$\mathcal{A}^{\text{ex}}$构建的树 $\mathbb{T}_0 = \mathbb{T}_{\{x_1,x_2\}}$（左侧）和 $\mathbb{T}_1 = \mathbb{T}_{\{x_2\}}$。

定理1中，我们不需要完整集合 $\{\mathbb{T}_Y\}^{Y\subseteq X}$（尽管证明第（2）项时确实需要它）。相反，我们只使用序列 $k+1$ 树 $\mathbb{T}_0 := \mathbb{T}_{\{x_1,\dots,x_k\}}$ $\mathbb{T}_1 := \mathbb{T}_{\{x_2,\dots,x_k\}}$ $\cdots$，$\mathbb{T}_{k-1} := \mathbb{T}_{\{x_k\}}$ $\mathbb{T}_k := \mathbb{T}_{\emptyset}$。至此，预处理阶段的描述完毕。

存储矩阵的逻辑 我们可以借助一组映射规则来理解数据结构的逻辑：对于一个 $i \in [0,k]$ 和 $1 \le l \le r \le n$，我们在 $[l,r]$ 内定义一组映射规则 $\tau_{>i}^{[l,r]} := \{x_1 \mapsto \emptyset, \dots, x_i \mapsto \emptyset, x_{i+1} \mapsto [l,r], \dots, x_k \mapsto [l,r]\}$。直观来说，这组规则是一个简单的过滤器，它会排除提及任何变量$x \prec x_i$的部分映射。为简化表示，我们不写 $\mathcal{M}\langle l,r\colon \tau_{>i}^{[l,r]}\rangle$，而写作 $\mathcal{M}\langle l,r\colon \tau_{>i}\rangle$。

▶ **引理 4**。 算法*2*结束时，对于每个有效下标 (l,r)，存储在 $\mathbb{T}_i\langle l,r\rangle$ 中的矩阵等于 $M\langle l,r\colon \tau_{>i}\rangle$。

复杂度分析 我们简要指出，每棵树总共有 $2\cdot n-1$ 个节点，每个节点都存有一个矩阵于 $\mathbb{N}^{Q\times Q}$ 中，该矩阵通过一次矩阵乘法计算得到。由于我们总共存储了 $k+1$ 棵树，因此预处理阶段耗时为 $O(|Q|^{\omega}\cdot|X|\cdot|w|)$。

3.4 更新子程序

在继续介绍算法中的访问阶段之前，我们先介绍一个辅助子程序，如算法 3 所述。引理 4 表明，在预处理阶段结束时，$\mathbb{T}_i$ 中的矩阵将统计不含变量 $x_1,\dots,x_i$ 的部分运行。更新子程序的目标是修改 $\mathbb{T}_i$，使这些矩阵统计在位置 s_1 看到 x_1、在位置 s_2 看到 x_2，依此类推的部分运行。我们将其描述为一个过程：修改某个版本的 $\mathbb{T}_i$（可能已经经过更新），使其包含单个映射 $x \mapsto s$。在访问阶段的第 i 次迭代中（即计算出值 s_i 后），我们将对每个满足 $j > i$ 的 $\mathbb{T}_j$ 调用一次该过程。

▶ **引理 5**。 固定 $\mathbb{T}$ 为某个可能已经更新过的 $\mathbb{T}_i$。令 (l,r) 为一个有效下标，并定义 $\tau_{l\,r}^{\text{curr}}$，为 $[l,r]$ 中的函数规则集，使得 $\mathbb{T}\langle l,r\rangle$ 中的矩阵等于 $M\langle l,r\colon \tau_{\text{当前},\,r}\rangle$；并假设，对于某个 $x \in X$，有 $(x \mapsto \emptyset) \in \tau_{l\,r}^{\text{curr}}$。将执行 更新$(\mathbb{T}, x \mapsto s)$ 后得到的树称为 $\mathbb{T}'$。那么：

- 如果 $s \in [l,r]$，$\mathbb{T}'\langle l,r\rangle$ 中的矩阵等于 $M\langle l,r\colon \tau_{l\,r}^{\text{new}},\rangle$，其中 $\tau_{l\,r}^{\text{new}}$ 是通过将 $\tau_{l\,r}^{\text{curr}}$ 中的 $x \mapsto \emptyset$ 替换为 $x \mapsto s$ 得到的集合。如果 $s \notin [l,r]$，$\mathbb{T}'\langle l,r\rangle$ 中的矩阵等于 $M\langle l,r\colon \tau_{l\,r}^{\text{curr}},\rangle$。

算法 3 更新树 $\mathbb{T}$ 以包含映射 $x \mapsto s$。

```
1: 过程 更新(𝕋, x ↦ s, l, r)
2:     如果 s < l 或 r < s 则
3:         令 𝕋_i⟨l, r⟩ :: M
4:         返回 M
5:     否则如果 l = r 则
6:         a ← w[l]
7:         Z ← 在变量 𝕋_i⟨l, r⟩
8:         𝕋_i⟩l, r⟩ :: M ← Δ_a^{Z∪{x}}(𝒜)
9:         在变量 𝕋_i⟨l, r⟩ ← Z ∪ {x}
10:        返回 M
11:    else
12:        m ← mid(l, r)
13:        M_1 ← 更新(𝕋, x ↦ s, l, m)
14:        M_2 ← 更新(𝕋, x ↦ s, m + 1, r)
15:        𝕋_i⟨l, r⟩ :: M ← M_1 · M_2
16:        返回 M
```

3.5 访问阶段

现在，我们以访问阶段模板（第3.1节）为基础，描述算法中实际使用的访问阶段。我们将简要说明该算法，同时为定理1中第（1）项给出正确性证明。

解释。令 μ^* 为 $[\![\mathcal{A}]\!](w)$ 中第 t 个映射，并固定 $s_1 := \mu^*(x_1) \dots, s_k := \mu^*(x_k)$。对于 $i \in [1,k]$ 和 $m \in [1,n]$，定义 $\tau_{x_i \le m} := \{x_1 \mapsto s_1, \dots, x_{i-1} \mapsto s_{i-1}, x_i \Mapsto [1,m]\}$。请注意，在访问阶段的模板中，所需的计数操作可以概括为：对某些 i和 m，计算 $\sharp[\![\mathcal{A}]\!](w)\langle\tau_{x_i \le m}\rangle$。

假设当前处于第 i 次迭代，且已经计算出值 $s_1, \dots, s_{i-1}$，并执行了算法4第9行和第10行的更新。注意，对于任意 m，我们都可以在 $[1,m]$ 中找到一个 τ_L，并在 $[m+1,n]$ 中找到一个 τ_R，使得 $\tau_L \cdot \tau_R = \tau_{x_i \le m}$；于是定义 $M_L^* := \mathcal{M}\langle 1, m\colon \tau_R\rangle$ 和 $M_R^* := \mathcal{M}\langle m+1, r\colon \tau_L\rangle$，并注意到，根据引理3，$\sharp[\![\mathcal{A}]\!](w)\langle\tau_{x_i \le m}\rangle = \sum_{p \in I, q \in F}(M_L^* \cdot M_R^*)$。

让我们更仔细地看看 τ_L 和 τ_R。我们看到 $\tau_L = \{x_j \mapsto s_j\}_{s_j \in [1,m]} \cup \{x_j \Mapsto \emptyset\}_{s_j \in [m+1,n]} \cup \{x_i \Mapsto [1,m]\}$；并且 $\tau_L = \{x_j \mapsto s_j\}_{s_j \in [m+1,n]} \cup \{x_j \Mapsto \emptyset\}_{s_j \in [1,m]} \cup \{x_i \Mapsto \emptyset\}$。根据引理 5，所执行的更新确保我们可以通过组合当前版本 $\mathbb{T}_{i-1}$中的矩阵找到 this M_L，并且可以通过组合当前版本 $\mathbb{T}_i$中的矩阵找到 M_R。更准确地说，对于任意选择的 $1 \le m_1 < \cdots < m'_\chi < n$，都可以通过组合存储在 $\mathbb{T}_{i-1}\mathbf{1}, m_1\ \mathbb{T}_{i-1}m_1+1, m_2 \dots, \mathbb{T}_{i-1}m_{\chi-1}+1, m_\chi\ \mathbb{T}_{i-1}m_\chi+1, m$ 处的矩阵得到 M_L^*；而对于任意选择的$m < m'_1 < \cdots < m'_\chi < n$，都可以通过组合存储在 $\mathbb{T}_i m, m'_1\ \mathbb{T}_i m'_1+1, m'_2 \dots, \mathbb{T}_i m'_{\chi-1}+1, m'_\chi\ \mathbb{T}_i m'_\chi+1, n$ 处的矩阵得到 M_R^*。

最后要说明的是如何高效地找到这些矩阵。[1] 我们观察到，$Algorithm\ 4$定义了两个矩阵 M_L^{out} 和 $M_{\text{输出}R}$，它们用作 $[1,m]$ 中某个前缀的缓存，

[1] 我们注意到，无需进一步操作，以上解释确实给出了一个对数平方访问算法：由于 mid 高度平衡，我们可以用对数数量的由 mid 实现的区间来组合任意范围，并利用这些区间计算 M_L^* 和 M_R^*。

算法4 访问 t中的第 $[\![\mathcal{A}]\!](w)$个元素。

1: **过程** 访问($\mathcal{A}$ $\mathbb{T}_1, \dots, \mathbb{T}_k, t$)
2: **声明** $s_1, \dots, s_k$
3: **对于** $i \in [1,k]$, **执行**
4: $(s_i, M_L^{\mathsf{out}}, M_R^{\mathsf{out}}) \leftarrow$ 二分查找
5: **设** $\mathbb{T}_i\langle s_i, \ s_i\rangle :: M^{\mathsf{in}}$
6: $M \leftarrow M_L^{\text{输出}} \cdot M_{\text{输入}} \cdot M_R^{\text{输出}}$
7: 差异 $\leftarrow \sum_{p\in I, q\in F} M(p, \ q)$
8: $t \leftarrow t -$差值
9: **对** $j \in [i+1, n]$ **执行**
10: 更新($\mathbb{T}_j$, $x_i \mapsto s_i$, $\mathbf{1}$, n)
11: **返回**$\{x_1 \mapsto s_1, \dots, x_k \mapsto s_k\}$

12: **过程**二分查找
13: $M_{\text{输出}_L} \leftarrow I_Q$; $M_R^{\mathsf{out}} \leftarrow I_Q$
14: $(l, r) \leftarrow (\mathbf{1}, n)$
15: **当** $l < r$ **执行**
16: $m \leftarrow \mathsf{mid}(l, r)$
17: **令** $\mathbb{T}_{i-1}\langle l, m\rangle :: M_L^{\mathsf{in}}$
18: **令** $\mathbb{T}_i\langle m+1, r\rangle :: M_R^{\mathsf{in}}$
19: $M \leftarrow M_L^{\mathsf{out}} \cdot M_{\text{属于}} \cdot M_{\text{输入}} \cdot M_{\text{输出}}$
20: **如果** $t \le \sum_{p\in I, q\in F} M(p,q)$ **则**
21: $(l, \ r) \leftarrow (l, \ m)$
22: $M_{\text{输出}_R} \leftarrow M_R^{\mathsf{in}} \cdot M_{\text{输出}R}$
23: **else**
24: $(l, \ r) \leftarrow (m+1, \ r)$
25: $M_L^{\text{输出}} \leftarrow M_L^{\mathsf{out}} \cdot M_L^{\text{输入}}$
26: **返回** $(l, M_{\text{输出}L}, M_R^{\mathsf{out}})$

以及 $[m+1, n]$ 中某个后缀的缓存。借助这些矩阵，在二分搜索的每一步，我们只需合成常数个矩阵（第 19 行），即可得到 M_L^* 和 M_R^*。每轮迭代结束时从 t 中移除的值，也可以通过使用缓存矩阵得到（第 7 行）。

恢复数据结构 若要多次执行访问操作，必须保留原始数据结构。我们注意到，只需存储被修改的索引即可维持这一点；这些索引最多涉及 $(k+1)\cdot c_{\mathsf{mid}} \cdot \log(n)$ 个矩阵，然后再将其恢复。至此，主算法的描述完毕。

复杂度分析 访问阶段的运行时间由更新操作主导，该操作需要 $O(|Q|^\omega \cdot |X|^2 \cdot \log(|w|))$ 时间。

3.6 若访问阶段给定顺序 $\prec$

我们说明如何处理定理1中的情形（2）。主要区别在于，我们保留完整集合 $\{\mathbb{T}_Y\}_{Y\subseteq X}$，而非仅保留树 $\mathbb{T}_0, \dots, \mathbb{T}_k$。随后，在访问阶段，我们识别序列 $y_1 \prec \cdots \prec y_k$ 中的变量，并利用树 $\mathbb{T}_{\{y_1,\dots,y_k\}}$ $\mathbb{T}_{\{y_2,\dots,y_k\}}$ $\cdots$,$\mathbb{T}_{\{y_k\}}$得到映射 μ^*。识别出这些树之后，后续过程与情形（1）完全类似。由此完成定理1的证明。

4 扩展到直线程序

本节将算法扩展为接收由直线程序压缩方案压缩的字符串。结果如下：

▶定理 *6.* 设 *A* 是定义在变量集合 X 上的无歧义*vset*自动机，$\mathfrak{S}$ 是表示字符串 $w \in \Sigma^*$ 的直线程序。*1.* 对于固定的 $\prec$ 在 X 上的顺序，我们可以在经过 $O(|Q|^\omega \cdot |X| \cdot |\mathfrak{S}|)$ 预处理后，以 $O(|Q|^\omega \cdot |X|^2 \cdot \log(|w|))$ 的时间对集合 $[\![A]\!](w)$ 进行直接（排名）访问。

算法 5 ⟦ 上直接访问的预处理阶段$\mathcal{A}$⟧(str($\mathfrak{S}$))。

过程 预处理($\mathcal{A}$ $\mathfrak{S}$)
  声明 $\mathbb{D}_1, \ldots, \mathbb{D}_k$
  对于 $i \in [0,k]$，**执行**
    构建($\mathbb{D}_i, S_0$)

过程 构建($\mathbb{D}_i, A$)
  如果 $R(A) = a$ **则**
    $\mathbb{D}_i\langle A\rangle :: M \leftarrow \Delta^{\{x_{i+1}, \ldots, x_k\}a}$
    返回 M
  else
    $BC \leftarrow R(A)$
    $M_1 \leftarrow$ 构建($\mathbb{T}_i, B$)
    $M_2 \leftarrow$ 构建($\mathbb{T}_i, C$)
    $\mathbb{D}_i\langle A\rangle :: M \leftarrow M_1 \cdot M_2$
    返回 M

2. 如果 $\prec$ 在访问阶段作为输入给出，则该问题需要 $O(|Q|^\omega \cdot 2^{|X|} \cdot |\mathfrak{S}|)$预处理时间。

本节余下部分将用于证明该定理。因此，我们固定$\mathcal{A}$ 和 $\mathfrak{S}$，其中 str($\mathfrak{S}$) $= w$ 和 $|w| = n$。预先说明一下，该算法与字符串情形中的算法基本相同。

我们首先强调直线程序的一项关键平衡性结果。借助该结果，在本节余下部分中，我们假设输入直线程序 $\mathfrak{S}$ 的深度为 $O(\log n)$。

▶**定理7** ([16])。 给定一个表示长度为 n 的字符串的直线程序 $\mathfrak{S}$，可以在线性时间内构造一个等价的直线程序，其大小为 $O(|\mathfrak{S}|)$，深度为 $O(\log n)$。

我们指出，这一结果并不意味着所得的直线程序是强平衡的。函数 mid 强平衡这一要求对于第3节中的结果并非必要；它只会在下一节中变得重要，因为届时我们将处理数据中的编辑。

接下来，我们注意到，可以通过在直线程序的有向无环图结构上采用线性时间的动态规划方法，预先计算 |str(A)| 在 $A \in N$ 中的值。这一思路在直线程序文献中十分常见 [30, 29, 24]。

4.1 预处理

预处理阶段如算法 5 所示。我们从树 $\mathbb{T}_Y$ 转而使用图结构 $\mathbb{D}_Y$，其结构由直线程序 $\mathfrak{S}$ 本身决定。索引不再由区间给出，而是由 $\mathfrak{S}$ 中的非终结符给出。每个索引处存储的信息仍然是 $\mathbb{N}^{Q\times Q}$ 中的一个矩阵和 X 中的一组变量；但索引本身则从区间变为 N 中的非终结符。预处理的思路基本不变，其工作方式可以直接借鉴字符串情形：每个 $\mathbb{D}_Y$ 都可以“展开”为一棵树 $\mathbb{T}_Y$，因此我们从 $\mathbb{D}_Y$ 得到的值满足正确性所需的条件。

4.2 访问阶段

访问阶段如算法6所述。同样，基本思路没有变化；很容易看出，第一次迭代会为 s_1 检索到正确的值，其论证与字符串的情况类似。

然而，更新操作的工作方式有所不同：矩阵不再与某个范围相关联，因此，在叶节点处修改矩阵并向上传播该值，可能会不加区分地影响其他索引。对于我们的更新操作，我们借鉴了现有文献中关于 SLP 更新的思路，修改数据结构$\mathbb{D}_i$、.....、 $\mathbb{D}_k$ [30, 21]。

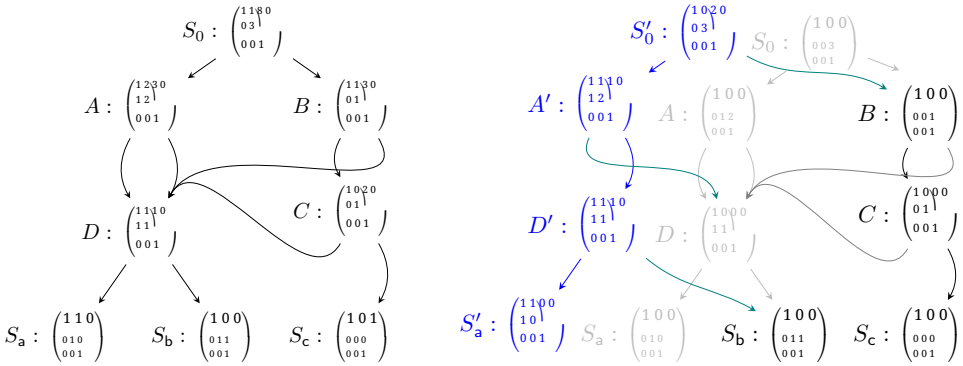


图 4 （左） $\mathbb{D}_0$ 中的矩阵；（右）灰色和黑色表示 $\mathbb{D}_1$ 中的矩阵，蓝色表示执行 Update($\mathbb{D}_1$, $x \mapsto 3$) 后的新矩阵。

算法6 访问 t在 $[\![\mathcal{A}]\!](\mathrm{str}(\mathfrak{S}))$中的第 t个元素。

过程 访问($\mathcal{A}$ $\mathbb{D}_1, \ldots, \mathbb{D}_k, t$)
  $s_1, \ldots, s_k$
  对于 $i \in [1,k]$ **执行**
    $(s_i, a, M_{\text{输出}L}, M_R^{\text{out}}) \leftarrow$二分查找
    令 $\mathbb{D}_i\langle A\rangle :: M^{\text{in}}$
    $M \leftarrow M_L^{\text{out}} \cdot M_{\text{输入}} \cdot M_{\text{输出}}$
    差异 $\leftarrow \sum_{p\in I,\ q\in F} M(p,\ q)$
    $t \leftarrow t -$差异
    对于 $j \in [i,n]$，**执行**
      更新($\mathbb{D}_j$, $x_i \mapsto s_i$, S_0)
  返回 $\{x_1 \mapsto s_1, \ldots, x_k \mapsto s_k\}$

过程二分查找
  $M_{\text{输出}L} \leftarrow I_Q$; $M_R^{\text{out}} \leftarrow I_Q$
  $(l,\ r) \leftarrow (1,\ n); A \leftarrow S_0$
  当 $l < r$ **时**
    $BC \leftarrow R(A)$
    $m \leftarrow l + |B| - 1$
    令 $\mathbb{D}_{i-1}\langle B\rangle :: M_L^{\text{in}}$
    设 $\mathbb{D}_i\langle C\rangle :: M_R^{\text{in}} \cdot M_{\text{入}} \cdot M_{\text{输出}}$
    $M \leftarrow M_{\text{出}}$
    若 $t \le \sum_{p\in I,\ q\in F} M_L^R(p,\ q)$ **则**
      $(l,\ r) \leftarrow (l,\ m)$
      $M_{\text{输出}}^{\text{输出}} \leftarrow M_R^{\text{in}} \cdot M_R^{\text{输出}}$
    else
      $(l, r) \leftarrow (m+1, r)$
      $M_{\text{输出}}^{\text{输出}} \leftarrow M_L^{\text{out}} \cdot M_L^{\text{输入}}$
  返回 $(l, R(A)\ M_{\text{输出}L}, M_R^{\text{out}})$

我们将简要说明这一思路，并给出一个示例。对于 Update($\mathbb{D}$, $x \mapsto s$)，我们找出一条由非终结符 S_0、A_1、……、A_m 组成的路径，该路径通向 w中的第 s个元素。随后，我们复制这些非终结符，并将它们连接到有向无环图的其余部分，使得访问第 s个元素的路径正是这串新生成的非终结符。特别地，S_0 会被其副本替换。然后，我们像在字符串情形中那样，对 $\mathbb{D}\langle A_m\rangle$ 执行更新，并向上传播信息。最终得到的 $\mathbb{D}$ 包含了计算值 s_2、……、s_k所需的正确信息。示例见图 4。

最后，我们指出，定理 6 中第 (2) 项的思路与字符串的情形完全类似。也就是说，在访问阶段给定顺序 $\prec$ 时，我们可以改用数据结构 $\{\mathbb{D}_Y\}^{Y\subseteq X}$。

复杂度分析 对于预处理阶段，可以看出算法 5 的运行时间为$O(|Q|^\omega \cdot |X|^2 \cdot |\mathfrak{S}|)$。对于访问阶段，我们注意到，二分搜索所需的步数等于 $\mathfrak{S}$的深度；由于平衡性有保证，该深度为 $O(\log n)$，因此访问阶段的运行时间为 $O(|Q|^\omega \cdot |X|^2 \cdot \log(|w|))$。至此，定理 6 的证明完成。

5 更新处理

本节将说明， [6] 中提出的复杂编辑操作的结果依然成立，同时保留对数时间访问时间。我们不会采用 [6], 提出的框架，而是将其作为 [30], 的**复杂字符串编辑**[2] 框架的一种改编来介绍，因为我们几乎无需改动就能借用直线程序编辑操作。遗憾的是，我们缺少 [6]中使用的拆分与剪切操作；我们认为这些操作同样可以计算，并计划将其纳入本文的完整版。

我们将采用与 [24], 类似的方式：作者在其中介绍了编辑框架，随后简要说明了 [30] 中的机制如何可直接应用于他们的数据结构。

需要说明的是，本节中的结果仅适用于强平衡直线程序。一般而言，若要在不损失强平衡保证的情况下平衡一个直线程序，其规模可能会增加一个 $\log n$ 因子 [15]。不过，我们也指出，这些结果适用于无需任何额外假设的字符串：我们可以使用第3节中定义的函数，将一个词表示为强平衡直线程序。

复杂字符串编辑在 Σ 上的字符串数据库是由有限多个 Σ 上的字符串组成的集合，其中每个字符串都有一个名称。形式化地说，字符串数据库是一个函数 $D\colon\{d_1,\ldots,d_m\}\to\Sigma^*$，其中$\{d_1,\ldots,d_m\}$ 是一个有限的字符串名称集合，而 $D(d_i)$ 是分配给名称 d_i 的字符串。我们还称 $\{d_1,\ldots,d_m\}$ 为字符串数据库 D 的模式。我们将 D 的大小定义为 $|D|=\sum_{i=1}^{m}|D(d_i)|$。

给定字符串数据库 D，我们可以通过一系列文本编辑操作创建新字符串。我们从 [30, 24], 借用了基于字符串名称 $\{d_1,\ldots,d_m\}$ 的**复杂字符串编辑表达式**（简称 CSE 表达式）这一概念，其由如下语法定义

$$\psi := d_\ell, \ell\in[1,m] \mid \text{拼接}(\psi,\psi) \mid \text{提取}(\psi,l,r) \mid \text{删除}(\psi,l,r) \mid \text{插入操作}(\psi,\psi,k) \mid \text{复制操作}(\psi,l,r,k)$$

其中值 l,r,k 为有效。 ψ 中字符串名称与操作的数目记作$|\psi|$。CSE 表达式 ψ 的语义由函数 $[\![\psi]\!]$ 给出，该函数将基于 $\{d_1,\ldots,d_m\}$ 的字符串数据库映射为 Σ^* 中的字符串，并按如下方式递归运作。

$$\begin{aligned}
[\![d_\ell]\!](D) &= D(d_\ell)\\
[\![\mathsf{concat}(\psi_1,\psi_2)]\!](D) &= [\![\psi_1]\!](D)\cdot[\![\psi_2]\!](D)\\
[\![\mathsf{extract}(\psi,l,r)]\!](D) &= [\![\psi]\!](D)[l,r]\\
[\![\mathsf{delete}(\psi,l,r)]\!](D) &= [\![\psi]\!](D)[1,l-1]\cdot[\![\psi]\!](D)[r+1\ldots]\\
[\![\mathsf{insertop}(\psi_1,\psi_2,k)]\!](D) &= [\![\psi_1]\!](D)[1,k-1]\cdot[\![\psi_2]\!](D)\cdot[\![\psi_1]\!](D)[k\ldots]\\
[\![\mathsf{copyop}(\psi,l,r,k)]\!](D) &= [\![\psi]\!](D)[1,k-1]\cdot[\![\psi]\!](D)[l,r]\cdot[\![\psi]\!](D)[k\ldots]
\end{aligned}$$

其中 ψ、 ψ_1和 ψ_2 是 CSE 表达式， l、 r、 k 是各自字符串中的有效索引； $u[k\ldots]$ 表示从位置 k 开始的 u 的后缀。如果 l、 r 或 k 无效，则 $[\![\psi]\!](D)$ 未定义——在算法中，我们假设它们始终有效，因为验证这一点需要在 $|D|$ 和 $|\psi|$ 中花费线性时间。我们将字符串数据库 D 上由 CSE 表达式 ψ 诱导的最大中间字符串大小 $|\max_\psi(D)|$ 定义为：对 ψ 的任意子表达式 ψ， $[\![\psi]\!](D)$ 的最大大小。

[2] 在 [30] 中，用于字符串 的术语是文档。除本脚注外，我们不再进一步关注这一变化。

字符串数据库与直线程序之间的联系由一个定根函数给出，该函数将模式映射到直线程序中的非终结符。形式上，从$\{d_1、\ldots\ldots、d_m\}$到$\mathfrak{S}$的定根函数是一个函数$\mathfrak{r}:\{d_1、\ldots\ldots、d_m\}\to N$。它们定义了字符串数据库$D_{\mathfrak{S},\mathfrak{r}}$，其定义为$D_{\mathfrak{S},\mathfrak{r}}(d)=\mathrm{str}(\mathfrak{r}(d))$。

为了陈述本节的主要结果，我们引入字符串数据库D关于$\{d_1,\ldots,d_m\}$和变量集自动机 A 的可编辑直接访问结构这一概念。若结构$\mathbb{D}$能对任意关于$\{d_1,\ldots,d_m\}$的 CSE 表达式ψ，实现对集合$[\![\mathrm{A}]\!]([\![\psi]\!](D))$的直接（排名）访问，则称$\mathbb{D}$是$D$和 A 的可编辑直接访问结构。此外，若存在一个耗时$f(\psi,\cdot)$的过程，接收ψ并将其修改为可编辑直接访问结构$\mathbb{D}_\psi$，使其能够在$g(\psi,\cdot)$时间内直接访问$[\![\mathrm{A}]\!]([\![\psi]\!](D))$，则称$\mathbb{D}$支持$f(\psi,\cdot)$时间的编辑和$g(\psi,\cdot)$时间的直接访问。

▶定理*8*。设A是一个状态集为Q、变量集为X的无歧义变量集自动机，设$\mathfrak{S}$是一个强平衡直线程序，且$\mathfrak{r}$是到$\mathfrak{S}$的定根函数。我们可以在$O(|Q|^\omega\cdot|X|\cdot|\mathfrak{S}|)$时间内构造一个适用于$D_{\mathfrak{S},\mathfrak{r}}$和$A$的可编辑直接访问结构$\mathbb{D}$，该结构支持耗时$O(|\psi|\cdot|Q|^\omega\cdot|X|\cdot\log|max_\psi(D)|)$的编辑，以及耗时$O(|Q|^\omega\cdot|X|^2\cdot|max_\psi(D)|)$的直接访问。

5.1 字符串上的编辑

鉴于我们的结果可以轻松适用于单词，因此我们只针对直线程序讨论编辑。不过，我们认为，字符串情形下的编辑过程可以更简单；尤其是，可以采用一种不改变数据结构中节点数量的方式来完成。其思路是采用编辑 AVL 树的经典结果，正如 [6] 中所做的那样。更多细节留待本文未来的完整版本中讨论。

6 结论与未来工作

我们提出了一种针对字符串和 SLP 压缩字符串上的单子二阶逻辑查询的动态直接（排名）访问算法。我们通过在访问过程中减少一个对数因子改进了 [6] 的结果，并展示了该方法向直线程序的扩展。值得注意的是，Bagan [4] 和 Kazana [17] 的博士论文中都已包含针对单子二阶逻辑查询（甚至是树）的直接访问方法版本，因此我们工作的主要贡献在于加入动态特性，并将其扩展到直线程序。

相较于 Bourhis 等人的 [6]，我们的工作有一个盲点：我们没有讨论半群上的查询求值——实际上，我们并未看出我们的结果能在这方面带来什么新见解。若能刻画哪些半群支持对数时间直接访问、哪些半群无法支持（并给出某种下界），将会很有意义。

对于未来的工作，我们认为主要问题是，本文提出的技术能否扩展到树上 MSO 查询的动态直接访问。我们相信这是可行的，例如可以通过 [19] 实现。不过，这在技术上也颇具挑战，因为这一框架无法保持连续性，而连续性对我们的解决方案至关重要。

此外，直线程序必须是强平衡的才能支持对数时间更新，这显然是一项缺陷。具体而言，这意味着该方法不适用于任意直线程序。我们将以下问题留待解决：是否存在某种较弱的直线程序平衡性判据，也能实现这一点。

参考文献

1 Antoine Amarilli、Pierre Bourhis、Louis Jachiet 和 Stefan Mengel。一种基于电路的高效枚举方法。载于*ICALP*，2017。

16 SLP 压缩字符串上 MSO 查询求值的动态直接（排序）访问

2 Antoine Amarilli、Pierre Bourhis 和 Stefan Mengel。重新标记下的树上枚举。载于 ICDT，2018。3 Guillaume Bagan。树可分解结构上的 MSO 查询可在线性延迟内计算。载于 CSL，2006。4 Guillaume Bagan。逻辑查询求值中枚举问题的算法与复杂性。（逻辑查询求值中枚举问题的算法与复杂性），2009。5 Philip Bille、Gad M. Landau、Rajeev Raman、Kunihiko Sadakane、Srinivasa Rao Satti 和 Oren Weimann。文法压缩字符串和树的随机访问。SIAM J. Comput.，44(3):513–539，2015。6 Pierre Bourhis、Florent Capelli、Stefan Mengel 和 Cristian Riveros。字符串上 MSO 查询求值的动态直接访问。载于 ICDT，2025。7 Pierre Bourhis、Alejandro Grez、Louis Jachiet 和 Cristian Riveros。单词上 MSO 逻辑的排名枚举。载于 ICDT，2021。8 Karl Bringmann、Nofar Carmeli 和 Stefan Mengel。连接查询上直接访问的紧细粒度界。载于 PODS，2022。9 J Richard Büchi。弱二阶算术与有限自动机。数学逻辑季刊，6:1–6，1960。10 Florent Capelli 和 Oliver Irwin。带否定的合取查询的直接访问。载于 ICDT，2024。11 Nofar Carmeli。以理想时间保证回答合取查询的并（特邀报告）。载于 ICDT，2022。12 Bruno Courcelle。线性延迟枚举与单子二阶逻辑。离散应用数学，157(12):2675–2700，2009。13 Ronald Fagin、Benny Kimelfeld、Frederick Reiss 和 Stijn Vansummeren。文档跨距器：信息抽取的一种形式化方法。ACM会刊（JACM），62(2):1–51，2015。14 Fernando Florenzano、Cristian Riveros、Martín Ugarte、Stijn Vansummeren 和 Domagoj Vrgoc。正则文档跨距器的高效枚举算法。ACM Trans. Database Syst，45(1):3:1–3:42，2020。15 Moses Ganardi。通过收缩直线程序进行压缩。载于第29届欧洲算法研讨会年会，ESA 2021，2021。16 Moses Ganardi、Artur Jez 和 Markus Lohrey。直线程序平衡。ACM会刊，68(4):27:1–27:40，2021。17 Wojciech Kazana。常数延迟的查询求值。（常数延迟的查询求值）。博士论文，卡尚高等师范学校，巴黎，法国，2013。18 John C. Kieffer 和 En-Hui Yang。基于文法的编码：一类新的通用无损源编码。IEEE Trans. Inf. Theory，46(3):737–754，2000。19 Sarah Kleest-Meißner、Jonas Marasus 和 Matthias Niewerth。树上的 MSO 查询：使用森林代数在更新下枚举答案。Log. Methods Comput. Sci.，21(4)，2025。20 Markus Lohrey。SLP 压缩字符串上的算法学：综述。Groups Complex. Cryptol.，4(2):241–299，2012。21 Markus Lohrey 和 Markus L. Schmid。压缩树上 MSO 查询的枚举。Proc. ACM Manag. Data，2(2):78，2024。22 Francisco Maturana、Cristian Riveros 和 Domagoj Vrgoc。用于抽取不完整信息的文档跨距器：表达能力与复杂性。载于 PODS，2018。23 Martin Muñoz 和 Cristian Riveros。嵌套文档上的流式枚举。ACM Trans. Database Syst.，49(4):15:1–15:39，2024。24 Martin Muñoz 和 Cristian Riveros。SLP 压缩文档的常数延迟枚举。Log. Methods Comput. Sci.，21(1)，2025。25 Gonzalo Navarro。高度重复字符串集合索引，第一部分：重复度度量。ACM Comput. Surv.，54(2):29:1–29:31，2022。doi:10.1145/3434399。

26 Gonzalo Navarro. 高度重复字符串集合索引，第二部分：压缩索引。*ACM Comput. Surv.*, 54(2):26:1–26:32，2022年。 doi:10.1145/3432999。27 Gonzalo Navarro、Francisco Olivares 和 Cristian Urbina. 广义直线程序。*Acta Informatica*，62(1):14，2025年。网址：https://doi.org/10.1007/s00236-025-00481-3，doi:10.1007/S00236-025-00481-3。28 Frank Rubin. 文本文件压缩实验。*Commun. ACM*，19(11):617–623，1976年。29 Markus L. Schmid 和 Nicole Schweikardt. 基于直线程序压缩文档的 Spanner 求值。载于 *PODS*，2021年。30 Markus L. Schmid 和 Nicole Schweikardt. 基于直线程序表示的文档数据库查询求值及复杂文档编辑。载于 *PODS*，2022年。31 Volker Strassen. 高斯消元并非最优。*Numerische Mathematik*, 13(4):354–356，1969年8月。

A 第3节的证明

A.1 引理3的证明

证明。设 ρ_L 是从 p 到某个 q'、关于 w 的 (l, m)-部分运行，且 ρ_R 是从 q' 到 q、关于 w 的 $(m+1, r)$-部分运行；此外，设 ρ_L 遵循 τ_L，且 ρ_R 遵循 τ_R。我们将这些运行连接起来，得到 $\rho_L{\cdot}\rho_R$，它是关于 w、遵循 $\tau_L{\cdot}\tau_R$ 的 (l, r)-部分运行。同样，设 ρ 是关于 w、遵循 $\tau_L{\cdot}\tau_R$ 的 (l, r)-部分运行。我们可以在 m 处分开它，得到两个部分运行 τ_L、ρ_R：它们都遵循 $\tau_L{\cdot}\tau_R$，但 τ_L 不涉及 $[m+1, r]$ 中的任何值，且 τ_R 不涉及 $[l, m]$ 中的任何值，因此特别地，ρ_L 遵循 τ_L，而 ρ_R 遵循 τ_R。我们得到 $\mathrm{M}_{l,r}[\tau_L{\cdot}\tau_R](p, q) = \sum_{q' \in Q} \mathrm{M}_{l,m}[\tau_L](p, q') \cdot \mathrm{M}_{m+1,r}[\tau_R](q', q)$。◀