arXiv:2301.12851v1 [cs.FL] 30 Jan 2023

带同步计数的正则模式快速匹配（技术报告）*

Luk´aˇs Holík、Juraj Síˇc、Lenka Turoˇnov´a 和 Tom´aˇs Vojnar

布尔诺理工大学，捷克{
holik,sicjuraj,ituronova,vojnar}@fit.vut.cz

摘要。带有有界重复（即计数）的正则表达式的快速匹配（例如 (ab){50,100}），也就是匹配时间与文本长度呈线性关系且独立于重复上限，至少二十年来一直是一个悬而未决的问题。我们证明，对于一大类带计数的正则表达式（我们称之为同步正则表达式），可以实现快速匹配。我们通过实验证明，该类涵盖了正则表达式匹配的常见应用中所使用的几乎所有计数。这一复杂度结果基于对一种近期匹配算法的改进与分析；该算法将正则表达式编译为确定性计数集自动机（具有保存数字集合的寄存器的自动机）。

1 引言

带有界重复（即计数）的正则表达式的快速匹配，至少二十年来一直是一个未解决的问题（参见例如 [1]）。对于 .*a.{100} 这样的正则表达式，标准匹配算法的时间复杂度至多受文本长度乘以重复界限所支配。这使得匹配容易出现不可接受的速度下降，因为文本长度和重复界限通常都很大。在本文中，我们为有界重复的匹配提供了一个具有更加可靠性能的理论基础。我们证明，一个规模庞大且具有实践意义的带计数正则表达式类从理论上支持快速匹配，其运行时间与计数器界限无关，并且关于文本长度呈线性增长。

这个问题还有很强的现实动因。正则表达式匹配用于搜索、数据验证、检测信息泄露、解析、替换、数据抓取、语法高亮等。大多数编程语言都原生支持正则表达式 [2], ，而且正则表达式无处不在（Java、JavaScript 和 Python 软件中有 30–40% 使用正则表达式[3,4,5,6]）。正则表达式匹配的效率和可预测性十分重要。匹配运行时间过长可能造成严重后果，例如输入验证无法抵御注入攻击 [7] ，以及 Cloudflare 服务中断等事件 [8]。正则表达式漏洞也为 ReDoS（正则表达式拒绝服务攻击）打开了方便之门：攻击者精心构造文本，使匹配器不堪重负（例如 StackOverflow 服务中断事件 [9] ，或因使用流行的 Express.js 框架而暴露风险的网站 [10]）。ReDoS 已被广泛视为一种常见且严重的威胁 [3,11,12], ，而在正则表达式中加入计数尤其危险 [13]。

* 将在 FoSSaCS'23 上发表。

匹配算法与复杂度。模式匹配器潜在的不稳定性，与匹配算法的最坏情况复杂度相一致。最广泛使用的匹配方法是回溯（例如，.NET、Python、Perl、PHP、Java、JavaScript 和 Ruby 的标准匹配器均采用这种方法），因为它简单易用，也便于实现反向引用或环视等高级功能。然而，其最坏情况复杂度相对于匹配文本长度呈指数级增长，因此容易遭受 ReDoS。尽管可以通过记忆化等方法改进这一点 [12], ，性能关键型应用中最快的匹配器仍都采用基于自动机的算法，而非回溯。这些方法以 Thompson 算法 [14] （亦称在线 NFA 模拟）为基础。该算法经过多种优化后，已应用于英特尔的 Hyperscan [15]。结合缓存后，它就成为 DFA 的即时子集构造，也称在线 DFA 模拟（应用于谷歌的 RE2、GNU grep、SRM，或 Rust 的标准匹配器 [16,17,18,19]）。不考虑计数时，最坏情况复杂度的主要影响因素为 $O(nm^2)$，其中 n 是文本长度，m 是正则表达式中字符出现次数的数量（m 小于正则表达式本身的大小，即定义该表达式的字符串长度）。我们将字符代价，即向文本追加一个字符所需的代价，记为 m^2。这是遍历某种经典构造 [20,21,22]根据正则表达式编译得到的 NFA 转移所需的代价；该 NFA 有 O(m) 个状态和 $O(m^2)$ 个转移。

扩展正则表达式的语法以支持有界量词（或计数器），例如 (ab){50,100}，会显著增加字符复杂度。给定 k 个最大上界为 ℓ 的计数器，NFA 状态数会增至 $O(m\ell^k)$，转移数和字符代价也会增至 $O((m\ell^k)^2)$。例如，.*a.{k} （即 a 出现在距末尾 k 个字符的位置）的最小 DFA 包含超过 2^k 个状态。此外请注意，由于 k 以十进制数字表示，其取值相对于正则表达式的大小呈指数级增长。这使得即使 k 只是中等偏高，匹配也容易显著变慢，并且几乎所有主流匹配器都存在 ReDoS 漏洞（见 [23,13] ）。与此同时，重复次数上限很容易达到数千，极端情况下甚至达到数千万（在实际 XML [24] 中）。编写一个危险的计数表达式很容易，也很难识别。即使额外投入精力设计和测试正则表达式，安全关键型解决方案仍可能受到与计数相关的 ReDoS [13] 影响，因此开发者有时会避免使用计数、采用变通方案并限制功能。

有界重复匹配问题已从理论和实践角度得到多位作者的研究 [25,24,26,27,28,29,30,23]。其中，近期工作 [23] 是唯一一项为具有实际意义的一类正则表达式提供快速匹配方案的研究。 [23] 的算法将带计数的正则表达式编译为非确定性计数自动机（CA）。这种自动机带有计数器，计数器可以递增、重置，并与常量进行比较。问题的关键在于将 CA 转换为一种精简的确定性机器，使其能够在匹配过程中快速模拟。 [23] 的工作通过将 CA 确定化为计数集自动机（CSA）实现了这一目标。CSA 是一种带有寄存器、用于存储数字集合的自动机。其大小不受计数器界限影响，并且只需通过少量操作更新这些集合；无论集合大小如何，所有操作都能在常数时间内完成。然而，支持类别之外的正则表达式确实存在；该类别既没有语法上的定义，也难以识别（ [23] 中提出的一个错误语法类别方案也证明了这一点

其本身，见附录 E）。例如，.*a{5} 或 (ab){5} 可以处理，但 .*(aa){5} 或 .*(ab){5} 则不行（这一要求属于技术细节，见第 4 节）。

我们的贡献。在本文中，我们

1. 将 **[23]** 的算法推广，以扩展所处理的正则表达式类别；2. 推导出扩展类别的一个有用的句法刻画。

所推导出的类别由平坦计数刻画（计数运算符不嵌套），其中每个被计数表达式 R 的重复都是同步的（来自 R^n 的单词不能具有来自 R^{n+1}的前缀）。这是第一个界限清晰且实用的、允许快速匹配的带计数的正则表达式类别。它包括一种易于识别且常见的情形：R 中的每个单词都恰好包含一个标记符、一个字母或来自有限标记符集合的一个单词，这些标记符能够明确识别 R 的每次出现（注意，即使这一简单类别，此前的任何快速算法也无法处理，包括[23]）。在我们对来自不同来源的大量正则表达式进行的实验中，99.6% 的非平坦计数是同步的，99.2% 是字母标记的。

为了获得上述结果 (1) 和 (2)，我们首先修改 **[23]** 的确定化，使其涵盖整个具有平坦计数的正则表达式类。简而言之，这通过两项改动实现：(i) 允许复制和求并集寄存器中存储的集合；(ii) 在确定化过程中，按计数自动机的状态为其计数器建立索引，以处理这样的计数自动机：到达不同状态的非确定性运行会得到不同的计数器值。

这些修改带来了本文要解决的主要技术挑战：复制和求并集操作并非常数时间，而是与集合的大小呈线性关系。这会使字符代价再次与计数器界限呈线性关系 ℓ。为了消除对计数器界限的依赖，我们通过能够避免复制和求并集的优化来增强确定化过程。首先，为了降低求并集的代价，我们将寄存器中存储的集合的交集存储在新的共享寄存器中，从而使交集不会增加寄存器求并集的代价。然后，为了进一步发挥交集共享的作用，我们同步寄存器更新，以使它们的交集更大。接着我们证明，如果 CSA 不复制寄存器，即每个寄存器在一次转移中最多只能出现在一个寄存器赋值的右侧，那么它就不会复制寄存器，并且并集的代价可以摊销。最后，我们定义具有同步计数的正则表达式类；对于该类，经过优化的 CSA 不会复制计数器，因此它们在匹配过程中的模拟速度很快。

相关工作。在正则表达式匹配的背景下，计数自动机以多种形式、冠以不同名称得到应用（例如 [29,23,24,25,28,31,1,32,33]）。除上文讨论的 [23] 外，其他用于匹配计数正则表达式的方案 [25,24,26,27,28,29,30] 要么只处理较小的正则表达式类别，要么无法实现匹配时间与文本大小线性相关且独立于计数器界限。文献 [29] 提出了计数自动机到计数自动机的确定化方法，对于由字母标记计数所涵盖的单子正则表达式这一有限类别，该方法生成的自动机比显式计数自动机确定化更小，但其确定有限自动机的大小仍取决于计数器界限。文献 [24] 使用了 [25] 提出的带计数器自动机概念。该文主要关注确定性正则表达式，这一类别远小于具有同步计数的正则表达式，并提出了一种匹配算法，但该算法仍取决于计数器界限。论文 [30] 提出了一种算法，其运行时间最坏情况下为文本长度的

平方级。 [28,31] 提出的扩展有限自动机（XFA）通过加入可表示计数器的比特临时存储器来扩展非确定有限自动机；即使对于 .*a.{k} 这样的正则表达式，其确定化的复杂度也已呈计数器界限的指数级增长。 [26,33] 提出的计数器 1-无歧义正则表达式可直接编译为称为 FAC 的确定有限自动机，与我们的计数自动机类似，且不受计数器界限影响；但其适用类别有限，例如不包括 .*a.{k}。

2 预备知识

我们用 $\mathbb{N}$ 表示包含 0 在内的自然数。对于集合 S，$\mathcal{P}(S)$ 表示其幂集，而 $\mathcal{P}_{\text{fin}}(S)$ 表示 S 的所有有限子集构成的集合。

一阶语言（f.o.l.） $\Gamma = (F, P)$ 由函数符号集合 F 和谓词符号集合 P 构成。一个论域为 D 的 $\mathbb{I}$ Γ 的解释 $_{\mathbb{I}}$ 为每个 n 元函数符号 $f \in F$ 赋予一个函数 $f^{\mathbb{I}}: D_{\mathbb{I}}^{n} \to D_{\mathbb{I}}$，并为每个 n 元谓词符号 $p \in P$ 赋予一个函数 $p^{\mathbb{I}}: D_{\mathbb{I}}^{n} \to \{0,1\}$。将变量集合 X 赋值到 $\mathbb{I}$ 的一个赋值是一个全函数 $\nu : X \to D_{\mathbb{I}}$。项的集合 $\text{Terms}_{\Gamma,X}$ 和量词自由公式（原子公式的布尔组合）的集合 $\text{QFF}_{\Gamma,X}$ （它们定义在 Γ 和 X 上），以及项 $t^{\mathbb{I}}(\nu)$ 和公式 $\varphi^{\mathbb{I}}(\nu)$ 的解释，均按通常方式定义。我们用 $\nu \models_{\mathbb{I}} \varphi$ 表示公式φ 被赋值 ν 满足（解释为真）。此时它是可满足的。当上下文清楚时，我们省略下标或上标 $\mathbb{I}$。我们写作 $\varphi[x]$ 和 $t[x]$，分别表示自由变量为 x 的一元公式 φ 或项 t；我们也可以滥用此记号，表示将其唯一的自由变量替换为 x 后得到的项或公式。我们写作 $t^{\mathbb{I}}(k)$ 和 $\varphi^{\mathbb{I}}(k)$，表示 $t^{\mathbb{I}}(\{x \mapsto k\})$ 和 $\varphi^{\mathbb{I}}(\{x \mapsto k\})$ 的值。对于公式集合 $\Psi = \{\psi_1, \ldots, \psi_n\}$，集合 $\text{Minterms}(\Psi)$ 由 Ψ 的所有最小项构成，即满足以下条件的可满足合取式：对于每个 i： $1 \le i \le n$， φ_i 为 ψ_i 或 $\neg\psi_i$。

在本文的其余部分中，我们固定一个由 Σ 符号/字母组成的有限字母表。单词是字母的序列，空字记为 ε。单词 u 和 v 的连接记为 $u \cdot v$，简写为 uv。字母表 Σ 上的单词集合称为语言，语言的连接为 $L \cdot L' = \{u \cdot v \mid u \in L \wedge v \in L'\}$，简写为 LL'。单词或语言 x 的有界迭代 x^i、$i \in \mathbb{N}$定义为：对于单词，$x^0 = \varepsilon$；对于语言，$x^0 = \{\varepsilon\}$；以及 $x^{i+1} = x^i \cdot x$。于是 $x^* = \bigcup_{i \in \mathbb{N}} x^i$。我们采用正则表达式（正则表达式）的通常基本语法，该语法由以下文法生成：$R ::= \varepsilon \mid a \mid (R) \mid R\,R \mid R|R \mid R^* \mid R\{m, n\}$，其中 $m \in \mathbb{N}$、$n \in \mathbb{N} \cup \infty$、$0 \le m$、$0 < n$、$m \le n$，并且 $a \in \Sigma$。对于 $R\{m\}$，我们使用 $R\{m、m\}$。包含形如 $R\{m, n\}$ 或 $R\{m\}$ 的子表达式的正则表达式称为计数正则表达式，其中 m、n 是计数器界限。我们用 $\max_R$ 表示正则表达式 R 的计数器界限中出现的最大整数，并用 cnt_R表示计数器的数量。具有平坦计数的正则表达式不包含嵌套计数，即在子正则表达式 $S\{m, n\}$ 中，S 不能包含计数。正则表达式 R 的语言按其结构归纳地构造：$L(\varepsilon) = \{\varepsilon\}$，$L(a) = \{a\}$ （对于 $a \in \Sigma$），$L(R\,R') = L(R) \cdot L(R')$，$L(R^*) = L(R)^*$，$L(R|R') = L(R) \cup L(R')$，以及 $L(R\{m, n\}) = \bigcup_{m \le i \le n} L(R)^i$。我们将 $|R|$ 简单理解为定义字符串的长度，例如 $|(ab)\{10\}| = 8$。我们将 $\sharp R$定义为 R 中字符出现次数的数量，形式化地，$\sharp a = 1$ （对于 $a \in \Sigma$），$\sharp\varepsilon = 0$，$\sharp(R) = \sharp R\{m,n\} = \sharp R$，以及 $\sharp R \cdot S = \sharp R|S = \sharp R + \sharp S$。

一个（非确定性）自动机（NA）是一个元组 $A = (Q\ \Delta, I, F)$，其中 Q 是状态集合，Δ 是形如 $q \xrightarrow{\{a\}} r$ 的转移集合，其中 q、$r \in Q$ 且 $a \in \Sigma$，$I \subseteq Q$ 是

初始状态集合，且 $F \subseteq Q$ 是终止状态集合。自动机 A 在单词 $w = a_1 \dots a_n$ 上从状态 p_0 运行到 p_n，$n \geq 0$ 是一个转移序列 $p_0 \xrightarrow{a_1} p_1, p_1 \xrightarrow{a_2} p_2$，……，$p_{n-1} \xrightarrow{a_n} p_n$ 来自 Δ。空序列是在 $p_0 = p_n$ 上运行 ε。如果 $p_0 \in I$ 且 $p_n \in F$，则该运行是接受运行；自动机 A 的语言 L(A) 是所有 A 存在接受运行的单词组成的集合。如果存在一条从 I 到状态 q 的运行，则称状态 q 是可达的。NA 的大小 $|A|$定义为其状态数加上转移数。自动机是确定性的（DA），当且仅当 $|I| = 1$，并且对于每个状态 q 和符号 a，Δ 至多存在一个转移 $q \xrightarrow{a} r$。子集构造将 NA 转换为具有相同语言的 DA：$DA(A) = (Q^{D}, \Delta^{D}, I^{D}, F^{D})$，其中 $Q^{D} \subseteq P(Q)$，且 Δ^{D} 是满足以下条件的最小状态集合和转移集合：$I^{D} = \{I\}$；Δ^{D} 对于每个 $a \in \Sigma$ 以及每个 $S \in Q^{D}$，都有从 $S \xrightarrow{a} \{s' \mid \text{到 } s \in S \wedge s \xrightarrow{a} s' \in \Delta\}$的转移；并且 $F^{D} = \{S \in Q^{D} \mid S \cap F \neq \emptyset\}$。当状态集合 Q 有限时，我们称其为（确定性）有限状态自动机（NFA、DFA）。[1]

本文研究快速模式匹配问题，基本上这是一个成员资格测试：给定一个正则表达式 R 和文本 w，判断 $w \in L(R)$。虽然 w 可能非常长，但 R 通常很小，因此对 $|w|$ 的依赖是复杂度中的主要因素。离线 DFA 模拟所需时间与 $|w|$ 成线性关系。它会（1）将 R 编译为 NFA NFA(R)，（2）将其确定化，以及（3）在 w 上执行 DFA 运行（即在 w 上模拟 DFA），所有步骤的时间和空间复杂度均为 $\Theta(2^{|\mathrm{NFA}(R)|} + |w|)$。然而，确定化的代价关于 $|NFA(R)|$ 呈指数增长，因此在实践中不切实际。现代匹配器（如 Grep 或 RE2 [17,16] ）因此采用在线 DFA 模拟技术，仅构造处理 w 时实际用到的 DFA 部分。这将复杂度降至 $O(\min(2^{|\mathrm{NFA}(R)|} + |w|, |w| \cdot |NFA(R)|))$（min 的第一个操作数表示在构造完整 DFA 时显式确定化的代价，加上 DFA 模拟的代价；第二个操作数表示在线 DFA 模拟的代价，因为每一步都可能需要构造一个新的 DFA 状态和转移，所需时间为 $O(|NFA(R)|)$）。对于计数正则表达式，因子 $|NFA(R)|$ 关于 $\max_R$ 呈线性依赖（或者在计数嵌套时呈更高阶依赖），因此关于 $|R|$ 呈指数增长。这使得计数在实践中非常棘手 [23,13,1]。我们将介绍一种匹配算法，它对于某一特定类别的正则表达式而言速度很快，也就是说，其运行时间关于 $|w|$ 仍然是线性的，但与 $\max_R$ 无关。

3 计数自动机

我们采用对 [23] 中计数自动机和计数集自动机的重新表述后的定义。我们将把它们作为寄存器自动机这一通用概念的特殊情形来介绍。

定义 1（寄存器自动机）。在一阶逻辑 Γ 下，通过解释 $\mathbb{I}$ 运行的寄存器自动机（RA）是一个元组 $A = (X, Q, \Delta, I, F)$，其中 X 是称为寄存器的变量集合；Q 是有限状态集合；Δ 是形如 $q \xrightarrow{a, \varphi, u} p$ 的有限转移集合，其中 $p, q \in Q$，$a \in \Sigma$，$u: X \to$ 项 $_{\Gamma, X}$ 是一个更新，且 $\varphi \in QFF_{\Gamma, X}$ 是一个守卫条件；I 是初始配置集合，其中配置是形如 $(q\ m)$ 的有序对，其中 $q \in Q$，且 $m: X \to D_{\mathbb{I}}$ 是一个称为存储器的寄存器赋值；$F: Q \to QFF_{\Gamma, X}$ 是一个终止条件赋值。

[1] 为避免对后文定义的寄存器自动机、计数器自动机和计数集自动机概念施加人为限制，我们在基本定义中不要求有限性。

A 的语言 L(A) 定义为其配置自动机 Conf(A) 的语言。Conf(A) 的状态是 A 的可达配置。I 是 Conf(A) 的初始状态集合。当且仅当 $(q\ m)$ 可达，且 A 存在一个转移 $\delta = q \xrightarrow{\{a,\varphi,u\}} q' \in \Delta$，使得 (q', m') 是 $(q\ m)$ 在 δ下的像时，它具有转移 $(q\ m) \xrightarrow{\{a\}} (q', m')$，记作 $(q', m') = \delta(q\ m)$，其含义是：(1) δ在 $(q\ m)$ 中启用，$m \models \varphi$；并且 (2)$m' = u(m)$，即对于每个 $x \in X$，都有 $m'(x) = u(x)^{\mathbb{I}}(m)$。对于配置集合 C，我们记 $\delta(C) = \{\delta(c) \mid c \in C\}$。如果 $m \models F(q)$，则配置 $(q\ m)$ 是终态。A 的运行是指 Conf(A) 的运行。RA A 是确定性的，当且仅当 Conf(A) 是确定性的。RA 的大小为 $|A| = |Q| + \sum_{\delta\in\Delta} |\delta|$，其中 $|\delta|$ 是更新和守卫条件大小之和。

定义 2（计数自动机）。计数自动机（CA）是一种带有寄存器的自动机，这些寄存器称为计数器，通过计数语言 Γ_{cnt} 进行操作。该语言包含一元递增函数（记作 $x+1$）、常量 0 和 1，以及谓词 $x > k$ 和 $x \leq k$（$k \in \mathbb{N}$），并在自然数上采用标准解释，我们将其记作 $\mathbb{I}_{cnt}$。

带计数的正则表达式可以通过多种方法转换为计数自动机（[23,1,32,33]）。我们使用经过轻微调整的 [32]——将 Glushkov 算法扩展到计数的版本 [21]。对于正则表达式 R，该算法生成一个计数自动机 $CA(R) = (X, Q, \Delta, \{\alpha_0\}, F)$。图 1 展示了这样一个计数自动机的示例。附录 B.1 详细讨论了这一构造；此处我们仅概述第 4–6 节所需的重要性质：

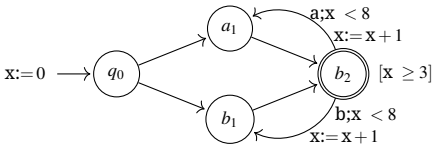


图 1：$R = ((a|b)b)\{3, 8\}$对应的计数自动机 CA(R)。除 b_2 外，所有状态的接受条件均为 $\perp$；b_2 的接受条件写在方括号中。

1. R 中每个计数子表达式 $T\ \{\min_S, \max_S\}$ 的出现位置 S 都对应一个唯一的计数器 x_S 和计数自动机 CA(R) 的一个子结构 A_S。在 A_S 之外，x_S 处于非活动状态（一个死变量），其值为 0；进入 A_S时，其值被赋为 1；每次经过 A_S 进行迭代时，读取一个来自 L(T) 的单词，同时 x_S 的值递增。我们对 [32] 所做的细微修改与以下事实有关：原始算法将非活动计数器赋值为 1，而我们需要赋值为 0。2. CA(R) 至多有 $\sharp R + 1$ 个状态、$cnt_R \cdot \sharp R^2$ 条转移和 cnt_R 个计数器。如果 R 是扁平的，则至多有 $\sharp R^2$ 条转移。3. CA(R) 只有一个初始配置 $\alpha_0 = (q0, \mathfrak{s}_0)$，使得对于每个 $x_S \in X$，$\mathfrak{s}_0(x_S) = 0$。4. 守卫条件和终止条件均为合取式；对于每个计数器 $x_S \in X$，至多包含一个 $\min_S \leq x_S$ 或 $\max_S > x_S$ 形式的合取项。转移更新对 $x_S \in X$ 的赋值只能是 0、1、x_S或 $x_S + 1$。若赋值为 x_S（即保持不变），则对 x_S 没有守卫条件；若 x_S 被重置为 0 或 1，则具有守卫条件 $x_S \geq \min_S$（计数器达到下界之前不能重置）；若 x_S 被赋值为 $x_S + 1$，则具有守卫条件 $x_S < \max_S$（计数器永远不能超过其最大值 $\max_S$）。因此，计数器永远不能超过 $\max_R$。

5. R 的扁平性意味着 CA(R) 的配置至多为一个计数器赋予非零值。这意味着 Conf(CA(R)) 的大小至多为 $|Q| \cdot \max_R$ 个状态，而且 CA(R) 也是笛卡尔的。这一性质将在第 4 节中定义，并且对于我们的计数自动机确定化的正确性至关重要（第 6 节中的定理 3）。

通过子集构造可以得到一个形式为 DA(Conf(CA(R))) 的确定有限自动机（DFA），这称为显式确定化。由于 Conf(CA(R)) 的大小中有因子 $\max_R$，即使 R 是扁平的，显式确定化的大小相对于 $\max_R$ 仍呈指数级增长；这意味着相对于 |R| 呈双指数级增长（R 的 $\max_R$ 以十进制数字书写）。如果 R 不是扁平的，则因子 $\max_R$ 将替换为 $(\max_R)^{cnt_R}$。

4 计数器子集构造

本节将 [23] 中的计数自动机确定化改写为一种修正版，该版本构造的机器大小与 $\max_R$ 无关。我们的方法适用于整个笛卡尔计数自动机类（下文定义），因此也适用于所有具有平坦计数的正则表达式。

确定化的主要思路与 [23] 中相同。我们在标准子集构造中加入寄存器，并称之为计数集；它们可以存储由计数自动机的非确定性运行产生的计数器值集合。以计数集作为寄存器的自动机称为计数集合自动机。我们对 [23] 所作的第一项修改，是按状态对计数器进行索引。直观地说，这使我们能够处理诸如 a*(ba |ab){5} 这样的情况：读入第一个 ab 后，计数器可能递增，也可能不递增（b 是否为计数子表达式的第一个字母）。这会违反 [23] 中计数自动机所需的一致性性质，即非确定性计数自动机运行产生的值集合必须对每个计数自动机状态都相同。在我们的修正版中，不同状态的值分别存储在以相应状态为索引的寄存器中，因此这些值可以不同。接下来，为处理带索引的计数器，我们必须引入计数器的一般赋值，以允许将其他计数器的并集赋给某个计数器。[2] 直观地说，当一次运行以非确定性方式分支到两个状态时，每个分支都需要继续使用一份独立的副本，该副本来自以该分支状态为索引的计数器中存储的集合。分支重新汇合时，则使用集合并集。这带来了一个技术难题：如何在使用集合并集和复制操作时，快速模拟计数集合自动机？我们在第 5 节和第 6 节介绍解决方案。

定义 3（计数集合自动机）。计数集自动机（CSA）是一种自动机，其寄存器通过计数集合语言 $\Gamma_{\mathtt{set}}$ 在数集解释 $\mathbb{I}^{\emptyset}_{\mathtt{cnt}}$ 下进行操作；该语言 $\Gamma_{\mathtt{set}}$ 通过加入常量 $\emptyset$、二元并 $\cup$ 和集合过滤函数 ∇_p 扩展了计数语言 $\Gamma_{\mathtt{cnt}}$，其中 p 是 $\Gamma_{\mathtt{cnt}}$ 的谓词符号。为简单起见，我们将转移更新赋给计数器的项限制为以下形式：$\mathtt{t} = \mathtt{t}_1 \cup \cdots \cup \mathtt{t}_n$，其中每个 $\mathtt{t}_i$ 要么是 (a) $\Gamma_{\mathtt{cnt}}$ 或 $\emptyset$ 的一个项，要么是 (b) 形如 $\nabla_{p(t')}$ 的项，其中 $\mathtt{t}'$ 是 $\Gamma_{\mathtt{cnt}}$ 的一个项。每个 $\mathtt{t}_i$ 都称为 t 的一个 r-项。

$\mathbb{I}_{\mathtt{set}}$ 的论域是自然数集合，$\mathcal{P}(\mathbb{N})$。$\Gamma_{\mathtt{cnt}}$ 在 $\mathbb{I}_{\mathtt{set}}$ 下的谓词和函数的解释，源自相同谓词和函数的基础数值解释：在基础语义下，函数返回作用于论证中集合所得的像，$\mathtt{f}^{\mathbb{I}_{\mathtt{set}}}(S) = \{\mathtt{f}^{\mathbb{I}_{\mathtt{cnt}}}(n) \mid n \in S\}$。如果集合中的某个元素满足该谓词的基础语义，则该集合满足此谓词，$\mathtt{p}^{\mathbb{I}_{\mathtt{set}}}(S) \iff \exists e \in S\colon \mathtt{p}^{\mathbb{I}_{\mathtt{cnt}}}(e)$。随后，过滤器会滤除不满足其谓词基础语义的值，$\nabla^{\mathbb{I}_{\mathtt{set}}}\mathtt{p}\,(S) = \{e \in S \mid \mathtt{p}^{\mathbb{I}_{\mathtt{cnt}}}(e)\}$。最后，$\emptyset$ 被解释为

[2] [23] 只能将一个常量或 x 当前值的函数赋给计数器 x。

空集，而 $\cup$ 被解释为集合的并集。我们用 $\mathfrak{s}$ 表示CSA的记忆，以区别于CA的记忆。我们用DCSA作为确定性CSA的缩写。

不太正式地说，CSA 的寄存器保存数字集合，并通过将所有值递增 $x+1$、赋值为常值集合 $\{0\}$、$\{1\}$ 和 $\emptyset$（分别记为 0、1 和 $\emptyset$）、过滤掉小于或大于某个常量的值（分别记为 $\nabla_{x\leq k}(x)$ 和 $\nabla_{x<k}(x)$），以及测试是否存在满足 $x \leq k$ 或 $x < k$ 的值（$k \in \mathbb{N}$）来进行操作。

我们将介绍一种算法，将在本节余下部分固定的 CA $A = (X, Q, \Delta, I, F)$ 确定化为 DCSA $\mathrm{DCSA}(A) = (X^{\emptyset}, Q^{\emptyset}, \Delta^{\emptyset}, I^{\emptyset}, F^{\emptyset})$。我们假设 Δ 中转移的守卫条件和终止条件均具有 $\bigwedge_{x\in Y} p_x[x], Y \subseteq X$ 的形式，即合取项中每个计数器至多包含一个原子谓词。对于任意正则表达式 R，所有 CA(R) 都满足这一条件（参见第 3 节中 CA(R) 的性质列表）。[3]

DCSA(A) 的运行将编码 DA(Conf(A)) 的运行，这些运行是通过对 A 进行显式确定化得到的。回顾一下，DA(Conf(A)) 的状态是 A 的配置集合，即由一个状态和一个计数器赋值组成的对 $(q, \mathfrak{m})$。DCSA(A) 将把 DA 状态中的计数器值集合表示为其寄存器的运行时值。

特别地，对于 CA 的每个状态 q 和计数器 x，DCSA(A) 都有一个寄存器 x_q，在该寄存器中，读取单词 w 后，它会记住基础 CA 在 w 上以 q 结束的运行中，x 所达到的所有值的集合。因此，我们有 $X^{\emptyset} = \{x_q \mid x \in X \wedge q \in Q\}$

定义 4（CA配置集合的编码）。DA(Conf(A)) 的一个状态 $S = \{(q_i, \mathfrak{m}_i)\}_{i=1}^{n}$ 被编码为 DCSA(A) 配置 $\mathrm{enc}(S) = (\{q_i\}_{i=1}^{n}, \mathfrak{s})$，其中 $\mathfrak{s}(x_q) = \{\mathfrak{m}_i(x) \mid q_i = q\}_{i=1}^{n}$。

由于与状态 q 一同出现的赋值集合会被分解为各个计数器的值集合，因此它忽略了不同计数器的值之间的关系。例如，在 DA 状态 $S_1 = \{(q, \{x \mapsto 0, y \mapsto 0\}), (q, \{x \mapsto 1, y \mapsto 1\})\}$ 中，x 和 y 的值要么同时为 0，要么同时为 1，但 $\mathrm{enc}(S_1) = (q, \{x_q \mapsto \{0,1\}, y_q \mapsto \{0,1\}\})$ 不会保留这一信息。它与另一个 DA 状态 $S_2 = \{(q, \{x \mapsto 1, y \mapsto 0\}), (q, \{x \mapsto 0, y \mapsto 1\})\}$ 的编码完全相同。这与所谓的笛卡尔抽象中信息的丢失相同。因此，只有当我们假设 DA(A) 的状态内部计数器之间的关系始终不受限制时，编码才是精确且无歧义的，因为此时没有信息会丢失。随后，我们将 CA 称为笛卡尔的，定义如下。此时，编码函数是无歧义的，我们将其逆函数称为解码，记作 dec。

定义 5（笛卡尔计数自动机）。假设 A 的计数器集合为 $X = \{x_i\}_{i=1}^{m}$，则 A 的配置集合 C 是笛卡尔的，当且仅当对于 A 的每个状态 q，都存在集合 N_1、……、$N_m \subseteq \mathbb{N}$，使得 $(q, \{x_i \mapsto n_i\}_{i=1}^{m}) \in C$ 当且仅当 $(n_1, \ldots, n_m) \in N_1 \times \cdots \times N_m$。当且仅当 DA(Conf(A)) 的所有状态都是笛卡尔的时，计数自动机 A 才是笛卡尔的。

例如，上面的 DA 状态 S_1 和 S_2 不是笛卡尔的，而 $S_1 \cup S_2$ 是笛卡尔的。

与第 3 节（附录 B.1）中讨论的 [23], 正则表达式到计数自动机构造类似，当应用于扁平正则表达式时，我们的正则表达式到计数自动机构造返回一个笛卡尔计数自动机。

[3] 每个计数自动机都可以通过将公式转换为析取范式，并为各个子句创建转换/状态的克隆，转换为这种形式。

笛卡尔计数自动机的子集构造。下面的算法是子集构造的一般化。我们将 $\mathrm{index}_q(t)$ 表示为由 t 得到的项：将每个变量 $x \in X$ 替换为 x_q；对于公式，类似地定义 $\mathrm{index}_q(\varphi)$。我们有 $Q^\emptyset \subseteq \mathcal{P}(Q)$、初始配置 $I^\emptyset = \{\mathrm{enc}(I)\}$，并且将 $R \in Q^\emptyset$ 的终止条件指定为其元素的终止条件的析取，即 $F^\emptyset(R) = \bigvee_{q \in R} \mathrm{index}_q(F(q))$。

我们将构造 DCSA(A)，它是确定性的，并且其运行编码 DA DA(Conf(A)) 的运行。Conf(DCSA(A)) 将与 DA(Conf(A)) 同构。为此，对于 DA(Conf(A)) 的每个转移 δ，我们需要在同一个字母上设置一个 DCSA(A) 的唯一转移；该转移在源配置的编码中启用 δ，并生成目标配置的编码 δ。换言之，对于 DA(Conf(A)) 的每个转移 $\mathrm{dec}(R\ s) \xrightarrow{\{a\}} \mathrm{dec}(R', s')$，我们需要一个唯一的转移 $\delta' = R \xrightarrow{\{a,\varphi,u\}} R' \in \Delta^\emptyset$，其中 $(R', s') = \delta'(R\ s)$。该转移 δ' 将通过总结 $\mathrm{dec}(R\ s)$ 的 CA 配置中所有已启用的基础 CA a-转移的效果来构造。

为了构造转移 δ'，我们首先将每个基础转移 $\delta = q \xrightarrow{\{a,\varphi_\delta,u_\delta\}} r \in \Delta$ 转换为其集合版本 $\delta^\emptyset$；该集合版本旨在将配置的一个（笛卡尔）集合 C 的编码 enc(C) 转换为其像的集合的编码 $\mathrm{enc}(\delta(C))$，并且当 δ 至少对 C 中的一个配置启用时启用。为此，假设 $\varphi_\delta = \bigwedge_{x \in X} p_x[x]$，，我们（1）从 u_δ 构造更新 u_δ^∇：在每个 $u_\delta(x)$ 中，将 $x \in X$ 变量 $y \in X$ 替换为其过滤版本 $\nabla_{p_y}(y)$；（2）为标记当前状态的寄存器添加索引，从而得到转移 $\delta^\emptyset = q \xrightarrow{\{a,\varphi_\delta^\emptyset,u_\delta^\emptyset\}} r$，其中 $\varphi_\delta^\emptyset = \mathrm{index}_q(\varphi_\delta)$，并且 $u_\delta^\emptyset$ 将每个 x_r、$x \in X$ 赋值为项 $\mathrm{index}_q(u_\delta^\nabla(x))$。

状态 $Q^\emptyset$ 和转移 $\Delta^\emptyset$ 随后被构造为满足以下条件的最小集合：$\mathrm{enc}(I) \in Q^\emptyset$，并且对于每个 $R \in Q^\emptyset$，以及每个 $a \in \Sigma$，其出边转移按如下方式构造。令 $\{q\ j \xrightarrow{\{a,\varphi_j,u\}} r_j\}_{j \in J}$ 为某个索引集合 J，表示 R 的构成 a-转移集合，即所有 $\delta^\emptyset$ 其中 $\delta \in \Delta$ 起源于 R 的 a-转移。为实现确定性，对于每个最小项 $\psi \in \mathrm{Minterms}(\{\varphi_j\}_{j \in J})$，$\Delta^\emptyset$ 具有转移 $R \xrightarrow{\{a,\psi,u\}} R'$。更新 u 和目标 R' 根据集合 $\{q\ j \xrightarrow{\{a,\varphi_j,u\}} r_j\}_{j \in K}$ 构造，其中 $K \subseteq J$，该集合由守卫条件 φ_j 与最小项 ψ 相容的构成转移组成，即 $\psi \wedge \varphi_j$ 可满足。R' 是其目标状态的集合，即 $R' = \{r_j\}_{j \in K}$；而 u(x) 合并其所有更新项 $u_j(x)$，即 $u(x) = \bigcup_{j \in K} u_j(x)$，对于每个 $x \in X^\emptyset$。

示例 1。在展示转移更新时，我们用 $x := t$ 表示 $u(x) = t$，并省略 CSA 中的赋值 $x := \emptyset$。

令 $R = \{p, q\}$，且从 R 出发的 a-转移为 $q \xrightarrow{\{a,\top,x:=x+1\}} s$、$p \xrightarrow{\{a,x<n,x:=x+1\}} r$ 和 $p \xrightarrow{\{a,x \geq m,x:=1\}} s$。它们为 R 和 a 诱导出三个组成转移：$q \xrightarrow{\{a,\top,x_s:=xq\}} s$、$p \xrightarrow{\{a,x_p<n,xr:=\nabla x<n(xp)+1\}} r$ 以及 $p \xrightarrow{\{a,x_p \geq m,xs:=1\}} s$。对于以下每个最小项 ψ，均构造一个转移 $R \xrightarrow{\{a,\psi,u'\}} R'$：$x_p<n \wedge x_p \geq m$、$\neg x_p<n \wedge x_p \geq m$、$x_p<n \wedge \neg x_p \geq m$、$\neg x_p<n \wedge \neg x_p \geq m$。对于第一个最小项，三个组成转移均相容，因此更新 u′ 为 $x_r := \nabla_{x<n}(x_p) + 1$；$x_s := x_q \cup 1$（$x_r$ 的更新取自通向 r 的第一个组成转移，x_s 的更新是通向 s 的后两个转移的更新的并集），目标状态为 $R' = \{r, s\}$。□

DCSA(A) 是确定性的，因为它只有一个初始配置，并且源自同一状态的转移的守卫条件都是最小项。DCSA(A) 的大小显然只取决于 A 的大小，而不取决于语言的解释。特别是，

当 A 是某个正则表达式 R 的 CA(R) 时，其大小不取决于 $\max_R$。下面的定理将在附录 A 中证明。[4]

定理 1. DCSA(A) 是确定性的，$|\mathrm{DCSA(A)}| \in O(2^{|A|})$，并且如果 A 是笛卡尔的，那么 $L(A) = L(\mathrm{DCSA(A)})$。

由于对于具有平坦计数的正则表达式，我们的正则表达式到计数自动机算法总是返回一个笛卡尔计数自动机，因此我们可以将它们转换为 DCSA。

5 计数集合自动机的快速模拟

本节讨论如何高效地模拟 DCSA 在给定单词上的运行，以实现快速匹配。设单词 $w = a_1 \cdots a_n$，以及 DCSA $A = (X, Q, \Delta, \{\alpha_0\}, F)$。我们希望构造该 DCSA 在 w 上的运行，并检验到达的配置是否为接受配置。我们的目标是使运行时间与 $|w|$ 成线性关系，且不受运行时存储在 A 的寄存器中的集合大小影响。

我们假设 A 的初始配置 α_0 将每个寄存器赋为单元素集合或空集。对于由第 4 节和第 6 节的算法从 CA(R) 构造的 CSA（其中 R 是任意正则表达式），这一假设均成立。[5]

具体而言，模拟会维护一个配置 $\alpha = (q, s)$，其初始值为 α_0；对于从 1 到 n 的每个 i，它都会构造 Conf(A) 的转移 $\alpha \xrightarrow{\{a_i\}} \alpha'$，并将 α 替换为后继配置 $\alpha' = (q', s')$。我们采用 [23], 中的快速模拟关键技术：使用偏移列表数据结构来存储数字集合，支持常数时间内对所有值加 0 或 1、比较最大值与常量、重置以及递增所有值。问题在于，新增的集合并集和集合复制仍然需要与集合大小成线性关系的时间，因此也与最大计数器界限成线性关系。我们将证明，在下文引入的条件下，可以完全避免集合复制，并且并集的开销可以通过集合递增的开销进行摊销。这将再次使我们能够在与 $\max_A$ 无关的时间内进行 CSA 模拟，其复杂度属于 $O(|A| \cdot |w|)$。

首先，我们定义 CSA 的一项足以支持快速模拟的性质：其转移上的更新不会复制计数器。

定义 6（计数器复制）。如果对于某个转移 $q \xrightarrow{\{a, \varphi, u\}} r$，某个计数器出现在 u 的像中两次，即它出现在某个 u(x) 的两个 r-项中，或者对于 $x \neq y$，它既出现在 u(x) 中，也出现在 u(y) 中，则称 CSA 复制计数器。不复制计数器的 CSA 称为非复制型 CSA。

例如，$\{x \mapsto x;\ y \mapsto x + 1\}$ 和 $\{x \mapsto x \cup x + 1,\ y \mapsto y\}$ 是 x 被复制的更新；$\{x \mapsto x + 1,\ y \mapsto y\}$ 不是复制更新。

[4] 值得注意的是，正如我们的确定化表述所显示的那样，该构造与用于操作寄存器的具体一阶逻辑及其解释无关。确定化可以应用于任何符合寄存器自动机定义的自动机。可以使用其他函数和测试来操作这些数，也可以用实数替代自然数等。计数集合自动机本身就是寄存器自动机的一个实例。还可以考虑下推自动机，或者经过少量修改的带寄存器的数据词自动机变体。[5] 这是一个重要的技术性假设，因为只有这样，初始集合的并集才不会影响模拟的整体复杂度。

偏移列表数据结构。 [23] 的偏移列表数据结构支持对所有元素递增、重置为 $\emptyset$ 或 $\{0\}$ 或 $\{1\}$、加上 0 或 1，以及将最大值与常量进行比较等集合运算的常数时间实现。

它为每个计数器 $x \in X$ 分配一个指针 $\mathrm{ol}(x)$，指向偏移列表对 (o_x, l_x)，其中偏移量为 $o_x \in \mathbb{N}$，有序列表 $l_x =$ 包含整数 $m1$、……、m_k。实现该列表的数据结构需要支持对首元素和末元素的常量访问、指针的前向和后向迭代，以及在指向某元素的指针处或其之前插入/删除元素。例如，维护指向首元素和末元素的指针的双向链表即可满足这些要求。偏移列表对表示集合 $\mathfrak{s}(x) = = \{m_1 + o_x$、……、$m_k + o_x\}$。两个此类集合的并集的计算时间仍与其大小呈线性关系，但我们将证明，如果 CSA 不复制计数器，则集合并的成本可以由递增操作的成本进行摊销。

查找 CSA 转移并计算更新。计算 α'的第一步是从 α 中查找转移 $q \overset{\{a_i, \varphi, u\}}{\mapsto} q' \in \Delta$，即从 q 出发唯一启用的 a_i-转移，也就是说满足 $\mathfrak{s} \models \varphi$的转移。最简单的算法遍历$\Delta$ 的转移，并针对每个转移测试 $\mathfrak{s}$ 是否满足其守卫条件。计算原子计数器谓词 p 的成本是常量，即判断 $\mathfrak{s} \models p$ 是否成立的成本为常量：由于列表 l_x是有序的，我们只需访问首元素或末元素以及偏移量，即可分别判断 $x < n$ 或 $x \geq n$。由此，计算 φ 的成本与φ的大小呈线性关系。遍历 Δ 的转移的成本随后与这些转移大小之和呈线性关系，该值属于 $O(|A|)$。[6]

找到 $q \overset{\{a_i, \varphi, u\}}{\mapsto} q'$ 后，我们对其更新求值，以计算 $\mathfrak{s}'$ 并将 α' 计算为 $(q', \mathfrak{s}')$。我们将解释该算法，并论证计算 $\mathfrak{s}'$的摊销成本为 $O(|X|)$。对于每个 $x \in X$，我们通过计算 $u(x)$ 中的所有 r项、将结果求并集，并将该并集赋给 $\mathrm{ol}(x)$，来执行更新。

首先，我们论证，对 $u(x)$ 中的一个 r项 t 求值，即计算 $t(\mathfrak{s})$，所需的摊销时间为常数。由于计数器是非复制的，我们可以原位计算每个 r项 $t[y]$ 的值。也就是说，我们修改偏移列表对 (o_y, l_y)，并返回指针 $\mathrm{ol}(y)$。求值 $t[y]$ 后，可以丢弃 y 的原始值，因为 y 不会出现在任何其他 r项中。共有 5 种情况：(1) 如果 t 是 0 或 1，那么我们返回一个指向新建偏移列表对的指针，其偏移量为 0，列表中分别仅包含 0 或 1。此操作可在常数时间内完成。

（2）如果 t 是 $y \in Y$，则返回 $\mathrm{ol}(y)$。

（3）如果 t 是 $y + 1$，则 o_y 递增 1。通过常数时间实现递增，正是将列表与偏移量配对的原因。

(4) 如果 t 为 $\nabla_p[y]$, , 则 l_y 会通过原子谓词 p 进行过滤。使用谓词 $x \geq n$ 进行过滤时，会利用 l_y的有序性不变量。具体做法是重复执行以下步骤：i) 检查列表头部是否小于 $n - o_y$；ii) 如果是，则移除头部，否则终止迭代。每次迭代的时间均为常数时间：移除元素的迭代成本由将该元素添加到列表中的成本摊销。剩下的只有最后一次迭代的常数成本，该次迭代用于检测是否存在大于或等于 $n - o_y$ 的元素，或者列表是否为空。使用 $x < n$ 进行过滤时情况类似（迭代会检查并移除最后一个元素，而不是列表头部）。

[6] 在后续章节中，我们将展示在对由我们的构造生成的 CSA 守卫施加额外假设的情况下，查找转移的更优复杂度界。

(5) 如果 t 为 $\nabla_p(y)+1$，则在上述常量过滤器之后应用常量增量的构造。

接下来，我们论证，计算 u(x) 中 r-项的值的并集，其开销可以由递增项的求值开销摊销。令 l1, . . . , l_n 为上述算法计算得到的 u(x) 中各项值的偏移列表表示。它们的并集的偏移列表表示通过一系列合并计算得到，即 merge(l_1, merge(l2, . . . merge(l_{n-1}, l_n) . . .))。具体而言，给定指向两个偏移列表 l、l′的指针，merge(l, l′) 会实现它们的并集：它选择表示最大值较大的集合的偏移列表，假设该列表为 l，然后将另一个列表 l′所表示的元素插入其中。我们称 l′ 被合并到 l 中。该操作通过标准的有序列表合并完成，时间为 O(|l′|)，其中 |l′| 是 l′的长度。由于 l′ 没有重复项且最小值为 0，因此 O(|l′|) ⊆O(max(l′))，其中 max(l′) 是最大元素。

O(max(l′)) 的成本由评估增量的成本进行摊销。l′ 处的偏移列表对至少经历了 max(l′) − 1 次增量，因为其中插入的唯一元素是 0、1，或者在合并期间来自小于 max(l′) 的其他集合的元素。这些对 l′ 的增量构成了用于支付将 l′ 合并到 l 中的预算。合并后，l′ 的偏移列表对会被丢弃（由于 CSA 是非复制的，不再需要它），因此该预算只使用一次。最后，将并集赋值给 c，只需将一个指针常数时间内赋值给合并返回的偏移列表即可。

模拟的总体复杂度。定义在模拟的一步中对计数器 x ∈X 进行操作的成本 cost(x)，它等于以下各项成本之和：(1) 评估所有包含 c 的 r-项；(2) 将它们的偏移列表合并到其他偏移列表中；(3) 为 u(x) 中的 0 或 1 这类项创建偏移列表，并将其合并到其他偏移列表中；(4) 将 u(x) 的结果赋值给 x。处理单个字母 a_i 的成本随后是 $\sum_{x\in X}$ cost(x) 之和，而 $|w|\cdot\sum_{x\in X}$ cost(x) 是整个模拟的成本。由于 CSA 是非复制的，且评估单个 r-项的成本是摊销常数时间，因此 (1) 的成本为摊销常数时间。(2) 的成本由 (1) 中的增量进行摊销。(3) 中单元素集合的创建和插入在 u(x) 中至多发生两次，耗时为常数时间。(4) 中的指针赋值耗时为常数时间。因此，cost(x) 的成本是摊销常数时间，评估更新 u 的摊销时间为 O(|X|)，模拟过程中执行更新的成本为 O(|X| · |w|)。根据上述分析，通过评估守卫条件来选择转移的成本为 O(|A| · |w|)。在到达的配置上测试接受条件的成本，通过类似论证可得为 O(|A|)。

定理 2。如果 A 是非复制的，那么其在 w 上的模拟耗时为 O(|A| · |w|)。

6 增强确定化

本节将在第 4 节的子集构造基础上加入优化措施，以避免计数器复制，从而扩展可通过模拟 CSA 快速匹配的正则表达式类别。这些优化针对第 3 节所列、具有 CA(R) 特殊性质的正则表达式 R 的计数自动机进行了定制。

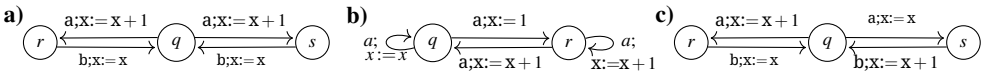


图2：计数器复制源自计数自动机的子结构。

优化措施的直观理解。图 2 通过计数自动机的简化片段，展示了增强构造中计数器复制的产生及其消除方法，所采用的技术包括计数器共享和增量延迟。

在 a) 中，DCSA(CA(R)) 的转移为 $\{q\}\vdash_{\{a,x_r:=xq+1,xs:=xq+1\}}\{r,s\}\vdash_{\{b,x_q:=xr\cup xs\}}\{q\}$。第一个转移复制 x_q 的全部内容，第二个转移则对这两个集合取并集。这两个转移的开销都很大。通过检测 x_s 和 x_r 的值相同（它们由语法相同的更新生成），并将这些值存储在共享计数器 $x_{\{s,r\}}$ 中，可以对其进行优化。这样得到的转移为$\{q\}\vdash_{\{a,x_{\{r,s\}}:=x_{\{q\}}+1\}}\{s,t\}\vdash_{\{b,x_{\{q\}}:=x_{\{r,s\}}\}}\{q\}$，其中复制和并集操作都被消除。

图 b) 进一步说明了为什么计数器 x_P, $P \subseteq Q$ 表示原始计数器 x_p, $p \in P$ 之间共享的值集合。也就是说，x_P 并不总是保存计数器 x_p, $p \in P$ 中存储的完整集合。如果它们的值不相同，它只存储它们的交集。每个 x_p 的值随后被划分到多个共享计数器 x_S ，其中 $p \in S$。在 b) 中，DCSA(CA(R)) 的转移为 $q\vdash_{\{a,x_q:=xq;xr:=1\}}\{q,r\}\vdash_{\{a,x_q:=xq\cup xr+1;xr:=1\cup xr+1\}}\{q,r\}$，这会复制计数器 x_r。计数器共享随后会生成转移 $q\vdash_{\{a,x_{\{q\}}:=x_{\{q\}};x_{\{r\}}:=1\}}\{q,r\}\vdash_{\{a,x_{\{q\}}:=x_{\{q\}};x_{\{r\}}:=1;x_{\{q,r\}}:=x_{\{r\}}+1\}}\{q,r\}$，其中计数器 $x_{\{q\}}$、$x_{\{r\}}$ 分别对应 x_q 和 x_r 独有的子集，而 $x_{\{q,r\}}$ 对应交集。

最后，在 c）中，我们演示增量延迟技术。DCSA(CA(R)) 将具有转移 $\{q\}\vdash_{\{a,x_r:=xq+1,xs:=xq\}}\{s,t\}\vdash_{\{b,x_q:=xr\cup xs+1\}}\{q\}$。由于两条分支上的增量发生在不同时间，在 x_r 的最后一次增量将它们同步之前，x_s 的值会有所不同。我们通过将从 x_q 获得的未递增值存储在由 x_r 和 x_s 共享的计数器中，并记住 x_r 的一次增量已被延后，来避免复制。共享计数器 $x_{\{r^+,s\}}$ 名称中的 $+$ 标记表示这一点。当 x_r 和 x_s 的值同步时（增量也应用于 x_s），我们计算延后的增量，并移除 $+$ 标记。我们将创建转移 $\{q\}\vdash_{\{a,x_{\{r^+,s\}}:=x_{\{q\}}\}}\{s,t\}\vdash_{\{b,x_{\{q\}}:=x_{\{r^+,s\}}+1\}}\{q\}$。如果在同步之前，被标记计数器的值再次被测试或递增，我们就声明发生了不可解决的复制，并中止整个构造（我们只允许延后一次增量）。为防止不必要地出现这种情况，我们让状态记住必须具有空值的计数器，并忽略这些计数器。

增强确定化算法。增强确定化从 $CA(R) = (X, Q, \Delta, \{\alpha_0\}, F)$ 生成 CSA $DCSA^a(CA(R)) = (X^a, Q^a, \Delta^a, \{\alpha_0^a\}, F^a)$。其在 X^a 中的计数器形式为 x_S，其中 $x \in X$ 且 $S \subseteq Q^+$，并且 $Q^+ = Q \cup \{q^+ \mid q \in Q\}$。该算法的指导原则是，$X^a$ 的一个赋值 $\mathfrak{s}^a$ 表示 DCSA(CA(R)) 中 $X^{\emptyset}$ 的计数器的一个赋值 $\mathfrak{s}$，即，对于每个 $x_q \in X^{\emptyset}$，

$$\mathfrak{s}(x_q) = \bigcup_{q\in S, S\subseteq Q^+} \mathfrak{s}^a(x_S) \cup \bigcup_{q^+\in S, S\subseteq Q^+} \{n+1 \mid n \in \mathfrak{s}^a(x_S)\}. \qquad (1)$$

我们将使用一些简化记号。如第 3 节所述，根据 CA(R) 的构造，c 的递增和守卫条件 $x < \max_x$ 始终一起出现在其转移上，且 x 上没有其他守卫条件。因此，在 DCSA(CA(R)) 中，包含递增或过滤的所有项均采用 $\nabla_{x<\max_x}(x_{q^\circ}) + 1$的形式。我们将其记为简写 $x_{q^\circ} \oplus 1$ （我们使用 q° 表示集合 Q^+中的一个元素，可以是 q 或 q^+，其中 $q \in Q$）。

$\mathrm{DCSA}^{\mathfrak{a}}(\mathrm{CA}(R))$ 的状态还将根据 $X^{\mathfrak{a}}$ 中哪些计数器处于活动状态（即可能具有非空值）加以区分。始终取值为 $\emptyset$ 的计数器可以忽略，这简化了转移并降低了无法解决计数器复制的可能性。因此，$\mathrm{DCSA}^{\mathfrak{a}}(\mathrm{CA}(R))$ 的状态形式为 (R,Act)，其中 $R \subseteq Q$ 且 $\mathrm{Act} \subseteq X^{\mathfrak{a}}$ 是活动计数器的集合。

初始配置为 $\alpha_0^{\mathfrak{a}} = (\{q_0\}, \{x_{\{q_0\}} \mid x \in X\})\ \mathfrak{s}_0^{\mathfrak{a}})$，其中 $\mathfrak{s}_0^{\mathfrak{a}}$ 将 $\{0\}$ 赋给每个 $x_{\{q_0\}}$，$x \in X$，并将 $\emptyset$ 赋给 $X^{\mathfrak{a}}$ 中的其他每个计数器。对于每个 $(R, \mathrm{Act}) \in Q^{\mathfrak{a}}$，最终条件赋值 $F^{\mathfrak{a}}((R, \mathrm{Act}))$ 由 $F^{\emptyset}(R)$ 构造而成，方法是将每个谓词 $p[x_q]$ 替换为析取 $p[x_q]^{Act} = \bigvee_{x_S \in Act, q \in S} p[x_S]$，后者按照 (1) 的含义，使用 Act 中的计数器对 $p[x_q]$ 进行编码。

$\Delta^{\mathfrak{a}}$ 中的转移由 $\Delta^{\emptyset}$中的转移构造而成。对于源状态 $(R,\mathrm{Act}) \in Q^{\mathfrak{a}}$、原始转移 $R \xrightarrow{\{a,\varphi,u\}} R' \in \Delta^{\emptyset}$ 以及活动计数器集合 $\mathrm{Act} \subseteq X^{\mathfrak{a}}$，$\Delta^{\mathfrak{a}}$ 具有转移 $(R, \mathrm{Act}) \xrightarrow{\{a,\varphi^{\mathfrak{a}},u^{\mathfrak{a}}\}} (R',\mathrm{Act}')$，其构造如下:

守卫条件 $\varphi^{\mathfrak{a}}$ 由 φ 构成，方法是将每个谓词 $p[x_q]$ 替换为带共享计数器的等价形式 $p[x_q]^{Act}$ （如上文构造 $F^{\mathfrak{a}}$ 时所述）。

更新 $u^{\mathfrak{a}}$ 分三步构造。首先，通过使用共享计数器 $X^{\mathfrak{a}}$表示 u 中的 r-项，由 u 构造更新 u^{sh}。每个$t[x_q]$ 替换为

$$t^{\mathfrak{a}} = \bigcup \Big(\big\{ t[x_S] \mid x_S \in Act, q \in S \big\} \cup \big\{ t[x_S] \oplus 1 \mid x_S \in Act, q^+ \in S \big\} \Big) .$$

注意，所有延迟增量都会在 u^{sh}中求值，并转换为标准增量。如果 u^{sh} 包含一个 r-项 $t \oplus 1 \oplus 1$，即一个双重增量，则整个构造中止，并声明发生无法解决的计数器复制。我们只允许延后一个增量。[7] 否则，我们继续解决计数器复制问题。首先，我们确保每个计数器在更新的像中只以一种 r-项的形式出现。我们收集 u^{sh} 中所有具有冲突增量的 r-项 $x_S \oplus 1$ 组成的集合 Conflict，即同时满足 x_S 也是 u^{sh}的 r-项。在更新 u^+中，冲突增量会被延后。对于 $x \in X$、$q \in Q$ 以及 $u^{\mathrm{sh}}(x_q) = \bigcup T$,

$$u^+(x_q) = \bigcup \big(T \setminus Conflict \big) \text{ and } u^+(x_{q^+}) = \bigcup \big\{ x_S \mid x_S \oplus 1 \in T \cap Conflict \big\} .$$

最终更新 $u^{\mathfrak{a}}$ 随后通过将 u^+ 中复制的 r项 按照共同的 l值进行分组来解决计数器复制问题（我们称 z 为 $u^+(z)$ 的 r项的 l值）。对于 u^+ 的一个 r项 t，令 lval(t) 为其 l值的集合。注意，lval(t) 始终具有$\{x_{q^\circ}\}_{x \in S}$ 的形式，其中某个固定的 $x \in X$（参见第 3 节中 CA(R) 的性质 4）。令 Act′ 为满足 $\mathrm{lval}(t) = \{x_{q^\circ}\}_{x \in S}$ 的计数器 x_S 的集合，其中 t 是 u^+的某个 r项。对于所有 $x_S \in X^{\mathfrak{a}}$，如果

[7]此外，$\mathrm{DCSA}^{\mathfrak{a}}(\mathrm{CA}(R))$ 的转换守卫和终止条件也不得包含 +-标记，因为无论延迟增量如何都对其求值会得出错误结果。不过，由于 CA(R) 的结构性质，在此处检测到双重增量时声明计数器复制即可涵盖这些情况。

$x_S \notin \mathrm{Act}'$，则 $u^{\mathtt{a}}(x_S) = \emptyset$ 否则

$$u^{\mathtt{a}}(x_S) = \bigcup \{\, t \mid t \text{ is an r-term of } u^+ \text{ and } \mathtt{lval}(t) = \{x_{q^\circ}\}_{q^\circ \in S} \,\} .$$

示例 2。设 $R \xrightarrow{\{a,\varphi,u\}} R' \in \Delta^{\mathrm{u}}$ 是示例 1 中创建的，且 $R = \{p, q\}, R' = \{r, s\}, \varphi = x_p < n \wedge x_p \geq m$，以及 $u = \{x_r := x_p \oplus 1, x_s := x_q \cup 1\}$。令 $\mathrm{Act} = \{x_{\{p,q\}}, x_{\{p,q^+\}}\}$。那么 $u^{\mathtt{sh}} = \{x_r := x_{\{p,q^+\}} \oplus 1 \cup x_{\{p,q\}} \oplus 1, x_s := x_{\{p,q^+\}} \oplus 1 \cup x_{\{p,q\}} \cup 1\}$。注意，$u(x_s)$ 中的 x_q 变成了 $x_{\{p,q^+\}} \oplus 1$，对应于 $t^{\mathtt{a}}$ 定义的右侧部分（延后递增 x_{q^+} 在 $u^{\mathtt{sh}}$ 中求值）。注意，r-项 $x_{\{p,q\}} \oplus 1$ 存在冲突，因为 $x_{\{p,q\}}$ 也是 $u^{\mathtt{sh}}$ 的一个 r-项。因此，它在 u^+ 中被延后，即 $u^{\mathtt{sh}}(x_r) = x_{\{p,q\}} \oplus 1 \cup \cdots$ 变成了 $u^+(x_{r^+}) = x_{\{p,q\}}$。我们得到 $u^+ = \{x_r := x_{\{p,q^+\}} \oplus 1, x_s := x_{\{p,q^+\}} \oplus 1 \cup x_{\{p,q\}} \cup 1, x_{r^+} := x_{\{p,q\}}\}$。最后，$u^{\mathtt{a}}$ 将 u^+ 中复制的 r-项按共同的 l值分组：$u^{\mathtt{a}} = \{x_{\{r,s\}} := x_{\{p,q^+\}} \oplus 1, x_{\{s\}} := 1, x_{\{s,r^+\}} := x_{\{p,q\}}\}$。接下来的活动计数器为 $\mathrm{Act}' = \{x_{\{r,s\}}, x_{\{s\}}, x_{\{s,r^+\}}\}$。注意，对于 $x_{\{p,q^+\}}$，p^+ 处的延后递增在这次转移中已同步，而 $x_{\{p,q\}}$ 处的冲突则通过延后递增并用 $^+$ 标记 r 得以解决。□

该算法要么返回 CSA $\mathrm{DCSA}^{\mathtt{a}}(\mathrm{CA}(A))$，要么检测到无法解决的计数器复制；在后一种情况下，$\mathrm{DCSA}^{\mathtt{a}}(\mathrm{CA}(A))$ 不存在。[8] 令 $m = \sharp R$，并回忆 n 表示匹配文本的长度，$|w|$。由于 CA(R) 至多有 m 个状态和 m^2 个转移，对算法数据结构进行基本分析可知，所得 CSA 至多有 2^{2m} 个状态，每个状态至多有 2^{m^2} 条出边转移，每个转移的大小为 $O(m2^m)$。由于 $\mathrm{DCSA}^{\mathtt{a}}(\mathrm{CA}(A))$ 对 $\mathrm{DCSA}(\mathrm{CA}(A))$ 进行编码，因此二者具有相同的语言，并且它也继承了后者的确定性。由于它不复制计数器，因此可以在模式匹配中快速模拟，所需时间与文本长度呈线性关系，且独立于计数器界限。以下定理在附录 C 中得到证明。

定理 3. 对于具有平坦计数的 R，如果 $\mathrm{DCSA}^{\mathtt{a}}(\mathrm{CA}(R))$ 存在，那么它不会复制计数器，其大小为 $O(2^{2^m}m)$，$L(\mathrm{CA}(R)) = L(\mathrm{DCSA}^{\mathtt{a}}(\mathrm{CA}(R)))$，并且可以在长度为 n 的单词 w 上以 $O(2^{2m}mn)$ 的时间进行模拟。

匹配所需的时间等于构造 CSA 及其模拟所需的时间；两者相加后确实很快，不依赖于 k，并且关于 n 呈线性。还应注意，上述指数中的 m 并不是整个正则表达式的大小，而只是计数子正则表达式的大小。

7 具有同步计数的正则表达式

最后，在本节中我们定义具有同步计数的正则表达式类，该类精确刻画了这样一种情况：我们的第 6 节构造所创建的 CSA 不会复制计数器，从而允许快速匹配（就定理 3 的意义而言）。

定义 7（具有同步计数的正则表达式）。当且仅当正则表达式不包含形如 $S\{n,m\}$的子表达式时，它才具有同步计数，其中对于某个 $k \in \mathbb{N}$，来自 $L(S)^k$ 的单词具有来自 $L(S)^{k+1}$的前缀。

[8] 在此处中止构造可以简化描述，但也可以继续进行构造，并返回一个不保证快速模拟的 DCSA。

例如，(ac*){1,4}(ab|ba){3,5}(a(ab)*){2,8} 是一个具有同步计数的正则表达式，因为来自 $L(ac^*)^k$ 的每个单词都必须恰好包含 k 个符号 a，来自 $L(ab|ba)^k$ 的单词必须恰好包含 2k 个符号，而来自 $L(a(ab)^*)^k$ 的单词可以在 a(ab)* 中的第一个 a 处分割且分割方式唯一。相比之下，(a|aa){2,5} 不具有同步计数，因为 a·a·a 是 aa·aa 的前缀。

直观地说，不存在一对经过计数自动机(S{m,n})的路径，它们从同一状态出发，经过同一个单词，最终到达同一状态，但递增次数相差二。在这种情况下，DCSA^a(计数自动机(S{m,n}))就必须延迟两次增量，而我们的构造不允许这样做。下面的定理在附录D中证明。

定理 4. 给定一个具有平坦计数的正则表达式 R，第 6 节的算法当且仅当 R 具有同步计数时返回 DCSA^a(计数自动机(R))。

推论 1。具有扁平同步计数的正则表达式拥有快速匹配算法。

证明。由定理 3 和定理 4 得出。

带标记的计数。尽管设计和识别同步计数通常很直观，但有时也可能很棘手。例如，(\\\\d+\\\\.){3} 来自我们实验中使用的真实场景中的正则表达式数据库，具有同步计数，而 ICE Dims.{92}((?(X|\d+)){13}) 则不具有同步计数。[9] 幸运的是，我们研究的绝大多数真实场景中的正则表达式都属于非常容易识别的同步计数子类。最广泛且最容易识别的是采用字母标记计数的正则表达式，其中每个子表达式 S{m,n} 都有一个标记字母集合，使得 L(S) 中的每个单词恰好包含一个标记字母。[10]

不过，标记字母可以推广为标记词，但不能使用可由 L(S) 中多个单词连接而成的标记。必须满足的条件是：$L(S)^k$ 中的任意单词，$k \in \mathbb{N}$，都恰好包含 k 个互不重叠的标记词作为子串。S 的另一个充分性质是其单词长度统一。标记的概念还可以进一步推广，直到标记词的集合由一般正则表达式指定；此时得到的恰好是同步计数。带字母标记计数的正则表达式易于人工识别，也易于机器识别（见附录 F 中一个简单的 $O(|R|^2)$ 时间算法）。

8 实践考量

尽管本文的重点在于论证采用同步计数进行快速匹配的理论可行性，但我们也将论证这些结果具有实际相关性。为此，我们通过实验表明，同步计数和标记计数涵盖了大多数实际正则表达式。我们还论证了使用第 6 节中构造的 CSA 可以高效地进行匹配。

[9] 一种自动识别同步计数的方法是运行第 6 节中的 CSA 到 DCSA 的确定化算法，但其时间复杂度对 |R| 来说是指数级的。[10] 字母标记计数严格包含于 [23] 所推测该算法能够处理的类别。[23] 的推测也不正确，附录 E 已对此作出证明。

8.1 实践中的同步计数出现情况

为了证实同步计数正则表达式的实际相关性，我们使用一个用于字母标记计数的简单检查器（见附录 F）检查了大量实际正则表达式样本。该基准数据集包含超过 540000 个正则表达式，收集自：(1) 对软件项目的大规模分析 [34]；(2) 网络入侵检测系统 Snort [35], 、Bro [36], 、Sagan [37], 所使用的正则表达式以及学术论文 [38,39]；(4) RegExLib 正则表达式数据库 [40]。

在我们能够解析的正则表达式中，有[11], 31 975 包含计数。我们选取了具有平坦计数且计数器上界之和大于 20 的正则表达式（正如 [23] 中所做的那样，以过滤掉上界较小、可以通过计数器展开和传统方法处理的计数）[12]。最终得到 5 751 个正则表达式。其中，只有 46 个正则表达式（0.8%）具有非字母标记的计数。此外，我们人工检查了这些正则表达式，发现其中 22 个具有同步计数。因此，我们仅发现了 24 个具有非同步计数的正则表达式，即在所检查的具有平坦计数的正则表达式集合中占 0.4%。

这 24 个非同步正则表达式列于附录 G 中。其中一些显然可以改写为具有同步计数的形式，例如 (.+){25}(.*) 可以改写为 .{25,}(.*)。我们推测，其中一些实际上可能代表错误，例如 (.*){1,32000}[bc]，其中计数器匹配空字；或者 (\n\s+)(criterion .*\n)(\s.+){1,99}，其中的 \s.+ 原本可能 intended 为 \s\S+（\s 为空白字符，\S 为所有其他字符）。同步计数似乎反映了人们通常书写计数时所依据的直觉，因此报告非同步计数可能有助于识别错误。

采用相同的方法，并基于几乎相同的基准数据集，[23] 得到了一个包含 5 000 个具有平坦计数的正则表达式的样本，其中计数器界限之和大于 20。[23] 的算法无法覆盖这 5 000 个正则表达式中的 571 个，占所检查的具有平坦计数的正则表达式集合的 11 %（相比之下，在非同步计数的情况下为 0.4%，在未进行字母标记的计数的情况下为 0.8%，后两者均在规模稍大的正则表达式集合上测得）。不过，我们的 5 751 个正则表达式与 [23], 的 5 000 个具有平坦计数的正则表达式并不完全相同。差异在较小程度上是由基础数据库的不同造成的（[23] 使用了约 18 个额外的专有正则表达式，并排除了 26 个计数器界限大于 1 000 的正则表达式），而在较大程度上则是由解析器之间的细微差异造成的。

8.2 同步计数匹配的实用效率

DCSA^a(CA(R)) 模拟的大小和最坏情况时间相对于 CA(R) 的状态数仍呈指数级增长（即分别为 $O(2^{2^m}m)$ 和 $O(2^{2^m}mn)$，其中 $m = \sharp R$ 等于 CA(R) 的状态数，参见定理 3）。潜在的问题在于，该算法最多可能生成 2^m 个计数器，这可能会对我们的匹配算法的实用性构成威胁。

[11] 我们未能解析 38558 个正则表达式，因为它们的语法有误或包含我们不支持的某些高级特性。12 926 个正则表达式包含嵌套计数，25297 个正则表达式包含较小的上界。

首先应注意，指数中的 m 可以从整个正则表达式的大小缩减为计数子表达式的大小，而后者通常非常小。其次，尽管高效实现超出了本文的范围，我们将其留作未来的工作，但我们还是为 CA 到 CSA 算法的实用性提供了一些间接论证。[13]

通过寄存器分配的标准技术 [41], ，可以大幅减少计数器的数量以及非恒等计数器赋值的数量。事实上，只需消除不必要的计数器重命名，并尽可能重复使用同一名称，我们的算法在处理 [23]所处理的正则表达式时，就会生成与 [23] 的 CSA 同构的 CSA。已有工作 [23] 表明，高效模拟这些 CSA 是可行的，并且在计数密集型示例中，相较于最佳匹配器，这种方法能带来显著改进。

根据我们在实际示例中手动模拟该算法的经验，对于 [23] 无法处理的情况，其表现并无太大不同，而且 CSA 中计数器的数量并没有明显的激增趋势。

9 结论

我们扩展了 [23] 的正则表达式匹配算法，并证明扩展后的版本能够对所谓的同步正则表达式实现快速模式匹配。同步正则表达式是我们新近引入的一类正则表达式。与此前所有已知可实现快速匹配的正则表达式类别相比，同步正则表达式的范围有显著扩展，并且涵盖了实际中出现的大多数正则表达式（根据我们的实证研究）。

未来，我们计划研究将本文提出的技术扩展到具有嵌套计数（非扁平）的正则表达式。这可能需要一种更复杂的偏移列表数据结构替代方案，用于存储数字关系的集合。另一个有趣的问题是，正则表达式能否以及何时可以改写为同步形式，以及这样做的代价是多少。

致谢

本工作得到了捷克教育、青年与体育部 ERC.CZ 计划项目 LL1908、捷克科学基金会项目 23- 06506S，以及布尔诺理工大学信息技术学院内部项目 FIT-S-23-8151 的资助。

参考文献

1. Sperberg-McQueen, M.：关于带计数器的有限状态自动机的笔记。
https://www.w3.org/XML/2004/05/msm-cfa.html 访问日期：2018-08-08。2. contributors, W.：正则表达式——维基百科（2019）

[13]一个能够在真实场景中的正则表达式上运行的高性能匹配器，需要完善的基础设施、针对共享寄存器进行优化的数据结构，以及理想情况下能够即时执行 CA 到 CSA 的确定化版本（类似于在线 DFA 模拟）。

3. Davis, J.C.：重新思考正则表达式引擎以应对正则表达式拒绝服务攻击。载于：ESEC/FSE'19，ACM（2019）1256–1258 4. Wang, P., Stolee, K.T.：真实场景中的正则表达式测试效果如何？载于 Leavens, G.T., Garcia, A., Pasareanu, C.S. 编：2018年ACM欧洲软件工程会议与软件工程基础研讨会联合会议论文集，ESEC/SIGSOFT FSE 2018，布埃纳维斯塔湖，佛罗里达州，美国，2018年11月4日至9日，678 5 2018 668ACM（）–。Davis, J.C., Coghlan, C.A., Servant, F., Lee, D.：正则表达式拒绝服务攻击（redos）在实践中的影响：一项生态系统规模的实证研究。载于 Leavens, G.T., Garcia, A., Pasareanu, C.S. 编：2018年ACM欧洲软件工程会议与软件工程基础研讨会联合会议论文集，ESEC/SIGSOFT FSE 2018，布埃纳维斯塔湖，佛罗里达州，美国，2018年11月4日至9日，ACM（2018 256 6 246）–。Chapman, C., Stolee, K.T.：探索 Python 中正则表达式的使用及其上下文。载于 Zeller, A., Roychoudhury, A. 编：第25届国际软件测试与分析研讨会论文集，ISSTA 2016，萨尔茨堡，德国，2016年7月18日至20日，ACM（ 2016 282 293 ）–。 Wubbeling, M.：正则表达式安全。ADMIN 55（2020）8. Graham-Cumming, J.：Cloudflare 于2019年7月2日发生服务中断的详情。https://blog.cloudflare.com/details-of-the-cloudflare-outage-on-july-2-2019/（2019）9. Exchange, S.：服务中断事后分析。http://stackstatus.net/post/147710624694/outage-postmortem-july-20-20（2016）10. Baldwin, A.：影响 Express.js 的正则表达式拒绝服务攻击。 https://medium.com/node-security/regular-expression-denial-of-service-affecting-express-js-9c397c164c43（ 2016）11. Davis, J.C., Coghlan, C.A., Servant, F., Lee, D.：正则表达式拒绝服务攻击（ReDoS）在实践中的影响：一项生态系统规模的实证研究。载于：ESEC/FSE'18，ACM（2018 256 12 246）–。Davis, J.C., Servant, F., Lee, D.：使用选择性记忆化抵御正则表达式拒绝服务攻击（ReDoS）。载于：第42届IEEE安全与隐私研讨会，SP 2021，旧金山，加利福尼亚州，美国， 24 27 2021， 2021 1 17 - May IEEE（）–。Turoˇnova, L., Holık, L., Lengal, O., Veanes, M., Vojnar, T.：正则表达式中的计数有害（2022）14. Thompson, K.：编程技术：正则表达式搜索算法。Commun. ACM 11 422 15 6 1968 419（） （）–。Wang, X., Hong, Y., Chang, H., Park, K., Langdale, G., Hu, J., Zhu, H.：Hyperscan：面向现代中央处理器的快速多模式正则表达式匹配器。载于：第16届USENIX网络系统设计与实现研讨会（NSDI 19），波士顿，MA，USENIX协会（2019年2月 648 16 631 2）–。谷歌：RE。 https://github.com/google/re2 17. Haertel, M. 等：GNU grep。https://www.gnu.org/software/grep/ 18. Saarikivi, O., Veanes, M., Wan, T., Xu, E.：符号正则表达式匹配器。载于 Vojnar, T., Zhang, L. 编：TACAS'2019 378 19 11427 2019 372 1 5 4 2021。LNCS 卷，施普林格（）–。docs.rs：regex - rust。https://docs.rs/regex/。./regex/（）20. Antimirov, V.：正则表达式的偏导数与有限自动机构造。理论计算机科学 155 319 21 2 1996 291（） （）–。 Glushkov, V.M.：自动机抽象理论。Russian Math. Surveys **16** 1961 1 53 （）–。Hromkoviˇc, J., Seibert, S., Wilke, T.：将正则表达式转换为小型无 ε 的非确定性有限自动机。载于 Reischuk, R., Morvan, M. 编：STACS 97，柏林，海德堡，施普林格·柏林·海德堡（1997）55–66

23. Turoˇnov´a, L.、Hol´ık, L.、Leng´al, O.、Saarikivi, O.、Veanes, M.、Vojnar, T.：带计数集合自动机的正则表达式匹配。《ACM程序设计语言学报》 **4** 2020 218 1 218 30（ 24 ¨
OOPSLA）（ ）：–：。Bjorklund, H.、Martens, W.、Timm, T.：简洁正则表达式的高效增量求值。载于：CIKM'15。ACM（2015）25. Gelade, W.、Gyssens, M.、Martens, W.：带计数的正则表达式：弱确定性与强确定性。《SIAM计算学报》41(1)（2012）160–190。MFCS'09论文的扩展版本。26. Hovland, D.：带数值约束的正则表达式与带计数器自动机。载于：ICTAC。LNCS 5684 2009 231 245 卷。施普林格（ ）–。Kilpelainen, P.、Tuhkanen, R.：带数值出现指示符的正则表达式的单一无歧义性。《信息与计算》205 916 28 6 2007 890（ ）（ ）–。Smith, R.、Estan, C.、Jha, S.：XFA：使用扩展自动机实现更快速的特征匹配。载于：IEEE安全与隐私研讨会，IEEE（ 2008）29. Hol´ık, L.、Leng´al, O.、Saarikivi, O.、Turoˇnov´a, L.、Veanes, M.、Vojnar, T.：通过球面构造对计数自动机进行简洁确定化。载于：APLAS'19论文集。LNCS 11893 2019 468 489 卷。施普林格（ ）–。Kilpelainen, P.、Tuhkanen, R.：带数值出现指示符的正则表达式——初步结果。载于：SPLST'03，库奥皮奥大学，计算机科学系（2003）163–173 31. Smith, R.、Estan, C.、Jha, S.、Siahaan, I.：使用扩展有限自动机（XFA）进行快速特征匹配。载于：ICISS'08 172 32 5352 2008 158。LNCS卷，施普林格（ ）–。Gelade, W.、Gyssens, M.、Martens, W.：带计数的正则表达式：弱确定性与强确定性。载于：2009年计算机科学数学基础会议，柏林、海德堡，施普林格·柏林·海德堡（2009 381 33 369）–。Hovland, D.：带无序连接和数值约束的正则表达式的成员判定问题。载于：语言与自动机理论及应用，柏林、海德堡，施普林格·柏林·海德堡（2012 324 34 313）–。Davis, J.C.、Michael IV, L.G.、Coghlan, C.A.、Servant, F.、Lee, D.：为什么正则表达式不是通用语言？关于正则表达式重复使用与可移植性的实证研究。载于：ESEC/FSE'19，1258 35 2019 1256ACM（ ）–。M. Roesch等：Snort：网络入侵检测与防御系统。http://www.snort.org 36. Robin Sommer等：Bro网络安全监控器 http://www.bro.org。37. Sagan团队：Sagan日志分析引擎 https://quadrantsec.com/sagan_日志_分析_引擎/。38. Yang, L.、Karim, R.、Ganapathy, V.、Smith, R.：使用有序二元决策图改进基于NFA的特征匹配。载于：入侵检测最新进展，施普林格·柏林·海德堡（ 2010 58 78）–。ˇCeˇska, M.、Havlena, V.、Holík, L.、Leng´al, O.、Vojnar, T.：用于高速网络入侵检测的有限自动机近似化简。载于：TACAS'18论文集。LNCS第10806卷，施普林格（2018）40. RegExLib.com：互联网最早的正则表达式库。http://regexlib.com/ 41. Aho, A.V.、Lam, M.S.、Sethi, R.、Ullman, J.D.：《编译原理、技术与工具》（第2版）。Addison Wesley（2006年8月）

定理 1 的证明

定理 1。DCSA(A)具有确定性， $|DCSA(A)| \in O(2^{|A|})$，并且如果A是笛卡尔的，则 $L(A) = L(DCSA(A))$。

证明。设 $A = (X,Q,\Delta,I,F)$。DCSA(A) 是确定性的，因为它只有一个初始配置，并且从同一状态出发的转移的守卫条件是最小项。

$|DCSA(A)| \in O(2^{|Q|} + 2^{|Delta|}.|Delta|) = O(|A|)$ 的大小，其中 $2^{|Q|}$ 是状态数， $2^{|Delta|}.|Delta|$ 是转移数乘以它们的大小。

由于 $L(A) = L(Conf(A)) = L(DA(Conf(A)))$ 且 $L(DCSA(A)) = L(Conf(DCSA(A)))$，我们只需证明 $L(DA(Conf(A))) = L(Conf(DCSA(A)))$。为此，我们证明 DA(Conf(A)) 与 Conf(DCSA(A)) 同构。

我们证明 enc 是给出该同构的双射。由于 DA(Conf(A)) 只有笛卡尔状态，enc 是单射的。为了证明 enc 是给出该同构的双射，我们需要证明

(1) 由于 I 是确定性自动机(Conf(A)) 的初始状态，因此 enc(I) 是 Conf(DCSA(A)) 的初始状态；(2) 对于确定性自动机(Conf(A)) 的每个状态 α，enc(α) 都是 Conf(DCSA(A)) 的一个状态；(3) 对于 Conf(DCSA(A)) 的每个状态 β，dec(β) 都是确定性自动机(Conf(A)) 的一个状态，即 enc 是满射的；(4) 如果 $\alpha \xrightarrow{\{a\}} \alpha'$ 是确定性自动机(Conf(A)) 的一个转移，那么 Conf(DCSA(A)) 中存在一个转移 $enc(\alpha) \xrightarrow{\{a\}} enc(\alpha')$；(5) 如果 $\beta \xrightarrow{\{a\}} \beta'$ 是 Conf(DCSA(A)) 的一个转移，那么确定性自动机(Conf(A)) 中存在一个转移 $dec(\beta) \xrightarrow{\{a\}} dec(\beta')$；(6) α 是确定性自动机(Conf(A)) 的一个终态，当且仅当 enc(α) 是 Conf(DCSA(A)) 的一个终态。

(1) 由构造可知显然成立。

我们现在假设该论证成立：对于 DA(Conf(A)) 的每个转移 $\alpha \xrightarrow{\{a\}} \alpha'$，如果 enc(α) 是 Conf(DCSA(A)) 的一个状态，那么 Conf(DCSA(A)) 中存在一个转移 $enc(\alpha) \xrightarrow{\{a\}} enc(\alpha')$。如果这一点成立，那么我们可以很容易地证明 (2) 成立，因为如果 α 是 DA(Conf(A)) 的一个状态，那么 DA(Conf(A)) 中存在一条路径 $\alpha_0 \xrightarrow{\{a_1\}} \alpha_1 \xrightarrow{\{a_2\}} \ldots \xrightarrow{\{a_n\}} \alpha$，通过对该路径长度进行归纳法，我们可以证明 $enc(\alpha_0) \xrightarrow{\{a_1\}} enc(\alpha_1) \xrightarrow{\{a_2\}} \ldots \xrightarrow{\{a_n\}} enc(\alpha)$ 是 Conf(DCSA(A)) 中的一条路径，这意味着 enc(α) 是 Conf(DCSA(A)) 的一个状态。由该论证和 (2) 可立即推出 (4) 也成立。由

– (4)，– Conf(DCSA(A)) 是确定性的这一事实，– DA(Conf(A)) 是确定性且全的（即对于每个状态 α 和符号 a，恰好存在一个转移 $\alpha \xrightarrow{\{a\}} \alpha'$）这一事实，– 以及对于 Conf(DCSA(A)) 的每个状态 β，$enc(dec(\beta)) = \beta$ 这一事实

很容易得出（同样通过对到达 β 的路径长度进行归纳法）(3) 和 (5) 也成立。

现在我们只需证明论证成立。因此，令 $\alpha \xrightarrow{\{a\}} \alpha'$ 为 DA(Conf(A)) 的一个转移。我们需要证明：如果 enc(α) 是 Conf(DCSA(A)) 的一个状态，那么在 Conf(DCSA(A)) 中存在一个从 enc(α) $\xrightarrow{\{a\}}$ 到 enc(α′) 的转移。令 $enc(\alpha) = (R, s)$，且 $enc(\alpha') = (R', s')$。令 $\Delta^A_{R,a} = \{ q \xrightarrow{\{a,\varphi,u\}} p \in \Delta \mid q \in R \}$ 表示 RA 中的转移集合

A 从 R 中的某个状态开始。转移 $\alpha \xrightarrow{\{a\}} \alpha'$ 由 Conf(A) 的组成转移 $(q\ m) \xrightarrow{\{a\}} (q', m')$ 构成，其中 $(q\ m) \in \alpha$，且 $(q', m') \in \alpha'$。对于每个这样的转移 $(q\ m) \xrightarrow{\{a\}} (q', m')$，A 中必须存在某个转移 $q \xrightarrow{\{a,\varphi,u\}} q'$，其中 $m \models \varphi$，并且对于每个 $x \in X$，都有 $m'(x) = u(x)(m)$。根据构造假设，我们知道 φ 是若干一元谓词 $p[x]$ 的合取项，这意味着 $m \models p[x]$。由此我们知道，对于每个这样的 $p[x]$，，$s \models p[x_q]$，因为 $m(x) \in s(x_q)$，并且如果在 $s(x_q)$ 中存在满足 $p[x_q]$ 的值，则 $s \models p[x_q]$ 成立。这也意味着 $s \models \mathrm{index}_q(\varphi)$。

因此，令 $\Delta^{\alpha}_{R,a} = \{ q \xrightarrow{\{a,\varphi,u\}} q' \in \Delta^{A}_{R,a} \mid m \models \varphi$ 对于某个 $(q\ m) \in \alpha \}$ 为 A 中创建 Conf(A) 中某些构成 $\alpha \xrightarrow{\{a\}} \alpha'$ 的转移的转移集合。注意，$\Delta^{\alpha}_{R,a}$ 是 $\Delta^{A}_{R,a}$ 的子集，并且对于 $\Delta^{A}_{R,a} \setminus \Delta^{\alpha}_{R,a}$ 中的每个转移 $q \xrightarrow{\{a,\varphi,u\}} q'$，不存在满足 $m \models \varphi$ 的 $(q\ m) \in \alpha$。利用这一点，我们可以证明 $s \not\models \mathrm{index}_q(\varphi)$，即 $s \models \neg\mathrm{index}_q(\varphi)$。假设这不成立，即 $s \models \mathrm{index}_q(\varphi)$。那么，对于 φ 中的每个原子谓词 $p[x]$，$s(x_q)$ 中必定存在某个满足 $p[x_q]$ 的值。对于每个 $x \in X$，φ 中至多有一个 $p[x]$，因此我们可以取 m，使得 $m(x)$ 是 $s(x_q)$ 中满足 $p[x]$ 的值（如果不存在 $p[x]$，，则从 $s(x_q)$ 中任取一个值）。由于 α 是笛卡尔的，α 中必定存在一个配置 $(q\ m)$。于是我们有 $m \models \varphi$，这与 $q \xrightarrow{\{a,\varphi,u\}} q' \notin \Delta^{\alpha}_{R,a}$ 矛盾。

根据 $\mathrm{enc}(\alpha) = (R\ s)$ 是 Conf(DCSA(A)) 的状态这一假设，我们知道 R 必须是 DCSA(A) 的一个状态，这意味着我们会创建一些标记为 a 的、从 R 出发的转移。我们有 $\mathrm{enc}(\alpha') = (R', s')$，现在证明算法会创建一个转移 $R \xrightarrow{\{a,\psi,u'\}} R'$，使得 (R', s') 是 $(R\ s)$ 在该转移下的像，即 $\mathrm{enc}(\alpha) \xrightarrow{\{a\}} \mathrm{enc}(\alpha')$ 是 Conf(DCSA(A)) 的转移。令 $\Delta_{R,a}$ 为第 4 节中定义的组成 a-转移的集合，即 $\Delta^{A}_{R,a}$ 中转移的集合版本。令 ψ 为一个合取项

$$\bigwedge_{q \xrightarrow{\{a,\varphi,u\}} q' \in \Delta^{\alpha}_{R,a}} \mathrm{index}_q(\varphi) \quad \text{and} \quad \bigwedge_{q \xrightarrow{\{a,\varphi,u\}} q' \in \Delta^{A}_{R,a} \setminus \Delta^{\alpha}_{R,a}} \neg\mathrm{index}_q(\varphi).$$

正如我们所展示的，对于第一个大合取项中的每个合取项，有 $s \models \mathrm{index}_q(\varphi)$，而对于第二个大合取项中的每个合取项，有 $s \models \neg\mathrm{index}_q(\varphi)$，这意味着 $s \models \psi$。此外，由于 $\Delta_{R,a}$ 中每个转移的守卫条件出现在 ψ 中，
$\psi \in$ 最小项$(\{ \varphi \mid q \xrightarrow{\{a,\varphi,u\}} p \in \Delta_{R,a} \})$。于是，为最小项 ψ 构造的转移 $R \xrightarrow{\{a,\psi,u'\}} R'$ 正是我们所需的转移，下面对此进行说明。设$\Delta_{R,a,\psi} \subseteq \Delta_{R,a}$ 是守卫条件与最小项 ψ 兼容的转移，即它们是来自$\Delta^{\alpha}_{R,a}$的转移的集合版本。显然，$R' = \{ p \mid q \xrightarrow{\{a,\varphi,u\}} p \in \Delta_{R,a,\psi} \} = \{ p \mid q \xrightarrow{\{a,\varphi,u\}} p \in \Delta^{\alpha}_{R,a} \}$ 是 $\mathrm{enc}(\alpha')$ 的正确状态集合。剩下唯一需要证明的是，对 s 应用更新 $u'(x_q)$ 后，对于每个 $x \in X, q \in Q$，所得结果是 $s'(x_q)$ 的正确值。

我们固定 $x \in X, q' \in Q$，并令 N 为对 s 应用更新 $u'(x_{q'})$ 所得到的值的集合，即 $N = \bigcup \{ u(x_{q'})(s) \mid q \xrightarrow{\{a,\varphi,u\}} q' \in \Delta_{R,a,\psi} \}$。我们需要证明 $N = s'(x_{q'})$。

首先，我们证明，如果 $n \in s'(x_{q'})$，那么 $n \in N$。由于 $n \in s'(x_{q'})$ 且 $\mathrm{enc}(\alpha') = (R', s')$，必定存在某个 $(q', m') \in \alpha'$，使得 $m'(x) = n$。为此，必定存在某个构成转移 $(q\ m) \xrightarrow{\{a\}} (q', m') \in$ Conf(A)，它属于某个转移 $\alpha \xrightarrow{\{a\}} \alpha'$，对应某个 $(q\ m) \in \alpha$。这意味着存在某个转移 $q \xrightarrow{\{a,\varphi,u\}} q' \in \Delta^{\alpha}_{R,\ a}$，其中$m \models \varphi$ 且 $m'(x) = u(x)(m)$。由此我们知道，项 t 是这样构造的：

从索引$^{u(x)}$(q) 出发，添加由 φ生成的过滤器；所得项必定是用于计算 N 的项之一。我们还知道，对于每个 y ∈X，值 m(y) ∈ s(y$_q$)，并且由于m ⊨ φ，这些值不会被 t 中的过滤器过滤掉。因此，n 必定属于 N。

现在令 n ∈ N，并证明 n ∈ s′(x$_{q'}$)。由 n ∈N 可知，必定存在某个 $\delta = q \xrightarrow{\{a,\varphi,\mu\}} q' \in \Delta_{R,\ a,\ \psi}$ 使得 n ∈ u(x$_{q'}$)(s)。我们可以构造 m，使得 m 的取值恰好是应用 u(x$_{q'}$) 后得到 n 的那些值。此外，在 $\Delta^{\alpha}_{R,\ a}$ 中存在一个转移 δ′，由此构造出 δ；由于 α是笛卡尔的，(q m) ∈α（并且根据过滤器的工作方式，它在 δ′ 中是允许的）。这意味着 n ∈ s′(c$_{q'}$)。

最后，我们需要证明 (6) 成立。令 α 为 DA(Conf(A)) 的一个状态，且 enc(α) = (R s)。如果 α是终态，那么必定有 (q m) ∈ α，其中 m ⊨F(q)。根据与证明守卫条件时类似的论证，s ⊨ index$_q$(φ)，这意味着 s ⊨F′(R) = ⋁q∈R index$_q$(F(q))，即 enc(α) 是 Conf(DCSA(A)) 的一个终态。反过来，如果 enc(α) 是终态，那么 s ⊨ ⋁q∈R index$_q$ (F(q))，这意味着必定存在某个 q ∈ R，使得 s ⊨ index$_q$(F(q))。现在我们可以构造 m，其中对于每个 x ∈ X，取值 m(x) 是 s(x$_q$) 的某个值，该值满足来自 index$_q$(F(q)) 的原子谓词 p[x$_q$]（这样的原子谓词至多有一个；若不存在，则从 s(x$_q$) 中任取一个值）。于是 m ⊨ F(q)，并且由于 α 是笛卡尔的，必定有 (q m) ∈α，因此 α 是一个终态。

B 计数正则表达式到计数自动机

我们展示如何为计数正则表达式 R 构造计数自动机 CA(R) = (X,Q Δ,I,F)，并给出它的一些有用性质。我们的构造基于 Glushkov 构造 [21] 的推广，其中我们在 [32] 的基础上进行了略微修改。

Glushkov 构造创建的自动机，其状态是正则表达式 R 中符号的出现位置以及一个特殊的初始状态。因此，我们定义标记正则表达式 R，使其基于字母表 Σ × ℕ，并且该字母表中的任何符号都不会在 R 中出现两次。例如，对于正则表达式 abc(bc){3}，其标记版本可以是 $a_1b_1c_1(b_2c_2)\{3\}$。此时，自动机的状态将是 a_1, 、b_1, 、c_1, 、b_2, 和 c_2，以及特殊的初始状态。我们还记作 a_i ∈R，表示 a_i 在 R 中出现。注意，R 中这些符号的数量为 ♯R。

对于 R 中的每个计数子表达式，都会创建 CA(R) 的计数器 X。为简化构造，我们假设：如果 R 中存在某个子表达式 S{m,n}，使得 ε ∈ L(S{m,n})，那么 m = 0（将正则表达式 R 转换为这种形式可以在线性时间内完成 [32]）。我们还假设 R 不包含任何子表达式 S*，而是用 S{0,∞} 来表示它们。但是，CA(R) 不会为这些计数子表达式设置计数器，因为这类计数器唯一的守卫条件是 c ≥ 0 和 c < ∞，它们显然始终为真。因此，计数器集合 X 包含 R 中除形式为 S{0,∞} 的计数子表达式之外的所有计数子表达式。此外，对于 R 的每个子表达式 S，我们定义集合 counters(S)，其中包含 S 的所有计数子表达式 T ∈ X，且满足 T ≠ S。我们还定义 lower(S{m,n}) = m，以及 upper(S{m, n}) = n。

现在，我们定义集合 first(R) 和 last(R)，它们分别由 R 所表示的某个单词中的首符号和尾符号组成。它们的定义是归纳式的：

- first(ε) = last(ε) = ∅ and first(a_i) = last(a_i) = {a_i}

– $\text{first}(\overline{R_1 | R_2}) = \text{first}(R_1) \cup \text{first}(R_2)$ 和 $\text{last}(\overline{R_1 | R_2}) = \text{last}(R_1) \cup \text{last}(R_2)$ – 若 $\varepsilon \in L(R_1)$，则 $\text{first}(R_1 R_2) = \text{first}(R_1) \cup \text{first}(R_2)$，否则 $\text{first}(R_1 R_2) = \text{first}(R_1)$ – 若 $\varepsilon \in L(R_2)$，则 $\text{last}(R_1 R_2) = \text{last}(R_1) \cup \text{last}(R_2)$，否则 $\text{last}(R_1 R_2) = \text{last}(R_2)$ – $\text{first}(R\{m, n\}) = \text{first}(R)$ 和 $\text{last}(R\{m, n\}) = \text{last}(R)$

利用这些集合，我们定义集合 follow(R)，其中包含三元组 (a_i, b_j, x)，其中 $a、b \in \Sigma$、$i、j \in \mathbb{N}$，而 x 要么为 null，要么是 R 的一个计数子表达式。该集合用于在计数自动机 CA(R) 的状态 a_i 和 b_j 之间创建转移，其中 x 提供有关应递增哪些计数器的信息。形式化地说，集合 follow(R) 是若干集合的并集

$$\left\{ (a_i, b_j, \text{null}) \;\middle|\; \overline{S_1 S_2} \text{ is a subexpression of } \overline{R},\ a_i \in \text{last}(\overline{S_1}), \text{ and } b_j \in \text{first}(\overline{S_2}) \right\}$$

and

$$\left\{ (a_i, b_j, \overline{S}\{m,n\}) \;\middle|\; \overline{S}\{m,n\} \text{ is a subexpression of } \overline{R},\ a_i \in \text{last}(\overline{S}), \text{ and } b_j \in \text{first}(\overline{S}) \right\}.$$

现在可以定义计数自动机 $CA(R) = (X, Q, \Delta, I, F)$。状态集合 Q 包含初始状态 q_0 以及 R 中出现的所有符号 a_i。集合 Δ 包含以下转移：

– 对每个 $a_i \in \text{first}(R)$，存在转移 $q_0 \xrightarrow{\{a, true, u\}} a_i \in \Delta$，其中对于每个 $S \in X$

$$u(\overline{S}) = \begin{cases} 1 & \text{if } a_i \in \overline{S} \\ 0 & \text{else} \end{cases}$$

– 对于每个 $(a_i, b_j, \text{null}) \in \text{follow}$ 集，均存在一个转移 $a_i \xrightarrow{\{b, \varphi, u\}} b_j \in \Delta$，其中

$$\varphi = \bigwedge_{\overline{S} \in X,\, a_i \in \text{last}(\overline{S})} \overline{S} \geq \text{lower}(\overline{S})$$

并且对于每个 $S \in X$

$$u(\overline{S}) = \begin{cases} 1 & b_j \in \text{first}(\overline{S}) \\ \overline{S} & a_i, b_j \in \overline{S} \text{ and } b_j \notin \text{first}(\overline{S}) \\ 0 & \text{otherwise} \end{cases}$$

– 对于每个 $(a_i, b_j, S) \in$ follow 集，均存在一个转移 $a_i \xrightarrow{\{b, \varphi, u\}} b_j \in \Delta$，其中以下任一条件成立

$$\varphi = \overline{S} < \text{upper}(\overline{S}) \wedge \bigwedge_{\overline{T} \in \text{counters}(\overline{S}),\, a_i \in \text{last}(\overline{T})} \overline{T} \geq \text{lower}(\overline{T})$$

对于 $S \in X$ 的情况，或

$$\varphi = \bigwedge_{\overline{T} \in \text{counters}(\overline{S}),\, a_i \in \text{last}(\overline{T})} \overline{T} \geq \text{lower}(\overline{T})$$

否则（即 $\mathrm{lower(S)} = 0$ and $\mathrm{upper(S)} = \infty$ 的情况），并且对于每个 $\mathrm{T} \in \overline{\mathrm{X}}$

$$u(\overline{T}) = \begin{cases} \overline{S}+1 & \overline{T} = \overline{S} \\ 1 & \overline{T} \in \text{counters}(\overline{S}) \text{ and } b_j \in \text{first}(\overline{T}) \\ \overline{T} & \overline{T} \notin \text{counters}(\overline{S}), \overline{T} \neq \overline{S} \text{ and } a_i, b_j \in \overline{T} \\ 0 & \text{otherwise} \end{cases}$$

初始配置的集合是单元素集合 $\{(\mathrm{q0}, \{\mathrm{S} \mapsto 0 \mid \mathrm{S} \in \mathrm{X}\})\}$。对于每个状态 q，终态条件 F 定义为

$$F(q) = \begin{cases} true & q = q_0 \text{ and } \varepsilon \in L(R) \\ \bigwedge_{\overline{S} \in X, a_i \in \text{last}(\overline{S})} \overline{S} \geq \text{lower}(\overline{S}) & \text{if } q \in \text{last}(\overline{R}) \\ false & \text{otherwise} \end{cases}$$

引理 1. 设 R 是一个包含 $\sharp\mathrm{R} = \mathrm{m}$ 个符号出现的正则表达式。那么 $\mathrm{L(R)} = \mathrm{L(CA(R))}$，且 CA(R) 有 $\mathrm{m}+1$ 个状态和至多 $\mathrm{cnt}_R.\mathrm{m}^2$ 个转移。若 R 是扁平正则表达式，则它至多有 m^2 个转移。

证明。根据 R 的结构进行归纳。

B.1 扁平正则表达式的 CA(R) 性质

之所以使用 Glushkov 构造而不是 [23] 中的 Antimirov 构造，是因为我们的 CA(R) 具有界限明确的计数循环，即对应于 $\underline{\mathrm{R}}$ 的计数子表达式的部分，并且计数循环的进入转移和退出转移具有特殊条件。

我们来总结一下扁平正则表达式 R 中 $\mathrm{x} \in \mathrm{X}$ 的计数循环的性质。它由状态集合 $\mathrm{Q}_x \subseteq \mathrm{Q}$ 表征，其中 $\mathrm{Q}_x \cap \mathrm{Q}_y = \emptyset$ 对于 $\mathrm{x} \neq \mathrm{y}$ 是进入状态集合 $\mathrm{first(x)} \subseteq \mathrm{Q}_x$，而退出状态集合为 $\llcorner\mathrm{last(x)} \subseteq \mathrm{Q}_x$。由于 x 是某个计数子表达式 S{m, n}，集合 first(x) 和 last(x) 按照上一节中的定义，而集合 Q_x 包含所有 $\mathrm{a}_i \in \mathrm{x}$。CA(R) 中不与 Q_x 相接的转移将 x 赋值为 0，且其守卫条件中不包含 x。与 Q_x 相接的转移划分为 $\Delta_x = \Delta_x^{inner} \uplus \Delta_x^+ \uplus \Delta_x^1 \uplus \Delta_x^{in} \uplus \Delta_x^{out}$，其中 Δ_x^1 是可选的：

- Δ_x^{in}是从外部进入 Q_x 的唯一转移。Δ_x^{out}是从 Q_x 向外的唯一转移。- Δ_x^{inner}中的转移起始于 Q_x，并且终止于 Q_x。其守卫条件为 $\top$，更新为 $\mathrm{c} := \mathrm{x}$。它们构成从进入状态到 last(x) 中某个状态的路径。- Δ_x^+具有如下转移：$\mathrm{q}_o \xrightarrow{\{a, x < \max_x, x := x+1\}} \mathrm{q}_e$，$\mathrm{q}_o \in$last(x)，使得 first(x) 恰好是所有目标 q_e 的集合。Δ_x^+还确定了其余的转移集合。- Δ_x^1由 Δ_x^+ 得到：将守卫条件替换为 $\mathrm{x} \geq \min_x$，并将更新中的 $\mathrm{x}+1$替换为 1。- Δ_x^{in}是从外部进入 Q_x 的所有转移。它是若干状态 $\mathrm{q} \in \mathrm{Q} \setminus \mathrm{Q}_x$对应的集合 Δ_q 的并集。每个 Δ_q 都由 Δ_x^1得到：将源状态 q_x 替换为 q，并从守卫条件中移除 $\mathrm{x} < \max_x$。

– Δ_x^{out}是离开 Q_x 的所有转移。它们的形式为 $q_o \xrightarrow{\{a, x \geq \min_x x := 0\}} q$，$q_o \in \text{last}(x)$，$q \in Q \setminus Q_x$。这些转移可能与 Δ_y^{in}共享；在这种情况下，更新还包含 $y := 1$。

Q_x 中没有状态是初始状态，而来自 last(x) 的状态是 Q_x 中唯一可能具有可满足的终结条件的状态；也就是说，它可能是 $x \geq \min_x$（或 $\bot$）。Q_x 之外的状态，其终止条件中不包含 x。

下面总结 CA(R) 中路径的特征。对于 CA（或 CSA），在单词 $a_1 \cdots a_n$、$n \in \mathbb{N}$上从 r_0到 r_n的一条路径，是一个转移序列：$r_0 \xrightarrow{\{a_1, \varphi_1, u_1\}} r_1, r_1 \xrightarrow{\{a_2, \varphi_2, u_2\}} r_2, \ldots, r_{n-1} \xrightarrow{\{a_n, \varphi_n, u_n\}} r_n$。我们将 $q \in Q$ 在单词 v 上的 c-分支定义为从 q 出发、沿 v 行进的路径集合；这些路径一开始会将 1 赋给 c（通过 Δ_x^{in}或Δ_x^{1}），之后只会复制或递增 c（通过 Δ_x^{inner}或 Δ_x^{+}）。

引理 2. 设 π是 c-分支中的一条路径，它被来自 Δ_x^{+}的转移分割成 $n+1$个片段，其中 $n \in \mathbb{N}$（新片段从来自 Δ_x^{+}的转移开始）。那么，假设 $c = S\{m, n\}$，每个片段读取的单词都属于 L(S)，或者最后一个片段是某个单词的前缀。此外，π 将 c 递增 n 次，并且除第一个片段外，每个片段都从 last(c) 中的某个状态开始。

C 计算和模拟 DCSA^a(R) 的复杂度

设 R 是一个具有平坦计数的正则表达式，并设 A = CA(R) = (X, Q, Δ, I, F)。若 X 非空（下文均假设如此），那么对于 DCSA^a(A) 的状态 (R, Act)，状态集合 R 是多余的，因为可以从活动计数器集合 Act 中推导出它（每个 x_r、$r \in R$ 都必须具有某个值；也就是说，某个带有 r 或 r^+ 且位于 S 中的 x_S 必须处于活动状态）。于是，DCSA^a(A) 的每个状态都可以由 $X^a = X \times \mathcal{P}(Q \cup Q^+)$ 的某个子集唯一表示，即 DCSA^a(A) 有 $O(2^{|X|2^{|Q|}})$ 个状态。然而，由于 R 具有平坦计数，对于每个 $q \in Q$，至多存在一个 $x \in X$，使得 x_q 在 A 中可以具有非零值。因此，DCSA^a(A) 中至多有 $O(2^{2^{|Q|}})$ 个状态。

对于每个状态 (R, Act)，从该状态出发的转移最多有 $O(2^{|\Delta|}) = O(2^{|Q|^2})$ 个（$|\Delta|$ 对于具有平坦计数的正则表达式，最多有 $|Q|^2$ 个转移，见引理 1），因为我们为 DCSA(A) 的每个最小项创建一个转移，而这些最小项是来自 A 的转移的守卫条件的合取项。此外，每个转移的大小取决于其守卫条件的大小和更新函数的大小。DCSA(A) 中转移的守卫条件（即最小项）的大小为 O(|Q|)，因为对于每个带有至多一个非零计数器 $x \in X$ 的 $q \in Q$，其守卫条件都是谓词 $x_q < \max_x$ 和 $x_q \geq \min_x$ 的合取项，以及它们的否定。对于 DCSA^a(A)，我们将该守卫条件中的每个 x_q 替换为 x_S、$q \in S$ 的析取，因此 DCSA^a(A) 中守卫条件的大小为 $O(|Q|2^{|Q|})$。由于每个 x_S 只能出现在更新中的一个项内（我们不能进行计数器复制），其大小为 $O(|X|2^{|Q|})$；但是，使用与状态数相似的推理，可以将更新的大小缩减为 $O(2^{|Q|})$。因此，DCSA^a(A) 的大小为 $O(2^{2^{|Q|}} * 2^{|Q|^2} * |Q|2^{|Q|}) = O(|Q|2^{2^{|Q|}})$

引理 3. DCSA^a(A) 的大小为 $O(|Q|2^{2^{|Q|}})$。

由 DCSA^a(A) 的构造，以及上述通过 DCSA^a(CA(R)) 的配置对 DCSA(A) 的配置进行编码的描述可知，L(DCSA^a(CA(R))) = L(DCSA(CA(R)))。此外，对于具有平坦计数的 R，CA(R) 是笛卡尔的，因此我们得到：

引理 4. L(CA(R)) =L(DCSA^a(CA(R)))。

第 5 节给出了使用带有非复制计数器的自动机进行成员资格检查的最坏情况时间近似。现在，我们将这些近似归约为 DCSA^a(A)
为了找出从状态 (R,Act) 出发、在集合解释$\mathfrak{s}$ 和符号 a 下应采取的下一条 CSA 转移，我们必须遍历所有从 (R,Act) 开始且守卫条件为最小项的 a-转移。然而，我们需要测试每个最小项；由于我们知道原子谓词的测试是常数时间操作，因此找出下一条 CSA 转移可以在 $O(|Q|2^{2|Q|})$ 时间内完成，因为最多有 $O(2^{|Q|})$ 个最小项，且其大小至多为 $O(|Q|2^{|Q|})$（理由与 DCSA^a(A) 的大小类似）。

应用所选转移更新的步骤的摊销时间为 $O(|X^a|) =O(|X|2^{|Q|})$，然而，对于平坦计数，我们可以将其降至 $O(2^{|Q|})$。因此，对单词 w 进行完整匹配的成本为 $O(|w|(|Q|2^{2|Q|}+2^{|Q|})) = O(|w||Q|2^{2|Q|})$。

引理 5。使用 DCSA^a(A) 对单词 w 进行完整匹配的开销为 $O(|w||Q|2^{2|Q|})$。

D 定理 4 的证明

定理 4。给定一个具有平坦计数的正则表达式 R，第 6 节的算法当且仅当 R 具有同步计数时返回 DCSA^a(CA(R))。

该定理通过下面两个技术引理得以证明，它们刻画了经过 CA(R) 和 DCSA^a(CA(R)) 的路径之间的关键关系。这里使用附录 B.1 中的记号。

引理 6。设 CA(R) 中存在一条关于单词 w、从其初始状态到某个状态 q_i 的路径，并设 π_1、π_2 是从状态 q 出发、关于单词 v 的两条 x-扇出路径，其中 π_1 将 x 递增 $k+1$ 次并最终到达状态 r，而 π_2 将 x 递增 k 次并最终到达状态 p。那么，如果 DCSA^a(CA(R)) 存在，则 DCSA^a(CA(R)) 中存在一条关于 wv 的路径，该路径以一个活动计数器 x_S 结束，并满足 $r^+ \in S$ 且 $p \in S$。

证明。假设 DCSA^a(CA(R)) 存在。由于 DCSA^a(CA(R)) 是通过扩展经典子集构造创建的，因此，沿 w 的路径必须通向某个状态 (T, Act)，使得 $q \in T$。令 $v = a_1 \dots a_n$，$\pi_1[i]$ ($\pi_2[i]$) 为包含 π_1 (π_2) 的第一个 i 转移的路径，令 r_i (p_i) 为路径 $\pi_1[i]$ ($\pi_2[i]$) 结束时所处的状态（即 $r_i = r$）。此外，令 π 为在 DCSA^a(CA(R)) 中从 (T, Act) 沿 v 出发的路径，令 (T_i, Act_i) 为对每个 i，$1 \le i \le n$，$\pi[i]$, 的结束状态。现在我们证明，对于每个 i，$1 \le i \le n$，在 $\pi_1[i]$ 和 $\pi_2[i]$ 中发生的 x 的递增次数要么

– 相同，且存在 $x_{S_i} \in Act_i$，使得 $r_i, p_i \in S_i$；或者 – 相差 1。这里，如果 $\pi_1[i]$ 比 $\pi_2[i]$，多一次递增，那么存在 $x_{S_i} \in Act_i$，使得 $r_i^+, p_i \in S_i$（$\pi_2[i]$ 多一次递增时也类似）。

我们通过对 i 作归纳法证明这一点。

对于 $i = 1$，由于 π_1 和 π_2 处于 x-扇出中，它们从 q 到 r_1 和 p_1的第一次转移必须将 1 赋给 x。因此，在构造过程中计数器将被共享，也就是说，存在一个 x_{S_1}，我们在 π 和 $r_1, p_1 \in S_1$的第一次转移中将 1 赋给它。

现在假设该命题对 $1 \leq k < n$ 成立，并且我们希望证明它对 $k+1$ 也成立。首先，由于 $k+1 > 1$ 和 π_1 以及 π_2 处于 x-扇出中，我们知道 $\pi_1[k+1]$ 和 $\pi_2[k+1]$ 末尾的转移必须复制或递增 x。现在假设 $\pi_1[k]$ 中发生的 x 的递增次数比 $\pi_2[k]$ 中多一次（两者相同时的情形推理类似）。我们知道存在 $x_{S_k} \in \mathrm{Act}_k$，其中 r_k^+、$p_k \in S_k$。如果 $\pi_1[k+1]$ 和 $\pi_2[k+1]$ 末尾的转移对 x 执行相同的操作，那么在 $\pi[k+1]$ 末尾将对 x^{S_k} 执行相同的操作（可能带有过滤器），并且命题对 $k+1$ 成立（因为计数器仍以相同的增量延迟方式共享）。如果 $\pi_1[k+1]$ 末尾的转移只是复制 x，而 $\pi_2[k+1]$ 末尾的转移递增 x，那么 x^{S_k} 会在 $\pi[k+1]$ 末尾递增（即增量是同步的），存在 $x_{S_i} \in \mathrm{Act}_i$，其中 r_i、$p_i \in S_i$，并且 $\pi_1[k+1]$ 和 $\pi_2[k+1]$ 具有相同的递增次数。最后，$\pi_1[k+1]$ 末尾的转移递增 x，而 $\pi_2[k+1]$末尾的转移复制 x 的情形不可能发生。这是因为 π 末尾的转移必须对 x 执行两次递增，而这是不允许的；也就是说，$\mathrm{DCSA}^a(\mathrm{CA(R)})$ 将不存在，这与假设矛盾。

引理 7. 在构造 $\mathrm{DCSA}^a(\mathrm{CA(R)})$ 的过程中，设存在一条路径 π 在单词 w 上，该路径以活动计数器 x_S 以及某个 $r^+ \in S$ 结束。那么存在单词 u、v、$w = uv$，使得 CA(R) 存在一条经过 u 的路径到达某个状态 q，并且从状态 q 出发，存在两条 π_1和 π_2 关于 x-扇出的路径，均经过单词 v，其中一条使 x 递增 $k+1$次并最终到达状态 r，另一条使 x 递增 k 次，其中 $k \in \mathbb{N}$。

证明。设 $w = a_1 \ldots a_n$，且 $(T_{i-1}, \mathrm{Act}_{i-1}) \xrightarrow{\{a_i, \varphi_i, u_i\}} (T_i, \mathrm{Act}_i)$，$1 \leq i \leq n$，为 π 的转移。设 x^{S_l}、$x^{S_{l+1}}$、...、x^{S_n}（其中 $1 \leq l \leq n$）为一个计数器序列，其中 $S_n = S$；对于每个 $l < j \leq n$，我们有：项 $u_j(x^{S_j})$ 包含 $x^{S_{j-1}}$，而 $u(x^{S_l})$ 不包含任何计数器。直观地说，这是同一个计数器的一系列名称：它首先将 0 或 1 映射到 x^{S_l}（我们将证明实际上只能是 1），随后该计数器的值被复制或递增到下一个计数器，直到最后得到 x_S。此外，$l \neq n$，否则 $u_n(x_S)$ 将不包含计数器，也就没有理由使 r^+ 属于 S。还可以得到，每个 $S_j \subseteq Q_x$，因为每个 $u_j(x^{S_j})$ 都包含 $x^{S_{j-1}}$；根据该构造的工作方式，这意味着从 S_{j-1} 中的每个状态到 S_j 中的某个状态都有一个转移，其中 x 被复制或递增，即这些转移来自 Δ_x^{inner}或 Δ_x^+（这也意味着 $S_l \subseteq Q_x$）。我们还可以证明 $S_l = \mathrm{first(x)}$，因为 $u(x^{S_l})$ 不包含任何计数器，即 $u(x^{S_l}) = 1$，而该更新由 Δ_x^{in}或 Δ_x^1 的某些转移构造而成。现在取 $u = a_1 \ldots a_{l-1}$、$v = a_l \ldots a_n$，且状态 q 是来自 Δ_x^{in}或 Δ_x^1 的转移的某个初始状态。显然，CA(R) 存在一条经过 u 的路径到达 q；我们可以通过归纳法证明，对于每个 x^{S_j}，都有：

– 如果 $q_1, q_2 \in S_j$，那么从状态 q 出发，存在沿单词 $a_l \ldots a_j$的路径 $\pi_1$to q_1 和 $\pi_2$to q_2，其 x- 扇出为：它们使 x 递增相同的次数；并且 – 如果 $q_1^+, q_2 \in S_j$，那么从状态 q 出发，存在沿单词 $a_l \ldots a_j$的路径 $\pi_1$to q_1 和 $\pi_2$to q_2，其 x-扇出为：π_1 使 x 递增的次数比 π_2 多一次。

对此的证明与引理 6 的证明非常相似。最后，S 必须包含某个未标记状态 p，否则就不可能有 r^+ 属于 S（S 不可能

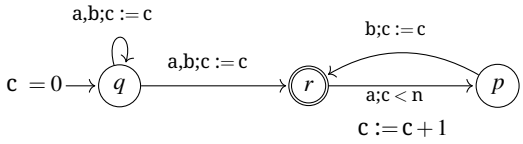
只包含已标记状态，因为否则递增就会在 $u_n(c_S)$) 中同步。）

证明（定理 4 的证明）。
(⇒) 通过反证法。假设 CA(R) 存在，但 R 不具有同步计数，因此它包含一个不具有同步计数的子表达式 S{m,n} 。存在单词 $u = u1, \ldots, u_k$ 和 $v = v1, \ldots, v_{k+1}$，它们构成了 S{m,n} 具有同步计数的反例，即 v 是 u 的前缀。不失一般性地，假设 k 是能够得到此类反例的最小数。在 CA(R) 中，由于它满足附录 B.1 的条件，存在一条路径 π，通向计数器 x 的计数循环上的一个入口转移的源状态 q_i；该计数器与子表达式 S{m,n}相连。设 w 为该路径对应的单词。CA(R) 中必须存在两条从 q_i 出发、在 x-扇出中读取 v 的路径。其中一条路径 π_1最终通过一次转移到达 last(x) 中的某个状态 q，并将 x 递增 k 次（根据引理 2，这对应于处理单词 $v_1 \cdots v_{k+1}$）；另一条路径 π_2对应于 $u_1 \cdots u_k$的一个前缀。根据引理 2，我们知道它至多将 x 递增 $k-1$ 次；如果它递增的次数更少，就意味着我们可以从 u 中移除一些单词 $u_l \ldots u_k$，$l \leq k$，而 v 仍将是该单词的前缀。这意味着我们本可以取更小的 k，并且仍能找到同步计数的反例，但 k 已经是满足条件的最小值。因此，π_2将 x 恰好递增 $k-1$次，现在可以应用引理 6，找到 DCSA^a(CA(R)) 的一条路径，该路径读取 wv，并最终到达状态 (T, Act)，其中包含某个活动计数器 $x_S \in$ Act，且 $q^+ \in S$。然而，由于 $q \in$ last(x)，CA(R) 中存在某个转移 δ，它从状态 q 开始，并在其更新中包含 $x := x+1$。现在，在构造使用 δ 的、从 (R, Act) 出发的某个转移时，我们将得到无法解决的计数器复制，因此 DCSA^a(CA(R)) 不存在，这就产生了矛盾。

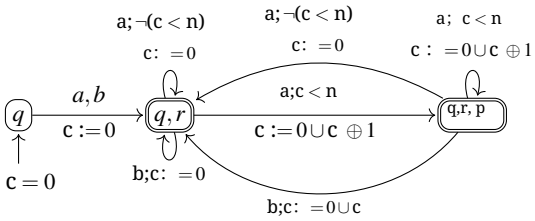
(⇐) 反证。假设 R 具有同步计数，但 DCSA^a(CA(R)) 不存在。取一条 π 在某个单词 w 上到达某个状态 (T, Act) 的最短路径，使得 DCSA^a(CA(R)) 在从 (T, Act) 创建某个转移时失败。这必然意味着存在某个 $x_S \in$ Act，以及某个 $q^+ \in S$ 且 $q \in$ last(x)。令 S{m,n} 为 R 的子表达式，其在 CA(R) 中的计数循环使用计数器 x。由引理 7 可得两个单词 u、v、$w = uv$，以及某个 q_i的某个 x-扇出中的两条路径 π_1 和 π_2，其中 π_1 以 q 结束并将 x 递增 $k+1$次，而 π_2 将 x 递增 k 次。根据引理 2 以及 $q \in$ last(x)，我们可以将 v 划分为 v1、...、v_{k+2}（对应于路径 π_1），其中 $v_i \in$L(S)。如果向 π_2 添加一条在某个单词 v′ 上的路径，使其以 last(x) 中的某个状态结束（且不添加任何使 x 递增的转移），那么我们便可将 vv′划分为 $u_1 \ldots u_{k+1}$。然而，v 是 vv′的前缀，这意味着我们找到了单词 v1、...、v_{k+2}和 $u_1 \ldots u_{k+1}$，它们表明 S{m,n}不具有同步计数，矛盾。

来自 **[23]** 中类的反例

[23] 的作者声称，如果正则表达式 R 仅包含类型为（$\alpha_1\alpha_2 \ldots \alpha_n$）{n}的计数循环，其中 α_1 表示一个与来自α_i、 $2 \leq i \leq n$ 的任何符号均不相交的符号集合，那么根据其算法构造的 CA 可以确定化为接受相同语言的 CSA。然而，对于正则表达式 .*.(ab){n}（它满足



(a) 使用 [23] 中的构造创建的计数自动机



（b）通过 [23] 中的算法确定化 CSA

图 3：正则表达式 .*.(ab){n} 对应的计数自动机和确定化 CSA，其中所标记状态的终止条件为 $c \geq m$，$\perp$ 所有其他状态的终止条件为此，缺少的守卫条件为 $\top$。

该句法判据），我们通过根据 [23] 的构造得到图 3a 中的计数自动机 A。图 3b 中所示的确定化 CSA A′ 是根据 [23] 中的算法构造的。然而，仅包含 $m+1$ 个符号 a 的单词被 A′ 接受，却不被 A 接受。

F 识别字母标记计数

以下简单过程确定正则表达式 R 是否具有字母标记计数。它通过归纳方式计算正则表达式 R 的集合 T_R，该集合包含所有标记字母集合。我们使用Σ(R)表示 R 中出现的所有符号的集合。

– 对于 $R = a$，$a \in \Sigma$，我们有 $T_R = \{\{a\}\}$，因为 L(R) 中只有一个单词 $a \in$。对于 $R = \varepsilon$，我们有 $T_R = \{\}$。– 对于 $R = R_1 | R_2,$，我们知道，对于每个 $T_1 \in T_{R_1}$，$L(R_1)$ 中的每个单词都恰好包含一个来自 T_1 的符号。然而，$L(R_2)$ 中可能存在不包含任何来自 T_1 的符号的单词，或者包含多个这样的符号。如果给定某个来自 T_{R_2} 的 T_2，我们可以看到，来自 $\Sigma(R_2) \setminus T_2$ 的符号可能出现多次，也可能一次都不出现，因此，我们只需测试 $T_1 \cap (\Sigma(R_2) \setminus T_2) = \emptyset$。然后我们取

$$T_R = \{ T_1 \cup T_2 \mid T_1 \in T_{R_1}, T_2 \in T_{R_2}, T_1 \cap (\Sigma(R_2) \setminus T_2) = \emptyset \text{ and } T_2 \cap (\Sigma(R_1) \setminus T_1) = \emptyset \}.$$

– 对于 $R = R_1 R_2,$，我们知道，来自 $T \in T_{R_1}$ 的符号在来自 R_1R_2 的单词中恰好出现一次，前提是它们不出现在由 R_2, 给出的单词中，即 $T \cap \Sigma(R_2) = \emptyset$。因此我们可以取

$$T_R = \{ T \mid T \in T_{R_1} \text{ and } T \cap \Sigma(R_2) = \emptyset \} \cup \{ T \mid T \in T_{R_2} \text{ and } T \cap \Sigma(R_1) = \emptyset \}.$$

– 对于 $R = R'^*$，不存在标记字母集合，因为重复使用 R′ 中的单词会迫使每个字母出现多次。对于这种情况，$T_R = \{\}$。

带非同步计数的 G 正则表达式

以下列出来自 [23] 的基准数据集中 24 个非同步扁平正则表达式，其计数器上界之和大于 20：

```
^(.*?){1,128}$ (.*){1,32000} [bc]^(.*){0,254}$ (.+){25}(.*) ((?:[^\n]*\n?){1,40}) (\n\s+)(criterion .*\n)(\s.+){1,99} ^((?:[\w\d\-_]+)\W(?:[\w\d]+)\W){1,32}? *(.+) ^((?:[\w\d\-_]+)\W(?:[\w\d]+)\W){1,32}? *(\w+.+) ^[a-z0-9]+(?:[._\\-]*[a-z0-9])*@(?:[a-z0-9]+(?:[-a-z0-9]*[a-z0-9]+.){1,63}[a-z0-9]+$ ^[a-z0-9]+(?:[._\\-]*[a-z0-9])*@(\w+(?:[-.]\w+)*\.){1,63}[a-z0-9]+$ ^((?:[0-9]+\s*){1,255})(.*)?$ REK\: (?:[a-zA-Z]{2}[0-9]{2}[a-zA-Z0-9]{4}[0-9]{7}(?:[a-zA-Z0-9]?){0,16}) ICE_Dims.{92}((_?(X\d+)){13})[a-zA-Z]{2}[0-9]{2}[a-zA-Z0-9]{4}[0-9]{7}(?:[a-zA-Z0-9]?){0,16} https?://(?:\S+/){4}(?:[0-9a-f]{40})/?(?:[^#\s]+)?(?:#(\S+))? /\r\n\s*接受语言\s*:3a\s*(?:[^\r\n]*?\x2c){20}/mi ^Java 数据库连接:db2:///((?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?).){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))|(?:(?:(?:(?:[A-Z|a-z]):(?:[\w+-]){0,61}(?:[\w]?[.])*)(?:(?:[A-Z|a-z]):(?:[\w+-]){0,61}(?:[\w]?))))):([0-9]{1,5})/([0-9|A-Z|a-z|_|#$]{1,16})$ ^[-\w&#39;+*$^&%=~!?{}#|/']{1}([-\w&#39;+*$^&%=~!?{}#|'.]?[-\w&#39;+*$^&%=~!?{}#|']{1}){0,31}[-\w&#39;+*$^&%=~!?{}#|']?@((?:[a-zA-Z0-9]{1}([-a-zA-Z0-9]?[a-zA-Z0-9]{1}){0,31})\.{1})+([a-zA-Z]{2}|[a-zA-Z]{3}|[a-zA-Z]{4}|[a-zA-Z]{6}){1}$ ^([a-z0-9]+([\-a-z0-9]*[a-z0-9]+)?\.){0,}([a-z0-9]+([\-a-z0-9]*[a-z0-9]+)?){1,63}(\.[a-z0-9]{2,7})+$[-\w'+*$^&%=~!?{}#|/']{1}([-\w'+*$^&%=~!?{}#|'.]?[-\w'+*$^&%=~!?{}#|']{1}){0,31}[-\w'+*$^&%=~!?{}#|']?@((?:[a-zA-Z0-9]{1}([-a-zA-Z9]?[a-zA-Z0-9]{1}){0,31})\.{1})+([a-zA-Z]{2}|[a-zA-Z]{3}|[a-zA-Z]{4}|[a-zA-Z]{6}){1} Java 数据库连接:db2:///((?:(?:(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?).){3}(?:25[0-5]|2[0-4][0-9]|[01]?[0-9][0-9]?))|(?:(?:(?:(?:[A-Z|a-z]):(?:[\w+-]){0,61}(?:[\w]?[.])*)(?:(?:[A-Z|a-z]):(?:[\w+-]){0,61}(?:[\w]?))))):([0-9]{1,5})/([0-9|A-Z|a-z|_|#$]{1,16}) /^\s*接受语言\s*:3a\s*(?:[^\r\n]*?\x2c){20}/mi /PUTOLF\s+((\S+\s+){4}[^\s]{256}|(\S+\s+){6}[^\x3c]{512})/i /^.*HTTP.*\r\n(.+\x3a\s+.+\r\n){31,}/
```