从历史谜题到语法约束：圆形划分、广义游程编码与多项式时间可判定性

Omid Khormali[1], * Ghaya Mtimet[1] Nuh Aydin[2]

[1]埃文斯维尔大学数学系，美国印第安纳州埃文斯维尔，邮编 47722
电子邮箱：ok16@evansville.edu，gm163@evansville.edu

[2]凯尼恩学院数学与统计学系，美国俄亥俄州甘比尔，邮编 43022
电子邮箱：aydinn@kenyon.edu

摘要

受一个类似于著名约瑟夫斯问题的历史组合问题启发，我们研究圆形划分，在确定有限自动机中构造问题，并开发用于解决此类问题的算法。该历史问题涉及将个体排列在一个圆上，并每隔 k 个人淘汰一人，直到剩下所需的组为止。我们分析圆形划分的移除方法和非移除方法，建立平衡划分的条件，并给出明确的算法。我们引入分割字母表上的广义游程编码，以捕捉交替字母模式，并利用第二类斯特林数计算其基数。通过将这些组合结构与形式语言理论联系起来，我们提出一个存在性问题：给定一个字典上的上下文无关文法以及关于字母的块模式约束，是否存在有效句子？我们通过证明块语言是正则的，并应用标准句法分析技术，证明该问题可在多项式时间内判定。文中给出了完整的算法及复杂度分析，并通过在历史实例和合成实例上的实现对其进行了验证。

关键词：确定有限自动机、约瑟夫斯问题、圆形划分、上下文无关文法、形式语言、可判定性、块语言。

2020年数学主题分类：68Q45,68R15, 68W40

1 引言

本文的灵感源自一种巧妙的历史策略，它用来解决一个出现在伊斯兰叙事传统中的有趣问题。这个故事的文字版本见于*Hāshiyat al-Aṭṭār alá Jam*

*通讯作者：ok16@evansville.edu

《伊本·苏布基汇编》，这是埃及学者 al-Aṭṭār 的注释；他生活在19世纪初（卒于回历1250年／公元1835年）。[1] 除此文本来源外，伊斯兰学者和传教士也熟知这个故事，他们曾在口头讲述中提及它，并将其归于伊斯兰早期时期。

据此记载，一组由30个个体组成的人，其中包括15名信徒和15名不信道者，正身处一艘陷入危机的船上。为了拯救船只，人们认为必须让一半乘客离船。为了避免一艘有沉没危险的船只沉没而让一定数量的乘客离船，这似乎是一个古老而常见的故事，在伊斯兰教出现之前就已存在于各种传统之中。众人向一名男子求助，请他想出公平的方法来决定谁应该留下。为了确保信徒的安全，这名男子根据阿拉伯字母表的结构和一句精心挑选的句子，设计了一种象征性策略。他先将阿拉伯字母分成两组：一组带点，一组不带点；随后编写了一句恰好由30个字母组成的句子，两类字母各占一半。乘客围成一个圆，每人按顺序对应句子中的一个字母。信徒对应的字母取自其中一组，不信道者对应的字母则取自另一组。从句子的第一个字母开始，每隔九人便循环淘汰一人，直到只剩下15人。目标是确保留下的所有个体都属于信徒组。

该书所保存版本中使用的诗句是 [2]

الله يقضي بكل يسر ويرزق الضيف حيث كان

尽管 al-Aṭṭār 的注释为这一记忆策略提供了易于查阅的书面记录，但类似数学谜题的更早记载见于中世纪阿拉伯传记文学。14 世纪学者 Ṣalāḥ al-Dīn al-Ṣafadī（卒于伊斯兰历 764 年／公元 1363 年）在其传记辞典*Ayān al-Aṣr wa-Awān al-Naṣr* [5]中记述了这类问题。al-Ṣafadī 报告说，他听到了上面的诗句（第一首诗句），并因此受到启发，为同一个问题创作了自己的记忆解法。他的诗句是：[3]

ولما فتنت بلحظ له عذلت فما خفت من شامت

al-Ṣafadī 还为这个基于圆圈的计数谜题创作了另外几首记忆诗句，我们将在第 3 节中探讨。这些 14 世纪的记载表明，至少在 al-Aṭṭār 的作品问世前五个世纪，阿拉伯语学者之间就已流传着此类计数问题及其记忆解法；这也说明，伊斯兰思想文化中的娱乐数学传统由来已久。第 3 节将详细考察 al-Ṣafadī 的记忆诗句，并展示它们如何编码了圆圈排列问题的数学解法。

这种对句子的策略性运用，结合数字 9 的节奏和二元字母划分，揭示了数学模式与语言之间引人入胜的结合。这一框架的

[1]参见*Hāshiyat al-Aṭṭār alá Jam al-Jawāmi*，第2卷，Dār al-Kutub al-Ilmiyya，第264页。[2]这句话可译为：“上帝轻易地裁定一切事务，并为客人提供所需，无论他身在何处。”[3]这句话可译为：“当我为他的一瞥所倾倒时，虽遭人责备，却不惧任何讥讽者。”

问题涉及从一个由 30 人组成的圆中移除每第九个人，与经典约瑟夫问题十分相似 [6] ，后者早于伊斯兰文献，并且一直是组合学和算法设计领域广泛研究的对象。然而，将字母分配融入一个经过精心构造、旨在控制结果的阿拉伯语句子中，似乎是伊斯兰叙事或注释语境所独有的。[4] 虽然数学参数相似，但 30 和 9 这两个数字究竟是在伊斯兰文学内部独立形成的，还是反映了更早的约瑟夫问题型表述所产生的影响，仍然是一个悬而未决的问题。这一主题更适合由伊斯兰科学史学家和思想传播领域的研究者来探讨。

一个极为相似的版本见于 James R. Newman 的《数学世界》（1956） [3]，其中一艘船上载有15名土耳其人和15名基督徒。在这个版本中，每第九个人都会被淘汰，目的是确保基督徒存活。相比之下，Newman 的版本没有伊斯兰表述中借助一个由30个字母组成的阿拉伯语句子进行的语言编码，也没有与之相关的字母划分。尽管目前尚不清楚 Newman 的版本是源自伊斯兰叙事，还是独立发展而来，但文献记录表明，阿拉伯语版本早于西方印刷品的出现，而这两个版本之间的关系仍不明确。还需要进一步的历史研究，以确定归于 Newman 的版本是否在更早的西方文学中有先例，或是否受到伊斯兰文献的影响。

本文从离散数学的视角分析这种深植于文化背景中的淘汰方法。我们探究其组合性质，将其与标准的约瑟夫斯问题形式进行比较，并讨论其对算法公平性和群体选择的影响。

首先，我们考察将 n 个对象划分为 r 组的问题。我们通过将对象放置在一个圆上，并使用循环步进程序分配组标签来处理这个问题。我们的目标是探究这种圆形方法如何实现均衡且公平的分组。在本文中，我们将步长 k 定义为圆上的位置之间的间隔。从一个初始位置开始（我们将其指定为位置 1，且一开始不移除该处对象），我们移除位置 k处的对象，然后依次移除位置 $2k$、位置 $3k$等处的对象，并以 n 取模的方式绕圆继续（因此，到达位置 n后，我们将位置超过 n 的部分除以 n并取余数，继续前进）。请注意，从起始位置到第一个被移除对象的距离为 $k-1$个位置，而任意两个连续被移除对象之间的距离恰好为 k 个位置。为清楚起见，我们所说的“步长 k”是指，相对于起始位置，被移除对象位于位置k、 $2k$、 $3k$，.....，并在绕圆时按 n 取模计算。

2 划分问题

设 n 为要划分成 $r \geq 2$ 组的对象数量。我们将这些对象随机排列在一个圆上，并为其位置贴上标签。标签可以沿圆顺时针或逆时针依次标记。然后，我们按标签顺序在各个位置间移动，执行步长固定为 $k \geq 1$ 的循环步进程序。我们考虑两种不同的方法：非移除方法，即已访问对象留在圆上，只标记为已访问；以及移除方法 ，即将已访问对象从圆上移除

[4]请参阅 https://niefrar.org/?p=74 ，了解该故事的现代描述。

分配完成后，与经典约瑟夫问题类似。

在非移除方法中，传统的均衡划分使用步长 $k = r$ 进行模运算，并且每一轮总是标记第一个对象。然而，许多应用在保持平衡的同时，还要求采用不同的分配顺序。我们的步长参数 k 将这一方法推广开来，其中与 r 的最大公约数关系决定了 k 的可行取值。无论 k 如何，最终的组大小都保持平衡，但对象分配的序列会有所不同，从而能够针对局部性、容错性和访问模式进行优化。

2.1 划分算法

在下述定理中，我们给出算法，分别使用移除方法和非移除方法来解决这一问题。

定理 2.1. 对于任意满足 $r \leq n$ 的正整数 n、r 和 k，都存在一种将 n-集合 $[n] = \{1, 2, \ldots, n\}$ 划分为 r 组的平衡划分，其步长为 k。

该定理可以通过移除方法或非移除方法证明。这两种方法可能会产生不同的划分，但都能达到目标。我们分别针对这两种情况给出解法。

注释1。当 $k > n$ 时，在由 n 个对象组成的圆上每隔 k 个位置移动一步，等同于每隔 k 取模 n 个位置移动一步。因此，在证明中不失一般性，我们可以假设 $k \leq n$，但定理对所有正整数 k 均成立。

证明 *1*：移除方法。我们通过类似约瑟夫问题的消除过程给出一个构造性证明。令 $s \geq 0$ 为满足 $r \mid (n+s)$ 的最小非负整数。我们可以将 $s = a \cdot$ 表示为 $\lceil n/r \rceil + b$，其中 $a \geq 0$ 且 $0 \leq b < \lceil n/r \rceil$。我们引入 s 个虚拟对象 $\{s_1, s_2, \ldots, s_{a \cdot \lceil n/r \rceil + b}\}$，然后按以下步骤进行：

1. 设 $N = n + s$ 为对象总数，$R = r + a$ 为组的总数。

2. 将所有 N 个对象围成一个圆，并按 1 到 N 编号，其中 n 个真实对象占据位置 1 到 n，s 个虚拟对象占据位置 $n+1$ 到 N。

3. 从位置 1 开始，以步长 k 选择对象，并移除每个被选中的对象。（也就是说，先选择位置 k，然后是位置 $2k$，依此类推，其中位置 1 作为参考点。）选择过程中：

   - 依次将对象分配到各组，使得前 $N/R = \lceil n/r \rceil$ 个对象分配给第一个 t 组，接下来的 N/R 个对象分配给第二组，依此类推。这样，每组恰好分得 N/R 个对象。
   - 为遇到的虚拟对象维护一个计数器 i。将初始值设为 $i = 0$。
   - 遇到虚拟对象时，我们执行– 递增计数器，
   $i \leftarrow i + 1$– 如果 $i \leq b$，则必须将此虚拟对象分
   配给第 i 组。
     - 如果当前正在填充组 i，则将虚拟对象直接分配给组 i

∗ 如果当前正在填充组 $j \neq i$，则：- 如果组 i 尚未填满，则将新的虚拟对象放入组 i，并继续用真实对象填充组 j - 如果组 i 已由真实对象填满，则将虚拟对象与组 i中的一个真实对象交换– 如果 $i > b$，则此虚拟对象会进入其中一个虚拟组（$r+1$至 R），通常分配给当前正在填充的组

- 此过程可确保：
  – 遇到的第1个虚拟对象总是被放入第1组– 遇到的第2个虚拟对象总是被放入第2组– .....以此类推，直到第 b个虚拟对象被放入第 b组– 第1组至第 b 组中，每组恰好包含一个虚拟对象– 其余所有虚拟对象都放入第 $r+1$ 组至第 R组

继续操作，直到所有对象都分配到第1、第2、.....、第 R组。

4. 从第1组至第 b组中移除 b 虚拟对象。

5. 丢弃　　这些 a 虚拟组（第 $r+1$ 组至第 R组）及其虚拟对象。

注意，由于 s 是满足 $r \mid (n+s)$ 的最小非负整数，因此有 $s < r$。因此，$b \leq s < r$，确保前 b 个虚拟对象可以分配到 b 个不同的真实组。

现在我们证明该算法生成的是平衡划分。步骤 3 之后，我们有：

- 组 1 至 b：每组有 $\lceil n/r \rceil$ 个对象（包括一个虚拟对象）
- 组 $b+1$ 至 r：每组有 $\lceil n/r \rceil$ 个真实对象
- 组 $r+1$ 至 R：每组有 $\lceil n/r \rceil$ 个虚拟对象

移除虚拟对象和虚拟组后：

- 组 1 到 b：每组有 $\lceil n/r \rceil - 1 = \lfloor n/r \rfloor$ 个实际对象
- 组 $b+1$ 到 r：每组有 $\lceil n/r \rceil$ 个实际对象

由于 $n = r \cdot \lfloor n/r \rfloor + b$（其中 $b = n \mod r$），要实现平衡划分，我们需要恰好有 b 组各包含 $\lceil n/r \rceil$ 个对象，以及 $r-b$ 组各包含 $\lfloor n/r \rfloor$ 个对象。这正是我们实现的目标。 □

注 2。 在证明中，为方便起见，前 b 个虚拟对象被分配给前 b 个真实组。不过，这一选择是任意的，前 b 个虚拟对象可以被分配给任意 b 个真实组，而不会影响论证。

证明 *2*：非移除方法。 我们提供另一种构造性证明，在整个划分过程中，对象始终保留在固定位置。我们首先定义两个将在分类讨论中使用的核心算法。

算法 A：
输入： m 个对象，使得 gcd$(m, k) = g$。
注意：与移除方法中位置 1 仅作为参考点不同，在算法 A 中，起始位置会分配给一个组。这一区别不会影响划分结构，因为该算法会将对象划分为由 gcd(m, k) 确定的循环。

1. 将对象围绕一个圆排列，并标记为 1 到 m。
2. 从位置 1 开始，将其分配给组 1。
3. 步进 k 位置，将已访问对象分配到同一组，直到返回起点 .
4. 从下一个未访问的位置开始，为下一个组重复此过程，直到所有对象都分组完毕。

算法 B：
输入： m 对象，使得最大公约数$(m, k) = g$ 且最大公约数$(m/g, k) = f$。

1. 应用算法 A 创建初始组。
2. 将每组的对象分别放置在一个圆上。
3. 使用算法 A，以m/g 个对象为单位，按照指定步长 k 将每个组划分为子组。

请注意，算法 A 会创建 g 个组，每组恰好包含 $\frac{m}{g}$ 个对象。而算法 B 会创建 gf 个组，每组恰好包含 $\frac{m}{gf}$ 个对象。此外， m（即对象的数量）将根据具体情况适当选取。

现在，我们根据 n、 r和 k之间的关系来考虑各种情况，并在每种情况下适当选取m 。

情况 1： $k \geq r$
情况 1.1：最大公约数$(k,\ r) = r$

- 如果最大公约数$(n,\ k) = r$：算法 A 使用 $m = n$ 个对象创建 r 个组，每组大小为 n/r 。由于$r \mid n$，因此这种划分完全均衡。
- 若 gcd$(n, k) \neq r$：添加 s 个虚拟对象（其中 $s < r$）后，算法 A 使用 $m = n + s$个对象创建 r 个大小为 $(n + s)/r$ 的组。虚拟对象分布在 s个组中，每组一个。请注意，在最坏情况下，如果某个组包含多个虚拟对象，我们可以将一个虚拟对象与一个不含虚拟对象的组中的真实对象交换。完成这些交换后，我们得到 s 个组，每组恰好包含一个虚拟对象。移除这些对象后，剩下 s 个包含 $(n + s)/r - 1$ 个对象的组，以及 $r - s$个包含 $(n + s)/r$ 个对象的组，从而实现平衡。

情况 1.2：最大公约数(k，r) $\neq r$

• 引入 t 个虚拟组，每组包含 $\lceil n/r \rceil$ 个虚拟对象，其中

$$t = \text{the smallest integer such that } \gcd(k, r+t) = r+t.$$

• 虚拟对象的总数为 $t \cdot \lceil n/r \rceil$，因此对象总数变为 $N = n + t\lceil n/r \rceil$，共有 $r+t$ 组，步长不变，为 k。

• 与情况 1.1 一样，我们现在考虑以下两种子情况

$$\gcd(N, k) = r+t \qquad \text{or} \qquad \gcd(N, k) \neq r+t.$$

• 如果最大公约数(N，k) $= r+t$，则对这些对象应用算法 A，会得到 $r+t$ 个大小相等的组。将出现在虚拟组中的任何真实对象与出现在真实组中的虚拟对象交换。完成这些交换后，所有虚拟组都只包含虚拟对象，且所有虚拟对象都只在虚拟组中。

• 如果最大公约数(N，k) $\neq r+t$，则添加 s 个虚拟对象（其中 $1 \leq s \leq r+t-1$），使得最大公约数($N+s$，k) $= r+t$，然后对这些对象应用算法 A。由此得到 $r+t$ 个组，每组大小为$\frac{N+s}{r+t}$。

对象总数为 $N+s = n + t\lceil n/r \rceil + s$。若所有组的大小均为 $\lceil n/r \rceil$，则总数将为 $(r+t)\lceil n/r \rceil$。超出的数量为

$$(n + t\lceil n/r \rceil + s) - (r+t)\lceil n/r \rceil = n + s - r\lceil n/r \rceil.$$

当 $r \nmid n$ 时，令 $b = n \bmod r$，其中 $1 \leq b < r$。于是 $\lceil n/r \rceil = \lfloor n/r \rfloor + 1$，因此

$$n - r\lceil n/r \rceil = n - r(\lfloor n/r \rfloor + 1) = (n - r\lfloor n/r \rfloor) - r = b - r.$$

因此，超出的数量为 $s + b - r$，这等于大小为 $\lceil n/r \rceil + 1$ 的组的数量。
我们通过从大小为 $\lceil n/r \rceil + 1$ 的组中选取，指定 t 个虚拟组（若有此类组）。如果此类组的数量少于 t，则将它们全部指定为虚拟组，并从大小为 $\lceil n/r \rceil$ 的组中选出其余部分。

情况 (a)： 如果 $s \leq r$，则有 $s + b - r < r$ 个大小为 $\lceil n/r \rceil + 1$ 的组。在 t 个虚拟组中放入虚拟对象后，留给 r 个真实组的虚拟对象至多有 $s < r$ 个。

情况 (b)： 如果 $s > r$，那么就有 $s + b - r$ 组，每组大小为 $\lceil n/r \rceil + 1$。我们从这些较大的组中选出所有 t 个虚拟组。将 $t(\lceil n/r \rceil + 1)$ 个虚拟对象放入虚拟组后，留给真实组的剩余虚拟对象为

$$(t\lceil n/r \rceil + s) - t(\lceil n/r \rceil + 1) = s - t < r.$$

在这两种情况下，真实组中至多剩下 $r-1$ 个虚拟对象。和之前一样，我们将虚拟组中的真实对象与真实组中的虚拟对象互换，使所有虚拟组只包含虚拟对象。然后，我们通过将多余的虚拟对象与不含虚拟对象的组中的真实对象互换，确保每个真实组至多包含一个虚拟对象。

• 最后，我们移除所有虚拟组并删除所有虚拟对象。此时，每个真实组中包含的真实对象数量为 $\lceil n/r \rceil$ 或 $\lfloor n/r \rfloor$。

情况 2： $k < r$情况 2.1： $k > 1$
且最大公约数$(r, k) = k$

• 我们添加 $s \geq 0$ 个虚拟对象，使得最大公约数$(n + s, r) = k$。

• 我们使用算法 B 处理 $m = n + s$ 个对象。该算法首先创建 k 组，每组大小为$(n + s)/k$。

• 然后将每组划分为 r/k 个子组，每个子组大小为$(n + s)/r$。

• 由于 $s < r$，移除虚拟对象（每个子组至多一个）即可实现平衡。

情况 2.2： $k > 1$ 且最大公约数$(r, k) \neq k$

• 与情况 1.2 类似， t添加虚拟组，使得最大公约数$(r+t, k) = k$。每个虚拟组包含 $\lceil n/r \rceil$ 个虚拟对象。因此，对象总数变为 $N = n+t \cdot \lceil n/r \rceil$.

我们再次考虑两种子情况：

– 如果最大公约数$(N, r + t) = k$，则对包含 $m = N$ 个对象、 $r + t$ 个组且步长为 k的情况应用算法 B。之后，将出现在虚拟组中的任何真实对象与实际组中的虚拟对象互换，使所有虚拟组仅包含虚拟对象。然后移除虚拟组和虚拟对象。

– 如果最大公约数$(N, r + t) \neq k$，则添加 s 个虚拟对象，使得最大公约数$(N + s, r + t) = k.$。对包含 $m = N + s$ 个对象、 $r + t$ 个组且步长为 k的情况应用算法 B。之后，将在虚拟组中找到的任何真实对象与最初分配给虚拟组的虚拟对象互换。对于真实组，确保每个真实组中至多有一个虚拟对象。最后，移除所有虚拟组和虚拟对象。

情况 2.3： $k = 1$

• 将 $\lceil n/r \rceil$ 个连续对象直接分配给前 $n \mod r$ 组，并将 $\lfloor n/r \rfloor$个对象分配给其余组，根据定义即可得到平衡划分。

在所有情况下，最终划分中的每组都包含 $\lfloor n/r \rfloor$ 个或 $\lceil n/r \rceil$ 个对象，其中恰有 $n \mod r$ 组包含 $\lceil n/r \rceil$ 个对象，从而实现平衡划分。 □

注3。 在第二个证明中，对于情形*1.1*、*1.2*和*2.2*，我们没有明确说明：在确保虚拟组中的所有对象都是虚拟对象后，真实组中剩余的虚拟对象应分配给标签较小的真实组。我们只给出了这样的条件：所有交换完成后，每个真实组至多包含一个虚拟对象。

2.2 划分示例

现在我们给出一些划分算法的示例。

示例 1。 移除方法（证明 *1*）：

设参数为 $n = 10$、$r = 3$、$k = 4$。我们需要找到最小的 $s \geq 0$，使得 $r \mid (n+s)$，即 $3 \mid (10+s)$。因此，我们有 $s = 2$。此外，$s = a \cdot \lceil n/r \rceil + b$，其中 $\lceil 10/3 \rceil = 4$。于是，我们有$a = 0$ 和 $b = 2$。并且有 $N = n + s = 12$ 个对象和 $R = r + a = 3$ 组。

我们将 *10* 个真实对象（位置 *1 - 10*）和 *2* 个虚拟对象（位置 *11-12*）围绕一个圆排列。从位置 *1*（参考点）开始，以 $k = 4$ 为步长，选取并移除位置为 4、8、12、*.....*（取模 *12*）的对象。

选择顺序	位置	*Type*	分配给
1	*4*	*real*	第*1*组
2	*8*	*real*	第*1*组
3	*12*	虚拟	第*1*组（第*1*个虚拟 $\rightarrow$ 第*1*组）
4	*5*	*real*	第*1*组
5	*10*	*real*	第*2*组
6	*3*	*real*	第*2*组
7	*11*	虚拟	第*2*组（第*1*个虚拟 $\rightarrow$ 第*2*组）
8	*7*	*real*	第*2*组
9	*6*	*real*	第*3*组
10	*9*	*real*	第*3*组
11	*2*	*real*	第*3*组
12	*1*	*real*	第*3*组

步骤*3*之后（移除虚拟元素之前），我们得到

$$G_1 = \{4, 5, 8, 12\}, \quad G_2 = \{3, 7, 10, 11\}, \quad G_3 = \{1, 2, 6, 9\}.$$

移除第*1*组和第*2*组中的 $b = 2$ 虚拟对象后，我们得到

$$G_1 = \{4, 5, 8\}, \quad G_2 = \{2, 7, 10\}, \quad G_3 = \{1, 2, 6, 9\}.$$

这使我们得到一个大小为*4*的组和两个大小为*3*的组，实现了平衡划分：其中$b = n$ 模 $r = 1$ 组有 $\lceil 10/3 \rceil = 4$ 个对象，$r - b = 2$ 组有 $\lfloor 10/3 \rfloor = 3$ 个对象。

示例2。 非移除方法，情况 *1.1*：

假设参数为 $n = 10$、$r = 4$、$k = 4$。这里，$k \geq r$，最大公约数$(k,\ r)$ =最大公约数$(4,\ 4)$ $= r$，但最大公约数$(n,\ k)$ =最大公约数$(10,\ 4)$ $= 2 \neq r$。因此，这是情况 *1.1*，即第二个子情况。

我们必须添加数量最少的虚拟对象 s，使得 最大公约数$(n + s, k)$ $= r$。这里，$s = 2$，因为 最大公约数$(12,4)$ $= 4 = r$。因此，我们在算法 *A* 中使用 $m = N = 12$ 个对象和 $r = 4$ 组。最后，移除这 *2* 个虚拟对象。

Step	起始位置	访问序列	组
1	*1*	*1, 5, 9*	第*1*组
2	*2*	*2, 6, 10*	第*2*组
3	*3*	*3, 7, 11*（虚拟）	第*3*组
4	*4*	*4, 8, 12*(虚拟)	第*4*组

移除这两个虚拟对象后，最终划分为

$$G_1 = \{1, 5, 9\}, \quad G_2 = \{2, 6, 10\}, \quad G_3 = \{3, 7\}, \quad G_4 = \{4, 8\}.$$

示例 3. 非移除方法，情况 *1.2*：
假设参数为 $n = 12$、$r = 3$、$k = 4$。这里 $k \geq r$ 且 最大公约数(k, r) =最大公约数$(4, 3) = 1 \neq r$，因此我们处于情况 *1.2*。

我们添加 $t = 1$ 个虚拟组，因为 最大公约数$(4,4) = 4$。每个虚拟组分配 $\lceil 12/3 \rceil = 4$ 个虚拟对象。因此有 $N = 16$ 个对象、$r + t = 4$ 个组，步长为 $k = 4$。将算法 *A* 中的 $(m = N = 16$ 应用于情况 *1.1* $r = 4, k = 4)$ 可得

Step	起始位置	访问序列	组
1	*1*	*1, 5, 9, 13*（虚拟）	第*1*组
2	*2*	*2, 6, 10, 14*（虚拟）	第*2*组
3	*3*	*3, 7, 11, 15*（虚拟）	第*3*组
4	*4*	*4*、*8*、*12*、*16*（虚拟）	第*4*组（虚拟）

交换步骤：如果有任何真实对象被放入虚拟组，我们就会将它们与落入非虚拟组的虚拟对象交换。因此，更新后的组为

元素	组
1、*5*、*9*、*4*	第*1*组
2, 6, 10, 8	第*2*组
3, 7, 11, 12	第*3*组
13（虚拟）, *14*（虚拟）, *15*（虚拟）, *16*（虚拟）	第*4*组（虚拟）

移除步骤：彻底移除第*4*组（因为其中只包含虚拟对象）。

最终划分为 $r = 3$ 组，如下所示

$$G_1 = \{1, 4, 5, 9\}, \quad G_2 = \{2, 6, 8, 10\}, \quad G_3 = \{3, 7, 11, 12\}.$$

3 历史示例与基于字母表的划分

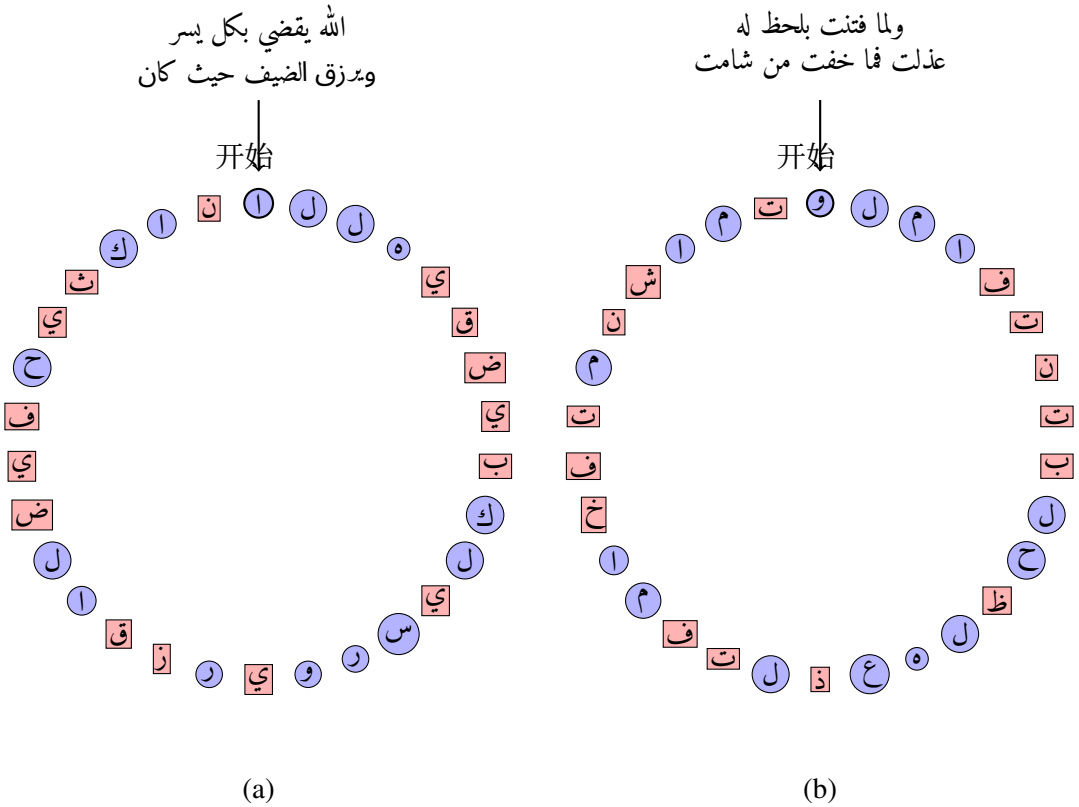
本节展示一个著名的历史示例：三十名个体被分成两个各有十五人的组。这些个体沿圆排列，每个人依次对应阿拉伯语句子中的一个字母，该句子长度为30。淘汰过程每次移除第九个人（等价地，步长为 $k = 9$），直至恰好剩下十五人。为了解释这一演示，我们首先回顾阿拉伯字母表及其经典的二分法：分为带点字母和不带点字母。这种二元划分在句子的构造和乘客的分配中起着核心作用。

阿拉伯字母表划分：完整的阿拉伯字母表（28个字母）可分为以下两个子组：

- 无点字母（*14* 个）：ا، ح، د، ر، س، ص، ط، ع، ك، ل، م، هـ ، و، ى
- 有点字母（*15* 个）：ب، ت، ث، ج، خ، ذ، ز، ش، ض، ظ، غ، ف، ق، ن، ئ، ي

注：字母阿拉伯字母 ي 在阿拉伯正字法中有多种形式。带点的 ي（标准 yā）、带点的 ئ（带 hamza 的 yā）以及不带点的 ى（*alif maqṣūrah*）。对于 ئ，Ryding [4, 第 16 页第 3.3.1 节1] 指出，当 yā 充当 hamza 的座（承载字母）时，会失去它的两个点。然而，由于其底层字母仍然是 yā，而 yā 本身带点，因此根据其基础字母，ئ 被归类为有点字母。[5]

图 1 展示了下方阿拉伯语句子的 30个字母，这些字母排列成一个圆形：蓝色节点表示无点字母，红色节点表示有点字母。从图中标出的初始位置开始计数，每隔九个字母就淘汰一个。结果发现，以这种方式删除的前 15 个字母全都是红色（有点）字母。该阿拉伯语句子此前已介绍过，其字母排列如下。



(a) (b)

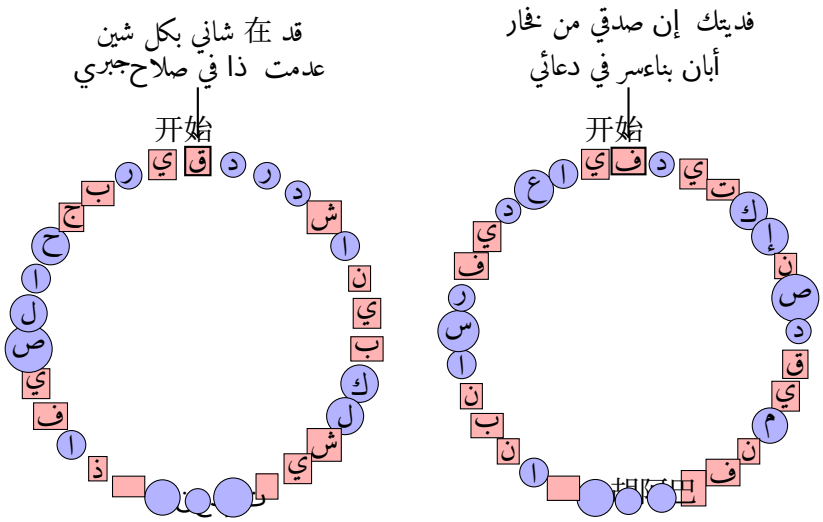
图 1：两首各含 30个字母的阿拉伯语诗句，按顺时针方向排列在圆形上。无点字母显示在圆形内，有点字母显示在矩形内。在这两种情况下，每隔九个位置淘汰一个字母（步长为 $k = 9$），被淘汰的字母均位于矩形内，因此只留下 15 个无点字母。(b) 中的诗句由 al-Ṣafadī 创作。

对应的数列： 我们可以为该句子关联一个整数序列，用于记录每个子字母表中的字母所构成的连续游程。从第一个字母开始，我们统计连续属于同一子字母表的字母数量，然后切换到另一个子字母表并继续计数。这样，该阿拉伯语诗句生成的序列为 (4，5，2，1，3，1，1，2，2，3，1，2，2，1)，反映了无点字母和带点字母交替出现的结构

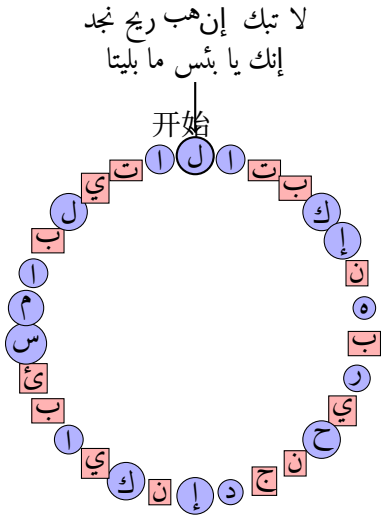
[5]在标准阿拉伯正字法中，*alifmaqṣūrah* (ى)只出现在词尾位置。

如图 1 所示。该序列展示了句子中每个块所包含的各子字母表字母数量，其中，块是由同一子字母表中的字母组成的极大字符串。

下图展示了 al-Ṣafadī 的其他诗句，选自 [5]，每首都排列在一个圆上。



(a) 步长 $k = 7$　　(b) 步长 $k = 7$



（c）步长 $k = 5$

图2：按顺时针方向排列在圆形上的三条阿拉伯语诗句。（a）30个字母（15个无点字母、15个有点字母），步长为 $k = 7$。（b）32个字母（16个无点字母、16个有点字母），步长为 $k = 7$。（c）30个字母（16个无点字母、14个有点字母），步长为 $k = 5$。无点字母（包括艾里夫字母变体，　（ 显示在圆形内，有点字母显示在矩形内。

整数序列　　图2 (a) 中诗句的整数序列为 (1，3，1，1，3，2，3，3，2，1，2，4，2，1，1)，其整数　r

图2 (b) 中诗句的整数序列为 (1，1，2，2，1，2，2，1，3，3，1，1，3，3，2，3，1)，而图2 (c) 中诗句的整数序列为 (2，2，2，1，1，1，1，1，1，2，2，1，1，1，1，2，3，1，1，2，1)，反映了其无点字母和有点字母的交替结构。

3.1 向英文字母表的适配

同样的思路可以从阿拉伯语适配到英语，方法是对英文字母表进行类似的划分。我们不再区分无点字母和有点字母，而是根据几何形状将英文字母表中的大写字母分为两个子组：一组仅由直线构成，另一组至少包含一条曲线。由此得到如下划分

$$\text{Straight-line group (15): } \{A, E, F, H, I, K, L, M, N, T, V, W, X, Y, Z\},$$

$$\text{Curved-line group (11): } \{B, C, D, G, J, O, P, Q, R, S, U\}.$$

举例来说，假设有八个个体，需要将其分成两组，每组四人。我们使用句子

“WE BURDEN”,

该句子恰好包含八个字母。个体围绕一个圆排列，每个个体按顺序分配给句子中的一个字母。从第一个字母开始，我们按照步长5在圆上依次淘汰个体（即每隔六个人淘汰一人）。当剩下四个个体时，淘汰过程停止。在这个例子中，幸存者组成两个平衡的组，每组各有两人。图2展示了该例子。

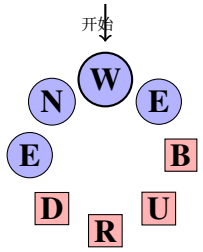


图3：英语句子“WE BURDEN”中的8个字母紧凑地排列在一个圆周上。由直线构成的字母显示在圆形内，由曲线构成的字母显示在矩形内。步长为5时，第6个位置上的个体最先被淘汰。

在这个例子中，对应的序列是 (2, 4, 2)。在第5节（图4）中，我们提供了更多英语句子及其对应数列的示例：(4, 5, 2, 1, 3, 1, 1, 2, 2, 3, 1, 2, 2, 1)、(1, 3, 1, 1, 3, 2, 3, 3, 2, 1, 2, 4, 2, 1, 1)、(1, 1, 2, 2, 1, 2, 2, 1, 3, 3, 1, 1, 3, 3, 2, 3, 1) 和 (2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 2, 3, 1, 1, 2, 1)。在第5节中形式化这个问题，并将上下文无关文法推广到形式语言，将有助于我们开发算法，以判定是否存在具有给定序列的句子。

4 广义游程编码

受历史示例中的数列启发，我们提出一种游程编码的推广形式。首先定义字母表的划分。

[分割字母表] 设 A 为一个有限字母表。A 的非空子集的集合 $\{A_1, A_2, \ldots, A_r\}$ 称为分割字母表，当且仅当：

1. 这些子集两两不交：对于所有 $i \neq j$，$A_i \cap A_j = \emptyset$

2. 每个子集都非空：对于所有 i，$A_i \neq \emptyset$。

我们将其记作 $(A; A_1, \ldots\ldots, A_r)$。此外，如果 $\bigcup_{i=1}^{r} A_i = A$，则称该分割字母表是完备的，且 $\{A_1, \ldots, A_r\}$ 构成 A 的一个划分。

现在，我们将一般化定义如下。

[广义游程编码] 设 $(A; A_1, A_2, \ldots\ldots, A_r)$ 是一个分割字母表。在这个分割字母表上的广义游程编码 （GRL编码）是一个序列

$$v = ((C_1, k_1), (C_2, k_2), \ldots, (C_m, k_m))$$

其中：

1. 每个 C_j 都是来自某个 A_{i_j} 的字母多重集，即对于某个 $i_j \in \{1, \ldots, r\}$，有 $\{C_j\} \subseteq A_{i_j}$，其中 $\{C_j\}$ 表示多重集 C_j 的底层集合（忽略重数）

2. 每个 $k_j = |C_j|$ 都是多重集 C_j的基数，

3. 对于每个 j，索引 $i_j \in \{1, \ldots, r\}$ 由条件 $\{C_j\} \subseteq A_i$ 唯一确定 $_j$

4. 对所有 $j = 1$，2、……、$m - 1$，$i_j \neq i_{j+1}$（连续游程来自划分中不同的子集）。

GRL编码的长度为 $n = k_1 + k_2 + \cdots + k_m$，而 m 称为游程数。

注 4. 对于具有 $r = 2$ 子集 A_1 和 A_2的分割字母表，交替条件可表述为：

$$\textit{either } \{C_{2i-1}\} \subseteq A_1 \textit{ and } \{C_{2i}\} \subseteq A_2, \quad \textit{or} \quad \{C_{2i-1}\} \subseteq A_2 \textit{ and } \{C_{2i}\} \subseteq A_1.$$

这明确展示了两个子集之间的交替。

例 4. 考虑字母表 $A = \{a, b, c, d, e\}$，其分割字母表为 $(A;\ A_1, A_2)$，其中$A_1 = \{a, b\}$ 和 $A_2 = \{c, d, e\}$。对于词 $w = aabccdd$，其 ***GRL*** 向量为 $v = ((C_1, 3), (C_2, 4))$，其中：

- $C_1 = \{a, a, b\}$ 是一个多重集，其中包含 $|C_1| = 3$ 和 $\{C_1\} = \{a, b\} \subseteq A_1$
- $C_2 = \{c, c, d, d\}$ 是一个多重集，其中包含 $|C_2| = 4$ 和 $\{C_2\} = \{c, d\} \subseteq A_2$

在某些应用中，尤其是排列问题和组合计数中，我们不需要追踪每个游程中的具体字母，只需知道它们来自哪个子集以及各有多少个。这引出了以下抽象：

[分块级广义游程长度编码] 设 $(A; A_1、A_2、\ldots\ldots、A_r)$ 是一个分割字母表。该分割字母表上的抽象广义游程长度向量（抽象 GRL 向量）是一个序列

$$v = ((A_{i_1}, s_1), (A_{i_2}, s_2), \ldots, (A_{i_m}, s_m))$$

其中：

1. 每个 $A_{i_j} \in \{A_1, A_2, \ldots, A_r\}$, i_j 表示该子集,

2. 每个 $s_j \in \mathbb{Z}^+$ 都是一个正整数（游程长度），

3. $i_j \neq i_{j+1}$ 对于所有 $j = 1$，2，……，$m-1$（来自不同子集的连续游程).

元组 (A_{i_j}, s_j) 表示 s_j 个连续位置均标记为子集 A_{i_j}，但不指定其中具体出现了哪些字母 A_{i_j}，也不指定它们的重数。

分区级广义游程长度向量为一个基本的组合问题提供了自然的编码方式：将 n 个可区分位置排成一行，并将每个位置分配给 r 组中的一组。由 r 组构成、长度为 n 的排列是一个包含 n 个位置的序列，每个位置都标记有一个来自 $\{1, 2, \ldots, r\}$的组索引。每个组索引 i 对应于划分中的子集 A_i。例如，对于 $n = 5$ 个位置和 $r = 3$ 组，排列 $[1, 1, 2, 2, 3]$ 将前两个位置分配给第1组（子集 A_1），接下来的两个位置分配给第2组（子集 A_2），最后一个位置分配给第3组（子集 A_3）。

分区级 GRL 向量通过将具有相同子集标签的连续位置编码为一个游程，自然地压缩了这类排列。下面的定理证明了这种编码定义了一个双射。

定理 4.1. 设 n 和 r 为满足 $n \geq r$ 的正整数，并设 $(A;\, A_1, \ldots, A_r)$ 为一个完全分区字母表。长度为 n 的各类分区级 ***GRL*** 向量集合的基数如下：

- 一般情形：由 $(A;\, A_1, \ldots, A_r)$ 上长度为 n 的所有分区级 ***GRL*** 向量组成的集合，其基数为 r^n。
- 满射情形：由长度为 n 且所有 r 个子集都至少出现一次的分区级 ***GRL*** 向量组成的集合，其基数为

$$\sum_{i=0}^{r-1} (-1)^i \binom{r}{i} (r-i)^n = r! \cdot S(n, r),$$

  其中 $S(n, r)$ 为第二类斯特林数。

- 平衡情形：设 $n = qr + s$，其中 $q = \lfloor n/r \rfloor$，且 $0 \leq s < r$。长度为 n 的平衡分区级 ***GRL*** 向量集合（其中恰有 s 个子集出现 $q+1$ 次，其余 $r-s$ 个子集出现 q 次）的基数为

$$\binom{r}{s} \cdot \frac{n!}{(q+1)!^s \cdot q!^{r-s}}.$$

  当 r 能整除 n（即 $s = 0$）时，这化简为 $\underline{n!\,(q!)^r}$，其中 $q = n/r$。

证明。我们先建立它与一个适当的排列集合之间的双射。如上所述，由 r 个组构成的长度为 n 的排列是一个序列 $a = (g_1, g_2, \ldots, g_n)$，其中每个 $g_k \in \{1, 2, \ldots, r\}$ 都是一个组索引。位置 k 分配给组 g_k，它对应于该划分的子集 A_{g_k}。为方便起见，我们使用方括号记法表示排列：$a = [g_1, g_2, \ldots, g_n]$。

我们定义两个从排列到分区级 GRL 向量的映射如下。

• 映射 ϕ：给定一个排列 $a = [g_1, g_2, \ldots, g_n]$，我们构造一个分区级 GRL 向量。定义位置 $1 = p_1 < p_2 < \cdots < p_{m+1} = n + 1$，其中 p_{k+1} 是大于 p_k 且满足 $g_{p_{k+1}} \neq g_{p_k}$ 的最小位置（如果不存在这样的位置，则为 $p_{k+1} = n + 1$）。然后

$$\phi(a) = ((A_{g_{p_1}}, p_2 - p_1), (A_{g_{p_2}}, p_3 - p_2), \ldots, (A_{g_{p_m}}, p_{m+1} - p_m)).$$

注意， $A_{g_{p_j}}$ 表示对应于组索引 g_{p_j} 的子集，而 $p_{j+1} - p_j$ 是游程的长度。

• 映射 ψ：给定一个分区级 GRL 向量

$$v = ((A_{i_1}, s_1), (A_{i_2}, s_2), \ldots, (A_{i_m}, s_m)),$$

我们通过展开每个游程来重构一个排列：

$$\psi(v) = [\underbrace{i_1, \ldots, i_1}_{s_1 \text{ times}}, \underbrace{i_2, \ldots, i_2}_{s_2 \text{ times}}, \ldots, \underbrace{i_m, \ldots, i_m}_{s_m \text{ times}}].$$

这里，GRL 向量中的每个 (A_{i_j}, s_j) 都对应一个由 s_j 个连续位置组成、且标记为组索引 $i_j \in \{1, 2, \cdots, r\}$ 的游程。

我们通过证明 $\psi \circ \phi = \mathrm{id}$ 和 $\phi \circ \psi = \mathrm{id}$，分别验证 ϕ 和 ψ 是双射且互为逆映射。设 $a = [g_1, \ldots, g_n]$ 是一个排列。$\phi(a)$ 标识出所有极大连续段。映射ψ 恰好重构这些游程，从而还原原始序列 a。因此，我们有 $\psi \circ \phi = \mathrm{id}$。现在，设 $v = ((A_{i_1}, s_1) \ldots, (A_{i_m}, s_m))$ 是一个分区级 GRL 向量。映射 ψ 创建一个排列，其中 s_1 个连续位置标记为 i_1，接着是 s_2 个连续位置标记为i_2，依此类推。使用 ϕ 将此排列压缩后，我们会识别出恰好相同的 m 个游程，从而还原原始向量 v。因此， $\phi \circ \psi = \mathrm{id}$。

因此， ϕ 和 ψ 在排列与分区级 GRL 向量之间建立了双射。现在，我们通过计算相应排列集合的基数来证明每一部分。

第（i）部分的证明：长度为 n、涉及 r 组的所有排列构成的集合，其基数为 r^n，因为 n 个位置中的每一个都可以独立地从 $\{1, 2, \ldots, r\}$个组索引中选择一个，共有 r 种选择。因此，根据双射 ϕ，所有长度为 n 的分区级 GRL 向量构成的集合具有相同的基数 r^n。

第（ii）部分的证明：令 $\mathcal{A}_{\text{surj}}$ 表示所有 r 组均至少出现一次的排列所组成的集合，并令 $\mathcal{G}_{\text{surj}}$ 表示所有 r 子集均出现的分区级 GRL 向量所组成的集合。我们证明，双射 ϕ 限制为一个双射 $\mathcal{A}_{\text{surj}} \to \mathcal{G}_{\text{surj}}$。如果排列 a 使用了全部 r 个组索引，那么 $\phi(a)$至少包含来自每个子集的一个游程。反之，如果 GRL 向量 v 包含来自全部 r 子集的游程，那么 $\psi(v)$使用了全部 r 个组索引。

我们使用容斥原理来计数 $|\mathcal{A}_{\text{surj}}|$。对于 $I \subseteq \{1, 2, \ldots, r\}$，令 B_I为不使用任何用于 $i \in I$ 的组索引 i 的排列集合。于是 $|B_I| = (r - |I|)^n$，因为我们只能使用不在 I中的 $r - |I|$ 个索引。

根据容斥原理:

$$\begin{aligned}|\mathcal{A}_{\text{surj}}| &= r^n - \sum_{\emptyset \neq I \subseteq \{1,\dots,r\}} (-1)^{|I|+1} |B_I| \\ &= r^n + \sum_{\emptyset \neq I \subseteq \{1,\dots,r\}} (-1)^{|I|} (r - |I|)^n \\ &= \sum_{i=0}^{r} (-1)^i \binom{r}{i} (r-i)^n \\ &= \sum_{i=0}^{r-1} (-1)^i \binom{r}{i} (r-i)^n,\end{aligned}$$

其中，最后一个等式使用了 $(r-r)^n = 0$，适用于 $n \geq 1$。根据第二类斯特林数与满射函数数量之间众所周知的恒等关系，此表达式等于 $r! \cdot S(n, r)$。

命题 (iii) 的证明：令 $\mathcal{A}_{\text{bal}}$ 表示这样一组排列：恰有 s 个组索引出现 $q+1$ 次，其余 $r-s$ 个索引出现 q 次；令 $\mathcal{G}_{\text{bal}}$ 表示具有相同计数分布的平衡分区级 GRL 向量集合。由于 ϕ 和 ψ 映射保持每个组索引的总计数，双射 ϕ 限制后仍是双射 $\mathcal{A}_{\text{bal}} \to \mathcal{G}_{\text{bal}}$。若排列 a 的计数向量为 $(k_1, \dots, k_r)$（其中 k_i 是标记为 i 的位置总数），那么 $\phi(a)$ 具有相同的计数向量，因为映射连续游程不会改变每个索引的总计数。同样，ψ 也保持计数向量。

为计算 $|\mathcal{A}_{\text{bal}}|$，我们首先从 $\{1, \dots, r\}$ 个组索引中选出 s 个，使其出现 $q+1$ 次。这可以通过 $\binom{r}{s}$ 种方式完成。然后，对于一个固定的选择，例如索引 $1, \dots, s$ 出现 $q+1$ 次，而索引 $s+1, \dots, r$ 出现 q 次，我们计算具有这种分布的 n 个位置的排列数。于是，这类排列的数量就是多项式系数

$$\frac{n!}{(q+1)!^s \cdot q!^{r-s}}.$$

将这些相乘可得

$$|\mathcal{A}_{\text{bal}}| = \binom{r}{s} \cdot \frac{n!}{(q+1)!^s \cdot q!^{r-s}}.$$

当 $s = 0$ 时，我们有 $(q+1)!^0 = 1$，并且所有索引恰好出现 $q = n/r$ 次，因此基数为 $n!\,(q!)^{-r}$。 □

注 5. 定理 *4.1* 中的双射提供了一种组合解释：每个划分级 *GRL* 向量都对应于一种将 n 个位置分配给 r 个组的唯一方式，其中组 i 对应于该划分的子集 A_i。

5 从示例到形式化

在前面的章节中，我们通过阿拉伯语和英语字母表中的具体示例，展示了一些划分程序。现在，我们将对其背后的问题进行形式化描述。

我们的目标是将核心要素——字母表、划分规则、编码子字母表之间交替关系的序列，以及具有语法结构的词汇表——抽象为一个统一框架。这样，我们便能针对符合语言和组合约束的有效句子的存在性、构造和枚举提出精确的问题。形式上，问题的设置由以下组成部分构成。

一个游程约束句子实例由以下部分组成：

- 一个字母表 Σ，其划分函数 Π 为： $\Sigma \to \{A, B\}$ 将符号划分为两类$A = \Pi^{-1}(A)$和 $B = \Pi^{-1}(B)$。
- 一个长度模式$(k_1,\ k_2,\ \ldots\ldots,\ k_m)$，用于指定所需的块大小。
- 一个有限的词典 $D \subseteq \Sigma^*$，其中包含有意义的单词； Σ^* 表示字母表 Σ上所有有限字符串（包括空字符串 ε）的集合。
- 一个定义在 D 上的文法 G，用于规定哪些词序列符合语法。

基于这一设定，我们要探讨的一个基本问题可以表述如下。

问题 1（主要问题：寻找有效序列和句子）。给定一个受游程约束的句子实例（$\Sigma,\ \Pi,\ (k_1, \ldots\ldots,\ k_m)\ D,\ G$），求：

1. 一个交替序列 $t =$（$(C_1,\ k_1)$，$(C_2,\ k_2)\ldots\ldots$，$(C_m,\ k_m)$），其中每个 C_i 都是一个多重集，且具有 $|C_i| = k_i$，并且满足以下条件之一

$$\{C_{2i}\} \subseteq A \textit{ and } \{C_{2i-1}\} \subseteq B, \quad \textit{or} \quad \{C_{2i-1}\} \subseteq A \textit{ and } \{C_{2i}\} \subseteq B,$$

其中 $\{C_i\}$ 表示多重集 C_i 中不同符号组成的集合，适用于 $1 \le i \le m$。

2. 令 D^* 为词典 D中所有有限词序列组成的集合。句子 $w \in D^*$满足：

- w 相对于文法 G而言是合乎语法的，并且
- w 的拼接字母（将每个词拼写出来后得到）满足由 t定义的块模式。

如果不存在这样的二元组(t, w)，则返回空值。

我们对这个问题的结构作出两点观察。

注 6。 问题 *1* 可以看作包含两个部分：

1. 序列构造：找出一个满足交替约束的交替序列 t 。 *2.* 句子生成：对于给定的交替序列 t，找出一个字母模式与 L_t匹配的合乎语法的句子 w。

注 7。 在以下小节中，我们使用形式语言理论将问题 *1* 第 *2* 部分中的条件形式化：条件（*1*）变为 $w \in L(G)$，其中 $L(G)$表示由文法 G生成的语言；条件（*2*）变为 $h(w) \in L_t$，其中 $h: D^* \to \Sigma^*$ 是一个将词序列映射为字母序列的拼写同态，而 L_t 是由序列 t定义的块语言。

5.1 形式语言理论背景

在本节中，我们回顾形式语言理论的一些基本概念。 [2]中的以下定义是自动机理论中的标准定义。

[[2], 确定有限自动机]一个确定有限自动机（*DFA*）是一个 5 元组 $M = (Q, \Sigma, \delta, q_0, F)$，其中：

- Q 是一个有限状态集,
- Σ 是一个有限字母表,
- δ： $Q \times \Sigma \to Q$ 是转移函数,
- $q_0 \in Q$ 是初始状态,
- $F \subseteq Q$ 是接受（终结）状态的集合。

如果由 $q_{i+1} = \delta(q_i, w_{i+1})$ 定义的状态序列 $q_0, q_1, \ldots\ldots, q_n$ 满足 $q_n \in F$，则确定有限自动机 M 接受字符串 $w = w_1 w_2 \cdots w_n \in \Sigma^*$。
我们将确定有限自动机所识别的语言定义如下。
[[2], 正则语言] 设 $M = (Q, \Sigma, \delta, q_0, F)$ 是一个确定有限自动机（DFA）。 M 所识别的语言为

$$L(M) = \{ w \in \Sigma^* \mid M \text{ accepts } w \},$$

其中， $\delta(q_0, w)$ 表示 M 从初始状态 q_0出发处理字符串 w 后到达的状态。若存在一个确定有限自动机 M，使得 $L = L(M)$，则称语言 $L \subseteq \Sigma^*$ 为 正则。
在介绍了有限自动机的基础知识后，我们现在转向表达能力更强的语法形式体系。下面的定义将上下文无关文法的标准概念推广为处理词语词典，而非单个符号。

[[2], 字典上的上下文无关文法] 一个上下文无关文法 $G = (V, D, P, S)$定义在词典 $D \subseteq \Sigma^*$ 上，由以下部分组成：

- V是一个有限的非终结符符号集合,
- D 是词典（终结符是单词，每个单词 $w \in D$ 都是由 Σ构成的字符串），
- P是一个有限的产生式规则集合 $A \to \alpha$，其中 $A \in V$且 $\alpha \in (V \cup D)^*$,
- $S \in V$是开始符号。

语言 $L(G) \subseteq D^*$ 是可由 S推导出的词序列集合。
现在，我们回顾由上下文无关文法生成的语言类的定义。
[[2], 上下文无关语言] 如果存在一个上下文无关文法 G，使得 $L = L(G)$，其中 $L(G)$ 是 G 的语言，那么称语言 L 为 上下文无关语言（*CFL*）。

5.2 块语言及其性质

根据我们对主要问题的表述，我们将块语言定义如下。[块语言] 对于序列 $t = ((C_1, k_1)...$ $..., (C_m, k_m))$，令 $\Sigma_{C_i} = \{C_i\}$ 表示多重集 C_i 中不同符号构成的集合。块语言定义为

$$L_t = (\Sigma_{C_1})^{k_1}(\Sigma_{C_2})^{k_2} \cdots (\Sigma_{C_m})^{k_m} \subseteq \Sigma^K, \quad K = \sum_{i=1}^{m} k_i.$$

其中，语言 $(\Sigma_{C_i})^{k_i}$ 由所有长度为 k_i、使用来自 Σ_{C_i} 的符号的字符串组成；Σ_{C_i} 是一个至多包含 $|\Sigma_{C_i}|^{k_i}$ 个字符串的有限集合。

定义了块语言后，我们现在来确立其关键的计算性质。

定理 5.1。对于上文定义的每个序列 $t = \big((C_1, k_1), (C_2, k_2) .. , (C_m, k_m)\big)$，语言 L_t 是正则的。

证明。对于每个 $i \in \{1, 2, \ldots, m\}$，集合 Σ_{C_i} 是字母表的有限子集；根据定义 5.1，每个有限语言都是正则的。块语言 $L_t = (\Sigma_{C_1})^{k_1}(\Sigma_{C_2})^{k_2} \cdots (\Sigma_{C_m})^{k_m}$ 是 m 个正则语言的连接。由于正则语言类在有限连接下封闭，因此 L_t 是正则的。 □

由于每种正则语言都是上下文无关语言，我们立即得到以下结果。

推论 5.2. 对于上述定义的每个序列 $t = \big((C_1, k_1), (C_2, k_2)...(C_m, k_m)\big)$， L_t 都是上下文无关语言（*CFL*）。

证明。根据 [2,练习 5.1.3]， L_t 是上下文无关语言，因为每种正则语言都是上下文无关语言。 □

现在，让我们研究本节主要问题的可判定性。

我们的问题涉及两个层面：

1. 字母层面：每个词 $w \in D$ 都是字母表 Σ上的字符串。块约束 L_t作用于这一层面，规定了拼接后的字母的交替模式。
2. 单词层面：文法 G 以词为原子单位，在 D^* 中生成序列。

假设 $h : D^* \to \Sigma^*$ 是一个同态（关于连接运算），通过将每个词序列映射为拼接后的字母序列来连接这两个层面。那么，我们的问题是：

是否存在一个词序列 $w \in D^*$，使得

- $w \in L(G)$（符合文法的词序列），并且
- $h(w) \in L_t$（拼接后的字母满足交替模式）？

等价地，是 $L(G) \cap h^{-1}(L_t) \neq \emptyset$，还是 $h(L(G)) \cap L_t \neq \emptyset$?

引理 5.3（上下文无关语言的同态像）。设 $L(G) \subseteq D^*$ 是一个上下文无关语言，并设$h: D^* \to \Sigma^*$ 是将每个词映射为其拼写（其字母的连接）的同态。那么 $h(L(G))$ 是上下文无关的。

证明。上下文无关语言在同态下保持不变 [2, 定理 7.24]。由于 h 是字符串同态，且 $L(G)$是上下文无关语言，因此 $h(L(G))$是上下文无关语言。 □

现在，我们将块语言的结构性质与同态引理结合起来，以确立主要的可判定性结果。

定理 5.4 (存在性问题的可判定性)。存在性问题（问题 *1*）可在多项式时间内判定。

证明。根据定理 5.1，L_t 是正则语言。根据引理 5.3，$h(L(G))$是上下文无关语言，其中 $h: D^* \to \Sigma^*$ 将每个词映射为其拼写形式。存在性问题询问$h(L(G)) \cap L_t \neq \emptyset$是否成立。由于上下文无关语言与正则语言的交集仍是上下文无关语言 [2, 定理 7.27]，$h(L(G)) \cap L_t$ 是上下文无关语言。上下文无关语言的空性问题可在线性时间内判定，所需时间取决于文法的大小 [2, 第 7.4 节3]。因此，存在性问题可在多项式时间内判定。 □

5.3 词级自动机构造

为证明块语言是正则的，并建立我们的可判定性结果，我们构造显式自动机来识别满足块约束的序列。我们首先构造一个在字母表 Σ 上运行的字母级确定有限自动机。

[块语言的确定有限自动机] 对于一个序列 $t = ((C_1, k_1), \ldots, (C_m, k_m))$，我们构造一个确定有限自动机$M_t = (Q, \Sigma, \delta, q_0, F)$，其识别方式如下：

- $Q = \{q_0, q_1, \ldots, q_K\}$ 其中 $K = \sum_{i=1}^{m} k_i$（状态表示从 0 到 K 的位置），
- 初始状态为 q_0（尚未读取任何字母），
- 接受状态 $F = \{q_K\}$（恰好读取了 K 个字母），
- 转移函数 δ 定义如下：对于状态 q_j，其中 $0 \leq j < K$，以及符号 $\sigma \in \Sigma$:

$$\delta(q_j, \sigma) = \begin{cases} q_{j+1} & \text{if } \sigma \in \Sigma_{C_i} \text{ where } i \text{ is such that } \sum_{\ell=1}^{i-1} k_\ell < j+1 \leq \sum_{\ell=1}^{i} k_\ell \\ \text{reject} & \text{otherwise} \end{cases}$$

  换句话说，从位置 j出发，当且仅当位置 $j+1$ 位于满足 $\sigma \in \Sigma_{C_i}$的块 i 中时，我们才能读取符号 σ 并前进到位置 $j+1$。

- 对于接受状态 q_K，我们定义 $\delta(q_K, \sigma)$ =对所有 $\sigma \in \Sigma$均拒绝。

虽然 M_t 处理的是单个字母，但我们的主要问题涉及词典 D 上的上下文无关文法，其中包含多个词。为弥合这一差距，我们将字母级确定有限自动机提升为词级自动机，以跟踪哪些词典单词与块约束相容。

[词级自动机] 给定一个识别块语言 L_t的确定有限自动机 $M_t = (Q, \Sigma, \delta_t, q_0, F)$和一个词典 $D \subseteq \Sigma^*$，词级自动机 是一个有向图 $\mathcal{W} = (Q, D, E)$，其中：

- Q 是来自 M_t 的状态集合（表示字母序列中的位置），
- D 是词典（边标签），
- $E \subseteq Q \times D \times Q$ 是定义如下的边集合：

$$(q, w, q') \in E \quad \text{if and only if} \quad \delta_t^*(q, w) = q'$$

其中 δ_t^*： $Q \times \Sigma^* \to Q$ 是扩展转移函数，它读取字符串（即，从 q 状态开始，依次读取词 w 中的字母，会在 M_t 中到达状态 q'）。

词级自动机 $\mathcal{W}$ 与字母级确定有限自动机 M_t具有相同的状态，但其转移标签是完整的词，而非单个字母。边（q， w， q'） $\in E$ 存在，当且仅当：

- 从 M_t 中的状态 q 开始，
- 逐个读取词 $w = w_1 w_2 \cdots w_\ell$ 中的字母，
- 我们成功到达 M_t中的状态 q' 。

注释 8. 词级自动机 $\mathcal{W}$ 有效地将字母级确定有限自动机 M_t “提升”为以词作为原子单位运行。从 q_0 到 $\mathcal{W}$ 中某个接受状态的一条路径，对应于一个词序列，其拼接后的字母构成 L_t中的一个字符串。

5.4 寻找有效句子的算法

我们现在介绍可判定性结果的算法实现，该实现分为两个阶段：首先，检查给定序列是否存在；然后，在所有可能的序列中进行搜索。

算法 1：判断给定序列是否存在要求：（输入）序列 $t = ((C_1, k_1) \ldots, (C_m, k_m))$，词典 D，文法 G确保：（输出）若存在，则返回有效句子 w；否则返回空值1：构造确定有限自动机 $M_t = (Q, \Sigma, \delta_t, q_0, F)$，用于识别 L_t，其状态数为 $K + 1$，其中 $K = \sum_{i=1}^{m} k_i$2：初始化词级自动机 $\mathcal{W} = (Q, D, \emptyset)$，边集合为空3：对于每个词 $w \in D$，执行4：　对于每个状态 $q \in Q$，执行5：　　从状态 q 开始，在 M_t 中模拟读取 w 的字母6：　　如果模拟到达状态 q' 且未被拒绝，则7：　　将边 (q, w, q') 添加到 $\mathcal{W}$ 中的 E 8：　　结束如果9：　结束循环10：结束循环11：在 $\mathcal{W}$ 上应用上下文无关文法解析算法（例如 CKY [7]），以查找从 q_0 到 F 中接受状态的符合文法的路径12：如果符合文法的路径 p 存在，则13：　从路径 p 中提取句子 w（边上词标签构成的序列）14：　返回 w 15：否则16：　返回空值17：结束如果

以下定理确立了算法 1 的复杂度。

定理 5.5。 算法 *1* 的运行时间为 $O(|D| \cdot |t| + |t|^3 \cdot |G|)$，其中 $|t| = \sum_{i=1}^{m} k_i$ 为模式总长度，假设字母表大小 $|\Sigma|$为常数。

证明。 令 $K = \sum_{i=1}^{m} k_i$ 表示模式总长度。我们分析算法各个组成部分的运行时间。构造确定有限自动机 M_t 需要创建 $K+1$ 个状态，其转移根据定义 5.3 确定。对于 K 个非接受状态中的每一个状态，以及 Σ中的每个符号，我们都必须确定相应的转移，这需要 $O(K \cdot |\Sigma|)$ 时间。构建词级自动机时，需要从每个状态 $q \in Q$测试每个单词 $w \in D$。对于长度为 ℓ_w的单词 w，从给定状态模拟读取该单词需要 $O(\ell_w)$ 时间。所有单词和状态所需的总时间为

$$O\left(\sum_{w \in D} |Q| \cdot \ell_w\right) = O(K \cdot \sum_{w \in D} \ell_w) = O(K \cdot |D|),$$

其中 $|D| = \sum_{w \in D} \ell_w$ 表示词典中所有单词的字母总数。在具有 $K+1$ 个位置的词级自动机上应用 CKY 解析算法需要 $O(K^3 \cdot |G|)$ 时间，其中 $|G|$ 为文法的大小。如果存在有效句子，从解析结构中提取该句子最多还需要 $O(K)$ 时间。将各项运行时间相加，并将 $|\Sigma|$ 视为常数，可得 $O(K \cdot |\Sigma| + K \cdot |D| + K^3 \cdot |G|) = O(|D| \cdot |t| + |t|^3 \cdot |G|)$，其中取$K = O(|t|)$。

□

要彻底解决问题 1，我们必须同时找到一个有效序列 t 和与之对应的句子。以下算法将序列枚举与检查过程结合起来。

算法 2 寻找有效交替序列和句子

输入： 字母表划分 (A, B); 长度模式 $(k_1, \ldots, k_m)$; 词典 D; 文法 G
输出： 一个对 (t, w)，其中 t 是有效序列，w 是有效句子；若不存在，则返回 **空值**

对于 $\alpha \in \{(A, B, A, \ldots), (B, A, B, \ldots)\}$ **执行** ▷ 尝试两种交替模式
  对于每个有效的多重集选择 $(C_1, \ldots, C_m)$ **执行**
    其中 C_i 有 $|C_i| = k_i$ 和 $\{C_i\} \subseteq \alpha_i$
    构造 $t \leftarrow ((C_1, k_1) \ldots, (C_m, k_m))$
    $w \leftarrow$ 算法 $1(t, D, G)$
    如果 $w \neq$ **为空值，则**
      返回 (t, w) ▷ 找到有效序列和句子
    结束 if
  结束 for
结束 for
返回 空值 ▷ 不存在有效序列

以下定理确定了算法 2 的复杂度。

定理 5.6。 令 $|A| = a$ 和 $|B| = b$ 表示各划分类的大小。算法 *2* 的运行时间为

$$O\left(N \cdot (|D| \cdot |t| + |t|^3 \cdot |G|)\right)$$

其中 N 是可能的序列 t、 $|\alpha_i| \in \{a, b\}$和 $N \leq 2 \cdot \prod_{i=1}^{m} \binom{|\alpha_i| + k_i - 1}{k_i}$ 的数量，

其在 k_i 和 $|\Sigma|$ 中可能呈指数级增长。

证明。 我们首先确定算法 2 必须检查的序列 t 数量，然后分析每个序列的检查成本。

该算法考虑两种交替模式：以划分类 A 开始并与 B 交替的序列，以及以 B 开始并与 A 交替的序列。对于每种交替模式，我们必须枚举满足约束的所有可能多重集 $(C_1, \ldots\ldots, C_m)$ 选择。对于给定的交替模式，每个块 i 必须使用其指定划分类 $\alpha_i \in \{A, B\}$ 中的符号，其中 $|\alpha_i| \in \{a, b\}$。块 i 需要一个大小为 k_i 的多重集 C_i，其元素取自大小为 $|\alpha_i|$ 的字母表。此类多重集的数量等于从 $|\alpha_i|$ 个不同符号中可重复地选择 k_i 个元素的方式数。等价地，它等于方程 $x_1 + x_2 + \cdots + x_{k_i} = |\alpha_i|$. 的非负整数解数量。众所周知，该数量由二项式系数 $\left(\!\!\binom{|\alpha_i|}{k_i}\!\!\right) = \binom{|\alpha_i| + k_i - 1}{k_i}$ 给出（获得这一恒等式的一种方法是使用“隔板法”组合论证）。由于每个块的多重集选择彼此独立，对于固定的交替模式，多重集序列总数为乘积 $\prod_{i=1}^{m} \binom{|\alpha_i| + k_i - 1}{k_i}$。计入两种交替模式后，需要检查的序列总数为 $N = 2 \cdot \prod_{i=1}^{m} \binom{|\alpha_i| + k_i - 1}{k_i}$。对于以这种方式构造的每个序列 t，算法都会调用算法 1，以检验是否存在有效的合乎语法的句子，并在存在时构造该句子。根据定理 5.5，每次此类调用需要 $O(|D| \cdot |t| + |t|^3 \cdot |G|)$ 时间。在最坏情况下，算法必须在找到有效句子（或确定不存在有效句子）之前检查全部 N 个序列。因此，总运行时间为 $O(N \cdot (|D| \cdot |t| + |t|^3 \cdot |G|))$。注意， N 会随着块大小 k_i 以及划分类大小 a 和 b 呈指数级增长。具体而言，当 k_i 较大或 $|\alpha_i|$ 较大时，二项式系数 $\binom{|\alpha_i| + k_i - 1}{k_i}$ 会变得非常大，从而使 N 在最坏情况下呈指数级增长。 □

我们强调几项可用于优化算法的观察。

注 9. 以下几种优化思路可以缩小搜索空间：

1. 字典引导的剪枝：仅考虑其符号确实出现在 D 中单词里的多重集 C_i。 *2.* 提前终止：一旦找到任意有效序列，就立即停止。 *3.* 增量构造：逐步构造格，并剪除无法生成有效句子的序列。

现在，我们正式证明算法 2 对于问题 1 的正确性。

定理 5.7。 算法 *2* 能正确求解问题 *1*。若存在有效对 (t, w)，算法将返回这样一对。如果不存在有效对，算法将返回空值。

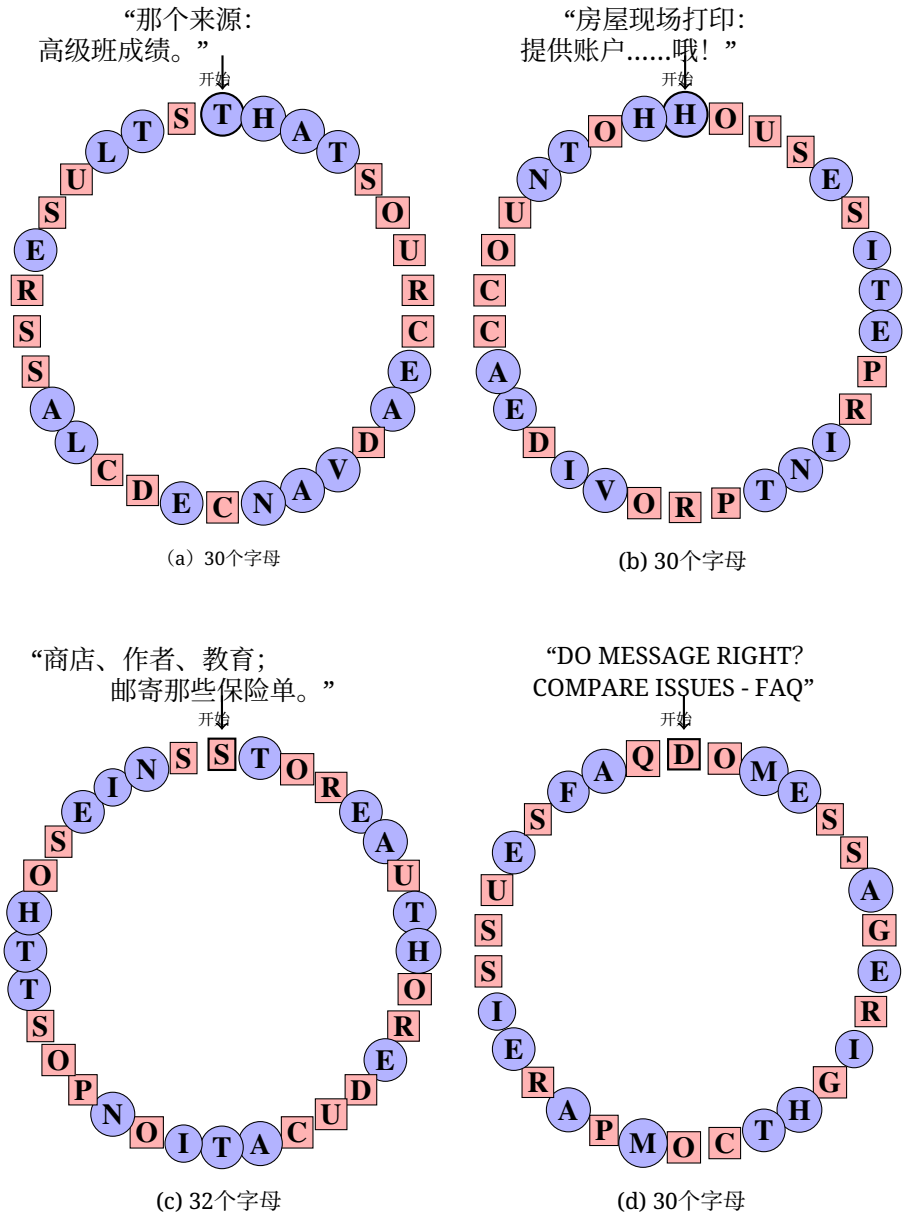
证明。 算法 2 通过枚举两种交替模式以及所有有效的多重集选择，穷举搜索所有可能的交替序列 t。对于每个候选序列 t，算法调用算法 1，以确定是否存在与该序列对应的合乎语法的句子。根据定理 5.1，每个 L_t 都是正则的。根据引理 5.3 和定理 5.4 的证明，算法 1 能正确判定 $h(L(G)) \cap L_t \neq \emptyset$，若存在，则返回见证句 $w \in h(L(G)) \cap L_t$。由于算法 2 会检查所有可能的序列，因此如果存在有效对 (t, w)，它就会找到该对。如果算法返回空值，这意味着不存在能构成有效合乎语法句子的交替序列，因此问题 1 无解。

□

为了验证我们的算法，我们使用 Python 实现了该算法，并将其应用于各种测试用例。完整实现可在我们的 GitHub仓库[6]中找到。其中一个示例是句子“THAT SOURCE: ADVANCED CLASS RESULTS.”，该句子由算法根据序列（4, 5, 2, 1, 3, 1, 1, 2, 2, 3, 1, 2, 2, 1）生成，该序列即历史示例中的序列。该算法还会根据序列（1, 3, 1, 1, 3, 2, 3, 3, 2, 1, 2, 4, 2, 1, 1）生成句子“HOUSE SITEPRINT: PROVIDE ACCOUNT... OH!”，根据序列（1, 1, 2, 2, 1, 2, 2, 1, 3, 3, 1, 1, 3, 3, 2, 3, 1）生成句子“STORE, AUTHOR EDUCA- TION; POST THOSE INS.”，并根据序列（2, 2, 2, 1, 1, 1, 1, 1, 1, 2, 2, 1, 1, 1, 1, 2, 3, 1, 1, 2, 1）生成句子“DO MESSAGE RIGHT? COMPARE ISSUES - FAQ”。

图4展示了这些句子的字母如何排列在圆周上，其中直线字母以蓝色标示，曲线字母以红色标示。

[6]github.com/omidkhormali/Historic_英语_句子_code



（a）30个字母

(b) 30个字母

(c) 32个字母

(d) 30个字母

图4：四个英语句子排列在圆形周围。(a) “那个来源：高级班成绩。”，包含30个字母，遵循历史示例序列。(b) “HOUSE SITEPRINT: PROVIDE ACCOUNT... OH!”，包含30个字母。(c) “商店，作者 教育；POST THOSE INS.”，包含32个字母。(d) “DO MESSAGE RIGHT? COMPARE ISSUES - FAQ”，包含30个字母。直线字母显示在圆形内；曲线字母显示在矩形内。

这些示例表明，我们的算法可以利用知名历史示例中的模式生成另一种语言的句子。

注10. 运行 *Python* 程序，使用包含英语中最常见的 *2,000* 个单词的词典来实现这些算法。对于历史序列（*30*个字母，图*4 (a)*），算法找到了一个满足所有约束的完整句子。对于另外三个序列（分别为*30*、*32*和*30*个字母，见图*4 (b)*、*(c)*和*(d)*），算法生成的句子长度为

分别为*28*、*29*和*27*个字母，随后通过手动添加较短的连接词（每个*2*至*3*个字母）将句子补充完整。此外，还手动添加了标点符号（逗号、冒号和省略号），以提高可读性并使句子更加连贯。请注意，这些序列仅取决于字母，而不取决于标点符号，因此这些补充不会影响解的有效性。这些结果表明，即使词典有限，该算法仍然有效。使用更全面的词汇表，很可能无需手动补全就能为所有序列找到更好的解。

虽然前面的示例展示了算法在历史模式上的输出，但现在我们将通过一个更简单的实例，完整演示算法 1，以详细说明计算步骤。

示例 5。 为了详细说明算法的运行过程，我们将通过一个小型实例，完整追踪一次执行过程。

输入规范。我们考虑将字母表 $\Sigma = \{a, b, c, d\}$ 划分为 $A = \{a, b\}$ 和 $B = \{c, d\}$。块序列为 $t = ((C_1, 2), (C_2, 1), (C_3, 2))$ 其中 $C_1 = \{a, a\}$ 为 $\Sigma_{C_1} = \{a\}$，$C_2 = \{c\}$ 为 $\Sigma_{C_2} = \{c\}$，且 $C_3 = \{a, b\}$ 为 $\Sigma_{C_3} = \{a, b\}$。由此得到模式长度总计为 $K = 5$。因此，块语言为

$$L_t = \{a\}^2 \cdot \{c\}^1 \cdot \{a, b\}^2 = \{aacaa, aacab, aacba, aacbb\}.$$

词典为 $D = \{$*“aa”*、*“ac”*、*“ab”*、*“c”*、*“ca”*$\}$，文法 G 的开始符号为 S，其产生式规则如下：

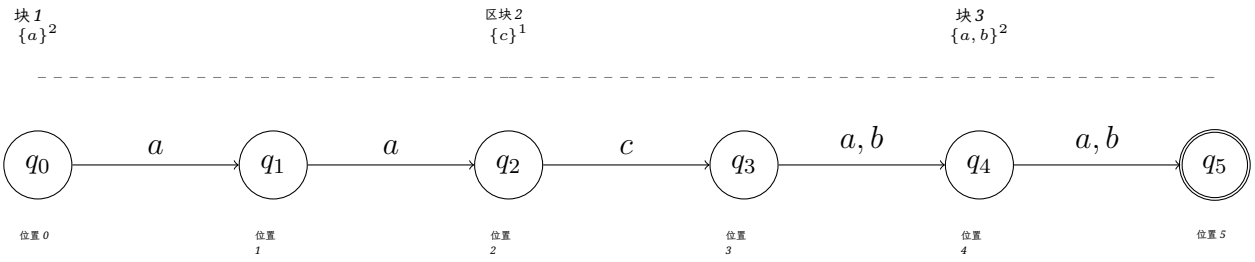
$$S \to W\,W\,W, \quad W \to \text{“}aa\text{”} \mid \text{“}ac\text{”} \mid \text{“}ab\text{”} \mid \text{“}c\text{”} \mid \text{“}ca\text{”}$$

该文法接受由 D 中恰好三个单词组成的任意序列。

构造字母级确定有限自动机。根据定义 *5.3*，我们构造 $M_t = (Q, \Sigma, \delta_t, q_0, F)$，其状态 $Q = \{q_0, q_1, q_2, q_3, q_4, q_5\}$ 表示位置 0 到 5，初始状态为 q_0，接受状态为 $F = \{q_5\}$。转移函数 δ_t 根据块约束定义如下：

状态	块	$\delta_t(\cdot,\ a)$	$\delta_t(\cdot,\ b)$	$\delta_t(\cdot,\ c)$
q_0	块 *1*（下一个位置）	q_1	拒绝	拒绝
q_1	块 *1*	q_2	拒绝	拒绝
q_2	区块 *2*（下一个位置）	拒绝	拒绝	q_3
q_3	区块 *3*（下一个位置）	q_4	q_4	拒绝
q_4	区块 *3*	q_5	q_5	拒绝
q_5	（接受）	拒绝	拒绝	拒绝

确定有限自动机图如下。

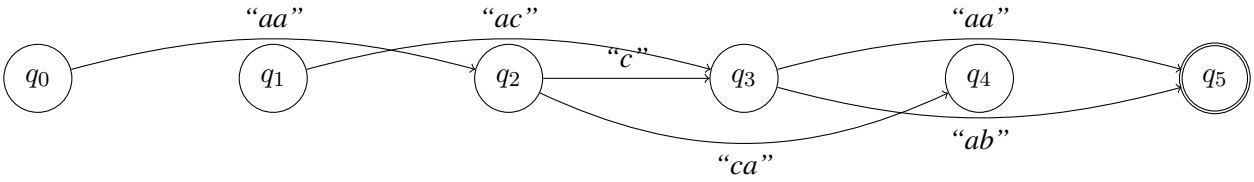


从状态 q_i （位置 i）出发，当且仅当位置 $i+1$位于某个约束集包含 σ的块中时，我们才能读取符号 σ 并转移到 q_{i+1}。例如，从 q_0出发，只有符号 a 有效，因为位置 *1* 属于约束集为 $\{a\}$ 的块 *1*。同样，从 q_3 出发， a 和 b 都有效，因为位置 *4* 属于约束集为 $\{a,b\}$ 的区块 *3*。

构造词级自动机。对于每个词 $w \in D$，我们通过模拟从每个状态 q读取 w 中的字母，确定词级自动机 $\mathcal{W}$ 中的边 (q, w, q') M_t。结果汇总如下：

Word	起始状态	目标状态	字母序列
“aa”	q_0	q_2	a, a
“aa”	q_3	q_5	a, a
“ac”	q_1	q_3	a, c
“ab”	q_3	q_5	a, b
“c”	q_2	q_3	c
“ca”	q_2	q_4	c, a

词级自动机图如下。



例如，从 q_0 开始的词*“aa”*会产生转换 $q_0 \xrightarrow{a} q_1 \xrightarrow{a} q_2$，从而创建边$(q_0$, *“aa”* $q_2)$。然而，从 q_1 处理*“aa”*会失败，因为 q_2 没有外出 a 转换。完整的词级自动机仅包含能够成功处理整个词的边。

寻找符合文法的路径。我们要寻找从 q_0 到 q_5 的路径，并且恰好使用三个词（这是文法 G 的要求）。通过检查词级自动机的结构，我们找到了两条有效路径：路径 1： $q_0 \xrightarrow{\text{“aa”}} q_2 \xrightarrow{\text{“c”}} q_3 \xrightarrow{\text{“aa”}} q_5$，得到句子*“aa c aa”*，其字母序列为*“aacaa”*。路径 2： $q_0 \xrightarrow{\text{“aa”}} q_2 \xrightarrow{\text{“c”}} q_3 \xrightarrow{\text{“ab”}} q_5$，得到句子*“aa c ab”*，其字母序列为*“aacab”*。这两个句子都符合文法（每个词都属于 D，且序列与产生式 $S \Rightarrow W\,W\,W$相匹配），并且这两个字母序列都属于 L_t。算法会返回其中一个解，例如*“aa c aa”*。

因此，此例有两个有效解：*“aa c aa”*和*“aa c ab”*。算法会返回其中一个。

6 结论

我们已经证明了块约束句子存在性问题的可判定性，并为其求解提供了多项式时间算法。我们结合自动机

理论与上下文无关解析的构造性方法表明，受约束文本生成问题可以通过形式语言理论加以解决。这项工作连接了历史上的组合谜题、形式语言理论和算法设计。通过广义游程编码将圆形划分与语法约束联系起来，我们展示了如何运用经典的自动机理论技术解决组合学与计算语言学交集处的问题。因此，古典阿拉伯诗歌及类似应用中出现的这类结构化模式约束，在计算上是可处理的。

致谢

前两位作者谨此感谢埃文斯维尔大学创新与变革中心颁发的 Global Scholar Award 所提供的支持。2025年十一月和十二月期间，作者偶尔使用 AI 语言模型（ChatGPT-5.2 Instant 和 Claude Sonnet 4.5）润色稿件、借助 TikZ 协助生成图表，并处理阿拉伯字体。作者还利用这些模型协作开发和调试 Python 程序，以进行计算验证。所有智力与科学贡献均为作者原创。

参考文献

[1]H. AL-AṬṬĀR, *Hāshiyat al-Aṭṭār alá Jam al-Jawāmi*, Dār al-Kutub al-Ilmiyya, 贝鲁特, 黎巴嫩, 1835年。含 Abd al-Rahmān al-Shirbīnī 的眉批及 Muḥammad Alī ibn Husayn al-Mālikī 的脚注。[2] J. E. HOPCROFT, R. MOTWANI, AND J. D. ULLMAN, 自动机理论、语言与计算导论, 第2版, Addison-Wesley, 马萨诸塞州雷丁, 2001年。[3] J. R. NEWMAN, 数学世界, 第4卷, Tempus, 华盛顿特区, 1988年, 第2403–2405页。[4] K. C. RYDING, 《现代标准阿拉伯语参考语法》, 剑桥大学出版社, 英国剑桥, 2005年。[5] Ṣ. AL-DĪN AL-ṢAFADĪ, *Ayān al-Aṣr wa-Awān al-Naṣr*, A. Abū Zayd, N. Abū Ashmah, M. Mawid, 和 M. Sālim Muḥammad 编, Dār al-Fikr, 大马士革, 1998年。[6] 维基百科贡献者, 约瑟夫斯问题 — 维基百科，自由的百科全书, `https://en.wikipedia.org/wiki/Josephus_problem`, 2025年。访问于2025年6月4日。[7] D. H. YOUNGER, 上下文无关语言的识别与解析，所需时间为 n^3, Inform. Control, 10 (1967年), 第189–208页。