arXiv:2602.10073v1 [cs.CC] 10 Feb 2026

若 P=$P=$，则 $P=$=*PSPACE* 的一些条件

（正在编写的草稿）

Ismael Rodríguez

计算机系统与计算系 信息学院 马德里康普顿斯大学，
28040 马德里，西班牙 电子邮件：isrodrig@sip.ucm.es

最后更新：2013年7月19日

摘要。 我们确定了若干条件 X 使得

$$(P = NP \wedge X) \Rightarrow P = PSPACE$$

关键词： 复杂性， P、 NP、 $PSPACE$。

1 引言

已知 $P \subseteq NP \subseteq PSPACE$，不过目前尚不清楚这两个包含关系是否都为严格包含。在这份草稿中，我们探讨条件

$$P = NP \Rightarrow P = PSPACE$$

是否成立。证明这一性质将使得 $P = NP$不太可能成立，因为在大多数研究者看来，$P = PSPACE$似乎是一个相当强的假设。

我们找出了几个充分条件 X，使得如果 X 成立，那么我们就有$P = NP \Rightarrow P = PSPACE$。我们证明了它们中的每一个都蕴含 $P = NP \Rightarrow P = PSPACE$。我们还讨论了每个 X的可行性，并提出了若干尝试证明它们的方法。遗憾的是，这份草稿并未给出其中任何一个的证明。

本文结构如下。我们的基本陈述在下一章节给出，并在第 3 节讨论第一个所提条件（实际上是一个更弱的版本）的可行性。第 4 节讨论了三种试图证明它的方法。第 5 节提出了另一种条件，第 6 节介绍了第一个所提条件的一个有趣推论。我们在第 7 节给出结论。

2 基本陈述

图灵机由元组 $M = (S, \Sigma, \Gamma, s_1, s_A, \Delta)$ 定义，其中 S是状态集， Σ 是输入符号集，Γ是带符号集（$\Sigma \subseteq \Gamma$），

s_1 是初始状态，s_A 是接受状态，Δ 是转移集（本文仅考虑确定性图灵机）。我们假设 σ：$S \cup \Gamma \to \{0,1\}^*$ 将 S 和 Γ中的每个符号编码为 $\alpha = \lceil log_2(|S \cup \Gamma|) \rceil$ 比特。M 的一个配置——其中 M 处于状态 s，（非空白部分的）纸带为 $c_1 \ldots c_v$，读写头位于第 i 个位置——由比特序列 $\sigma(c_1) \ldots \sigma(c_{i-1})\sigma(s)\sigma(c_i) \ldots \sigma(c_v)$ 表示。因此，M 对于输入 $x = x_1 \ldots x_n$ 的初始配置为 $\sigma(s_1)\sigma(x_1) \ldots \sigma(x_n)$。若 M 能从其初始配置 $\sigma(s_1)\sigma(x_1) \ldots \sigma(x_n)$ 出发，经过 0 步或更多步到达配置 y，则我们记作 $x \Rightarrow_M y$（M不一定在y处终止）。该初始配置将简记为 $\sigma(x)$。

设 Q是 $PSPACE$中的一个判定问题。存在一台确定性图灵机 $M' = (S', \Sigma, \Gamma, s_1, s'_A, \Delta')$，使得 M'在空间$T(n)$内求解 Q，其中 T为某个多项式。由 M'构造一台新的图灵机$M = (S, \Sigma, \Gamma, s_1, s_A, \Delta)$是平凡的，使得 M的行为与 M'相同，不过它在到达 s'_A 之后擦除纸带，并随后停在新的接受状态 s_A。这样，我们保证 M 将具有唯一的接受配置 $\sigma(s_A)$。我们将此接受配置记为 c_A。

让我们以另一种方式计算 $PSPACE$问题 Q。给定输入 x，问题 $x \in Q$? 可以按如下方式判定：

1. 由 M 和 x构造一个能够计算有限布尔函数的计算设备（例如命题逻辑公式、逻辑电路、有限自动机、图灵机等），其行为如下一个布尔函数$f_{M,x}$:

$$f_{M,x}(y) = \begin{cases} 1 & \texttt{if } x \Rightarrow_M y \\ 0 & \text{otherwise} \end{cases}$$

   由于 M 在多项式空间 $T(n)$ 内运行，我们可以仅对适合该大小的配置定义 $f_{M,\,x}$。因此，$f_{M,\,x}$ 的有限定义域的大小为某个 $k \in \mathbb{N}$ 的 $2^{T(n)} \in \mathcal{O}(2^{n^k})$。

2. 如果 $f_{M,\,x}(c_A) = 1$，则回答 是，否则回答 否。

回想一下，n是 $x = x_1 \ldots x_n$的大小。我们引入条件 (a) 和 (b)，稍后将用它们来构造引言中提到的一些充分条件 X。显然，如果

(a) 存在一个多项式 R，使得步骤（1）可以在时间$R(n)$ 内完成；并且 (b) 存在一个多项式 P，使得在所有大小受 $T(n)$ 限制的配置下，(1) 中构造的计算设备都能在时间 $P(n)$ 内运行，

那么我们可以通过执行 (1)，然后执行 (2)，在多项式时间内求解 Q，适用于所有 x。

请注意，条件 (b) 所施加的所构造计算设备的执行时间（即 $P(n)$）是根据 n（x 的大小）定义的，而不是根据该计算设备的输入（一个配置）的大小定义的。然而，我们所关注的配置的大小以 $T(n)$ 为界。

因此，该计算设备的执行时间也（间接地）以其自身输入的大小的某个多项式为界。

还值得指出的是， M 对于 Q是固定的，因此这个求解 Q的过程的复杂性不取决于 M 的大小（因而也不取决于 S、 Σ、 Γ或 Δ的大小）。

我们来考虑以下替代条件：

(c) 存在多项式 D 和 P，使得对于所有 x，存在 一个计算设备计算$f_{M,\ x}$ 其表示大小（例如命题公式的长度、逻辑电路的大小、有限自动机或图灵机的状态数等）为 D（n），并且 对于所有大小不超过 T（n）的配置，该计算设备都能在P（n）时间内运行完毕。

请注意，对于某些总能在输入大小的多项式时间内运行的计算设备（例如命题公式、有限自动机），(c)中的第一项要求（即and之前的部分）蕴含第二项要求。事实上，对于总能在输入大小的多项式时间内运行的计算设备，必定存在一个多项式 P，使得对于任何大小不超过 T（n）的输入，P（n）步计算就足以完成处理。

考虑条件 (a) 和 (b)，与考虑前述条件 (c) 之间的重要区别在于：前一种情况下，计算设备必须在多项式时间内完成计算；而后一种情况下，我们只要求存在一个多项式规模的计算设备。

在第 2.1 节中，我们将证明，若 (c) 成立且 $P = NP$，则 (a) 和 (b) 也成立。由于利用 (a) 和 (b)，我们分别可以通过执行步骤 (1) 和 (2) 来求解 Q，因此可以推知，若 (c) 成立，则只要 $P = NP$，就能在多项式时间内求解 Q。令 A 表示以下条件：

$$A \equiv \text{对于所有在某个多项式空间 } T(n) \text{ 中运行的 } M\text{，(c) 都成立}$$

若 A 成立，则只要 $P = NP$，就能在多项式时间内求解 $PSPACE$ 中的所有问题。也就是说，

$$(P = NP \ \wedge \ A) \Rightarrow P = PSPACE$$

因此， A 是引言中提到的充分条件之一 X。遗憾的是，这份草稿没有给出条件 A 的证明。条件 A 是否可行，将在第 3 节讨论。第 4 节将介绍一些尝试证明该条件（的较弱版本）的方法。

接下来我们证明，(c) 和 $P = NP$ 蕴含 (a) 和 (b)。

2.1 证明如果 (c) 且 $P = NP$，则 (a) 和 (b)

由于 (c) 包含 (b)，因此 (c) 显然蕴含 (b)。下面证明，如果 $P = NP$ 和 (c) 成立，那么 (a) 也成立。顺便说一句，下面的构造将让我们发现一个新的充分条件 A'，它比 A更弱。

如果某个（可能长达指数级的）图灵机执行过程存在一个多项式大小的表示，那么我们如何在多项式时间内构造出该执行过程的模型呢?

让我们构造一个函数 chained：给定一台图灵机 M、该机器的输入 x 、一个配置 c，以及一个布尔函数 f （更准确地说，是一个计算 f 的计算设备；为简便起见，两者都称为 f），chained(M, x, f, c) 会在函数 f 于配置 c 满足某个特定的必要条件、从而成为计算 $f_{M,x}$ 的合适候选项时返回 1。具体而言，chained(M, x, f, c) 将检查 f 是否按照 M 在配置 c 处与前一个/后续配置正确“衔接”，也就是说，检查 f 对 c 的回答是否与所有可能紧接在 c 之前、且能在 M 中出现的配置对应的回答一致，同时也与 f 对按照 M 紧接在 c 之后的配置所作的回答一致。如果 f 表示，当初始配置为 x 时， M 最终会经过 c （$f(c) = 1$），那么要么 $c = \sigma(x)$（它就是初始配置），要么存在另一个配置 c'，使得 $f(c') = 1$，并且 c 可从 c' 经由 M 的单步计算到达。我们认为， $previous_M(c)$ 表示配置集合 c' ，其中的配置满足：根据 M 的转移，c' 的后续配置将是 c。类似地，如果 f 表示 c 已经到达，那么要么 $c = c_A$ （接受配置），要么 f 必须表示 M 在 c 之后的后续配置也已到达。我们认为，$next_M(c)$ 表示 c 在 M 中紧接着的那个后续配置。

由于根据条件 (c)，我们将假设存在一个计算设备，它能在关于 n （ x的大小）的多项式时间内计算$f_{M,x}$ ，因此，对于我们的目的而言，任何执行步数超过某个多项式 P的 f 函数都可以视为无关紧要；所以，在 chained 的执行过程中，只会检查至多执行 $P(n)$步的 f 计算。实际上，从现在起，我们将针对每个多项式 P讨论一个 chained$_P$ 函数，而不是讨论单个 chained 函数。在下面对 chained$_P$的定义中，我们用 $f(a)^{\leq G(b)}\downarrow w$ 表示： f 在 $G(b)$步或更少步内完成其计算，输入为 a ，并返回 w。

根据这些考虑，函数 chained$_P$ 定义如下。回顾一下， $\sigma(x)$是 M的初始配置。

$$\texttt{chained}_P(M,x,f,c) = \begin{cases} 1 \text{ if } \left(f(c)^{\leq P(n)}\downarrow 0 \ \wedge \ c \neq \sigma(x)\right) \\ \qquad \vee \\ \qquad \left(\begin{array}{l} f(c)^{\leq P(n)}\downarrow 1 \wedge \\ (c \neq \sigma(x) \rightarrow \exists\, c' \in previous_M(c) : f(c')^{\leq P(n)}\downarrow 1) \wedge \\ (c \neq c_A \rightarrow f(next_M(c))^{\leq P(n)}\downarrow 1) \end{array}\right) \\ 0 \text{ otherwise} \end{cases}$$

我们可以看出，chained$_P$ 相对于其输入的大小（一个元组（M、 x、 f、c））在多项式时间内运行。对函数 f的所有调用都会执行一定次数（n），该次数就是输入元组（x）中一部分的大小。此外，我们注意到，计算 $previous_M$(c) 只需反向遍历所有根据 M能够到达配置 c 的转移；这些转移的数量相对于 n 是常数，因为大小 M （以及其转移集 $_P$ 的大小）相对于 M是常数，而

其转移集 Δ 的大小）相对于 n是常数。此外， $next_M(c)$ 返回单个配置，计算起来很简单。

我们将使用函数 chained_P 来找出检测 $f \neq f_{M,x}$ 的一个充分（但非必要）条件：如果对于某个大小不超过 $T(n)$ 的 c，$\text{chained}_P(M,x,f,c) = 0$，那么显然 f 并未计算 $f_{M,x}$，因为对于某个前一个/后一个配置，f 未能正确衔接其行为。需要指出的是，即使对于所有这些 c，$\text{chained}_P(M,x,f,c) = 1$，也可能出现 $f \neq f_{M,x}$。具体而言，可能存在一些配置 $c_1, \ldots, c_z$，使得 $f(c_1) = 1, \ldots, f(c_z) = 1$，但 M 在接收输入 x 时最终不会遍历其中任何一个配置。令 $c^* \in \{c_1, \ldots, c_z\}$，因此当 M 接收 x 时，c^* 最终不会被遍历，但我们有 $f(c^*) = 1$。注意，如果 f 对某个配置返回 1，那么它也必须对某个紧邻的前一个配置以及某个紧邻的后一个配置返回 1。从 c^* 出发，沿着 M 的转移逆向经过一些满足 $f(c') = 1$ 的配置 c'，我们永远不会到达一个这样的配置 c''：它会被 M 从配置 $\sigma(x)$ 出发最终遍历到。如果 $f(c'') = 1$，那么 $f(next(c'')) = 1$，依此类推，因此先前逆向遍历经过的所有配置确实都会被 M 遍历到（包括 c^*，这就产生了矛盾）。因此，只有当 c^* 属于一个由连续配置构成的环时，$f(c^*) = 1$ 才有可能成立；该环中的配置均未被 M 从 $\sigma(x)$ 出发实际到达（但 f 对它们全部返回 1）。需要指出的是，M 会在 c_A 处停止（即唯一的接受配置），因此环中不可能包含 c_A；而 M 无法脱离环，因为它是确定性的。也就是说，$f(c_A) = 1$ 当且仅当 M 确实从 x 到达 c_A。

这意味着，对于任何满足 $\text{chained}_P(M,x,f,c) = 1$ 的计算设备 f，只要该条件对所有大小小于 $T(n)$ 的配置 c 都成立（回想一下，$T(n)$ 是 M 处理输入 x时所用空间的上界），就可以在前一节所示算法的步骤 (2) 中使用该设备：即使 f 包含前文提到的伪循环，M 也会在且仅在 $f(c_A) = 1$时接受 x。因此，前一节给出的条件 (c) 可以放宽。我们确实不必要求，具有所需大小且在所述时间内完成计算的函数必须是$f_{M,x}$。我们也可以使用任何满足 $\text{chained}_P(M,x,f,c) = 1$ 的函数 f，只要该条件对所有大小符合要求的 c都成立，即使在 $f \neq f_{M,x}$ 中也是如此。这就引出了前一节所述条件 (c) 的另一种定义：

(c') 存在多项式 D 和 P，使得对所有 x，都存在一个计算设备 f，其大小不超过 $D(n)$；对于所有大小至多为 $T(n)$ 的 c，都有 $\text{chained}_P(M,x,f,c) = 1$；并且对于该大小范围内的所有 c，f都在 $P(n)$ 时间内运行。

这个替代要求 (c') 推出了以下替代条件 A'：

$A' \equiv$ 对于所有在某个多项式空间 $T(n)$ 中运行的，(c') 成立

在本章节的其余部分，我们将证明

$$(P = NP \ \wedge \ A') \Rightarrow P = PSPACE$$

不难看出，A 蕴含 A'（因为 (c) 蕴含 (c')，我们可以取 $f = f_{M,\ x}$），因此证明前一条陈述也就证明了前一节中给出的原始陈述（即涉及 A 而非 A' 的那条）。

继续证明之前，我们先简要介绍两种也可采用的替代构造：

(I) 我们很容易修改构造，使得链式$_P$只允许不含伪循环的函数。只需考虑到配置还包含一个额外的数值参数，用来表示 M执行过程中的当前步数。注意，执行步数可以用多项式规模表示：由于 M 在多项式空间 $T(n)$内运行，只要 M 不重复任何配置并无限循环，其对任意输入的执行步数最多为 $T(n)$的指数级（可能的配置数量为 $T(n)$的指数级），因此只需多项式数量的比特即可表示任意执行步数。给定附有编号 r的配置，其前一个配置必然附有编号 $r-1$，下一个配置则附有编号 $r+1$。这样一来，链式$_P$就不允许循环行为。

(II) 再来考虑允许伪循环的情况。不失一般性，我们可以修改 M，使其到达 c_A 时不停止，而是在纸带上写下初始配置，并返回状态 s_1。也就是说，它会从 c_A 返回 $\sigma(x)$，因此，从初始配置到接受的路径也会处于一个环内。在这种情况下，只有表示执行循环集合（且仅此而已）的函数 f 才能满足对所有 c都有链式$_P(M,\ x,\ f,\ c) = 1$ 这一条件。不过，满足该条件的任何函数 f也可用于我们算法的步骤（2）：由于 c_A 会导向$\sigma(x)$，只有当 $\sigma(x)$也会导向 c_A时，c_A 才可能属于某个环。因此，我们也有 $f(c_A)$当且仅当 M 接受 x。

接下来我们继续证明。考虑以下集合：

$$G_{P,T} = \left\{ (M,x,f) \,\middle|\, \begin{array}{c} \exists\, c : (\text{the representation size of } c \text{ is} \leq T(n) \wedge \\ \texttt{chained}_P(M,x,f,c) = 0) \end{array} \right\}$$

$G_{P,\ T}$由所有满足以下条件的三元组 $(M,\ x,\ f)$ 构成：f在某个点“中断”执行链（即，对于某个大小低于所需阈值 $T(n)$ 的 c，我们有 chained$_P(M,\ x,\ f,\ c)$）。

下面证明 $G_{P,\ T} \in NP$。给定一个三元组（$M,\ x,\ f$），非确定性图灵机（NDTM）可以按如下方式在多项式时间内判定（$M,\ x,\ f$）$\in G_{P,\ T}$ 是否成立。首先，它以非确定性方式生成任意一个 c，其表示大小小于 T（n）。接着，它以确定性方式检查我们是否已串接 $_P$（$M,\ x,\ f,\ c$）$= 0$，这需要多项式时间。因此，该非确定性算法可在多项式时间内判定（$M,\ x,\ f$）$\in G_{P,\ T}$是否成立。

由于我们假设 $P = NP$，因此有 $NP = co-NP$，所以集合

$$\overline{G_{P,T}} = \left\{ (M, x, f) \,\middle|\, \begin{array}{l} \forall\, c : (\text{the representation size of } c \text{ is} \le T(n) \rightarrow \\ \qquad \texttt{chained}_P(M, x, f, c) = 1) \end{array} \right\}$$

也属于 NP。由于 $P = NP$，我们推得 $G_{P,\ T} \in \overline{P}$。

考虑以下集合：

$$H_{P,T} = \left\{ (M, x, f, s) \,\middle|\, \begin{array}{l} (M, x, f) \in \overline{G_{P,T}} \wedge \\ \text{the bit sequence representing } f \text{ is} \\ \qquad \text{lexicographically lower than the bit sequence } s \end{array} \right\}$$

我们有 $H_{P,\ T} \in P$ 因为 $G_{P,\ \overline{T}} \in P$，且第二个条件也可以在多项式时间内检验。

因此，集合

$$W_{P,D,T} = \left\{ (M, x, s) \,\middle|\, \begin{array}{l} \exists\, f : (\text{the representation size of } f \text{ is} \le D(n) \wedge \\ \qquad (M, x, f, s) \in H_{P,T}) \end{array} \right\}$$

属于 NP：一台非确定性图灵机只需非确定性地生成任意一个表示大小小于 $D(n)$ 的 f，然后确定性地检验 $(M,\ x,\ f,\ s) \in H_{P,\ T}$ 是否成立，这可以在多项式时间内完成。

由于 $P = NP$，我们也有 $W_{P,\ D,\ T} \in P$。因此，判断是否存在 f，使得其大小至多为 $D(n)$、f 在字典序上小于给定的比特串 s，且 f 对我们的目的而言有效（即 $(M,\ x,\ f) \in G_{P,\ T}$）的问题，可以在多项式时间内解决。

根据 $W_{P、\ D、\ T}$ 的这一性质，我们得出结论：前一节提出的算法中的步骤（1）可以通过以下操作在多项式时间内解决。给定 M 和 x，我们执行二分搜索，找出满足 $(M, x, f) \in G_{P,T}$ 的某个 f；这通过多次调用过程 $(M, x, s) \in W_{P,D,\ T}$ 来完成，每次使用不同的 s 值。我们首先将 s 设为 f 所有可能取值的中点，接着反复调用 $(M, x, s) \in W_{P,D,T}$，并根据 $(M, x, s) \in W_{P,D,T}$ 是否成立，将 s 设为某一半范围的中点。调用检查 $(M, x, s) \in W_{P,D,T}$ 的过程的次数是多项式级的，且每次调用都需要多项式时间，因此函数 f 可以在多项式时间内求得——前提是它存在。

根据假设（c'），f 确实存在，因此步骤（1）成功。

在步骤（1）中得到 f 后，我们检查 $f(c_A) = 1$ 是否成立，这可以告诉我们 $x \in Q$ 是否能在多项式时间内完成。

因此，只要满足所需条件，前述过程就能让我们在多项式时间内判定 $PSPACE$ 中的任意问题。

3 关于条件 A'可行性的讨论

本章节讨论条件 A 和 A' 是否可行。由于 A' 弱于 A，下文只提及 A' 。本章节中支持或反对 A' 的论证可能会引起争议，有时也含糊不清，而且通常说服力有限。

首先，我们注意到，如果只要求（c'）中规定的两个条件之一，那么 A' 就可以轻易满足。一方面，我们可以构造一个大小恒定的计算设备 f，用于判断 M 在接收到 x 后是否遍历了任何配置 c，尽管它可能需要指数时间才能作出判断：我们只需模拟 M，并检查是否遍历了 c。另一方面，我们可以构造一个大小呈指数增长的计算设备 f，用多项式时间判断 M 是否遍历 c： f 被构造成一棵二叉决策树，从根到叶的每条路径分别表示大小为 T（n）的每个可能配置，而叶节点则根据该配置是否被 M 从 x遍历来给出“是”或“否”。 f 的一次执行包括从根节点沿着表示所选配置的路径走到叶节点（T（n）步），并提供叶节点中的答案。因此，我们当然可以单独满足（c'）规定的任一个多项式界限，或者单独满足另一个界限（此外，第一个界限还可以设为常数，远低于多项式要求）。于是出现了以下问题：能否同时满足这两个多项式限制?

请注意，大多数布尔函数都无法用相对于输入大小为多项式规模的比特序列表示 [6], ，无论我们如何对这些比特序列进行编码（也无论它们表示的是图灵机、有限自动机、命题逻辑公式、逻辑电路等）。如果我们考虑接收 n 比特并返回 1 比特的布尔函数，那么这样的函数共有 2^{2^n} 个；若要用相同位数的二进制编码唯一标识它们，则需要$log_2(2^{2^n}) = 2^n$ 比特。或者，我们可以为其中一些函数分配较短的编码，但这样做的函数只能是少数。例如，我们可以只用 $log_2(2^{n^k}) = n^k$ 比特对 2^{n^k} 个函数进行编码，但这样一来，我们能够以多项式规模表示的布尔函数比例就只有 $\frac{2^{n^k}}{2^{2^n}}$。也就是说，只有少数函数能够获得多项式规模的编码。

对于属性 A' 而言，这似乎是个坏消息，因为它要求使用 多项式大小的代码来表示一个布尔函数，该函数用于识别给定输入下图灵机所遍历的所有配置。然而，我们需要注意，被遍历配置集合并不是任意类型的集合，而是受到所考虑图灵机约束的集合。具体而言，被遍历配置必然彼此关联，正如图灵机的转移所定义的那样：如果到达了某个配置，那么它的某个前置配置以及下一个配置也都已到达。这一要求严格限制了所考虑布尔函数的形式。

然而，图灵机施加的最明显约束是：相对于输入 x 的大小，图灵机的大小是常数 。令 p, $n \in \mathbb{N}$。令 J 表示所有有序对 (M, x) 的集合，其中 M 是一台具有 p 个状态的图灵机，x 是一个具有 n 个比特的输入。能够识别所有有序对 (, p) 所遍历的所有配置的布尔

函数是否能包含从 n 个比特映射到 1 个比特的所有 布尔函数（即 2^{2^n} 个函数）？尽管当 p 远高于 n 时这或许是可能的，但我们只需考虑更高的 n，使其不可能实现。回想一下，性质 A' 是关于 n 以渐近方式定义的，因此让我们以这种方式来考察它。M 的大小相对于 n 是常数，所以从渐近意义上说，有序对 $(M, x) \in J$ 最多只能表示 $\mathcal{O}(2^n)$ 个不同的布尔函数（有 2^n 个可能的输入 x，其大小为 n，而我们至多可以为每个输入指定一个不同的函数，因此最多能表示 2^n 个不同的函数）。也就是说，对于大小为 n 的输入，从渐近意义上看，需要考虑的可能布尔函数有 2^n 个，而不是 2^{2^n} 个。如果只有 2^n 个函数需要表示，那么根据某种编码方式，所有这些函数都可以用某个多项式大小的代码加以识别。直观地说，当我们考虑较大的输入时，图灵机所遍历的配置集合的“简单性”会在渐近意义上显现出来。

遗憾的是，简单性有几种彼此不兼容的定义：对于某种表示系统而言简单的事物，对另一种表示系统而言可能很复杂。例如，已知奇偶函数无法用深度为 $\mathcal{O}$（1）且规模为多项式 [4]的逻辑电路计算。特别是，这意味着奇偶函数无法由多项式规模的析取范式或合取范式计算；乍看之下，考虑到这个函数如此简单，这一点或许令人惊讶。然而，一个只有两个状态的平凡有限自动机就能计算奇偶函数。因此，几种简单性的定义可能彼此不兼容，性质 A' 所要求的简单性即使在渐近意义下也无法满足。

在这场讨论中，另一种可能的论据是，许多经典的可计算性与复杂性结果直接或间接地告诉我们：如果你想知道一台图灵机接下来 n 步会做什么，那么一般来说，你就必须执行或模拟这 n 步，没有捷径可走。例如，要发现一个程序不会停机，本质上就需要永远模拟它。然而，第 2 节给出的算法（1）—（2）能让我们用多项式时间得知一台图灵机在（可能长达）指数时间内会做什么。因此，如果 A' 成立，我们就能“作弊”，用 k步推断出 $n > k$ 步后会发生什么。

请注意，我们的构造并不能证明只要 A' 成立，我们就能这样作弊。它只证明了，只要我们有 A'和 $P = NP$，我们就能这样作弊。如果 $P \neq NP$成立（大多数研究者都这么认为），那么这种作弊方式就不可能实现，这确实符合我们的直觉。

让我们将性质 A' 与电路复杂性类的某些相关要求进行比较。通常，电路复杂性类要求存在一台图灵机，它在给定待解决输入的大小后，能够构造出一组有限电路，用于解决所有具有该大小的输入所对应的问题。因此，如果我们要解决一个大小为 n的输入所对应的问题，首先，我们的图灵机会生成一组适用于该大小输入的电路，然后使用这些电路来解决问题。如果要求多项式时间或对数空间一致性，那么构造这些电路的图灵机就必须在多项式

时间或对数空间内分别构造这些电路。然而，我们的性质 A' 并不要求能够在特定时间或空间内计算出满足 chained$_P(M,\ x,\ f,\ c)$、适用于某个大小的所有 c的计算设备 f。它只要求这种计算设备（图灵机、命题公式等）存在，这显然是一个弱得多的要求。

4 讨论：尝试证明 A' 的一些思路

本节讨论证明 A' 的可能方法。所有这些想法都非常具有推测性。

我们考虑以下几种可能性:

(a) 链式 $_P$ 的定义展示了 f 为保证正确性必须满足并保持的要求。对于每台图灵机 M，我们都可以将这一要求定义为一个函数方程，其中函数 f 的定义是一个命题逻辑表达式，它取决于自身 在其他参数（根据 M 表示前一个 和下一个 配置的参数）下的取值。能否利用不动点求值技术，找出满足这类函数方程的任意函数 f 的特定形式，并证明这种形式总能被简化为多项式大小？如果可以，性质 A' 就成立，因为命题逻辑公式总能在关于其大小的多项式时间内求值为真或假。

(b) 图灵机可以很容易地由一维元胞自动机（CA）模拟。CA 中每个单元在时间 $t+1$ 的状态由一个布尔函数确定，该函数取决于时间 t 时数量固定的邻近单元；并且所有单元都使用同一个以其局部邻居为参数定义的布尔函数。如果要用 CA 模拟一台 TM，我们可以为这台 TM 构造一个特定的 CA，也可以使用通用 CA 来模拟任意 TM（例如通用自动机 110 [2, 5], ，其中所有单元的布尔函数仅取决于该单元当前的值及其左右两个相邻单元的值）。考虑一个在 n^k 空间内运行的 TM，由 CA 对其进行模拟。从时间 t 到时间 $t+1$ 的每次 CA 转移中，都会将某个函数 f（从 n^k 比特映射到 n^k 比特）应用于所有单元。因此，若 x 是所有单元的初始状态，那么模拟 z 步 TM 之后，CA 的状态为 $f \circ f \circ \ldots \circ f(x) = f^z(x)$。此外，每一步所应用的函数 f，都是将同一个大小为 $\mathcal{O}(1)$ 的局部布尔函数应用到纸带上的所有单元所得的结果，因此 CA 的计算相当于在空间和时间上对称地应用某个简单的大小为 $\mathcal{O}(1)$ 的逻辑电路。这个特性是否意味着由 x、$f(x)$、$f^2(x)$、.....、$f^{2^{n^k}}(x)$ 构成的集合（即原 TM 对输入 x 所经历的所有配置）总能被简化为多项式大小的表示？如果可以，A' 就成立。

(c) 如果我们将函数 f 视为一个 n^k 维超立方体，其中顶点表示配置，那么从一个配置到另一个配置的转移就可以视为这个 n^k 维超立方体空间中的向量。每个顶点都只有一个出向量（也就是说，它们会指向另一个配置），因此我们可以将整个图景视为一个向量场。由于 f

M 的转移集 Δ 中存在 $\mathcal{O}(1)$ 个转移，因此 n^k 维超立方体中的所有向量都属于 $\mathcal{O}(1)$ 种类型之一。假设 M 的一条转移规定了当 M 处于状态 s_4 且读写头所指向的带符号为 1 时， M 必须如何改变。假设配置子串 1011 表示“状态是 s_4 ，读写头位于此处”，而 0100 表示“此处的符号是 1”。根据第 2 节开头提出的配置表示法，读写头位于表示状态的配置子串右侧的符号处，因此，任何包含子串 10110100 的配置都会触发这条转移。特别是，任何包含此子串（无论位于何处）的配置都会以相同的方式作出反应。

因此，我们在向量场中便有了这种“简单性”或“重复性”模式。它能否使 n^k-立方体拥有多项式 数量的 $n^{k'}$-立方体（带有 $k' < k$），从而使得所有表示被遍历配置（或如第 2.1 节所解释的、位于伪循环之内的配置）的顶点都被包含在其中之一？请注意， n^k-立方体中的每个 $n^{k'}$-立方体都可以由一个合取项表示。例如， x_1 为真、 x_4 为假且 x_7 为真的立方体由项 $x_1 \wedge \neg x_4 \wedge x_7$ 表示。该项表示满足该条件的所有顶点，而不论它们对 x_2、 x_3、 x_5 等的取值如何。一个恰好对属于这些 $n^{k'}$-立方体中某些立方体的所有配置返回 1 的命题公式，只需将所有这些项进行析取即可构造出来。也就是说，所得的析取范式将计算 f。如果 f 中合取项的数目（即 $n^{k'}$-立方体的数目）是多项式的，那么该析取范式具有多项式规模（并且，显然，f 可以在多项式时间内计算）。

我们可以选择是否将每个伪循环纳入 f （它们并非必需，但无害），这一点增加了挑选这些$n^{k'}$立方体时的灵活性。此外，还可以人为添加一些伪转移（即不会改变 M 对 x的行为的转移），以进一步增加灵活性。

到目前为止，我们还没有考虑过任何一种证明 A' 的方法，其中 f 由一台在多项式时间内运行的图灵机表示（而不是像思路 (a)、(b) 和 (c) 中那样，由逻辑电路或命题公式表示）。要推理在给定阈值以下（例如状态数少于该阈值）的图灵机能够或不能够做什么，通常很困难。也许可以在这里运用某种对角化论证?

柯尔莫哥洛夫复杂度 通过求解问题的程序大小来研究问题，并且已有一些有时间界的柯尔莫哥洛夫复杂度概念被提出，其中包括多项式时间的概念（例如参见 [3]）。遗憾的是，据我们所知，目前尚无专门针对以下对象的研究结果：多项式时间图灵机的柯尔莫哥洛夫复杂度，这类图灵机能够识别某个多项式空间图灵机在某个输入下所经过的有限配置集合。无论如何，我们希望对这一领域的文献进行更深入的研究，能为解决我们的问题提供一些线索。

5 另一种方法：处理一个 $PSPACE$完全问题

本章节中，我们考虑一种截然不同的方法，以寻找一个足以推出 $P = NP \Rightarrow P = PSPACE$ 的条件。我们不会笼统地考虑任意 PSPACE 问题，而是聚焦于一个特定的 $PSPACE$ 完全问题。具体而言，我们将证明一个性质：若该性质成立，那么只要 $P = NP$ 成立，某个 $PSPACE$ 完全问题就可以在多项式时间内求解。由于所有 $PSPACE$ 问题都可以在多项式时间内归约到任意 $PSPACE$ 完全问题，这便可以证明 $P = PSPACE$。

让我们考虑全称量化布尔公式问题（真量化布尔函数问题）。这个 $PSPACE$完全判定问题定义如下。设 φ是一个由命题符号 x_1，.....，x_n构成的命题逻辑公式。考虑以下表达式：

$$Q_1 x_1 Q_2 x_2 Q_3 x_3 Q_4 x_4 \dots Q_k x_k \ \varphi$$

其中，$Q_i \in \{\forall, \exists\}$ 以及量词 $\forall$ 和 $\exists$ 在序列 Q_1，.....，Q_k中严格交替出现。如果我们得到一个完全量化的表达式（即不存在自由变量），那么就称其为一个QBF 公式。

全称量化布尔公式问题是所有求值结果为 $\top$的量化布尔公式表达式的集合。因此，求解全称量化布尔公式问题就是检查给定表达式是否等价于 $\top$ 或 $\bot$。由于所有变量都已量化，每个表达式必定等价于 $\top$ 或 $\bot$。

一种直接求值此类表达式的方法，是从最后一个量词到第一个量词，迭代地消去量词，具体做法是进行以下替换：

$$\forall x \ \varphi \equiv \varphi[x/\top] \ \wedge \ \varphi[x/\bot]$$
$$\exists x \ \varphi \equiv \varphi[x/\top] \ \vee \ \varphi[x/\bot]$$

这种方法在每一步都会使所得公式的大小翻倍，因此公式会呈指数级增长。然而，移除最后一个量词后，表达式中不再含有变量，并且等价于 $\top$ 或 $\bot$。

设 γ 是一个量化布尔公式表达式，n 是它的大小。设 $q_\gamma < n$ 表示 γ 中量词的数量。假设我们按照之前提出的方法对 γ 求值，不过在进行下一次替换之前，先将每次替换后所得量化布尔公式的命题部分化简至最简形式。我们用 s_1^γ、.....、$s_{q_\gamma}^\gamma$ 分别表示第一次、第二次、.....以及第 $q_\gamma -$ 次替换后，公式中（化简后的）命题部分的大小。

我们引入以下条件：

$$\begin{aligned} B \equiv \ & \text{there exists a polynomial } P \text{ such that,} \\ & \text{for all QBF } \gamma,\ s_i^\gamma \le P(n) \text{ for all } 1 \le i \le q_\gamma \end{aligned}$$

我们来证明：

$$(P = NP \ \wedge \ B) \Rightarrow P = PSPACE$$

如果性质 B 成立，那么只要我们能在多项式时间内简化命题公式，就可以在多项式时间内推断 QBF 是 $\top$还是 $\bot$。每去掉一个量词后，简化 QBF 表达式，所得公式的大小始终保持为多项式。由于公式中量词的数量是线性的（q_γ 不可能高于 n），因此这一操作必须重复线性次。整个过程所需的时间因此是多项式时间。

假设 $P = NP$。简化布尔函数是一个 Σ_2^p 问题（事实上，在图灵归约 [1] 下它是 Σ_2^p -完全的）。由于如果 $P = NP$，整个多项式层次就会坍塌，因此我们知道，$P = NP$ 意味着命题公式可以在多项式时间内简化。

我们推断（$P = NP \wedge B$）$\Rightarrow P = PSPACE$。

我们有什么直观理由怀疑 B 成立吗？与其从一般角度讨论这个问题，不如来看一个特定的量化布尔公式（QBF）实例。

任何量化布尔公式（QBF）实例都可以在多项式时间内归约为一个等价的 QBF，其中命题部分（φ）尤其可以用合取范式（CNF） [7] 表示。让我们看看当 φ 用 CNF 表示时，QBF 的求值会发生什么。

给定一个由 S 个子句组成的集合，我们用 S_x 表示 S 中包含 x 文字的子句子集，用 $S_{\neg x}$ 表示 S 中包含 $\neg x$ 的子句子集，并用 $S_{[x]}$ 表示 S 中既不包含 x 也不包含 $\neg x$ 的子句子集。如果 c 是一个子句，我们用 $c_{[x]}$ 表示从子句 c 中删除 x 的所有出现后得到的结果。例如，$(x_1 \vee \neg x_2 \vee x_3)_{[x_2]} = (x_1 \vee x_3)$。

令 S 为 φ中的子句集合。我们有：

$$\begin{aligned}\exists\, x\, \varphi \equiv \varphi[x/\top] \vee \varphi[x/\bot] \equiv \\ (\textstyle\bigwedge_{c\in S_{[x]}} c \wedge \bigwedge_{c\in S_{\neg x}} c_{[x]}) \vee (\bigwedge_{c\in S_{[x]}} c \wedge \bigwedge_{c\in S_x} c_{[x]}) \equiv \\ \textstyle\bigwedge_{c\in S_{[x]}} c \wedge (\bigwedge_{c\in S_{\neg x}} c_{[x]} \vee \bigwedge_{c\in S_x} c_{[x]})\end{aligned}$$

$$\begin{aligned}\forall\, x\, \varphi \equiv \varphi[x/\top] \wedge \varphi[x/\bot] \equiv \\ (\textstyle\bigwedge_{c\in S_{[x]}} c \wedge \bigwedge_{c\in S_{\neg x}} c_{[x]}) \wedge (\bigwedge_{c\in S_{[x]}} c \wedge \bigwedge_{c\in S_x} c_{[x]}) \equiv \\ \textstyle\bigwedge_{c\in S_{[x]}} c \wedge \bigwedge_{c\in S_{\neg x}} c_{[x]} \wedge \bigwedge_{c\in S_x} c_{[x]}\end{aligned}$$

注意，$\forall\ x\ \varphi$的大小不小于它所转化成的表达式的大小，即 $\bigwedge_{c\in S_{[x]}} c \wedge \bigwedge_{c\in S_x} c_{[x]} \wedge \bigwedge_{c\in S_{\neg x}} c_{[x]}$。即使子句数量相同，其中一些子句也可能更短。因此，公式大小增加的唯一风险来自将 $\exists\ x\ \varphi$替换为$\bigwedge_{c\in S_{[x]}} c \wedge (\bigwedge_{c\in S_x} c_{[x]} \vee \bigwedge_{c\in S_{\neg x}} c_{[x]})$。如果我们对 $(\bigwedge_{c\in S_x} c_{[x]} \vee \bigwedge_{c\in S_{\neg x}} c_{[x]})$ 这个子公式应用分配律，将其重新转换为合取范式，那么通常子句数量会随着该子公式大小呈二次增长。

对该子公式应用分配律后生成的新子句（每个子句的大小至多加倍）不能同时包含某个文字及其否定文字：在这种情况下，整个子句显然为真，可以从子句合取中消去。此外，在所有新子句都生成之后，

我们可以按照 $(\varphi_1 \vee x)$ 和 $(\varphi_1 \vee \neg x)$ 的形式匹配一些有序对的子句。在这种情况下，两者都可以替换为单个子句 φ_1。

很容易构造出这样的量化布尔公式表达式：在消去最前面的量词时，由于在 $\exists$ 次消去中应用分配律而导致的二次扩张，显然超过了之后在 $\exists$ 次消去中或全部 $\forall$ 次消去中产生的简化。如果在许多这样的消去过程中，这种趋势持续下去，那么量化布尔公式的大小显然会呈指数级增长。然而，在一些手工构造的例子中，我们观察到，在消去最前面的量词时，可用的简化也会迅速增加。“简化力”是否可能在公式呈指数级增长之前 就开始超过“扩张力”？请注意，简化最终总会胜出：公式最终会是 $\top$ 或$\bot$。然而，它们 何时开始胜出？它们能否在表达式达到指数级大小之前就做到这一点?

如果 $P = NP$，那么命题公式可以在多项式时间内化简。这同样适用于命题公式的受限子集，例如合取范式和析取范式。因此，如果简化力在量化布尔公式呈指数级增长之前超过扩张力，那么通过逐个消去量词，并接着化简所得公式（由于 $P = NP$，这可以在多项式时间内完成），我们就能在多项式时间内解决全称量化布尔公式问题，因此 $P = PSPACE$。

5.1 研究性质 B 的可能方法

通过计算机实验对第 2.1 节中给出的性质 A' 进行经验验证，似乎是一个相当困难的目标。首先，这需要挑选在多项式空间内工作的图灵机（因此也许需要指数时间），并让它们处理一些输入。对于某个输入，收集这些图灵机之一遍历的所有配置，可能需要收集指数数量的配置。收集完这些配置后，我们必须检查是否存在一种表示它们的多项式规模计算装置，并且该装置能在多项式时间内运行。例如，我们可以用命题公式表示它们；显然，命题公式的运行时间相对于其大小是多项式的。在这种情况下，寻找多项式规模的表示，就需要对一个外延地覆盖所有被遍历配置的命题公式进行简化。值得注意的是，后者可以直接构造为一个析取范式，其中每个被图灵机遍历的配置对应一个合取项。简化这一外延表达式，需要对一个指数规模的析取范式求解指数时间问题（目前已知最好的简化算法都是指数时间的）。因此，整个方法将需要双重指数时间！如果我们还想检查在存在或不存在某些伪循环或无害的额外转移时，简化效果是否更好，难度就更高了。

相反，通过类似实验来实证检验性质 B 似乎容易得多。第 5 节提出的求解全称量化布尔公式的方法，是在消去每个量词后进行一次新的简化。由于量词的数量与 QBF 公式的大小呈线性关系，因此必须解决一个指数级问题（简化），并重复线性次。

这使得这一目标更容易实现，因此可以分析更大的实例。于是，我们的第一步是进行实验，以实证检验大量案例中的 B。

如果实验表明 B 在许多情形下成立，我们就会尝试证明它。一种可行的做法是：为量化布尔公式在消去量词的过程中所受的扩张力与收缩力建立形式模型，并找出简化后的量化布尔公式的大小随时间变化的某种不变量。

6 A'的一个生动推论

让我们简要介绍性质 A'的一个奇特推论。

在第 2.1 节中我们看到，若 A' 成立，则对于所有在多项式空间内运行的图灵机 M 以及所有输入 x，存在一个多项式规模计算装置，它关于 M 是良好链接的，并且以多项式时间运行。也就是说， A'保证存在一个针对 M以及每一个特定输入 x 的小且快的良链计算设备。接下来我们证明，对于 M 以及某一大小的所有输入，也存在一个小而快的良链计算设备，即一个同时处理某一大小的所有输入的装置。

设 M 是一台在多项式空间内运行的图灵机。为简单起见，假设 M 的输入是二进制序列，即 $\{0,1\}^*$ 中的字符串。设 s_A 是 M 唯一的接受状态。不失一般性，我们还假设它只有一个拒绝状态 s_R （否则，我们可以修改 M，使得所有会导致 M 停机的状态与纸带符号组合都实际转移到 s_R，然后才真正停机）。

我们修改 M ，使它在一次运行中处理某个大小的所有输入。在 M 的初始状态，我们引入新的转移，使其立即将输入复制到纸带的另一处（假设原机器 M 永远不会到达复制区域）。接着，机器回到原始输入的开头，并进入原来的初始状态 s_1。

此外，我们还添加其他转移，使得当 M 到达 s_A 时，它会擦除纸带上除上一段所述输入副本之外的所有内容，移至该副本的左边界，并进入一个新状态 s'_A。接着，它将该序列加 1，把所得序列复制到 M 原先接收输入的区域，将读写头移至新副本的第一个符号处，并进入 s_1。

我们对拒绝状态 s_R 也做类似处理：添加新的转移，使得当 M 到达 s_R 时，它也会擦除除输入副本以外的所有纸带内容，到达副本的左边界，然后到达一个新状态 s'_R。接着，它将输入副本加 1，在 M 的原始工作区复制该副本，指向新副本的第一个符号，并到达状态 s_1。

令 $m \in \mathbb{N}$。我们还对 M 做了修改：当它将输入副本加 1 时，会检查该副本是否大于 m。如果是，那么它不会复制该副本并再次返回 s_1 ，而是直接停止。

令 M' 为引入所有这些修改后得到的新图灵机。我们可以看到， M' 会迭代模拟 M ，处理从其原始输入到 m 的所有输入。考虑 $m = 1 \ldots 1$（n 1 个 s），并令输入为 0.....0（n 0 个 s）的 M。对于每个可能的、大小为 n 的输入， M' 最终都会到达这样一种配置：纸带上只包含序列 y，读写头位于 y 的左边界，且当且仅当 M 接受 y 时，状态为 s'_A 。类似地，它最终也会到达这样一种配置：纸带上只包含序列 y，读写头位于 y 的左边界，且当且仅当 M 拒绝 y 时，状态为 s'_R 。

我们注意到， M' 相对于 n在多项式空间内运行：对于每个长度为 n的序列 y ，它对输入 y执行 M ，而该执行相对于 n在多项式空间内运行。 M' 所使用而 M 未使用的唯一额外空间，是用于保存当前输入副本的空间（其大小为 n）。因此， M' 在多项式空间内运行。

由于性质 A' 适用于任何在多项式空间内运行的图灵机，它也适用于 M'。因此，若 A'且 $P = NP$，则我们有一种替代方法，可在多项式时间内得知原始图灵机 M 是否接受大小为 n的任何输入 y 。我们只需按第 2.1 节所述，为机器 M' 和输入 0...0 找到良链式多项式大小多项式执行时间函数 f ，然后对一个表示 y 位于纸带上、读写头位于其左边界且状态为 s'_A的配置调用 f。我们有 f 返回 1 当且仅当 M 接受 y。该方法与第 2 节所给方法的区别在于，找到 f 之后，我们可以将其用于所选大小的所有输入，也就是说，我们无需为该大小的每个输入寻找一个不同的函数 f。

请注意，即使 $P \neq NP$, A'也意味着对每个大小 n，都存在这样的函数 f。也就是说，对于任何 $PSPACE$问题，如果 A'成立，那么就存在一个快速且小型的计算设备，能够针对给定大小的所有输入 n解决该问题，即使 $P \neq NP$。如果我们已经拥有 f，就可以利用它快速解决任何大小为 n的实例。

7 结论

在这份草稿中，我们介绍了一些能够推出 $P = NP \Rightarrow P = PSPACE$的性质。据我们所知，$A'$既不能推出 B，反过来也不成立，因此其中一个不成立并不意味着另一个也不成立。

如果我们目前用于证明 A' 和 B 的思路仍然受阻，而且我们也没有新的思路，那么下一步将是进行关于 B 的实验，该实验在第 5.1 节中提及。

致谢

感谢 Javier Rodríguez-Laguna、Fernando Rosa 和 Fernando Rubio 对这份草稿提出的宝贵意见。

参考文献

1. D. Buchfuhrer 和 C. Umans。布尔公式最小化的复杂性。计算机与系统科学杂志，77(1)：142–153，2011年。2. M. Cook。初等细胞自动机的普适性。复杂系统，15 40，2004年 1：–。3. Lance Fortnow。柯尔莫哥洛夫复杂度与计算复杂性。载于计算的复杂性与证明。数学笔记，2004年。4. J. Hastad。小深度电路的计算局限，博士论文。博士论文，麻省理工学院，1987年。5. T. Neary 和 D. Woods。细胞自动机规则 110 的 P 完全性。载于计算机科学讲义 *4051*，第 132–143 页。Springer，2006年。6. C.E. Shannon。双端开关电路的综合。贝尔系统技术期刊，28(1)：59–98，1949年。7. M. Sipser。计算理论导论。Thomson，2006年。