算法信息论：物理学家和自然科学家的综述

S D DEVINE
维多利亚大学维多利亚管理学院，邮政信箱600，
惠灵顿，6140，新西兰，
sean.devine@vuw.ac.nz

2014年5月30日

前言

我最初是一名研究物理学家，在政府实验室工作，发表了一些论文。 在成为新西兰科学体系中的一名经理的过程中，我成为了一名经济学家，探究通过技术变革创造财富。 在这个过程中，我发现了新古典均衡经济学观点的严重困难，因为这种观点未能认识到经济远离均衡。 在均衡状态下，发展路径是无关紧要的。 然而，苏联和中国转向市场经济时出现的不同路径和结果表明路径是至关重要的。同样，2007-2008年对美国和世界经济的冲击表明宏观经济驱动因素与新古典均衡观点无关。

在寻找一种处理非平衡问题的方法时，我偶然发现了Chaitin在《John Casti的书“复杂化”中概述的算法信息论（AIT）》的工作。 这使我接触到了Gregory Chaitin的作品，最终是Li和Vit´anyi的[76]综合性著作。

虽然这本书是一座宝库，但对于一个外行人来说，要掌握其中的内容是很困难的，因为很多发展的背景对非数学家来说并不清楚。
然而，这本书指出了Kolmogorov在算法复杂性、概率论和随机性方面的工作。 因此，虽然将其应用于经济系统暂时搁置了，但我意识到AIT为科学家提供了一个有用的工具来研究自然系统。

简而言之，AIT、概率和随机性的发展似乎为应用于自然界中研究的有组织复杂系统提供了一种有价值的工具。 不幸的是，这些见解并不容易

对科学界来说易于理解，或以易于应用的形式呈现。 我发现Zurek和Bennett关于算法熵的早期工作提供了对非平衡系统的丰富理解-然而这项工作似乎已经在时间的迷雾中失去了。因此有了这本书。 希望读者能够吸收AIT背后的关键思想，以便更好地理解数学发展并将这些思想应用到自己感兴趣的领域。

肖恩·德文

目录

摘要

第1章

引言

近年来，对复杂性和有组织的复杂系统的兴趣日益增加，无论这些系统是物理、生物还是社会系统（参见例如Casti [23]）。 然而，并不总是有一个一致的理解词汇“复杂性”的含义。 从数学的角度来看，复杂性与随机性相关，即最复杂的结构是那些信息冗余最少或熵最大的结构。 而从物理、生物或社会科学的角度来看，使用短语“复杂系统”来标识那些显示出秩序或结构但不能简单描述的系统。 这些系统的熵相对较低，并且由于它们远离平衡，它们绝不是随机的。 由于本书是从自然科学的角度来写的，短语“有组织的系统”或“有组织的复杂系统”通常用于应用于非随机且远离平衡的有序系统；即自然科学家可能称之为复杂的系统。 然而，由于有时这些有组织的系统也被描述为“复杂”或展示“复杂性”，需要区分“复杂”指的是数学意义上的随机性的情况。

希望上下文能够清楚地表达意思，以避免产生歧义。

一个关键问题是是否应该从“自下而上”的角度探究一个复杂的有组织系统，即从最基本的层面上产生理解。 然而，虽然自下而上的理解可能揭示了有组织系统如何从更简单的系统构建起来，但在某个复杂度水平上，自下而上的意义构建过程变得过于复杂，细节太多。 如果要有更深入的理解，就需要从“自上而下”的高层次方法来探究系统。 自下而上的方法使用最简单的模式生成结构，例如细胞自动机（参见Wolfram [107]）或图灵机，以查看基本层面上的理解是否能够对高层次的复杂或有序系统产生启示。 在更详细的高层次上，离底部还有几步，这些模式生成结构可以被解释为对外部环境做出响应的适应性代理（例如Crutchfield [39]）。

在实践中，观察到的高层复杂行为可以说是新兴的，因为它不能用简单的术语来理解。 例如，在这些更高层次的术语中，可以讨论生物系统中生命形式的适应性反应。 在这些层次上的理解主要是现象学的，通常将整体视为“部分之和”的总和，因为这是人类在有限信息下理解的最好方式。

然而，这里的论点是，算法信息论可以提供“部分之和”的方式，以便深入了解现象学方法背后的原理。 算法信息论（AIT）的方法（参见Li和Vit´anyi [76]和Chaitin [33]）提供了一个可以将简单系统的基本描述与新兴系统的行为理解联系起来的线索，并且在提供对更现象学描述中隐含的底层过程的洞察方面非常有帮助。 当高层描述涉及底层结构时，例如在Stafford Beer的可行系统模型[9, 10]中发生嵌套，算法信息论可以在自上而下的现象学方法和自下而上的原子化或微系统方法之间提供桥梁。

1.1 复杂性的一些方法

Crutchfield和他的同事们使用“因果状态”和统计复杂性的概念发展了计算力学（[41，39]），以描述离散值、离散时间的稳态随机过程中的模式、结构和组织。 计算力学方法认识到，在观察到字符串中的嘈杂模式作为时间或空间序列时，有可能预测序列的未来成员。 这种方法[40，91]表明，为了最佳地预测观察到的随机序列的未来，不需要知道整个过去，只需要知道序列来自哪个因果状态。 生成相同随机未来的过去集合的所有成员属于同一个因果状态。 与因果状态方法相关的度量包括统计复杂性，即因果状态集合的香农熵，过量熵 E和熵率 h_μ 每个符号的熵。 这种方法已经扩展到使用一维自旋链的空间序列的例子[52]。

Wolfram [107]已经证明简单的元胞自动机可以产生丰富的模式。 他提出宇宙本身可以用简单的计算系统来解释。 Bennett [13]提出了“逻辑深度”的概念，即计算系统产生所需模式或结构所需的步骤数，是系统复杂性的度量。

这些自下而上的方法为有组织的系统提供了洞察力。
然而，本书的目的是关注算法信息论（AIT），通过最短描述来衡量结构或对象的复杂性，从而提供对有组织系统的全面理解。

通过其最短描述来衡量结构或对象的复杂性，算法信息论（AIT）可以提供对有组织系统的全面理解。 开发一种允许噪声或模式变化的AIT方法（Devine [46]和Vereshchagin和Vit´anyi [103]）与Crutchfield及其同事的因果状态方法并不矛盾。
沃尔夫拉姆的方法（沃尔夫拉姆[107]最后一章）也可以在AIT框架中解释，而贝内特的逻辑深度是完成计算过程所需的时间或步骤的度量。 下一节概述了AIT的关键要点。

1.2 算法信息论（AIT）

AIT已经出现，为几个不同的数学和物理问题提供了洞察力。 Solomonoff [93]对归纳推理和人工智能很感兴趣。 Solomonoff认为，要发送一系列随机事件的结果（例如一系列足球比赛的结果），每个结果都需要以一串数字的形式传输。 另一方面，前100位小数 π的传输可以通过传输生成前100位小数的算法来更高效地进行。 苏联数学家科尔莫戈洛夫[69]在概率、随机性和互信息的研究中发展了这一理论。 格雷戈里·查伊廷[26]独立开发了这一理论，用于描述字符串中的模式，然后转向研究形式系统的不完备性和数学中的随机性。 在科尔莫戈洛夫学派内部发生了重大发展，但直到该学派的研究人员移居欧洲和美国后，这些发展才变得容易获取。

尽管下一章将更深入地描述该理论，但AIT的核心是，如果一个结构可以用一串符号表示，那么能够生成该字符串的最短算法就是数学家所称的字符串的“复杂性”和结构的度量。

（在这里，为了避免歧义，短语“算法复杂性”将用于表示这个概念。）由于任何算法的长度将取决于所使用的计算系统，这样的度量似乎没有什么用处。幸运的是，生成字符串的指令集的长度并不特别依赖于计算机（见第3.1节）。 因此，由一个有模式的字符串表示的结构可以用比字符串本身更短的算法来描述。 另一方面，一个由符号随机序列组成的字符串表示的系统没有可辨认的组织。

这样的字符串只能用比字符串长度更长的算法来描述。 例如，由一千个重复的‘01’组成的字符串有两千个字符，形式为$=010101...0101$。 这个字符串可以用算法来描述

$$PRINT\ 01,\ 1000\ 次。$$

该算法相对较短。 其大小将由代码‘1000’主导，对于二进制算法来说，它接近于 $log_2 1000$。 另一方面，只能通过指定每个符号来生成相同长度的随机二进制字符串，即P RIN T10011...0111。

指令集至少需要2,000个字符长，再加上一些额外的指令来指定打印语句。 在物理情况下，通常方便从一个前缀无关的代码集中选择每个计算机指令的代码；即没有任何代码是其他代码的前缀。

在这种情况下，计算机不需要任何标记来指示一条指令或代码的结束和另一条指令或代码的开始。 这种编码被称为“自限定”编码（或出于某种奇怪的原因称为“前缀”编码），因为代码携带有关代码长度的信息。在本文中，将使用“前缀无关集合”中的代码词或“自限定”代码词来描述这些代码。 自限定编码的重要性在于指令集遵守Kraft不等式。正如后面所示（第3.2节），将Kraft不等式扩展到无限代码集可以用于确保程序本身是自限定的。 这样一来

- 允许算法和子程序连接（合并），整体长度几乎不增加；
- 提供了描述字符串的算法长度与该算法在所有算法集合中的概率之间的联系 - 从而更紧密地将AIT与信息论联系起来；以及
- 允许定义一个通用分布，并通过这样做提供对投注回报和归纳推理的洞察；

算法信息论和熵

字符串的算法熵被认为与使用自限制编码的算法复杂度测量相同。即，字符串的算法熵等于使用自限制编码生成该字符串的最短程序的长度。 一旦考虑到计算开销（参见4.1和5.2节），计算表示热力学系统平衡态的字符串的算法熵几乎与平衡态的香农熵相同[92]，并且在考虑不同单位的情况下几乎等于统计热力学的熵[101]。 然而，从概念上讲，算法熵与传统熵是不同的。 算法熵是系统确切状态的度量，因此在平衡态之外也具有意义，并且似乎更加基础。Zurek [108]已经展示了算法信息论如何解决统计热力学中的粒度问题。

Zurek还展示了如何在容器中指定Boltzmann气体（即粒子没有内部自由度的气体）的 N粒子的状态。 所有粒子的位置和动量坐标可以在一个$6N$位置和动量相空间中指定。 首先，相空间的颗粒或细胞结构是通过算法定义的。一旦完成了这一步骤，

气体的特定微观状态，对应于实际瞬时配置，可以通过一个由0和1的序列来描述，这取决于相空间中的细胞是空的还是占据。 在平衡态下，这种微观状态的算法熵几乎与系统的Shannon熵相同。 Zurek已经定义了“物理熵”的概念，用于量化一种信息可以通过算法描述的系统，并将剩余的不确定性封装在一个Shannon熵项中。 然而，现在清楚的是，一种新类型的熵，如物理熵，是不必要的，因为可以找到一种能够生成具有结构但也具有结构变化或不确定性的给定字符串的算法，并且其长度将返回与Zurek的物理熵相同的熵值（见下一节和第4节）。 Zurek的物理熵的极限值，其中捕获了所有模式化信息，可以被认为是噪声或可变模式化字符串的真实算法熵。

AIT的问题

正如Crutchfield和他的同事Feldman和Shalizi [91, 42]指出的，AIT存在一些明显的困难。其中一些与习惯或符号有关。 例如，数学家使用“复杂性”来衡量随机程度，这与自然科学家的直观方法不符合。在自然科学家看来，最复杂的系统既不简单也不随机，而是展现出新兴或非简单行为的系统。 只要每个人都保持一致（但事实并非如此），问题应该不大。 正如前面提到的，这里的方法是使用“有组织的系统”一词来指代自然科学家可能称之为“复杂系统”的概念，而使用“算法复杂性”或“算法熵”来表示数学家使用的随机度量。 这些作者认为AIT方法还存在另外两个关键问题：

- 由于图灵停机问题，算法熵是不可计算的；而且
- 字符串的算法描述必须是精确的；似乎没有明显的容忍噪声的空间，因为噪声的描述很可能会淹没一个带有噪声模式的字符串的算法描述。

然而所谓的停机问题是一个事实而不是一个问题。 任何避免停机问题的复杂度度量都不可避免地牺牲了精确性。 然而，当通过任何方式在字符串中发现模式时，它表明存在一个更短的算法。 上面的第二个要点更为关键。作者发现由于假设无法处理噪声或变异，AIT甚至不能应用于最简单的自然系统。 然而，现在已经清楚AIT能够有效地描述带有噪声的字符串（Devine [46]和Vereshchagin和Vit´anyi [103]）。 实质上，表现出噪声模式或变异的字符串的算法熵

基本模式由两部分算法定义。 第一部分指定了模式字符串的集合，第二部分标识了集合中的特定字符串。第二阶段的贡献结果与模式集合的香农熵几乎相同，即集合中固有的不确定性。

使用这种方法描述模式的算法可能不是最短的，算法复杂度或算法熵没有绝对定义。 相反，它是算法熵的上限度量。 在这里，这个上限将被称为“临时”熵，理解为真实熵可能较小（见第4节）。 以前Devine [46]使用了“揭示”的熵这个词。 然而，“临时”可能更好，因为它与Zurek的“物理熵”和第4.2节中概述的特定字符串的算法最小充分统计量测量相吻合。这些不同的度量，以及G´acs的度量（[58, 59]将这个想法扩展到无限字符串）本质上是相同的。它们提供了基于可用信息或观察到的模式或结构的噪声字符串的熵度量。

基于最小描述长度（MDL）的方法（见第7节）也提供了对噪声或可变模式字符串描述的相同见解。 理想的MDL方法通过找到最小描述长度来建模系统；即适合数据的最短模型。 这个描述包含了假设或模型的描述（等同于集合的描述），以及一个用于识别观测数据的代码。 给定模型，特定字符串 x_i 通过基于字符串出现概率p_i的代码进行识别。 如果使用两条竖线表示线之间的字符串长度，那么这个代码的长度，即 $|code(x_i)| = -log_2(p_i)$。 期望值 $|code(x_i)|$是与模型一致的可能字符串集合的香农熵。 然而，虽然MDL方法压缩了数据，但它实际上并没有满足AIT要求算法必须生成字符串的要求。 MDL方法只是解释了字符串，完整的算法描述必须将MDL描述与O(1)解码程序结合起来，以实际生成特定的字符串。 所有这些方法都强调了算法熵包括假设或模型描述的长度，以及与不确定性的香农熵度量几乎相同的进一步贡献。

上述论证表明，具有模式的噪声或可变字符串的算法熵可以通过结合两个算法来有效描述。 一个指定了所有相似字符串集合的结构或模式特征，而另一个则识别集合中的特定字符串。实际上，第二个字符串对应于Shannon熵或相似字符串集合中的不确定性。 当一个单独的字符串没有特定的结构时，该字符串是典型的，算法描述或临时熵返回与所有这些字符串集合的Shannon熵相同的值。 所有相似模式字符串集合的成员具有相同的临时熵。 然而，少数非典型或非代表性的字符串

可能展示出进一步的模式或结构。 这些字符串的算法熵将小于相似字符串集合的典型成员的熵。

1.3 算法信息论和数学

AIT的大部分发展都发生在数学学科中，以解决与随机性、形式系统和与自然科学似乎没有太多联系的问题有关的问题。

然而，由于这些定理是用数学术语表达的，它们对于典型的科学家来说并不容易获得。

算法信息论的一个重要成果是其在形式公理系统中的应用。 一个形式系统的假设和推理规则可以用一个有限的字符串来描述。 通过实现公理和推理规则的程序，可以递归地枚举由适当字符串指定的形式系统的定理。 Chaitin [30]已经证明了一个定理可以通过公理、推理规则和证明定理的指令集来进行压缩。 一个定理的算法复杂度是生成该定理的算法的长度，从而证明它是可证的。

Chaitin$n$ [30] 还表明，一些可能的定理或命题无法被压缩，即无法用更简单的公理和规则来表述。 由于这些命题由无法压缩的字符串表示，它们是不可判定的。 它们可能是真的或假的。 这当然是哥德尔的定理，但有一个扭曲的地方，即无法证明的定理无处不在。 由于长度为n的字符串中只有$2^n - 1$个比它短的字符串，大多数字符串无法被压缩，因此相关的定理是不可判定的。 虽然通过将它们作为新公理包含在更复杂的形式系统中（在描述上更长），这些定理可以变得可判定，但与可证明的定理相比，与不可证明的状态相对应的随机字符串的数量增长更快。 正如Chaitin$n$ [32] 所说：“如果有十磅的公理和二十磅的定理，那么那个定理无法从那些公理中推导出来。” 这使得Chaitin$n$推测数学应该接受这种不确定性并成为实验性的[31, 34, 36]。

随机性，哥德尔定理和柴廷对数学的观点在第3.7节中讨论。

1.4 真实世界系统

在关于自然系统的算法描述的讨论中，需要区分基于物理定律的持续计算产生的物理系统和仅从外部观察者的角度描述最终状态的系统。 沃尔夫勒姆[107]给出了一些细胞自动机的例子，这些自动机在简单规则下产生高度有规律的结构。 在经过固定步骤后生成结构的规则往往比仅描述结果的算法提供更短的描述。

描述结果的算法往往比仅描述结果的算法提供更短的描述。另一个例子是一群大雁飞行在V字形编队中（见《新科学家》[2]）。这种编队也可以由外部观察者开发的算法来描述。然而，这种模式的出现是因为每只大雁都优化其位置，以便与其他大雁保持足够的距离，以节省能量，并允许足够的视野向前看。形成的编队是每只大雁的计算过程的结果。作为熵度量的算法描述将是这两种计算描述中最短的描述。这将是每只大雁作为计算机进行的计算，或者是外部观察到的计算。然而，哪种描述是最小的可能取决于观察者感兴趣的自由度。在必须包括所有自由度的熵度量中，基于物理定律的持续计算产生的系统往往比仅从观察中得出的系统更加压缩。事实上，宇宙本身通常被认为是一个进行持续计算的计算机，通过所有有效状态的宇宙进行步进。然而，如果我们只对宇宙的一小部分感兴趣，算法描述可以比描述整个宇宙更简洁地压缩。

算法信息论在多种方式上有助于理解现实世界系统。虽然本书的主要方法是关于新兴性的，并将自下而上与自上而下的方法联系起来以理解系统，但也会提到其他贡献。

算法信息论提供了一套工具来解决一些深层次的哲学问题。
这些问题可能涉及信息和熵的保持，以及宇宙为何如此的问题。例如，保罗·戴维斯在《第五个奇迹》[44]中利用了算法信息论。下面将概述其中提出的一些问题和方法。本书还阐述了为什么科学还原主义是一种成功的理解方法的观点（见第10.3节）。简而言之，这是因为可观察宇宙的大部分可以用子程序嵌套描述算法。即宇宙似乎是算法上简单的。子程序映射的物理领域可以在子程序级别上进行研究。例如，受重力作用下下落的石头可以用简单的算法描述，因为通常可以忽略与宇宙的其他部分（包括观察者）的相互作用。描述这种行为的子程序具有较低的算法熵，因此可以通过我们非常有限的认知能力进行分析。如果宇宙不是算法上简单的，那么还原主义过程将毫无成果，因为人脑的计算能力本身也是宇宙的一部分，无法对物理系统做出任何意义的理解（参见Chaitin [35]）。一般来说，预计我们的大脑在算法上不够复杂（即使考虑到通过人脑的连接或利用计算机来扩展我们的大脑能力），无法确定属于更复杂算法的宇宙的命题的真假。然而，由于我们周围的宇宙似乎很大一部分可以

可以通过算法简单子程序来描述，还原论使我们能够理解整体的这个子集。

描述整个宇宙的字符串长度是令人难以置信的长。 显然，正如Calude和Meyerstein [21]指出的那样，字符串的某些部分可能在大尺度上呈有序状态-例如我们存在的尺度，然而在嵌入这个有序部分的整个时间和空间尺度上，字符串可能完全是随机的。 换句话说，我们宇宙的排序可能是一个更大的随机字符串中的波动，该字符串指定了一个更大的宇宙。 虽然可能如此，但本文认为，无论大尺度的秩序如何产生，算法信息论都可以用来展示宇宙的法则如何从现有的秩序中生成新的有序结构。 这在第9节中详细讨论。

然而，有一个根本问题；描述的算法测量需要算法停止。 例如，以下算法，逐个生成整数，非常简短，因为它不停止。

$$\begin{array}{ll} & N = 0 \\ 1. & N = N + 1 \\ & \text{打印} N \\ & \text{转到} \quad 1. \end{array} \tag{1.1}$$

然而，指定特定整数的算法在达到该整数时必须停止。它要长得多。 考虑以下生成 N_1的算法，例如可能是第1000000个整数，形式为

$$\begin{array}{l} \text{对于} N = 0 \text{ 到} N_1 \\ \quad N = N + 1 \\ \quad \text{下一个} N \\ \text{打印} N. \end{array} \tag{1.2}$$

需要大约 $log_2 N$位来指定 N_1. 例如，如果$N_1 = 1000000$，该算法比不停止地生成整数的程序要长约20位。 描述宇宙的算法也存在类似的问题。 宇宙似乎更像一个不终止的图灵机（至少目前是这样）。 我们观察到的模式是永远在变化的。

然而，正如遍历所有整数的算法需要一个停止指令来指定一个特定的整数一样，指定宇宙的一个特定配置或其中一部分在某个时间点需要一个停止指令。 在某个时间点强制算法停止的步骤数的规定，无论是隐式还是显式，与仅循环遍历所有算法相比，是非常大的。

宇宙的状态不停地变化。 正如在第6.3节中讨论的那样，指定停机状态的整数值是对于与初始状态相距较远但尚未达到平衡的宇宙特定配置的算法熵的主要贡献者。

第2章

计算和算法信息论

2.1 可行算法的计算要求

算法信息论通过生成定义结构的字符串的算法长度来定义对象的顺序或结构。

这个长度被称为科尔莫哥洛夫复杂度，算法复杂度或字符串的程序大小复杂度。 然而，有两种类型的代码可以用于指令和算法。 第一种是每个编码指令都需要结束标记，以告诉计算机何时结束一个指令并开始下一个。 正如后面所示，这种编码产生了纯粹的算法复杂度。 或者，当不允许一个代码成为另一个代码的前缀时，代码可以被瞬间读取，不需要结束标记。 这要求代码被限制在一组自解释指令或来自无前缀集合（Levin, 1974, Chaitin, 1975）。

使用自解释编码的算法描述的长度将被称为算法熵，因为在物理情况下，这比纯粹的算法复杂度更受青睐。

然而，显然算法的类型和长度取决于所使用的计算机。 幸运的是，出于两个原因，事实证明计算机的具体情况通常不是复杂性的重要指标。

首先，为通用图灵机编写的算法（即足够复杂以模拟任何其他计算机）可以与任何其他计算机相关联，一旦知道了模拟指令。 模拟算法的长度只是在给定通用机器上的算法上添加了一个固定长度的字符串。 其次，在科学中，只有熵差才是相关的物理度量，因为熵是状态的函数。 在使用相同计算机测量熵差时，任何模拟常数都会被取消。 因此，熵差是机器无关的。 此外，常见的算法指令，例如

那些涉及物理定律或描述常见物理情况的情况，如相空间的解析，也会被取消。 在这种情况下，算法熵与香农熵密切相关。 下面的章节涉及被称为图灵机（TM）的计算设备的理论。 对于那些想要跳过细节的人，主要观点是：

- 图灵机（TM）是生成任何可计算函数的标准过程（见2.2）。一组指令可以使人们能够从给定的输入明确计算出特定函数，这些指令可以为TM或等效于TM的任何计算设备编写。每个TM等效于一个函数。 例如，给定一个整数 n，存在一组指令可以将 n作为输入转换为函数 n^2作为输出。即实现这组指令的TM对应于函数 n^2。

- 有许多不同的计算过程和设备等价于图灵机。

- 图灵机只能在某些输入上停机。 如果机器不停机，则不会产生输出。 因为可以设计一个图灵机来检查数学定理的真假，所以不停机可能意味着计算需要运行更长时间或永远不会停机。 在这种情况下，该定理无法被证明为真或假。

- 通用图灵机（UTM）可以模拟任何图灵机或其他UTM。对科学家来说，典型的计算机是一台被编程的UTM（见第2.2节）。 通常希望使用更简单的UTM和更长的程序，其中程序指令是透明的。 另一种选择是使用复杂的UTM，其中大部分计算都体现在硬件或计算机的指令集中。由确定性物理定律驱动的经典宇宙是一个可逆UTM。这就是为什么我们可以使用计算机和合理的定律来映射宇宙行为的原因。

那些希望的人可以阅读下一节，那些可能想稍后回来的人可以跳到第3节。

2.2 图灵机

图灵机（TM）是一种简单的计算设备，更准确地说，是一种能够计算任何直观可计算函数的计算过程。 这被捕捉在丘奇-图灵论题中 - 任何有效的计算过程都可以用图灵机表示。通常，图灵机被认为是确定性的，因为每一步都有一个明确定义的输出，否则机器会停止。 TM可以描述如下（参见Casti [23]的实例）：

- 它有一个任意长的带子（向一个方向延伸），被分成单元格。 每个单元格包含来自有限字母表的符号。 其中一个符号对应于空白。 带子上的有限数量的非空白单元格表示输入数据。

- 图灵机具有一个可读写的头部，可以读取一个符号，向左或向右移动一个单元，并写入一个符号。 在每个计算步骤的开始，读写头都准备好读取当前单元的内容。
- 该机器处于有限状态集中的一种状态，这些状态确定了机器在每个步骤中的动作。
- 给定读取的符号和机器的状态，转换表显示了应在单元中写入的内容，机器的新状态以及头部是向右还是向左移动。即从旧配置到新配置的转换可以写成五元组的符号。

(当前状态，当前读取的符号) →(新写入的符号，向左/向右移动，新状态)。

(读者应该注意，一些定义通过限制机器在同一步骤中移动和写入的能力，将这些移动表示为四元组。)转换表实际上是确定机器进行的计算的程序。 不需要物理机器，可以在Excel电子表格上进行图灵计算，或者按照纸上的指示进行。 重要的是指令，而不是机器的结构体现了机器的本质。

所谓的机器从初始配置开始，在计算过程中通过不同的配置步骤，在每个步骤中根据转换表在磁带上读写。 如果当前配置没有有效的指令来执行，机器将停止，并且磁带上剩余的符号将成为计算输出。

图灵机使用两个或更多的符号。 如果使用二进制字母表，则需要以二进制形式对符号进行编码。 一种简单但低效的自然数编码形式是用零表示数字^n，并在其后跟随 $n+1$个1和另一个零，即01 n^{+1}0。 然后，零可以用010表示，5可以用01111110表示。 从某种意义上说，每个图灵机等效于一个单独的程序，并且在现代术语中，磁带充当机器的内存。 有等效的简单图灵机变体。Chaiti^n [27] 概述的双磁带机更直接地表示了算法复杂性测量的计算过程的基础。 该机器具有工作磁带和程序磁带。 工作磁带最初包含输入字符串 y。 计算可以表示为 $T(p, y) = x$，这意味着给定输入 y和程序 p，机器 T停止输出 x。

从时间角度来看，双带机器更高效，因为大部分指令都在程序带上，所以这台机器更像一个传统的计算机，输出由外部的程序控制。

通用图灵机

每个可能的图灵机可以被看作是实现特定算法的机器，也可以被看作是枚举函数的算法过程。 通用图灵机（UTM）是一种可以通过将模拟指令作为程序输入来模拟任何其他图灵机的机器。 正如已经提到的，图灵机由其五元组（或四元组）集合来定义。

这些五元组映射了转换表的前后状态。 当给定表示转换表的五元组时，UTM可以模拟图灵机。可以通过称为其哥德尔数的唯一数字来定义特定图灵机的五元组字符串。 表示五元组的字符串可以使用自限定编码（见第3.2节）进行编码。一个简单的例程可以通过按字典顺序遍历所有字符串来生成所有可能的五元组，并拒绝那些不表示有效转换的字符串。

这类似于一个例行程序，通过选择所有自然数，例如偶数。 第 i个五元组集合给出了Gödel数 i。 所有图灵机的集合是可数的，即存在一一对应关系与整数。 一旦给出了图灵机的Gödel数 i，它的转换表可以通过解码例程生成。 让 e成为这个简短的解码例程。 以 U表示的通用图灵机可以通过在p前加上一个模拟程序来模拟 $T_i(\ p)$。即 $U(e1_i0p) = T^i(p) = \ x$。 这个例程 e计算1的个数，直到遇到0。 这为通用图灵机生成从压缩编码版本的表中提取第$_i$个转换表提供了 i。 由于转换表封装了第i个机器，通用图灵机可以模拟第 i个机器。

虽然存在无限多个可能的通用图灵机，Marvin Minsky描述了一个7状态4符号的UTM [81]，而Stephen Wol-fram描述了一个2状态3颜色的图灵机，最近被Alex Smith [3]证明是通用的。

替代图灵机

有许多与图灵机等效或是图灵机的扩展的替代计算过程或过程。下面简要概述了这些。

1. 寄存器机。寄存器机是图灵等效的，其名称来自于其一个或多个“寄存器”，取代了图灵机的纸带和头。 每个寄存器保存一个正整数。

   寄存器通过简单的指令进行操作，如增加和

减少。 在简单条件下跳转到另一个指令，提供了编程的灵活性。

2. 标签系统。标签系统也可以看作是等效于图灵机的抽象机器。有时被称为Post标签机，因为它是由Emil Leon Post在1943年提出的。 它是一个有限状态机器，具有一个无限长度的纸带。 纸带符号形成一个先进先出队列，每一步机器读取队列头部的符号，删除固定数量的符号并添加符号。 可以构造一个标签机来模拟任何图灵机。

3. λ演算，由Church和Kleene在1930年代开发，是最小的通用编程语言。 它是一个形式系统，专门设计用于研究函数定义、函数应用和递归。 虽然它等价于图灵机，但它的重点是转换规则而不是计算步骤。

4. 可逆图灵机。 图灵机可以被设计成可逆的，其中过去状态的信息要么被存储，要么它的指令本身是可逆的。 由于物理定律是可逆的，它们只能通过可逆图灵机进行充分的模拟。这在第6节中讨论了物理上的含义。

5. 概率图灵机。通常，图灵机是确定性的，因为每一步都有明确定义的输出，否则机器会停止。 但是，可以设想一个概率图灵机，在计算的某些点上，机器可以随机访问0或1。 这样的机器的输出可以是随机的。 然而，由于没有有效的随机数生成器，只有伪概率机器似乎存在。 然而，Casti和Calude [25]提出使用量子数生成器来提供真正的随机输入。 他们在《新科学家》上的文章提出了一些有趣的真正概率图灵机的应用。然而，至少在目前，AIT仅限于确定性计算。

还应该注意，如果量子计算机成为可能（例如，NewScientists [43]），那么强可计算性论题将不适用，高效的量子算法可以在多项式时间内执行任务。

不可计算函数

每个图灵机可以被看作是将自然数作为输入映射到自然数作为输出的函数。这样的函数只有可数个。 然而，实数是不可数的，并且所有将自然数作为输入映射到自然数作为输出的函数包括实数，这些函数是不可数的。 由此可得

图灵机不足以计算每个函数。 因此，图灵机不能对所有输入停机。 如果图灵机在指定的有限步骤后停机，则该函数是可计算的。 然而，如果图灵机在指定的步骤中没有停机，则不确定它是否会在继续运行时停机。 图灵证明了[102]，将计算表述为决策算法时，停机问题意味着存在一些不可判定的命题。 停机问题的证明依赖于类似于Epimendides悖论“这个陈述是假”的自指算法。 因此，停机问题等价于哥德尔的不完全性定理，并导致Rice定理，该定理表明关于图灵机行为或输出的任何非平凡问题都是不可判定的。

部分递归函数或部分递归计算是可计算的。即表达函数的图灵机在某些输入上停机，但并非所有输入都停机。因此，图灵计算机可以被视为部分递归函数；一种将输入集的某些成员映射到输出集的过程。 另一方面，递归函数（或更严格地说，总递归函数）对所有参数（输入）都有定义，并且体现该过程的图灵机在所有输入上都停机。

可递归枚举或可计算枚举集是存在算法或决策过程的集合，当给定一个自然数或一组数时，如果该数或组数是集合的成员，则会停机。等效地，如果存在一种过程来列出集合的每个成员，则可以确定任何字符串是集合的成员。该集合实际上是部分递归函数的定义域。 然而，由于无法确定计算是否会停机，部分递归函数是半可判定的；任何计算的结果是是/可能是。所有有限集都是可判定和可递归枚举的，许多可数无限集也是如此，例如自然数。

这些概念应该与原始递归函数区分开来，原始递归函数是递归函数的一个子集。 这些原始递归函数是通过递归过程应用于几个简单函数而生成的。 科学中使用的大多数函数实际上是原始递归函数，因为它们可以通过计算来实现，其中包括一个在指定的循环次数后终止的“Do”或“FOR循环。

2.3 测度论

测度论已经发展起来，为可能是无限的集合（如实数集 $\Re$）上建立不同的测度提供了一个框架。
有用的测度包括概率、计数测度（子集中的元素数量）和称为勒贝格测度的距离测度。 在多维空间中，勒贝格测度成为该空间中的体积测度。 测度论的关键特点是它提供了工具

确定如何将分离子集上的测度与包含这些子集的较大集合的测度相关联。 例如，在计数测度的情况下，人们期望没有共同元素的分离子集中的元素总数等于每个子集中的元素之和。

集合及其成员的属性的一般度量将用μ表示. 度量必须具有适当的可加性，如上所述的计数度量. 对于所有可能的事件和事件的排列，度量也必须能够被定义. 为度量理论提供框架的关键是理解 Borel sigma 代数或 Borel 域. 这个框架只是一系列定义，用于操作集合的成员（例如实数集），以允许一致地定义概率等属性. Borel 代数是一种限制在度量空间 Ω 上的 sigma 代数；即，一个空间，例如实数上的开集合的集合，在其中存在一个距离函数. 记住这一点，度量空间的子集集合 $\mathcal{B}$of 具有以下属性:

- 空集合在 $\mathcal{B}$中
- 如果子集 B在 $\mathcal{B}$中，则其补集也在 $\mathcal{B}$中
- 对于任意数量的集合的集合在 $\mathcal{B}$中，它们的并集也必须在 $\mathcal{B}$中。即如果 B_1和 B_2在 $\mathcal{B}$中，则 $B_1 \cup B_2$也在 $\mathcal{B}$中，或者对于可数无限成员，
$$\bigcup_{i}^{\infty} B_i.$$

公理还暗示了集合的集合在可数交集下是封闭的，并且集合还包括完整集合 Ω。 对于无限实数，Borel σ代数定义了包含所有区间的最小σ代数族；但它不包括所有子集 Ω。 （所有子集的集合被称为幂集。）这个限制避免了Banach-Tarski悖论。根据这些对度量空间子集的属性的定义，Borel代数上的一般度量 μ，如概率或成员数量，定义如下。

- $\mu(\mathcal{B}) \geq 0$，并且对于空集 $\mu(\emptyset) = 0$
- 度量是可加的。
- 对于任何具有可数不相交集的集合 B_1, B_2, B_3的并集 $\mathcal{B}$of，(即$B_1 \cap B_2 = 0$)，则 $\mu(\mathcal{B}) = \sum$ 注意到B可以是可数的无限集合。

最后一条要点表明，如果（子）集之间没有共同元素，则度量只是每个子集的度量之和。

在度量 $\mu(\Omega) = 1$的整个样本空间的特殊情况下，度量被称为概率，通常用P表示。 度量定义的 P对应于科尔莫戈洛夫定义的

在AIT中使用的概率。然而，对于无限多个事件，Kolmogorov概率需要以下额外的公理（见[76]页18）：

- 对于一个递减的事件序列$B_1 \supset B_2 \supset B_n$，使得所有这些事件的交集为空；$\bigcap_n B_n = 0$，则极限概率为$\lim_{n\to\infty} P(B_n) = 0$。
  在极限情况下，概率为$\lim_{n\to\infty} P(B_n) = 0$。

科尔莫哥洛夫概率方法提供了一个比传统的“频率”方法更有用的概率基本定义。
然而，它具有一些反直觉的特点。 当这种方法应用于实数[0,1]的数线时，无理数集的概率为1，而一个特定的无理数的概率为零，因为无理数有无限多个。 由此可知，所有有理数集的概率为零。 换句话说，从数线上的所有数字集合中随机抽取一个有理数的科尔莫哥洛夫概率不仅几乎为零，而且是零。

柱状集合

并非所有的数字都是有理数。在科学中，一个测量值，例如指定一个原子的位置和动量坐标，当它被指定到一定的精度时，可以被视为有理数。

然而，有时候需要开发数学工具来处理无理数。 这些是十进制或二进制展开是无限的数字，因为它们不能被表示为简单分数的形式a/b，其中 a和 b是整数且 b不为零。 实数包括有理数和无理数。 通常认为实数集由位于0和1之间的无限数字构成，因为任何大于1的实数的性质取决于其在0和1之间的性质。 同样，使用二进制表示法，存在一个以序列 x开始的无限实数集，即数字上的数字

由$0.x$表示的线 . . . . . 圆柱集合的概念提供了一个方便处理这种序列的符号表示。圆柱集合允许指定属于数轴上相同区域的实数集合的公共前缀，即上面的示例中的 x。圆柱集合用Γ_x表示，并且是与小数点后的前缀 x相关联的集合。例如，二进制字符串$0.1011\ldots$表示以1011开头的无限序列的一种方式，并且用Γ_{1011}表示。圆柱集合 x表示从$0.x0^\infty$到$0.x1^\infty$的实数（即$0.x$到$0.x111\ldots$，其中 ∞表示重复的0或1）。就像十进制表示法中的$0.0\,0009^\infty = 0.0001$一样，在二进制表示法中，$0.00001^\infty$等于$0.0001$。可以看出，序列$0.00001^\infty$中的无限个1可以用$2^{-|0000|}$（$= 2^{-4}$）来替换。

一般来说，$0.x1^\infty$可以被替换为$0.x + 2^{-|x|}$。 （注意这里的竖线表示封闭数字的位数。）这使得人们可以看到Γ_x表示半开集合$[0.x$到$0.x + 2^{-|x|})$。对于那些不熟悉符号的人来说，

方括号“[”表示起点是集合的成员，而“)”表示终点不是。

对于基于实数集合的度量 μ，例如概率度量，$\mu(\Gamma_x) = \sum_b \mu_b(\Gamma_{xb})$，或者用简化符号表示 $\mu(x) = \sum \mu_b(xb)$。 这意味着，例如以前缀 x开始的圆柱集合上的概率度量是以该前缀开始的所有字符串的度量之和。 例如，用 P表示的概率度量为 $P(\Gamma_{10}) = P(\Gamma_{1010})+P(\Gamma_{1011})$。 有用的度量包括以下内容。

- 勒贝格测度 λ是实数之间的距离测度，并且在平移下是不变的。 对于实数线段$[a,b]$，这对应于我们直观的距离概念，且 $\lambda= (b - a)$。 勒贝格测度对于三维空间即 $\Re^3$是由一维空间的笛卡尔积形成的，是一个体积测度。

  在统计熵计算中，$6N$维等价物是用来量化相空间中的体积的测度。 勒贝格测度$\lambda(\Gamma_x)$是由表示为$[0.x$到$0.x+ 2^{-|x|})$的圆柱体上的测度。一个后面会用到的重要观点是，该测度对应于圆柱体两端的距离$2^{-|x|}$。例如，以空集 $\emptyset$表示的圆柱体$\Gamma_\emptyset$的勒贝格测度 $\lambda(\emptyset) = 1$，因为该圆柱体包含了整个数线从.$\emptyset$开始。$0000000\ldots$到 $.\emptyset1111\ldots$，即 $.00000\ldots$到 1。

- 科尔莫戈洛夫概率测度是在实数区间$[0,1]$上，以圆柱集合为子集的概率测度，在前面已经提到过。 它是一个一致的测度。
- 计数测度很直观；它是集合或子集 $\mathcal{E}$中的成员数量，并用 $|\mathcal{E}|$表示。

半测度。 半测度不具备真正测度的简单可加性质。 它是一种有缺陷的概率测度，因为定义被放宽了。 集合Ω的半测度 μ具有以下性质

$$\mu(\Omega) \leq 1.$$

在离散情况下，这对应于 $\Sigma_i \mu(s_i) \leq 1$。 对于在0到1之间的实数线上指定的连续半测度，其可加性质与真正测度不同。即

$$\mu(\Gamma_x) \geq \Sigma_b \mu(\Gamma_{xb}).$$

半测度在算法信息论中尤为重要作为$2^{-H(x)}$的半测度，其中 $H(x)$是算法熵。正如第3.2节所示， $\sum 2^{-H(x)} \leq 1$是由Kraft-Chaitin不等式导致的。

这种联系意味着该测度在处理停机概率方面很有用，并且在发展通用半测度的思想方面很有用，如第3.8节所述。半测度可以从下方计算，即可以从下方近似计算。 这可能对非数学家来说有点困难。

对于非数学家来说，这可能有点困难。 重要的是，集合论通过使用圆柱集合来表示具有相同初始序列的所有字符串，以一种能够处理无限集合的方式定义了多种不同的测度。

第三章

AIT和算法复杂性

正如在1.2节中指出的那样，算法信息论（AIT）最初是由Ray Solomonoff [93]提出的。 这种方法由Andrey Kolmogorov [69]和Gregory Chaitin [26]分别形式化。 这两位作者展示了如何在算法复杂性度量中大部分消除机器依赖性。 后来，这种方法被Leonard Levin [74, 75]和Gregory Chaitin [27]扩展到计算机指令的自限编码。 算法信息论（AIT）基于这样一个思想：具有可识别模式的高度有序结构可以比没有可识别特征的结构更简单地描述。 例如，表示π的序列可以通过一个相对较短的计算机程序生成。 这使得AIT方法能够提供一种形式工具来识别由字符串表示的模式或顺序。 在这种方法中，字符串“s”的算法复杂性被定义为能够生成该字符串的最短算法 p^*of长度 $|p^*|$的长度。 这里 $||$表示垂直线之间程序的位长度。 生成一个字符串的最短算法或程序的长度也被称为Kolmogorov复杂性或程序大小复杂性，而且它也是字符串中包含的信息的度量。 一个短程序意味着字符串 s是有序的，因为它很容易描述。 这与一个与字符串本身一样长且明显没有顺序的程序形成对比。 AIT对于自然科学的重要性在于，物理或生物系统总是可以转化为表示系统瞬时配置的数字字符串，该字符串位于适当的状态空间中。

举个例子，考虑如何表示运动场上N个运动员的瞬时配置。 如果允许跳跃，需要三个位置和三个速度维度来指定每个运动员的坐标。 对于N个运动员，瞬时配置是系统的6N维状态空间中的一个二进制字符串。 如果配置是有序的，例如所有运动员都在一条直线上奔跑，那么描述会比所有运动员随机放置并以不同速度和方向奔跑的情况简单。一个实际的

一组粒子在状态空间中的配置将指定粒子的位置、动量或等效状态以及电子和其他状态。 如果描述特定结构的字符串显示出有序、特征或模式，可以找到一个简短的算法描述来生成该字符串。

在实践中，字符串和算法通常以二进制形式表示。
一个简单的例子可能是自然发生的磁性材料，如天然磁铁。

每个铁原子的磁态可以通过其磁矩或自旋的方向来指定。 当自旋垂直对齐时，方向可以用1来表示，当自旋指向相反方向时用0来表示。

磁系统的瞬时配置可以通过一串0和1来表示，表示每个自旋的对齐情况。 在所谓的居里转变温度以上，所有自旋将随机对齐，没有净磁性。 在这种情况下，时间点的配置可以用随机的0和1序列来指定。 然而，在居里温度以下，自旋变得对齐时，结果高度有序的配置可以用一串1来表示，如下所示，并且可以用一个简短的算法来描述。

考虑以下两个序列，可能代表物理系统的配置。 第一个序列是一个由 N个重复的1组成的字符串，而第二个序列是一个由0和1组成的随机字符串。 这些可以被视为抛掷硬币 N次的结果，其中正面用1表示，反面用0表示。 在第一个情况下，硬币显然被篡改了，而在第二个情况下，硬币很可能是一个公平的硬币。 或者，这两个结果可以被视为磁系统中自旋的方向，如前一段所述。

1. 第一种情况是结果有序，由一串 N个1组成，即S=“11111. . .”。 由于这是高度有序的，可以用一个简单的算法P′来描述，形式如下；

$$\begin{array}{l} \text{对于} I \quad = 1 \text{ 到} N \\ \qquad \text{打印 “1”} \\ \text{下一个} I. \end{array} \tag{3.1}$$

在这种情况下，算法只需要编码数字 N，打印的字符
以及重复打印命令的循环指令。
在接下来的内容中，符号 |. . . . . |用于表示表示字符或计算指令的二进制字符串在竖线之间的长度。 在这种情况下，算法p’的长度为：

$$|p'| = |N| + |1| + |\text{打印} \qquad | + |\text{循环指令} \qquad |. \tag{3.2}$$

整数 N通常由其词典顺序表示的长度来指定。 这在第3.3节中详细讨论）。在

词典排序表示，$\mid N \mid = \lfloor \log_2 (N+1) \rfloor$. 在对数项周围的取整函数表示最大整数 $\leq \log_2 (N+1)$. 在接下来的内容中，$\log N$ ^ ，没有下标，用于表示这个整数 - 词典排序表示的长度。 如果 N 很大，算法将受到接近 $\log_2 N$ 位的 N 的长度的主导。 如果 p' 是在计算机 C 上生成 s 的二进制算法，那么 $\mid p^* \mid$，生成 s 的最短算法描述的大小必须 $\leq \mid p' \mid$.

2. 在这种情况下，字符串是由 N 个形式为“110010...1100”的字符表示的随机序列。 因为序列是随机的，只能通过一个逐个指定每个字符的二进制算法来生成。即

$$p' = P\ RIN\ T\ \text{“}s\text{”}.$$

如果 p^* 是最短的算法来完成这个任务，那么它的长度是

$$|p^*| \approx |p'| = |s| + |P\ RIN\ T|$$

由于这个算法的长度必须包括序列的长度，以及$P\ RIN\ T$指令的长度，所以它必须比序列长度稍微长一些。

在AIT框架中，最复杂的字符串是那些没有模式且不能用更短的算法来描述的字符串。 然而，如下所述，为了保持一致，需要使用标准的过程来编码指定结构的算法，并且尽量减少算法对计算机的依赖。 可以用两种类型的代码来进行计算指令。 第一种是每个编码指令都需要结束标记，以告诉计算机一个指令何时结束，下一个指令何时开始。 这种编码产生的纯算法复杂度表示为 $C(s)$。 另一种情况是当没有一个代码是另一个代码的前缀时，代码可以瞬间读取，不需要结束标记。 这要求代码被限制在一组自限定指令或来自前缀自由集合的指令中（Levin, 1974, Chaitin, 1975）。 使用这种编码的算法复杂度将表示为 $H(s)$，因为它是字符串s及其表示的配置 s的熵度量，因此被称为算法熵。

总结一下，任何显示有序的自然结构都可以通过一个短的算法来描述，而显示无序的结构只能通过指定每个字符来描述。

3.1 机器依赖性和不变性定理

关于实现程序的计算设备还没有提到任何内容程序 p^*。 幸运的是，如下所示，这种机器依赖性可以最小化。 设 p是在UTM上生成 s的程序，表示为

U，即 $U(p) = s$。 在机器$_U$上，算法复杂度度量 $C\,U(s)$ 是通过最短的程序 p 来实现的。

$$C_U(s) = min_{U(p)=s}\,|p|。 \tag{3.3}$$

当UTM根据输入字符串 y生成 s时，即 $U(p, y) = s$，度量变为

$$C_U(s|y) = min_{U(p,y)=s}\,|p|。 \tag{3.4}$$

由于任何通用图灵机都可以模拟其他机器上的计算，如果另一台通用图灵机 W实现了一个类似的程序来生成 s，那么

$$C_U(s) \leq C_W(s) + c, \tag{3.5}$$

其中常数 c考虑了需要指导机器 W模拟机器 U所需的额外项的长度。 常数 c 的数量级为1，因为它与生成的字符串无关。即 $c = O(1)$。 在实践中，可以在标准参考通用图灵机上评估算法复杂度。在大多数物理情况下， $O(1)$项可以忽略不计，因为只有差异才是重要的。

算法复杂度的度量，即生成字符串S的最短程序必须是

$$C(s) \leq C_U(s) + O(1). \tag{3.6}$$

当将字符串 y作为输入来计算S时，即 $U(p, y) = s$，不变性定理说明：

$$C(s|y) \leq C_U(s|y) + O(1). \tag{3.7}$$

这个定理在直觉上是合理的，因为可以建立一个简单的通用图灵机，并且一个解释器可以在另一台计算机上模拟一台计算机。 该定理的证明可以通过第2.2节中概述的方法变得健壮。该节展示了一个通用图灵机 U可以模拟第 i台图灵机 T_i。 设 $T_i(p^*) = s$。让机器 U读取一个字符串 e，该字符串中的短程序提取 i，并遍历每个图灵机的转换表以找到第^{i}个图灵机的转换表。有了这些信息， U可以模拟在第 i个机器上生成的。即 $U(e1\ i0\ p^*) = T^i(p^*) = s$。因此，使用图灵机 T_i的字符串的复杂度度量 $C_{Ti}(\ s))$ 是 $|p^*|$，而在 U上的度量是 $|e|$ $+_i$+2+$|p^*|$。因此， $C_U(s) = C_{Ti}(\ s))$+$sim(Ti)$，其中$sim(T_i) = |e| +_i+ 2$。 （常规情况下， e不包含在模拟字符串中，因为它通常被视为通用图灵机的定义的一部分。）使用一个具有最小指令集的简单通用图灵机作为参考机器是有意义的。

这种方法将大部分指令封装在程序中，因为复杂的程序指令可以从最小的指令集编译而来。 出于这个原因，Chaitin使用LISP作为程序语言。

他已经证明对于LISP编程，模拟或 $O(1)$项是

大约300位的顺序。 Li和Vit´anyi（第210页）使用基于组合逻辑的λ演算

$$C(x) \leq |x| + 8 \text{和 } C(\emptyset) = 2。$$

Li和Vit´anyi（第107页）给出了 $C(s)$的另一种两部分定义

$$C(s) = min_{i,p}\{|T_i| + |p| : T_i(p) = s\} + O(1) \tag{3.8}$$

其中最小化是在所有图灵机和计算程序上进行的，计算出的。 这个定义最小化了程序长度和实现程序的图灵计算机的描述的组合。

可以将大部分描述放在图灵机中，这样程序会很短，或者将大部分描述放在程序中，拥有一个简单的图灵机。显然会有一个最佳描述。 正如在第4.2节中讨论的那样，当所有模式都包含在机器描述中，所有随机结构都在程序中时，最佳情况发生。。

上述定义的算法复杂性将被称为“纯算法复杂性”，以区别于后来的定义，该定义将编码字限制为自解释的编码字；即没有关于一个编码字何时结束和另一个编码字开始的歧义（第3.2节）。

AIT的问题

Crutchfield和他的合作者（Shalizi [91] Feldman, [42]）指出了以下与AIT方法测量字符串复杂性相关的困难。

- 无法事先知道字符串 $s$has是否有更短的描述。 即使找到了一个更短的描述， s'也不能确定它是否是最短的描述。 这等同于图灵停机问题。

- AIT方法必须明确指定一个单一的字符串。 从观察者的角度来看，似乎必须精确指定与观察模式无关的信息，包括噪声，这使得描述过长，并且对大多数观察到的模式可能没有用处。

- 运行AIT算法所需的计算资源（时间和存储）可能会无限增长。

- 对于随机序列，复杂性最大。

- 使用UTM可能会限制区分可以由更弱的计算模型描述的系统和UTM之间的能力，而且UTM缺乏唯一性和最小性。

- AIT方法无法捕捉代数对称性。

前两个要点在引言1中已经讨论过。 停机问题是一个基本问题，使用其他复杂度度量只是掩盖了问题。 AIT能够包含字符串中的噪声和变异，这不再是一个问题。 正如在第4节中进一步讨论的那样，临时熵的概念或算法最小充分统计量绕过了这个被认为的问题。 以下要点讨论了剩下的问题。

- 第三个要点取决于适用于特定应用的复杂度度量。 对于许多物理情况，简单的AIT提供了洞察力。 然而，[13]提出了逻辑深度的附加概念，以捕捉与资源使用相关的问题，而其他几位作者则讨论了尺寸和资源之间的权衡[17，77，104]。

- 第四个要点是一个定义问题，批评者们也承认这一点。

- 第五个要点是因为Crutchfield [39]能够证明随机过程可以通过一系列机器来解释。 比图灵机简单的过程可以用比它简单得多的计算设备来描述。 如果我们使用算法复杂性的定义，即优化计算设备描述和程序的总和（如方程（3.8）所示），那么这并不是一个问题。每当一个简单的图灵机捕捉到模式时，较简单设备的描述就会很短。 我们始终需要注意与唯一性和最小性相关的问题。 但对于许多物理情况，例如测量系统的算法熵，差异而不是绝对值是关键的度量，这些问题并不重要。

- 作者不确定如何理解最后一个要点。 除非这个观点被误解了。 在识别代数对称性和缩放对称性的情况下，任何识别对称性的算法描述都会比不识别对称性的描述更短。

Crutchfield和他的同事[41, 39]开发了因果状态方法
to提供一个系统的过程来突出一个稳态随机
系列中的结构。 然而，AIT的目的与因果状态方法相反，
是描述模式，而不是发现模式。因此，任何能够突出
迄今未观察到的结构的模式识别过程都可以被
纳入AIT描述中。 这些包括因果状态方法，
实际的最小描述长度观点（见第7节）和其他
系统性过程，无论是基于对称性还是物理理解。
一旦通过任何方式确定了模式，进一步的压缩就
是可能的。

3.2 自限定编码和Kraft不等式

实现算法的机器必须识别一条指令的结束和另一条指令的开始。 这可以通过使用结束标记来简单地完成，以分隔代码或指令。 然而，正如Lev$_i$n [75]，G´acs [55]和Cha$_i$t$_i$n [27]所示，如果每个指令或数字的编码来自于一个前缀自由集合，则有许多优点。在这样的集合中，没有编码可以是任何其他编码的前缀，也不需要结束标记。 在这种情况下，这些编码可以称为瞬时编码，因为它们可以在读取时解码，或者称为自限定编码，因为编码指示了其自身的长度。 它们有时被称为前缀编码，这似乎令人困惑，因为严格来说，它们来自于一个前缀自由集合。正如已经提到的，Kraft不等式对于自限定编码成立。 让前缀自由集合中第$_i$个编码的编码长度为 $|x\ i|$；那么对于二进制编码，Kraft不等式的表述如下：

$$\sum_i 2^{-|xi|} \leq 1. \tag{3.9}$$

这可以推广到其他字母表。

通过生成所有可能的编码字的树来理解这个不等式（见图1）。 这个过程有点像把一个正方形切成两半，将一半写一个长度为1的编码，将另一半保留给更长的编码。 由于没有编码是其他编码的前缀，因此只能分配一个给定长度的编码。 例如，如果011被分配为长度为3的编码，任何以011开头的编码都不符合要求。 只能通过使用未使用的长度为3的编码010，并将其扩展为0101和0100来生成更多的编码。 这些长度为4的扩展中只能使用其中一个，并且另一个可以进一步扩展以生成更长的编码。 由于每个编码长度只有一个分配的编码，所以分配的编码的总和为 $\sum_i 2^{-|xi|}$最多为$_2{}^1 + {}_4{}^1 + {}_8{}^1$等，最多等于^1。 图^{1}说明了码字需要满足克拉夫特不等式的要求。 从图^{1}可以看出，每当分配一个码字时，为了确保没有一个码字是另一个码字的前缀，分配的码字不能与之前的码字或后续的码字直接连接。 因此，任何以已分配码字开头的二叉树分支必须被阻断，因为该分支的剩余部分不能被使用。

每当分配一个码字，例如0111和/或0110，如图1所示，对克拉夫特和的贡献最多为$2^{-|0111|} + 2^{-|0110|}$。这最多是$2^{-3}$的贡献；如果分配了父码字‘011’，它将对和的贡献相同。 由于每次分割时贡献减半。 检查显示总和永远不会大于1。

Kraft-McMillan不等式的逆命题也成立。 这个不等式表明，只要码字的大小满足上述不等式，集合中的每个码字都可以被一个同样长度的瞬时码替换，而且也满足这个不等式。

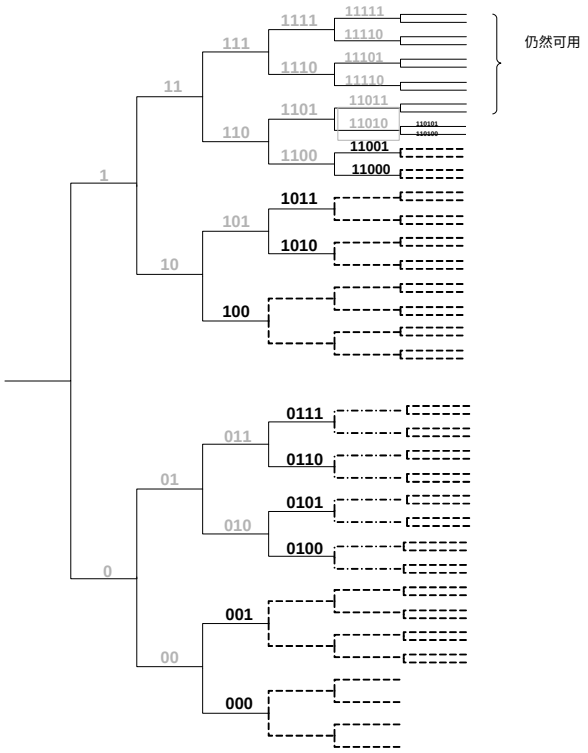

图3.1：选择二进制码，使得没有选中的码是另一个码的前缀

这种方法可以扩展到完整的程序，而不仅仅是指令。 如果算法没有结束标记，那么没有停机算法（例如表示算法的字符串 x）可以是任何其他算法（例如字符串 xy）的前缀，因为从左边开始读取，机器会在进行计算之前停机。 因此，如果没有结束标记，算法也必须来自无前缀集合。由于可能的算法数量是无限的，Kraft不等式需要扩展到无限无前缀算法集合。 Chaitin [27]证明了这个扩展，称为Kraft-Chaitin不等式，适用于无限递归可枚举集合。

Chaitin的论证认识到，每个分配的长度为 $|p|$的算法$p$of禁止使用以 p开头的任何其他算法。 如果将这个算法设想为$0.pxxx\ldots$它在0和1之间的数线上阻塞了$2^{-|p|}$的区域。即$0.p0$和$0.p1$之间的区域。 所有这些算法的长度之和最多可以等于1，即数线上的长度。

更正式地说，考虑计算机 $T(p) = x$，其中程序 p是一个自然数的二进制形式。 如果 p是一个自限定程序，没有其他程序可以以 p开头。 如果考虑所有介于[0,1]之间的实数，以$0.p$开头的无限字符串集合必须被阻塞并且不能使用。

用于另一个程序。 第2.3节介绍了圆柱集的概念，其中圆柱体Γ_p表示$0.p0^\infty$到$0.p1^\infty$之间的实数，覆盖了所有这些不允许的程序。 正如第2.3节所示，这个圆柱体上的勒贝格或距离测度 $\lambda(\Gamma_p)$等于$2^{-|p|}$。 如果所有的 p都来自一个无前缀集合，那么所有的圆柱集都不会重叠（即它们是不相交的）。 测度性质意味着对所有可能的无限 p的求和满足 $\sum_p \lambda(\Gamma_p) = \sum_p 2^{-|p|}$。这个求和不能大于1，即所有实数[0,1]上的勒贝格测度的值。 即克拉夫特等式成立，即 $\sum_p 2^{-|p|} \leq 1$，当所有圆柱集合的并集完全覆盖实数时，等式成立。

使用来自前缀自由集合或自限编码的代码还有另一个优点。 在这种情况下，克拉夫特不等式确保两个独立的停机算法可以通过插入适当的连接指令连接在一起。 Bennett [13]以以下方式概述了这一点（见3.4）。如果 U（p，w）= x并停机，且 U（q，x）= y，则U（rpq，w）= y，其中 r是将第一个例程的输出作为第二个例程的输入的指令。 最多，连接代码会将组合例程的长度增加O（1）。

在物理情况下，使用在第3.1节中概述的纯算法复杂度不如使用将码字限制为前缀自由集合的定义更有用。相应的算法复杂度表示为 K（x）。

$$K_U(x|y) = min_{U(p,y)=x} |p|$$

其中 U只接受来自前缀自由集合的字符串。因此

$$K(x) \leq K_U(x) + O(1). \tag{3.10}$$

还有

$$K(x|y) \leq K_U(x|y) + O(1), \tag{3.11}$$

在后一种情况下，输入 y被输入到计算过程中。 在实践中，人们在标准参考计算机上评估 $K_U(x)$，有时称为Chaitin计算机（参见Calude [18]），知道真正的算法复杂性最多为O(1)。 只要算法复杂性的比较是在同一参考计算机上进行的，O(1)项的意义就不大。

然后可以定义算法熵为前缀自由或自我分隔的算法复杂性，即

$$H(x) \equiv K(x)$$

给定输入字符串y的条件熵为

$$H(x|y) \equiv K(x|y).$$

在物理情况下，熵是首选的概念，因为它避免了对复杂性一词含义的不确定性，其次，正如所示

稍后（第5.2节），算法熵几乎与Shan-non熵相同。 此外，互信息、条件熵等概念自然而然地出现。 因为算法熵是Kraft和的算法长度，所以算法熵也必须满足Kraft不等式；即$\sum$

$^{\text{所有}x\in}$以下$^{H(x)}$或$^{H}_{a}\mathrm{lgo}^{(x)}$将表示算法熵或算法复杂性，并且通常优先使用，而不是$K^{(x)}$。 集合的Sh_{a}nnon熵将被表示为^{H}_sh。

3.3 最佳编码和香农无噪声编码定理

如果想要高效地编码一条消息，使用出现频率最高的消息符号的最短编码是有意义的。 香农无噪声编码定理展示了如何实现这一点。 考虑一组消息符号s_1, s_2, s_3, s_4, s_5,.... s_k是预期消息中出现的符号，其概率为p_1, p_2, p_3, p_4, p_5, ... p_k。 . 让这些符号的自限定或前缀无重复二进制编码词为 $code($ $s_1)$, $code(s_2)$, $code(s_3)$, $code(s_4)$, $code(s_5)$, ...code(s_k)。香农-法诺编码是一种满足克拉夫特不等式且受到以下约束的高效编码方法：

$$-log_2 p_k \leq |code(s_k)| \leq 1 - log_2 p_k. \tag{3.12}$$

这意味着 $2^{-|code(s_k)|} \leq p_k \leq 2^{1-|code(s_k)|}$. Shannon-Fano 编码可以通过将消息符号从最可能到最不可能的顺序进行排序来实现。 将列表分成两半，使得每半部分的总概率尽可能相等。 将最可能的组分配为 0，将其他组分配为 1。 这是代码的第一个数字。 对每个子组依次重复此过程，分配进一步的 0 或 1，直到每个概率都被分配。 通过这种方式进行分割，该过程确保没有任何代码是其他代码的前缀，并且代码长度不超过$-log_2 p_i + 1$。

一种替代方法，霍夫曼编码将概率组合成更高效的编码。 算术编码[82]是一种更高效的基于熵的编码，原则上略微更好。 因此，给定消息中符号的概率，编码的平均长度可以非常接近源的香农熵；即基于符号的预期出现的熵。 以这种方式编码的消息中每个符号的预期码长（即 $\sum p_k|code(s_k)|$）由以下公式给出

$$H_{sh} \leq \sum p_k|code(s_k)| \leq H_{sh} + 1 \tag{3.13}$$

其中 $H_{sh} = \overline{k}p\rangle\!\rangle log_2 p_k$是源符号集的香农熵。 因此，给定符号的概率分布，由 N个符号组成的消息的预期长度可以接近$N\,H_{sh}$。这个方程被称为香农的无噪声编码定理。一个编码

每个符号的预期码长等于熵，对于给定的概率分布是最优的，无法改进。

下面的小节展示了如何在不知道自然数的概率分布的情况下，对自然数进行最优编码。 在第3.8节中将讨论香农编码定理的算法等价物。

自然数的分隔编码

由于算法熵要求指令是自限定的，因此需要考虑如何对自然数进行自限定编码。 使用前一节中概述的香农的无噪声编码定理，对于具有已知概率分布的有限指令集或数字，分配自限定编码是直接的。 程序可以唯一地读取每个编码，然后使用表格或解码函数进行解码。 然而，通常需要将自然数编码为算法的输入，例如为了提供计算循环中的步骤数。

在这个过程中存在两个问题。 第一个问题是传统的二进制表示自然数时存在歧义，不清楚字符串'010'是否与字符串'10'相同。 通过使用词典（字典）排序对数字进行编码可以避免二进制表示的歧义。例如，整数从0到6等按词典顺序编码为$\emptyset, 0, 1, 00, 01, 10, 11,$等等，其中 $\emptyset$是空字符串。 可以看出，自然数 n的词典等价物是 $n+1$的二进制等价物，去掉前导的1。 这相当于 $|n| = \lfloor log_2(n+1) \rfloor$，其中方程中的地板函数就是L形括号中项的下方整数。 这里将用 $l\hat{o}gn$ 表示，即 $|n| = l\hat{o}gn$ ，注意这最多比 $log_2 n$少一位。 第二个问题是，如果没有数字的概率分布，无法使用香农的定理，必须找到另一种方法使代码自限定。

虽然词典编码不是自限定的，但可以通过在词典表示前面添加一个前缀字符串，该字符串由一串^{n}个1和一个0组成，来构造一种低效的自限定指令。该前缀字符串用于指示编码大小 n。 更正式地说，如果^{n}在词典形式中表示一个自然数，那么一个可能的自限定通用编码是$1^{|}\ n^{|}0^n$，或者（使用 $l\hat{o}g$符号）$1^{lo}{}^{gn}\ 0\ n$。 编码的长度变为$2l\hat{o}gn + 1$。例如，数字5在词典中表示为‘10’，在这种通用的自限定形式中变为11010（即两个1用于标识‘10’的长度，该长度是表示数字5的编码，然后是一个0，最后是10）。 读取该编码的机器计算在遇到‘0’之前的‘1’的数量。 ‘1’的数量（例如示例中的2）告诉解码器要读取跟在‘0’后面的编码的长度。

虽然这段代码是通用的，因为它优化了平均代码长度，不论未知的概率分布如何，但这是一个冗长的表示，不是渐近最优的（Li和Vit´anyi [77] p.78,79）。

Ca*l*ude [18] 第23页有一个更紧凑的自限定表示，长度为$1 + \log n + 2\log nl + \log n\hat{} + 2\log nl\hat{}$ bits。 或者，可以通过迭代过程生成更紧凑的代码。 如果 $\mathrm{code}_1(n) = 1^{|n|}0n$，则第一次迭代给出 $\mathrm{code}_2(n) = \mathrm{code}_1(n)n$。 与Ca*l*ude的情况一样，这个长度为$1 + \log n + 2\log nl + \log n\hat{} + 2\log nl\hat{}$ bits。 进一步迭代，其中 $\mathrm{code}_3(n) = \mathrm{code}_2(n)n$，长度为$1 + \log n + \log l + \log n\hat{} + 2\log l\hat{} + \log g\hat{}\,1$。 原则上存在一个长度由无穷级数给出的代码，用于$n$。

$$1 + l\hat{o}gn + l\hat{o}gnl\hat{o}gn + l\hat{o}gl\hat{o}gl\hat{o}gn \ldots$$

在实践中，表示 n的大小将$\leq 1 + l\hat{o}gn + l\hat{o}gl\ \hat{o}gn + 2l\hat{o}gl\ \hat{}og$或更高的扩展。 自限定编码所需的附加信息可以解释为 $H(|n|)$，即指定 n大小的算法长度。

一个重要的问题是，对于自然数的代码，长度信息有多重要来确定算法熵。 是否总是需要$\log\log$和更高阶的项？ 答案是，在热力学系统的平衡（或典型）状态中，长度信息是微不足道的。例如，1摩尔玻尔兹曼气体中大约有 $N = 10^{23} = 2^{69}$个粒子。算法熵和平衡状态需要指定每个2^69个粒子的位置和动量到所需的精度。 假设每个粒子的位置和动量坐标都定义为2^8的一部分。 这需要每个粒子16位。 对于2^69个粒子，需要16×2^69位。 因此，平衡配置的算法熵约为2^73位。 第一个捕捉长度信息的$loglog$项为 $log_2 2^73 = 73$。 这对算法熵的贡献微不足道。 系列中的下一个项是 $log_2(73)$，它更小。 然而，当一个特定的字符串高度有序时，算法描述将更短，需要包括更高的 $loglog$项的贡献。

这里的讨论与第4.1节中讨论的状态集中字符串的临时熵的评估相关。临时熵由两部分代码组成，一部分定义了展示特定结构或模式的字符串集合，另一部分标识了集合中的字符串。在这种情况下，可以将长度信息体现在描述集合的方式中。

与熵、条件熵等的关系

一个重要的问题是何时使用单个算法同时计算两个字符串x和y比分别计算更高效。换句话说，何时

$$H(x, y) < H(x) + H(y)?$$

如果x和y是独立的，那么在单个算法中计算它们不会带来任何收益，但如果y对x有信息，那么先计算y将导致对给定y的x的描述更短。在这种情况下

在一般情况下，香农熵的定义为 $H_{sh}(X,Y) = H_{sh}(Y) + H_{sh}(X|Y)$。 如果随机变量 X不依赖于 Y，那么知道 Y不会减少对 X的不确定性，即 $H(X|Y) = H(X)$。 在这种情况下，香农熵的定义为 $H_{sh}(X,Y) = H_{sh}(Y) + H_{sh}(X)$。[1]类似但不完全相同的关系也出现在算法熵中。 如果要同时描述字符串 x和 y，了解y可能会简化对 x的描述。 然而，与香农熵相反，Chaitin [31]表明，除非使用压缩描述 y^*，否则需要进行算法修正。 假设参考计算机在给定空字符串 $\emptyset$和最小程序 y^*的情况下进行计算，即$U(\ y^*, \emptyset) = y$。 如果将 y^*而不是 y作为输入提供给生成最小程序 x^*的计算机，那么在给定 y^*的情况下，$U(\ x^*, y^*) = x$。 由于可以通过了解 y^*来计算 x和 y，

$$H(x,\ y) = H(y) + H(x|y^*) + O(1)$$

注意 y^*包含关于 y和最短描述长度的信息。 如果使用 y而不是 y^*，则需要将 y的最压缩描述长度传递给计算 x，即

$$H(x,\ y) = H(y) + H(x|y,\ H(y)) + O(1)。$$

上述方程认识到，在许多情况下，一旦知道字符串 y的最小描述，可以更高效地计算字符串 x。 在这种情况下， $H(x|y^*) \leq H(x)$ 表明 x和 y可以更高效地一起计算。也就是说，

$$H(x,\ y) \leq H(x) + H(y) + O(1).$$

由于计算 x可以通过比计算 x和 y两者都要长的程序更短的程序来实现，所以 $H(x) \leq H(x,y)$。一般来说，

$$H(x) \leq H(y) + H(x|y^*) + O(1). \tag{3.14}$$

当完整描述 y^*对于计算 x是必要的时候，等号是必要的。 这发生在 y必须作为计算 x的最短算法的一部分时。 例如，当 y^*表示在物理情况下计算 x所需的表达物理定律的子程序的压缩版本时，等号是必要的。 一般来说，方程3.14显示程序可以以亚可加的方式连接或连接到O(1)。在这里， $O(1)$项是连接两个独立算法所需的额外位数，如3.2中所述，并在3.4中进一步讨论。

Shannon已经定义了两个集合X和Y之间的互信息为$I(X{:}Y) = H(X) + H(Y)$ - $H(X,Y)$。互信息是一个衡量当其中一个变量已知时，有多少位的不确定性被消除的度量。
在算法情况下，两个字符串之间的互信息为

1. 对于Shannon熵，使用大写标签来表示随机变量的集合。

表示为$I(x{:}y)$（或$H(x{:}y)$）。 给定x的最小程序，互信息$I(x{:}y)$可以定义为：

$$I(x{:}y) = H(x{:}y) \equiv H(x) \text{ - } H(x|y^{\star})$$

或者

$$\text{我}(x:y) \equiv H(x) + H(y) - H(x,\, y) + O(1)$$

读者应该注意，互信息的定义在不同的作者之间可能会有些不同。 例如，Chaitin在参考文献[31, 28]中使用了第二个方程式定义了一个对称的互信息，但没有包含O(1)项。 在这种情况下，O(1)项被转移到了第一个方程式中。 然而，在更基础的论文[27]中，Chaitin使用了上述的定义。 然而，对于大字符串的度量来说，这个定义的差异是微不足道的。 算法熵与香农熵之间的显著相似性不容忽视。 这些也不应被视为完全相同。 允许使用 y^{*}代替 y和O(1)项，它们看起来是相同的。然而，香农的关系式展示了通过了解一个相关的随机变量，可以减少观察随机变量的不确定性。 算法关系式展示了计算一个字符串如何简化计算相关字符串中的共同信息。 算法表达式涉及到两个字符串之间的实际共享信息。 从上述算法结果中还可以得出其他重要的结果。

$$H(x,\, y) = H(y,\, x) + O(1),$$

$$H(x) \leq |x| + H(|x|) + O(1)$$

对于典型或接近随机的字符串，等式成立。

$$H(x) = H(|x|) + H(x||x|) + O(1),$$

对于长度为 n的随机字符串的情况，

$$H(x) = H(n) + H(x|n) + O(1).$$

在算法信息论中，一个字符串所包含的信息，即指定系统或结构状态所需的最小位数。 这是直观的，因此信息 $I(x)$ 等同于熵，即

$$I(x) = H(x).$$

第3.5节定义了一个停机概率 Q。 虽然 Q不是一个真正的概率，但两个特定字符串 x和 y之间的算法互信息可以用 Q来定义。即

$$I(x:y) = log_2(Q(x,\, y)/(Q(x)Q(y)) + O(1).$$

互信息的期望值与使用概率的香农定义相同，但使用 $Q(x, y)$等等。

3.4 相对于常见框架的熵

如果 y作为生成 x的输入是必要的，并且 q^*是生成 x的例程，那么（$U(q^*, y) = x$）。要求 y是必要的是要求 y在生成 x的计算过程中必须被生成。然而，例程 p和 q不能直接组合，因为它们是自限定的。换句话说，字符串 p^*q^*不是一个有效的算法。

如果我们将计算顺序从最左边的例程开始，那么运行p^*q^*的计算将在运行 p^*时终止，并且只给出输出y。然而，一个小的链接例程， r^*的O(1)次方，[13, 27]允许通过形式为 $U(r^*p^*q^*) = x$的组合程序生成 x。这里的*符号用于指示这些是参考UTM上的最小程序。算法熵是由$|p^*| + |q^*| + |r^*|$给出的组合熵。对于这种特殊情况，其中生成 x是必要的， $H(x) = H(x, y)$成立。字符串 x的算法熵可以表示为 $H(x) = H(\ y)+H(x|y, H(y))+O(1)$，因为在生成 x的过程中必须生成y。然而，这个简单的加法表达式只涵盖了最优压缩的基本例程的情况。在字符串 y不是生成 x的基本输入的情况下， $H(x) \leq H(x, y)$且 $\leq H(\ y)+H(x|y, H(y))+O(1)$。由于熵差异而不是绝对熵具有物理意义，在任何物理情况下，共同信息字符串，实际上是共同指令，可以视为已知。实际上，这些信息起到了前一段中例程 p^*的作用。共同信息可以包括等效于“PRINT”和“FOR/NEXT”等的共同指令的二进制规范，以及参考UTM所隐含的O(1)的不确定性。在物理情况下，共同信息可以包括物理定律以及热力学系统的相空间粒度的算法描述[108]。在接下来的内容中，共同信息例程将用‘CI表示，并且在给定共同信息的情况下，物理上显著的熵将用条件熵 $H_{algo}(x|CI)$表示。由于两个配置 x和 x'之间的熵差异是 $H_{algo}(x|CI) - H_{algo}(x'|CI)$，熵为零可以是 $H(CI)$，而常见的贡献被忽略。

有序序列之所以被认可，是因为隐含地引用了一个可以递归枚举的模式化集合。这个集合的任何序列 x都可以至少部分压缩（见第4节），而主要的熵贡献将取决于字符串的长度和系统中可能状态的数量等参数。这个模式化集合的成员的条件算法熵将被称为“临时”熵。临时熵是模式化集合的典型成员的熵；它是成员的条件算法的上界度量。然而，这个集合中非典型的成员很少可以进一步压缩。每当一个集合的成员可以进一步压缩时，就需要一个更精细的模型来捕捉那些特定字符串中的模式。这在第4.2节的算法最小充分统计量中进一步讨论。

3.5 熵和概率

算法停机概率 $Q_U(x)$是指在参考$_U$TM上随机生成的程序字符串停机的概率，用$_U$表示，输出为x[27]。 具有停机概率 $Q\ U(\ x)$的随机生成程序可以看作是通过抛硬币生成的0和1的序列。 在这种情况下， $Q_U(x)$的定义为

$$Q_U(x) = \sum_{U(p)=x} 2^{-|p|} \tag{3.15}$$

所有这些概率的总和，$\Omega_U\ = \sum$ 由于每个生成x的算法都来自一个前缀自由集合，根据Kraft-Chaitin不等式，$\Omega_U \leq 1$。 然而，由于$\Omega_U = 1$， $Q_U($ $_x$)不能是真正的概率，而是一个半测度。 此外，通过除以Ω_U来归一化 $Q_U(x)$没有任何收益，因为Ω_U是不可计算的。 即使如此，这个概念允许算法熵和停机概率之间建立一个非常重要的联系。 论证如下。

- $Q_U(x)\ \geq\ 2^{-H_U(x)}$ 作为 $H_U(x)$ 是生成 x 的最小 $|p|$ ，而定义 $Q_U(\ x)$ 的总和包括生成x的所有程序。 基本上，在 $Q_U(\ x)$ 的总和中生成 x的最短程序是 $H(x)$，并且是一个主要贡献。 这导致了不等式，

$$H_U(x) \geq -log_2 Q_U(x) \tag{3.16}$$

- 但是， $H(x)$ 不能比 $-log_2 Q_U(x)$ 大太多，因为只有很少的短程序能够生成 x。 所有更长的描述对总和的贡献很小，如下所示。 因此，也可以证明

$$H_U(x) \leq \lceil -log_2 Q(x) \rceil +\ c \tag{3.17}$$

由于方程3.16，必须遵循

$$H_U(x) = -log_2 Q(x) +\ c. \tag{3.18}$$

上限 $H_U(x)$（方程（3.17））可以通过注意到（Downey私人通信） $H_U(x)$是一个整数， $H_U(x) \geq \lceil -log_2 Q_U(x) \rceil$来看到。但是 $\lceil -log_2 Q_U(x) \rceil$也可以用于生成字符串 x的算法中。 更具体地说，一个例程 p接受 $\lceil -log_2 Q_U(x) \rceil$并解码为字符串 x，可以衡量 x的算法复杂性。 基本上，例程 p取 $-log_2 Q(x)$的整数值，并从字典顺序中最低的长度开始，逐步遍历所有允许的例程，以查看哪个例程生成字符串 x。 如果没有例程生成x，则程序将长度增加一，并再次遍历该长度下的所有允许例程。 （这是一个思想实验，因为某些例程可能不会停止。然而，我们只对能够停止的例程感兴趣。）

最终，接近于 -$\log_2 Q(x)$ 的一个数将生成s。 根据定义，这个例程的长度不能小于$H_{U}(x)$。也就是说，程序p的长度$|p| = \lceil$-$\log_2 Q(x)\rceil + |stepping\,\text{routine}|$ 是算法熵或复杂性的上限度量。 根据上面的第二个要点，可以得出$H_{U}(x) \leq |p|$ 或

$$H_U(x) \leq \lceil -log_2 Q(x)\rceil + \ c \tag{3.19}$$

其中c代表解码例程的长度。 这个上限可以与方程(3.16)结合，得到以下关系；

$$\text{-}log_2 Q(x) \leq H\_{U}(x) \leq \lceil\text{-}log_2 Q(x)\rceil + c.$$

方程（3.18）的推导包括四舍五入的分数 $-log_2 Q$（x）在常数 c中。 注意O（1）的常数 c不依赖于 x。 此外，由于它在不同的UTM中只会略微变化，可以忽略表示特定UTM的后缀 U。可以看出常数 c表示与 H（x）接近的2^c个程序。 如果有2^c个与 H（x）接近大小的程序，并且忽略长度显著较长的程序，则有 Q（x）$\leq 2^c 2^{-H}$（x）。 这意味着上述所述的 H（x）， H_U（x）$\leq -lo g_2 Q_U$（x）$+ c$。 实际上，常数2^c成为对 Q（x）做出重要贡献的少数短程序的度量。

$Q(x)$是一个未归一化的概率度量，表示一个字符串在UTM上产生x的概率。然而，可以将其视为通过将未使用的概率空间分配给一个虚拟结果来表示概率。 这使得 $Q(x)$可以被视为一个特定消息 x在消息集合中的概率度量。

相对于这个概率，对于一个消息或字符串集合， $H(x)$的期望值接近于 $\langle H(x)\rangle = -\sum$ 这表明前一段落中的方程式3.19是Shannon编码定理的算法等价物。 一个字符串的算法编码，即其算法熵，与该字符串的停机概率只相差一个常数。

这再次强调了之前所示的，算法熵和Shannon熵之间的关系。

一种更一般的方法来证明相同的关系，依赖于一个通用半测度的性质，将在第3.8节中概述。

3.6 所有知识的源泉：Chaitin的Ω

Chaitin [27]提出了这个问题：“从一个前缀自由程序集中随机选择一个程序，在特定的UTM上运行时，它停机并给出给定的输出的概率是多少 x？”第3.5节将 Q_U定义为这个概率。 严格来说，这是一个有缺陷的概率，应该称为半测度（见第3.5节）。停机概率Ω_U通过对由随机输入生成的所有停机结果 x的 Q_U求和得到。

$$\Omega_U = \sum_{U(p)\text{停机}} 2^{-|p|} = \sum_x Q_U(x)。$$

Ω_U介于0和1之间。 由于Ω包含了所有可能的停机算法，如果宇宙遵循可计算的规律，Ω包含了关于宇宙的所有可以知道的东西；它包含了所有可能的知识。 虽然Ω的值取决于计算机，但可以在UTM之间进行转换。不幸的是，只有通过运行所有可能的程序来查看它们是否停机，才能确定Ω_U的所有贡献。 每个停机的 k比特程序 p（即$|p| = k$）对Ω_U贡献2^{-k}比特。 在Chaitin或Martin-L ¨of意义上，Ω是随机的（第8.2节）。如果Ω可以由一个更简单的程序生成，那么就可以预测一个对Ω有贡献的特定程序是否会停机。 这是不可能的，因为这样做将违反图灵停机定理。 Chaitin（给这个数命名为Ω）证明了，就像没有可计算的函数能够事先确定计算机是否会停机一样，也没有可计算的函数能够确定Ω的小数位。

Ω是不可计算的，只能通过检查从最小到最大的每个可能的程序来近似确定在给定时间内可能停止的程序。

3.7 G¨odel定理和形式公理系统

Chaitin [30]已经发展出了算法等价于哥德尔定理。 他已经证明了一些可能的定理或命题不能从一个更简单的公理和规则集中推导出来。 由于这些命题用一个不能通过公理压缩的字符串表示，它们是不可判定的。

它们可能是真的或假的。 这当然是哥德尔的定理，但是不可证明的定理无处不在。 由于长度为n的字符串中最多只有$2^n - 1$个比它短的字符串，所以没有足够的字符串来压缩大多数字符串。 因此，相关的定理是不可判定的。

考虑一个与乘法一样丰富且要求既一致又正确的形式逻辑公理系统。 在这样的系统中，没有一个陈述既可以是真的又可以是假的，只有真陈述可以被证明为真。 为了使用AIT探索Chaitin的方法，让‘A’表示系统的公理的符号字符串。 描述这些公理的最短算法的长度用 $|A|$表示。

所有可证明定理的集合是递归可枚举的。 这意味着可以通过一个按长度递增顺序遍历所有可能的定理证明的过程，并使用一个证明检查算法来确定生成的特定陈述是否是一个有效的定理。

原则上，每个可证明的定理都可以从公理加上推理规则生成。 那些可能表示定理但没有更短描述的字符串不属于递归可枚举的真定理集。 由于它们不能用公理和推理规则来表达，所以无法证明它们是真还是假。它们是不可判定的。 属于定理集的陈述远远多于有效的可能定理。

更糟糕的是，随着可能的定理字符串长度的增加，与不可证明陈述相关的字符串集增长速度比可证明陈述的速度更快。
一个人永远不能确定一个潜在的定理是否可判定 - 只有当找到一个证明（或其逆证明）时，才会知道。也就是说，只有当一个潜在的定理可以用更简单的定理（或公理）和推理规则的压缩描述来表达时，该定理才能被认为是被证明的。 这当然是哥德尔的定理，但有一个令人不安的扭曲，即不可证明的定理无处不在；可证明的定理只是所有可能定理空间的一个小部分。 另一种看待这个问题的方式是将论证与停机问题联系起来。 停机问题的不可判定性是一个丰富形式系统的不可判定性的基础。

上述论证可以更加健壮。 将字符串 x的长度表示为 n，其中语句“$H(x) \geq n$”是一个需要在形式系统中证明的语句。 这个语句断言存在一个字符串 x，其算法复杂度至少与其长度一样大。 这样的字符串将被视为随机的，因为它的算法描述至少与其长度一样长。 生成随机字符串的计算算法需要指定每个数字以及依赖于计算机的一些开销。 已经有人认为许多这样的字符串必须存在，但问题是，是否可以找到并证明一个算法复杂度至少等于 n的字符串是真实存在的。也就是说，给定 n，是否可能找到一个特定的 x，它是随机的或者没有比其长度更短的描述？ 有两种看待这个问题的方式。 第一种是开发一个过程，逐步通过公理和推理法则推导出所有有效的证明，直到算法证明 p找到第一个 x，使得

正如已经提到的，这类似于逐个遍历所有整数，直到找到具有特定属性的整数。 证明 p将包含公理，体现算法逐步检查证明的过程，以及指示感兴趣的字符串何时被找到的规范。 由于算法中的 n的大小可以用 $log_2\, n$位表示，因此该算法的长度为：

$$|p| = |A| + log_2 n + |\text{步进算法和证明检查器}| + O(1). \quad (3.20)$$

这意味着存在一个计算过程T，可以找到 p使得 $T(p) = x$。 但是 $|p|$不能小于由 $H(x)$定义的最短描述，否则 p可以用来提供一个更压缩的定义of x。 如果 c被视为系统描述和步进算法的长度，并且因为 $|p|$不能小于对 x的最短描述，那么就有

$$H(x) \leq |p| = log_2 n + c.$$

在这里，有限常数 c已经吸收了方程3.20中固定字符串的长度。这意味着已经找到了一个输出第一个 x的证明，其中 $H(x) \geq n$。 但是由于 $log_2 n + c > H(x)$，所以 $log_2 n + c \geq n$。 但是这变得不可能，因为对于任何给定的有限 c，总会存在一个 n

足够大，以便$log_2\ n \geq c.$对于大的n，这样的证明搜索永远无法停止，因为找不到这样的证明。

一个稍微不同的论证更符合原始的G¨odel方法，在系统内部对形式系统进行元陈述编码。
让我们尝试证明元陈述存在一个 n，使得陈述“$H(x) \geq n$不可证明”。 这个陈述在形式系统中要么为真，要么为假。 假设这个陈述是假的，这意味着我们可以找到一个相反的例子 - 一个值为 x的x，使得 $H(\mathrm{x}) \geq n$。 就像在之前的论证中一样，如果系统是完备的，因为不能存在矛盾，对于足够大的n，其中$n > c$，就找不到这样的字符串 p。再次

$$log_2 n + c \geq n。$$

我们无法证明这个陈述是假的，因此原始陈述必须是不可判定的。

虽然这些不良陈述可以通过将它们作为新公理包含在更复杂的形式系统中（在描述更长的意义上）而变得可判定，但与可证明的陈述相对应的随机字符串的数量增长速度比可证明的陈述更快。 正如Chaitin [32]所说：“如果一个人有十磅的公理和二十磅的定理，那么那个定理不能从那些公理中推导出来”。 这使得Chaitin推测数学应该接受这种不确定性并成为实验性的[31, 34, 36]。

这可以告诉我们一些关于柴廷的神秘数字Ω的信息。 Ωcan be constructed fr om the provable theorems in a logical system - there is nocontribution from und ecidable theorems as the generating computation will nothalt for these. 如果Ω代表了包含所有物理定律的逻辑系统，那么了解Ω就等于了解一切。 如果Ω是可计算的，所有的物理定律都可以简单地表达出来。 有趣的是，Calude、Din neen和Shu [20]计算了Ω的64位二进制值，得到了Ω =

0.0000001000000100000110001000011010001111110010111011101000010000... . . ..

虽然在任何形式系统中都存在一些不可判定的数学命题，但是确定这些命题是哪些是不可能的，尽管这些不可判定的命题随着命题描述长度的增加而增长得更快。 如果假设一个可能的不可判定命题是真的，形式系统可以被扩展。 然而，总是存在这样的危险，即命题实际上是可判定的，而假设的公理与形式系统不一致。 这导致柴廷提出[34, 32]数学应该成为实验性的；有用的数学假设可以被认为是真实的，除非后来出现了不一致性。 例如，黎曼猜想可能是不可判定的（尽管在《新科学家》的Kvaalen [70]文章中提到它可能是可证明的），但实验上似乎很可能是真实的。 然后，这个假设可以被作为额外的公理，可能会有更多新的有趣的定理可以使用这个新的公理证明。

这还有一个进一步的含义，许多人一直在努力解决。 当进行逻辑推理时，人类思维是否像有限图灵机一样运作？

思考？ 如果是这样，它将受到哥德尔定理的限制。 Barrow第8章[8]概述了辩论中一些关键人物的观点，而最近Feferman [51]讨论了历史发展。 强决定论和可计算性周围的特定问题已经被Calude等人[19]研究过。 对于现在的作者来说，虽然人脑或思维不需要像有限图灵机一样来发现宇宙的新真理，但形式逻辑过程似乎是证明这些真理的必要条件。 在这种情况下，证明过程而不是发现过程将受到哥德尔定理的限制。 虽然人类思维可以添加新的定理并使用外部设备扩展其计算能力，但一旦公理和推理法则变得过于庞大，思维可能无法保证形式过程的一致性，因为其复杂性不足。 因此，可能存在一些超出人类理解范围的定律，这些定律无法从更简单的原则中推导出来或以更简单的术语理解。显然，并非所有人都同意。

3.8 算法编码定理，通用的半测度，先验和推理

在方程（3.12）中已经证明了对于一个可枚举的概率分布P，其中源符号的概率为 p_1，$\mathrm{p}_2 ... p_k ..$等等，对于每个符号s_j，可以找到一组最优编码 $code(\ s_j)$。由这些符号组成的消息的期望编码长度，给出了香农的无噪声编码定理

$$H_{sh} \leq \sum p_k |code(s_k)| \leq H_{sh} + 1.$$

即编码消息的期望长度在香农熵的一位之内。 正如第3.5节所示，算法信息论有一个编码定理，称为算法编码定理，它是香农无噪声编码定理的等价物。本节介绍了一种基于建立最优的通用半测度的算法编码定理的替代方法。就像可以有一个在固定边界内可以模拟任何其他图灵机的通用图灵机一样，可以有一个在编码消息时与源词分布无关的渐近最优的通用编码。 如下所述，并且在第3.5节中暗示，对于 x的这个编码由半测度给出

这两者在 $O(1)$常数范围内是等价的。

该推导基于一个事实，即存在一个通用可枚举的半测度，它是离散样本空间的最优码。 从乘法上讲，它是最优的，因为它必须优于所有其他可构造的半测度。也就是说，它对一个结果的概率分配比任何其他可计算的分布都要高。 换句话说，如果$m(x)$是通用可枚举的半测度，那么$m(x) \geq c\mu(x)$，其中$\mu(x)$是离散样本空间中的任何其他半测度。 这一点的重要性将变得明显，但首先应注意到这个通用属性。

仅适用于可枚举半测度，而不适用于所有半测度的类或更受限制的递归半测度类（参见Li和Vit´anyi第246页[76]和Hutter [64]）。 G´acs [57]已经证明 $m(x)$ 也是可构造的（即可从下方计算）。

虽然有可数无穷多个通用半测度，但普遍停机概率 $Q(x)$可以作为参考通用半测度。 原则上，所有半测度都可以通过建立停机概率 $Q_T(x) = \sum$来枚举。

$_{T_i(p)=x}$ 2对于每个图灵机 T_i，$^{-|p|}$都成立。让普遍停机概率 $Q(x^{)}$为可以模拟任何图灵机的通用机器定义。 所有产生 $Q(x^{)}$的随机输入集将包括模拟每个图灵机和每个运行在每个图灵机 T_i上的程序 p。 由于 $Q(x^{)}$包括每个半测度，它在乘法上支配它们所有。因此，普遍概率也是一个通用半测度。 然而，$2^{-H(x)}$也是一个构造性半测度，因为 $H(x^{)}$等于 $-log_2Q(x^{)}$加上一个常数，所以$2^{-H(x)}$等于 $Q(x^{)}$乘以一个乘法常数。 $^{H}(x^{)}$也必须是一个通用半测度。. 换句话说，半测度 $m(x^{)}$，$Q(x^{)}$和2^{-} $H^{(x)}$都是通用的，并且它们彼此之间相等，只是乘法常数不同。 因此，它们的对数在加法上支配彼此。

这导致了编码定理的算法等价性[75]。 以下等式在一个加法常数 c_p或者一个O(1)项的范围内成立。

$$H(x) = -log_2m(x) = -log_2Q(x)$$

这种方法提供了方程(3.18)的另一种推导。出于实际原因，采用通用半测度 $m($ $x) = 2^{-H(x)}$是方便的。

Shannon编码定理(3.12)表明，对于给定的分布$P(x)$，可以构造一个二进制自限定编码，平均而言，生成的代码长度接近 $log_2P($ $x)$，这是该概率分布的最优长度。 然而，这样的编码对于所有的概率分布都不是通用的。 相比之下，算法编码定理表明，对于任何可枚举分布（即可以指定给定数量的有效数字的分布），都存在一个通用编码。 这个编码词对于x是生成x的最短算法。 它的长度是 $H($ $x)$或者 $log_2Q(x)$，与x无关的一个加法常数 c_p。

通用可枚举半测度 $m(x)$ 是对情况的最大无知的度量。 它将最大概率分配给所有对象（因为它在乘法常数上支配其他分布），实际上是一个通用分布。 作为一个通用分布，通用半测度提供了定义集合中典型成员的洞察力，并为基于贝叶斯原理的归纳提供了基础，如下一节所述。

推理和贝叶斯统计以及通用分布

拉普拉斯发展了被称为中立原则或不充分原因原则的概念，这个概念最早由雅各布·伯努利提出。这个概念提供了一个处理可能有几个可能原因的情况的工具。 因为没有理由偏好一个原因而不是其他任何原因，该原则规定它们都应该被视为等可能的。即在没有任何证据表明抛硬币是有偏差的情况下，该原则会给予正面和反面相等的概率。 [2]该原则导致贝叶斯定理，该定理确定了在给定信息 D的情况下，假设 $\mathcal{H}_i$ 的可能性有多大。即给定数据的假设的概率是

$$P(\mathcal{H}_i|D) = P(D|\mathcal{H}_i)P(\mathcal{H}_i)/P(D)。 \tag{3.21}$$

这个规则实际上只是条件概率定义的一个重写，为推理提供了基础。 如果 $\mathcal{H}_i$ 表示可能解释数据 D的假设或模型的成员，则推断的概率或后验概率定义为 $P(\mathcal{H}_i|D)$；即给定数据的特定假设的概率。 如果一个特定假设$\mathcal{H}_i$的概率 $P(\ \mathcal{H}_i|D)$明显大于其他假设的概率，那么有一定的信心认为 $\mathcal{H}_i$是更可靠的假设。也就是说，在观察到数据 D后， $P(\mathcal{H}_i|D)$衡量了数据与假设的一致性。 然而，贝叶斯方法似乎没有太大价值，除非对于所有可能性，都知道 $P(\mathcal{H}_i)$，即 $\mathcal{H}_i$ 是正确假设的概率。也就是说，需要准确衡量每个可能假设的 $P(\mathcal{H}_i)$，并确保 $\sum_i P(\mathcal{H}_i) = 1.$

然而，如果在没有任何数据的情况下存在某个假设 $P(\mathcal{H}_i)$ 的概率估计，那么这个估计可以通过使用贝叶斯定理来衡量给定数据的特定假设的一致性。在这种情况下， $P(\mathcal{H}_i)$ 被称为先验概率或初始概率，而结果概率 $P(\mathcal{H}_i|D)$ 则成为给定数据的推断概率。 一旦对一组假设建立了 $P(\mathcal{H}_i|D)$ 的估计，这些信息可以作为新的先验用于对新数据D' 的 $P(\mathcal{H}_i|\ D')$ 进行进一步估计。 如果这个迭代过程继续下去， $P(\mathcal{H}_i|D)$ 将根据证据或数据的不同，通常会偏向某个假设。 因此，这个推理过程尽可能地利用了可用的数据。 然而，从频率学派的角度来看，这种方法存在一个主要困难。 在频率学派的框架中，需要大量的样本结果来推断概率测度。 对于一次性事件，无论是基于无差别原则还是基于可能原因的估计，都没有为其分配概率的地方。尽管如此

[2] 当然，这种方法没有考虑错误的代价
没有偏见。 虽然每个原因可以假设是等可能的，但人们需要一种投注或保险策略来应对隐藏的偏见，这将在后面讨论。

贝叶斯统计学现在被广泛使用，杰恩斯将这一原则概括为最大熵原理（第7节）。

结果发现，通用半测度 $m(\mathcal{H}_i) = 2^{-H(\mathcal{H}_i)}$，提供了先验概率 $P(\mathcal{H}_i)$的最佳估计。 简单来说，当 $P(\mathcal{H}_i)$未知时，可以将$2^{-H(\mathcal{H}_i)}$视为先验概率的最佳猜测。 此外，Vit´anyi和Li [104]认为，最优压缩几乎总是识别假设的最佳策略。 上述论点在第7节进一步讨论了最小描述级别方法来进行模型拟合。

通用半测度和推理

最初，Solomonoff [93]在处理类似的推理问题时，开发了实际上是算法方法的方法。 Solomonoff认为，一系列测量结果以及实验方法论一起，提供了输入贝叶斯规则以给出下一个数据点的最佳估计的数据集。 Solomonoff提出了使用通用分布作为贝叶斯先验概率的概念；即初始概率估计。

然而，Solomonoff的初始方法遇到了问题，因为他没有使用自限定或来自前缀自由集的编码。Levin [75]通过将通用分布定义为自限定算法来解决了这个问题。 因此，第3.8节中的通用分布可以作为离散样本空间的通用先验概率。 Levin认为，通用分布在所有分布中具有最大的概率，并且可以从下方近似计算。 有效地 $m(x)$最大化了无知，或者概括了不关心的原则，因此是分配先验概率的最不偏见的方法。 G´acs [57]使用$2^{-H(x)}$作为通用分布（见第8.2节）证明了一个引人注目的性质，即在给定通用半测度的情况下，对于 x的随机性测试 $d(x|m)$显示所有结果都是随机的（见第8.2节中的讨论）。如果真实分布是 $\mu(x)$，则 $m(x)$和 $\mu(x)$彼此足够接近，以便可以使用 $m(x)$代替 $\mu(x)$。 因此，在真实分布未知的假设检验中可以使用 $m(x)$。 换句话说，通用分布作为先验几乎和精确分布一样好。 这证明了在贝叶斯规则中使用通用分布作为先验的合理性，前提是所有假设的真实分布是可计算的。 实际上，正如Li和Vit´anyi [76]在第323页指出的那样，这个过程既符合奥卡姆剃刀原则，又符合伊壁鸠鲁的原则。 奥卡姆剃刀原则认为，不应该超出必要的限度增加实体；实际上是说，简单的解释最好。 另一方面，伊壁鸠鲁认为，如果有几个理论与数据一致，那么应该保留所有理论（Li和Vit´anyi [76]第315页）。 因为任何概率大于零的假设都包含在通用分布中作为先验，而且该方法更加重视最简单的解释，所以该方法同时满足奥卡姆和伊壁鸠鲁的原则。

Solomonoff的主要兴趣在于预测可能的下一个成员

无限序列（参见Li和Vit´anyi [76]中对基础历史的引用）。 然而，无限序列需要一种修改过的方法，这里将以概要形式进行概述，以保证完整性。 然而，初次阅读者可能会发现忽略以下段落更方便。

在无限情况下，输出不仅仅是 x，而是以 x开头的所有无限序列的输出。这些以x开头的所有序列构成了称为圆柱集（见2.3）的集合，用Γ_x表示。 与离散情况相反，由于输出以特定程序输出起始序列 x的UTM可能不会最终停止，因此可能的非停机程序也必须包含在定义半测度的求和中。 作为结果，UTM必须是一个单调机器；即定义为具有单向输入和输出带（以及工作带）的UTM。 连续情况下的通用半测度为；

$$M_U(x) = \sum_{U(p)=x*} 2^{-|p|} \tag{3.22}$$

其中 M_U表示使用单调机器。 这里$*$表示
输出以 x开头，计算可能不会真正停止。
这个方程定义了 $M_U(x)$ 作为以硬币投掷生成的程序 p的输出以x开头的概率。 连续的通用半测度 M具有类似但不完全相同的性质，与离散情况类似。此外，

$-log_2 M(x) = H(x) - O(1) = Km(x) - O(1)$，其中 $Km(x)$ 是单调机器上运行的最短程序的长度，该程序输出以 x开头的字符串。 例如，如果序列的开头是字符串 x，则该方法暗示序列中的字符按照可计算的测度 μ分布。 在这种情况下，下一个元素是 y的概率 μ为

$$\mu(y|x) = \mu(x, y)/\mu(x).$$

基于连续通用半测度 $M(x)$ 的估计结果具有相同的形式;

$$M(y|x) = M(x, y)/M(x).$$

这个方程迅速收敛到上面的方程，Solomonoff已经证明[94]，在进行了n次预测之后，使用 $M(x)$ 而不是 $\mu(x)$ 的预期误差下降速度比$1/n$ 快。

第4章

表现出有序性但具有变化或噪声的字符串的算法熵

由于大多数现实世界的结构既具有结构又具有随机性，如果算法信息论能够为自然系统提供洞察力，它必须能够处理捕捉结构但也展现变化的不同字符串。 这些字符串都展现了结构的模式和可以称为噪声的特定字符串的变化（见第3.1节中的讨论）。例如，表示指定噪声图像的像素的字符串将具有一些像素捕捉图像，而其他像素只是噪声。 由于算法熵是从描述字符串的算法中导出的，包括结构和任何噪声或随机性，人们认为随机成分会支配生成算法的长度，从而掩盖任何模式[90]。然而事实并非如此。如下所述，我们只需要将字符串与所有相似字符串中的模式进行匹配。 在这种情况下，噪声对算法熵的贡献等于通过识别特定字符串的不确定性所暗示的香农熵。 三种等效方法不仅解决了噪声字符串的问题，还将算法方法与香农的信息论和统计热力学更清晰地联系起来。 这三种不同的等效表述被称为；

- 临时熵方法,
- 算法最小充分统计方法，以及
- 最小描述级别的理想形式。

前两个将在下面讨论，第三个将在第7节中详细讨论。 此外，Zurek的物理熵[108]，它是算法熵和Shannon熵的混合，返回相同的值作为这三个

等效的表述。 尽管在概念上不同，但由于下面的原因，可以认为它们是相同的。

4.1 临时熵

Devine [46]已经表明，在识别出噪声序列中的模式时，隐含地引用了包含所有类似字符串的集合，这些字符串展示了该模式。 由于这个集合是递归可枚举的，可以通过由两个例程组成的算法生成展示这个模式的噪声字符串。 第一个例程枚举了展示共同模式或结构并包含感兴趣字符串的字符串集合 S。 第二个例程根据集合确定感兴趣的字符串。特定字符串 x_i在集合中的算法度量将被称为“临时算法熵”，表示为 $H_{prov}(x_i)$。 Devine [46]使用了“揭示熵”而不是“临时熵”这个短语，以表示该值取决于观察到的或揭示的模式。 “临时熵”这个术语似乎更可取，因为观察到的模式可能不能全面解释字符串中的所有结构。 该度量仅是临时的，因为可能存在更深层次的未观察到的模式。 然后，这个两部分的描述变为

$$H_{prov}(x_i) \leq H_{algo}(S) + H_{algo}(x_i|S) + O(1).$$

这里 $H_{algo}(S)$ 是描述集合 S 及其特征模式的最短自限定程序的长度，而 $H_{algo}(x_i|S)$ 是选择集合中的 x_i的最短自限定程序的长度。第二项对算法熵的贡献在实践中与集合的香农熵相同。如果集合中有Ω_S个元素，并且所有成员等可能地出现，每个字符串可以用长度为 $|\Omega|$的代码来识别。 在这种情况下，忽略由于自限定编码而产生的额外贡献 $H_{algo}(x_i|S) = log_2\Omega_s$. 这当然是集合的香农熵。临时熵则变为；

$$H_{prov}(x_i) = H_{algo}(S) + \log_2\Omega_S + O(1). \tag{4.2}$$

正如在3.3节中讨论的那样，如果自限定编码所需的信息可以内置到定义集合成员的算法中，那么它可以被输入到识别集合中实际字符串的算法中。实际上；

$$\begin{aligned} H_{prov}(x_i) \quad &= H_{algo}(\text{描述集合的算法} \quad \text{'的结构} \quad) \\ &+ H_{algo}(\text{识别集合中的字符串}). \end{aligned} \tag{4.3}$$

第一个贡献是描述集合的算法的熵，
而第二个贡献等同于集合的香农熵。
在这种情况下，方程（4.2）变为

$$H_{prov}(s_j) \simeq H(S) + H_{sh} \tag{4.4}$$

正如Shannon熵是识别集合中每个成员的不确定性的度量，等效的临时熵术语需要与Shannon度量相同数量的位来识别给定集合的特定成员 x ofthe set give n the set's characteristics. 换句话说，临时熵是模式化集合的典型成员的熵。

作为算法熵的上限度量，根据可用信息，临时熵是实际熵的最佳估计，表示例如物理系统的微状态的特定嘈杂模式化字符串的熵。 如果所有结构都被识别，临时算法熵将对应于真实的算法熵，解决了第3.1节中提出的问题。然而，由于无法确定是否已经识别出所有结构，最小描述

此外，即使典型字符串的所有结构都被识别出来，仍然会有一些高度有序的非典型字符串可能显示出更多的结构。 这些字符串的算法熵也较低。 "临时"的名称表示可能存在未被识别的结构。 然而，讨论表明，在1.2节中提出的对算法方法的担忧是没有根据的。 虽然算法确切地指定了一个特定的字符串，但变异或噪声并没有淹没算法。 $\mathrm{Zu}_r\mathrm{ek}$的物理熵，在第5.2节中讨论，包括给定有限信息的状态的最短算法描述，以及一个$\mathrm{Shann}_o\mathrm{n}$项来指定剩余的不确定性。与上述讨论不同，物理熵的$\mathrm{Shann}_o\mathrm{n}$项不是从一个算法中导出的。 虽然物理熵不是一个完整的算法度量，但它的值与临时熵相同，因为识别集合中的算法度量几乎与$\mathrm{Shann}_o\mathrm{n}$熵的值相同。

在集合中编码第j个字符串

定义临时熵的两部分算法需要一个例程来识别一组具有相同模式或结构的相似字符串中感兴趣的字符串。 本节将详细介绍如何指定特定字符串以使过程更具体。

为了在相似字符串集合中识别字符串，特定字符串的算法编码必须能够解码。 为简单起见，假设集合中展示该模式的所有Ω_s个字符串具有相同的长度。每个字符串可以按字典顺序排列并分配一个依赖于该顺序的代码。 如果所有字符串的概率都是$1/(\Omega_s)$，根据方程(3.12)，第j个字符串的最小代码长度将为$|code(\ j)\ | = \lceil log_2 \Omega_s \rceil$。

这个长度将被视为 N。 这个实际问题可以通过一个简单的例子来看出，假设集合中有 ${}_{22}0$ 个成员。在这种情况下，$\log_2({}_{22}0) = 7.78$，这意味着集合中的所有字符串可以通过一个由 8 位数字组成的代码来识别，范围在 00000000从第一个成员的代码为 1，到第 ${}_{22}0$ 个成员的代码为 11011011（即 ${}_{22}0$-1 的二进制）。一般来说，有序列表中的第 j 个序列由 $00000...$ 编码。$0 + j$-1 的二进制表示。 虽然这些代码中没有一个是另一个代码的前缀，正如之前讨论的那样，但是算法需要关于 N 的信息以及 N 的大小，以确定何时读取完所有字符。

生成特定字符串的算法需要包括以下内容。 算法需要包含以下内容。

- 代码代码(s_j) 用于 $j$th字符串的代码。 这将有长度 $N = |\Omega_s|$，接近于 $log_2\Omega_s$。这个代码是子程序的关键 它能够识别出字符串 j的特征集。
- 算法还必须隐含或明确地知道代码(s_j) 的长度，以便知道何时读取完代码 。 这是由 $H(N)$ 给出的它指定了 N的大小。 要么长度信息必须与代码(s_j) 关联，使得自解码代码的长度为 $H(N) + |$代码$(j)|$，要么它必须成为 一条指令的一部分，指示要读取多少代码。 后者是下面算法的情况。 长 度为 $H(N)$ 的代码提供了足够的信息让计算机知道何时读取完代码(s_j) 。

- 下面概述的程序需要以初始字符串0000开始，长度为N，存储在计算机中 。从算法的角度来看 需要$H(N)$来指定程序开始时的$STORE$例程。即，$|STORE('0', N)| = |0| + |N| + O(1)$。
- 一种模式字符串集合的规范，可以方便地作为子程序'CRITERIA'，如果 一个字符串是该集合的成员，则输出为'YES'，否则为'NO'。 实际上， $H(S) = |\mathrm{CRIT\ ERIA}|$

生成第j个字符串的程序的形式如下...

$$
\begin{aligned}
&INPUTH(N)\ \{\text{指定需要读取代码}(j)\text{的位数}\}\ INPUTcode(sj)\ \{\text{长}\\
&\text{度为}H(N)\text{的位数}\}\\
&\qquad s = STORE(0,\ N)\ \{\text{创建一个由}N\text{个零组成的字符串。}\}\\
&\qquad FOR\ I = 0\ to\ code(s_j)\\
&\qquad A.\ \ GOSUB\ CRITERIA\ \{\text{输出}\quad = NO\text{或}YES\}\\
&\qquad 1\ \ NO\quad s = s + 1\\
&\qquad\quad \text{转到}A\\
&\qquad\quad YES\quad s = s + 1\\
&\qquad\quad NEXT\ I\\
&\qquad PRINT\ \ s - 1\ \{\text{需要因为循环已经超出范围}\}
\end{aligned}
\tag{4.5}
$$

步骤A指定了定义模式集的标准。该算法遍历所有可能的序列，计算符合 标准的序列的数量。
即，递增 i直到达到第j个模式字符串。 因此，字符串的临时算法熵可以表示 为：

$$H_{prov}(s_j) = |code(j)| + H(N) + |\text{定义模式集的标准}| + O(1).$$

当集合成员数量很大时，这将减少为；

$H_{prov} \simeq log_2\Omega_s + log_2 N + (log_2 log_2 N \ldots) + |0| + |$定义模式化集合的标准$| + O(1).$

这里的O(1)项包括标准指令，如STORE，PRINT，GO-
SUB等。如果需要，熵的零可以被选择为给定的这些项。
包含 $log_2 log_2 N$项是因为可能需要一些关于 $H(N)$长度的信息。 更正式地说，
字符串 s_j的算法熵可以表示为；

$$H_{prov}(s_j) = H(s_j|S) + H(S) \tag{4.6}$$

其中第一项通过其在集合中的位置来识别 s_j，而第二项 $H(S)$是表征集合模式或结构的子程序的长度。 如果集合表示由模型生成的字符串，则使用模型的参数来指定集合成员。

当 Ω 很大时，即 Ω 远大于集合的定义时，
对于相反和平凡的情况，当集合只有一个或两个成员时，$\mathrm{l}_o g_2 N$ 项成为主要的熵项，如第4.3节所讨论的。

一个简单的例子

Devine [46] 使用上述方法确定了一组有限序列的算法熵，其中1和0以不同的概率出现。 这样的序列可以被看作是给定温度下物理系统的微观状态。 一个例子可能是一个只有一些原子处于激发态的 N 个两能级原子系统，或者是一些自旋与一个小磁场对齐而其他自旋不对齐的 N 个自旋系统。在第一种情况下，'0' 表示系统中的原子处于基态，'1' 表示它处于激发态。 如果系统是孤立的，1 的数量是一个守恒量，因为能量是守恒的。

这个例子在某种意义上虽然平凡，但它阐明了关键点。 系统的微观状态可以用长度为 N的字符串来指定，其中有 R个1和（$N - R$）个0。 这些构成了超几何序列（参见Jaynes p69 [18]），通过从一个罐子中抽取有限数量的0和1直到罐子为空来生成。 如果罐子中有 N个由 R个1组成的对象，则具有不同排列的 R个1和（$N - R$）个0的序列集合有$\Omega_s = N!/[(N - R)!R!]$个成员。 模式集合中字符串的临时算法熵，对应于系统的不同微观状态，可以通过前一节中概述的方法来确定。 让$COUNT(S)$是计算字符串 S中1的数量的子程序。如果$COUNT(S) = R$，则字符串 S不是模式集合的成员。子程序“CRITERIA”（4.5）变为

$CRITERIA$
如果计数$(S) = R$，即不具有所需数量的$1's$)
否，则为NO，否则为YES。 (4.7)

使用这个成员资格标准，并忽略 log_2log_2和更高阶的项$|R|$和 $|N|$，一个微观状态的相关熵为： $H_{prov} \approx log_2\Omega_s + log_2N + log_2R$。 需要 log_2N位来指定字符串的长度并将所有单元格设置为零；需要 log_2R位来指定1的数量，并且需要 $log_2\Omega_s = log_2(N!/[(N-R)!R!])$位来指定每个配置中的每个 R个音符的位置。 如果所有的原子或自旋都处于基态，实际上处于零温度，算法熵将为$\approx log_2N$。 由于温度升高，算法熵的增加为 $log_2\Omega_s + log_2R$。 增加是由于不确定性的增加和需要指定模式。

然而，对于大 R，给定 N，与其直接指定 R，通过一个简短的算法从 R/N计算 R可能更短。如果R/N是一个简单的比例，比如 $^3{}_4$，只需要很少的比特来指定将3除以4的算法。 在这种简单情况下，给定 N， $log_2R = log_2(^3{}_4) + log_2N$，只需要很少的比特来指定将3除以4的算法。 另一方面，对于大 R，需要大量的比特来直接指定 R。 当出现一个简短的规范时，可以看作是一种偶然的排序，即从典型配置中偏离的有序波动。 然而，由于能量或比特必须在自然系统中保持不变，这只能发生在能量丢失或转移到系统的另一部分时。 换句话说，如果所描述的简单系统是一个自然系统，它不能完全与描述系统的振动或动量状态隔离开来。 在第9节中将更详细地讨论这个问题。

4.2 算法最小统计方法和临时熵

正如已经讨论过的那样，临时熵是一组相似字符串中典型字符串的算法熵。 早期，科尔莫戈洛夫引入了费舍尔最小充分统计概念的算法等价物，即科尔莫戈洛夫最小充分统计量，或算法最小充分统计量（AMSS）。 结果证明，这与临时熵完全相同。 这种方法在Li和Vit´anyi [76]的第2.1.1节中是隐含的，它寻求一个最小的集合，其中感兴趣的字符串是一个典型成员。 如果最小集合捕捉到字符串中的所有规律性，那么集合的最短算法描述以及在集合中识别字符串的算法应该对应于算法熵。 满足这一条件的集合也是给出临时熵的集合。 然而，人们永远无法确定是否找到了最小集合。

在Devine [46]关于有模式噪声集的熵的工作之前，Vereshchagin和Vit´anyi [103]开发了AMSS方法来通过优化生成字符串的程序的长度和运行程序的图灵机的描述来指定字符串的最短描述。 该方法通过优化所有机器$_i$上的图灵机$T\ i$的组合 $T_i(p) = x$和程序 p来实现。在这种情况下

(见第3.8节);

$$H(x) = min_{i,p}\{H(T_i) + |p|\} + O(1).$$

有许多图灵机可以生成字符串 x。 为了用随机性和结构描述一个字符串，需要在图灵机的描述较短但需要较长的程序 $|p|$之间进行权衡，以及在程序较短但描述较长的机器之间进行权衡。 当所有的模式或结构都包含在图灵机的描述中，只有变化或随机部分由程序 p捕获时，最佳情况发生。 再次，特定字符串的最短描述涉及到找到一个最佳集合，其中字符串是典型成员；即相对于集合中的其他成员，它是随机的。 由于图灵机是一个函数，它可以被认为是一个捕捉观察到的字符串中的基本结构或模式的模型。 或者，模型的特征可以通过UTM上的程序模拟。在这种情况下，程序捕捉了体现模型的图灵机指令，而剩余部分，即随机部分，则由标识感兴趣的字符串的代码捕获。 如果模型的描述过于复杂，它将过度拟合数据，但程序长度 $|p|$的减少不足以弥补模型描述长度的增加。 或者，如果模型过于简单，程序的大小 $|p|$将需要增加以提供详细的修正。 对于给定的 x，最佳情况由下式(4.8)找到。 这与方程4.1相同

在集合的语言中，该模型定义了一组具有共同结构或模式的字符串。 字符串中的所有结构信息都由定义集合的算法捕获。完成此操作后，特定字符串可以由生成字符串集的算法以及在集合中识别特定字符串的算法来指定。字符串的算法复杂度为[103]。

$$H(x_i) \leq H(x_i|S) + H(S) + O(1)。 \tag{4.8}$$

同样从临时熵的角度来看， $H(x_i) \leq H_{prov}(x_i)$，其中 $H_{prov}(x_i)$等于方程式4.8的右侧。当等号成立时， $H(x_i)$是算法最小充分统计量。它是一个最小统计量，因为对于集合的典型成员，所有信息都已捕获，字符串中没有进一步的模式可检测。 在这种情况下，如果给定该集合$_S$，并且Ω S表示集合的成员数，

$H(x_i|S)$，给定$_S$的描述长度是识别特定x_i的算法的长度。集合中的典型随机成员将在常数 c的范围内 $log_2\Omega S$，即 $H(\ x_i|S) \geq log_2\Omega_S - c$。 . 就像临时熵（4.6）一样，字符串由两部分算法定义。 在AMSS情况下，第一个描述了最优集合的结构 - 最简单的集合，其中字符串对其他成员是随机的。 第二部分识别哪个字符串是感兴趣的字符串。 这个术语捕捉到了由于不确定性而产生的水平。

数据。 一般来说，最优集合 S的算法充分统计量是一个代表性成员的算法描述的长度。

$$H(x_i) = H(S) + log_2\Omega_S + O(1)。$$

$H(S)$代表集合的描述， $log_2\Omega_S$ 是集合中典型成员的算法描述的长度，几乎与$_S$hannon熵测量的不确定性相同。尽管文献中的主要数学讨论使用AM_{SS}一词来定义最优集合以定义最短算法，但在这里，临时熵这个词更常用，因为它更容易与自然科学的熵概念联系起来。

4.3 临时熵和真实算法熵

有许多编码方法，例如基于熵的编码，可以压缩模式化字符串以提供其算法熵的上限度量。 问题是这些方法与临时熵或等效的AMSS方法有何关联。 通过下面的简单示例，可以最好地展示这种关系。 但是，回顾一下，递归可枚举集合是指存在一种可计算的过程来生成集合的所有元素，例如按顺序生成所有奇数。 同样，对于这样的集合，存在一种可计算的过程，可以遍历所有可能的字符串，并确定特定字符串是否是集合的成员。之前讨论的例子中的CRITERIA例程就是这样做的。 所讨论的例子显示了两种过程得到相同的结果。 此外，将结果与直接枚举感兴趣的特定字符串的过程进行比较，而不经过临时熵方法。

结论是，几个长度相似的算法可以生成一个特定的字符串。 虽然通常不清楚哪个是最短的，但对于典型的字符串来说通常并不重要。

要讨论的示例是由长度为N的形式为s_i=1y_1y_1y_1...1y_1y...1$y$$（其中y是随机的0或1）的嘈杂的周期^{2}的二进制字符串集合组成。 集合的$^{2\wedge}(N^{/2})$个成员给出$\log^2\Omega_S$=$N^{/2}$，并且可以通过递归可枚举的过程来指定该集合。 临时熵或AM_{SS}方法会认为熵是基于定义相似字符串集合的算法的长度加上标识集合中字符串的项。以下是两个不同的过程来指定所有这样的字符串的集合。

- 一个枚举集合中所有字符串的过程是从$N/2$个零开始逐个递增字符串形成

  $N/2$个二进制字符串从000 ..... 00到111 ..... 11。在开始时，以及每次递增后，可以在每个二进制位之间交错插入$N/2$个1来生成嘈杂的周期2二进制字符串的固定部分。

这个算法需要指定$N/2$个零，需要 $log_2(\ N/2)$位，

$$H_{prov}(s_i) \simeq N/2 + \log_2(N/2) + |0| + |1|.$$

- 或者，如果使用检查算法来识别哪些字符串在集合中，并且然后指定集合中的第i个字符串，则定义集合成员的算法将需要读取$N/^2$个字符来确定每对的第一个成员是否为'1'，近似于$log_2(N/^2)$。 这是定义集合的要求。还需要$N/^2$个比特来标识$^{2n/2}$个字符串中的第i个字符串。 如果忽略了确定字符串是否在集合中的算法的细节，结果与之前相同。

然而，与其使用临时熵方法，可以有一种直接指定字符串的算法。 在这个嘈杂的周期2序列中，一个特定的字符串可以通过将嘈杂的周期2字符串中的每个'11'替换为'1'，每个'10'替换为'0'来编码。例如，$11101011\ldots$变成$1001\ldots$。 这些代码相对于彼此和编码字符串是自我定界的，编码字符串的长度是原始字符串的一半。 然而，这个压缩的编码字符串本身并不是一个算法度量，因为该代码不能生成原始字符串。 需要附加一个简短的解码程序来生成原始字符串。 此外，代码必须相对于解码指令以及彼此自我定界的要求将另一个 $\sim \log_2 N$ 添加进来。在这种情况下，基于这种编码的相关算法熵为，

$$H(s_i) \simeq |code(s)| + |\text{解码例程}| + O(1)。$$

在这种情况下，解码例程扫描编码字符串，并将‘1’替换为‘11’，将‘0’替换为‘10’。 为了自我界定，字符串的整体规范必须包含关于 $code(\ s)$长度的信息。 可以通过将其包含在代码中，给出 $|code(s)| = N/2 + |N/2|$等方式来实现，或者在解码例程中包含$\sim |N/2|$，以便例程知道何时停止读取代码。 对于大的 N，忽略$\mathrm{O}(1)$解码例程的细节，结果与之前的方法几乎相同。即。

$$H_{prov}(s_i) \simeq N/2 + \log_2(N/2) + |1| + |0|. \tag{4.9}$$

需要注意的是，几个算法可以生成具有所需特征的特定字符串。 虽然很难确定这些算法中哪个是最短的，因此哪个给出了真正的算法熵，但主要项是相同的，差异在于相对较短的算法，这些算法指定了步进、检查或解码例程。 Chaitin认为，有一簇长度相似的短描述主导了所有可以生成特定字符串的算法。

然而，上述方程假设没有隐藏的结构。 如果进一步确定了结构，则集合的描述不是最优的。例如，在周期为2的序列集合中，存在诸如字符串s_i'= 10101 0. . .10这样的非典型字符串，存在更短的描述。 在这种情况下，真正的算法熵将小于AMSS度量或临时熵。 特别地，$H_{algo}(s_i') \simeq \log_2 N$对应于“$PRINT$ 10 $N/2$次 ”。 换句话说，对于集合的非典型成员，$H_{algo}(s_i') \leq H_{prov}(s_i')$。 方程4.9中的第一项变得不必要，表明由于该字符串不典型，集合或模型需要进行改进。 因此，熵度量被称为“临时的”，如前一节所讨论的，更多的信息可能表明最优集合更小，从而导致更低的算法熵。 然而，在许多物理情况下，集合的绝大多数成员都是典型的。

再次强调，正如方程4.3所示，大多数模式化集合中的字符串的算法熵由定义模式的所有二进制字符串集合的算法长度以及在集合中识别感兴趣字符串的算法长度组合而成。AMSS或临时熵方法为类似字符串集合的典型成员提供了最短的算法。 典型成员相对于类似字符串集合是随机的。 这对于确定物

理系统的熵有一些影响。 在物理系统中，可能存在偶然有序但仍满足系统定义基本条件（如粒子数和总能量以及物理约束）的配置。 这些偶然有序是典型配置的罕见波动，类似于热力学系统中远离平衡的偶然波动。 例如，玻尔兹曼气体中粒子的瞬时配置可能均匀分布在空间中。 这显示了结构或有序性，类似于一个字符串，如1010101. . .10。 在这些偶然有序的情况下，实际观察到的配置的真实熵将低于指定粒子位置的大多数状态的典型临时熵。

然而，正如后面所讨论的，对于真实的热力学系统来说，与没有物理意义的数学字符串相反，当系统的某一部分（例如位置状态）发生有序的波动时，为了保持能量守恒，熵必须转移到系统的动量状态。

4.4 以概率分布的形式进行规范

上述临时熵和AMSS方法以算法的形式指定了描述感兴趣的字符串的最佳集合，以及指定该集合中的字符串的算法。这适用于集合中的所有成员等可能的情况。 这种方法适用于指定表示真实世界系统的字符串。

真实世界系统的配置。 例如，一个孤立的气体激光系统可能包含处于激发态或基态的激光原子，不相干和相干的光子以及捕捉粒子运动的动量状态。

适用于给定能量的所有可能配置构成了可能状态的集合。 由于宏观态中的每个可能配置都是等可能的，实际状态首先需要定义集合，并给出用于识别集合中特定配置的香农熵项。

然而，对于不是所有字符串都等可能且字符串集可以由概率分布定义的情况（例如由伯努利过程生成的情况），集合的特性可以由成员的概率分布来指定。 然而，如果要根据概率分布来定义集合，则分布必须能够以约定的精度进行计算。 在这种情况下，香农的编码定理可以通过基于分布中每个数据点的概率值来为分布中的每个字符串指定唯一的编码。

让 $H(P)$代表计算参考UTM上分布的最短程序的长度。现在，属于分布的特定字符串 x_i的算法熵可以通过两部分代码来指定。 第一部分是以概率 P来描述可能字符串集的结构，而第二部分则是给定分布的实际字符串的标识。因此

$$H(x_i) \leq H(P) + H(x_i|P^*). \tag{4.10}$$

方程式4.10中的等号适用于绝大多数分布中的字符串，但是少数字符串的描述会更短。
例如，在伯努利过程中，以概率p发射1，以概率$1-p$发射0，长度为n的字符串中，由^{r}个1和$n-^r$个0组成的大多数字符串将以概率分布$p\,r(1-p)\ n^-r$出现。 在这种情况下，等号是适当的。 然而，一些非典型的字符串，例如前^{r}个位置都是1的字符串，可以通过算法进一步压缩，超过方程式4.10所暗示的压缩程度，这时需要使用另一种分布或算法。 这个方程式的右边也是字符串的算法最小充分统计量或算法最小充分统计量。

根据香农的编码定理（方程（3.12）），分布的每个成员都可以通过一个编码来最优地表示，使得$|code(x_j)\ | = -\lceil log_2 P(x_i)\rceil$，其中天花板函数表示值应该四舍五入上取整。与其不断使用天花板符号，作为惯例，这个整数将被表示为 $-log_2 P(x_i)$，认识到代码长度必须是一个整数。考虑到这一点

$$H(x_i) = -log_2 P(x_i) + H(P) + O(1), \tag{4.11}$$

对于集合的典型成员来说， $O(1)$ 包括任何解码过程，如果算法中没有明确说明的话。 对于 $H(x_i)$ 的期望值（即

$\langle H_{algo}(x_i)\rangle)$, 给定这个分布，通过将方程式(4.11)中的每个项乘以 x_i的概率并对所有状态求和来找到。由于大多数贡献的总和都有等号(s在(4.11)中)，而那些没有等号的贡献概率非常低，所以期望的代码长度接近于香农熵 H_{sh}，即 $H(P)$很小。这在5.2节中有更详细的讨论。换句话说，

$$\langle H_{algo}(x_i)\rangle = H_{sh}(\text{所有状态的集合}) + H(P) + O(1) \tag{4.12}$$

隐含在生成字符串的算法中的 $O(1)$解码程序。
给定 x_i的编码，其中 $|code(x_i)| = -log_2 P(x_i)$，解码过程按顺序遍历具有相同概率的所有字符串，直到找到与正确编码相关联的x_i。 然而，在实践中，编码和字符串必须保持一致的顺序。 例如，在二项分布中，具有相同数量的零和一的字符串将具有相同的概率。 如果编码和字符串按字典顺序排序，具有相同概率的情况下，按字典顺序的第五个字符串由与该概率相关联的按字典顺序的第五个编码指定。 字符串可以被明确地编码和解码，因为总是有足够长度的编码来编码具有相同概率的所有字符串 [1]

如何指定有噪声的数据

临时熵或算法最小充分统计量方法展示了如何通过两部分算法找到具有可识别结构但也具有随机成分的特定字符串的算法压缩规范。 第一部分指定了具有相同模式或结构的所有字符串的集合，第二部分在给定集合的情况下指定了感兴趣的字符串。

在此之后，第4.4节将该过程扩展到第一部分以可枚举概率的形式指定集合，并且第二部分通过基于概率的编码指定了特定的字符串。 这与第一种方法不同，第一种方法中字符串本身包含噪声或随机部分。 概率方法适用于观测数据存在噪声的情况。即需要找到一种算法，使用模型或物理定律来指定字符串，然后使用这种可枚举概率分布来指定精确的字符串及其与模型的偏差。在概率公式中，噪声是与理想值的偏差，而不是像第一种情况中那样体现在字符串中。

模型示例

在二维空间中的一个例子可能是最佳拟合由方程描述的噪声数据集的直线模型：$y = \mathrm{ax} + b + \mathrm{no}_i\mathrm{se}$

这有效是因为概率已经被定义为特定精度级别 如果精度级别增加，log(概率)的长度增加，可用的编码也会增加

$y(x_i)$ + 噪声 其中噪声可以由概率分布P来指定，例如正态分布：$y_i = y(x_i) + P(y_i \text{ - } y(x_i))$ 为了使用算法在字符串集合中指定一个特定的字符串，需要认识到模型预测的偏差只是$y_i \text{ - } y(x_i)$的概率分布 根据香农编码定理，一个特定字符串与模型的偏差可以通过最佳编码来生成：$|\text{code}(y_i \text{ - } y(x_i))| = \text{-log}_2 P(y_i \text{ - } y(x_i))$。也就是说，给定这个编码，可以从其编码中生成偏差，如第4.4节所述。

在所述直线示例中，特定字符串 y_i的临时熵是从其方程和相关概率分布定义字符串的算法的长度，加上指定该字符串与方程之间的偏差的代码。结果是

$$H_{prov}(y_i) = H[ax_i + b,\ P] - log_2 P[y_i - y(x_i)] + |\text{解码例程} \qquad\qquad | + O(1),$$

或者

$$H_{prov}(y_i) = H\,[ax_i + b,\ P] + |\text{编码}(y_i - y(x_i))| + |\text{解码例程}| + O(1).$$

这里的O(1)解码例程被明确标识。 第一项捕捉了模型的最短描述及其相关的可枚举概率分布，而第二项指定了根据第4.4节的论证从精确模型中的偏差。第二项的期望值是偏差的香农熵。

这种方法有效地选择了字符串的算法最小充分统计量（AMSS）或者是临时熵的最佳估计。
它与后面讨论的最小描述长度方法密切相关，在噪声数据中，理想的MDL形式给出了相同的模型。 生成点y_i的算法包括基于概率的编码程序、解码程序、计算与模型偏差的概率分布以及模型的具体规范。 该算法执行以下计算步骤：

- 通过知道code和 $|code(y_i - y(x_i))| = -log_2 P(y_i - y(x_i))$，生成该字符串的概率值；
- 使用P的例程计算从概率值得到的 $y_i - y(x_i)$。
- 从 $ax_i + b$生成(x_i)；
- 将第二个项目中的 $y_i - y(x_i)$ 添加到(x_i) 中，并生成所需的特定 y_i。

一个更一般的例子

不需要列出地球在一年中的每一秒的位置，牛顿的

定律可以通过一组方程来描述运动。如果这些方程完全描述了地球围绕太阳的运动，那么这些方程就代表了数据的极度压缩版本。

然而，这种捕捉并不完全准确，因为会出现变化。 模型与实际数据的差异可能是由于测量误差、月球的扰动或简化球体假设的破裂等原因。当这些偏差表现为噪音时，可以用概率分布来表示。 在这种情况下，观测数据仍然可以通过算法描述来进行压缩 - 最初是根据牛顿的定律（指定特定字符串应该完全预测数据点）以及编码观测数据点与牛顿的简单预测之间的偏差。 如果以后发现了变化中的进一步原因或模式，例如考虑到月球的影响，可以改进压缩。 如果以后发现了变化中的进一步原因或模式，例如考虑到月球的影响，可以改进压缩。

一个好的模型是最大程度地压缩了数据。 所有的规律都在模型中，而变化（可能是由于模型的限制、噪音或实验误差）将表现为与模型的随机偏差。 这再次是用于找到数据讨论中的算法最小充分统计量的论证，但模型是由物理定律捕捉到的，而随机性则由识别给定定律的特定字符串的代码捕捉到。

人们会期望这个最佳拟合与一系列数据点的确定有关，通过最小化每个点与模型的偏差来确定。

模型选择

前两节展示了如何在理解生成字符串的定律和与定律的偏差的基础上指定表示数据点的字符串。 Vereshchagin和Vit´anyi [103]应用这些思想，根据Kolmogorov的AMSS方法，提供了一种全面的方法来选择描述特定字符串或字符串的最佳模型。正如Vereshchagin和Vit´anyi [103]指出的：“统计推断和学习理论的主要任务是提取数据中的有意义的信息”。

简而言之，给定一组数据，该数据的最佳模型是能够捕捉到所有有用信息的模型。 一般来说，基于自然法则的模型能够对手头的数据进行最佳压缩。 由此可见，可以尝试用不同的模型来解释数据，并选择具有最小描述的模型作为最佳解释。 因此，对于数据的每个可信模型，可以使用一个两部分算法来定义每个字符串。 第一部分使用模型，第二部分编码与模型的偏差。

整体描述最短的模型是首选模型。 描述使用最佳模型的数据字符串的算法是算法最小充分统计量，因为所有有用信息都包含在算法中。 正如Vereshchagin和Vit´anyi [103]解释的那样：“对于每个数据项和每个复杂性水平，在给定复杂性的模型类上最小化两部分编码，一部分是模型描述，一部分是数据到模型的编码，可以确定选择在严格意义上是数据的最佳解释的模型”。 提供两部分编码的代码

模型的最短算法描述，加上与模型的偏差，是首选。 换句话说，从找到字符串的最短描述到找到最佳模型，只有一小步之遥，该模型可以对一组字符串进行建模。

基于这一点，生成一组字符串中的典型字符串的算法，由模型和与模型的偏差定义，需要以下组件。

- 模型的描述，在上述直线模型中，需要$2d$位来指定a和b的精度d。
- 与模型的偏差的描述。即给定输入x_i，偏差为y_i - $y(x_i)$，其中y_i是完美模型的预测值。 偏差y_i - $y(x_i)$按照概率模型分布。 根据香农编码定理，偏差可以通过长度为-$\log_2 P[y_i$ - $y(x_i)]$的编码进行最优编码。 对于高斯分布，对数概率项变为$(y$ - $y(x_i)]$^$_2$ / $(_2\sigma)$^$_2$ - $\log_2(\sigma \sqrt{2\pi})$ 注意标准差， σ是由数据确定的。如果结果不是高斯分布或其他对称概率分布，则意味着存在隐藏的结构，偏差不会是随机的。在这种情况下，需要一个更精细的模型。

应注意，参数 a和 b以及分布的特性不是由模型选择过程确定的。 它们必须来自决定最佳模型的其他过程。 在简单的直线情况下，它们可能由拟合过程确定；例如，最小化平方偏差的过程。 考虑这样一种情况，怀疑直线模型是解释一组数据的最佳方法。 如后面的第7节中所讨论的，如果二次多项式提供比直线更好的拟合，这是因为模型描述的增加被从模型中减少的偏差所抵消。 在3个参数的情况下，需要额外的 d位来描述模型（总共$3d$位），这必须被变量部分的对数概率项的减少所抵消，即 $(y_i - y(x_i))^2/(2\sigma)^2$。 现在可以看到算法方法的局限性。 必须首先确定每个模型的最佳参数，然后才能进行比较。 然而，一旦通过任何方式找到最佳模型，基于模型的临时算法熵是每个字符串熵的最佳估计。

模型选择方法与稍后在第7节讨论的理想最小描述级方法相同。 只有在感兴趣的字符串或感兴趣的字符串在两部分描述中是典型字符串时，才有效。

它们在集合中是随机的，没有明显的特征。 如果不是这样的话，就像在第4.4节讨论的那样，可能会有另一个概率分布更好。

另外一个要点是，通过最小化两部分描述获得的模型是数据的最佳解释，但不一定是正确的解释。

第5章

顺序和熵

5.1 秩序的含义

原则上，可以从两个物理系统或一个物理系统的两个可识别部分中提取能量，前提是其中一个比另一个更有序。 这样做会增加宇宙的总无序度。 然而，为了提取能量，观察者必须能够识别秩序及其性质，以便能够有用地利用它。 有序状态集是系统所有可能的有序和无序状态的子集。 通常只有在子集的成员展示出共同的模式时，才能识别和描述秩序。 对于外星人来说，一个人类DNA序列的字符串不会显示出明显的模式，因为化学序列只会被视为无数可能性之一。 虽然聪明的外星人可能有能力进行随机性测试（第8.2节）来确定发现的DNA字符串是否显示出秩序，但这不太可能成功，因为DNA字符串可能已经被最大程度地压缩。 外星人确定模式的最可靠方法是观察来自各种人类的DNA集合，并注意到它们有很多共同之处。

一般来说，正如第8.2节所讨论的那样，秩序的程度或者缺乏随机性很难量化。 然而，有序的序列可以被认为展示了一种模式，因为我们的思维，就像外星人的思维一样，感知到该序列属于一组类似的有模式的序列的子集。

我们的思维认为这个子集与表面上普通或随机的字符串集是不同的。 如果模式被识别出来，子集的任何有序序列 $s$of the subsetcan be compressed i.e. be described by an algorithm much shorter than thesequence length. 因此，任何常见的模式都意味着存在一个算法 s^* existswhich can specify the patterned string and for which $|s^*| << |s|$. 一旦通过观察或某种试验过程确定了这样的模式，无法确定模式的计算过程变得无关紧要 - 它可以被压缩（参见[91]和[42]的评论）。"一个简单的例子说明了这个论点。考虑所有长度为 $N$of the form $1y1y1y \ldots 1y1y$ where $y$is randomly either '0' or '1'的字符串集合。

由于该集合具有可识别的模式，因此可以将集合中的序列压缩成较短的描述。 这与长度为 N的一般序列形成对比，其中没有可辨别的顺序，并且没有任何存在顺序的确定性。 由于典型的集合成员 s无法压缩，一般情况下$|s| \approx$字符串的长度。 更正式地说，任何有模式的集合都是所有可能字符串的子集。

Devine（2006）认为，使用AIT框架来指定模式方法的算法复杂性具有一些优势。 此外，至少在原则上，它似乎适用于不仅仅是带有噪声的简单模式的系统。 实际上，如果可以扩展AIT概念，它们将提供一些有趣的潜在应用。

例如，某个时刻的特定植物由基于其DNA的算法以及编码物理和生化规律的指令集来指定（更详细地讨论在第6.3节）。算法的输入是可访问的营养物质和来自外部环境的影响，这些因素决定了植物的生长轨迹。 描述植物在某个时刻的算法熵是由在该时刻停止的算法生成的。 随着时间的流逝，同样的算法在稍后的时间停止，生成了更复杂的生物模式。 描述植物生长的算法在每个细胞中复制。 每个细胞算法的输入是相邻细胞的化学输出，相邻结构以及由相邻细胞介导的外部环境。

生物细胞簇在适合极其复杂非线性系统的状态空间中似乎是吸引子，如第9.6节所述。 该簇的组成部分通过平衡细胞之间的相互作用来维持稳态，整体在一个稳定区域中保持稳定。 当无法维持稳态时，算法终止，导致死亡；可能是由于外部输入的冲击；可能是由于重复复制导致算法损坏；或者可能是由于资源或物理限制限制了细胞算法的持续复制。

植物或动物中的所有细胞都具有潜在的模式和变异。 例如，眼睛中的细胞与肌肉中的细胞不同，它们仅因为不同的环境输入而有所不同，但它们由相同的DNA指定。 原则上，有两种算法方法可以指定不同的细胞类型。 第一种方法可以通过来自包含基本细胞模式不同变体的集合的字符串来指定不同的细胞类型。 第二种方法可能会指定生成细胞的算法，该算法还包括确定表达不同细胞类型的基因的机制的信息。 在随后的一节中，将论证，一旦考虑到所有因素，最短的算法是从种子生成植物的算法。 然而，无论哪种方法给出更短的算法，AIT都可以用于研究任何生物复杂系统（即有组织的复杂系统）在不同尺度上的不同层次，并且在每个嵌套层次上，只需要解释相关的模式集。

这种方法与Beer的自上而下的方法是一致的

可行系统模型（[10, 9]）是一种用嵌套子系统来探索有组织系统的框架。 这种方法在第9节中得到扩展，并与Chaitin的d-直径复杂性概念[28]联系起来。

是否有可能将这种方法应用于如此复杂的问题是另一回事，但对于一些极其简单的系统，可能会取得进展。 这种可能性在前一章中得到了说明，其中算法信息被用来指定有序集合中的一个成员。这表明了变异的感知问题，即噪声的出现，是可以避免的（第3.1节）。

5.2 算法熵和传统熵

一个微观状态，对应于热力学系综状态空间中的一个特定配置，可以表示为一串二进制字符。 这里的状态空间包括所谓的相空间，即指定粒子位置和动量的空间，以及其他状态，如电子状态，与相空间坐标相反，可能是离散的。 当粒子的位置或动量的规范不是离散的时，需要用二进制表示法指定坐标的精度。 一种方法是从粒子1开始，指定其位置坐标到约定的精度，然后对x轴、y轴和z轴进行相同的过程。每个粒子的动量坐标也以此类推。 粒子的离散电子状态也可以用二进制字符串表示。

粒子1的位置和动量坐标在一个6维相空间中定义。 这个空间指定了所有N个粒子的位置，被称为Γ空间，由6N个维度组成。 指定一个配置的字符串 s_i看起来像这样；

$$x_n, y_n, z_{,n}, p_{x1}, p_{y1}, p_{z1}, p_{x2}, p_{y2}, p_{z2}, \ldots\ldots\ldots\ldots p_{zn}$$

这里 x_1是原子1的 x坐标等，而 p_{x1}是其在 x方向上的动量坐标。 坐标的规定精度实际上是在空间上施加了一个网格或细胞结构。 通过加倍指定每个坐标的位数来加倍分辨率，将使Γ空间中的细胞数增加2^{6N}。 Zurek使用了一种替代方法来将微观状态定义为一个字符串[108]。 他将$6N$维的Γ空间分成细胞，然后论证细胞的尺寸选择得足够细，以至于一个细胞要么包含一个粒子，要么不包含粒子。 特定微观状态的配置可以通过有序地列出细胞并用0表示空细胞、用1表示占据的细胞来进行系统化的描述。 微观状态的具体规定方式并没有真正的区别，只要存在一个简单的短程序可以将一个规定转换为另一个规定。 这可以针对刚刚概述的两种方法来完成。

当微观状态没有更短的算法描述时，算法熵略大于描述的长度。

另一方面，有序状态的算法描述会更短，因为描述可以被压缩。 有效地，细胞分辨率或网格分辨率可以在一定限制内选择为物理上合理的。 严格来说，经典系统中微观状态的算法描述需要对网格结构进行描述，并在该网格结构内指定微观状态。 然而，这是给定信息的一部分，就像物理定律一样，并且在熵差中抵消。 对于量子系统，网格单元的分辨率由不确定性原理确定。 这总是可以作为经典情况下的极限分辨率要求。

需要注意的是，算法方法与玻尔兹曼和吉布斯熵度量一样，也必须假设存在基础的网格结构。
Zurek [108] 表明，吉布斯熵中的歧义，源于对相空间的粗粒化选择，可以被吸收到系统的背景算法描述中。 换句话说，相空间或其他连续坐标的划分方式被封装在关于系统的消息中。 Zurek [108] 指出，整体描述必须在给定特定UTM的情况下最小化。复杂的网格结构，使吉布斯熵的粗粒化问题变得复杂，需要长的描述并变得不合格。 实际上，正如Zurek指出的那样，在算法方法中，网格结构的选择的复杂性被吸收到UTM的选择中。在实际应用中，由于关键的物理参数是状态之间的熵差异，与所使用的UTM或关于系统的消息（即共同信息和分辨率）有关的问题只会将算法熵零点进行平移，因此可以忽略不计。

一旦微观状态的规范被定义，计算微观状态在状态空间坐标中的短-自界定算法的长度就是算法熵。 下面将论证一个典型或平衡态的算法熵与Shannon熵有一种形式上的相似性，即 $H_{sh}(\mathcal{E}) = -\sum p_i log_2 p_i \text{of the ensemble } \mathcal{E}$。 然而，算法熵和Shannon熵之间存在一个概念上的差异。 算法熵确切地定义了给定的微观状态，而Shannon熵是对微观状态集合中的不确定性的度量。

有一些论证可以阐明Shannon熵和算法熵之间的关系。 第一个论证更多地是一个提供洞察的“手挥”论证。 第二个论证更为严谨，而第三个是一个详细的算法熵计算，显示了玻尔兹曼气体的算法熵与Sackur-Tetrode方程给出的熵相同，而第四个则显示了临时熵的主导项是Shannon熵。

- 让热力学系统的第$_i$个微观态的状态空间坐标被指定为长度为N的字符串 s_i。 同时，让 s^*_i是生成字符串 s_i的最小程序，其中 s_i在集合中以概率 p_i 出现。 如果算法熵字符串的期望值

集合中字符串s_i的算法熵的期望值用$\langle|s_i^*|\rangle$表示，第一个参数认识到所有字符串 s_i的算法熵的期望值为
$\sum$这个求和可以重新排列为算法描述的长度之和，即$\langle|s^*i|\rangle = \sum_{l} p_{ll}$其中 p_l是算法描述长度 l的概率。在热力学系综中，绝大多数可能的状态属于平衡态的集合中。 这些状态无法显著压缩。 如果系综中有2^N个状态，大多数状态可以用长度为 $log_2(2^N)$的算法表示。 在这种情况下，求和中绝大多数字符串的长度将接近 N。 如 $\sum_{l} p_l l$ 变为 $l\sum_{l} p_l$ 和 $\sum_{l} p_l = 1$时，字符串的预期算法熵结果约为 N。 换句话说，大多数字符串的算法熵将与系综的香农熵相同。 香农熵本身与统计力学的熵S 相关，其中$S = k_B ln 2 H_{sh}(\mathcal{E})$，而后者与宏观热力学熵相关。

- Zurek [108]和Bennett [1_2]（参见[76]）的第二个参数使用了第4.4节的方法来突出算法熵和香农熵之间的关系。 令 $P(x)$为热力学系综中所有状态的概率分布，以 $H_{sh}(\mathcal{E})$为香农熵。 方程（4.1_2）显示，算法熵不会比香农熵大 $H(P)$。 但是，因为Shannon的无噪声编码定理3.3表明，每个字符串的编码长度 l_i可以在一个比特范围内，即这个长度的期望值 $\langle H_{algo}(x_i)\rangle$最多略大于 $H_{sh}(\mathcal{E})$，因此 $H_{sh}(\mathcal{E}) \leq \langle H_{algo}(x_i)\rangle$。 正如Bennett [$1_2$]所提出的，计算由 $H(P)$给出的概率分布的算法长度只有几千位，而香农熵通常非常大。 如果位置和动量坐标被定义为8个有效的二进制数字，那么对于一个摩尔气体中的10^{23}个粒子，每个粒子需要16位。因此，与之相比， $H(P)$可以忽略不计，可以看出微观状态的期望熵 $\langle H_{algo}(x_i)\rangle$与系综的香农熵相差O(1)。 对于一个处于平衡状态的系统，或者具有许多状态的孤立系统，典型的算法熵几乎与香农熵相同。

- 第三个参数，由Zurek开发，表明在理想的Boltzmann气体中，特定平衡态的算法熵与Sackur-Tetrode方程[108]提供的熵相同。 一个简单的论证来证明这一点是给出Ω个可能配置中的一个特定配置的地址。 地址的典型值约为$\sim\Omega$。 这需要一个长度约为$\log_2\Omega$位的描述。 任何配置都是由0和1的序列组成。 如果相空间中有$\mathcal{C}$个细胞，则会有$\mathcal{C}$^$N/N!$个不同的配置。 由于细胞的数量给定

byt $\mathcal{C} = (V/\Delta V)(\sqrt{mk_BT})/\Delta_p)^3$，算法规范导致Sackur-Tetrode方程。这里m是粒子质量，ΔV是三维单元体积，而Δ_p是动量空间中的单元体积。
这个论点表明，一个典型状态的算法熵与统计力学的熵相同。

- 第4节，特别是(4.2)表明，一个字符串的临时熵由两部分代码给出。 第一个定义了包含字符串的集合的结构，第二个标识了集合中的特定字符串。后者实际上是集合的香农熵。 对于一个热力学宏观态，如果集合的规范是一个短算法，那么临时熵(算法熵的最佳猜测)的主要贡献者是宏观态中字符串集合的香农熵。

算法方法可以用来证明，随着时间的推移，一个孤立系统将趋向于与最可能状态集合对应的平衡配置。 正如后面的第6.3节讨论的那样，Zurek [108]表明，在一个孤立系统在物理定律下演化时，经过大量步骤，但相对于Poincar´e重复周期 $\mathcal{P}$而言很小，轨迹将在大部分时间内定居在平衡区域，当状态是算法随机的并且典型描述是 $H_{algo}(s_k) = |s_k^*| \approx log_2\mathcal{P} = H_{sh}(\mathcal{E})$。这在第6.3节中进一步讨论。

玻尔兹曼、吉布斯、香农和算法熵

玻尔兹曼、吉布斯和香农熵实际上是关于整个系统的无知程度的度量，与算法熵相反，它们对系统的特定状态没有任何说明。 本节进一步讨论了玻尔兹曼、吉布斯和香农熵与算法熵之间的关系。 首先讨论了算法方法和统计力学方法之间的差异。 在这个讨论之后，介绍了Zurek在5.2节中提出的物理熵的概念。Zurek的熵度量是算法熵和香农熵的结合，既捕捉了关于物理系统微观状态的已知信息，又捕捉了不确定性。 Zurek使用这种方法来提供对可逆性和从这样的系统中提取功的强大理解。然而，现在清楚的是，没有必要对Zurek的物理熵进行特殊定义，因为它对应于第4节中讨论的临时熵，[46, 48]，或者等价的Gacs的替代熵[59]或Vereshchagin和Vit´anyi提出的算法最小充分统计量[103]。它们都是一样的。 尽管它们有不同的标签，但它们只是基于可用信息的最佳猜测算法熵。 因此，Zurek对从物理系统中提取功的要求的洞察力同样适用于临时算法熵（或其等价物），而无需定义物理熵。 此外，整个方法表明算法熵是更基本的。

这个概念比玻尔兹曼熵和吉布斯熵（或香农熵）更重要。 事实上，这些熵可以看作是算法描述的极限情况，当没有关于确切状态的信息，只有关于可能状态集合的信息时。本节的其余部分将更详细地讨论这些观点。 然而，再次强调，算法熵测量的是系统的微观状态；它是实际状态的最短描述。 然而，整个状态空间上的算法熵的期望值几乎与香农熵相同，因此除了使用的单位不同之外，它等同于统计力学的熵。 与其他熵不同，算法方法必须指定研究系统的条件，因此对齐并不完全准确。 这些条件在其他情况下是可以理解的。 然而，由于这些条件对整个系统的规范是共同的，它们只会将熵的零点进行偏移，对熵的差异没有影响。

玻尔兹曼方法考虑了一个容器中的 N 粒子气体。
玻尔兹曼通过在6维位置-动量空间中的有限单元格中的气体粒子数量来指定气体系统的瞬时宏观状态；即系统的相空间。 给定的气体宏观状态以温度、压力和体积来指定，将包含无数个微观状态。 微观状态的数量被认为是Ω。 玻尔兹曼用现代符号描述统计熵，即S_B =- $k_B log_2 \Omega ln2$。 在6维相空间中，将其分为 m个单元格，第 m个单元格中有 N_i个粒子，微观状态的数量由$\Omega = N!/(N_1!N_2!...N_m!)$给出。 使用斯特林近似，这变为熟悉的 S_B =-$k_B N \sum$

这里 p_j表示单个粒子在细胞$_j$中的概率。 对于非相互作用系统和一个细胞的细网格，这等价于 $S_B = -k_B ln\Delta\Gamma$，其中$\Delta\Gamma$是在6N维$\Gamma$空间中占据的体积。

另一方面，吉布斯方法设想了一个系统的多个副本，它们都处于不同的微观状态，并且能够相互交换能量。 该方法通过$6N$位置-动量坐标来确定系统的一个微观状态；即系统的Γ空间。即微观状态$X = q_{1x}, q_{1y}, q_{1z}...q_{nx}, q_{ny}, q_{nz}, p_{1x}...p_{nz}$其中 p_{1x}, q_{1x}是粒子1的位置和动量矢量等。在这种情况下，吉布斯熵为

$$S_G = -k_B \int \rho(X) ln[\rho(X)] dX,$$

其中 ρ表示状态 X的概率密度。 在实践中，熵测量基于将Γ空间粗粒化为大小为$_\Delta$的单元格，以避免Liouville定理的后果。 在这种情况下，方程变为所有粗粒化单元格 $S_\Delta = -k_B \Sigma_i p_\Delta(i) ln p_\Delta(i)$和 $p_\Delta(i)$是微观状态在单元格 i中的平均概率。 这与玻尔兹曼测度相反，玻尔兹曼测度指定了粒子在单元格 i中的概率。 然而，对于一个大的非

在平衡的相互作用粒子中，熵的吉布斯测度与玻尔兹曼测度相同。

缺失信息和算法描述

在大多数状态是算法随机的情况下，系统最有可能处于平衡状态。 在这种情况下，如上所示，算法熵的值几乎与香农熵的值相同。

由于平衡态占据了预期的算法描述，大多数算法描述的数量级为 $log_2\Omega$。 然而，当状态显示出一定的有序性时，算法熵远低于香农熵，从系统中提取能量变得可能。 由此可知，如果在某一时刻对系统的微观状态一无所知，则最佳猜测，即最可能的算法度量，将是香农值。 然而，观察可能提供信息，表明系统处于非平衡或有序状态。 在这种情况下，算法描述将比平衡态的描述更短。 例如，如果观察到玻尔兹曼气体的所有粒子都在容器的一侧，这种有序状态的算法描述将低于香农熵。 随着对有序状态了解的信息越多，压缩算法描述变得可能。 这与处于平衡态的系统相反，其中更准确的算法熵规范不会导致任何变化。 正如已经提到的，这一观点促使Zurek [108]定义了微观态的'物理熵' $\mathcal{S}_d$。 这是基于可用信息的最简洁但部分的算法描述的总和，以及一个香农熵项来考虑剩余的不确定性。 在给定信息 d的情况下，最佳算法描述将算法熵表示为 $H_{algo}(s_i|d)$。 这相当于定义字符串中的结构或模式的信息，即临时熵或AMSS。Zurek将剩余的不确定性视为条件香农熵项 $H_{sh} = -\sum$

在给定可用信息的情况下，物理熵为 $\mathcal{S}_d = H_{algo}(s_{i|d}) + H_{sh}$。 Zure_k使用这种方法来论证，当物理熵较低时，系统是有序的，可以提取能量。 然而，通过与第4.1节和第$4._2$节的比较，可以看出 $\mathcal{S}_d$可以被视为临时熵。此外，方程（4.4）与上面的方程形式相同，因此不需要引入新的熵概念。即

$$\mathcal{S}_d = H_{prov} = H_{algo}(description\ of\ pattern) + H_{sh},$$

因为识别集合中特定字符串的算法与Shannon熵具有相同的值。 临时算法熵是基于给定信息的真实算法熵的最佳猜测。 随着更多信息的获取，有序状态下的真实算法熵将下降，但在平衡状态下将保持不变。 我们的图2使用Zurek的论证，但用临时熵替代他的物理熵来说明这两种情况（a）和（b）。 由于Shannon熵是

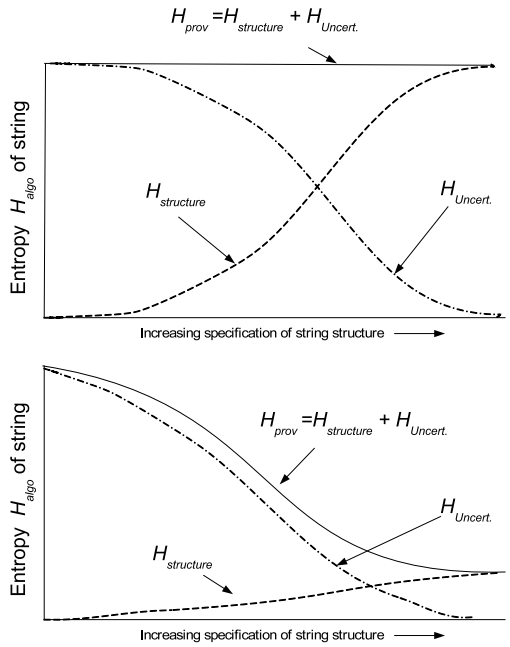

图5.1：临时熵（a）字符串是集合的典型成员时（b）字符串来自有序配置时。

根据可用信息的变化，它被表示为 H不确定
在图中。

当微观状态是平衡状态（图2**a**），即典型或随机状态时，即使更多信息减少了Sh**a**nnon熵项（标记为 H不确定在图中），临时熵仍然保持不变，从而更清楚地识别特定的随机状态。 算法术语增加以补偿不确定性的减少。对于这样的状态，状态的最短描述返回与Sh**a**nnon熵相同的值，即使有更多信息可用于更好地识别确切的状态。

实际上，算法最小充分统计量（AMSS）对于状态来说是平衡状态的集合（见4.2节）。

当微观状态按照图2b中的顺序排列时，额外的信息会导致在确定排序的程度后，临时熵的减少。 这种方法适用于任何噪声描述，其中模式、结构或顺序通过更精确的测量来揭示。

这相当于通过进一步的信息来改进模型，从而缩小噪声或不确定性的范围。 Li和Vit´anyi [76]以另一种方式看待这个问题。他们将揭示顺序的过程描述为更准确地定义气体的微观状态。在低分辨率下，只有足够的信息来识别宏观状态，相应的临时熵很高。 增加系统的分辨率会降低算法熵，因为不确定性减少。 对于每个6N个轴的分辨率加倍的过程，相当于将每个轴的坐标的有效（二进制）数字的数量加倍。 这个过程最终将确定配置是有序的，并且随着分辨率的增加，临时算法熵下降，直到最终接近实际算法熵。 更准确地确定系统的确切配置的这个过程相当于找到包含该配置的最佳集合。 即确定状态的AMSS，如第4.2节所讨论的。

对算法熵和香农熵的评论

前面的章节已经论证了典型字符串的算法熵，表示平衡态状态下的一个配置，与香农熵几乎相同。 如果平衡态配置中有Ω个状态，只有极少数的这些字符串可以被压缩。 因此，典型状态只能由长度不短于$\log_2\Omega$的字符串表示。 但要重申的是，香农熵和热力学熵在概念上是不同的。 算法熵返回一个实际状态的熵值，并且对非平衡态配置具有意义。 另一方面，传统熵返回一组状态的值，在热力学情况下，通常是平衡态配置的集合。 此外，香农熵将重要的背景信息视为已知。 它不详细说明系统，也不指示应该包括哪些消息或成员在可能结果的集合中。 另一方面，算法信息论方法必须：

- 指定系统，包括分辨率或颗粒大小，在这个例子中，系统是玻尔兹曼气体；
- 指定计算开销和相关的物理定律；即表达等效于“PRINT”和“FOR/NEXT”的计算语言；
- 指定定义系统中元素的字符串的大小（因此在规范中经常出现的$\log_2 log_2$项）。

正如在第3.4节中讨论的那样，必须通过算法来指定的给定信息可以被视为定义算法熵的零点。 在这种情况下，香农熵被看作是平衡态中典型状态的算法熵的特例。

尽管在概念上存在差异，算法熵和传统熵之间的关系足够稳健，以至于可以在描述之间切换，只要允许对算法度量进行舍入，并认识到算法方法的系统规范要求。 例如，在算法情况下，系统规范，如相空间分辨率，要么包含在描述中，要么作为给定的[108, 48]。 这意味着避免了玻尔兹曼方法的歧义

一个简单的思想实验显示了算法熵和热力学熵之间的联系。 实际上，两个状态之间的算法熵差异大约是信息位的传输，而热力学熵是能量的传输。这些是相关的。 当在温度为 T的计算系统中在两个状态之间传输ΔHbit时，能量传输为 $k_BTln2\Delta H$。 这遵循了兰道尔原理，即改变一位信息所需的能量为$KT\ ln2$焦耳。 考虑属于不同宏观状态的状态 i和 o。 在系统孤立时，每个状态都是宏观状态中的典型配置。 必须传输到系统中或从系统中传出的最小位数，以将状态 i更改为状态 o为 $|o^*| - |i^*|$[15]。 由于这是 $H_{prov}(i) - H_{prov}(o)$，在温度为 T的真实世界计算系统中设置新配置中的位所需的能量流入为 $k_BTln2[H_{prov}(o) - H_{prov}(i)]$（见第6节） [1]。 从热力学的角度来看，将温度为 T的真实世界系统从包含状态i的微观状态集转移到包含状态 o的新平衡宏观状态所需的热量流 δQ将与从 i到 o的位设置变化所需的能量流相同。 因此，状态之间的热力学熵差异也是$\delta Q/T = k_Bln2[H_{prov}(o) - H_{prov}(i)]$（参见[109]）。 一个例子可以说明这一点，即在0 K下的一块冰和在相同温度下约束在相同形状的熔化冰之间的差异。 给定原始冰的配置，需要输入系统的位来指定熔化的配置为 $\delta Q/(k_BTln2)$。 因此，两个平衡状态之间的热力学熵差异与算法熵差异（以位为单位）乘以 k_Bln2相同。 对于非平衡或非典型配置，两个熵差异不仅相同，而且临时熵对于非平衡或非典型配置也具有意义。

临时熵的概念最终将这两个框架联系起来。
当一个字符串指定一个配置，显示配置部分有序部分随机时，该字符串只能部分压缩

[1]请注意，我们可以用任何温度的计算机来测量比特数，但由于参考计算机模拟了真实世界的通用图灵机，可以将参考通用图灵机设想为与真实世界系统具有相同温度的操作。

其他字符串可能具有相同的基本结构。 总之，临时熵是表示热力学系统瞬时配置的字符串的最短已知描述，无论是平衡状态还是非平衡状态。临时熵是理解具有模式或结构的系统所包含的熵的关键。

第6章

可逆性和计算

统计方法应用于热力学的一个历史性困难通过涉及一个被告知的代理人的思想实验来说明，这个代理人通常被称为麦克斯韦的恶魔，他能够通过巧妙地操纵容器中气体的粒子而不花费任何代价来提取能量。 思想实验的一个版本认为，一个装满气体的容器可以被分成两半，两半都处于相同的温度。 麦克斯韦的恶魔打开隔板上的陷阱门，使速度较快的粒子聚集在容器的一侧，并关闭门以阻止速度较快的粒子返回，从而使速度较慢的粒子保持在另一侧。 随着时间的推移，速度较快的粒子所在的一侧将比速度较慢的粒子所在的一侧温度更高。 看起来可以提取能量，违反了热力学第二定律。

最初人们认为（参见Szilard [99, 98]和Brillouin [16]），作为一个恶魔需要测量一个粒子的位置-动量坐标以知道何时打开陷阱门，恶魔需要与所研究的系统进行相互作用。 然后有人认为，测量过程将需要相应于每个获得的信息位的熵增加 $k_B ln2$。

Landauer [71]证明了这种解释并不完全令人满意，因为热力学上可逆的过程不能导致总体熵的增加。 Landauer指出，过程的不可逆性是关键。这种不可逆性反映在物理系统进行的计算的逻辑上的不可逆性中。 当关于先前状态的信息不可用时，这种情况就会发生，无论是因为信息已被丢弃，还是因为存在多个先前状态。 因此，当等效的计算过程变得逻辑上不可逆时，系统就不再是热力学上可逆的。 Landauer的方法解决了麦克斯韦的恶魔悖论 - 这个智能的干预者本来可以无成本地提取能量。 Bennett [12, 13, 11, 14]和Zurek [109]表明，恶魔不能违反热力学第二定律，因为恶魔必须抹去在制作一个

测量，为了给下一次测量腾出空间。 除非它通过将能量返回给环境来实现这一点，否则恶魔将需要持续不断地供应能量来翻转新的位，从而导致其温度升高。
恶魔实际上是整个系统的一部分，并受到物理定律的约束。 Bennett（参见[109]）以一个粒子的气体为例，并使用恶魔将粒子困在分隔板的一侧以提取能量。 当恶魔的记忆被重置以允许过程循环时，熵损失发生。 然而，正如Bennett [14]指出的那样，存在一些微妙之处。当信息最初是随机的时，如果随机信息被抹去，系统的熵会下降，但环境的熵仍然增加。 可以将这种抹去想象为将所有位设置为零，从而有效地对系统进行排序并将能量转移到其他地方。 关于麦克斯韦恶魔问题的不同解释以及涉及抹去和逻辑不可逆性的“新解决方案”的关键论文可以在Leff和Rex [73]中找到。

模拟一个不可逆计算对可逆真实世界计算的影响

兰道尔原理不仅对可逆性有重要影响，还对AIT、热力学第二定律和调节有重要影响。 然而，真实世界计算与典型参考计算机之间存在重要区别。 真实世界计算中的广义门（即原子、分子和计算元素）是可逆的，它们在计算过程中通过系统传递信息并改变自身状态。 可逆性通过决定计算行为的物理定律被硬编码到计算过程中的组件（如原子和分子）中。 计算的输入字符串存储在计算开始时原子或分子的初始状态中，而在给定初始状态的情况下，物理定律决定系统通过其状态空间的轨迹。 同样，正如兰道尔[72]指出的那样，任何传统的UTM也受到物理定律的约束，也是一个真实世界设备。 在这种情况下，输入和输出不是写在输入/输出设备（如磁带）上的数字，而是计算机在计算开始和结束时存储的位的初始配置和最终配置。 确定系统如何在其位空间中进行步进的计算是由初始位设置决定的，其中包括操纵嵌入在门中的物理定律的程序[71, 72, 100]。 然而，典型参考UTM的AND门和OR门是不可逆的。

虽然可以构建可逆门，但在实践中，当需要模拟真实世界的UTM时，需要一个不可逆的参考UTM，其中包含了可逆性所需的计算历史信息必须被保留和跟踪。 正如下一节所概述的，Bennett [11] 列出了从这个计算历史生成初始状态所需的步骤。 可以通过氧气和氢气燃烧形成水并增加系统温度的例子来看出这是必要的。如果没有什么

转义，这是一个可逆过程。 给定系统的初始状态，定义算法熵的参考UTM上的程序必须还要指定最终的动量状态，而不仅仅是原子和分子种类。 这些动量状态包含了可逆性所需的历史信息。 如果热量不可逆地丢失，降低温度并留下水，传统计算机必须丢失与这些动量状态相关的位，以模拟真实世界系统的最终配置。 这是一个非常重要的观点，本书将在全书中多次提及。 简而言之，真实世界计算机的状态空间配置被包含在Landauer [72]和Bennett [14]所称的信息载体自由度（IBDF）中。 参考UTM的位空间配置捕捉了与其模拟的真实世界计算机相同的信息，前提是参考UTM中的历史信息被正确跟踪。

以上概述的兰道尔原理阐明了物理系统进行计算过程的基本要素。 计算的输入字符串存储在物理过程开始时的原子和分子状态中。 在一个孤立的计算机系统中，比特被保留，能量也被保留。 因为系统的哈密顿动力学保持信息，所以当一个过程变得不可逆时，信息不再被保留；即当信息丢失时，必须从至少具有相应熵增的系统中转移出2自由度的熵单位。 正如兰道尔所示，关键不是信息的测量成本，而是当信息通过转移到外部环境时产生的能量成本。 正是这个原因导致了不可逆性。 $_B$ennett [12, 13, 11, 14] 和 Zure$k$ [109] 进一步阐述了这个论点。 从温度为T的I$_B$DF中丢弃1比特的信息，对环境贡献了约$k_B T\ln 2$焦耳的能量。 当信息从计算中移除时，逻辑和热力学的不可逆性都会发生。 这导致系统的熵减少，同时宇宙的熵增加。 当热量从系统中传递出去或者物质逃逸时，不可逆性就会发生，物质携带着信息的自由度。 将信息传递到系统中也会破坏可逆性，除非传递过程保持可逆性。 Li和Vit´anyi [76]的第8章回顾了与可逆性和现实世界计算的热力学相关的问题。 图6.1是传统计算机位空间中不同区域的示意图。 这些区域在传统计算机中更或多或少是不同的，而在现实世界的计算中，表达物理定律的程序区域并不如此清晰定义。 在传统计算机中，程序区域中的比特设置在程序员的控制下确定了输入在位空间中的轨迹，直到输出。 尽管在计算机中，输入和程序被编码为“开”和“关”的比特，但程序和输入之间的区别变得任意。 Chaitin [27]中展示了一个简单UTM的更严格的论证，证明了这一点。

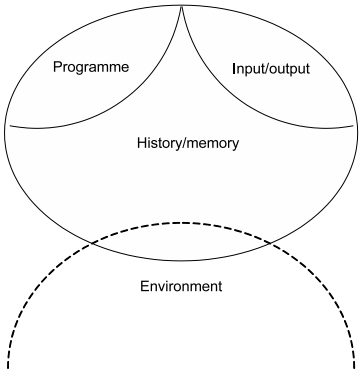

图6.1：计算系统的示意图

对于类似的物理情况（如第1.2节的第2个要点所概述的），可以忽略常见例程对算法熵的贡献。 由于只有熵差才是显著的，因此常见例程（如相关物理定律和系统规范）会相互抵消。 这是非常幸运的，因为规范化所有自然定律将是一项不可能的任务。 实际上，可以将熵零设置为大多数定律都是给定的，并且只考虑那些需要描述状态变化的定律。

Fredkin的简单弹道计算机[54]说明了物理过程如何充当计算。 然而，一个更相关的例子是

展示了物理或化学过程与计算过程之间的关系，是Bennett的布朗计算机[12, 11]。 这是RNA聚合酶复制DNA字符串的过程中体现的计算。 只有在零速度下才能实现可逆性，因为计算会随机地在可能的计算路径中进行。 事实上，由于自然DNA复制的不可逆误差纠正例程的推动，该过程不再严格可逆。

6.1 算法熵和第二定律的影响

物理过程原则上是可逆的，并且可以映射到一个逻辑可逆的图灵机，该图灵机对输入字符串（表示初始状态）进行操作，以产生一个输出字符串（表示最终状态）。 计算是通过决定系统在其状态之间的轨迹的物理定律的操作来进行的。 然而，在实践中，大多数真实世界的系统都是开放系统，信息和物质可能进入或离开系统。 正如前面所提到的，在信息被移除（无论是因为能量被移除还是分子离开）时，系统的计算可能性会发生改变。

然而，可逆和不可逆过程不能在相同的参考计算机上进行比较，除非对于不可逆情况，跟踪维持可逆性所需的信息。 这是根据兰道尔原理得出的，无论是在传统的通用图灵机还是现实世界的计算机上，如果在计算过程中丢弃了有关先前状态的信息，则任何计算过程都无法逆转。

然而，如果将可逆性所需的信息存储在计算机内存中，可逆性可以在不可逆计算机中保留。 所保留的信息是计算历史。 如果这些信息可以在反向计算过程的关键步骤中插入，那么可逆性就变得可能。 现实世界的等效情况是保留将要传递给环境的信息，以保持可逆性。 一个例子是冰从水中结晶时产生的热量。 如果保留这种热量，过程就是可逆的，但如果将其传递给环境，可逆性就会丢失。

通常的不可逆UTM描述了一个不可逆的物理过程，当计算的历史丢失时。 然而，通过使用一个在正向和反向方向上都有效的算法，可以将可逆过程映射到一个不可逆UTM上，插入在不可逆计算步骤中丢失的信息[77, 11, 108]。

正如Bennett [11]详细展示的那样，这样一个可逆程序大体上具有以下能力：

- 从输入生成输出
- 复制输出

- 反向计算以消除累积的历史并重新生成输入。

- 交换重新生成的输入和复制的输出，以允许反向计算。

问题是，通常从描述输出给定输入的最短不可逆计算中导出的算法熵 $H(o|i)$ 与可逆计算有何关系。 让 $KR(\ o|\ i)$，或其等价形式$KR(i|o)$，是将相同物理过程映射的最短可逆计算的长度。 因为很可能需要额外的信息来使过程可逆，

$$H(o|i) \leq KR(o|i)\ or\ KR(i|o).$$

算法熵由最小可逆程序 p^*给出，该程序从输入在参考UTM上生成输出。它的大小不能超过将可逆物理过程映射为可逆程序的程序。

然而，由Ben-nett [12, 11]描述的布朗计算机中隐含的程序可能可以缩短，如果涉及到催化剂的路径被更直接的计算路径所替代。 没有催化剂的过程速度可能较慢，但结果将是相同的。 一些作者[17, 77, 105]已经考虑了在复制给定输出的计算机存储和计算步骤数量之间的权衡。 计算越短，实现可逆计算所需的信息存储就越多。 对于正在进行的计算，作为历史记录存储的信息必须被擦除。 只有当熵丢失到环境中时，这才能发生，使得过程更难以逆转。

算法熵在输入状态 i和输出状态 o之间的差异测量了系统进出的比特数当 i 转变为 o时。 这种熵差异与使用的通用图灵机无关。

如果要将系统从 o恢复到 i，则比特流必须被反转。
根据兰道尔原理[71]，维持秩序或恢复系统的热力学成本是每比特 $k_B log_2$。 这在下一节中将详细讨论。

Zurek [109]和Bennett等人[15]详细讨论了这一点，并指出在不可逆过程中，从宇宙传递给物理系统的最小熵为 $H(i) - H(o)$，其中 $H(i)$表示系统初始状态的信息内容， $H(o)$表示最终状态的信息内容。 如果系统最初有序而最终状态更加无序， $H(i) < H(o)$并且熵流入。在这种情况下，系统可以进行工作，就像热输入迫使气体对抗活塞做功一样。

熵的变化代表了用于指定原始输入字符串的最小位数增加和在最终输出描述中丢弃的最小位数之间的差异。 Zurek [109, 108] 认为，如果人们知道这些状态的确切描述，基于这些信息的循环过程将是提取能量最高效的。 通过考虑以下示例，可以看出这个论点的一致性。

- 理想气体对无摩擦活塞的绝热膨胀会增加空间的无序度。 活塞的运动实际上类似于一个程序，改变了气体组分的计算轨迹。算法熵是通过以每个气体粒子的位置和动量坐标来描述气体瞬时构型的最短描述得出的。从算法角度来看，尽管位置坐标的算法熵通过无序化而增加，但动量坐标的算法熵减少以进行补偿。 这种减少对应于温度的下降。 绝热压缩是相反的过程。 绝热去磁类似于绝热膨胀。 随着外部磁场的减小，绝热去磁将一组对齐的自旋随机化。 这种无序化吸收了热度自由度中所包含的熵，降低了系统的温度。 当系统保持孤立时，不会发生信息的抹除或总熵的改变。

- 另一方面，气体对活塞的等温膨胀可以使系统无序或随机化，因为热量进入以对抗活塞做功。 与绝热情况相比，随着热量流入，系统的算法描述增加。热力学熵每位算法描述的位变化增加 $k_B ln2$。 相反地，等温压缩使气体有序。对于等温磁性系统，等温压缩的等效物是外部磁场被应用于使自旋对齐，并通过这样做将 $k_B T ln2$焦耳/自旋传递给环境。

- 在一个开放的真实世界系统中，重置计算元素（即原子、分子、自旋等）的计算过程需要消耗或释放能量。 这可以通过电能、使自旋对齐的磁场梯度、进入或离开系统的光子和分子结构、热量流入或流出系统以及系统上或由系统完成的工作来实现。 系统内的能源也可以重新分配以重置计算状态，例如，当氢和氧被这些真实世界的计算过程转化为水时。在氢氧情况下，释放的能量传递给系统的动能状态。 只要系统是孤立的，就没有熵的变化。 该过程增加了动量状态的算法描述的长度，同时减少了位置坐标的算法描述的长度。 当一个水分子取代一个氢原子和一个氧原子时，物种的描述也会发生变化。 系统内的物理（或计算）过程会分散熵。 与环境相比，那些在动能自由度中体现的无序区域可以将多余的熵传递给环境，留下更有序的区域。

仅当总体上排出熵时才会发生排序 - 例如当热量传递到环境中的低熵汇或高熵的分子碎片逃逸系统时。 关于参考UTM的程序必须捕获与真实世界UTM相同的信息。而真实世界的状态是由物理定律的操作设置的，在参考计算中，程序捕获了定律所体现的信息。 当物质进入或离开真实世界系统时，扩展或收缩状态空间，参考计算机中的相应过程涉及扩展或收缩位空间。 这可能对应于位传递到与环境相对应的计算机内存区域。 或者，它可以通过连接或移除存储设备（例如存储棒）来发生。 例如，当物种进入真实世界系统并通过其状态空间改变轨迹时，相应的参考计算机可以被设想为通过添加程序位通过存储棒进行的程序更改。

6.2 将系统恢复到先前状态的成本

正如已经提到的，热力学第二定律的退化过程可以被视为一种计算过程。 输入字符串 i指定了系统的组成部分，营养物质和任何浓缩能源的来源。 程序字符串 p指定了体现物理定律的算法，这些定律尚未明确表达。 输出字符串 o是由计算机 U上的计算生成的，其中 $U(p^*, i) = o$。 如果所有信息都保留在计算中；即没有任何信息逃离系统，该过程是热力学和计算可逆的。

然而，当系统通过热传递或物质离开系统而失去信息时，该过程变得不可逆。 如果系统要恢复到其原始宏观状态，必须替换这些信息。 这意味着一个生命系统必须通过抵消热力学第二定律所暗示的退化来维持自身处于远离平衡的配置。 第1.2节介绍了Chaitin的组织程度 D_{org}作为系统远离平衡的度量。 如果平衡配置中有Ω_{set}个状态，则典型字符串的算法熵将接近 $log_2\Omega_{set}$。 用 s 表示有序字符串的算法熵为 $H(s)$， $D_{org} = log_2\Omega_{set} - H(s)$。系统远离平衡的这种距离只能通过注入适当的秩序和排除无序来维持。

Zurek [109]已经应用了第6节的论证来量化恢复系统到其输入状态的成本 i 当信息被丢弃以产生输出状态 o。 为了至少恢复系统，$|i^*| - |o^*| =$

$H(i) - H(o) = H(i|o^*)$必须返回给系统[109]。 虽然Zurek[108, 109]还表明恢复过程的稍强形式涉及向系统返回 $H(i|o)$位，但较弱版本生成 i

从 o^*提供了恢复成本的下界。 较弱版本更有用，因为它是反对称的，因此可以追踪热力学循环中的熵增益和损失的方向（附录A，[15]）。 此外，恢复路径不需要是正向路径的精确逆计算。

考虑一个最初处于微状态ε_i的系统，它属于一个稳定状态集合ε_i，ε_j等等，这些状态具有相同的临时熵$Hprov(\varepsilon_i)$。 扰动δ将系统的轨迹切换到稳定集合之外的新配置η_j。这类似于将一个带有计算机病毒的存储器插入计算机。 扰动或病毒引起的相应计算$U(p^\star\delta, \varepsilon^\star{}_i) = \eta_j$。 由于扰动进入系统的比特数为

$$|p_{\star\delta}| = H(\eta_j) - Hp_{rov}(\varepsilon i)。$$

这里正号表示熵流入。 这个度量是一个熵差异，无论是在现实世界的计算机上还是在参考的UTM上都是相同的。应用Zurek的论证，至少需要返回$H_{prov}(\epsilon_i|\eta_j^*)) =$
$H_{prov}(\epsilon_i)) - H(\eta_j)$比特必须返回到系统中，以将其恢复到原始稳定集合中的一个状态。这导致了热力学要求方程A.1中至少

$$k_B ln2 H(\epsilon_i|\eta_j^*) = k_B ln2[H_{prov}(\epsilon_i) - H(\eta_j)] \tag{6.1}$$

每度必须引入焦耳到系统中以恢复 i。 当然，实际成本可能更高，因为系统很少是热力学上高效的。

Ashby [5]认为，一个开放系统必须具备调节能力，以适应不断变化的环境，以保持稳态。 从控制论的角度来看，维持系统变量在可行范围内需要一个外部调节器。 这种方法导致了Ashby的必要多样性定律，该定律指出系统调节的要求是调节状态集合的香农熵至少等于干扰集合的香农熵[5] [22]。 从计算的角度来看，Zurek的上述论证表明，当外部调节器在系统内外传递信息时，类似于Ashby的必要多样性定律的法则出现。在这里，该法则变为：“纠正过程引入的算法熵必须等于干扰引入的算法熵。”

这种观点认为干扰和调节器是外部系统，并且两者都对其起作用。然而，当系统远离平衡时，驱使系统朝向平衡的第二定律过程就起到了干扰的作用，但在这种情况下，它们是系统的一部分。 同样，当系统自我调节时，调节过程在系统内部起作用。 例如，森林可以通过利用太阳光子来维持自身。如果光线被阻挡，森林将降解到局部平衡。 第9.6节推广了上述讨论的论点，针对在第二定律退化过程下运行的自我调节系统的情况。

6.3 第二定律的算法等效性

热力学第二定律要求孤立系统的熵在系统轨迹从非平衡态向平衡态移动时达到最大值。玻尔兹曼方法通过系统可用状态的增加来解释熵的增加，因为能量变得均匀分散。 最终在平衡态时，可用状态的数量为Ω，玻尔兹曼熵变为 $k_B ln \Omega$。 另一方面，算法方法将熵增量定义为系统的微观状态从有序态向典型态或平衡态移动时算法描述长度的增加。这种增加实际上是由于在计算开始时必须添加到系统中的程序位数，以便计算在适当的停机态结束。 正如后面所讨论的，在平衡态下，轨迹的历史被丢失，绝大多数状态需要长度为$\log_2\Omega + O(log_2\Omega)$的算法描述。 相应的算法熵接近 $k_B log_2\Omega$，这个值与平衡态下的玻尔兹曼熵 $k_B ln\Omega$相吻合。

当系统既可逆又孤立时，不会丢失任何信息，每个状态都有唯一的前驱。本质上，系统至少是准遍历的，系统的轨迹将在Ω步中访问所有Ω个状态。

这样一个系统的演化可以被追踪。 第二定律过程的算法细节，即一个孤立的真实世界经典系统从一个初始有序状态 s_0到达平衡状态的过程，已经在Zurek的文章附录C中得到了发展[108]。 假设有一个有限但很大的可访问状态集，或者通过选择适当的分辨率来定义一个离散的状态集。 考虑一个孤立的可逆自然系统，其中初始状态 s_0是系统中最高有序的状态，其算法熵为 $H(s_0)$。 让 t_h是从状态 s_0输出状态 s_h并停止的计算步骤数。 将 s_0转移到 s_h的真实世界计算机具有定义轨迹的物理定律，这些定律体现在广义门中，是计算门的自然等效物。

模拟这条轨迹的可逆参考UTM还必须模拟驱动系统演化的程序中的物理定律。 由于系统演化的每一步都是可逆的，将物理定律映射到参考UTM上以产生最终或停止状态的可逆算法可以用以下示意图形式表示（例如参见Zurek，[108]）：

$$\begin{array}{l}\text{状态 } =_0\\ \text{对于步骤}=0\text{到}t_h\\ \quad\text{计算下一个状态。}\\ \text{下一步。}\end{array} \tag{6.2}$$

只有当该算法是最短的时，其长度才能给出算法熵 s_h。 由于这个计算映射了可逆的物理定律，当保留计算过程的历史时，不可逆的参考UTM上也允许可逆性。 在这种可逆的情况下，第二定律过程之前和之后的总熵是恒定的。 表观熵增加以位或状态设置的形式进入系统，这些位或状态设置体现在程序中，包括指定停止指令 t_h的位。 然后可以根据其历史从最终状态重新构建初始程序和初始状态的位设置[11, 13]。 此外，由于科学是建立在自然定律简单而少的假设上的，表示物理定律的子程序（即计算下一个状态）相对于整个程序的大小来说应该高度压缩，随着 t_h的增长。 该算法长度（即方程6.2）的主要贡献是 $log_2 t_h$，其中t_h是到达停止状态的步骤数。 程序位与计算机的初始设置一样重要，就像起始状态 s_0一样，并且在某种意义上代表了计算的前历史。 算法熵预计由初始状态的熵、例程的长度组成

$H(t_h)$ 指定了停机要求和例程中的熵计算下一个状态的例程的熵。即如果生成轨迹的二进制算法是最短的，算法熵由以下给出：

$$H(s_h) = H(s_0) + H(t_h) + H(\text{计算下一个状态} \quad) + O(1). \tag{6.3}$$

上述方程表明最终的算法熵由初始状态的熵， $H(t_h)$，指定停机配置的例程的长度以及计算下一个状态的例程的长度组成。 严格来说，上述方程应该有一个 $\leq$，但对于较大的 t_h，如下所示，较短的算法描述极其罕见。

虽然这个过程的每一步都是可逆的，但任何轻微的前进驱动都会使得当 t_h变大时，可逆性变得不可行。 然而，一般情况下，如第6节所讨论的，当历史被保留时，没有熵丢失给环境，过程在逻辑上和热力学上是可逆的[12, 11, 109]。 此时需要解决一个概念问题。

正如后面第9.5节中所讨论的，上述方程6.3意味着熵
在可逆系统从 s_0移动到 s_h时是增加的。然而存在一个
悖论。 如果没有熵传递给
环境，自然过程的熵沿着可逆路径必须保持不变[12, 11, 109]。 一旦意识到
系统是孤立的，并且驱动程序位中所包含的熵也被认定为
系统初始设置的一部分，这个悖论就得到了解决。

计算中所包含的熵必须被视为系统初始设置的一部分。 如果 p_h表示生成停机状态的程序位设置，给定 s_0，初始配置的最短描述，包括程序和所谓的初始状态，严格地是$(p_h s_0)^*$。 请注意， p_h不是常数，当停机配置增加一步时， p_h中的位数将需要增加以覆盖这一点。 由于通过位空间可逆地生成最终状态，熵是守恒的[108]，并且

$$H(p_h s_0) = H(s_h) = H(s_0) + H(p_h).$$

由于没有其他路径可以到达 s_h（除了从后继状态 s_{h+1}进行反向计算），因此这个（或从 s_{h+1}进行反向计算）必须构成生成停机状态的最短可逆算法，并且

$$H(s_h) = H(s_0) + H(s_h|s_0^*) = H(p_h s_0) \tag{6.4}$$

由于系统是孤立的，程序位被传输到字符串s_h并成为输出的一部分。 因此，可逆性的要求体现在 s_h中。 如果使用不可逆的UTM来映射可逆过程，则可以通过保留计算的历史记录或将初始状态与最终状态一起保存在内存中（[108]）来保持可逆性。 当可逆系统按照等式6.3中的允许状态进行步进时， $H(t_h)$ 的平均值接近于 $log_2 t_h$曲线（忽略用于自限定编码的更高对数项）。 虽然等式6.2的平均增长为 $H(t_h)$，但正如Li和Vitanyi [76]的图3.2所示， $H(t_h)$ 在某些情况下会下降低于这个值。 这是因为某些值存在更短的整数 t_h的描述。 当 $logt_h$接近 $log_2\Omega$时，给定由 $|\ t_h|$表示的t_h的规范的大小变化非常缓慢，系统会稳定在平衡配置集中。 典型或平衡状态的熵为

$$H(s_{equil}) \approx log_2\Omega$$

一旦历史作为废物传递给外部环境（例如，作为振动自由度的热量），剩下的字符串可以用一个不可逆算法来描述，该算法比轨迹短。

即， s_h 由 $s_h = \hat{s}_h w_h$ 给出，其中$\hat{s}_h$ 在计算过程的内存作为废弃字符串 w_h被弹出时出现。 如果可以通过比指定t_h更短的不可逆算法生成状态$\hat{}\ s_h$，则该状态将更有序。

问题是上述步进算法是否代表了系统停止状态的最短描述，或者是否可以找到一个更短的算法来描述最终状态$\hat{s}_h$。 如下所示，对于远离平衡的演化系统，绝大多数构型的算法熵都来自于方程6.3的步进或演化算法，其中等号成立。

物理定律和起始状态支撑着进化算法，因此描述演化系统的算法高度压缩，

因此系统的描述主要由 t_h的规范决定。即

$$log_2 t_h >> |\text{计算下一个状态}|。$$

随着系统的演化，每个连续的停机状态（由 t_h给出）都会遍历表示0到 t_h之间的自限定字符串。 因为没有比长度为m的程序更短的程序多于$2\ m$ 个，所以比 $m\ -^{10}$短的程序最多只能有1个在210 个程序中。 这大约是1000分之1。 实际较短程序的数量远远少于这个，因为自限定编码的要求消除了许多可能的程序。

对于绝大多数结果来说，方程6.3的步进或演化算法代表了最压缩的最终状态描述， 因此可以假设在绝大多数情况下方程6.3中的等号成立。

然而，当t指定一个平衡状态时，上述论证就会崩溃。 随着t的增长，算法熵将被$H(t)$所主导，体现了停机要求。 除了罕见的偶然排序之外，算法熵最初平均增长为$\log_2(t)$并紧贴$\log_2(t)$曲线[76]，这表明系统的配置在达到停机状态所需的步骤数增加时变得更加无序。

最终，当t变得足够大时，系统将处于平衡配置。 如果t=$\alpha\Omega$，其中α小于1但趋近于1，算法将受到$\log_2(\Omega) + \log_2(\alpha)$的主导。 因此，平衡配置的集合将具有给定的算法熵。

$$H(s_{equil}) \approx log_2\Omega + log_2\alpha.$$

当 $log_2\alpha$相对于 $log_2\Omega$可以忽略时，方程简化为 $log_2\Omega$.
临时算法熵等于玻尔兹曼值，但单位不同。 就像系统演化过程中发生的那样，即使与环境没有熵交换，也可能发生与平衡态偏离的自发波动。

同样，在远离平衡态的一组具有共同结构的状态存在时，指定轨迹的算法可能比指定该状态集合的共同模式或结构的算法更长。 在这种情况下，该状态集合的临时熵提供了最短的算法描述，而不是轨迹算法。

在特定情况下，当演化算法本身涉及基本单元的复制时，最短算法是直接描述结构的算法，还是使用演化算法来映射结构的增长的算法不清楚。即从种子生成树的过程的算法是否比直接定义树的算法更短，后者通过指定树的每个单元的最终状态以及单元类型和单元如何组合在一起来定义树。 逐个指定树的每个单元的放置和单元类型可能需要比从遗传密码生成树的算法更长的算法。 原因是基于从树生成的算法

遗传密码，根据邻近细胞的环境和外部资源输入，指定某些基因将被表达。细胞类型由基因表达确定。 也许一个更简单的论点是将自然界视为一台计算机。 在真实世界计算机上生成树的程序涉及种子中体现的算法、生长种子所需的外部资源以及决定树生长程度的最终停止状态。 如上所述，对于绝大多数结果，不存在更短的程序。 因此，从起始条件（种子的初始状态、资源输入和程序）推导算法熵在几乎所有情况下都比从指定树最终状态的算法中推导度量更短。

将在第9.7节讨论在这些情况下熵的变化跟踪。

第7章

最小描述级别方法

最小描述长度方法（MDL）是由Rissanen [83, 87]（参见Barron等人[7]）开发的一种实用方法，用于找到描述数据点序列的最佳模型。 基本上有两种形式的MDL。第一种是真正的算法方法，被称为理想MDL [104]。 这是一种寻找

数据集最短描述的理论方法，由两部分算法组成。 第一个算法使用假设或模型捕捉数据的规律性。 第二部分是根据模型捕捉实际数据与规律性之间的偏差的算法。 如果模型过于简单，可能会有一个短的描述，但这将被模型所暗示的与规律性偏差的更长描述所抵消。 显然存在一个最优解。 算法或理想版本是AMSS或临时熵方法在第4.3节中描述的一种扩展。然而，虽然理想MDL方法找到了最适合数据的算法，但它不会按照算法信息论的要求生成数据。然而，一旦附加了一个简短的解码例程，数据就可以从最优编码算法中生成。 在这种情况下，加上这个解码例程的MDL算法的长度几乎与临时熵相同。 在理想形式中，模型捕捉了集合的规律性，并且给定集合可以识别出感兴趣的字符串。与临时熵一样，理想方法也面临着算法描述不可计算以及没有系统的程序来决定假设或模型中体现的理解是否是最佳的问题。 可以研究不同的模型，但这个过程是特定的。

实用的MDL，第二种方法，通过用随机复杂性取代算法复杂性的思想来解决非可计算性问题。即，与其找到最短实际算法的长度，随机复杂性寻求基于香农编码定理的最小代码长度。正如后面所概述的，实用的MDL提供了一种选择工具

第94章 最小描述级别方法

通过系统化的过程从一类模型中选择最佳模型。 如果类别足够广泛，例如多项式类别或马尔可夫模型类别，该过程可以确定哪个模型以最少的比特数描述数据。 不可避免地，最优性是相对于所选择的类别或类别的。 如果不是这样，该方法可以用于确定所有物理定律。

在接下来的内容中，将讨论理想的或算法性的MDL和MDL的一般原则。 在此之后，将简要概述实际的MDL方法作为模型选择过程。 一旦建立了一个模型，它可以用来定义适当的集合，就像为临时熵方法所做的那样。 然而，由于实际的MDL方法是一个独立的学科，它不是本文的重点。 相反，本文的主要重点将放在算法版本及其与临时熵的对应上。 但首先需要重新讨论最优编码（3.3）。

本节要讨论的大部分内容与第4.4节中讨论的内容相同，即将算法方法推广到模型开发中。 这种推广方法与最小描述级别几乎完全相同。取一个表示计算过程输出数据序列（y_1，y_2，$y_3...y^n$）的字符串 $y^{(n)}$，该序列是给定输入数据字符串$x^{(}{}_n)$的输出数据。 理想的MDL方法提供了一种更加形式化的方法，将算法信息论与模型选择联系起来。 正如在第4.4节中提到的，模型实际上是一种压缩数据的方式。

一个好的模型是具有数据的最大压缩描述的模型。
以牛顿定律下的行星运动为例。 规律性在实施定律的算法中。这就是模型。 每个数据点的变化（可能是由于模型的限制、噪声或实验误差）将以随机偏差的形式出现。 特定数据点的最佳压缩由临时熵或算法最小充分统计量给出，如第4.2节所讨论的。模型指定了有效结果集的特征，而偏差或偏离理想的代码由识别集中的特定字符串。偏差将呈现随机性，因为如果不是这样，模型可以被修改以获得更好的拟合。 理想的MDL方法为每个数据点选择算法最小充分统计量（AMSS）或临时熵的最佳估计。 如下所述，模型定义的集合中的所有典型字符串将具有相同的描述长度。 一些非典型的字符串将具有较短的描述。 下面的段落通过基于贝叶斯规则的替代推导来清楚地说明这一点。 这对于提供洞察力和与MDL的实际版本的联系非常有帮助。L_i和$\mathrm{V}_i\mathrm{t}\acute{}\,\mathrm{an}y_i$ [76]概述了这些原则。

回到贝叶斯规则，在第3.8节中概述，给定数据 D，从一组备选假设或模型 $\mathcal{H}_i$中，特定假设的概率满足：

$$P(\mathcal{H}_i|D) = P(D|\mathcal{H}_i)P(\mathcal{H}_i)/P(D)。 \tag{7.1}$$

假设或者在我们的情况下是模型，可以被表述为物理定律；一个简单的模型（例如对数据的直线拟合）；或者来自一类模型，例如多项式、伯努利或马尔可夫模型。 下面概述的方法并不确定可能模型的参数；相反，给定从多个可能模型中获得的最佳拟合，该方法确定哪个最佳拟合被优选。 例如，对数据的试验拟合可以使用基于最小二乘法等标准拟合过程的直线。 然后可以尝试二次多项式等。该过程确定每个潜在模型的最佳参数。 在给定这些带有参数的模型的情况下，MDL方法确定哪个可能拟合提供了数据的最短算法描述。 因为高阶多项式需要更多的参数，所以高阶多项式需要更长的描述。 只有当与模型的偏差足够减少以弥补更长的描述时，才会优选这样的多项式。

理想的MDL方法是通过取Bayes方程（7.1）的负对数得到的，即

$$-log_2P(\mathcal{H}_i|D) = -log_2P(D|\mathcal{H}_i) - log_2P(\mathcal{H}_i) + log_2P(D). \qquad (7.2)$$

Bayes方法选择最大化 $P(\mathcal{H}_i|D)$的假设。 这相当于选择最小化方程（7.2）右侧的假设，并且通过这样做，提供了数据的最短描述。

然而，由于 $log_2P(D)$与假设或模型无关，它对最小化过程无关紧要，可以忽略。 在上述方程中，通常情况下，先验概率未知，可以使用通用分布$m(\mathcal{H}_i)$作为先验（见第3.8节）来替代 $P(\mathcal{H}_i)$。 通用分布是未知概率的最佳估计。 此外，只要满足下面所述的限制条件， $m(D|\mathcal{H}_i)$也可以替代 $P(D|\mathcal{H}_i)$，并且要最小化的方程变为：

$$-log_2m(D|\mathcal{H}_i) - log_2m(\mathcal{H}_i).$$

由于香农编码定理的算法版本（第3.8节），通用分布 $m(.)$ 等于 $2^{-H(.)}$. 因此 $H(\mathcal{H}_I$，模型假设 $\mathcal{H}_i$的最短算法描述，可以替代 $log_2m(\mathcal{H}_i)$在上述方程中。这导致了上述方程在算法熵术语中的等效表达式。即，在 $O(1)$常数内，最优模式最小化：

$$H_{algo}(D|\mathcal{H}_i) + H_{algo}(\mathcal{H}_i).$$

这相当于找到假设或模型的临时熵
其中

$$H_{prov}(D) = H_{algo}(D|\mathcal{H}_i) + H_{algo}(\mathcal{H}_i).$$

即最小化临时熵或选择由方程4.6和4.8给出的算法最小充分统计量的假设是首选。 如果有多个可能性具有相同的最小化，则

选择的假设 $\mathcal{H}_i$是具有最短 $H(\mathcal{H}_i)$的那个。 有效地说，根据可用信息，最短的算法描述是首选的假设。

对于有限的假设集，贝叶斯方法与理想的MDL重叠，使用 $m(\mathcal{H}_i)$作为先验概率，前提是将 $H(D|\mathcal{H}_i)$替换为 $m(D|\mathcal{H}_i)$是有效的。 只有在数据字符串，由 D表示的情况下，才能进行这种替换，该字符串是由假设隐含的数据集的典型成员。 要成为典型的，该字符串必须在集合中是随机的。这可以通过考虑一个将特定字符串描述为伯努利过程的模型来说明，其中'1'的概率为 p，'0'的概率为$1-p$。 如果假设是 p= 0.75，该方法只在结果是满足假设的字符串集合中的典型字符串时才有效。 75个1后面跟着25个0的字符串不是可能字符串中的典型字符串。 上述特定字符串只对不同模型而言是典型的（或随机的）。 一般来说，将 $m(D|\mathcal{H}_i)$替换为

当数据字符串相对于假设是随机的时，$H(D|\mathcal{H})$才有效。 如果数据字符串不典型，就像在AMZZ案例中一样，需要一个新的假设或模型。 当找到正确的假设，并且集合中的每个成员在新假设下是等可能的时，$\log_2[P(D|\mathcal{H})] = -log_2(1/\Omega)$，其中$\Omega$是新集合中的成员数。同样，在算法术语中，$H(D|\mathcal{H}) = log_2\Omega$。 此外，像AMSS和临时熵方法一样，最小描述长度编码将字符串编码为两部分：一部分定义模型，另一部分指定特定字符串，给定模型。 或者，这个模型可以从描述理论的第一部分和描述如何在理论的基础上精确拟合带有偏差的数据的第二部分来思考。

严格来说，虽然MDL可以识别出最优的假设或模型，但只有在包含一个简短的解码程序以从模型的参数生成字符串时，它才成为一种算法测量。

MDL方法再次强调了这样一个事实，即对观测集合 $y^{(n)}= (y_1, y_2, y_3, \ldots y_n)$的最压缩描述涉及找到最适合观测集合的模型，以最小化每个数据点 y_i与模型之间的偏差。 理想的MDL方法与第4.4节中更详细概述的论证完全相同。例如，用于MDL方法的直线模型将尝试通过方程 $y_i = ax_i + b + noise$来解释一组带有噪声的数据。 这里的假设是直线模型，但也包括一个概率分布来覆盖噪声或变化。 最小描述方法最小化了模型的算法描述长度以及由噪声引起的任何偏差。 这种方法需要$2d$位来指定常数 a和 b的精度 d，参数 c和 σ以相同的精度指定分布。 使用基于Shannon编码定理的编码来指定与模型的偏差，如第4.4节中概述的那样。在线性情况下，其中 $y(x)$是一条直线，对于特定情况的最小描述为

给定输入的字符串，变成了临时熵：

$$H_{prov}(y_i) = H[(ax_i^{(n)} + b), P] + |code[y_i - y(x_i)]| + |\text{解码例程} \quad |$$

这里 $|code(y_i - y(x_i))| = -log_2 P(y_i - y(x_i)$，需要将概率四舍五入为整数编码。一旦MDL选择了最佳模型或等效集合，并且满足了典型性要求，该模型就为特定字符串提供了临时熵。然而，总是可能存在一个更短的描述，而这个更短的描述可能指向一个更好的模型。为了确定哪个模型是最佳的，一个更实际的过程克服了理想MDL中的不可计算性问题，在下一节中进行了讨论。

实用的MDL

Rissanen开发的实用MDL方法通过定义随机复杂性来替代算法复杂性[83, 86, 87]，从而绕过了理想MDL方法中的可计算性问题。 随机复杂性是可计算的。它对应于基于概率的最优编码来描述观测数据字符串的最小描述，而不是基于算法（尽管有时最高效的编码可能与最短算法几乎相同）。 因此，随机复杂性被定义为相对于一类概率模型的数据的最短描述。 随机复杂性受到AIT方法和Solomonoff的归纳方法的启发（见第3.8节）。评论可参见Gr¨unwald [60]和Hansen和Yu [61]。

Shannon的编码定理（在第3.3节中讨论）表明，可以为一个由概率 $P(\ y_i)$ 分布的可数对象集找到最优编码。 这些对象可以是字符串，在本节中，这些字符串被认为是从特定模型派生的。正如第3.3节所指出的，Shannon的无噪声编码定理（方程式（3.12））证明了可以选择来自前缀自由集的最优编码，使其几乎等于四舍五入的概率，即：

$$|\text{代码}(\mathsf{Y}_i)| = -\lceil log_2(P(\mathsf{Y}_i))\rceil 。$$

由于概率通常不是整数，编码长度将是下一个最大整数。 惯例是用 $-log_2 P$表示编码长度，理解为整数。 另一个要点是，当概率模型表示为概率密度时，变量Y_i被取到选定的精度级别，例如四个有效数字。 在这种情况下，每个Y_i的概率 $-log_2 P(\mathsf{Y}_i)$被 $-log_2[P(\mathsf{Y}_i)\delta]$替代，其中$\delta$表示由分辨率确定的网格大小。 由于 δ在整个MDL优化过程中保持不变，可以忽略它。

再次，这种方法的原则是，模型对观察数据的拟合越好，描述就越紧凑。这表明，最好的模型或模型将是提供最短描述的模型。

然而，问题是，这个过程是否比选择假设或模型的其他统计方法更有优势。 一般来说，实际的MDL方法避免了过度拟合，即模型中的参数相对于数据过多，以及欠拟合，即模型过于简单。 当参数过多时，误差项很小，但整体描述包括模型描述会更长。 欠拟合数据会导致数据与简单模型的值之间的过度变化，这需要对偏差进行更长的描述。 例如，一个包含20个数据点的数据字符串可以通过简单的直线拟合，但偏差可能需要从$-log(probability)$项中得到大量贡献。 然而，一个具有20个参数的模型可以完全拟合，但模型描述可能太长。 最佳拟合在这两者之间。 从实际意义上讲，MDL满足奥卡姆剃刀原则。

这种方法与传统的数据拟合有一些重要的区别，因为它没有假设该过程产生了真实的模型或真实的参数来支持数据。 Rissanen指出[86]，最优模型不是对任何东西的近似；它只是适应数据。 实际的MDL方法可以从贝叶斯定理中推导出来，就像之前概述的理想MDL一样。或者，可以认为当数据被最大压缩时，所有有用的信息都已被提取出来。 最小化随机复杂性实际上是确保数据相对于模型类是随机的，并且所有模式都已被捕获在模型中，类似于AMSS方法。

给定一个数据集，实际的MDL方法寻求找到模型类中提供数据最短描述的成员。 通常，考虑的MDL模型类是用定义类的参数来表示的。 该类可以是一组线性回归，其中数据 y 是几个变量的线性函数，比如 θ_1 到 θ_n。 在非线性回归情况下，可以使用类似的一组参数 θ_1 到 θ_{k+1}来表示多项式的系数 $_k$。 在这种情况下，模型可以用参数集 θ 和模型标签 $_k$ 来表示。 一般的马尔可夫模型由其阶数 $_k$ 描述，表示定义未来所需的历史长度，而在这类模型中，参数集 θ 变成一组概率 $p(_k)$。 伯努利过程是最简单的马尔可夫模型，其中 $_k = 1$，概率为 p。 （注意这里 $P(y)$ 表示集合中字符串 y 的概率，而 p 是字符串 y 中下一个字符的概率。）

非预测版本[83, 84]表明，在某些情况下，长度为n的字符串的最小描述减少为：

$$-log_2 P(y|p^{(k)}, k) + (k/2)log_2 n + O(log_2 n).$$

第二项参数化模型类，而第一项是基于概率的编码，用于编码由参数 p^k和 k 指定的模型给定的数据。 考虑由伯努利过程生成的数据字符串，其中 r $1's$在长度为n的字符串中，且 k= 1。最短描述

对于大的n，最短描述取决于 $nh(p) + (log_2\ n)/2$，其中 h是熵率-每个符号的熵[85]。 概率的最大似然估计给出 $p = r/n$。 然而，如果要考虑更高阶的马尔可夫模型作为数据的更好解释， k将增加，增加$(k/2-1)log_2(n/2)$位。 如果这种增加超过了由概率项导出的代码长度的减少，那么后者的模型将被优先选择。 最优描述是在整个模型类中最小化总长度的描述。

自从Rissanen [83, 86, 87]的早期工作以来，实用的MDL方法已经显著发展。后续版本使用归一化的最大似然模型来更好地描述数据。

最大熵公式

Jaynes开发的“最大熵”方法是在建模系统时充分利用信息的尝试。 这种方法的核心是拉普拉斯的不可区分性原则，尽管传统的频率学派对概率的理解持怀疑态度，但它似乎有效。有趣的是，如下所述，最小描述长度方法可以用来证明Jaynes的最大熵方法。 换句话说，最小化描述长度等价于最大化香农熵。

在实际的实验情况中，特定结果的真实概率 p_i可能是未知的。 如果有 k个结果和$_in$个独立试验，并且结果 i发生了$_in\ i$次，则结果$_i$的概率由$_in\ i/n$近似。 然而，虽然$_in_i$可能实际上是未知的，但观察到的任何信息都确定了可能的结果分布的约束。 例如，一个约束应该是概率的总近似值应该为1，即$\Sigma_i n_i/n = 1$。 类似地，对于具有能量 E_i和总能量E的多个结果的能量分布，每个状态的平均观察能量受到$\Sigma E_{ii} n_i/n = E$的约束。这些约束限制了每个结果的可能概率值。

在这种情况下，最大熵原理可以通过确定概率分布 $\hat{p}_i$来识别最符合约束条件的模型。如果香农熵

$$H(p_1 \ldots, p_k) = -\Sigma_i p_i log_2 p_i. \tag{7.3}$$

当 p_i=$\hat{p}_i$时，熵达到最大值，概率集合$\hat{p}_i$是首选分布。

在最大熵的表述中，拉格朗日乘子用于捕捉约束中所包含的信息，以找到最大化熵的值$\hat{p}_i$。 这实际上是在推导麦克斯韦-玻尔兹曼统计时使用的方法。Li和Vit´anyi在第76页给出了所谓的布兰代斯骰子问题的例子（也可参考Jaynes [67]），其中掷骰子的平均结果显示出偏差。 观察到的偏差限制了可允许的概率。 最大化熵的概率集合$\hat{p}_i$，

在给定的约束条件下，然后找到。 以下概述了为什么最小化 $-log_2P(x|p)$等同于最大化香农熵。

一般来说，线性独立的约束可以用一系列方程表示：$\Sigma_j a_{i,j} n_j/n = x_i$，其中 x_i表示物理量的观测值（Rissanen [84]）。 将会有一组可能的分布 p= $(p_1, p_2 \ldots p_k)$，每个分布都满足上述约束，并且每个分布都有一个与之相关的概率作为解释约束的假设。 （注意这里的假设概率 P代表一个特定模型，与满足约束的一组特定概率 p有所区别。）任何可接受的概率分布都适用于数据，因此 $P(\ x|p) = 1$。 在这种情况下，与模型相关的理想编码长度应该是 $-log_2P(\ x|p) = 0$。 因此，最佳的概率集合是最小化 $-log_2P(x|p)$的集合。 对于大量的数据序列，这等同于最大化 $n!/(n_1!\ldots n_k!)$。 根据斯特林近似，这等同于最大化方程（7.3）的熵，证明了杰恩斯方法的有效性。

有趣的是，杰恩斯[68]的附录A认为科尔莫哥洛夫的更一般的概率方法与他自己的方法是一致的。

第8章

非典型字符串和随机性

8.1 随机性的观点概述

数学家们在定义随机性方面遇到了很大的困难。例如，冯·米塞斯[106]试图为无限序列定义一个随机性测试，要求所有子序列的相对频率收敛。尽管从直觉上来看这似乎是合理的，但数学家们已经发现了这种方法的弱点。然而，算法信息论提供了多个不同的随机性视角，最终导致了更健壮的方法。从算法信息论的角度来看，大多数字符串在随机性上是随机的，即它们不可压缩。然而，虽然长度为100的序列'111 . . . 11'与同样长度的任何其他字符串出现的概率是相同的，但我们不认为它是随机的。原因是'111 . . . 11'是可压缩的，因此不是所有长度为100的字符串集合中的典型成员。

典型性与统计热力学中的平衡概念有关。相空间（位置-动量空间）中气体的不同可能配置可以用一系列的1和0表示，取决于相空间中每个单元格是否被占据（见第5.2节）。在平衡配置中，绝大多数字符串都是典型的。然而，一些罕见的表示有序并且可以被压缩的字符串不是典型的。一个例子是配置'111 . . . 11100000 . . . 000'，由重复的1和重复的0组成。虽然这样一个高度有序的字符串可能偶然发生，但这种情况非常不可能发生。

科尔莫戈洛夫的同事马丁-洛夫认识到随机性与典型性和不可压缩性的概念有关。他开发了以他的名字命名的随机性测试。Chaitin发展了等价概念的m-随机性，这可能更容易理解。以下概述了AIT关于随机性的关键观点。然而，由于不同的观点可能有点令人不知所措，首先对其进行总结，并将在后续讨论中详细讨论。

更多细节在下面的章节中。

- 大多数字符串在下面讨论的情况下都是算法随机的，因为很少有字符串可以通过算法压缩超过10%。
- Martin-L¨of [80]定义了一种用于随机性的通用 P测试，它与任何可能的有效随机性测试一样好。 如果一个字符串在特定级别上被认为是随机的，那么没有任何有效的随机性测试可以做得更好。
- Kolmogorov开发了一种他称之为“随机性缺陷”的度量方法，即字符串越有序，其随机性缺陷越大。 随机性缺陷定义了一种随机性测试，它是Martin-L¨of P测试，因此是通用的。
- Chaitin的 m-随机性度量，其中 m是字符串可以通过算法压缩的字符数，等同于Martin-L¨of通用随机性测试，考虑到在Chaitin的情况下使用了自解码码（即来自前缀自由集的码）。
- 另一种替代测试是G´acs [56]的和 P-测试。 这背后的想法是，一个明智的投注过程可以在字符串被声称为随机但实际上是有序时获利。 和 P-测试的结果与基于随机性缺陷的Martin-L¨of P-测试几乎相同，只是和 P-测试要求计算限制在一个前缀自由的例程集合中。
- 算法信息论广泛使用普适概念；即在该类中占主导地位的概念。 在这个意义上，随机性的普适测试的概念类似于通用计算机的概念（加法占主导地位）和通用半测度的概念（乘法占主导地位）。

在详细讨论不同观点之前，首先要概述随机性缺陷的概念。 算法信息论认识到，给定长度的大多数实数都是算法随机的。 这可以看作是在给定长度小于某个n的情况下，有2^{n-1}个二进制字符串。也就是说，长度小于n的字符串少于2^n个。 因此，长度小于$n-m$的字符串少于2^{n-m}个。 例如，如果 $n=100$， $m=10$，长度为100的字符串有1.27×10^{30}个，而长度小于90的字符串只有1.24×10^{27}个。 因此，长度为100的字符串中，最多只有1024个字符串可以被压缩超过10%，因为较短的字符串数量太少，无法提供压缩描述。 实际上，可用的字符串会更少，因为许多较短的可能性已经被使用过。 例如，其中一些较短的字符串不可用，因为它们是长度大于 n的字符串的压缩表示；其他一些较短的字符串不可用，因为它们本身可以进一步压缩。 由于大多数字符串无法显著压缩，字符串 x的算法熵很可能接近于 $|x| + ||x||$；即

字符串的实际长度和其长度的编码大小。 那些无法压缩到小于$n - m$的字符串被称为“m-随机”（参见Chaitin [28]）。

在均匀分布的情况下，随机性的不足，即字符串的非随机程度，最初由Kolmogorov定义为长度为n的字符串的度量。

$$d(x|n) = n - C(x|n).$$

均匀分布对应于每个结果等可能的情况。 该定义使用 $C(x|n)$，即给定长度 $|\ x\ | = n$的字符串x的简单算法复杂度。 对于高度有序的字符串，例如重复的字符串，知道长度n可以得到更低的 $C(x|\ n)$值，如 $C(1n|^n) = O(1)$。 虽然定义可以使用自限编码的 $K(x|n)$来表示，但下面讨论的示例将使用简单编码，因为使用 $C(x|n)$作为复杂度度量更直观。

然而，正如后面所讨论的，这意味着像使用自限编码或来自前缀自由集的编码的随机性测试的结果将与使用普通编码的随机性测试略有不同。

回到随机性不足的概念，其中 $d(x|n)$很大，接近 $|x|$，字符串是高度非随机的。 这样的字符串是非典型的，因为它的算法复杂度远小于其长度。 因此，标签为‘随机性不足’，因为 $C(x|n)$的低值表明它可以被显著压缩。 例如，字符串‘11. . .1’，其中有一百个1，可以压缩到约 log_2100，即 $C(x) = log_2100 + O(1)$（方程（3.1））。由于 $C(x|100) = O(1)$，字符串的随机性不足为$100 - O(1)$；即接近100。显然，111. . .1是长度为100的字符串集合中的非典型字符串。 这种方法解决了直观问题，即虽然抛硬币并连续得到一百个正面与任何其他结果一样可能，但却完全出乎意料。

另一种观点是将一组结果分为两组，其中典型集包含那些可以被压缩很少的字符串，而非典型集包含那些可以被显著压缩的元素。

大多数结果将落入包含更随机结果的典型集中。 在抛硬币中，一个可以被显著压缩的或非典型的字符串只会偶尔出现（除非硬币有偏差）。 正如已经提到的，这类似于统计热力学中使用的论证，即平衡（即随机）配置是最有可能的。 虽然原则上可以观察到来自平衡集的有序配置，但这只会作为一种罕见的波动发生。 可以看出，随机性缺乏是一个配置与随机之间的距离的度量。 类似地，Chaitin的[28]组织程度是一个类似的度量，但它使用的是 $H(x)$，而不是条件概率在结果集上。 Chaitin的度量是平衡配置的算法熵与感兴趣配置的算法熵之间的差异。 换句话说，它是感兴趣配置与平衡之间的距离的度量。

现在需要注意的另一个要点，但稍后将被看作是重要的，是看可以压缩给定数量字符串的比例

数量。 考虑由长度小于或等于的算法生成的长度为的字符串。 它可以至少压缩到位。 在这种情况下，（|）≤−，并且随机性缺乏，（|）≥。分数2−+1，具有（|）≥的字符串很少。例如，对于=100和=10，只有51252个字符串中的1个可以具有（|）≥10。可以看出可以用作随机性的度量。

之前的讨论涉及对给定长度的字符串定义随机性。 这意味着均匀分布。 然而，随机性不足的度量可以扩展到字符串集合中的一个字符串 x。 这里使用的符号表示方式与常用于集合 S中元素数量的符号表示方式不同。 通常用 $|S|$表示，但在本书中，竖线表示算法规范的长度。 然而，在本书中，竖线被认为是算法规范的长度。

因此为了避免混淆，Ω_S将用于表示集合的成员数量。对于大集合，大多数字符串将是典型的，并且将用长度接近于 $log_2\Omega_S$的代码表示。 典型字符串的算法熵接近于集合的香农熵。

字符串 x在集合 S中的随机性缺陷由 $d(x|S) = log_2\Omega_S - C(x|S)$给出。 如果字符串是有序的并且可以被压缩，它的随机性缺陷将很高，因为它是集合中的非典型成员。考虑长度为 n且包含 r个1和$(n-r)$个0的所有字符串的集合 S。 一个例子是长度为100的序列集合，其中包含80个1和20个0。 在这种情况下，集合的成员数为$\Omega_S = 100!/(80!20!)$，典型字符串可以压缩为$log_2\Omega_S = 69$。 因此，80:20集合中的任何成员都可以以一种方式编码，将表示压缩到不超过69个字符，而真正随机的字符串则需要100个字符。 因此，虽然80:20集合中的字符串相对于所有字符串是有序的，但集合中的典型字符串的$d(x|S) \approx 69\ 1$。 然而，这些是集合中的非典型字符串，可以进一步压缩。 相对于80:20集合中的典型字符串，它们可以被视为“随机性不足”，其中典型字符串被压缩为69。 一个非典型成员是由80个1后跟20个0组成的集合，即$111\ldots11000\ldots00$。 生成这个字符串的算法形式为‘$PRINT\ 1, (100-20) times\ and\ PRINT\ 0, 20\ times$。 由于 $n = 100$是集合定义的一部分，只需要值为20来生成这个字符串，因为$80 = 100 - 20$可以计算出来。 因此 $C(x|S) \approx log_2 20 + O(1)$。 这里的 $O(1)$包括0和1的规范以及计算100-20。 相对于典型的80:20字符串，有序字符串的随机性缺陷为 $d(x|S) \approx 69 - 4 - 0(1) = 65 - O(1)$。 可以看出，对于长度为 n的所有字符串的集合S， $d(x|\ S)$的范围从长度为69的随机字符串的约0到 $log_2\Omega_S - O(1)$的有序字符串的范围。

扩展是指相对于随机性的不足之处

对于大字符串，可以使用斯特林逼近来显示$\log_2\Omega_s = nh$，其中nh是所谓的熵率。 熵率是由0和1序列生成的字符串的每个符号的熵，例如，生成0的概率是p，生成1的概率是1-p。 在这种情况下，概率p为0.2，$h = -p\log_2 p + (1-p)\log_2(1-p) = 0.7_2$，而不是69。 对于斯特林逼近有效，字符串需要更长

一个可递归可枚举的概率分布P。 在这种情况下，集合中的典型成员将通过其在集合中的概率进行编码。根据香农的无噪声编码定理，这将产生一个长度为-$\log_2 P$的编码。 这是这样一个字符串的最佳编码长度。

在这种情况下，典型的成员是符合概率分布的成员。
大多数字符串可以压缩成长度为 $-log_2 P$的代码。 然而，一个有序的字符串或者相对于概率来说非典型的字符串，可以通过超过 $log_2 P$的压缩，而随机性的不足变为$d(x|P) = -log_2 P - C(x|P)$。 对于所有长度为 n的字符串的均匀分布的简单情况来说，每个2^n字符串都是等可能的， P= 2^{-n}。

显然，随机性的不足衡量了一个有序字符串相对于随机字符串的可压缩程度。 正如下面的概述所示，随机性的不足为稳健的随机性测试提供了基础。

8.2 马丁-洛夫随机性测试

与科尔莫戈洛夫合作了一段时间的马丁-洛夫开发了一种称为马丁-洛夫测试的随机性测试[80]。 然而，在详细讨论该测试之前，将先探讨这个想法。 一个长度为100的字符串，由50个1和50个0的混合组成，除非存在明显的模式，例如前50个1形成的初始序列，否则可能被认为是随机的。 但是，如果是49个1和51个0的混合，或者40个1和60个0的混合呢？ 换句话说，与50:50比例相比，多大程度的变化将对应于非随机字符串。

马丁-洛夫认识到，虽然集合中的每个字符串可能是等可能的，但可以基于字符串的某些可量化特征将集合划分为区域。 如果可以做到这一点，并且其中一个区域的字符串较少，那么落入较少字符串区域的字符串的概率将低于落入较大区域的典型字符串的概率。 随机性始终相对于一个隐含或明确的字符串集合。 例如，在长度为100的所有字符串集合中，具有100个1的序列的字符串在直观上似乎比任何具有大致均匀的1和0混合的字符串更不随机。 关键是定义将具有100个1的字符串放入具有少量字符串的区域的特征，因此任何字符串落入该区域的概率很低。 这样的区域将包含被认为是非典型的字符串。虽然观察到所有字符串的概率是相等的，但是落入具有适当特征或特性的区域的字符串的观察概率与落入典型区域的字符串的观察概率可能大不相同。 尽管没有绝对的分界线来区分典型和非典型，但典型和非典型区域之间的边界可以根据一个正整数 m定义的所选显著性水平来确定。

将字符串分成区域的可量化特征是测试
函数$T\,est(x)$。 这个测试函数的设计是为了使字符串的概率

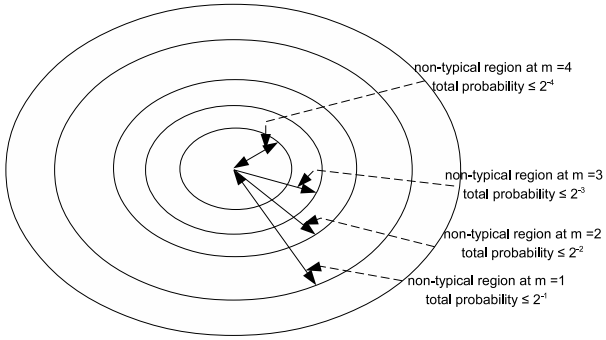

图8.1：$Test(x) \geq m$将字符串 x定义为非典型或非随机的区域 m，其中总概率 $\leq 2^{-m}$

字符串落入非典型区域的概率为 $\epsilon = 2^{-m}$，而大多数典型区域的概率为$1 - 2^{-m}$。 典型/非典型区域的连续性可以通过每次增加 m时减半 ϵ来描述。 随着m的增加，典型区域扩大，被认为是非典型或有序的字符串越来越少，而典型或随机的字符串越来越多。 在 $m = 0$时，即$\epsilon = 1$，没有字符串是典型的。在 $m = 1$时，一半的字符串将落入典型的区域，因为 m的值意味着截断值为 $\epsilon = 2^{-1}$。在 $m = 2$时，75%的字符串将落入典型区域，并在这个级别被认为是随机的，而25%的字符串将被认为是非典型的，因此更有序。 如图3所示，级别 $m + 1$的大多数典型区域包括级别 m、 $m - 1$等的典型区域。随着 m的增加，越来越多的字符串将被认为是随机的，因为典型区域增长。 换句话说，随着 m的增加，将字符串归类为非典型的要求更加严格。

Martin-L¨of随机性测试有效地使用测试函数来询问字符串effectively是否在显著性水平 m外的大多数或典型区域之外，因此它可以被拒绝为随机的吗？ 这样的测试必须是可枚举的和下半可计算的，以便能够实施。 该测试并不显示一个字符串是否随机，而是确定一个字符串可以被拒绝为随机的依据。如果有2^n个字符串，测试的关键约束是它不能将超过$2^{n-} m$个字符串分配给由特定m定义的非典型区域。即如果 $m = 2$，则不能有超过25%的字符串落在非典型区域中，否则测试将不一致，因为可用的二进制字符串不足。 因此，测试有两个方面。 第一个是确保在测试下，属于拒绝或非典型区域的字符串的比例不超过$2^{-} m$，第二个是有一个测试过程来确定一个特定的字符串是否属于拒绝区域。 由$Test(\ x)$定义的测试过程确定$Test(x) \geq^m$的字符串x的位置。如果该比例小于$2^{-} m$，则 x位于显著性水平^{m}的典型/非典型边界的非典型一侧，因此

x可以被拒绝为随机的。当给出由递归（对于所有x可计算）概率测度$P(x)$表征的相关字符串集时，其数学表达式为[76]

$$\{\Sigma P(x) : Test(x) \geq m\} \leq 2\text{^}(\text{-}m).$$

简言之，上述测试表明，当测试的值$\geq m$时，可以确定所有满足条件的字符串。如果所有这些字符串的概率之和$\leq 2\text{^}(\text{-}m)$，则可以拒绝该字符串为随机并认为其是有序的。换句话说，如果区域中所有字符串的累积概率$\leq 2\text{^}(\text{-}m)$，则可以认为该字符串是有序的。如果满足这个要求，对于给定的m值，可以一致地将字符串分配到拒绝或非典型区域。如果该区域中的字符串超过$2\text{^}(\text{-}m)$个，那么该测试将不具有足够的区分度，因为不可能有那么多非随机字符串。

这个任务是一个马丁-洛夫P-随机性测试。如果集合中有2个^{n}个字符串，可以基于满足条件的字符串数量（用$\#(x)$表示）来给出等价定义，即$Test(x) \geq m$。即在这种情况下，定义变为：

$$\{\#(x)\text{其中 } Test(x) \geq m\} \leq 2^{n-m}\text{。}$$

马丁-洛夫测试示例

马丁-洛夫过程可以测试通过抛掷可能有偏差的硬币产生的序列是否随机；更准确地说，观察到的字符串在哪个水平上被认为是随机的。设 $f(x)$是表示可能有偏差的硬币抛掷结果的字符串中1的数量。

如果结果是随机的，人们可能期望 $f(x)$约为 $n/2$，但是离 $n/2$有多远才算是一个合理的随机字符串？该过程是设计一个涉及 $f(x)$的上述形式的测试。例如，Cover、G´acs和Gray [38]使用切比雪夫不等式来开发一个测试函数$Test(x) = log_2\{2n^{-1/2}|f(x) - n/2|\}$。由于满足$Test(x) \geq m$的字符串数量为$2^{n-2}\, m$，因此满足测试要求，因为$2^{n-2m} < 2^{n-m}$。

另一种测试方法是使用前面提到的熵率，在第8.1节中。再次让长度为n的字符串中1的数量为$f(x)$，定义$p = f(x)/n$，使得$(1\text{-}p) = [(n\text{-}f(x))/n]$。对于较大的$n$，香农熵每个符号被称为熵率；即$h(x) = \text{-}p\log_2(p)\text{-}(1\text{-}p)\log_2(1\text{-}p)$。

通过适当的编码，对于较大的n，长度为n的字符串可以至少压缩到$nh(x)$，即原则上可以通过一个整数m来缩短字符串，其中$(1\text{-}h(x)) >= m$。例如，一个有75%的1的字符串可以压缩到$0.811n$。长度为100的这样一个字符串可以至少压缩18个字符。让Martin-L¨of测试为$T\text{est}(x) = n(1\text{-}h(x))$。$T\text{est}(x)$有效地衡量了字符串$x$基于此可以被压缩的程度，而$T\text{est}(x)$是一个有效的测试，因为所有$T\text{est}(x) >= m$的字符串的累积概率$<= 2\text{^}m$。

熵率测试可以通过考虑一个长度为170的字符串来进行演示。（选择这个长度是因为它足够长，使得熵率是

有意义，但不要太长，以至于无法在Excel电子表格上计算出需要计算的170!（阶乘）的累积概率。考虑75:95的1和0的比例。在什么水平的 ϵ下，这个字符串会被拒绝为随机的？在这种情况下，$T\text{est}(x) = 1.701 > 1$；它只能压缩到长度为169。在$m > 1$的区域中，所有字符串的累积概率为$0.2499$，小于$2^{-1}$。因此，75:95的字符串只会在 $m = 1$的截断水平即50%的水平上被拒绝为随机的。它仍然相当随机。

另一方面，一个70:100的字符串有$Test(x) = 3.8 > 3$. 取 $m = 3$，字符串在该区域的累积概率为 $< 2^{-3} = 0.038$. 这是$\epsilon < 0.125$的区域。 70:100的字符串可以被拒绝为随机的，或者在 $m = 3$或 $\epsilon = 0.125$的水平上被认为是有序的。 另一方面，70:100的字符串在 $m = 4$时不被认为是有序的。

随机性缺乏作为Martin-L ¨of测试

在第8节中，随机性缺乏 $d(x|P)$可以成为一个测试。即对于一个由可枚举概率分布 P定义的字符串集合，测试 [2]是

$$\delta_0(x|P) = d(x|P) - 1.$$

如果考虑均匀分布，其中 $P = 1/n$，随机性缺失测试将字符串长度为 n的字符串分配到一个区域，使用 $\delta_0(x|P) = n - C(x|n) - 1$.
由于具有简单科尔莫哥洛夫复杂度 $C(x|n) \leq n - m - 1$的字符串数量 $\leq 2^{n-m}$，具有 $\delta_0(x|P) \geq m$的字符串数量必须 $\leq 2^{n-m}$. 将此转化为概率术语，通过除以字符串数量得到结果：

$$\{\Sigma P(x) : \delta_0(x|P) \geq m\} \leq 2^{-m}.$$

这明显满足了测试的标准，但更重要的是，如下一节所示，这是一个普适的随机性测试。

马丁-洛夫普适P-随机性测试

可以证明存在一种普适的随机性测试，它优于所有其他随机性测试。 该论证在Li和Vit´anyi的第131页[76]中详细给出，表明所有有效的随机性测试都可以通过一种系统的过程列出。 如果 $\delta_y(x)$是第 y个测试，则相对于具有概率分布 P的集合，可以选择普适测试为

$$\delta_0(x|P) = max\{\delta_y(x) - y\}$$

因此

$$\delta_0(x|P) \geq \delta(x|P) - c.$$

普适测试与任何其他可想象的甚至不可想象的可枚举测试一样好（允许存在一个常数）。 然而，对“如何”有一些理解

[2]读者会注意到，由于 $C(x|n) < n - m$，需要‘-1’来确保关系是 $\leq$而不仅仅是 $<$。

在这个背景下，称之为“好的”是必要的。 如果普通的Martin-L¨of测试暗示 $\delta(x|P) \geq m$。 虽然字符串 x在 m水平上处于拒绝区域内，但是通用测试具有 $\delta_0(x|P) \geq m - c$。 因此，当 $\delta(x|P)$以 m= 4拒绝一个字符串时，测试 $\delta_0(x|P)$以随机方式拒绝该字符串，并将其视为有序的，即 $m - c$。 在拒绝随机性和分配顺序方面，它更加敏感。 这意味着，一般而言， $\delta(x|P)$需要比通用测试 $\delta_0(x|P)$更高的 m值来识别顺序。 对于给定的 m，通用测试将挑选出较少的随机字符串。 然而，并不存在一个唯一的通用 P测试。 相反，存在一个数字，使得每个通用测试可以在任何其他测试上进行加法优势。 在这种情况下，所有测试都返回相同的随机性度量值，考虑到常数。 虽然通用测试可能使用不同的 m值来对随机性水平进行分类，但它仍然以相同的顺序排列所有字符串的随机性水平。

论证中的关键是基于随机性不足的测试；即测试 $\delta_0(x|P) = d(x|P) - 1$，是一种通用测试。 更重要的是，它与第8.1节中概述的 m-随机性的定义一致。 即，Chaitin [27] 将随机性定义为一个字符串 x，其中 $H(x)$ 约等于 $|x| + H(|x|)$ 或 $H(x|n)$ 接近于 n。 具体来说，长度为 n的字符串是Chaitin m-随机的，如果它可以被压缩成长度为 $n - m$的字符串。即 $H(x|n) > n - m$。 这也适用于无限字符串 α，其中测试应用于无限字符串的序列，即存在m对于所有 n使得 $H(\alpha_n) > n - m$。 可以看出（Schnorr [89]）这个定义等价于Martin-L¨of随机性的定义在显著性水平 m上，考虑到Chaitin定义使用前缀自由复杂度而不是纯算法复杂度。

评估 $\delta_0(x|P)$ 并得出结论： x是随机的，使用通用测试等同于确定该字符串满足所有可计算的随机性测试，包括尚未设计的任何测试。 然而，不幸的是， $\delta_0(x|P)$ 是不可计算的。

总和 P-测试

G´acs [56, 57] 提出的另一种观察随机性的方法是使用公平赌注的回报水平作为识别非典型成员的过程（参见Li和Vit´anyi [76]中的概述）。高度非随机的结果是令人惊讶的，如果下注发生在这样的结果上，应该能够产生高额回报。例如，如果一枚硬币被抛掷100次，假设所有结果的可能性相等。然而，如果所谓的公平硬币偏向正面的概率为3:1，很可能正面出现的次数将是反面的三倍。如果每个结果都被认为是等可能的，这样一个令人惊讶的结果的发生应该能够通过一个足够狡猾的赌注获得回报。关键点是，令人惊讶的结果将比典型结果更具算法压缩性，因此可以被识别出来。为了说明这个想法，让 $H(x)$ 成为每个结果 x的算法熵。与一个（可能有偏差的）硬币抛掷100次的结果相关的回报是

如果结果是随机的，那么支付将很小，因为$H(x)$会接近100，这是一个典型字符串的算法描述长度。但是如果
如果实际结果的$H(x)$很小，那么惊喜的度量就是结果产生的支付的数量。 这个支付实际上以$2^{-H(x)}$作为通用概率分布 $m(x)$ (3.8)

如果所有结果等可能，每个结果的概率为 $p(x) = 2^{-100}$.
对于一个无偏结果 x的预期支付变为

$$\sum_{|x|=100} 2^{-100} 2^{100-H(x)} \leq 1.$$

由于基于抛硬币100次的赌注的预期回报不超过赌注本身，所以这个赌注是公平的。 这是因为Kraft不等式确保
$\sum_{|x|=100} 2$然而，如果硬币有偏差或者存在作弊，回报就会变得显著。例如，在完全偏向的情况下，结果是' $hhh\ldots$'，即100个正面，注意到$H(hhhh\ldots hh)$ $\sim log_2\, 100 = 6$。 对于1美元的赌注，2^{100-} $H(^x)$的回报接近于$\$2 \times 10^{28}$；这是一笔可观的金额。

对于不太出乎意料的情况，即硬币偏向3:1的情况下，字符串仍然可以被压缩。 典型字符串的算法熵接近于 $log_2\Omega_s$，其中Ω_s是展示这种比例的所有字符串的成员数（见第4节）。 在集合中有2×10^{23}个成员，给出 $H(x) = 77$。 在3:1的情况下，预期回报为$\$2^{100-77} = \$8.4$百万。 这个回报也是相当可观的。

虽然提供硬币的人可能不会接受一个依赖于结果的算法可压缩性的赌注，但这个想法可以形式化为定义一个随机性测试。 这个赌注测试被称为和 P测试[56, 57]。 从概念上讲，这种方法是通过一个明显随机的字符串是否通过一个明智的投注程序产生正回报来判断它是否真正随机。 如果可能定义一个公平的投注制度，使得某些结果产生正回报，那么结果字符串中必定存在隐藏的结构或模式。 另一方面，对于一个真正随机的结果，没有一个公平的投注制度会给出比赌注更高的预期回报。 不仅直观地理解赌注对随机性的理解是有意义的，在和P测试框架内它还具有数学上的鲁棒性。

为了推广这种方法，将 $\delta(x)$定义为一个求和 P-测试，其中 P同样是一个递归或可计算的概率。 在上面的示例中，$\delta(x) = 100 - H(x)$。 这是Chaitin的组织程度（第）和是随机性缺陷的变体。 一般来说，对于求和P-测试，$\delta(x)$必须满足期望回报小于一的支付要求（上述提到）。即。

$$\sum P(x)2^{\delta(x)} \leq 1。$$

可以证明，如果 $\delta(x)$是一个求和 P-测试，它也是一个Martin-L¨of P-测试（Li和Vit´anyi第257页[76]），但求和 P-测试稍微更强大。

方程意味着 $\delta(x)$ 使用自限编码，因此满足Kraft-Chaitin不等式 (3.2)。因此，一个普通的 P-检验，表示为 $d(x)$，与总和 P-检验之间的差异小于$2log_2 d(x)$。 这是由于简单编码和自限编码之间的差异。

类似地，对于任何递归概率分布 P，而不仅仅是示例中的简单分布，存在一个通用的总和 P-检验。 令 $\kappa_0(x|P) = log_2[m(x)/P(x)] = log_2[2^{-H(x)}/P(x)]$ 作为通用分布 $m(x)$ 等价于 $2^{-H(x)}$(第3.8节)。对于一次赔付为$2^{\kappa_0(x|P)}$的赌注的预期回报为$\sum P(x)2$ 如果这样的赌注回报率显著大于1，那么字符串x不能是随机的。 正如 $log\, 2m^{(x)}$对所有其他半测度具有支配作用一样，$\kappa_0(\ x|P)$对所有其他和P测试具有支配作用。 例如，长度为$_n$的字符串的均匀分布 $L\ n$的 $\kappa_0(x|L_n) = n^{-H}(x|n)$。 （注意，这与基于随机性不足的 P测试略有不同，因为（1）和（$_2$）使用自限编码提供熵度量 $H(x|n)$和随机性不足中的额外 -1没有包含在内。即，如果使用前缀自由UTM定义 $\kappa_0(x|L_n)$是不确定的，它与随机性不足测试 $\delta_0(x|Ln) =_n -C(x|\ n) -1$之间的差异不超过$_2 log_2\, C(x|n)$。这表明，与平衡的距离是自然界中特定配置的稳健Martin-L¨of阶测度或随机性缺乏的度量。这对于讨论自然界中的秩序是否表示设计具有重要意义。

G´acs'和 P-测试在直观上很有吸引力，因为它意味着如果一个人可以通过一个明智的投注过程在一组表面上随机的结果上赢钱，那么这些结果就不可能是随机的。 在这种情况下，有序程度与公平赌注的回报相关。

使用通用概念

AIT使用通用概念。 这些度量要么以加法方式支配其他度量，要么以乘法方式支配其他度量。 在加法支配的情况下，通用度量给出与任何其他通用度量相同的结果，但零点会被一个常数偏移。 例如，通用计算机可以模拟任何其他计算机，通用计算机上的算法长度在加法上支配所有其他计算机。 除了零点的设置外，所有通用计算机都是一样好的（尽管在实践中，如果使用具有几个核心指令的简单UTM，度量更加透明）。 再次，通用半度量 $m(x)$被定义为乘法方式支配所有其他半度量-它和任何其他选择一样好。 这相当于度量的对数以加法方式支配所有其他半度量。 由于 $-log_2 m(x)$以加法方式支配所有算法复杂度的度量，这允许将 $H(x)$替换为$-log_2 m(x)$或反之亦然。

同样对于随机性测试，有一个测试能够在所有其他随机性测试中加法地占主导地位 - 它和最好的一样好。 不同的通用随机性测试按照字符串的排序列出时，生成相同的顺序

根据它们的随机性水平。 然而，随机性标签可能在不同的通用测试之间通过一个常数进行偏移。 因此，简单的随机性缺陷提供了一个与所有其他通用随机性测试一致的随机性度量。 同样地，随机性缺陷的自限定等价物是 $-log_2[m(x)/P(x)]$，它等价于 $-log_2 P - H(x|P)$。这也可以被视为一个通用随机性测试。

这些结果表明，高概率结果具有低随机性缺陷，而低概率结果则是非典型的。

正如已经提到的，物理学家将会认识到随机性论证适用于统计热力学中的平衡态配置和来自平衡态配置的典型字符串应该通过所有随机性测试。 然而，在可能的平衡态集合中总是存在非典型的配置。 例如，玻尔兹曼气体中的所有原子可能会偶然地聚集在容器的一侧。 由于这种情况极不可能发生，因此它被认为是从平衡态中的有序波动。 虽然这样的波动可以被视为平衡态配置，但它对热力学熵的贡献可以忽略不计。 因为有序配置是非典型的，一旦这样的配置被识别出来，系统的各部分之间就存在熵梯度。 这样的熵梯度可以被利用来从系统中提取功。

第9章

复制过程如何保持系统远离平衡

9.1 介绍如何保持系统远离平衡

正如第1.2节指出的那样，以适当编码的最短算法的长度，在UTM上生成特定字符串，定义了该字符串的算法熵H，以比特为单位。一旦认识到驱动系统从一个状态到另一个状态的物理定律实际上是对真实世界通用图灵机进行计算（见第5.2节），算法熵可以为自然界提供洞察力。由于算法熵的差异与所使用的UTM无关，从在实验室中通过操作参考UTM中的位对真实世界计算进行映射的程序中得出的算法熵与从真实世界UTM中得出的等效位数相同。通过这两个系统的位和信息流是相同的。

算法熵是特定状态的熵，而传统熵返回与一组微观状态对应的宏观状态的值。尽管在第4节中概述的临时算法熵是平衡宏观状态中典型微观状态的算法熵度量，并返回相同的值，但允许使用与传统熵相同的单位。临时熵是指定特定配置的最短程序的长度。由于这种关系，可以轻松地在算法度量和传统熵描述之间进行转换。这使得临时熵可以用于跟踪系统随时间演变的熵变化和流动。

一个生命系统被认为远离局部平衡，在无情的热力学第二定律的作用下，熵的增加导致实时退化（参见第6.3节）。如果没有补偿-

对于熵增加的解释，系统将恢复到局部或全局平衡。 只是为了澄清符号，通常在本节中，平衡通常是指一个长时间稳定的局部平衡。

尽管如此，给予足够的时间，它将进一步降解为全局平衡。
例如，如果太阳光被阻挡无法到达森林，经过足够的时间，森林及其生态系统将主要转化为二氧化碳、水和矿物质。 从热力学角度来看，第二定律确保这样一个系统中的熵增加，因为有序结构中集中的能量变得更加分散并且无法用于工作。[1] 一个生命系统必须具有内建机制，以维持其处于远离平衡的配置，以抵消热力学第二定律所暗示的降解。 可以将其视为一种调节过程，其中维护的成本是将系统恢复到初始宏观状态的成本。 本章的主要论点是，复制过程生成算法上简单且熵低的结构，是生成和维持远离平衡的生命系统的主要机制。

Landauer原理[71]形式化了对现实世界计算中体现的计算过程的理解，并提供了一种工具来探究这些现实世界系统中秩序的产生和维持（参见[11, 12, 13]和[109]）。 他的原理指出，在温度为 T的计算机中，当擦除一个比特的信息时，至少需要耗散k_BTln2焦耳的能量。 在传统计算机中，这种耗散表现为热量传递到系统的非计算部分。 然而，当系统的状态的算法描述的长度在去除ΔH比特时发生变化时，相应的热力学熵变在现实世界系统中为$k_Bln2\Delta H$。 同样，必须将ΔH比特返回到系统中以恢复其初始状态。

生物和非生物自然系统的熵变受到相同的定律约束。 这使得我们可以将复制过程的熵和能量要求应用于非常简单的系统，并将最初的洞察力延伸到生物上更复杂的生物系统。 然而，为了进一步推进这个论证，下一节将探讨复制过程，而下一节将概述AIT的原则。稍后，第9.5节将涉及自然法则可逆性的一些概念问题。

临时算法熵不仅可以量化系统中所包含的秩序，捕捉其信息内容，还提供了控制局部秩序出现的热力学约束。 当这些自然计算过程在物理定律下运行时，排出无序，留下更有序或低熵结构。

算法方法通过信息位数来衡量自然系统与平衡之间的距离。

[1] 对于生命来说，第二定律过程将系统推向平衡的速率受到自然过程和障碍物（如化学活化能）所涉及的速率的限制。

必须向系统中添加物质以将其从有序状态转变为平衡状态。
Chaitin的组织程度 D_{org}实际上是衡量系统与平衡之间距离的一种方法。 如果平衡构型中有Ω个状态，则典型字符串的算法熵将为 $log_2\Omega$。 将有序字符串的算法熵表示为 s by $H(s)$， $D_{org} = log_2\Omega - H(s)$。 系统与平衡之间的距离只能通过排出无序和注入有序（或高级能量）来维持，从而创建有序结构。

从AIT方法提供的视角出发，以下见解可以得出对远离平衡状态的系统维护的认识

- 由于复制过程对许多自然系统至关重要，因此理解复制对于理解这些系统非常重要。 关键点在于，具有复制结构的系统的算法熵较低。只需要相对较少的比特来指定这样的系统，因为算法只需要一次指定结构，然后通过一个简短的例程复制或生成结构的重复。 这与需要独立指定每个结构的系统所需的比特数量形成对比。 由此可见，复制是一种自然的排序过程，以可量化的方式降低系统的算法熵。只要资源可用，并且高熵废物可以被排出，复制结构比其他自然过程产生的类似结构更有可能出现。

- 复制过程可以在系统的动力学状态空间的吸引子区域中创建和维持一个远离平衡的有序系统，其中所有状态具有相同的临时算法熵。 当通过热力学第二定律驱动的降解过程破坏秩序时，能够访问高质量能量并排出无序的复制过程能够将系统恢复到原始有序的配置集。 实质上，复制过程利用自然法则进行自我调节，以维持远离平衡的系统。

- 在一个复制结构系统中的变化导致算法熵的增加。 然而，变化也提供了一种自然机制来稳定系统，通过允许系统演化到其状态空间的更受限制的区域。 换句话说，变异可以通过适应性进化过程维持系统的稳定配置。 AIT提供了一个相当令人信服的论证，在许多情况下，那些在不断变化的环境中持续存在的变体是那些更有效利用资源并具有较低熵通量的变体。

- 耦合和嵌套的复制系统通过共同演化创造了更大的稳定性，从熵的角度来看，复制的结构更有效地利用资源。

这种效率，在这里称为“熵效率”，似乎是进化选择过程的一个重要约束。 然而，尽管每个复制的结构在这个意义上是高效的，但为了确保整体上高质量能量的降解比否则情况下更有效，相互依赖或嵌套的复制结构的数量增加了。 [88]。

许多生命系统也需要做工作，增加了维持系统远离全局平衡的热力学成本。 离平衡更远的系统具有更大的工作能力。 然而，随着系统离平衡越远，这些系统需要在资源使用上更加高效，即在计算方面更加高效。 此外，离平衡更远的系统需要在输出字符串中排除体现的无序性方面更加高效。 维持系统处于有序配置的成本在第6.2节中已经讨论过，而更一般的方法将在第9.6节中概述。

三个思维实验以阐明自然系统中的熵流

有一些概念问题需要解决，以避免对秩序的混淆，以及秩序来自何处。 以下三个简单的代表性思维实验有助于阐明计算问题和在热力学第二定律的影响下维持系统脱离平衡的困难。从算法的角度来看，这三个描述的过程类似于第6.3节中概述的第二定律的扩展。产生的无序性不包含在原始描述中，而是由引入系统以驱动系统达到最终停止状态的位引起的无序性。 第一个也是最简单的例子是从系统中移除热量，使系统更有序。 从某种意义上说，秩序是从外部环境中引入的。 第二个例子是保持光电二极管处于激发状态，而第三个例子是保持氢氧混合物远离平衡。 最后两个例子描述了从一个存储高级能量的配置开始，并随着时间的推移将能量在整个系统中重新分配的系统。

- 考虑一个容器中的气体，其中非平衡态配置需要气体的某个区域比气体的大部分区域更热。两个区域之间的温度差产生能量梯度，第二定律将驱使系统朝着均匀温度的方向发展。

  起始时的系统并没有比结束时更有序，因为没有任何东西离开系统-熵没有改变。 从某种意义上说，气体较热部分的动量自由度中的位设置是驱动系统达到平衡的程序，并通过系统传播能量。熵只有

当多余的热量被移除到环境中时，温度会下降。 如果这样的系统保持在非平衡状态，热量需要被移除，并且需要引入补偿热量到气体的高温区域。这个例子的重点是强调非平衡系统的维持要求是系统和环境之间的熵梯度必须保持。 热量需要被引入到系统的一部分，并从另一部分中提取出来。 在这个例子中，秩序不是通过注入热量引入的，而是通过移除热量引入的。 这相当于注入冷量，就像当来自环境的更有序的声子进入系统时发生的情况。

- 损耗问题的第二个例子是光电二极管，最初电子从价带激发到导带。 系统的宏观状态由包括激发电子态、振动态和系统的其他态在内的微观态集合来定义。 激发产生导电带和价带之间的电势差，原则上可以直接用于加热系统（通过电子返回价带的碰撞过程），或者间接地通过加热与光电二极管连接的电阻器来加热。 如果激发产生的电势差用于驱动电动机，光电二极管系统也可以做功。 无论是通过消耗热量进行机械工作，还是直接产生热量而不进行工作，如果系统自行演化，它将经历第二定律的退化过程，并在局部平衡中稳定下来，其中大多数电子返回价带，增加温度。 这种增加对应于系统的动量态的无序传递。

原则上，光电二极管可以在不平衡的光子流中保持，这些光子不断重新填充导带，前提是通过传热将过程产生的多余热量传递给环境。 然而，正如所讨论的，这将是一个偶然事件。

- 第三种代表性情况是氢气和氧气的化学计量混合物燃烧，形成液态水或蒸汽，取决于温度和压力。 该系统最初是有序的（见下面的讨论），但在点火后，系统从初始配置移动，并将稳定在化学平衡状态下。

$$O_2 + 2H_2 \rightleftharpoons 2H_2O + heat.$$

如果系统是孤立的，大部分氢气和氧气将被转化为 H_2O，并且局部平衡的温度将增加。 发生的是储存的能量被释放出来形成水和热量。 然而，这个过程是可逆的，并且本身并不产生有序性。 只有当热量损失时，剩余的部分才会产生有序性。

系统变得更有序。 一旦热量丧失，并且如果压力足够使液态水形成，系统可以通过散热和利用外部能量来恢复到原始配置，例如通过电解水来再生氢氧。

这些例子突出了一个重要的观点。 在上述项目中的第一点中，原始能量最初以热量的形式引入，以在系统和环境之间创建熵差。 虽然系统的部分之间的熵差可以用来做功，但只有在将无序作为废物或热量排出时，才会发生有序化（即熵降低）。

孤立系统的熵不会减少，即使发生化学反应，就像氢氧系统中所看到的那样。 当组分燃烧成为水+热量时，熵是相同的。 关键因素是通过系统中扩散能量创造了一种情况，使得多余的熵可以传递给环境。 有序化发生是因为这种无序可以被排出，通常作为废物物种或热量，留下一个更有序的结构。 从某种意义上说，可以将环境设想为向系统注入有序性。

维持一个非平衡系统相当于在系统和外部环境之间维持一个熵梯度。 正如已经提到的，维持非平衡系统的热力学成本由兰道尔原理[71]捕捉。 一个真实世界系统的轨迹可以被看作是在自然法则中体现的算法操作的结果。 随着替换丢失信息的热力学成本成为维持非平衡系统的成本，算法信息论提供了一种探究这些真实世界系统中秩序的产生和维持的工具。

9.2 通过生成秩序来抵消热力学第二定律的复制过程

一个一般的系统，如生命系统，必须通过抵消热力学第二定律所暗示的退化来维持自身在远离平衡的配置中。 一个系统可以通过使用储存的能量和排除无序来重新创建原始秩序。 正如已经提到的，兰道尔原理[71]表明维持非平衡系统的热力学成本需要替换丢失的信息位。

然而，大多数系统，如所描述的惰性系统，只能在特殊情况下维持在初始配置集中。 它们无法自我调节，因为这需要一个自主过程来替换丢失的信息。 光电二极管系统只能在一组稳定的配置中偶然地维持，例如，当系统存在于激发电子进入导带的光子的偶然流量中，并且系统可以排出足够多的多余热量以维持能量平衡。 类似地，氢氧系统需要一组不太可能的情况来维持非平衡状态。

一个惰性的地球（没有生命）将达到一个稳定状态，其中表面温度正好平衡来自太阳的辐射和地球的热辐射。 惰性地球作为一个系统在很大程度上类似于光电二极管系统。 当能量输入和输出平衡时出现的稳态情况并不是真正的调节。 与之形成对比的是一个活细胞。 活细胞可以通过获取营养物质和排出废物来积极维持自身。 地球上生命的自组织能力或细胞系统的自组织能力的很大部分似乎是结构复制的进化过程的结果。 下面发展的论证表明，地球的自组织能力是地球上生命结构复制能力的结果。 虽然惰性系统可以维持稳定的配置，但复制过程为真正的调节提供了一种自然机制。

通过生成有序的复制过程来对抗热力学第二定律的9.2个过程

正如已经讨论过的那样，算法信息论表明复制过程是生成我们周围观察到的大部分局部有序的关键。与之前讨论的其他结构不同，一个由复制结构组成的系统在生成该系统的算法更加压缩时更有序。 由于描述结构所需的信息位数较少，并且由于复制发生在自然法则下，复制过程既可以创建又可以维持秩序。 事实上，许多自然系统似乎依赖于复制过程作为自组织和自调节的手段。 给定一个复制体，其他复制体将形成，并且可以以一种补偿热力学第二定律的方式维持复制结构的系统。

通过考虑一个由形式为“重复复制结构 N次”的短算法生成的一组 N个重复复制结构，可以看出这一点。 这个算法的算法熵对应于

$$H_{algo}(\text{系统}) = |N| + |\text{复制体描述}| + |CI|,$$

其中CI指的是共同信息。 由于算法熵主要由表示 $|N|$的相对较短的字符串所主导。 该描述的长度接近于 $log_2 N$位。 这样一个短的算法意味着有序的结构。 与非复制结构的类似集合相比，这种复制过程必须单独指定每个 N结构，至少需要N位 [2]。

复制过程就像是一个连锁反应，使用结构或储存能量创建特定结构的重复。虽然复制的结构-

[2]请注意，这里的“复制体”一词将用于表示那些利用外部能量和资源复制自身的物理或生物结构。 这意味着生物细胞是复制单元，而不是作为复制过程的一部分被复制的DNA信息字符串。

当这些结构在系统中累积时，其他物理过程，如碰撞或化学反应，将破坏这些结构。 如果系统是孤立的，它将最终在一个长时间的平衡配置中安定下来。 由于复制而导致的有序增加只发生在捕捉计算过程历史的状态中出现的无序被排出，从而使复制结构能够主导系统。 当系统孤立时，总的熵变化是不可能的。

一般来说，复制涉及利用可用能量和资源进行物理或生物结构的再生，并同时将多余的熵作为热量和废物排放到环境中。 由于剩下的结构熵较低，因此是有序的。 此外，有人认为在这样一个系统中，复制的驱动力成为了一个自我调节的过程，使系统即使在第二定律的热力学下，复制的结构仍然会退化，保持在非平衡状态。 简单的物理复制过程可以用来说明通过复制实现自然排序的关键特征，尽管生物复制和系统退化涉及更复杂的路径。

这种复制系统的物理示例包括从熔融物质中形成的晶体，磁性材料中排列的自旋集合，以及通过受激发射产生的相干光子。 例如，分子更有可能在熔融物质中的种子晶体上凝固。

生物学的例子包括自催化集合体、在营养物质环境中生长的细菌以及生物结构（如植物）的生长。 在最后一个例子中，随着不同基因在植物的不同部位表达，基本结构的变化在这些点上出现。

复制体的特征是，在资源丰富的环境中，结构复制的概率随着现有结构的数量 N的增加而增加。 考虑一个经典物理系统，最初与宇宙的其他部分隔离开来，系统的瞬时微观状态由多维状态空间中的一个点表示。 随着时间的推移，系统的状态点将在状态空间中根据物理定律的作用而移动。 随着复制结构出现的概率与它们的出现次数增加，复制将使状态点驱动到由大量重复复制结构主导的状态空间区域。 在这种情况下，观察到复制结构的可能性比其他选择更高。

然而，为了使复制的结构主导系统，因为在孤立系统中比特是保持不变的，补偿性的无序必须转移到其他状态。 例如，当自旋在磁性材料中变得对齐时，无序被转移到振动态（即温度升高）。 振动态变得更加无序，捕捉了计算过程的历史或记忆。 除非相应的热量可以被移除，系统将保持可逆性，自旋将在一段时间后恢复到更随机的配置。

考虑一种情况，复制结构出现的概率与 N^x成正比（其中$x>0$），则 $\mathrm{d}N/\mathrm{d}t \propto N^x$ [97]。
为了实现这一点，假设熵作为废物或热量可以被排出，以使系统内不会积聚熵和废物。

以便熵和废物不会在系统内积聚。 例如，在适当的环境中给定一个DNA的双螺旋，出现第二个双螺旋的概率相对较高。

这里的重点是真实的复制过程，其中复制结构的存在触发更多的复制品。[3] 在资源有限的情况下，复制结构的数量随时间增长，直到达到稳态。达到稳态时，复制体系和环境之间形成了一种长期稳定的关系。

这个过程的一个示意图是逻辑生长方程

$$\mathrm{d}N/\mathrm{d}t = \hat{r}N(1 - N/K). \tag{9.1}$$

这个方程虽然非常简单，但捕捉到了复制的本质。 这里，$\hat{r}$表示通常的指数增长率， K 表示系统的承载能力。 在一个复制系统中，复制品的数量随时间增长，直到系统达到稳态；在稳态中，复制结构和环境之间形成了一种长期稳定的关系。 在稳态中，复制品可能会死亡和诞生。 $\mathrm{d}N/\mathrm{d}t = 0$)并且系统通过营养和能量的流动以及废物的排出来维持稳态。系统能够获取资源和排出废物的速率决定了稳态平衡的位置。对于一些简单的物理过程，废物可以通过热度自由度排出为潜热，就像系统的振动状态向环境散热一样。或者废物可能是相对无序的光子，或者是离开系统的降解化学产物。只要高熵废物能够从系统中排出，剩余结构的算法熵，因此热力学熵，就会减少。

许多生物系统也需要做工作，增加了维持系统远离全局平衡的热力学成本。 为了飞行，鸟类的工作速率比哺乳动物或爬行动物高。 因此，它们的体温更高。 离平衡更远的系统具有更大的工作能力，但它们需要在使用计算过程的输入资源方面更加高效。 它们还需要在从计算的角度看，排出混乱的效率更高的输出字符串。 正如后面所讨论的，作用于复制结构系统的进化类型过程选择更加高效的变化。

9.3 复制的简单说明性示例

与光电二极管系统或氢氧系统不同，复制过程是自然的排序过程。虽然有很多

[3] 真正的复制需要一个单位的存在引发连锁反应来创建更多的单位。 这与氧气和氢气燃烧等过程不同，其中燃烧的热量而不是水的存在触发了该过程。

在生物系统中发生的复制过程可以通过考虑简单的物理示例来学习很多。 两个具有说明性但简单的例子是在居里温度以下磁自旋（即磁矩）的排列和通过模拟发射复制光子。 在第二个例子中，受激发射通过将无序传递给原子的电子态，然后传递给热自由度，提供了有序的结构重复。 下面详细讨论这些例子，以阐明有序过程中的熵流。

复制旋转

对于不熟悉磁性系统的人来说，在接下来的内容中，“自旋”一词指的是旋转电子的磁矩。 人们可以将其想象为与磁性原子（如铁原子）相关的微小磁体。

如果所有这些微小磁体都排列一致，就可以说“自旋是排列一致的”等等。因此，在居里温度以上，自旋系统将变得更加无序，因为热能传递到自旋系统中。 如果无序系统是孤立的，它将达到一个平衡配置，其中自旋部分排列一致。 只有当温度降至居里温度以下，并排出潜热时，才会发生完全的自旋排列。 在这种情况下，自旋之间的铁磁耦合将主导热能。 通过丢弃热量的过程，系统从外部较冷的环境中提取有序，变得更加有序。 这些思想可以通过一个简单的模型来说明-自旋链的有序化。

这个模型不仅说明了复制，还说明了在排序过程中信息发生了什么 让耦合的一维链的构型由一个由N个无序自旋组成的字符串来指定，其中x可以是1（表示自旋向上）或0（表示自旋向下） 让振动态在居里温度以下 由于振动态处于低温状态，自旋子系统和振动子系统之间存在熵梯度 有序的振动态构型可以用一个有序的由N个零组成的字符串来表示

初始构型i由一组随机自旋和有序振动态组成

$$i = 1xxx...xx000000...$$

第一个字符是'1'，表示将复制特定的自旋，因为在适当的条件下，自旋方向将由于自旋之间的耦合而对齐 自旋对齐是一种自发的对称破缺过程 复制过程的计算等效于将所有自旋与第一个自旋对齐的计算，它会扫描字符串，并且由于自然法则的耦合，找到复制的结构'1'，并将下一个字符改为'1' 随着这个过程在随机对齐的字符串中继续进行，信息（即位）必须传递到字符串的$00...00$部分，扰乱振动自由度

热能必须从自旋态转移到更有序的振动态中，使其无序化。 这有效地提高了振动自由度的温度。 当系统孤立时，熵不会减少，振动态保留了物理过程的记忆，保持可逆性。 如果我们考虑系统与热汇接触，振动态可以通过将无序作为热量传递给外部热汇而恢复到原始的更有序的配置，留下更有序的系统。 在这个阶段，通过复制来对齐自旋的过程的输出字符串是

$$o = 11111111\ldots11xxxxx\ \ldots\ xxxx.$$

正如前面提到的，随机性已经转移到了之前有序的字符串部分。 这表明振动态的温度已经提高，但是没有排出废热，熵得到了保持。 只有当无序被排出时，由于振动态与低温外部热汇之间的耦合，系统才会变得更有序，而牺牲了外部宇宙的其他部分。

对于大的N，忽略计算开销，输入字符串的信息内容由算法熵$H(i)$给出。

$$H(i) \approx (N + log_2 N)\ (\text{指定随机}1xxxx..x) + log_2 N + |0|\ (\text{指定}0\text{，重复}N\text{次})。 \tag{9.2}$$

设U_1是将真实世界计算与当前自旋对齐的计算。 整体排序过程可以被看作是自旋链中这个对齐过程的重复。 在这种情况下，经过N步之后的输出为：

$$o = (U_1)^N i = {}_{11111\cdots11}xxxxxx... \tag{9.3}$$

)输出字符串可以通过类似的算法来指定，但这次自旋被排序，振动状态被打乱。即算法熵变为algorithm

$$\begin{aligned} H_{algo}(o) \quad &\approx log_2 N + |1|\ (\text{排列的自旋}\quad) \\ &+N + log_2 N + |0|\ (\text{无序的振动态}) \end{aligned} \tag{9.4}$$

.

在这个阶段，由于系统没有损失，可逆性隐含在最终状态中的计算历史中。因此 $H_{algo}(o) =$
$H_{algo}(i)$. 一旦振动态向外部环境散失热量，算法熵降低，输出变为；

$$o = 11111\ldots1100\ldots00.$$

将 i转移到 o的算法隐含在状态的位设置中，因为它们承载了物理定律中的算法

在真实世界系统的广义门中，即指令 U_1^N隐含在原始状态中。

相干光子作为复制体

另一个简单的例子是通过模拟发射来复制相干光子。 考虑一个物理系统，其中有 N个相同的双态原子，当激发态返回基态时会发射光子[47]。 发射的光子可以处于相干态或一组不相干态。 系统的状态空间最初包括；

- 原子的位置和动量状态；
- 光子的动量状态；
- 每个原子的激发态和基态电子态。

假设光子和粒子系统被包含在端镜和墙壁内，没有能量损失发生在与容器的碰撞中。
初始有序的配置将被认为是 N个原子存在于激发态，光子状态为空。 如果原子的激发态寿命很短，只会发射不相干的光子，并且系统将达到由激发态和基态原子状态组成的平衡。 这些动量向量散布在所有方向的不相干光子不会被显著有序。 此外，在第二定律过程下，一些激发态原子在发射光子之前通过碰撞或其他过程返回基态，并以此增加系统的热能。 最终，系统将在较高温度下达到局部平衡，大多数原子处于基态。

另一方面，如果激发态的寿命很长，就有可能产生受激光子。 这可以被看作是一个单个光子状态的复制品。 一旦一个偏离的光子复制，光子将继续复制，直到激发原子的数量、相干光子的数量和非相干光子状态的数量之间达到平衡。 一个对称性破缺过程发生，因为受激光子具有对齐的动量矢量，而不是随机分布在所有方向上。

在系统孤立的情况下，每个光子复制都是可逆的，一个相干的光子可以被吸收，使原子返回到激发态。 光子复制品在系统进入其状态空间的吸引子区域时死亡和诞生。 对于一个孤立系统的物理计算过程既是热力学上可逆的，也是逻辑上可逆的。

在一段时间后，这个孤立系统的瞬时配置的算法描述将包括原子状态的描述；每个非相干光子的状态描述；由原始刺激光子复制品组成的相干光子状态的描述；以及被认为是随机的原子动量状态。让系统的电子态表示为一个字符串，其中一个

‘1’代表一个激发态的原子，‘0’代表基态。 输入字符串，表示初始高度有序的激发态原子系统，没有光子，其示意形式如下：

[原子电子态] [光子态]
我=[1111111111111111111] [空]。

在讨论中，不考虑非相干光子态，因为这些态很少。原子的动量态和位置态也可以忽略。 在复制过程中，随着温度的升高，它们会变得更加无序，并且如果最终温度与初始温度相同，则会恢复到先前的描述。 最初，所有原子都处于激发态，没有光子。 如果可能发生受激发射，并且系统是隔离的，在一段时间后，将发射一个光子，并且这将刺激其他相干光子的发射。

同样，也可能发射非相干光子。 随着时间的推移，系统将由原子态和个相干光子的混合态组成。 总系统的瞬时配置的算法描述将包括原子态的描述，即它们是激发态还是基态，以及相干光子态的描述，这些光子态是原始刺激光子的副本。即

[原子电子态] [相干光子态]
o=[0011111011011010110] [n] .

相干光子态比不相干态更有序，因为动量矢量不再随机定向。 光子产生过程的记忆需要保持可逆性，并传递给原子的动量态。 实际上，当与环境发生热交换时，这种记忆会消失。 如果光子不复制，系统将趋向于平衡，因为储存的能量会传递给原子的动量态，类似于自由膨胀。 然而，光子复制会在将熵传递给环境后产生有序的相干光子态。

然而，如果光子能够逃逸，系统将需要访问外部能源来重新激发一些原子态，以弥补光子的损失。 激发原子的外部能源可以是电放电或闪光灯。

这些例子，虽然看起来很琐碎，但捕捉到了从AIT角度理解排序的一些关键概念问题。 如果物理定律体现在运行在UTM U上的程序 p中，这个过程可以被设想为 $U(p, i) = o$。 正如上面的例子所示，一般来说，复制过程通过将混乱传递给其他自由度来创建秩序；从效果上看，部分随机化其他自由度。 如果系统在长时间内孤立，系统将在状态空间的一个区域中稳定，其中复制的结构会死亡并在整个系统移动可能的配置中重新产生。

然而，系统在原则上仍然是可逆的。然而，当熵
转移到动量状态（热度自由度）时，

系统，增加其温度，系统动量状态和环境之间的温度差允许熵转移到外部汇中的热度自由度。因为计算过程现在是不可逆的，系统将定居在其状态空间的一个受限区域内。算法描述的长度减少，这个过程的熵成本为每比特传输到汇中的 $k_B ln 2$。

维护系统在这样的配置下的过程在第6.2节和9.6节中讨论。

9.4 具有变异的复制的熵成本

前面的章节考虑了复制结构完全相同的情况。然而，许多真实世界的系统由几乎相同但不完全相同的复制结构组成。 例如，眼睛中的细胞表达与肝脏中的细胞表达不完全相同，但它们都包含相同的DNA编码。 复制结构的这种变异具有一定的熵成本，可以通过算法信息论方法进行量化。 对于几乎相同的复制结构，临时算法熵确定了由变异引起的不确定性的熵增加。

因此，算法描述会更长。

一个简单的例子可以推广，说明了这个原理。 考虑允许将简单字符串1010 10...10的‘10’和‘also 11’都视为有效的复制结构变体。 这个字符串就变成了在第4.3节中讨论的带有噪声的周期-2字符串，其中每隔一个数字是随机的。 因此，有效的复制结构系统可以是任何具有形式o= $1y1y1y1y..1y$的长度为 N的输出字符串，其中 y为0或1。 所有可能变体的集合中有Ω_s= $2^{N/2}$个成员。 一般来说，临时算法熵（在第4.1节中给出）包括标识字符串在所有变体中的代码长度，以及标识集合本身的模式或模型的规范。即

$$H_{prov}(o) \simeq log_2 \Omega_s + |\text{描述复制单元模式的描述}|。 \tag{9.5}$$

由于有几种类似的算法可以用来指定临时熵，这里使用‘$\simeq$’表示可以忽略特定算法中的小的低效性。 用于识别集合中特定成员的代码需要 $|\Omega_s|$位。

在嘈杂的周期-2字符串中，Ω的规范大小，即 $|\Omega_s|$，接近于用于识别变化集合中特定成员所需的 $N/2$位。 为了定义允许的字符串集合，还需要额外的$\log_2 N/2$位来指定集合中字符串的长度，以便进行自限定编码，并且需要在第4.3节方程4.9中讨论的变体‘10’和‘11’的规范。临时熵变为；

$$H_{prov}(o) \simeq N/2 + log_2(N/2) + |1| + |0|, \tag{9.6}$$

忽略更高的“$loglog$”项。 相比之下，字符串 o的临时算法熵$= 101010...10$，表示复制单元中没有变异（即不是$1y$），其算法熵为

$$H_{prov}(o) = log_2(N/2) + |10|. \qquad (9.7)$$由于表示‘10和表示‘1和‘0的代码长度几乎相同，具有变异的字符串的临时算法熵比方程式9.7的简单字符串增加了 $N/2$。正如前面提到的，这是允许原始结构的变体所产生的不确定性增加（即香农熵）的增加。

一般情况

更一般地，让 r_1，r_2，r_3，... $\mathrm{r}_{\mathcal{M}}$是复制单元的变化
r。 如果以复制结构类型的数量为底数取对数
$\mathcal{M}$，则识别特定混合物的代码在此
基数中的长度为L，因为在基数为- $\mathcal{M}$的数制系统中，每个位置有$\mathcal{M}$个选择。相同规范在二进制表示法中的长度将为 L（$log_2\mathcal{M}$）。由于代码是自限定的，必须在代码中或者在集合的定义中体现 L和 M的长度信息。这些需求对算法熵的贡献需要被认识到并追踪。

量化临时熵的第一步是确定每个复制体的临时熵。 让第$_i$个复制体通过一个代码在 $\mathcal{M}$个复制结构集合中被识别。 由于代码需要是自限定的，用于识别特定复制体的代码长度将是集合中成员数量的对数，即 $log_2\mathcal{M}$，再加上代码长度的信息。即第$_i$个复制体可以通过长度为 $log_2\mathcal{M} + |log_2\mathcal{M}|$加上更高的“$loglog$”项的代码在集合中被识别。 算法$Algo$（$r_i$）读取第 i个复制体的代码，并根据定义复制体集合的标准输出 ri。 由于每个复制品具有相同的算法熵 H_{prov}（ri）对于任何i变为；

$$H_{prov}(r_i) = |Algo(r_i)| \simeq log_2\mathcal{M} + |log_2\mathcal{M}| + |$定义副本集的标准|。

一旦确定了每个非相同副本的规范，就可以确定非相同副本集的描述。 让 S_L是一个混合L非相同复制结构的组合 $S_L = r_i r_j r_j r_i . . .$。 下一步是输出 S_L，知道每个复制体的代码 $Algo(r_i)$ 在 S_L的代码字符串中。长度为 L的所有可能副本变体的集合有 $\mathcal{M}^L$个成员，因为每个副本位置有 $\mathcal{M}$个选择。 生成 S_L的算法，表示为 $Algo(S_L)$，需要包含以下内容。

- 为了自我界定， L的规范需要包括指定 L的代码长度。即 $||L||$。
- 算法需要从生成特定复制体变异的子程序中访问 $\mathcal{M}$的值，即变异的数量。

- 由于可能的 S^L变异集合中有$\Omega_s = \mathcal{M}_L$个成员，特定的变异由长度为 $L(log_2\mathcal{M})$的代码给出。
- 一旦使用先前提到的过程确定了复制体的特定混合，给定 $\mathcal{M}$，指定算法读取 $Algo(r_i)$在S_L中指定第一个复制体，实施算法并输出 r_i。
- 算法读取下一个复制体的下一条指令 $Algo(r_j)$，并输出字符串 r_j并将其连接到 r_i。 这个过程继续进行，直到处理完字符串中每个复制体的 L算法，并且在按顺序输出 L个复制体后，算法最终停止。

特定复制品混合物的临时算法熵变为；

$$H(S_L) \simeq L(log_2\mathcal{M}) + |L| + |log_2\mathcal{M}| + |\text{定义复制品集合的标准}| + O(1). \quad (9.8)$$

这里的 $O(1)$项包括次要计算的贡献，例如调用算法内部例程的指令。

另一种不太详细的论证认识到，与 S^L具有相同形式的字符串集合中有 M_L 个成员。 为了指定一个特定的字符串，需要定义集合本身以及香农熵贡献项 $L(log_2\mathcal{M})$，它是识别集合中特定成员的代码长度的度量。可以遍历集合中的所有字符串S_L，直到达到特定的代码来定义一个特定的字符串，如在章节中概述的那样。结果与之前一样

$$H(S_L) \sim L(log_2\mathcal{M}) + |L| + |log_2\mathcal{M}| + |\text{副本集的标准}| + O(1)。$$

对于所有副本都相同的特殊情况，进一步存在模式，
在这种情况下，当 $\mathcal{M}$= 1时， $L(log_2\mathcal{M})$和$\log_2\mathcal{M}$|项被消除。
然后，方程9.8变为，正如人们所预期的；

$$H_{prov}(\text{相同的复制字符串}) \simeq log_2 L + |r|。 \quad (9.9)$$

这相当于方程4.9中的周期-2字符串，其中 $|L| = log_2 L$ =$log_2 N/2$，而 $|r| = |10|$，指定了特定结构的特征。 如果 N 相对于模式描述很大，方程9.8允许变化，会导致熵的增加$L(log_2\mathcal{M}) + |log_2\mathcal{M}|$超过所有副本都相同的情况。然而，这是一个上限，因为它基于变化的排列是随机的假设。 在生物物种中，这很少发生，例如，所有与眼细胞对应的变异都聚集在一起。 因此，临时熵的实际增加可能要小得多。 即当内部触发器确定要表达哪些基因时，捕获此信息的信息已经嵌入在生长算法中。

另一方面，外部开关（如食物输入、光线等）可能决定基因表达，并增加简单复制算法的算法熵。 从本质上讲，由核心复制结构的变异构建的生物的临时熵可能与具有相同复制单元的结构没有显著差异。

9.5 可逆系统中的熵平衡

方程式3.14表明，一般情况下，计算过程的算法熵可能小于初始状态的算法熵和条件熵 $H(\ o|\ i^*)$的总和。 当从i到o时丢弃信息时，就会发生这种情况。 在这种情况下， o被称为相对于 i是多余的（参见Zurek [109]）。 然而，如果在可逆过程中所有信息都被考虑在内，则方程式3.14中应使用等号，因为熵不会发生变化。 重要的是要注意，定义算法熵的程序必须停止，因为程序必须精确指定配置。 例如，从DNA指令中生成树的算法必须停止，要么是因为观察者定义了停止要求，要么是因为输入了自然事件来终止生长算法。 在孤立系统中，当系统从表面有序的配置移动到停止状态时，熵不会增加。 这将导致一个悖论，因为系统应该从有序且因此熵较低的配置移动到较不有序的配置，然而可逆性意味着没有熵变化。

一旦意识到输出的熵的表面增加是从程序的初始位设置中可逆地引入到系统中的，悖论就得到了解决（见第6.3节）。初始状态 s_0和驱动系统的程序之间的分离是任意的。一旦将程序的位设置 p_h作为初始配置的一部分考虑进去，就可以看到 $H(p_h s_0) = H(s_h, \zeta)$，其中 $H(s_h, \zeta)$ 是包括计算历史 ζ在内的停机配置的熵，这是为了保持可逆性而需要的。 这一理解在Zurek [108]和Benne tt [11, 13]的讨论中是隐含的。

然而，由于这是一个重要的观点，下面的定理说明了为什么算法熵在可逆过程中真正是守恒的。

定理：在一个孤立的、可逆的、离散的系统中，进入系统的算法熵作为初始配置和带有停机指令的程序必须保持不变，出现在计算的最终状态的熵和计算历史中体现的熵中。

证明：考虑通过系统的比特空间（或状态空间）的最短可逆轨迹，形式为方程6.2。让停机配置为s后t步，并且需要维持可逆性的计算历史由字符串ζ表示。 因此，在选择最短的可逆路径生成s后t步的历史ζ之后，需要证明

$$H(p|s_0) = H(s\zeta) = H(s\rightarrow) + H(\text{在}t\text{步之后的轨迹})。 \tag{9.10}$$

案例1. 对于一个特定的 t_h在状态 s_hh没有更短的不可逆轨迹， s_h和 ζ由同一个程序一起生成，这意味着
根据方程式3.14，H $(\zeta|s_h^*) = 0$。没有更短的程序可以生成 $s_h\zeta$，

由于可逆性，到达配置 $s_h\zeta$ [4]的路径只有一条，在这种情况下，方程式9.10中的等号是有效的。

案例2. 如果存在一个比上述轨迹暗示的更短的 s_h规范，需要一个具体的论证来证明等号仍然有效。 在这种情况下，与案例1相反，$H(s_h) < H(s_h\zeta)$。 这对应于那些罕见情况之一，其中有序状态作为波动出现在轨迹中。 然而，虽然可能存在一个更短的程序生成 s_h，但没有更短的程序可以同时生成 s_h和 ζ，同样由于可逆性，到达组合字符串 $s_h\zeta$的路径只有一条。 由此可见，最短的程序必须是不可逆的，计算的历史记录必须与产生轨迹的步进程序生成的历史记录不同。 可逆算法和最短算法之间的熵差异为 $H(s_h\zeta) - H(s_h)$。 这意味着 $H(\zeta|s_h^*)$现在大于零。即必须丢弃信息才能定义 $H(s_h)$而不涉及计算历史。因此，对于s_h的更短描述意味着对于 ζ的具体且更长的描述。 如果考虑到这个更长描述中的信息位数，可以看到总体熵没有减少，如果系统要达到 s_h，则需要排出更多的位数以消除可逆性。

当步数增加且轨迹通过平衡点进入更有序的构型时，另一侧的平衡集合，$H(s_h\zeta)$ 会随着 $H(\zeta)$ 的增长而继续增长。 然而，在平衡点之后， s_h会变得更有序（即存在更短的描述），因为额外的位从 s_h的描述中转移到历史中，就像上面的情况2中所描述的那样。 事实上，在平衡点之后，由于系统是遍历的，最短的计算路径可能是从原始状态向时间倒退确定 s_h的路径。

当系统通过一组具有相同临时熵的状态 s_h时，发生了一种类似的情况。 同样， $H(s_h\zeta)$ 随着轨迹通过状态集合中的每个状态而增长，但临时熵除非 s_h指定了一个高度有序的波动，否则不会改变。 在这种情况下， $H(s_h\zeta)$ 可以分为 $H(s_h)$ 和

$H(\zeta)$ 作为指定 s_h不提供可以用来指定 ζ更简单的信息。 这一点用来描述下一节中一个稳态的熵需求。

还需要注意另一点。 当催化剂提供了一个可逆的替代轨迹时，由于需要包括催化剂，过程的历史将更长。 在这种情况下，适当的

$H(s_h\zeta)$ 将更大。 虽然使用催化剂的计算仍然是可逆的，并且对于方程式9.10来说等号是适当的。然而，生成 s_h和新的废弃字符串 ζ的算法将比较长。

[4] 在这里，忽略了算法在时间上倒退可能更短的可能性。 这只有在系统经过平衡时生成的状态才可能发生。

生成 s_h的原始算法，具有更短的历史或浪费字符串。
算法熵是从生成 s_h的最短路径（在这种情况下是可逆的）中推导出来的，而不是在时间意义上最高效的路径。

只要跟踪所有熵的输入和输出，系统就可以被视为孤立的，并且在保持熵不变的同时保持可逆性。 对于可逆情况下跟踪所有熵变化的等式9.10的等效形式变为；

$$H(s_h\zeta) = H(s_0) + H(s_h\zeta|s_0^*) + O(1). \tag{9.11}$$

例如，正如之前更详细地讨论的那样，氢和氧的化学计量混合物是有序的，因为能量集中在系统的初始状态中。 在适当的条件下，氢和氧将点燃，如果系统是孤立的，它将进入涉及氢、氧和水的平衡状态。 平衡温度高于初始温度，因为储存的能量已经通过系统扩散，使动量状态变得无序。 如果没有丢失或添加信息（即系统是可逆的），总系统的熵不会发生变化，并且原则上系统可以恢复到原始状态。 只有当与历史 ζ对应的信息丢失时（例如当系统排出过多热量时），可逆性才会丢失。 在这种情况下，系统的熵减少，因为动量状态的无序性减少。

9.6 自稳态和第二定律演化

第6.2节提供了外部驱动调节的要求，其中调节过程来自系统外部。 相比之下，本节讨论了自主调节对抗第二定律的要求。即调节过程从系统内部运行，而不是外部施加。 自主调节对于维持一个远离平衡的自然生命系统至关重要。 这里的论点是自主调节不是自然法则复杂相互作用的偶然结果，而是通过复制过程实现的，至少在简单情况下，也许在许多情况下都是如此。 在研究复制过程本身之前，首先概述了系统维持稳态或稳态的热力学信息要求。

自稳态的信息要求

为了看到自主过程如何在热力学第二定律的影响下将系统约束到稳定的远离平衡的配置集合中，我们首先定义系统的可行区域。 让系统的可行区域 E是一个包含一组远离平衡配置的宏观状态，用 ϵ_i表示。 这些微观状态中的每一个都指定了系统状态空间中的一个配置，其热力学熵相同（给定状态空间网格的特定分辨率）。

如果建立了一个约定的熵零点，允许单位存在，那么宏观态的热力学熵将返回与临时算法熵相同的值，临时算法熵是典型微观态 ϵ_i在宏观态5.2中的熵（即绝大多数微观态的算法熵）。 第4节，显示了微观态的临时熵由所有状态的香农熵和定义了集合中所有微观态共同结构的项组成。因此，临时熵的变化可以用来跟踪系统中的热力学熵流的进出。 在其可行区域内，系统通过状态空间的一个“吸引子般”区域移动，其中每个微观态 ϵ_i具有相同的临时熵 [5]。 这个吸引子般的区域在漂移过程中保持其形状，其中变量的标签在变化，但变量的类型不变。 如果系统最初处于远离平衡的宏观态的微观态 ϵ_i中，第二定律过程会将计算轨迹从可行宏观态E中偏离。 只有通过访问由字符串 σ表示的物种（如光子、原子、分子或化学物质）所体现的储存能量，并同时排出由字符串 ζ表示的无序或废物，系统才能在可行区域内维持。 或者，工作可以创建一个熵梯度，例如当热泵产生一个低熵汇来吸收多余的热量[6]。

在一个稳态集合中，系统从一个可行区域的状态转移到另一个状态，同时熵和替代物种进入系统，热量和退化废物离开。 在稳态稳定区域的特定状态的快照中，会显示出一些结构的混合；一些高度有序的，一些部分无序的，还有一些无法进一步降解的。 为了使稳态集合中一个典型状态的算法熵保持恒定，每一步增加的计算历史必须不断地被排出作为废物。

降解物种的第二定律过程和恢复计算重新创建秩序，最好看作是两个独立的过程。 将过程的第一阶段视为将系统从可行区域驱使到包括计算历史在内的非可行配置 η_j的干扰。 这种干扰会降解系统的结构。 从计算的角度来看，这种干扰可以用算法 p^*_{2ndlaw}表示，它是实现第二定律过程将系统转移到 η_j的最短算法。 在这种情况下，自然过程等同于对参考UTM进行以下计算。

$$U(p^*_{2ndlaw}, \epsilon_i) = \eta_j. \tag{9.12}$$

[5]我不认为这是一个真正的吸引子，因为变量的标签会改变。即随着时间的推移，原子 i会被另一个相同的原子 j替换。

[6]由于任何机器都需要一个熵梯度来进行工作，工作只是通过这个熵梯度将有序从一个系统转移到另一个系统的另一种方式。

在这个阶段没有任何东西离开系统，计算是可逆的，算法熵 η_j由以下公式给出：

$$H(\eta_j) = H_{prov}(\epsilon_i) + |p^*_{2ndlaw}| + O(1). \tag{9.13}$$

在这里，等号表示我们用 $H(\ \eta_j) -$表示第二定律过程的最短算法，该算法将 ϵ $_i$转换为η_j。
$H_{prov}(\epsilon_i)$ 位。

如果没有排除混乱的能力，并且在需要时没有接触到浓缩能量或低熵源，系统将会退化，并且随着时间的推移，在热力学第二定律下达到局部或全局平衡。 可以称之为调节或恢复过程，通过调节算法 p^*_{reg}来抵消第二定律的干扰效应。 这个过程对应于对输入字符串 σ_j进行计算，该字符串描述了包含存储能量的物种，以及字符串 η_j，该字符串指定了不可行的配置。 调节过程的净输出是具有与 ϵ_i相同的临时熵的最终微观状态 ϵ_k，并带有额外的废弃字符串 ζ $_l$。 通常，像 p_{reg}这样的计算会同时生成 ϵ_k和 ζ_l，因此结果的两个组成部分之间存在高水平的互信息。 根据方程9.12，这个调节过程由计算表示；

$$U(p^*_{reg}, \eta_j, \sigma_j) = \epsilon_k, \zeta_l. \tag{9.14}$$

从方程式9.14中,

$$H_{prov}(\epsilon_k, \zeta_l) = H(\sigma_j) + H(\eta_j) + |p^*_{reg}| + O(1). \tag{9.15}$$

表示链接独立停机子程序所需的位数的 $O(1)$项将被忽略。 方程式9.13和9.15可以相加，重新排列后，可以确定浪费位数 $H(waste)$。 这是需要从系统中排出的位数，以使其恢复到原始的宏观状态。 一个复制系统自我调节的基本要求变为,

$$H(waste) = H_{prov}(\epsilon_k, \zeta_l) - H_{prov}(\epsilon_i) = |p^*_{2ndlaw}| + |p^*_{reg}| + H(\sigma_j). \tag{9.16}$$

正如人们所预期的那样，废弃字符串的熵必须包括第二定律扰动算法、调节算法以及进入系统的提供集中能源的比特。 如果系统要返回到初始宏观状态，所有这些额外的熵必须被排出。 这可以与Zurek的方法相关联（在第6.2节中讨论），该方法认为，当 $H(\eta_j) - H(\epsilon_i)$ $(= |p^*_{2ndlaw}|)$ 比特将系统从稳定配置移开时，相同数量的比特需要返回到系统中，以使其返回到原始宏观状态。

然而，在自主调节的现状下，参与调节过程的比特是系统的一部分，需要加以考虑。

而在Zurek的情况下，它们被设想为系统外部的。
方程9.16表明，对于自主调节，调节过程中的位数净减少必须等于通过干扰进入的位数增加。也就是说。

$$H(\text{废物}) - |\mathsf{P}^*_{reg}| - H(\sigma_j) = |\mathsf{P}^*\text{第二定律}|。$$

与调节过程外部于系统的情况相比，自主调节要求将这些位数作为更广泛系统的一部分进行跟踪。

如果通过系统流动的熵和与高级能量 σ_j相关的位数没有被排出，位数将在系统中积累。
一个简单的例子是外部低温汇从系统中提取热量。 低温汇比系统更有序。

通过物理定律的作用，从环境中吸收热量或废物实际上是将秩序转移到系统中的转移。
方程9.16表明系统维持稳态的能力不直接依赖于 $H(\epsilon_k)$，因此系统与平衡的距离。 相反，正如在第9.6节中讨论的并且由方程9.19的要求所暗示的，第二阶降解过程可以被恢复过程抵消的速率取决于：

1. 能量浓缩源进入系统的速率，以及

2. 过剩熵从系统中排出的速率。

这两个要求确定了系统可用的远离平衡的构型的熵平衡。 系统退化的速率越大，系统距离平衡越远，以便排出足够的废热。 此外，系统距离平衡越远，其进行工作的能力越大，因为这取决于系统与其周围环境之间的熵差异。

由于自然法则中体现的第二定律将轨迹从稳态配置集中移出，通过增加熵$|p_{2ndlaw}|$，任何体现在自然法则中的调节程序，比如复制过程，必须将多余的熵转化为废物，这与第6.2节中描述的情况类似。正如方程式9.16所暗示的，位在调节过程中在系统内部移动。 这与方程式6.1形成对比，其中调节位不成为系统的一部分[5]。 最终结果仍然相同。

许多自然系统，比如上述的光电二极管，除非有偶然的情况或外部干预，否则无法自我调节。
如果氢氧水混合物中有氢气和氧气资源可供系统利用，那么可以维持，但需要输入热量来触发燃烧过程。 然而，一般来说，为了调节，系统必须能够感知或预测与稳态偏差。

配置和实施纠正过程[5]。 除了人类或动物的认知反应外，自然调节过程必须以某种方式编程到系统中。在这里，我们展示了复制过程的行为类似于一种原始形式的调节，不需要智能的外部干预。

真正的复制过程能够自然地调节，只要有足够的资源，创造或维持秩序的动力就是不可避免的。 因为复制将系统推向一个更有序的配置，当发生退化时，自然复制过程会寻求访问外部资源来创建或维持秩序。 这就是为什么激光中产生相干光子、自旋的对齐或细菌培养基中细胞的复制都能在远离平衡的状态下持续进行的原因。 这样的复制系统会调整复制单元的数量。 通过这样做，它们可以对抗第二定律的退化，只要能够访问资源输入并排出废物，如方程式9.16所示。这样的复制系统会在状态空间的吸引子区域中稳定，具有相同的临时熵。 随着资源在系统中流动，复制品被创建和销毁。 在可行吸引子区域之外，轨迹对其初始状态非常敏感，系统行为是混沌的。 复制系统调节的能力表明，复制过程是维持远离平衡的生命系统的基础，这应该不足为奇。

下一节将讨论复制过程的算法结构以及这些过程如何将系统恢复到稳态配置。

通过复制过程进行自然排序

与无法自我维持的惰性系统相比，方程式9.1表明，复制过程可以通过排除混乱并在必要时访问高质量的集中能量（如光子等）来自然地生成和再生具有低算法熵的有序结构。 高质量的能量重新包装系统，将热量和降解物种与有序区域分离开来，使自然过程能够排出废物。 一个简单的例子是细菌在营养物质的培养基中生长，最终在废物可以被排除并且营养物质可用于替代降解细菌的情况下达到稳定状态。 系统的承载能力取决于营养物质的补充速率以及细菌死亡产生需要排出的废物的速率。

为了理解反映这种复制过程的计算过程，可以修改方程6.2，使复制算法能够将许多微步骤嵌入到更大的复制子程序中。 这里的参数N将定义复制例程被调用以在字符串“$STATE$”之前附加进一步副本的次数，然后最终停止。 算法的示意形式是：

$$
\begin{aligned}
&ST\,AT\,E = r_0\\
&FOR\,I = 0\ 到\ N\\
&REP\,LICAT\,E\ r_I\\
&ST\,AT\,E = ST\,AT\,E, r_I\\
&N\,EXT\,I.
\end{aligned}
\tag{9.17}
$$

最终输出为$ST\,AT\,E = r_0, r_1, r_2, \ldots r_N$. 同样，如果没有任何东西逃离系统，复制情况是可逆的。 以下示例是帮助设想涉及复制的计算过程的简单模型。

- 磁自旋的排列或水结冰是复制过程的简单例子。 在磁性情况下（见第9.3节），在居里温度以上，磁性材料中的自旋是无序的，因为热能超过了自旋之间的磁耦合。 冰块的结晶类似。

  磁化自旋系统（或冰块）的特定配置可以通过指定结构的单位单元并使用复制算法将单位单元的副本转换到格点中的每个位置来定义整个系统。 由于振动态是热量传递到环境的手段，因此在描述中需要包括振动态瞬时配置的规范，以正确跟踪熵流。

  另一方面，可以使用复制算法来指定系统，该算法通过实施物理定律逐步生长有序系统。 在每个步骤中，能量从结构传递到振动态，然后传递到外部环境。 在这种情况下，给定初始振动态，当热量在复制过程中传递给环境时，铁磁系统（或类似的冰系统）的最终振动态出现。 在复制算法方法中，这些状态通过复制过程出现，无需单独指定。 如果我们有兴趣比较两个不同物理状态（例如水和冰）的算法熵，其中自然定律可以视为给定，那么只有在选择忽略严格属于系统的振动态和其他相关态时，复制算法才可能比复制算法更短。

  以冰晶为特例。如果冰中的每个水分子用 r表示，那么由 N个分子组成的冰结构可以用字符串 r_1，$\mathrm{r}_2 \ldots$ 表示 其中标签表示每个分子在结构中的位置。 然而，还需要指定与这些振动态耦合的状态。 因此，完整的规范变为r_1，$\mathrm{r}_2 \ldots r_N$，$\phi_i(T)$。 这里 $\phi_i(T)$表示瞬时规范

振动和电子状态以及温度 T下整个系统的其他自由度。

- 另一个模拟典型行为的例子是气体中的一组相同的双态原子，其中光子可以从激发的原子中发射出来（见第9.3节）。与结晶情况不同，受激发射需要高质量的能量嵌入到激发的原子状态中来供养复制过程。 当受激光子具有对齐的动量矢量而不是随机定向时，会发生对称性破缺过程。 受激发射可以看作是光子复制过程。 一旦失去热量，宏观态只能通过排热和获取能量来维持，以填充原子态。 宏观态包括相干光子态、极少数非相干光子和大多数原子处于基态。 在这种情况下，光子复制体会死亡和诞生，系统会稳定在一个吸引子中。

代表相干光子系统的微观态的字符串需要指定相干光子态、非相干光子态和与光子交换能量的原子能级，当系统从一个微观态转移到另一个微观态时，都在同一个宏观态中。 系统的其他状态，如动量状态，可以用 $\phi_i(T)$表示，值得注意的是，随着相干光子的补充，进入系统的能量传递给动量状态，然后传递给环境，保持恒定温度。

一个重要的问题是，一个生命系统，比如一棵树，最好由其DNA中体现的算法描述，还是由一个复制算法描述，将每个细胞及其变异复制到树的不同位置。 然而，在确定最短算法描述时，需要定义一致的熵为零。 这可以通过将自然法则和系统基本结构的规范作为给定的共同例程来实现。 在这种情况下，从种子中的遗传密码生成树的复制算法必须访问外部资源作为输入，当树按照方程9.17的线路发展时，细胞的变异取决于哪些基因被表达。 虽然DNA是相同的，但复制算法根据相邻细胞的环境和外部输入（如水和光的可用性）调用不同的子例程。 相邻细胞的环境实际上显示了过程的历史如何决定下一个状态。 因此，叶子中的细胞与根系中的细胞表达方式不同。

然而，一个复制每个细胞的例程，以及其变化，通常只会比描述轨迹的例程短，如果忽略了 $\phi_i(T)$中的状态，其中包括每个细胞组成部分的瞬时状态。

通过调用基于历史的例程来指定细胞的变化，几乎肯定比将每个变化指定为单独的结构更高效。 例如，如果有 $\mathcal{M}$个复制细胞的变化，

从 $\mathcal{M}^N$个变化中识别需要复制例程的特定配置，至少需要额外的 $N(log_2\mathcal{M})$位。 此外，每个复制的细胞都需要在空间中定位，并且还需要携带动能，原则上需要指定以精确定义状态。

上述论点表明，在描述真实物理情况的大多数情况下，复制算法在第一次出现时可能更长，但考虑到需要正确指定生物复杂（即非简单）结构，它很可能比复制算法更短。

然而，如果完整的复制结构数量要出现，熵作为热量或废物最终必须传递给环境。 原则上，一个生长中的树是一个可逆系统，但在生长过程中，废物离开系统。 随着废物的漂移，宇宙生命中的可逆性变得极不可能。 有了这些理解，就可以追踪算法熵流和热力学熵流在真实系统中的流动。

复制算法

简单的复制过程，如冰的结晶或自旋的排列通过排出过多的熵来形成，而大多数复制和排序过程需要输入高质量或浓缩能量以及排出废物。 在一般情况下，复制过程可以看作是对输入字符串 σ进行计算，该字符串表示作为集中能量和要复制的结构 r_1的资源字符串 $\sigma\ (=\sigma_1,\sigma_2...\sigma_N)$。 输入可以用$i=r_1\sigma$来表示，而输出是一系列复制的结构以及一个废物字符串。 如果复制循环循环 N次，将出现 N个复制的结构。

$$U(p_N,r_1,\sigma)=r_1r_2r_3...r_N,w(T'). \tag{9.18}$$

在这个过程中，复制的结构数量 r_1,r_2等增加，其中下标表示连续的结构。 字符串 $w(T')$ 是最终必须从系统中排出的历史或废弃字符串，以抑制可逆性，并允许有序结构出现。 对于复制系统的算法熵的贡献，其他状态在温度 T下原本基本保持不变，最终也基本保持不变。 最初，这些状态可以用 $\phi_i(T)$ 表示，变为$\phi_j(T')$ 其中 $T'>T$。 例如，当温度升高时，振动状态发生变化。 当废弃字符串 $w(T')$ 被排出时， $\phi_j(T')$分离为 $\phi_k(T)+w(T')$ 留下 $\phi_k(T)$。 因为表示为 $\phi_i(T)$ 的状态在之前和之后对临时熵的贡献是不变的，即 $|\phi_k(T)|=|\phi_j(T)|$，所以可以忽略这些状态。 总体而言，通过废弃物的注入，临时熵降低，结构更有序。

下面的论证跟踪了从计算视角和现象学视角对退化过程和恢复过程的处理。
我们假设系统最初处于稳定的可行配置，用ϵ_i表示，它指定了 N个重复结构以及与系统相关的其他自由度。 所有稳定配置的集合都具有相同的临时熵。 这意味着复制结构的数量是固定的，然而，热能可以以许多不同的方式分布在其他自由度上，例如系统的动量或振动状态。 可能的配置中，绝大多数都具有相同数量的复制结构。

第二定律的退化过程由方程9.12给出，即

$$U(p^*_{2ndlaw}, \epsilon_i) = \eta_j$$

。 这将使系统转移到不可行的配置 η_j。 从现象学的角度来看，复制的结构以 αN的速率被破坏，这取决于存在的复制结构的数量。 方程式9.14给出的调节计算访问资源字符串 σ以重新生成结构。 这种情况通常受到资源结构进入系统的速率的限制。 这通常被认为是每单位时间 β个结构。 从计算的角度来看，不可行的配置 $\eta_j = r_1$，r_2，...，r_{N-M}，$\mathrm{w}(T')$。 其中 M个复制结构已被破坏或衰变，创建了损耗字符串 $w(T')$，捕捉了过程的历史。 相应的恢复程序$p_{reg,M}$循环 M次以重新生成 M个结构并排出废物。

$$U(p_{reg,M}, \eta_j, \sigma_1, \sigma_2 \ldots \sigma_M) = r_1, r_2, \ldots r_N, w(T''),$$

总体上排出的废料包括导致复制品衰变的位；在调整算法 $p_{reg,M}$中的位，将复制结构的数量从 $N - M$恢复到 N；以及指定额外资源的位。即所有额外输入都需要排出以阻止系统中废料的积累。

复制算法的增长特性类似于离散版本的逻辑增长方程。 然而，逻辑增长方程无法充分跟踪复制过程中的资源流动。 一种更现实的方法是识别复制过程对资源供应的依赖关系，这类似于Holling II型功能反应，类似于Monod函数或Michaelis-Menten方程。 这个方程形式可以用来描述捕食者和其猎物之间的相互作用（在这种情况下是复制体-资源相互作用）。 如上所述，下面的方程包括一个项 αN，以捕捉由于生长方程中的第二定律效应而导致的复制结构的衰变或破坏。 假设是任何时候结构的去除（即它们的死亡）与它们的数量成比例。 此外，需要通过资源的恒定流动 β来推动复制过程。 虽然进一步的假设是所有计算废料材料都可以排出，正如后面提到的，但无法

高效地排出废物可以限制复制过程。 在这种理解下，让 N代表复制结构的数量，让 σ代表复制过程所需的资源单位数量。在这种情况下，

$$dN/dt = \frac{\hat{r}\sigma N}{(b+\sigma)} - \alpha N$$
$$d\sigma/dt = -\frac{\hat{r}\sigma N}{\gamma(b+\sigma)} + \beta. \tag{9.19}$$

参数 γ表示从消耗一个资源单位产生的复制结构数量，即 $-\gamma\Delta\sigma \approx \Delta N$。最大生长速率是初始生长速率 $\hat{r}$当$\gamma\sigma >> N$时，而 b是生长速率减半的资源水平，当 $\sigma = b$时。 稳定的长期稳态条件发生在两个导数接近零的情况下。 在这种情况下，复制结构的数量增加，趋向于一个长期值 $N_\infty = \gamma\beta/\alpha = K$其中 K可以被认为是系统的承载能力。

方程式9.12和9.18是方程式9.19捕捉到的复制过程的计算等效。 在每个单位时间内，β个资源流入系统，保持资源字符串的长度为σ。 在稳定的配置中，资源恰好足够替换在该时期内衰变的αN个复制结构。 从计算的角度来看，复制结构的长期描述变为$r_1\ r_2 r_3 \ldots r_K$，其中$K = N$。 由复制例程产生的复制结构字符串的算法描述的长度将接近于$|r_1 r_2 r_3 \ldots rK, | = |r_1^*| + log_{2K}$。即一个复制结构需要由$r_1^*$指定，并重复 K 次。 这里的星号表示这是复制品的最短描述。 另一方面，如果长度是从复制算法而不是复制算法派生的，主要贡献将来自于$|r_1^*| + log_2 N$。 最佳的复制结构数量将在$N = K$且 $log_2\ N = log_2\ K$时出现。

如果复制算法的输入不同，复制结构可能会有轻微的结构差异。 例如，眼细胞和心脏细胞由相同的算法生成，但由于眼细胞周围的特定环境提供了不同的输入，所以表达的基因不同。 在功能上的灵活性是通过调用不同的外部触发器来实现的，而算法熵的增加很小。

复制系统作为泵将熵排出以形成有序状态。
熵排出在降低熵梯度和降解高质量能量方面非常高效。 例如，当水通过复制过程结晶时，与缓慢冷却相比，会产生潜热的峰值，这是因为系统和环境之间的温差增加了热量传递速度。同样，当高能量物质

在复制过程中，优质能量转化为废物物种，与其他过程相比，被排出的废物更有效地转移到环境中。 这与Schneider和Kay [88]的观点一致，他们认为生物复杂结构的出现是从平衡状态转变的过程。

由于自然系统抵制任何这种运动，生物系统在耗散能量、降解高质量能量和对抗熵梯度方面比惰性结构更有效。 Schneider和Kay表明，生态系统中成分的蒸发和蒸腾是能量和熵耗散的主要形式。 在这里，人们会看到复制是系统层面上更有效耗散的驱动因素。

虽然现实世界的复制是一个复杂的并行计算过程，但上述方法捕捉到了其要点。 只要副本的创建速率超过其衰减速率，副本的数量将增长，直到系统稳定在一组稳定的配置中。 在这种稳态情况下，复制结构的承载能力受到废物排出系统的速率或资源输入速率可用于替换衰变副本的速率的限制。 许多生物系统还需要进行工作，增加了维持系统远离全局平衡的热力学成本。 虽然远离平衡的系统具有更大的工作能力，但它们需要更有效地利用作为计算过程输入的资源。 它们还需要更有效地排出从计算角度来看是输出字符串的混乱。 如下所讨论的，作用于一组复制结构的进化类型过程选择更有效的变异。

9.7 选择过程以维持系统的可行配置

选择过程是结构更适应环境的出现的关键。 本节从计算的角度讨论选择过程，表明在复制的结构中的多样性，同时更好地在逆境环境中保持系统远离平衡，也推动了不同结构类型的出现。 本节论证了，如果在几乎相同的复制品中存在足够多的变化，复制结构的系统不仅可以通过替换丢失的物种来对抗第二定律的退化过程，而且适应过程可以对影响系统的更严重干扰进行对抗。 为了实现这一点，复制的结构中的变化必须在外部环境变化的时间尺度上快速出现。 这种适应可以被看作是一种调节形式，因为一般来说，由简单复制品的变化组成的有组织系统对变化更加稳定。 它们具有在远离平衡的情况下维持自身的能力增强，因为可行配置集合更大。

然而，随着多样性的增加，它是一个可行的集合，正在扩大其视野。 变异主要是通过对遗传密码的改变而产生的

复制错误、基因转移和更一般的突变。 流入一个副本系统的新资源实际上是对输入字符串的扩展，扩大了其状态空间。 同样，系统中流出的资源会导致信息的丧失和状态空间的收缩。 考虑到复制体-资源动力学，其中一种干扰威胁到了复制结构系统的可行性。 这种干扰可能涉及物种或能量的流入或流出，将改变系统所采取的计算路径。 在不断变化的环境中，一些复制体的变异可能变得不太可能，而其他一些变得更有可能。 那些最适应的复制体将主导复制体混合物。例如，某个树种的变种可能更能应对干旱或药物耐药细菌的出现。 在这些情况下，“最适应”的短语通常意味着从熵流的角度来看，系统在使用资源方面更高效，也就是说，系统在远离平衡状态时更容易维持。

这种资源效率将被称为“熵效率”。 本质上，具有足够多样性的复制结构系统增加了系统在恶劣环境中生存的能力。 然而，需要认识到选择过程在短期内降低多样性，算法熵也会降低。 但是随着时间的推移，多样性必然重新出现，可以说这些适应性过程决定了系统演化的长期轨迹。

Eigen及其合作者在自复制多聚核苷酸的出现和增长方面的见解[49, 50]可以应用于更一般的复制系统。 在多聚核苷酸的情况下，复制错误提供了遗传密码的变异，其中一些通过选择过程主导了分子混合物。 有趣的是，Eigen表明，密切相关的多聚核苷酸分布，他称之为“准种群”，主导了混合物，而不是一个特定的变体。 如果在几乎相同的复制品之间存在足够的多样性，新的结构可以出现，因为适应过程可以对影响系统的更强烈的干扰进行抵消。

一个简单的模型，基于捕食者-猎物动力学（或在这种情况下等效的复制者-资源动力学），为我们提供了有关变异和选择过程如何使系统适应的有用见解，当两个复制结构的变体竞争有限资源时。 也许并不奇怪，就像对于单个复制体的情况一样，一个简单的模型，其中两个变体竞争有限资源，表明具有最高承载能力的变体将主导混合物。如下所示，承载能力取决于复制过程的效率，即资源的有效利用，以及变体抵御第二定律过程的能力。

然而，在这种情况下，承载能力不是外部参数，耦合的Lotka-Volterra方程版本是不合适的。 Eigen的方法[49]更加现实。 这确定了稳定平衡的要求，其中存在与滋养过程相对应的单体的恒定流量。 他的结果与以下简单讨论基本相同，该讨论基于第9.6节中讨论的简单捕食者-猎物模型。在这里的方法中，有两个变体

基本复制品与竞争资源σ的数字N_1和N_2 以下两个方程式分别捕捉到两个变体的增长率，给定它们各自的衰减率α和其他参数c_1和c_2，这些参数代表一个变体对另一个变体的影响。 最后一个方程式捕捉到资源变化的速率，给定变体对资源的消耗和新资源的恒定流入β。

$$
\begin{aligned}
dN_1/dt &= \frac{r\hat{}\,\sigma N}{(b_1+\sigma)} - \alpha_1 N_1 - c_1 N_1 N_2 \\
dN_2/dt &= \frac{r\hat{}\,\sigma N}{(b_2+\sigma)} - \alpha_2 N_2 - c_2 N_1 N_2 \\
d\sigma/dt &= -\frac{\hat{r}_1 \sigma N_1}{\gamma_1(b_1+\sigma)} - \frac{r_2 \sigma N_2}{\gamma_2(b_2+\sigma)} + \beta.
\end{aligned} \tag{9.20}
$$

尽管这些方程与捕食者-猎物关系的方程相似（见[4]），但最后一个方程有一个项 β 表示资源以恒定速率供给系统（见[49]）。 另一个假设是，该过程产生的废物能够被系统排出。

非周期稳定解的要求是所有导数都变为零。 对于几乎相同的复制变体， dN_1/dt和 dN_2/dt都是正的，并且一起趋近于零。 在这种情况下，要求与9.6中概述的单个复制变体的要求类似，系统再次在由资源被复制过程消耗的速率确定的区域内稳定下来。 然而，假设 $dN_1/dt > dN_2/dt$，因为 $\gamma_1 > \gamma_2$ 和/或$\alpha_2 > \alpha_1$，两个亚种最初都会增长，但最终 N_2停止增长，并且最终会随着项 $c_2 N_1$驱使 dN_2/dt趋近于零。 一旦发生这种情况，子系统 N_2的人口将变为零，而 N_1趋向于其最大值。 假设变体$_2$变为零，则 N_1的极限，即 $N_{1\infty}$是 $N_{1\infty} = \gamma_1\beta/\alpha_1$。 这是变体主导系统的承载能力 K。 主导变体是具有最高复制效率和由于第二定律效应而具有最低衰变速率的变体（即最高的γ/α）。 由于这是需要最低熵通量的变体，资源适应性对应于所谓的“熵效率”。 该结果可以推广到超过两个变体，如Eigen的方程II-47所示[49]。 上述方程基本上给出了Eigen对于稳定的长期配置的相同结果。 然而，Eigen的方法表明，一个紧密相关的准种群主导了混合物，而不仅仅是一个变体。 在Eigen的方法中， ϕ_M被设置为等于所有变体的平均增长率和死亡率。 这里， ϕ_M对应于 $\gamma\beta$。

可以争论这就像是一个选择最高效的复制变体将系统恢复到与原始状态略有不同的调节过程。这使得系统能够抵御威胁性干扰。 然而，这存在一个权衡。 增加复制结构的多样性会增加系统的熵，使其更接近平衡，而多样性则稳定了系统对环境变化的抵抗能力。对于一个

系统由 N个相同的结构组成，例如从种子生长的树，生成算法中的算法表示将为 $\approx log_2\ N$，而临时熵将类似于 $H(\text{树}) + log_2 N$。 然而，如果种子有 $\mathcal{M}$个变体代表遗传密码中的变体，则现在需要一个算法来为每个结构识别特定的变体。 由于存在 $\mathcal{M}\ N$种可能的分布变体，临时算法熵测量中的 $log_2 N$被$N\, log_2\ \mathcal{M}$取代，临时熵类似于 $H(\text{树}) + N\, log_2 \mathcal{M}$。 虽然多样性增加了临时算法熵，但如果变体很少，相对于整体生成算法，这种增加将很小。 如果变异是在基因水平而不是结构水平上编码的，正如已经提出的，在大多数生命系统的竞争环境中，选择过程将在基因水平上运作。 就像遗传算法模拟现实世界的选择过程一样，算法方法将这些现实世界的计算视为遗传算法。 虽然机会而不是物理定律可能是许多进化选择过程中的决定因素，但在半稳定环境中，熵效率仍然可能很重要。 问题是这种熵效率的选择过程是否在这里被广泛应用。

此外，选择熵效率的过程可能被视为选择行为的过程。 例如，具有一定运动的分离结构可能被选择以增加运动，使结构能够一起或分开移动以控制整体温度。 另一个例子可能是那些通过拍打翅膀来使蜂巢保持温暖的蜜蜂。 可以推测，这种行为驱动是通过对个体蜜蜂进行选择过程来将调节内置到遗传密码的软件中的。 结果是整个蜜蜂系统的熵效率提高了。 在生物层面上可能被视为适应性或“适者生存”的东西可以被解释为在算法层面上对熵效率的驱动。 这是一种利用自然法则进行复制过程来调节系统的驱动。 总的来说，核心复制结构的变化可以被环境选择以提供对外部因素的自主响应，因此可以被视为准调节过程。

然而，尽管系统选择了优化每个副本的熵效率的变体，但随着系统中结构的数量增加以抵消个体结构水平的任何增益，与环境的整体熵交换并不会减少。 正如后面讨论的那样，这与Schneider和Kay [88]的观点一致，他们表明随着生物复杂性的增加，整个系统在降解高质量能量方面变得更加高效。

当变异增加临时算法熵时，如果变异较少，相对于整体生成算法，这种增加将很小。 偏好某个复制品的选择过程会在短期内降低系统的临时熵。在长期内，

长期的进一步变化可能会增加未来选择过程对系统的临时熵。

这在调节方程9.16中意味着什么？ 输入字符串 σ包括维持复制品所需的营养物质，废物可能需要被排出到环境中的较低温度汇。 这些原则被捕捉在一组细菌中，这些细菌存在于一组可行的配置中，通过获取营养物质并将废热和物质排出到外部更冷的环境中。 显然，如果营养物质变得不可用，或者环境温度升高，或者抗生素进入细菌食物链，大多数细菌将死亡，而适应性更好的细菌将生存并主导种群。 从计算的角度来看，某些计算会终止，而其他计算则不会。

开放系统中副本的适应性和相互依赖

复制结构的开放系统可以相互连接和相互依赖，从而实现更高效的资源利用。 当一个系统的废弃字符串 ζ成为另一个系统的资源输入 σ时，这种相互连接可以出现。 维护组合系统所需的资源更少。 捕食者和猎物就是这样一种关系。 一个系统（猎物）中存储的能量被用作另一个系统的资源输入。 捕食者只是一种高效降解猎物中所蕴含的高质量能量的方法。

另外，当两个不同的物种增强彼此的生存能力时，互利共生也可能发生。

相互依赖的系统倾向于相互稳定，因此，远离平衡的耦合系统在维持系统离平衡状态的总熵成本较低时更具生存能力。 副本也可以共享资源，例如，当副本聚集时，总热损失减少。 当一个组分集合增强另一个组分集合的生存能力时，可以通过方程9.20来描述，其中 c_1为负数， c_2为零。 通过适当的参数值，耦合系统可以稳定在一个稳定的种群状态。 此外，当整体温度升高时，系统具有更多的工作能力。

一个简单的例子是光子从一个激光系统中出来，以创建另一个激光系统中的人口反转，结果是更少的信息丢失给环境。 在整体资源受限且第二个系统具有足够的多样性的情况下（例如，激光之间具有足够的线宽重叠），该系统可以通过在其状态空间的更受限区域中定居来适应。 与两个独立系统相比，由于组合系统的算法规范涉及信息共享，因此需要输入的位数更少，因此需要从耦合结构中弹出的位数更少。 由于耦合系统通过更有效地使用资源而共同演化，在资源受限的环境中，耦合系统对输入变化更稳定。 相互作用或复制系统之间的相互依赖增加了秩序，从而改变了有组织系统的平衡。

进一步远离平衡。 在资源约束下，如果这种相互依赖增加了系统的熵效率，相互依赖更有可能发生。它们的相互吸引区域不会以与类似但未耦合的系统相同的速率在状态空间中漂移。 然而，随着耦合结构的人口增加，输入资源的整体退化速度比其他情况下更快且更完全。 尽管如此，如果这样的结构是可能的，它们似乎很可能出现。

嵌套系统和组织程度

嵌套在其他结构中的结构表现出一种特定形式的相互依赖性，这增加了整体的有序性，因为嵌套的结构可以通过算法映射到一组嵌套的子程序中，这些子程序相互传递信息。 因此，排出的浪费或信息较少，嵌套的结构在算法上更有效。 许多生物系统由嵌套的子结构组成，其中子结构通常是基本单元的变体。 例如，细胞本身可以嵌套在像眼睛这样的器官中，而器官则嵌套在更高的结构中。 当遗传开关在优先表达某些基因时，子结构（如细胞）的变化就会发生。 这些开关的算法等效物对应于从更高级例程传递给生成子例程的信息。 这可能来自相邻的细胞，就像一组特定的细胞产生眼睛这样的器官的情况一样。或者信息可能来自外部环境的输入。 嵌套还允许功能的专门化，而算法熵几乎没有增加，特别是如果嵌套算法的输出仅取决于信息输入。 这允许变化的模式在结构通过类似于方程6.2的生成算法演化时出现。正如在第6.3节中提到的，描述生成结构的轨迹的算法（例如树的种子中体现的算法）很可能比描述结构最终形式的算法短，除非可以忽略重要信息。

Chaitin的[28]“d-直径复杂性”概念，量化了不同尺度上的顺序，有助于理解嵌套结构。 想象一下，我们只能在最小的尺度上观察一棵树，无法在更大的尺度上识别任何顺序。 当无法感知到大尺度的顺序时，树只能通过一个冗长的算法来描述，该算法指定了分子级别的结构以及分子结构如何组合形成树。

图9.1展示了一个嵌套的复制结构系统。 在这个尺度上，最小的尺度用 d_n 表示，其中 n很大， d_n很小。
在这个尺度上，由于没有可辨认的模式，系统只能通过一个算法来描述，该算法指定了看起来是随机排列的基本结构。 临时算法指定的排列将非常长，算法熵将无法与局部平衡宏态的熵区分开来。

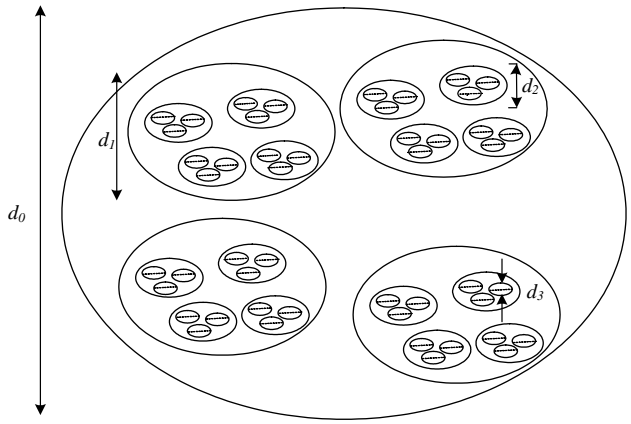

图9.1：嵌套结构在尺度 d_0， d_1， d_2和 d_3上。

然而，以更高的尺度观察系统可以发现一些结构，从而使得描述系统的算法变得更短。
在更高的尺度上，可能会识别出细胞。 在这种情况下，一旦细胞被定义，整体算法变得更短，降低了算法熵。 这是因为结构可以通过重复细胞构建块更简单地构建。

使用这种方法，Chaitin [28] 量化了结构 X 在尺度 n上与局部平衡的距离，局部平衡对应于最小尺度上没有可识别模式的描述。 组织程度是Kolmogorov的“随机性缺乏”概念的变体，使用自限定编码。 使用自限定编码的尺度 n上的组织程度D_{org}定义如下：

$$D_{org} = H_{max}(X) - H_{dn}(X)。$$

H_{max}对应于可能状态数量的对数，其中无法辨别出结构。 结构随着尺度增加而被识别出来，生成算法缩短到达最大尺度时达到最大值，即 D_{org}的最大值。

图9.2说明了算法熵随尺度参数 d变化的情况。 在图中， d_2， d_1和 d_0表示图9.1中的尺度维度，其中 $d_n << d_2 < d_1 < d_0$。 让 H_{d0}表示系统的算法熵的最小值，基于其在最大尺度 d_0 [47]的最短描述。 嵌套结构可以由相互连接的子例程描述。 然而，当大尺度结构被抑制时，系统被视为一组子结构，例如一组细胞或在较小尺度上，一组分子。 相应的算法描述现在必须在该尺度上指定每个子结构以及如何组装子结构。 随着大型模式的丢失，熵增加。 在最大尺度上， $H_{d0} << H_{d\infty}$，其中 d_∞是最小尺度。

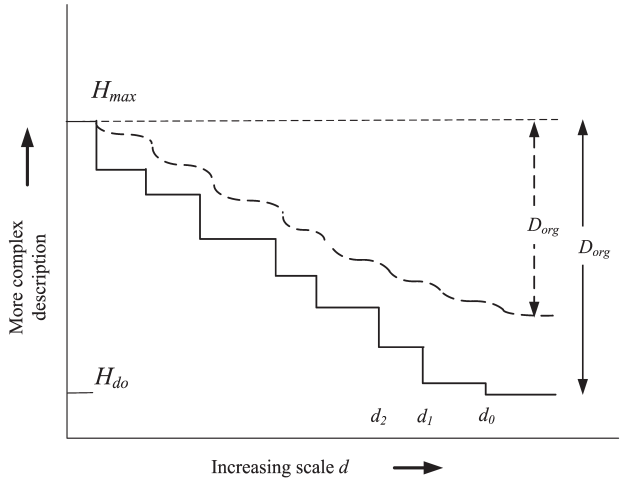

图9.2：随尺度变化的d-直径复杂性； —嵌套复制体-
没有变化的复制体；– – – – –有变化的嵌套复制体；- - - - - - - - - - - - - 没有任何尺度上都没有组织。

在图9.2中。阶梯状的粗线显示了一个理想系统，在每个尺度级别上的嵌套系统都是相同的复制结构。 熵保持大致相同，直到一个较低的尺度水平抑制结构。
随着尺度参数 d从 d_0减小到 d_∞熵以明确定义的步骤增加。 然而，当发生复制结构的变化时，例如在相同尺度上的不同细胞类型中发生的情况，步骤在虚线中被平滑。 多样性增加了每个尺度级别上的临时熵。 当完全没有组织存在时，尺度是无关紧要的，算法熵在所有尺度级别上都相同，如图9.1中的点状水平线所示， Hmax。

这意味着，如果一个算法（比如DNA中的算法）包含可以根据特定输入进行切换的子程序，那么多样性会随着算法熵的轻微增加而增加。 由于软件变异发生在较低层次的尺度上，似乎通过软件（例如DNA中的变异）生成变异比独立指定相似结构更加高效。 可以合理推测，结构的嵌套方式会降低熵，这种降低的熵超过了在复制的结构中构建多样性的熵成本[48]。 事实上，这可能是结构选择过程的必然结果。 具有高 D_{org}的相同复制结构可能比具有变异或更大多样性的系统更难适应变化。 允许变异的成本是较低的 D_{org}。观察上看，与较简单的结构相比，一个嵌套子系统的系统（例如森林生态系统嵌套物种，物种嵌套器官，器官嵌套细胞）在远离平衡状态时更具生存能力。

系统的一部分的浪费是另一部分的资源输入。 总的来说，每个嵌套单元需要排出的废物更少，因为存在相互依赖关系，并且排出无序（如热量）的效率更高。
由于选择过程偏爱熵效率高的复制结构，减少对外部资源的依赖意味着相互依赖的嵌套系统的承载能力高于不存在相互依赖的系统。 当资源有限时，更有可能出现相互依赖的系统。 这些因素表明，这样的系统会寻找熵效率高的配置来维持复制结构的生存。 这倾向于建立相互依赖关系，并因此使系统远离平衡。

也许地球上的生命在结构层面上比没有生命更熵效率高，因为生命由相互依赖和嵌套的复制系统组成，创造了自己的自主生存和适应能力。 然而，有一个限制条件，即必须能够获得来自太阳的光子等浓集能源，并且必须能够排出废物。 然而，生命以更高效地耗散能量为代价，使整个系统远离平衡。 虽然嵌套系统中的资源利用更高效，但随着不同复制结构的数量增加，消耗的资源也增加，以更高效地处理高质量能量。 这与Schneider和Kay [88]的观点一致，他们认为层次结构系统（如生态系统）在降解高质量能量方面比简单或惰性结构更高效。 在相互连接的层次结构中，一个系统排出的废物可以被层次结构中较低的系统降解。这种情况发生在昆虫、真菌或细菌比非生物过程更快地降解植物材料的情况下。 简而言之，单个复制结构追求熵效率，但在系统级别上，人口增长以增加总体耗散。 层次结构系统更好地利用资源，以更快地推动宇宙走向平衡的代价。

9.8 系统调节和AIT的总结

算法信息论通过在通用图灵机上描述系统所需的最小比特数来确定系统的算法熵或信息内容，而兰道尔原理[71]认为擦除一比特信息对应于熵转移 $k_B ln2$到环境中。 这些理解共同提供了一种一致的方式来看待维持系统在远离平衡的稳定配置集中的热力学成本[109]。 由于只有熵差才具有物理意义，将系统从一种配置转移到另一种配置所需的比特数乘以 $k_B ln2$，对应于系统的等效热力学熵变。 从热力学角度来看，扰动或第二定律的退化过程通过增加算法熵 H 比特或等效地增加系统的

通过热力学熵 $k_B Hln2$焦耳/开尔文度，恢复系统的热力学成本是通过移除$k_B Hln2$焦耳/开尔文度来补偿变化。

此外，有人认为，由于复制结构的系统可以用少量比特来描述，自然复制过程利用所有可用资源生成具有低算法熵的简单有序结构。

这个观点使人们能够认识到自然复制过程可以被看作是在能够在远离平衡的真实世界UTM上进行计算，从而产生有序。 复制过程不仅可以生成有序，这些过程还可以自我调节或自组织，因为它们可以自然地通过访问外部资源作为输入来重新生成退化的复制单元，同时排出熵作为废物，满足兰道尔原理的要求。

当环境发生变化，并且复制单元之间存在多样性时，选择过程就会起到自主的准调节作用。
持续存在的复制变体很可能是在资源使用和废物排放方面更熵效率的变体。 在这些情况下，环境适应性要求优化系统中结构的熵效率。 熵效率意味着维持系统的熵成本最小化，并且系统通过在尽可能接近可逆的远离平衡的配置中定居来适应。 然而，尽管选择过程允许系统返回接近原始配置，但随着新的变异出现，系统随着时间的推移漂移或演化到新的配置。

同样，当不同复制结构之间存在相互依赖性时，简单的计算表明选择过程将偏好展现相互依赖性的变体。 由于相互依赖性在结构层面上更熵效率，它们的数量增加。 复制结构的层次结构似乎很可能出现，因为选择过程偏好浪费更少资源的结构。 但是随着这些结构的数量增加，总体能量耗散增加，并且在层次结构系统持续存在的同时，加速了供养生命过程的高质量浓缩能量的降解。 正如Schneider和Kay所展示的[88]，生态系统由无数自我复制的高效单元组成，但是由于食物链中较低物种消耗较高物种的废物，整体系统的退化程度比非生命系统甚至更简单的生命系统更大。 生命系统加速了宇宙的死亡。

在系统需要工作的地方，它必须进一步远离平衡，并且再次复制过程可以将系统维持在适当的稳定配置中。

结果是普遍的，并且原则上适用于任何系统。 然而，在许多实际情况下，对系统的细节了解不足，无法量化维持系统远离平衡的成本。 尽管如此，这种方法可能对理解真实系统的渐进变化很有用，并提供一些系统行为的广泛描述。 虽然一个生命系统不仅仅是复制

或者熵流，正如能量约束决定系统行为一样，熵要求限制了结构可能性。

第10章

AIT和哲学问题

10.1 算法描述、学习和人工智能

算法信息论为学习和人工智能提供了一些实用的见解。 可以将学习视为一个过程，通过该过程，当一个代理给出与某种现象相关的数据时，可以对数据进行足够的理解，以部分预测未来的结果。 在观察结果没有模式或结构的情况下，学习是不可能的。 由于这些观察结果无法压缩，每个未来的结果都是一个惊喜。 Leibnitz（[36]认为，最好的理论比其他理论更有效地压缩数据。因此，理论越好，学习越好，它可以更好地预测未来的结果。由于学习和人工智能领域是专业学科，这里只能概述算法洞察力的风味。希望对此感兴趣的人可以深入研究该主题。

Solomonoff [95, 96] 的两次讲座回顾了压缩、学习和智能之间的联系。 学习可以在AIT框架内进行解释。 特别是，学习过程的结果与最小描述长度程序一样有效。 正如第7节中概述的那样，最小描述长度（MDL）方法认为，最符合观测数据的拟合是通过提供数据最压缩版本的物理模型来实现的。 指出该方法既满足奥卡姆剃刀原则所体现的简洁性，又满足伊壁鸠鲁原则的广泛性，因为该方法在不排除可能原因的情况下寻求最简单的解释。 然而，关于MDL还存在一些更深层次的问题。Vit´anyi和Li [104]指出，使用MDL进行数据压缩几乎总是最佳的模型选择和预测策略，前提是与设想的概率假设相比，与模型的偏差是随机的。

这意味着要有效，偏差中不能有可识别的顺序。 正如第7节中提到的，MDL的理想方法可以从贝叶斯方法中得出，使用算法通用分布

作为先验分布的第3.8节。 使用这个通用概率，MDL方法最小化了两个项的和。 第一个项通过通用分布表征模型，这实际上是模型的算法描述，第二个项通过一个长度等于该偏差概率的对数的编码来编码模型的偏差；即满足Shannon的无噪声编码定理3.3的编码。Barron和Cover[6]将该方法应用于选择最合适的概率分布来解释从未知概率分布中抽取的一组随机变量。 他们表明，两部分MDL方法最终将从候选分布类中找到最优的数据分布。 然而，该方法是基于寻找潜在模型的有效策略，而不是对真实状态的简单性的任何信念。 与原始的贝叶斯方法相比，MDL对潜在分布或任何主观评估的概率不做任何假设。 正如Gr¨unwald指出的[60]，MDL在没有关于真实事物状态的形而上学假设的情况下是渐近一致的。

此外，实际的MDL方法是一种有效的归纳推理方法，只有少数例外。

Vereshchagin和Vit´anyi [103]表明，结构函数确定了数据的所有随机属性：对于每个受限模型类，它确定了类中最佳拟合模型，无论“真实”模型是否在模型类中。 在这种情况下，这是确定的，而不是像经典情况下那样高概率发生。 这使得能够准确量化单个模型相对于单个数据集的拟合优度。

Hutter [62, 63, 65]将通用序列预测思想与决策理论代理相结合，形成了广义的通用半测度。 当代理不影响结果时，归纳得到优化，而将其与决策理论相结合允许代理与世界互动。 他提出的模型是不可计算的，但对其进行可计算的修改比任何时间或空间限制算法更智能。

10.2 算法信息论的数学含义

Chaitin [37] 认为，由于存在一些数学命题是真实的但无法证明的，数学应该被视为准经验的，而不是通常的先验方法。 这将允许在形式化的数学结构中添加新的公理，其中存在一些支持无法知晓命题的真实性或可证伪性的证据。 Chaitin [35] 认为爱因斯坦将数学视为经验的，甚至暗示整数是人类思维的产物。 Chaitin 还认为，虽然哥德尔以他的柏拉图世界观将数学视为先验活动，但哥德尔确实说过：“如果数学描述了客观世界，那么归纳方法在数学中应该被应用得和物理学中一样。”

这导致哥德尔认为对于一个新的公理的真实性的可能决策是归纳的。 如果这样一个可能的公理有大量的可验证的结果和强大的影响力，那么它可以被视为一个公理，就像一个假设被接受为物理理论一样。 这意味着数学通过明智地选择那些在经验上似乎是真实的公理，但尚未可证明的公理来决定“真实性”。

10.3 我们如何理解宇宙？

那些将宇宙的演化路径视为计算过程的研究人员提出了宇宙本身是数字计算机的可能性。 根据Chaitin [36]的说法，第一个这样做的人是莱布尼兹，因为莱布尼兹相信数字物理学。 最近，像约翰·惠勒（在[29]中引用）这样的知名人士创造了“从位来”的短语，而埃德·弗雷德金[54, 53]则将宇宙的基本粒子视为信息位的处理器。 在普朗克尺度下的$1.6 \times 10^{-35}\mathrm{m}$，量子效应占主导地位，宇宙似乎几乎可以被定义为一个量子计算机。

塞斯·劳埃德[79]（参见[78]）探讨了宇宙作为计算机的概念，因为它以系统化的方式处理信息。 他提出宇宙可能进行量子计算，其中每个自由度都会记录信息，而自由度之间的动态相互作用会处理信息。 劳埃德对宇宙的处理能力进行了限制。

如果考虑到引力，总位数约为 $\approx 10^{120}$，而操作数约为 $\approx 10^{120}$，它们是自然常数、h、c、G和宇宙的年龄的简单多项式函数。 时间视界 t内的操作数是普朗克时间，在这个时间尺度上，引力效应与量子效应的量级相同。 劳埃德研究了一个物质主导的宇宙和一个辐射主导的宇宙，并发现描述操作和位数的公式在两种情况下都大致相同。 本研究的目的不是深入探讨劳埃德提出的观点，而是指出将宇宙看作量子计算机的可能影响。即使宇宙是一个量子计算机，强丘奇图灵论题暗示，在没有无穷大的情况下，每个有限可实现的系统都可以被有限图灵机模拟[45]。 然而，基于连续对称性和隐藏变量的暗示，存在反对基于数字物理学的论证[1]。

即使如此，假设在普朗克尺度以上，数字计算可能有效地描述物理过程也是合理的。 计算机的内部状态可以被认为在量子限制内是离散的。

同样，将宇宙设想为一个在其相当明确的状态空间中步进的计算机并不困难。 例如，当碰撞将系统从一个微观状态转移到另一个微观状态时，容器中气体的行为可以被视为一个计算过程。

保罗·戴维斯在第5个奇迹（第95页[44]）中指出宇宙具有

与计算机相同的逻辑结构。 他从算法信息论的角度论证物理定律显示出模式，因此在信息上是贫乏的，即定律在算法上是简单的。 定律中体现的数据被认为是可压缩的。 另一方面，基因组富含信息且接近随机。 （但请参阅后来的xxx的研究，表明基因组是可压缩的。）然而，基因组不仅接近随机，而且高度特定。 戴维斯随后认为这是可能的，因为基因组是通过随机突变和自然选择的进化过程产生的。 这导致了一种压缩但具体的信息结构。 就个人而言，我认为复制过程，正如我在第9节中所论证的，是生成有序结构的一种自然方式。

Wolfram [107]提出了一个更强的观点，并认为我们在宇宙中看到的秩序是通过类似于细胞自动机所进行的简单计算过程产生的。 如果是这样，宇宙就不是随机的，而是伪随机的，因为产生的复杂结构是通过相对简单的计算过程产生的。 然后，Wolfram调查了简单的计算世界，希望能够对宇宙本身的计算过程有所了解，并声称一种万有理论（TOE）在原则上是可能的，通过尝试不同的简单计算机。 他认为，将宇宙以数字形式解释可能会提供比基于连续动力学的传统数学更好的见解。 有趣的是，在他的书的第12章中，Wolfram [107]详细介绍了算法信息论。

另一种方法是Chaitin [29]的方法，他讨论了宇宙的可理解性问题。他引用了魏尔（Weyl）引起了对莱布尼茨（Leibniz）的主张的关注，即物理定律必须简单。 Chaitin支持魏尔的观点，即简单性是科学认识论的核心，并举了一些著名科学家的简单性和复杂性的例子。 这些科学家支持这样一种观点：如果法则超出了我们的理解能力，那么相信宇宙是理性和有法则的是没有价值的。 那么，万有理论（TOE）是可能的吗？ 这样的TOE将最大程度地压缩可观察的宇宙。 Chaitin引用Barrow [29]的观点，认为即使发现了TOE，也不能确定是否存在更深层次的解释。 这种可能性似乎与Wolfram上面概述的一个简单的物理现实生成器的观点相矛盾。 然而，正如Chaitin指出的那样，他和Barrow描述的TOE是一种算法描述，受到哥德尔不可证明性的限制，而Wolfram则建议从最简单的开始系统地搜索宇宙的简单生成器。

Calude和Meyerstein [21]提出了一个问题，即宇宙是否有法则。他们的结论是，宇宙在“法则”一词的意义上是无序的。他们的论点实际上是说人类的思维无法理解宇宙的行为。 尽管这些作者首先提到了从Plato到Galileo的历史观点，但他们自己的论点集中在Gödel的定理上，该定理表明了一组无法证明但却是真实的陈述的集合

很大，无法证明无处不在。 这些论点很有趣，并捕捉到了Chaitin的说法：“上帝不仅在量子力学中玩骰子，甚至在整数中也是如此。” Calude和Meyerstein [21]认为，宇宙最好由他们称之为通过抛硬币产生的词汇表来描述。 在这个无限的词汇表中，每个可能的序列都会出现。 代表我们宇宙的这个字符串的部分可能是部分有序的，我们希望这个有序部分的词汇表可以通过科学部分解释。 但在其他地方，这个字符串可能是随机的。

然而，我们只有在有序的词汇部分才能理解。 Calude和Meyerstein认为，由于人类的思维永远无法完全理解数学，人类的思维也永远无法完全理解物理世界。 我们是系统的一部分，没有足够的能量资源来探索相对论宇宙。 另一方面，Casti和Karlqvist [24] 希望，虽然数学宇宙对人类思维来说可能过于算法复杂，但原则上物理宇宙可能并非如此。

为什么我们能理解宇宙?

我们似乎能够理解宇宙，但我们的能力受到三个原因的限制。

- 人类思维的存储能力有限。 即使通过外部存储和信息处理（如在四色定理的证明中所做的），人类处理信息的数量仍然有限。 如果计算机为我们处理信息，那么如果我们的思维无法直接处理信息，我们对结果会有相同的信心吗?

- 即使人类可以获得足够的存储空间，他们可能没有足够的时间来进行足够的计算步骤，以对宇宙的某个定律有足够的理解。 这类似于计算天气所需的时间可能比宇宙生成观察到的天气所需的时间更长。

- 最后，哥德尔限制可能会限制人类的意义构建能力。 即使其他问题不限制人类的理性，哥德尔的定理中隐含的论证表明，描述宇宙行为的大多数陈述对我们来说可能是无法证明的，无论是作为个体还是作为人类集合。 实际上，如果科学理解是基于形式逻辑过程的，大多数定理将超出我们的能力范围。

那么问题就出现了，为什么我们能够对任何事情有所理解。 正如爱因斯坦在Chaitin [29]引用的话中所说：“宇宙最令人费解的地方就是它是可以理解的”。 可能通过物理定律进行的意义构建在形式系统意义上并不严格。

因此，它不受哥德尔定理的限制。 我们可能有解释，但在某种程度上不需要一致性，而是可能体现在概率中的一种较低形式。 可能是因为我们的大脑是量子计算机，不受哥德尔定理的限制。

然而，以下论证表明，还原主义在我们的意义构建中起作用，是因为宇宙相对简单。

尽管一些哲学家和后现代主义者对此进行了攻击，科学还原主义的方法似乎在理解宇宙方面非常成功，这是令人惊讶的。

然而，我们似乎能够通过将整个看似不可渗透的世界简化为较小的可管理的组件来理解世界。 这无疑是因为世界的结构是这样的。换句话说，令人惊讶的不是宇宙的复杂性，而是它的算法简单性。 这是一种简单性，使得观察到的宇宙在很大程度上可以简化为简单的法则。 这些法则非常简单，以至于它们可以被一个相对于宇宙其他部分微不足道的有限大脑处理。 这种简单性似乎是因为宇宙的小部分可以简单地描述。 我可以算法地描述一个下落的石头的行为；一个作为更大整体的一部分的石头。 我对下落的石头的算法描述可以看作是一个更复杂算法中的子程序。 但在很大程度上，宇宙其他部分的输入对结果几乎没有影响。 宇宙其他部分对下落的石头的影响主要是小的扰动。 似乎是因为我们观察到的宇宙由嵌套在嵌套中的结构或顺序嵌套组成，我们才能理解它。每个嵌套层次可以映射到描述该层次结构的子程序。 在我们的研究水平上，许多子程序描述具有低算法复杂性的高度有序系统。

这表明我们可以在特定的嵌套层次上理解，因为系统的算法复杂性小于我们的认知过程的算法复杂性。 虽然我们可能没有计算能力完全理解系统，但我们似乎有能力理解嵌套在系统中的算法复杂性较低的子程序，并可以模拟这些子程序被整合到整体中的方式。 这种方法可能并不总是有效，但在实践中似乎经常有效。 然而，在重建子程序时，我们经常需要考虑明显随机的贡献；即超出我们理解能力的贡献。 外部影响通常会表现为随机性。 无论是隐藏的法则、噪音还是量子不确定性，在宏观世界中对我们来说似乎是偶然事件，对我们能够理解的宇宙的一部分只有很小的影响。也许我们依赖现象学的方式来进行理解，避免了哥德尔困境。 现象学法则的隐含或显性公理可能已经经验性地选择了两个可能的真实陈述之一，甚至选择了一个无法证明的真实陈述的二叉树路径。 这可能使我们能够在宏观层面上对高度算法组织的系统进行相对满意的描述。 然而，将宏观层面上的高度算法组织的系统与微观层面上的物理过程相关联可能是不可能的。

由于我们的认知限制，将这些更高层次的理解转化为更基本的理解是困难的。 当整体无法通过组合部分来解释时，可能是因为我们的大脑无法在严格的形式逻辑过程之外运作。

甚至可能是不确定性原理通过这样的过程产生。 然而，包括哥德尔在内的伟大物理学家和数学家对他的定理与不确定性原理之间的任何关系持怀疑态度，而量子物理学的形式框架不允许隐藏变量，谁知道呢？

附录A

扰动的确切成本

Zurek [109]已经证明，当信息在从输入状态i到输出状态o时丢失时，需要 $H(\ i\ |\ o)$比特来恢复系统到输入状态 i。 然而，一般情况下，由于 $H(i|\ o) = -H(o|\ i)$，除非计算是可逆的，否则这与从 i到o的信息丢失不匹配。 因此，扰动的熵影响与恢复过程的熵之间似乎没有简单的关系。

Zurek认为，稍微弱一点的要求是从算法压缩的描述中恢复系统

$$H\ (i\ |\ o^{\star}\ ) = |\ p^{\star}{}_{reg}\ | = H\ (i\ |\ o,\ H\ (o)\ ) = H\ (i)\ -\ H\ (o)\ , \tag{A.1}$$

其中恢复程序由$p^{\star}$ reg给出。 由于知道$o^{\star}$等同于同时知道o和H（o），较弱的版本需要更少的比特，因为必要的信息被捕捉在H（o）中。但是

$$U\ (p^{\star}{}_{reg},\ oH\ (o)\ ) = U\ (p^{\star}{}_{reg} H\ (o)\ ,\ o) = i,$$

因为H（o）可以是程序字符串的一部分，而不是输入字符串。 这意味着H（$i\ |\ o$）$= H$（$i\ |\ o^{\star}$）$+\ |\ H$（o）$\ |$ 或H（$i\ |\ o$）$= H$（$i\ |\ o^{\star}$）$+ \log_2 H$（o） （也参见Zurek [109]。

较弱的版本将用于描述干扰和任何调节响应

- 作为调节响应的较强版本必须替代丢失的历史$H(o)$，较强版本中的调节响应与干扰引入的算法熵并不简单相关
- 较弱版本是反对称的，即$H(o|i^*) = -H(i|o^*)$，在干扰后恢复系统的熵成本与系统的熵变相同，但流动方向相反
- 熵差$H(i)$ - $H(o)$与UTM无关且与路径无关

参考文献

[1] 数字物理学http://en.wikipedia.org/wiki/Digital_physics

[2] 飞行呈V字形可以以最小的努力获得最佳视野 新科学家，(四月)，2007年

[3] 沃尔夫拉姆的2,3图灵机是通用的，2007年。

[4] J. Alebraheem和Y. Abu-Hasan。在一个非周期解的两个捕食者-一个猎物模型中的捕食者的持久性。 应用数学科学，6(19)：943-956，2012年。

[5] W. R. Ashby。引论到控制论。 大学平装书，伦敦，1964年。

[6] A. Barron和T. Cover。最小复杂度密度估计。 *IEEE* 信息论交易，37(4)：1034-1054，1991年。

[7] A.R. Barron，J. Rissanen和B. Yu。模型和编码中的MDL原则。*IEEE*信息论交易，*I*T-44(6)：2743-2760，1998年。

[8] J. D. Barrow。不可能性：科学的极限和极限的科学。 Ra*ndom* Hou*se*，伦敦，*I*SBN：0-09-0772116，1999年。

[9] S. Beer.企业的核心. Wiley, Chichester, 1979.

[10] S. Beer. 公司的大脑，第二版 . John Wiley & Sons, Chichester, 1981.

[11] C. H. Bennett. 计算的逻辑可逆性。 *IBM*研究与发展杂志, 17:525–532, 1973.

[12] C. H. Bennett. 计算的热力学-一篇综述。国际理论物理学杂志, 21(12):905–940, 1982.

[13] C. H. Bennett. 逻辑深度和物理复杂性。 在R Herken, 编辑, 通用图灵机-半个世纪的调查, 页 227–257. 牛津大学出版社, 牛津, 1988.

[14] C. H. Bennett. 关于兰道尔原理、可逆计算和麦克斯韦的恶魔的注释。http://xxx.lanl.gov/PS_cache/physics/pdf/0210/0210005.pdf, 2003.

[15] C. H. Bennett, P. G´acs, M. Li, P. M. B. Vitanyi, and W. H. Zurek. 信息距离。I*EEE*信息论, 44(4):1407–1423, 1998.

[16] L. Brillouin. 科学与信息论。 学术出版社, 纽约, 第二版, 1962.

[17] H Buhrman, J. Tromp, and P. Vitanyi. 可逆模拟的时间和空间界限。Journal of Physics *A: M*athematical and *G*eneral,34:35:6821–6830, 2001.

[18] C. Calude. 信息与随机性：算法视角。
Springer-Verlag, 第二版, 2002.

[19] C. Calude, D. I. Campbell, K. Svozil, and D. Stefanescu. 强确定性与可计算性。在W. Depauli-Schimanovich, E. Koehler, 和F. Stadler, 编辑, *The Foundational Debate, Complexity and Construc-tivity in Mathematics and Physics*, 页115–131。 Kluwer,, 多德雷赫特,波士顿, 伦敦, 1995.

[20] C. Calude, M. J. Dinneen, and C. K. Shu. 计算随机性的一瞥。实验数学，11(3):361–370，2002年。

[21] C. S. Calude and F. W. Meyerstein. 宇宙是否有法则？ 混沌，孤立子和分形，106:1075–1084，1999年。

[22] J. Casti. 伟大的阿什比：复杂性、多样性和信息。 复杂性，2(1):7–9，1996年。

[23] J. L. Casti. 复杂化：通过惊喜科学解释一个矛盾的世界。哈珀柯林斯，1995年。

[24] J. L. Casti and A. Karlqvist, 编辑。 边界和障碍。. Addis-onWiley，纽约，1996年。

[25] 约翰·卡斯蒂和C. 卡鲁德。随机性：混乱的计算机。 新科学家，183(2466)：36-37，2004年。

[26] G. 赛丁。关于计算有限二进制序列的程序长度。*J. ACM*，13：547-569，1966年。

[27] G. 赛丁。一个与信息理论形式上相同的程序大小理论。AC*M*杂志，22：329-340，1975年。

[28] G. 赛丁。朝着“生命”的数学定义。在R. D. 莱文和M. 特里布斯，编辑，最大熵形式主义，页477-498。 麻省理工学院出版社，马萨诸塞州，1979年。

[29] G. 赛丁。关于宇宙的可理解性和简单性、复杂性和不可约性的概念。在W. 霍格雷贝和J. 布罗曼德，编辑，*Grenzen und Grenzberschreitungen, XI X. Deutscher Kongressfr Philosophie, Bonn, September 2002*，页517-534，柏林，2004年。
Akademie Verlag.

[30] G. J. Chaitin. 信息论中形式系统的信息论限制。 *J. ACM*，21(3)，1974年。

[31] G. J. Chaitin. 算法信息论。 在统计科学百科全书，卷1，第38-41页。Wiley，纽约，1982年。

[32] G. J. Chaitin. 哥德尔定理和信息。 理论物理国际期刊，22:941-954，1982年。

[33] G. J. Chaitin. 信息、随机性和不完备性：算法信息论论文集。 W*orld* S*cie*ntifi*c*，新泽西，1987年。

[34] G. J. Chaitin. 纯数学中的随机性和复杂性。国际分叉与混沌杂志，4:3-15，1994年。

[35] G. J. Chaitin.数学的极限. Springer-Verlag, 伦敦, ISBN 1-85233-668-4, 2003.

[36] G. J. Chaitin. 元数学！Ω的追求. Pantheon ISBN: 0375423133, 2005.

[37] G. J. Chaitin. 理性的极限。*Sc*ienti*fic A*meri*c*an, 294:74–81, 2006.

[38] T. M. Cover, P. Gcs, and R. M. Gray. Kolmogorov对信息论和算法复杂性的贡献。 *Ann. Probab*, 17:840–865, 1989.

[39] D. Crutchfield. 出现的演算法：计算、动力学和归纳。*Phys*i*c*a *D*, 75:11–54, 1994.

[40] J. P. Crutchfield 和 C. R. Shalizi. 因果状态的热力学深度：当在奥卡姆的池塘中划水时，肤浅是一种美德。
工作论文 98-06-047，圣塔菲研究所，1999年。

[41] J. P. Crutchfield 和 K. Young. 推断统计复杂性。物理评论快报，63:105–108，1989年。

[42] Feldman D.经典自旋系统的计算力学。 博士，加利福尼亚大学戴维斯分校，1998年。

[43] Saswato Das. 对我们的秘密数据构成的量子威胁。 新科学家，2007年9月13日(2621)。

[44] P. C. W. Davies. 第五个奇迹：寻找生命起源的探索。
企鹅图书有限公司，2003年。

[45] D. Deutsch. 量子理论，图灵原理和通用量子计算机。《皇家学会会议录》*A*系列，400:97–117，1985年。

[46] S. D. Devine. 算法信息论在噪声模式字符串中的应用。复杂性，12(2):52–58，2006年。

[47] S. D. Devine. 用简单复制模型的算法信息论方法研究秩序的出现。在第一届国际宇宙演化与发展会议。2008年。

[48] S. D. Devine. 算法熵的洞察力。熵，11(1):85–110，2009年。

[49] M. Eigen. 物质的自组织和生物大分子的演化。*N*atur*w*issenshaften，58:465–523，1971年。

[50] M. Eigen, J. McCaskill, and P. Schuster. 分子准种群。物理化学杂志，92(24):6881–6891，1988年。

[51] S. Feferman. 哥德尔，纳格尔，心智和机器：厄内斯特·纳格尔演讲，哥伦比亚大学，2007年9月27日。哲学杂志，106:201–219，2009年。

[52] D. P. Feldman and J. P. Crutchfield. 发现非临界组织：一维自旋系统中模式的统计力学、信息论和计算视角。(SFI工作论文98-04-026)，1998年。

[53] E. Fredkin. 数字力学。*Ph*y*s*ica D，页码254–270，1990年。

[54] E. Fredkin and T. Toffoli. 保守逻辑。国际理论物理杂志，21:219–253，1982年。

[55] P. G´acs. 关于算法信息的对称性。苏联数学-学报, 15:1477–1780, 1974.

[56] P. G´acs. 一些随机性测试的精确表达式。数学逻辑和数学基础杂志, 26:385–394, 1980.

[57] P. G´acs. 描述复杂性和随机性的讲义。讲-义，波士顿大学计算机科学系，1988年。最新版本在http://www.cs.bu.edu/~gacs/papers/ait-notes.pdf。

[58] P. G´acs. 波尔兹曼熵和随机性测试- 扩展摘要。在物理和计算研讨会的论文集中，页码209–216。IEEE计算机学会出版社，1994年。

[59] P. G´acs. 布尔兹曼熵和随机测试。技术报告，波士顿大学计算机科学系，2004年。http://www.cs.bu.edu/~gacs/papers/ent-paper.pdf。

[60] P.D. Gr¨unwald. 最小描述长度教程。在P.D.Gr¨unwald，I.J. Myung和M. A. Pitt，编辑，最小描述长度的进展：理论和应用。MIT出版社，2005年。

[61] M. H. Hansen和B. Yu. 模型选择和最小描述长度原则。美国统计协会杂志，96:746 774，2001年。

[62] M. Hutter. 基于算法复杂性的通用人工智能理论。http://arxiv.org/abs/cs.AI/0004001，2000年。

[63] M. Hutter.通用人工智能. Springer出版社，ISBN：3-540-22139-5，伦敦，2005年。

[64] M. Hutter. 关于广义可计算通用先验及其收敛性。http://arxiv.org/PS_cache/cs/pdf/0503/0503026v1.pdf，2005年。

[65] M. Hutter. 通用算法智能：一种数学top→down方法。在B. Goertzel和C. A. Pennachin编辑，人工智能，第227-290页。Springer出版社，柏林，2007年。

[66] E. T. Jaynes. 信息论和统计力学。 物理评论，106：620-630，1957年。

[67] E. T. Jaynes. 最大熵理论的现状。在R.D. Levine和M. Tribus编辑，最大熵形式主义，第15-118页。麻省理工学院出版社，剑桥，马萨诸塞州，1979年。

[68] E. T. Jaynes和G. L. Bretthorst。概率论：科学的逻辑。剑桥大学出版社，纽约，2003年。

[69] K. Kolmogorov。三种定量定义信息的方法。*Prob. Info. Trans.*，1:1–7，1965年。

[70] E. Kvaalen。黎曼猜想最终被证明了吗？ 新科学家，(2648):40–41，2008年。

[71] R. Landauer。计算过程中的不可逆性和热量产生。I*BM*研究与发展杂志，5:183–191，1961年。

[72] R. Landauer。信息是物理的。在*Proceedings of PhysComp 1992*，页1–4。洛斯阿拉米托斯：IEEE计算机学会出版社，牛津，1992年。

[73] H. S. Leff 和 A. F. Rex. 麦克斯韦的恶魔：熵、信息、计算. 普林斯顿大学出版社，普林斯顿，1990年。

[74] L.A. Levin. 关于随机序列的概念。 苏联数学报告，14(5):1413–1416，1973年。

[75] L.A. Levin. 信息规律（非增长）和概率论基础的一些方面。 信息传输问题，10(3):206–210，1974年。

[76] M. Li 和 P. M. B. Vitanyi. 科尔莫哥洛夫复杂性及其应用导论. 斯普林格出版公司，纽约，第二版，1997年。

[77] M. Li 和 P.M.B. Vitanyi. 可逆性和绝热计算：时间和空间与能量的交换。伦敦皇家学会*A*系列，452:769–789，1996年。

[78] S. Lloyd. 编程宇宙：量子计算机科学家探索宇宙. Knopf, ISBN 1-4000-4092-2, 纽约, 2006.

[79] Seth Lloyd. 宇宙的计算能力. *Phys. Rev. Lett.*, 88:237901, 2002年5月.

[80] P. Martin-L¨of. 随机序列的定义. 信息和控制, 9:602–619, 1966年.

[81] M. Minsky. 使用标签系统的通用图灵机的大小和结构. 在递归函数理论: 会议论文集, 纯数学, 卷5, 页码229–238, *P*rovelance, 1962年. AMS.

[82] J. Rissanen. 广义克拉夫特不等式和算术编码. *IBM* 研究与发展杂志, 20(3):198–203, 1976年.

[83] J. Rissanen. 通过最短数据描述进行建模。 *A*uto*m*atica，14:465–471，1978年。

[84] J. Rissanen. 用于整数的通用先验和最小描述长度估计。Annals of *S*tatistics，11(2):416–431，1983年。

[85] J. Rissanen. 随机复杂性和建模。 *A*nnals o*f* Statistics，14:1080–1100，1986年。

[86] J. Rissanen. 随机复杂性。J. Royal Statistical Society，49B(3):223–265和252–265，1987年。

[87] J. Rissanen. 统计调查中的随机复杂性。 W*orl*d *Sc*ientific，新泽西，1989年。

[88] E. D. Schneider和J. J. Kay。生命作为热力学第二定律的表现。*Mathl. Comput. Modelling*，16(6-8):25–48，1994年。

[89] C. P. Schnorr. 过程复杂性和有效的随机测试。 计算机和系统科学杂志，7:376–388，1973年。

[90] C. R. Shalizi 和 J. P. Crutchfield. 模式发现和计算机力学。ar*x*iv.or*g*/a*b*s/cs/000*10*2*7*v*1*，*2*000年。

[91] C. R. Shalizi 和 P. Crutchfield. 计算机力学模式和预测，结构和简洁。 统计物理学杂志，104:819–881，2001年。

[92] C. E. Shannon. 通信的数学理论。贝尔系统技术杂志，27:379–423，1948年。

[93] R. J. Solomonoff. 归纳推理的形式理论；第1部分和第2部分。信息与控制，7:1–22,224–254，1964年。

[94] R. J. Solomonoff. 基于复杂性的归纳系统：比较和收敛定理。*IEEE*信息论杂志，*IT*-24(4)：422–432，1978年。

[95] R. J. Solomonoff. 讲座1：算法概率。http://world.std.com/~rjs/iaplect1.pdf，2005年。

[96] R. J. Solomonoff. 讲座2。 算法概率的应用。http://world.std.com/~rjs/iaplect1.pdf，2005年。

[97] E Szathmary和J Maynard Smith。从复制体到繁殖体：导致生命的第一个重大转变。理论生物学杂志，187：555–571，1997年。

[98] L. Szilard. 通过智能体的干预在热力学系统中降低熵，2003年。

[99] S. Szilard. 关于智能生物对热力学系统熵减的影响。 物理学杂志，53:840–856，1929年。

[100] T. Toffoli. 物理和计算。国际理论物理学杂志，21:165–175，1982年。

[101] R. C. Tolman. 统计力学。 牛津大学出版社，伦敦，1950年。

[10*2*] A. Turing. 计算数的可计算性及其在决策问题中的应用。 伦敦数学学会论文集，第*2*系列，4*2*，1936年。

[103] N. K. Vereshchagin and P. M. B. Vit´anyi. Kolmogorov的结构函数和模型选择。 *IEEE*信息论交易，50(12):3265–3290，2004年。

[104] P. M. B. Vit´anyi 和 M. Li. 最小描述长度归纳，贝叶斯主义和科尔莫哥洛夫复杂性。*IEEE Trans. Inform. Theory*，46:446–464，2000年。

[105] P.M.B. Vit´anyi. 可逆计算中的时间、空间和能量。在*2005*年*ACM*国际计算前沿会议论文集，页码43*5*–444，意大利伊斯基亚，*2005*年。

[106] R. von Mises. 概率论的基础。 Mathe*ma*-*tische Zeitschrift*，5:52，1919年。

[107] S. Wolfram. 一种新的科学。 Wolfram M*edi*a，伊利诺伊州香槟市，2002年。

[108] W. H. Zurek. 算法随机性和物理熵。 *P*hysical *Review A*，40(8):4731–4751，1989年。

[109] W. H. Zurek. 计算的热力学，算法复杂性和信息度量。自然，341:119–124，1989年。