The stochastic thermodynamics of computation

The Stochastic Thermodynamics of Computation

David H. Wolpert

Santa Fe Institute, Santa Fe, New Mexico

Complexity Science Hub, Vienna

Arizona State University, Tempe, Arizona,

http://davidwolpert.weebly.com

One of the central concerns of computer science is how the resources needed to perform a given computation depend on that computation. Moreover, one of the major resource requirements of computers — ranging from biological cells to human brains to high-performance (engineered) computers — is the energy used to run them, i.e., the thermodynamic costs of running them. Those thermodynamic costs of performing a computation has been a long-standing focus of research in physics, going back (at least) to the early work of Landauer, in which he argued that the thermodynamic cost of erasing a bit in any physical system is at least $kT \ln[2]$.

One of the most prominent aspects of computers is that they are inherently *non*-equilibrium systems. However, the research by Landauer and co-workers was done when non-equilibrium statistical physics was still in its infancy, requiring them to rely on equilibrium statistical physics. This limited the breadth of issues this early research could address, leading them to focus on the number of times a bit is erased during a computation — an issue having little bearing on the central concerns of computer science theorists.

Since then there have been major breakthroughs in nonequilibrium statistical physics, leading in particular to the new subfield of "stochastic thermodynamics". These breakthroughs have allowed us to put the thermodynamic analysis of bit erasure on a fully formal (nonequilibrium) footing. They are also allowing us to investigate the myriad aspects of the relationship between statistical physics and computation, extending well beyond the issue of how much work is required to erase a bit.

In this paper I review some of this recent work on the "stochastic thermodynamics of computation". After reviewing the salient parts of information theory, computer science theory, and stochastic thermodynamics, I summarize what has been learned about the entropic costs of performing a broad range of computations, extending from bit erasure to loop-free circuits to logically reversible circuits to information ratchets to Turing machines. These results reveal new, challenging engineering problems for how to design computers to have minimal thermodynamic costs. They also allow us to start to combine computer science theory and stochastic thermodynamics at a foundational level, thereby expanding both.

I. INTRODUCTION

A. Computers ranging from cells to chips to human brains

Paraphrasing Landauer, real world computation involves thermodynamic costs [3] — and those costs can be massive. Estimates are that computing accounts for $\sim 4\%$ of current total US energy usage in 2018 — and all of that energy ultimately goes into waste heat.

Such major thermodynamic costs arise in naturally occurring, biological computers as well as the artificial computers we have constructed. Indeed, the comparison of thermodynamic costs in artificial and biological computers can be fascinating. For example, a large fraction of the energy budget of a cell goes to translating RNAs into sequences of amino acids (i.e., proteins), in the cell's ribosome. The thermodynamic efficiency of this computation — the amount of heat produced by a ribosome per elementary operation — is many orders of magnitude superior to the thermodynamic efficiency of my current artificial computers [4]. Are there "tricks" that cells use to reduce their costs of computation that we could exploit in our artificial computers?

More speculatively, the human brain consumes $\sim 10-20\%$ of all the calories a human being requires to live [5]. The need to continually gather all those extra calories is a massive evolutionary fitness cost that our ancestors faced. Does this fitness cost explain why human levels of intelligence (i.e., of neurobiological computation) are so rare in the history of life on earth?

These examples involving both artificial and biological computers hint at the deep connections between computation and statistical physics. Indeed, the relationship between computation and statistical physics has arguably been a major focus of physics research at least since Maxwell's demon was introduced. A particularly important advance was made in the middle of the last century when Landauer (and then Bennett) famously argued that erasing a bit in a computation results in a entropic cost of at least $kT\ln[2]$, where k_B is Boltzmann's constant and T is the temperature of a heat bath connected to the system.

This early work was grounded in the tools of equilibrium statistical physics. However, computers are highly *non*equilbrium systems. As a result, this early work was necessarily semiformal, and there were many questions it could not address.

On the other hand, in the last few decades there have been major breakthroughs in nonequilibrium statistical physics. Some of the most important of these breakthroughs now allow us to analyze the thermodynamic behavior of any system that can be modeled with a time-inhomogeneous continuous-time Markov chain (CTMC), even if it is open, arbitrarily far from equilibrium, and un-

dergoing arbitrary external driving. In particular, we can now decompose the time-derivative of the (Shannon) entropy of such a system into an "entropy production rate", quantifying the rate of change of the total entropy of the system and its environment, minus a "entropy flow rate", quantifying the rate of entropy exiting the system into its environment. Crucially, the entropy production rate is non-negative, regardless of the CTMC. So if it ever adds a nonzero amount to system entropy, its subsequent evolution cannot undo that increase in entropy. (For this reason it is sometimes referred to as *irreversible* entropy production.) This is the modern understanding of the second law of thermodynamics, for systems undergoing Markovian dynamics. In contrast to entropy production, entropy flow can be negative or positive. So even if entropy flow increases system entropy during one time interval (i.e., entropy flows into the system), often its subsequent evolution can undo that increase.

This decomposition of the rate of change of entropy is the starting point for the field of "stochastic thermodynamics". However, the decomposition actually holds even in scenarios that have nothing to do with thermodynamics, where there are no heat baths, work reservoirs, or the like coupled to the system. Accordingly, I will refer to the dynamics of the entropy production rate, entropy flow rate, etc. as the "entropy dynamics" of the physical system. In addition, integrating this decomposition over a non-zero time interval, we see that the total change in entropy of the system equals the total entropy flow plus the total entropy production. Since entropy production is non-negative, this means that the minimal value of total entropy flow is the total change in entropy, which is sometimes called the "Landauer cost".[1]

B. The effect of constraints on how we are allowed to perform a computation

These recently derived equations for the entropy dynamics of physical systems allow us to revisit the entire topic that was historically referred to as "the thermodynamics of computation", but now in a richer, and fully formal manner. For example, as discussed in Section VI in the case of bit erasure, we now understand that any function $f : X \subseteq \mathbb{Z} \to X$ can be implemented in a thermodynamically reversible manner, with no (irreversible) entropy production. This result holds for functions ranging from bit erasure, where $X = \{0, 1\}$) (see Section VI), to simple algebraic functions like $y = x \mod 3$ (where $X = \mathbb{Z}$) to the input-output function of finite deterministic automaton (where X is the joint state of all allowed input strings to the automaton and the internal state of the automaton). It even

[1] While I will focus on CTMCs in this review, the reader should be aware that nonequilibrium statistical physics extends beyond Markovian systems. See for example [6] and references therein, in which there is no restriction to Markovian systems.

holds for partial functions representing the input-output behavior of universal Turing machines (in which $X = \mathbb{Z}$ encodes the set of all finite input bit strings). Importantly, this thermodynamic reversibility holds even if f is logically irreversible (as bit erasure is).

These results are now well understood. However, they all assume that we can use an unconstrained physical system to implement f, with no restrictions on how the Hamiltonian couples the components of the full system-state, x. The richness of the current research on the thermodynamics of computation arises from the fact that if we wish to implement the function f in a real-world physical system, then *for practical reasons*, we rarely have the freedom to exploit an arbitrary Hamiltonian, that couples the components of x in an arbitrary manner. Instead, in practice f must be physically implemented by iterating one or more interconnected functions, $\{g_i\}$, each of which is constrained to lie in some highly restricted set of functions.

One example of this is where f is the function implemented by a particular Boolean circuit and the g_i are Boolean gates in the circuit. Another example is where f is the function implemented by a particular deterministic finite automaton. In this example there is a single g_i, specifying how the internal state of the automaton is updated based on its current state and the current input character, and that function is iterated using the successive characters of the input string until the end of the string is reached. The crucial point is that that g_i can only couple a very limited number of the components of the full input string x at a given time.

Most of conventional computational complexity theory is, in essence, the investigation of how the constraints on the functions $\{g_i\}$ affect the capabilities and resource demands of the function f constructed from those functions [7–11]. In particular, in general for any given function f we want to implement, and any fixed set of functions $\{g_i\}$ we are allowed to couple together to implement f, there are many (typically infinitely many) different ways of using those $\{g_i\}$ to implement f. For example, in general, for any fixed function f from one finite space to another, there are an infinite number of circuits, all using the same set of gates $\{g_i\}$, that all implement f. However, there are hard constraints on the relationship among the various resources (e.g., the number of gates, the maximal depth through the circuit from the inputs to the outputs, etc.) that each of those circuits requires. A central concern of computational complexity theory is understanding those constraints on the resource requirements, and how they changes as we vary f.

Similarly, many of the interesting issues in the thermodynamics of computation concern how the constraints on the functions $\{g_i\}$ affect the *thermodynamic* resource demands of any physical system constructed from those functions (in other words, their entropic costs). In the words of [12], "different implementations of the same computation (input to output) can have radically different

thermodynamic consequences". We can illustrate such consequences by again considering the fact that in general there are an infinite number of circuits, all using the same set of gates $\{g_i\}$, that implement the same function f. In general, for any fixed distribution over the inputs to the function f, those circuits will all differ in both their Landauer costs and their entropy production. This results in a highly nontrivial optimization problem, of finding the circuit with least entropic costs to perform a given function f.

This shared concern with constraints in both computational complexity theory and the thermodynamics of computation suggests that we can partially synthesize the two fields. In particular, it may be possible to relate the resource demands they separately investigate to each other, both in general, and in the context of particular computational machines more specifically.

An example of what such a synthesis might look like, described below in Section XIV A, involves the entropy dynamics of Turing machines. Loosely speaking, the Kolmogorov complexity of a given bit string σ, with respect to a given Turing machine M, is defined as the shortest length of any bit string x that we can input to M that results in M computing the output σ and then halting. Similarly, we can define the "Kolmogorov work" of σ as the smallest amount of *thermodynamic work*, W, such that there is some input string x to M that results in M computing σ while requiring work W. In general, both the Kolmogorov complexity and the Kolmogorov work of σ will vary depending on the Turing machine one uses to compute σ. However, it turns out that for any Turing machine M, the Kolmogorov complexity for M of computing σ and the Kolmogorov work for M of computing σ have a simple (though uncomputable!) relationship with one another.

C. Roadmap

I begin with background sections, first presenting notational conventions, then reviewing the relevant parts of information theory, and then reviewing relevant computer science theory. These are followed by a section that summarizes the modern understanding of the dynamics of entropy in arbitrary dynamical systems. I then briefly discuss the modern understanding of the relationship between logical and thermodynamic (ir)reversibility. (In short, the laws of physics no more forbid the erasure of a bit in a thermodynamically reversible manner than they forbid the compression of a piston of gas in a thermodynamically reversible manner.)

Computers, by definition, are open systems. Accordingly, "where one draws the dotted line", saying which precise physical variables are the "computer", and which are not, can affect the entropic costs one ascribes to the computer. In particular, in order to be able to use a computer multiple

times, and have both the dynamics of the logical variables and the thermodynamic variables behave the same way each time one runs the computer, we need to reinitialize variables internal to the computer when it finishes its computation. Accordingly, we need to carefully specify how the entropic costs associated with the reinitialization of variables are allocated among the computer and the offboard systems interacting with the computer. (Some of the primary sources of confusion in the literature arise with papers that are not fully explicit about how they do this.) The convention I adopt in this paper for how to do this is presented in Section VII.

The following sections consider the entropic costs of various information processing systems. These sections are ordered from the simplest forms of information processing to progressively more complicated forms. Those who are interested in the costs of full computational machines, in the computer science sense of the term, can skip Section VIII, whichs consider simpler kinds of information processing systems, jumping from Section VII directly to Section IX.

In Section VIII, I review the entropic costs of a physical process that implements a single, arbitrary conditional distribution over its state space, without any constraints on its rate matrices, Hamiltonians, etc. (Bit erasure is a special example of such a process.)

The analysis to this point in the paper concerns information processing systems that have no *a priori* constraints on their operating behavior. However, subcomponents of full computers only have access to a limited number of degrees of freedom of the overall computer. These provide major constraints on how any full computer is allowed to transform its input into an output. (Indeed, full computers, in the Chomsky hierarchy sense, are *defined* by the constraints in how they transform their inputs into their outputs.) General results concerning the relationship between such constraints on how a subcomponent of a computer can be coupled to the rest of the computer on the one hand, and the consequences for the entropy dynamics of the overall computer on the other hand, are reviewed in Section IX.

These results are the foundation of a recent analysis of the entropy dynamics of "straight line" circuits, that have no loops or branching structure. I review this analysis in Section X.

There have been confusing claims in the literature concerning the possible thermodynamic benefits of implementing a computation using a circuit made out of logically reversible gates, as opposed to implementing that same computation using a circuit made out of arbitrary (in general logically irreversible) gates. In Section XI, I apply the convention for how to allocate entropic costs that was introduced in Section VII, to clarify some of these claims.

In Section XII I review some work on the Landauer costs of finite automata that only run once.

Then in Section XIII I review some recent results on the thermodynamic costs of information

ratchets, which can be viewed an extension of what are called "transducers" in computer science. These results range from analyzing detailed physical models of systems that act as information ratchets [13, 14] to a relatively large body of literature on infinite-time limiting behavior of the entropy dynamics of information ratchets [15–18].

After this, in Section XIV I review recent research on the entropy dynamics of Turing Machines (TMs) that uses modern nonequilibrium statistical physics. I also review some earlier work that (because it was done before modern nonequilibrium statistical physics was developed) had to use *equilibrium* statistical physics to analyze TMs.

Either implicitly or explicitly, most of the work in the literature on the entropy dynamics of computers, stretching back to the early work of Landauer but also including modern work, has assumed that a computational machine is implemented as the discrete-time dynamics induced by an underlying CTMC, e.g., a master equation. (In particular, this is the case for all systems analyzed in stochastic thermodynamics). In Section XV I discuss a recently discoverd surprising limitation of this assumption. Specifically, any non-trivial computation over a set of "visible" physical states cannot be implemented as the discrete-time dynamics induced of *any* underlying CTMC. For such "visible" discrete-time computation to arise, the underlying CTMC must couple the visible states to a set of "hidden" states. In general, the number of hidden states needed will depend on the computation being implemented. Moreover, the continuous-time dynamics of the joint visible-hidden system has to proceed through a nonzero, countable number of successive time-intervals which are demarcated by the raising or lowering of infinite energy barriers. The number of such intervals will depend on both the visible computation being implemented and the number of hidden states.

In essence, any non-trivial visible computation is embedded in a larger, hidden computation. To give a simple example, any CTMC that implements a simple bit flip has to have at least one hidden state in addition to the two visible states of the bit, and requires at least three hidden time-intervals to complete.

In the last section, I present some possible future directions for research. In particular, I describe a set of "computer-science-style" open questions that concern the thermodynamic properties of computational machines rather than the other kinds of resources (amount of memory, number of timesteps, etc.) traditionally considered in computer science.

D. Topics not covered in this paper

In this paper I will review the literature on the dynamics of entropy in physical systems that implement the computational machines considered in conventional computer science theory [10]. To clarify the precise topics considered in this paper, it is worth listing some of the related topics that I do *not* consider.

I do not consider all computational machines, but only a few of the most common ones. So for example, I do not consider the thermodynamics of running push-down automata. More broadly, I do not consider the thermodynamics of running analog computers (see [19] for work on this topic).

I also do not review any work that analyzes abstract dynamical systems (e.g., cullular automata) using mathematical techniques that just happen to have been developed in the statistical physics literature. The focus of this paper is not the *mathematical techniques* of statistical physics, but rather the *actual* statistical physics of physical systems. (See [20] for some interesting recent work on applying statistical physics techniques to analyze abstract dynamical systems.)

In this paper I will not consider the thermodynamics of quantum mechanical information processing, or indeed any aspect of quantum computation. See [21] for a good overview of quantum computation in general, [22, 23] specifically for overviews of the thermodynamics of quantum information processing, [24] for a review of quantum speed limits, and [25–27] for work on the related topic of quantum mechanical "resource theory".

However, it should be noted that even if quantum computers become a reality, they may have limited impact on the thermodynamic costs of real-world computers. One reason for this is that quantum computers as currently understood could only reduce the costs (and increase the speed) of an extremely limited set of possible computations, like factoring products of prime numbers. More importantly, many of the major challenges facing modern computing, both in terms of speed and thermodynamic costs, do not arise in the operation of the CPU (which is where quantum computation might be helpful), but rather in the i/o. In applications ranging from widely popularized applications of deep learning to search engines to simple back-office database management, modern computation exploits *massive*, heterogeneous data sets. Perhaps the primary difficulty faced by computers running these applications is in accessing that data and performing low-level (e.g., cache-based) processing of that data, not in the elaborate transformation of that data within a CPU. Yet as currently envisioned, quantum computers would not help reduce the thermodynamic costs of all this i/o.

Similarly, I do not consider the fundamental computational limitations of physical systems in

our universe. Some good discussions of this topic can be found in [28–32].

I will also not consider the (copious) literature on the thermodynamics of biochemical processes in cells that can be viewed as noisy types of information processing. (See [33–39] for some work on this topic.) The key criterion for including some literature in the current paper is whether it concerns thermodynamic properties of one of the types of computational machine considered in the computer science literature. (See [40–42] for some work on the preliminary issue of how closely the standard computational machines considered in computer science can be modeled with biochemical systems in the first place, and see [43–47] for work concerned with modeling such machines with arbitrary chemical reaction networks, irrespective of whether they arise in naturally occurring biological systems.)

In addition, there have been recent papers, building on [48], that have analyzed the thermodynamics of systems that gather information about their noisy environment and build models from that data, mostly using neural networks. These too are out of scope, since their focus is not on computation per se.

Furthermore, I do not consider the joint thermodynamics of control systems interacting with their environment, since that would require analyzing the environment as well as the controller, along with their coupling. Some papers that analyze the thermodynamics of control systems interacting with their environment are [49–57]. In addition, of course, there is a huge literature on the special type of control system designed to extract work from an equilibrium heat bath, i.e., the thermodynamics of Maxwell's demon. Some representative papers on this topic, showing how to use modern nonequilibrium statistical physics to resolve most (arguably all) of the mysteries concerning Maxwell's demon, are [6, 13, 58, 59].

In addition, in all the analysis below I identify the *microstates* of a physical system with the logical variables of the associated computation (i.e., with the so-called "information-bearing degrees of freedom" [60, 61]). In real-world computers of course, it is rarely the case that microstates directly code for logical variables. Instead the logical variables are typically identified with macrostates, usually defined by coarse-graining the space of microstates. (Some authors prefer to characterize the logical variables as an intermediate level of coarse-grained "meso-states".) The analysis below can be extended to apply to such scenarios (see [62, 63] for examples). However, this requires extra notational complexity, and in many cases also requires extra assumptions (concerning the microstate dynamics among the macrostates), and so is not pursued here.

I focus here on the integration of topics in *theoretical* computer science and the theory of nonequilibrium statistical physics. I will not consider the thermodynamic properties of real-world artificial

computers in any detail. However, it is important to note that several major engineering communities are working to reduce the entropic cost of such computers. At present, these efforts are based on a phenomenological characterization of the thermodynamics of computation. These have led to techniques like approximate computing (in which noise is added to the computation to reduce the heat it produces [64–66]) adaptive slowing (in which subcomponents of the computer are slowed down to reduce the heat they produce) and automatic code reconstruction to minimize energy consumption [67] (using techniques similar to those in optimizing compilers). It is intriguing to note that at a high level, these macroscopic phenomena of real-world computers parallel analogous phenomena that are central to the theoretical analyses reviewed below, even though those analyses concern behavior on the far smaller energy scale of $k_B T$.

Even within these strong restrictions on the subject matter of this paper, there are directly relevant results in the literature that are not discussed. In particular, [68] presents preliminary theorems concerning the extension of computational complexity theory [7] to include thermodynamic costs, in addition to the more conventional costs considered in computer science. [68] goes on to introduce a number of nontrivial open questions. That paper is aimed at the computer science theory community however, and without a lot more background than what is presented in Section IV below, it would be opaque to most physicists. This is why that paper is not discussed here.

II. TERMINOLOGY AND GENERAL NOTATION

In this section I introduce some of the terminology and notation I will use. I also define "islands", which will be needed in much of the analysis concerning entropy production.

A. Notation

I take the binary values to be $\mathbb{B} \equiv \{0, 1\}$. As usual, for any set A, A^* is the set of all finite strings of elements from A. Given a directed acyclic graph (DAG) Γ, I indicate its root nodes as $R(\Gamma)$. I write the Kronecker delta function as

$$\delta(a, b) = \begin{cases} 1 \text{ if } a = b \\ 0 \text{ otherwise} \end{cases} \tag{1}$$

So in particular, the vector indexed by values b that equals 1/0 depending on whether b equals some particular scalar a is written as $\delta(a, .)$.

I identify 1 as the Boolean TRUE, and 0 as FALSE. In agreement with this, I write the indicator function for any Boolean function $f(z)$ as

$$\mathbf{I}(f) = \begin{cases} 1 \text{ if } f(z) = 1 \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

I write $\ell(\vec{\lambda})$ (or sometimes $|\vec{\lambda}|$) to mean the length of any finite string $\vec{\lambda}$. So given any integer m, $P(\ell(\vec{\Lambda}) = m)$ means $P(\vec{\lambda} : \ell(\vec{\lambda}) = m)$. As usual, the concatenation of a string $\vec{\lambda}'$ after a string $\vec{\lambda}$ is written as $\vec{\lambda}\vec{\lambda}'$. In addition, I use the common notation that $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$ are the ceiling and floor operators, respectively.

In general, random variables are written with upper case letters, and instances of those random variables are written with the corresponding lower case letters. When the context makes the meaning clear, I will often also use the upper case letter indicating a random variable, e.g., X, to indicate the set of possible outcomes of that random variable. For any distribution $p(x)$ defined over the set X, and any $X' \subseteq X$, I write $p(X') = \sum_{x \in X'} p(x)$. Given some set X, I write the set of all distributions over the elements of X whose support is restricted to some subset $X' \subseteq X$ as $\Delta_{X'}$. Finally, given any conditional distribution $\pi(y|x)$ and a distribution p over X, I write πp for the distribution over Y induced by π and p,

$$(\pi p)(y) := \sum_{x \in X} \pi(y|x) p(x) \tag{3}$$

B. Islands

Computational machines are most often defined in terms of single-valued, logically-irreversible state-update functions. Often that update function f operating over the entire domain X can be expressed as a set of distinct functions $\{f_i\}$, "operating in parallel", each with its own distinct domain and distinct image. Intuitively, one might expect that one should analyze the entropic costs of such an update function f by appropriately averaging the entropic costs of each of the distinct parallel functions $\{f_i\}$. That is indeed the case, as elaborated in this subsection.

Suppose that a given physical process implements a single-valued function f over a space X. The **islands** of f are defined as the pre-images of f, i.e., the sets $\{f^{-1}(x) : x \in X\}$ [69]. I will write the set of islands of a function f as $L(f)$. As an example, the logical AND operation,

$$\pi(c|a,b) = \delta(c, a \wedge b)$$

has two islands, corresponding to $(a,b) \in \{\{0,0\},\{0,1\},\{1,0\}\}$ and $(a,b) \in \{\{1,1\}\}$, respectively. I write the set of all distributions over an island $c \in L(f)$ as Δ_c. I make the obvious definitions that for any distribution $p(x)$ and any $c \in L(f)$, the associated distribution over islands is $p(c) = \sum_{x \in c} p(x)$. As shorthand, I also write $p^c(x) = p(x|X \in c) = p(x)\mathbf{I}(x \in c)/p(c)$.

Note that the islands of a dynamic process depends on how long it runs. For example, suppose $X = \{a,b,c\}$, π is a single-valued, deterministic function, and $\pi(a) = a, \pi(b) = a$, while $\pi(c) = b$. Then π has two islands, $\{a,b\}$ and $\{c\}$. However if we iterate π we have just a single island, since all three states get mapped under π^2 to the state a.

In the more general case where the physical process implements an arbitrary stochastic matrix π that may or may not be single-valued, we extend the definition of islands to be the transitive closure of the equivalence relation,

$$x \sim x' \Leftrightarrow \exists x'' : \pi(x''|x) > 0, \pi(x''|x') > 0 \tag{4}$$

(Note that x and x' may be in the same island even if there is no x'' such that both $P(x''|x) > 0$ and $P(x'|x) > 0$, due to the transitive closure requirement.) Equivalently, the islands of π are a partition $\{X^i\}$ of X such that for all X^i, $x \notin X^i$,

$$\operatorname*{supp}_{x' \in X} \pi(x'|x) \cap \bigcup_{x'' \in X^i} \operatorname*{supp}_{x' \in X} \pi(x'|x'') = \varnothing \tag{5}$$

Although the focus of this paper is computers, which are typically viewed as implementing single-valued functions, most of the results below also hold for physical processes that implement general stochastic matrices as well, if one uses this more general definition of islands. Note that for either single-valued or non-single-valued stochastic matrices π, the islands of π will in general be a refinement of the islands of π^2, since π may map two of its islands to (separate) regions that are mapped on top of one another by π^2. This is not always the case though, e.g., it is not the case if π permutes its islands.

III. INFORMATION THEORY

In this section I review the parts of information theory that are relevant for this paper, and specify the notation I will use. I also introduce some new information theoretic concepts, which arise naturally in the analysis of the entropy dynamics of circuits presented below in Section X.

The Shannon entropy of a distribution over a set X, the Kullback-Leibler (KL) divergence between two distributions both defined over X (sometimes called the "relative entropy" of those

two distributions), and the cross-entropy between two such distributions, respectively, are defined as

$$S(p(X)) = -\sum_{x\in X} p(x)\ln p(x) \tag{6}$$

$$D(p(X)\|r(X)) = \sum_{x\in X} p(x)\ln\frac{p(x)}{r(x)} \tag{7}$$

$$S(p(X)\|r(X)) = S(p(X)) + D(p(X)\|r(X)) \tag{8}$$

$$= -\sum_{x\in X} p(x)\ln r(x) \tag{9}$$

(I adopt the convention of using natural logarithms rather than logarithms base 2 for most of this chapter.) I sometimes refer to the second arguments of KL divergence and of cross-entropy as a **reference distribution**. Note that the entropy of a distribution p is just the negative of the KL divergence from p to the uniform reference distribution, up to an overall (negative) constant.

The conditional entropy of a random variable X conditioned on a variable Y under joint distribution p is defined as

$$S(p(X|Y)) = \sum_{y\in Y} p(y)S(p(X|y))$$

$$= -\sum_{x\in X, y\in Y} p(y)p(x|y)\ln p(x|y) \tag{10}$$

and similarly for conditional KL divergence and conditional cross-entropy. The *chain rule* for entropy [70] says that

$$S(p(X|Y)) + S(p(Y)) = S(p(X,Y)) \tag{11}$$

Similarly, given any two distributions p and r, both defined over $X \times Y$, the conditional cross entropy between them equals the associated conditional entropy plus the associated conditional KL divergence:

$$S(p(X|Y)\|r(X|Y)) = S(p(X|Y)) + \ D(p(X|Y)\|r(X|Y)) \tag{12}$$

$$= -\sum_{x\in X, y\in Y} p(x,y)\ln r(x|y) \tag{13}$$

The mutual information between two random variables X and Y jointly distributed according to p is defined as

$$I_p(X;Y) \equiv S(p(X)) + S(p(Y)) - S(p(X,Y)) \tag{14}$$

$$= S(p(X)) - S(p(X|Y)) \tag{15}$$

(I drop the subscript p where the distribution is clear from context.) The *data processing inequality* for mutual information [70] states that if we have random variables X, Y, and Z, and Z is a stochastic function of Y, then $I(X;Z) \leq I(X;Y)$.

Where the random variable is clear from context, I sometimes simply write $S(p)$, $D(p\|r)$, and $S(p\|r)$. I also sometimes abuse notation, and (for example) if a and b are specified, write $S(A = a|B = b)$ to mean the conditional entropy of the random variable $\delta(A, a)$ conditioned on the event that the random variable B has the value b. When considering a set of random variables, I usually index them and their outcomes with subscripts, as in $X_1, X_2, \ldots$ and $x_1, x_2, \ldots$. I also use notation like $X_{1,2}$ to indicate the joint random variable (X_1, X_2).

When considering a set of random variables, I usually index them and their outcomes with superscripts, as in $X_1, X_2, \ldots$ and $x_1, x_2, \ldots$. I also use notation like $X_{1,2}$ to indicate the joint random variable (X_1, X_2).

One extension of mutual information to more than two random variables is known as total correlation, or **multi-information** [71]:

$$\mathcal{I}(p(X_1; X_2; \ldots)) \equiv \left[\sum_i S(p(X_i))\right] - S(p(X_{1,2,\ldots})) \tag{16}$$

which when the context is clear I abbreviate as $\mathcal{I}(p(X_{1,2,\ldots}))$. I sometimes use this same notation when X has just two components, in which case multi-information is the same as mutual information. Like mutual information, multi-information is always non-negative [71]. The multi-information of a distribution p over $X_{1,2,\ldots}$ is a measure of the amount of information I can learn from the random variables $X_1, X_2, \ldots$ considered together, that I cannot extract from them considered in isolation from one another. In other words, it is a measure of the strength of the statistical dependencies of the variables $X_1, X_2, \ldots$, under p.

I sometimes use superscript conditioning bars to indicate conditional versions of these information theoretic quantities. In particular, given two joint distributions $p^{a,b}$ and $r^{a,b}$ over a product space $X^{a,b} = X^a \times X^b$, I sometimes write the conditional KL divergence between p and r of X^a given X^b as

$$D(p^{a|b}\|r^{a|b}) = \sum_{x^a, x^b} p(x^a, x^b) \ln \frac{p(x^a|x^b)}{r(x^a|x^b)} \tag{17}$$

I write $S(\pi p)$ to refer to the entropy of distributions over Y induced by $p(x)$ and the conditional distribution π, as defined in Eq. (3). I use similar shorthand for the other information-theoretic

quantities, $D(\cdot\|\cdot)$, $S(\cdot\|\cdot)$ and $\mathcal{I}(\cdot)$. In particular, the *chain rule* for KL divergence and the *data-processing inequality* for KL divergence, respectively, are [70]:

1. For all distributions p, r over the space $X^a \times X^b$,

$$\begin{aligned} &D(p(X^a, X^b)||r(X^a, X^b)) \\ &\qquad = D(p(X^b)||r(X^b)) + D(p(X^a|X^b)||r(X^a|X^b)) \end{aligned} \tag{18}$$

2. For all distributions p, r over the space X and conditional distributions $\pi(y|x)$,

$$D(p\|r) \geq D(\pi p\|\pi r) \tag{19}$$

(Note that by combining the chain rule for KL divergence with the chain rule for entropy, we get a chain rule for cross entropy.)

Some of the results reviewed below are formulated in terms of the **multi-divergence** between two probability distributions over the same multi-dimensional space. This is an information-theoretic measure recently introduced in [69], which can be viewed as an extension of multi-information to include a reference distribution. It is defined as follows:

$$\begin{aligned} &\mathcal{D}(p(X_1; X_2, \ldots)\|r(X_1; X_2; \ldots)) \\ &\equiv \sum_{x_1, x_2, \ldots} p(x_1, x_2, \ldots) \ln \frac{p(x_1, x_2, \ldots)}{r(x_1, x_2, \ldots)} \prod_i \frac{r(x_i)}{p(x_i)} \qquad (20) \\ &= D(p(X_{1,2,\ldots})\|r(X_{1,2,\ldots})) - \sum_i D(p(X_i)\|r(X_i)) \qquad (21) \end{aligned}$$

When the context is clear I sometimes abbreviate this as $\mathcal{D}(p(X_{1,2,\ldots})\|r(X_{1,2,\ldots}))$.

Multi-divergence measures how much of the divergence between p and r arises from the correlations among the variables $X_1, X_2, \ldots$, rather than the marginal distributions of each variable considered separately. See App. A for a discussion of the elementary properties of multi-divergence and its relation to conventional multi-information, e.g., as a measure of the "joint information" among a set of more than two random variables.

Finally, the difference between conventional multi-information and multi-divergence is another new information-theoretic quantity, called **cross multi-information**:

$$\begin{aligned} &\mathcal{I}(p(X_1; X_2; \ldots)\|r(X_1; X_2; \ldots)) \\ &\quad \equiv \mathcal{I}(p(X_{1,2,\ldots})) - \mathcal{D}(p(X_{1,2,\ldots})\|r(X_{1,2,\ldots})) \\ &\quad = \left[\sum_i S(p(X_i)\|r(X_i))\right] - S(p(X_{1,2,\ldots})\|r(X_{1,2,\ldots})) \end{aligned}$$

(When the context is clear, I abbreviate cross multi-information as $\mathcal{I}(p(X_{1,2,\ldots})\|r(X_{1,2,\ldots}))$.)

Adopting an information-theoretic perspective, $\mathcal{I}(p(X_{1,2,\ldots})\|r(X_{1,2,\ldots}))$ is the reduction in expected codeword length if we use one type of coding rather than the other. Under the first, less efficient coding scheme, a codeword is created by concatenating codewords produced separately for each component of x_i using distinct codebooks that are separately optimized for each of those components, where all of these codebooks are optimized for r while events are actually generated by sampling p. Under the second, more efficient coding scheme, a codeword is created by using a single codebook that is optimized for joint events $(x_1, x_2, \ldots)$. Thus, $\mathcal{I}(p(X_{1,2,\ldots})\|r(X_{1,2,\ldots})) = 0$ if r is a product distribution.

As shorthand, I often write $\mathcal{I}(p)$, $\mathcal{D}(p\|r)$, and $\mathcal{I}(p\|r)$ when the set of random variables $X_{1,2,\ldots}$ is clear from context. In the usual way, if $r(x)$ and $p(x)$ both equal 0 for some x's, then I implicitly redefine $D(p\|r)$ and $\mathcal{D}(p\|r)$ to be the limit as those probabilities go to zero.

IV. COMPUTATIONAL MACHINES

The term "computer" means many different things in the literature. To avoid this imprecision, here I will instead use the term **computational machine**, defining it to mean one of the members of the Chomsky hierarchy [10].

In this section I introduce the particular computational machines whose entropy dynamics I will consider in this paper. These machines are introduced in order of increasing computational power; the set of computations that straight-line circuits can perform are a strict subset of the set of computations that finite automata can perform; these in turn are a strict subset of the set of computations that transducers can perform; and these in turn are a strict subset of the set of computations that a Turing machine can perform.[2]

However, I will also consider some digital systems that "process information", but occur as subsystems of some computational machines, and are less computationally powerful than any of them. As an example, the information processing system that has perhaps been studied more than other one in the physics literature is simple bit erasure, which is not a member of the Chomsky hierarchy. Another example is a single gate in a circuit, e.g., a XOR gate. I will sometimes refer to these kinds of extremely simple information processing systems as **sub-computers**, and use the term **(computational) device** to refer to either sub-computers or full-fledged computational

[2]The reader should be warned that there are some statements in the recent nonequilibrium statistical physics literature to the effect that transducers are as computationally powerful as Turing machines. These are mistaken.

machines. I will sometimes make the further distinction that a device considered as an abstract mathematical transformation (as in computer science theory) is a "logical device", while a device considered as a physical system (as in entropy dynamics) is a "physical device".

A. Bayes nets

Although Bayes nets are not normally viewed as computational machines, they provide a natural framework for investigating several kinds of computational machines, in particular straight-line circuits (considered in the next subsection). Accordingly, I summarize what Bayes nets are in this subsection.

Formally, a **Bayes net** C is a tuple (V, E, F, X) [72]. The pair (V, E) specifies the vertices and edges of a directed acyclic graph (DAG). X is a Cartesian product $\prod_v X_v$, where each X_v is the space of the variable associated with node v. F is a set of conditional distributions, one for each non-root node v of the DAG, mapping the joint value of the (variables at the) parents of v, $x_{\mathrm{pa}(v)}$, to the value of (the variable at) v, x_v.

An **input** node is a root node, and an **output** node is a leaf node. I assume that no output node is also an input node. I indicate the set of all nodes that are the parents of node v as $\mathrm{pa}(v)$. For any Bayes net C, I write the conditional distribution implemented at its node v (specified in F) as $\pi_v^C(x_v|x_{\mathrm{pa}(v)})$. When C is implicit, I sometimes shorten this to π_v. Note that this conditional distribution reduces to a prior distribution $\pi_v(x_v)$ whenever v is a root node,

In terms of this notation, the conditional distribution implemented by the overall Bayes net is

$$\pi^C(x|x_{\mathrm{IN}}) = \prod_{v \in V} \pi_v^C(x_v|x_{\mathrm{pa}(v)})$$

Similarly, the combination of the distribution $p_{\mathrm{IN}}(x_{\mathrm{IN}})$ and the conditional distributions at the non-root nodes of the Bayes net specifies a joint distribution over $x \in X$, the joint space of all nodes in the Bayes net, given by

$$p(x) = p_{\mathrm{IN}}(x_{\mathrm{IN}})\pi^C(x|x_{\mathrm{IN}})$$

This joint distribution in turn specifies a conditional distribution of the joint state of the output nodes of the Bayes net, given the joint state of the input nodes. As an important example, when the distributions in F are all single-valued functions, this conditional distribution reduces to a single-valued input-output function implemented by the Bayes net as a whole. In general, there are an infinite number of Bayes nets that implement the same conditional distribution of outputs given inputs.

For each node v in the Bayes net, I write the distribution formed by marginalizing $p(x)$ down to X_v as $p_v(x_v)$. I also write the marginal distribution over the joint state of the parents of any node v in the Bayes net as $p_{\mathrm{pa}(v)}(x_{\mathrm{pa}(v)})$. I refer to these two distributions as the distribution p_{IN} **propagated** to v and to $\mathrm{pa}(v)$, respectively.

B. Straight-line circuits

A **(straight-line) circuit** C is a tuple (V, E, F, X) that can be viewed as a special type of Bayes net [72–74]. Intuitively, the DAG of the Bayes net, (V, E), is the wiring diagram of the circuit. In conventional circuit theory, the all conditional distributions in the set F are single-valued functions.

Note that following the convention in the Bayes nets literature, with this definition of a circuit we orient edges in the direction of logical implication, i.e., in the direction of information flow. So the inputs to the circuit are the roots of the associated DAG, and the outputs are the leaves. The reader should be warned that this is the *opposite* of the convention in computer science theory. When there is no risk of confusion, I simply refer to a circuit C, with all references to V, E, F or X implicitly assumed as the associated elements defining C.[3] Just like in Bayes nets, we refer to the maximal number of nodes on any path going from a root node to a leaf node of a circuit as the **depth** of that circuit. (In other conventions, the depth is the number of links along that path, which is one less than the number of nodes on it.)

I use the term **gate** to refer to any non-root node in a circuit. In the special case where all non-output nodes in a circuit have outdegree 1 and there is a single output node, the circuit is called a (circuit) **formula**. In the context of formulas, I use v_{OUT} to indicate the single element of V_{OUT}. In the context of circuits, I use the term **AO circuit** to mean a circuit with a single gate, and write $AO(C)$ to mean an AO circuit that implements the same conditional distribution π^C as C.

Finally, I sometimes use the terms "gate", "circuit", etc., to refer to physical systems with physical states, and sometimes to refer to the associated abstract mathematical conditional distributions in a DAG. The intended meaning should be clear from context – when I need to be explicit, I will refer to **physical circuits** and **computational circuits**.

Straight line circuits are an example of **non-uniform** computers. These are computers that can only work with inputs of some fixed length. (In the case of a circuit, that length is specified by the

[3] A more general type of circuit than the one considered here allows branching conditions at the nodes and allows loops. Such circuits cannot be represented as a Bayes net. To make clear what kind of circuit is being considered, sometimes the branch-free, loop-free type of circuit is called a "straight-line" circuit [9].

number of root nodes in the circuit's DAG.) One can use a single circuit to compute the output for any input in a set of inputs, so long as all those inputs have the same length. If on the other hand one wishes to consider using circuits to compute the output for any input in a set that contains inputs of all possible lengths, then one must use a **circuit family**, i.e., an infinite set of circuits $\{C_i : i \in \mathbb{Z}^+\}$, where each circuit C_i has i root nodes.

In contrast to non-uniform computers, **uniform** computers are machines that can work with arbitrary length inputs.[4] In general, the number of iterations a particular uniform computational machine requires to produce an output is not pre-fixed, in contrast to the case with any particular nonuniform computational machine. Indeed, for some inputs, a uniform computational machine may never finish computing. The rest of this section introduces some of the more prominent uniform computational machines.

C. Finite Automata

One important class of (uniform) computational machines are the finite automata. There are several different, very similar definitions of finite automata, some of which overlap with common definitions of "finite state machines". To fix the discussion, here I adopt the following definition:

Definition 1. *A **finite automaton** (FA) is a 5-tuple $(R, \Lambda, r^{\varnothing}, r^{A}, \rho)$ where:*

1. *R is a finite set of **computational states**;*
2. *Λ is a finite (input) **alphabet**;*
3. *$r^{\varnothing} \in R$ is the **start state**;*
4. *$r^{A} \in R$ is the **accept state**; and*
5. *$\rho : R \times \Lambda \to R$ is the **update function**, mapping a current input symbol and the current computational state to a next computational state.*

A finite string of successive input symbols, i.e., an element of Λ^*, is sometimes called an **(input) word**, written as $\vec{\lambda}$. To operate a finite automaton on a particular input word, one begins with the automaton in its start state, and feeds that state together with the first symbol in the input word into the update function, to produce a new computational state. Then one feeds in the next symbol

[4] The reader should be warned that computer scientists also consider "uniform circuit families", which is something that is related but different.

in the input word (if any), to produce a next computational state, and so on. I will sometimes say that the **head** is in some state $r \in R$, rather than say that the computational state of the automaton is r.

Often one is interested in whether the head is in state r^A after the last symbol from the input word is processed. If that is the case, one says that the automaton **accepts** that input word. In this way any given automaton uniquely specifies a **language** of all input words that that automaton accepts, which is called a **regular** language. As an example, any finite language (consisting of a finite set of words) is a regular language. On the other hand, the set of all palindromes over Λ, to give a simple example, is *not* a regular language.

Importantly, any particular FA can process input words of arbitrary length. This means that one cannot model a given FA as some specific (and therefore fixed width) circuit, in general. The FA will have properties that are not captured by that circuit. In this sense, individual FAs are computationally more powerful than individual circuits. (This does not mean that individual FAs are more powerful than entire circuit *families* however; see the discussion in Section IV E of how circuit families can be even more powerful than Turing machines.)

Finite automata play an important role in many different fields, including electrical engineering, linguistics, computer science, philosophy, biology, mathematics, and logic. In computer science specifically, they are widely used in the design of hardware digital systems, of compilers, of network protocols, and in the study of computation and languages more broadly.

In all these applications of FAs, the automaton is viewed as a system that maps an input *word* to an output *computational state*. This motivates the following alternative definition of an FA:

Definition 2. *A **word-based** finite automaton is a 6-tuple* $(R, \Lambda, r^{\varnothing}, r^A, \rho^*, \tau)$ *where:*

1. R *is a finite set of **computational states**;*
2. Λ *is a finite (input) **alphabet**;*
3. $r^{\varnothing} \in R$ *is the **start state**;*
4. $r^A \in R$ *is the **accept state**;*
5. $\tau \in \mathbb{Z}^+$ *is the **counter**; and*
6. $\hat{\rho} : R \times \Lambda^* \times \mathbb{Z}^+ \to R$ *is the **(computational state) update function**, mapping a current input word, current counter pointing to one of that word's symbols, and current computational state, to a next computational state.*

When I need to distinguish FAs as defined in Def. 1 from word-based FAs, I will refer to the former as **symbol-based** FAs.

To map a symbol-based finite automaton (Def. 1) with update function ρ into an equivalent word-based update function $\hat{\rho}$, we set

$$\hat{\rho}(r, \lambda^*, \tau) = \rho(r, \lambda^*_\tau) \tag{22}$$

At the first iteration of a word-based FA, not only is the computational state set to the start state, but the counter has the value 0. In addition, at the end of each iteration m of the FA, after its computational state is updated by $\hat{\rho}$, the counter is incremented, i.e., $\tau_m \rightarrow \tau_m + 1$. From now on I will implicitly move back and forth between the two definitions of an FA as is convenient.

To allow us to analyze a physical system that implements the running of an FA on many successive input words, we need a way to signal to the system when one input word ends and then another one begins. Accordingly, without loss of generality we assume that Λ contains a special **blank** state, written b, that delimits the end of a word. I write Λ_- for $\Lambda \setminus \{b\}$, so that words are elements of Λ^*_-.

In a **stochastic** finite automaton (sometimes called a "probabilistic automaton"), the single-valued function ρ is replaced by a conditional distribution. In order to use notation that covers all iterations i of a stochastic finite automaton, I write this **update distribution** as $\pi(r_{i+1}|r_i, \lambda_i)$. The analogous extension of the word-based definition of finite automata into a word-based definition of a stochastic finite automata is immediate. For simplicity, from now on I will simply refer to "finite automaton", using the acronym "FA", to refer to a finite automaton that is either stochastic or deterministic.

Typically in the literature there is a set of multiple accept states — called "terminal states", or "accepting states" — not just one. Sometimes there are also multiple start states.

D. Transducers - Moore machines and Mealy machines

In the literature the definition of FA is sometimes extended so that in each transition from one computational state to the next an output symbol is generated. Such systems are also called "transducers" in the computer science community.

Definition 3. *A **transducer** is a 6-tuple $(R, \Lambda, \Gamma, r^\varnothing, x^A, \rho)$ such that:*

*1. R is a finite set, the set of **computational states**;*

2. Λ *is a finite set, called the* ***input alphabet****;*

3. Γ *is a finite set, called the* ***output alphabet****;*

4. $r^{\varnothing} \in R$ *is the* ***start*** *state;*

5. $r^{A} \in R$ *is the* ***accept state****;*

6. $\rho : R \times \Lambda \to R \times \Gamma$ *is the* ***update rule****.*

Sometimes the computational states of a transducer are referred to as the states of its **head**. I refer to the (semifinite) string of symbols that have yet to be processed by a transducer at some moment as the **input (data) stream** at that moment. I refer to the string of symbols that have already been produced by the information ratchet at some moment as the **output (data) stream** at that moment.

To operate a transducer on a particular input data stream, one begins with the machine in its start state, and feeds that state together with the first symbol in the input stream into the update function, to produce a new computational state, and a new output symbol. Then one feeds in the next symbol in the input stream (if any), to produce a next computational state and associated output symbol, and so on.

In a **stochastic** transducer, the single-valued function ρ is replaced by a conditional distribution. In order to use notation that covers all iterations i of the transducer, I write this **update distribution** as $\pi(\gamma_{i+1}, r_{i+1}|r_i, \lambda_i)$. Stochastic transducers are used in fields ranging from linguistics to natural language processing (in particular machine translation) to machine learning more broadly. From now on I implicitly mean "stochastic transducer" when I use the term "transducer".[5]

As with FAs, typically in the literature transducers are defined to have a set of multiple accept states, not just one. Sometimes there are also multiple start states. Similarly, in some of the literature the transition function allows the transducer to receive the empty string as an input and/or produce the empty string as an output.

A **Moore machine** is a transducer where the output γ is determined purely by the current state of the transducer, r. In contrast, a transducer in which the output depends on both the current state x and the current input λ is called a **Mealy machine**.

[5]The reader should be warned that some of the literature refers to both FAs and transducers as "finite state machines", using the term "acceptor" or "recognizer" to refer to the system defined in Def. 1. Similarly, the word "transducer" is sometimes used loosely in the physics community, to apply to a specific system that transforms one variable — often energy — into another variable, or even just into another form.

As a final comment, an interesting variant of the transducers defined in Def. 3 arises if we remove the requirement that there be accept states (and maybe even remove the requirement of start states). In this variant, rather than feeding an infinite sequence of input words into the system, each of which results in its own output word, one feeds in a single input word, which is infinitely long, producing a single (infinitely long) output word. This variant is used to define so-called "automata groups" or "self-similar groups" [75].

Somewhat confusingly, although the computational properties of this variant of transducers differs in crucial ways from those defined in Def. 3, this variant is also called "transducers" in the literature on "computational mechanics", a branch of hidden Markov model theory [76]. Fortunately, this same variant have also been given a *different* name, **information ratchets**, in work analyzing their statistical physics properties [13]. Accordingly, here I adopt that term for this variant of (computer science) transducers.

E. Turing machines

Perhaps the most famous class of computational machines are Turing machines [8–10]. One reason for their fame is that it seems one can model any computational machine that is constructable by humans as a Turing machine. A bit more formally, the *Church-Turing thesis* states that, "A function on the natural numbers is computable by a human being following an algorithm, ignoring resource limitations, if and only if it is computable by a Turing machine." The "physical Church-Turing thesis" modifies that to hypothesize that the set of functions computable with Turing machines includes all functions that are computable using mechanical algorithmic procedures admissible by the laws of physics [29, 30, 77, 78].

In part due to this thesis, Turing machines form one of the keystones of the entire field of computer science theory, and in particular of computational complexity [7]. For example, the famous Clay prize question of whether $\mathsf{P} = \mathsf{NP}$ — widely considered one of the deepest and most profound open questions in mathematics — concerns the properties of Turing machines. As another example, the theory of Turing machines is intimately related to deep results on the limitations of mathematics, like Gödel's incompleteness theorems, and seems to have broader implications for other parts of philosophy as well [79]. Indeed, it has been argued that the foundations of physics may be restricted by some of the properties of Turing machines [28, 80].

Along these lines, some authors have suggested that the foundations of statistical physics should be modified to account for the properties of Turing machines, e.g., by adding terms to the definition

of entropy. After all, given the Church-Turing hypothesis, one might argue that the probability distributions at the heart of statistical physics are distributions "stored in the mind" of the human being analyzing a given statistical physical system (i.e., of a human being running a particular algorithm to compute a property of a given system). Accordingly, so goes the argument, the costs of generating, storing, and transforming the minimal specifications of the distributions concerning a statistical physics system should be included in one's thermodynamic analysis of those changes in the distribution of states of the system. See [81–83].

There are many different definitions of Turing machines that are "computationally equivalent" to one another. This means that any computation that can be done with one type of Turing machine can be done with the other. It also means that the "scaling function" of one type of Turing machine, mapping the size of a computation to the minimal amount of resources needed to perform that computation by that type of Turing machine, is at most a polynomial function of the scaling function of any other type of Turing machine. (See for example the relation between the scaling functions of single-tape and multi-tape Turing machines [8].) The following definition will be useful for our purposes, even though it is more complicated than strictly needed:

Definition 4. *A **Turing machine** (TM) is a 7-tuple $(R, \Lambda, b, v, r^{\varnothing}, r^{A}, \rho)$ where:*

1. *R is a finite set of **computational states**;*
2. *Λ is a finite **alphabet** containing at least three symbols;*
3. *$b \in \Lambda$ is a special **blank** symbol;*
4. *$v \in \mathbb{Z}$ is a **pointer**;*
5. *$r^{\varnothing} \in R$ is the **start state**;*
6. *$r^{A} \in R$ is the **accept state**; and*
7. *$\rho : R \times \mathbb{Z} \times \Lambda^{\infty} \to R \times \mathbb{Z} \times \Lambda^{\infty}$ is the **update function**. It is required that for all triples (r, v, T), that if we write $(r', v', T') = \rho(r, v, T)$, then v' does not differ by more than 1 from v, and the vector T' is identical to the vectors T for all components with the possible exception of the component with index v;*[6]

[6] Technically the update function only needs to be defined on the "finitary" subset of $\mathbb{R} \times \mathbb{Z} \times \Lambda^{\infty}$, namely, those elements of $\mathbb{R} \times \mathbb{Z} \times \Lambda^{\infty}$ for which the tape contents has a non-blank value in only finitely many positions.

r^A is often called the "halt state" of the TM rather than the accept state. (In some alternative, computationally equivalent definitions of TMs, there is a set of multiple accept states rather than a single accept state, but for simplicity I do not consider them here.) ρ is sometimes called the "transition function" of the TM. We sometimes refer to R as the states of the "head" of the TM, and refer to the third argument of ρ as a **tape**, writing a value of the tape (i.e., semi-infinite string of elements of the alphabet) as T. The set of triples that are possible arguments to the update function of a given TM are sometimes called the set of **instantaneous descriptions** (IDs) of the TM. (These are sometimes instead referred to as "configurations".) Note that as an alternative to Def. 4, we could define the update function of any TM as a map over an associated space of IDs.

Any TM $(R, \Sigma, b, v, \rho, r^\varnothing, \rho)$ starts with $r = r^\varnothing$, the counter set to a specific initial value (e.g, 0), and with T consisting of a finite contiguous set of non-blank symbols, with all other symbols equal to b. The TM operates by iteratively applying ρ, until the computational state falls in r^A, at which time it stops. (Note that the definition of ρ for $r = r^A$ is arbitrary and irrelevant.)

If running a TM on a given initial state of the tape results in the TM eventually halting, the state of T when it halts is called the TM's **output**. The initial state of T (excluding the blanks) is sometimes called the associated **input**, or **program**. (However, the reader should be warned that the term "program" has been used by some physicists to mean specifically the shortest input to a TM that results in it computing a given output.) We also say that the TM **computes** an output from an input. In general, there will be inputs for which the TM never halts. The set of all those inputs to a TM that cause it to eventually halt is called its **halting set**.

As mentioned, there are many variants of the definition of TMs provided above. In one particularly popular variant the single tape in Definition 4 is replaced by multiple tapes. Typically one of those tapes contains the input, one contains the TM's output (if and) when the TM halts, and there are one or more intermediate "work tapes" that are in essence used as scratch pads. The advantage of using this more complicated variant of TMs is that it is often easier to prove theorems for such machines than for single-tape TMs. However, there is no difference in their computational power. More precisely, one can transform any single-tape TM into an equivalent multi-tape TM (i.e., one that computes the same partial function), as well as vice-versa [8, 11, 84].

Returning to the TM variant defined in Definition 4, a **universal Turing machine** (UTM), M, is one that can be used to emulate any other TM. More precisely, a UTM M has the property that for any other TM M', there is an invertible map f from the set of possible states of the tape of M' into the set of possible states of the tape of M, such that if we:

1. apply f to an input string σ' of M' to fix an input string σ of M;

2. run M on σ until it halts;

3. apply f^{-1} to the resultant output of M;

then we get exactly the output computed by M' if it is run directly on σ'.

An important theorem of computer science is that there exists universal TMs. Intuitively, this just means that there exists programming languages which are "universal", in that we can use them to implement any desired program in any other language, after appropriate translation of that program from that other language. This universality leads to a very important concept:

Definition 5. *The **Kolmogorov complexity** of a UTM M to compute a string $\sigma \in \Lambda^*$ is the length of the shortest input string s such that M computes σ from s.*

Intuitively, (output) strings that have low Kolmogorov complexity for some specific UTM M are those with short, simple programs in the language of M. For example, in all common (universal) programming languages (e.g., *C, Python, Java*, etc.), the first m digits of π have low Kolmogorov complexity, since those digits can be generated using a relatively short program. Strings that have high (Kolmogorov) complexity are sometimes referred to as "incompressible". These strings have no patterns in them that can be generated by a simple program. As a result, it is often argued that the expression "random string" should only be used for strings that are incompressible.

It is common to refine the definition of "halting set" implicitly, to include only those input strings such that the TM halts *with its head positioned on the last symbol in that input string*. Use this refined definition from on. Suppose we have two strings s^1 and s^2 where s^1 is a proper prefix of s^2. If we run the TM on s^1, it can detect when it gets to the end of its input, by noting that the following symbol on the tape is a blank. Therefore, it can behave differently after having reached the end of s^1 from how it behaves when it reaches the end of the first $\ell(s^1)$ bits in s^2. As a result, it may be that both of those input strings are in its halting set, but result in different outputs.

A **prefix (free) TM** is one in which this can never happen: there is no string in its halting set that is a proper prefix of another string in its halting set.[7]

[7] It is not trivial to construct prefix single-tape TMs directly. For that reason it is common to use prefix three-tape TMs, in which there is a separate input tape that can only be read from, output tape that can only be written to, and work tape that can be both read from and written to. To ensure that the TM is prefix, we require that the head cannot ever back up on the input tape to reread earlier input bits, nor can it ever back up on the output tape, to overwrite earlier output bits. To construct a single-tape prefix TM, we can start with some such three-tape prefix TM and transform it into an equivalent single-tape prefix TM, using any of the conventional techniques for transforming between single-tape and multi-tape TMs.

We can use the Kolmogorov complexity of prefix TMs to define many associated quantities, which are related to one another the same way that various kinds of Shannon entropy are related to one another. For example, loosely speaking, the conditional Kolmogorov complexity of string s conditioned on string s', written as $K(s|s')$, is the length of the shortest string x such that if the TM starts with an input string given by the concatenation xs', then it computes s and halts. If we restrict attention to prefix-free TMs, then for all strings $x, y \in \Lambda^*$, we have [84]

$$K(x, y) \leq K(x) + K(x|y) + O(1) \leq K(x) + K(y) + O(1) \tag{23}$$

(where "$O(1)$" means a term that is independent of both x and y). Indeed, in a certain technical sense, the expected value of $K(x)$ under any distribution $P(x \in \Lambda^*)$ equals the Shannon entropy of P. (See [84].)

The **coin-flipping prior** of a prefix TM M is the probability distribution over the strings in M's halting set generated by IID "tossing a coin" to generate those strings, in a Bernoulli process, and then normalizing.[8] So any string σ in the halting set has probability $2^{-|\sigma|}/\Omega$ under the coin-flipping prior, where Ω is the normalization constant for the TM in question.

The coin-flipping prior provides a simple Bayesian interpretation of Kolmogorov complexity: Under that prior, the Kolmogorov complexity of any string σ for any prefix TM M is just (the log of) the maximum a posterior (MAP) probability that any string σ' in the halting set of M was the *input* to M, conditioned on σ being the *output* of that TM. (Strictly speaking, this result is only true up to an additive constant, given by the log of the normalization constant of the coin-flipping prior for M.)

The normalization constant Ω for any fixed prefix UTM, sometimes called "Chaitin's Omega", has some extraordinary properties. For example, the successive digits of Ω provide the answers to *all* well-posed mathematical problems. So if we knew Chaitin's Omega for some particular prefix UTM, we could answer every problem in mathematics. Alas, the value of Ω for any prefix UTM M cannot be computed by any TM (either M or some other one). So under the Church-Turing hypothesis, we cannot calculate Ω. (See also [20] for a discussion of a "statistical physics" interpretation of Ω that results if we view the coin-flipping prior as a Boltzmann distribution for an appropriate Hamiltonian, so that Ω plays the role of a partition function.)

It is now conventional to analyze Kolmogorov complexity using prefix UTMs, with the coin-flipping prior, since this removes some undesirable technical properties that Kolmogorov complexity

[8] Kraft's inequality guarantees that since the set of strings in the halting set is a prefix-free set, the sum over all its elements of their probabilities cannot exceed 1, and so it can be normalized. However, in general that normalization constant is uncomputable, as discussed below. See [84].

has for more general TMs and priors. Reflecting this, all analyses in the physics community that concern TMs assume prefix UTMs. (See [84] for a discussion of the extraordinary properties of such UTMs.)

Interestingly, for all their computational power, there are some surprising ways in which TMs are *weaker* than the other computational machines introduced above. For example, there are an infinite number of TMs that are more powerful than any given circuit, i.e., given any circuit C, there are an infinite number of TMs that compute the same function as C. Indeed, any single UTM is more powerful than *every* circuit in this sense. On the other hand, it turns out that there are circuit *families* that are more powerful than any single TM. In particular, there are circuit families that can solve the halting problem [8].

Finally, some terminological comments and definitions will be useful below. It is conventional when dealing with Turing machines to implicitly assume some invertible map $R(.)$ from $\mathbb{Z}$ into Λ^*. Given such a map $R(.)$, we can exploit it to implicitly assume an additional invertible map taking $\mathbb{Q}$ into Λ, e.g., by uniquely expressing any rational number as one product of primes, a, divided by a product of different primes, b; invertibly mapping those two products of primes into the single integer $2^a 3^b$; and then evaluating $R(2^a 3^b)$.

If a function from Λ^* into itself is defined for all inputs, it is called a **total function**; otherwise it is a **partial function**. If the input-output pairs defining a particular partial function are exactly the pairs of inputs and associated outputs produced by some TM, that function is called a **partial recursive** function. (so the TM in question must never halt for all inputs on which the partial function is undefined.) Similarly, a total function that is computed by some TM which always halts is called a **total recursive** function, or just called a **recursive** function for short. In addition, a real number z is **computable** iff there is a recursive function f mapping rational numbers to rational numbers such that for all rational-valued accuracies $\epsilon > 0$, $|f(\epsilon) - z| < \epsilon$.

V. ENTROPY DYNAMICS

This section reviews those aspects of stochastic thermodynamics that are necessary to analyze the dynamics of various types of entropy during the evolution of computational machines. As illustrated with examples, the familiar quantities at the heart of thermodynamics (e.g., heat, thermodynamic entropy, work) arise in special cases of this analysis.

In the first subsection, I review the conventional decomposition of the entropy flow (EF) out of a physical system into the change in entropy of that system plus the (irreversible) entropy creation

(EP) produced as that system evolves [85, 86]. To fix details, I will concentrate on the total amount of EF, EP and entropy change that arise over a time-interval $[0, 1]$.[9]

In the second subsection, I review recent results [69] that specify how the EP generated by the evolution of some system depends on the initial distribution of states of the system. These recent results allow us to evaluate how the EF of an arbitrary system, whose dynamics implements some conditional distribution π of final states given initial states, depends on the initial distribution of states of the system that evolves according to π. (As elaborated in subsequent sections, this dependence is one of the central features determining the entropic costs of running any computational machine.)

I end this section with some general cautions about translating a computer science definition of a computational machine into a physics definition of a system that implements that machine.

A. Entropy flow, entropy production, and Landauer cost

To make explicit connection with thermodynamics, consider a physical system with countable state space X that evolves over time interval $t \in [0, 1]$ while in contact with one or more thermal reservoirs, while possibly also undergoing driving by one or more work reservoirs.[10] In this chapter I focus on the scenario where the system dynamics over the time interval is governed by a continuous-time Markov chain (CTMC). However many of the results presented below are more general.

Let $W_{x;x'}(t)$ be the rate matrix of the CTMC. So the probability that the system is in state x at time t evolves according to the linear, time-dependent equation

$$\frac{d}{dt} p_x(t) = \sum_{x'} W_{x;x'}(t) p_{x'}(t) \tag{24}$$

which I can write in vector form as $\dot{p}(t) = W(t)p(t)$. I just write "W" to refer to the entire time-history of the rate matrix. W and $p(0)$ jointly fix the conditional distribution of the system's state at $t = 1$ given its state at $t = 0$, which I write as $\pi(x_1|x_0)$. Note that in general no finite rate matrix can implement a map π that is a single-valued function. However, we can always implement such a function to any desired finite accuracy by appropriate construction of the rate matrix [87].

[9] In this paper I will not specify units of time, and often implicitly change them. For example, when analyzing the entropy dynamics of a given circuit, sometimes the time interval $[0, 1]$ will refer to the time to run the entire circuit, and the attendant entropic costs. However at other times $[0, 1]$ will refer to the time to run a single gate within that circuit, and the entropic costs of running just that gate. In addition, for computational machines that take more than one iteration to run, I will usually just refer to a "time interval $[0, 1]$" without specifying which iteration of the machine that interval corresponds to. Always the context will make the meaning clear.

[10] In statistical physics, a "reservoir" R in contact with a system S is loosely taken to mean an infinitely large system that interacts with S on time scales infinitely faster than the explicitly modeled dynamical evolution of the state of S. For example, a "particle reservoir" exchanges particles with the system, a "thermal reservoir" exchanges heat, and a "work reservoir" is an external system that changes the energy spectrum of the system S.

Accordingly, throughout this paper I will refer to CTMCs implementing a single-valued function, when what I really mean is that it implements that function up to any desired finite accuracy.

As shorthand, I sometimes abbreviate $x(0)$ as x, and sometimes abbreviate the initial distribution $p(0)$ as p. (So for example, πp is the ending distribution over states.) I will also sometimes abbreviate $p(1)$ as p', and $x(1)$ as x'; the context should always make the meaning clear.

Next, define the **entropy flow (rate)** at time t as

$$\sum_{x';x''} W_{x';x''}(t) p_{x''}(t) \ln \left[\frac{W_{x';x''}}{W_{x'';x'}} \right] \tag{25}$$

Physically, this corresponds to an entropy flow rate out of the system, into reservoirs it is coupled to.

In order to define an associated total amount of entropy flow during a non-infinitesimal time interval (EF), define $\boldsymbol{x} = (N, \vec{x}, \vec{\tau})$ as a trajectory of $N+1$ successive states $\vec{x} = (x(0), x(1), \ldots, x(N))$, along with times $\vec{\tau} = (\tau_0 = 0, \tau_1, \tau_2, \ldots, \tau_{N-1})$ of the associated state transitions, where $\tau_{N-1} \leq 1$ (the time of the end of the process), $x(0)$ is the beginning, $t = 0$ state of the system, and $x(N)$ is the ending, $t = 1$ state of the system. Then under the dynamics of Eq. (24), the probability of any particular trajectory is [86, 88, 89]

$$p(\boldsymbol{x}|x(0)) = \left(\prod_{i=1}^{N-1} S^{\tau_i}_{\tau_{i-1}}(x(i-1)) W_{x(i),x(i-1)}(\tau_i) \right) S^1_{\tau_{N-1}}(x_N) \tag{26}$$

where $S^{\tau'}_{\tau}(x) = e^{\int_{\tau}^{\tau'} W_{x,x}(t)dt}$ is the "survival probability" of remaining in state x throughout the interval $t \in [\tau, \tau']$. The total EF out of the system during the interval can be written as an integral weighted by these probabilities:

$$\mathcal{Q}(p_0) = \int p_0(x_0) p(\boldsymbol{x}|x(0)) \sum_{i=1}^{N-1} W_{x(i),x(i-1)}(\tau_i) \ln \frac{W_{x(i),x(i-1)}(\tau_i)}{W_{x(i-1),x(i)}(\tau_i)} \, D\boldsymbol{x} \tag{27}$$

(Note that I use the convention that EF reflects total entropy flow *out* of the system, whereas much of the literature defines EF as the entropy flow *into* the system.)

EF will be the central concern in the analysis below. By plugging in the evolution equation for a CTMC, we can decompose EF as the sum of two terms. The first is just the change in entropy of the system during the time interval. The second, is the **(total) entropy production** (EP) in the system during the process [86, 90, 91]. I write EP as $\sigma(p)$. It is the integral over the interval of the instantaneous EP rate,

$$\sum_{x';x''} W_{x';x''}(t) p_{x''}(t) \ln \left[\frac{W_{x';x''} p_{x''}(t)}{W_{x'';x'} p_{x'}(t)} \right] \tag{28}$$

I will use the expressions "EF incurred by running a process", "EF to run a process", or "EF generated by a process" interchangeably, and similarly for EP.[11] EF can be positive or negative. However, for any CTMC, the EP rate given in Eq. (28) is non-negative [86, 90], and therefore so is the EP generated by the process. So

$$\mathcal{Q}(p_0) = \sigma(p_0) + S(p_0) - S(\pi p_0) \tag{29}$$

$$\geq S(p_0) - S(\pi p_0) \tag{30}$$

where throughout this paper, π refers to the conditional distribution of the state of the system at $t = 1$ given its state at $t = 0$, which is implicitly fixed by $W(t)$.

Total entropy flow across a time interval can be written as a linear function of the initial distribution:

$$\mathcal{Q}(p_0) = \sum_{x_0} \mathcal{F}(x_0) p_0(x_0) \tag{31}$$

for a function $\mathcal{F}(x)$ that depends on the entire function $W_{x;x'}(t)$ for all $t \in [0, 1)$, and so is related to the discrete time dynamics of the entire process, $\pi(x_1|x_0)$. (See Eq. (27).) However, the *minimal* entropy flow for a fixed transition matrix π occurring when EP is zero, is the drop in entropy from $S(p_0)$ to $S(\pi P_0)$. This is not a linear function of the initial distribution p_0. In addition, the entropy production — the difference between actual entropy flow and minimal entropy flow — is not a linear function of p_0. (So the two nonlinearities "cancel out" when they are added, to give a linearity.) These nonlinearities are the basis of much of the richness of statistical physics, and in particular of its relation with information theory.

There are no temperatures in any of this analysis. Indeed, in this very general setting, temperatures need not even be defined. However, suppose that exists there exists an "energy function" $E : X \to \mathbb{R}$ such that for all $x, x' \in X$,

$$\ln \left[\frac{W_{x;x'}}{W_{x';x}} \right] = E(x') - E(x) \tag{32}$$

Then we can interpret the dynamics of the system at t as though the system were coupled to a heat bath with a well-defined temperature T, and the energy function obeyed "detailed balance" (DB) with that heat bath. (We also say that such a rate matrix is "reversible" [92].) In this case EF can be written as [88]

$$\mathcal{Q} = k_B T^{-1} Q \tag{33}$$

[11] Confusingly, sometimes in the literature the term "dissipation" is used to refer to EP, and sometimes it is used to refer to EF. Similarly, sometimes EP is instead referred to as "irreversible EP", to contrast it with any change in the entropy of the system that arises due to entropy flow.

where k_B is Boltzmann constant, and Q is the expected amount of heat transfered from the system into bath ν during the course of the process. In addition, if DB holds for a system at time t for a heat bath with temperature T, then we say that the system is **at (thermal) equilibrium** at that time if

$$p_t(x) \propto e^{-E(x)/k_B T} \tag{34}$$

This is just the familiar canonical ensemble from equilibrium statistical physics [93]. At thermal equilibrium, the Shannon entropy analyzed in this paper is identical to thermodynamic entropy.

Example 1. *Consider the special case of an* isothermal *process, meaning there is a single heat bath at temperature T (although possibly one or more work reservoirs and particle reservoirs). Suppose that the process transforms an initial distribution p and Hamiltonian H into a final distribution p' and Hamiltonian H'.*

In this scenario, EF equals $(k_B T)^{-1}$ times the total heat flow into the bath. EP, on the other hand, equals $(k_B T)^{-1}$ times the dissipated work *of the process, which is the work done on the system over and above the minimal work required by any isothermal process that performs the transformation $(p, H) \mapsto (p', H')$ [6]. So by Eq.* (30) *and energy conservation, the minimal work is the change in the expected energy of the system plus ($k_B T$ times) the drop in Shannon entropy of the system. This is just the change in the* nonequilibrium free energy *of the system from the beginning to the end of the process [6, 94, 95].*

There are many different physical phenomena that can result in nonzero EP. One broad class of such phenomena arises if we take an "inclusive" approach, modeling the dynamics of the system and bath together:

Example 2. *Continuing with the special case of an isothermal process, suppose that the heat bath never produces any entropy by itself, i.e., that the change in the entropy of the bath equals the EF from the system into the bath. Then the change in the sum, {marginal entropy of the system} + {marginal entropy of the heat bath} must equal the EF from the system to the bath plus the change in the marginal entropy of the system by itself. By Eq.* (29) *though, this is just the EP of the system*

On the other hand, Liouville's theorem tells us that the joint entropy of the system and the bath is constant. Combining establishes that EP of the system equals the change in the difference between the joint entropy and the sum of the marginal entropies, i.e., EP equals the change in the mutual information between the system and the bath.

To illustrate this, suppose we start with system and bath statistically independent. So the mutual information between them originally equals zero. Since mutual information cannot be negative, the change of that mutual information during the process is non-negative. This confirms that EP is non-negative, for this particular case where we start with no statistical dependence between the system and the bath. See [96].

Variants of Eq. (30) are sometimes referred to in the literature as the **generalized Landauer's bound**. To motivate this name, suppose that there is a single heat bath, at temperature T, and that the system has two possible states, $X = \{0, 1\}$. Suppose further that the initial distribution $p(x)$ is uniform over these two states, and that the conditional distribution π implements the function $\{0, 1\} \mapsto 0$, i.e., it is a 2-to-1 'bit-erasure' map. So by Eq. (33) and the non-negativity of EP, the minimal heat flow out of the system accompanying any process that performs that bit erasure is $k_B T(\ln[2] - \ln 1) = k_B T \ln[2]$, in accord with the bound proposed by Landauer.

Note though that in contrast to the bound proposed by Landauer, the generalized Landauer's bound holds for systems with an arbitrary number of states, an arbitrary initial distribution over their states, and an arbitrary conditional distribution π. Most strikingly, the generalized Landauer bound holds even if the system is coupled to multiple thermal reservoirs, all at different temperatures, e.g., in a steady state heat engine [97, 98] (see Ex. 5 below). In such a case $k_B T \ln 2$ is not defined. Indeed, the generalized Landauer bound holds even if the system does not obey detailed balance with any of the one or more reservoirs it's coupled to.

Motivated by the generalized Landauer's bound, we define the **(unconstrained) Landauer cost** as the minimal EF required to compute π on initial distribution p using *any* process, with no constraints:

$$\mathcal{L}(p, \pi) := S(p) - S(\pi p)\,. \tag{35}$$

With this definition we can write

$$\mathcal{Q}(p) = \mathcal{L}(p, \pi) + \sigma(p) \tag{36}$$

Example 3. *Landauer's bound is often stated in terms of the minimal amount of* work *that must be done in order to perform a given computation, rather than the* heat *that must be generated. This is appropriate for physical processes that both begin and end with a constant, state-independent energy function. For such processes, there cannot be any change in expected energy between the beginning and end of the process. Moreover, by the first Law of thermodynamics,*

$$\Delta E = W - \mathcal{Q}(p)$$

where ΔE is the change in expected energy from the beginning and end of the process, W is work incurred by the process, and as before, $\mathcal{Q}(p)$ is the expected amount of heat that leaves the system and enters the bath. Since $\Delta E = 0$, $W = Q$. So the bounds in Example 1 on the minimal heat that must flow out of the system also give the minimal work that must be done on the system.

Any process which achieves $\sigma = 0$ (i.e., the generalized Landauer's bound) for some particular initial distribution p is said to be **thermodynamically reversible** when run on that distribution. (For simplicity, I use this terminology even if there are no heat baths connected to the system, so that we cannot interpret entropic costs as thermodynamic quantities.) In the special case that the system is coupled to a single heat bath and obeys DB, for its dynamics over the interval [01] to be thermodynamically reversible the system must be at equilibrium with that heat bath at all $t \in [0, 1]$.[12]

A necessary condition for a CTMC to be thermodynamically reversible when run on some q_0 is that if we run it forward on an initial distribution q_0 to produce q_1, and then "run the process backward", by changing the signs of all momenta and reversing the time-sequence of any driving by work reservoirs, we return to q_0. (See [85, 99–101].) Moreover, it has recently been proven that for any π and initial distribution q_0, we can always design a CTMC that implements π on any initial distribution, and in addition is thermodynamically reversible if the initial distribution is q_0. (See [87] for a proof based on stochastic thermodynamics, [62, 102–106] for earlier, less detailed analyses based on general nonequilibrium statistical physics, and Section X A and Section XV below for general discussion.)

Example 4. *Suppose we design a CTMC to implement bit erasure and to be thermodynamically reversible if run on some initial distribution q_0 [107]. So if we run the bit erasure process backwards from the ending (delta function) distribution, we have to arrive back at the initial distribution q_0. This means that if we run that bit-erasure process on any initial distribution $p_0 \neq q_0$ and then run it backwards, we would not arrive back at p_0 (we arrive at q_0 instead). This proves that the bit-erasure process cannot be thermodynamically reversible when run on any such $p_0 \neq q_0$. This phenomenon is analyzed in the next subsection, and bit erasure is discussed more broadly in Section VI.*

[12] This way of characterizing thermodynamic reversibility has been central to statistical physics since it was invented in the 19th century, at least in the context of systems that are always at thermal equilibrium.

B. Mismatch cost and residual EP

Computational machines are built of multiple interconnected computational devices. A crucial concern in calculating the entropic costs of running such a computational machine is how the costs incurred by running any of its component devices, implementing some distribution π, depends on the distribution over the inputs to that device, p_0.

For a fixed π, we can write Landauer cost of any process that implements π as a single-valued function of the initial distribution p_0; no properties of the rate matrix W matter for calculating Landauer cost, beyond the fact that that matrix implements π. However, even if we fix π, we cannot write EP as a single-valued function of p_0, because EP *does* depend on the details of how W implements π. (Intuitively, it is the EP, not the Landauer cost, that reflects the "nitty gritty details" of the the dynamics of the rate matrix implementing the computation.) In this subsection I review recent results establishing precisely how W determines the dependence of EP on p_0.

It has long been known how the entropy production *rate*, at a single moment t, jointly depends on the rate matrix $W(t)$ and on the distribution over states p_t. (In fact, those dependencies are given by the expression in Eq. (28).) On the other hand, until recently nothing was known about how the EP of a discrete time process, evolving over an extended time interval, depends on the initial distribution over states. Initial progress was made in [108], in which the dependence of EP on the initial distribution was derived for the special case where $\pi(x_1|x_0)$ is nonzero for all x_0, x_1. However, this restriction on the form of π is violated in deterministic computations.

Motivated by this difficulty, [69] extended the earlier work in [108], to give the full dependence of EP on the initial distribution for arbitrary π. That extended analysis shows that EP can always be written as a sum of two terms. Each of those terms depends on p_0, as well as on the "nitty gritty details" of the process, embodied in $W(t)$.

The first of those EP terms depends on p_0 linearly. By appropriately constructing the nitty gritty details of the system (e.g., by having the system implementing π run a quasi-static process), it is possible to have this first term equal zero identically, for all p_0. The second of the EP terms instead is given by a drop in the KL divergence between p and a distribution q that is specified by the nitty gritty details, during the time interval $t \in [0, 1]$. For nontrivial distributions π, this term can*not* be made to equal zero for any distribution p_0 that differs from q_0. This is unavoidable EP incurred in running the system, which arises whenever one changes the input distribution to a different device from the optimal one, without modifying the device itself.

To review these recent results, recall the definition of islands c and associated distributions Δ_c

from Section II. Next make the following definition:

Definition 6. *For any conditional distribution π implemented by a CTMC, and any island $c \in L(\pi)$, the associated* ***prior*** *is*

$$q^c \in \underset{r:\text{supp}(r)\in\Delta_c}{\arg\min}\ \sigma(r)$$

We write the associated lower bound on EP as

$$\sigma^{min}(c) := \min_{r:\text{supp}(r)\in\Delta_c} \sigma(r)$$

It will simplify the exposition to introduce an arbitrary distribution over islands, $q(c)$, and define

$$q(x) := \sum_{c\in L(\pi)} q(c) q^c(x)$$

In [69] it is shown that

$$\sigma(p) = D(p\|q) - D(\pi p\|\pi q) + \sum_{c\in L(\pi)} p(c)\sigma^{min}(c) \tag{37}$$

(Due to the definition of islands, while the choice of distribution $q(c)$ affects the precise distribution q inside the two KL divergences, it has no effect on their difference; see [69].)

The drop of KL divergences on the RHS of Eq. (37) is called the the **mismatch cost** of running the CTMC on the initial distribution p, and is written as $\mathcal{E}(p)$.[13] Given the priors q^c, both of these KL divergences depend only on p and on π; no attributes of W beyond those that implicitly set the priors q^c matter for mismatch cost. By the data-processing inequality for KL divergence, mismatch cost is non-negative. It equals zero if $p = q$ or if π is a measure-preserving map, i.e., a permutation of the elements of X.

The remaining sum on the RHS of Eq. (37) is called the **residual EP** of the CTMC. It is a linear function of $p(c)$, without any information theoretic character. In addition, it has no explicit dependence on π. It is (the $p(c)$-weighted average of) the minimal EP within each island. $\sigma^{min}(c) \geq 0$ for all c, and residual EP equals zero if and only if the process is thermodynamically reversible. I will refer to $\sigma^{min}(c)$ as the **residual EP (parameter) vector** of the process. The "nitty-gritty" physics details of how the process operates is captured by the residual EP vector together with the priors.

Combining Eq. (37) with the definitions of EF and of cross entropy establishes the following set of equivalent ways of expressing the EF:

[13] In [108], due to a Bayesian interpretation of q, the mismatch cost is instead called the "dissipation due to incorrect priors".

Proposition 1. *The total EF incurred in running a process that applies map π to an initial distribution p is*

$$\begin{aligned}\mathcal{Q}(p) &= \mathcal{L}(p,\pi) + \mathcal{E}(p) + \sum_{c \in L(\pi)} p(c)\sigma^{min}(c) \\ &= [S(p\|q) - S(\pi p\|\pi q)] + \sum_c p(c)\sigma^{min}(c)\end{aligned}$$

Unlike the generalized Landauer's bound, which is an inequality, Prop. 1 is exact. It holds for both macroscopic and microscopic systems, whether they are computational devices or not.

I will use the term **entropic cost** to broadly refer to entropy flow, entropy production, mismatch cost, residual entropy, or Landauer cost. Note that the entropic cost of any computational device is only properly defined if we have fixed the distribution over possible inputs of the device.

It is important to realize that we can*not* ignore the residual EP when calculating EF of real-world computational devices. In particular, in real-world computers — even real-world quantum computers — a sizable portion of the heat generation occurs in the wires connecting the devices inside the computer (often a majority of the heat generation, in fact). However, wires are designed to simply copy their inputs to their outputs, which is a logically invertible map. As a result, the Landauer cost of running a wire is zero (to within the accuracy of the wire's implementing the copy operation with zero error), no matter what the initial distribution over states of the wire p_0 is. For the same reason, the mismatch cost of any wire is zero. This means that the entire EF incurred by running any wire is just the residual EP incurred by running that wire. So in real-world wires, in which $\sigma^{min}(c)$ invariably varies with c (i.e., in which the heat generated by using the wire depends on whether it transmits a 0 or a 1), the dependence of EF on the initial distribution p_0 must be linear. In contrast, for the other devices in a computer (e.g., the digital gates in the computer), both Landauer cost and mismatch cost can be quite large, resulting in nonlinear dependencies on the initial distribution.

Example 5. *It is common in the literature to decompose the rate matrix into a sum of rate matrices of one or more* ***mechanisms*** *v:*

$$W_{x;x'}(t) = \sum_\nu W^v_{x;x'}(t) \tag{38}$$

In such cases one replaces the definitions of the EF rate and EP rate in Eq. (25),(28)*, with the similar definitions,*

$$\sum_{x';x'',\nu} W^\nu_{x';x''}(t) p_{x''}(t) \ln \left[\frac{W^\nu_{x';x''}}{W^\nu_{x'';x'}}\right] \tag{39}$$

and

$$\sum_{x';x'',\nu} W^{\nu}_{x';x''}(t) p_{x''}(t) \ln \left[\frac{W^{\nu}_{x';x''} p_{x''}(t)}{W^{\nu}_{x'';x'} p'_{x}(t)} \right] \tag{40}$$

respectively.

When there is more than one mechanism, since the log of a sum is not the same as the sum of a log, these redefined EF and EP rates differ from the analogous quantities given by plugging $\sum_{\nu} W^{v}_{x;x'}(t)$ *into Eq.* (25),(28). *For example, if we were to evaluate Eq.* (25) *for this multiple-mechanism* $W(t)$*, we would get*

$$\sum_{x';x'',\nu} W^{\nu}_{x';x''}(t) p_{x''}(t) \ln \left[\frac{\sum_{\nu'} W^{\nu'}_{x';x''}}{\sum_{\nu''} W^{\nu''}_{x'';x'}} \right] \tag{41}$$

which differs from the expression in Eq. (39).

Nonetheless, all the results presented above apply just as well with these redefinitions of EF and EP. In particular, under these redefinitions the time-derivative of the entropy still equals the difference between the EP rate and the EF rate, total EP is still non-negative, and total EF is still a linear function of the initial distribution. Moreover, that linearity of EF means that with this redefinition we can still write (total) EP as a sum of the mismatch cost, defined in terms of a prior, and a residual EP that is a linear function of the initial distribution.

By themselves, neither the pair of definitions in Eq. (25),(28) *nor the pair in Eq.* (39),(40) *is "right" or "wrong". Rather, the primary basis for choosing between them arises when we try to apply the resulting mathematics to analyze specific thermodynamic scenarios. The development starting from Eq.* (25),(28)*, for a single mechanism, can be interpreted as giving heat flow rates and work rates for the thermodynamic scenario of a single heat bath coupled to the system. (See Ex. 2 and 3 above.) However, in many thermodynamic scenarios there are multiple heat baths coupled to the system. The standard approach for analyzing these scenarios is to identify each heat bath with a separate mechanism, so that there is a separate temperature for each mechanism,* T^{ν}*. Typically one then assumes* ***local detailed balance*** *(LDB), meaning that separately for each mechanism* ν*, the associated matrix* $W^{\nu}(t)$ *obeys detailed balance for the (shared) Hamiltonian* $H(t)$ *and resultant (*ν*-specific) Boltzmann distribution defined in terms of the temperature* T^{ν}*, i.e., for all* ν, x, x', t*,*

$$\frac{W^{\nu}_{x;x'}(t)}{W^{\nu}_{x';x}(t)} = e^{[H_{x'}(t) - H_x(t)]/T^{\nu}} \tag{42}$$

This allows us to identify the EF rate in Eq. (39) *as the rate of heat flow to all of the baths. So the EP rate in Eq.* (40) *is the rate of irreversible gain in entropy that remains after accounting for that EF rate and for the change in entropy of the system. See [85***?** *, 86].*

It is important to emphasize that all of the analysis above assumes that there are no constraints on how the physical system can implement π. For example, the Landauer cost given in Eq. (35) and Proposition 1 is the unconstrained minimal amount of EF necessary to implement the conditional distribution π on any physical system, when there are no restrictions on the rate matrix underlying the dynamics of that system. However, in practice there will always be *some* constraints on what rate matrices the engineer of a system can use to implement a desired logical state dynamics. In particular, the architectures of the computational machines defined in Section IV constrain which variables in a system implementing those machines are allowed to be directly coupled with one another by the rate matrix. Such constraints can substantially increase the minimal feasible EF, as illustrated by the following example.

Example 6. *Suppose our computational machine's state space is two bits, x^1 and x^2, and that the function $f(x)$ erases both of those bits. Let $p_0(x)$ be the initial distribution over joint states of the two bits. (As a practical matter, p_0 would be determined by the preferences of the users of the system, e.g., as given by the frequency counts over a long time interval in which they repeatedly use the system.) In this scenario, the unconstrained Landauer cost is*

$$\begin{aligned} S(p_0(X)) - S(p_1(X)) &= S(p_0(X)) \\ &= S(p_0(X^1)) + S(p_0(X^2|X^1)) \end{aligned} \tag{43}$$

Now modify this scenario by supposing that we are constrained to implement the parallel bit erasure with two subsystems acting independently of one another, one subsystem acting on the first bit and one subsystem acting on the second bit. This changes the Landauer cost to

$$S(p_0(X^1)) - S(p_1(X^1)) + S(p_0(X^2)) - S(p_1(X^2) = S(p_0(X^1)) + S(p_0(X^2)) \tag{44}$$

The gain in Landauer cost due to the constraint is $S(p_0(X^2)) - S(p_0(X^2|X^1))$. This is just the mutual information between the two bits under the initial distribution p_0, which in general is nonzero.

To understand the implications of this phenomenon, suppose that the parallel bit erasing subsystems are thermodynamically reversible when considered by themselves. It is still the case that if they are run in parallel as two subsystems of an overall system, and if their initial states are statistically correlated, then that overall system is not thermodynamically reversible. Indeed, if we start with p_0, implement the parallel bit erasure using two thermodynamically reversible bit-erasers, and then run that process in reverse, we end up with the distribution $p_0(x^1)p_0(x^1)$ rather than $p_0(x^1, x^2)$.

This phenomenon is a key aspect of the thermodynamics of computation, coupling the global structure of a computational machine that implements some function π to the minimal EF one

could achieve with that machine by optimizing all of its components. It is the focus of Section IX below.

As a final comment, it is important to emphasize that all of the results in this section for mismatch costs and residual EP hold for processes that implement general stochastic matrices as well as those that implement single-valued functions, if one uses the appropriate generalization of islands.

Despite the importance of EP to the entropic costs of real world systems, after Section X I will assume that the residual EP of any island of any device I consider equals zero. The resultant analysis gives the least possible EF, which would arise if (as is almost always the case in the real world) we don't design the prior of a device to match the actual distribution of its input states, and so must account for its possible mismatch cost. In addition, this assumption simplifies the calculations, since we no longer have to consider the islands of the processes.

VI. LOGICAL VERSUS THERMODYNAMIC REVERSIBILITY

As mentioned in Section V, it is now understood that for any initial distribution over states, and any conditional distribution over those states, it is possibly to design a process that implements that conditional distribution and in addition is thermodynamically reversible if run on that specified initial distribution. In particular, this is true if the conditional distribution is a single-valued, logically irreversible map. As a concrete example, [109] shows how to build a dynamic process over the state of a (binary) quantum dot that implements (an arbitrarily accurate approximation of) bit erasure, while having arbitrarily small total EP, if the process is run on an (arbitrary) given initial distribution over states.

Ultimately, logical and thermodynamic reversibility are independent for the simple reason that they concern different properties of physically evolving systems. For a physical process to be logically reversible means two things. First, if the process is run up to time $t = 1$ after starting in some state x_0 at $t = 0$, it always executes some (continuous time) trajectory $x_{0...1}$ that ends at some specific state x_1 with probability 1, i.e., it is deterministic. Second, any ending state x_1 that arises from some initial state x_0 only arises for that initial state, i.e., the map is invertible over the states of the system.

However, as discussed at the end of Section V A, thermodynamic reversibility involves changes in marginal distributions, not changes in states. Moreover, for a process to be thermodynamically reversible does *not* mean it implements an invertible map over the space of all distributions. Indeed,

if a process implements a single-valued, non-invertible function over states (e.g., if it is a quasi-static process [87, 107]), then in general the process will map multiple initial distributions to the same final distribution. As an example, bit erasure is a non-invertible map over the associated unit simplex, and so maps any initial distribution to the same ending, delta function distribution.

Time-reversing a process might recover the initial distribution even if the dynamical system is very noisy. For example, evolving a bit in a process that (quasi-statically) takes an initially uniform distribution into a uniform distribution, but is so noisy that it loses all information about the precise initial state by the time it ends, is thermodynamically reversible but logically irreversible. Conversely, one can easily build a system that implements the identity map while producing an arbitrarily large amount of EP, e.g., if one has sufficiently high energy barriers between the states of the system.

In sum, the fact that a given process obeys one kind of reversibility (or not), by itself, has no implications about whether it obeys the other kind of reversibility.[14] See [6, 95, 100, 107] for more on this topic.

As a historical comment, it is worth noting that modern nonequilibrium statistical physics wasn't developed when Landauer, Bennett, and co-workers did their pioneering work on the thermodynamics of computation. As a result, they had to couch their insights concerning a fundamentally *non*-equilibrium process — computation — in terms of equilibrium statistical physics. Unfortunately, this led to confusion in the early work on the thermodynamics of computation, even in some of the most important and path-breaking of that work (e.g., in [84, 110]). This confusion resulted in significant controversy, which lasted for over three decades in the philosophy of science community and even longer in the physics and computer science communities [101, 103, 111, 112].

Only with the recent breakthroughs in nonequilibrium statistical physics do we have a fully formal framework for understanding the issues that Landauer and Bennett focused on. In particular, only now do we have the tools to analyze the relationship between the thermodynamics of a logically irreversible computer and the thermodynamics of a logically reversible computer that computes the same input-output function in a fully rigorous manner. (See Sections XI, XIV C and XIV D below for some preliminary work related to this topic.)

[14] Of course, this does not mean that there are no thermodynamic distinctions between logically irreversible and logically irreversible processes. To give a trivial example, the Landauer cost of a logical reversible process is zero, no matter what the initial distribution is, which is not true of logically irreversible processes in general.

VII. CONVENTIONS FOR CALCULATING ENTROPIC COSTS IN COMPUTATIONAL MACHINES

The definitions of computational machines given in Section IV are standard ones found in computer science textbooks. However, these definitions do not specify enough detail of how to physically implement those machines to uniquely fix the associated entropy dynamics. (A related point was made in [113].) In fact, in general there are an infinite number of different (continuous time) Markov chains that implement the same (discrete time) dynamics of a given computational device. Those different CTMCs will result in different entropic dynamics, in general.

One example of these phenomena arise with circuits. Suppose that we require that the dependency structure of the rate matrices in a CTMC we use to implement some specific circuit reflects the dependency structure of the wiring diagram of that circuit. Concretely, this means that the dynamics of every physical variables in a given gate in the circuit is not allowed to depend on any information in the circuit that resides outside of the inputs to that gate, even if its inputs are statistically correlated with such information in the state of other variables. When there is such a constraint it is possible that long-range mutual information between the physical variables in different gates gets lost as the circuit performs its computation. That results in nonzero irreversible entropy production in running the circuit. (This phenomenon is analyzed in depth in [69], and summarized in Section X below; see also [114].) In contrast, if there are no such constraints on how the CTMC implements the circuit, allowing each gate to exploit the state of any variable in the circuit, then we can achieve optimal thermodynamic efficiency, with zero irreversible entropy production. However, the computer science definition of a circuit does not specify whether there are such constraints on the CTMC that can be used to to implement the circuit.

Much of the material in Section X through **??** below consists of filling in these physical details that are absent from the computer science definitions of computational machines, and then analyzing the consequences for the entropic costs of physical implementations of those machines. In addition, Section XV describes some of the surprising aspects of the relationship between a given discrete time dynamical system and the set of CTMCs that can implement that system.

First though, in the rest of the current section I present some limitations I will impose on how we are allowed to "fill in the details" in going from a computer science definition of a computational machine to a CTMC that implements it. Then in Section VI and Section VIII I illustrate CTMCs that adhere to these limitations and that implement some very simple information processing systems (sub-computers, in essence), before moving on to consider full-blown computational machines.

A. Conventions that are often explicit

The goal in this paper is to focus as much as possible on the entropic costs in a physical process that implements a computation that are due to the computation being implemented, rather than due to details of the underlying physical system. Accordingly, as is standard in the literature, I assume that the Hamiltonian at both the beginning and the end of each iteration of a computational device is uniform across all states.

In addition, the processes discussed in this paper are all time-*in*homogeneous, i.e., the rate matrices can vary with time. In practice this is typically accompanied by a time-dependency of the Hamiltonian function of the system, $H_t(x)$, providing it arbitrary freedom at all times $t \in (0, 1)$. (The dependence of the Hamiltonian on time is sometimes referred to as a "protocol" in the literature.) Physically, that time-dependence will arise if the Hamiltonian actually has the form $H(x, \lambda(t))$, where $\lambda(t)$ is a physical variable outside of the system of interest, whose state varies in time. However, the thermodynamics of the evolution of $\lambda(t)$ is not considered in most of the literature on stochastic thermodynamics. (See [94, 115, 116] for some exceptions.)

In the interests of circumscribing the scope of this paper, I adopt this same convention here, and simply allow the Hamiltonian to depend explicitly on time, without considering the thermodynamics of external systems that cause the Hamiltonian to do that. As is also conventional in much of the literature, from now on I choose units so that $k_B T = 1$, except where explicitly stated otherwise.

B. Conventions that are usually implicit

In addition to the explicit conventions described in Section VII A, there are many other conventions that are implicit, and which vary from paper to paper. In particular, much of the confusion in the literature concerning the entropic costs of computers can be traced to researchers using different implicit "accounting conventions", for how to measure the entropic costs of an iteration of a computational device. In general, there is no right or wrong choice for such a convention. However, it is imperative that the convention one adopts be made explicit. Moreover, some conventions are easier to motivate than others.

First, I fix some terminology. Computational machines involve connected sub-computers, e.g., circuits involve connected gates. Define an **iteration** of such a machine to be the process of it simultaneously updating the state of all its sub-computers. So each sub-computer runs the same logical map in each iteration. Define a **run** of a machine as the entire sequence of iterations it must

run in order to complete a computation. The number of iterations in each run is not pre-fixed for some computational machines (and in fact might not be well-defined for certain inputs), e.g., for typical FAs or TMs. We cannot choose units of time so that such a machine always finishes its computation in the same interval, $[0, 1)$. So I implicitly choose units so that each iteration of the machine occurs in a time interval of the form $[t, t + 1)$ for $t \in \mathbb{Z}^+$.

No computer operates in isolation. At a minimum, it must be coupled to an external system to generate its inputs, and to a (possibly different) external system to record or act upon its outputs. For simplicity assume that this coupling occurs through an explicitly designated set of input variables and output variables of the device, x^{IN} and x^{OUT}, respectively. In many real-world systems there is some overlap between x^{IN} and x^{OUT} (indeed, they may even be identical). However, for simplicity, in this paper I restrict attention to devices in which x^{IN} and x^{OUT} have zero overlap.

In thermodynamics, it is conventional to focus on complete cycles of physical systems, in which the system ends in the same state that it started (i.e., to focus on processes that ultimately map an initial distribution over states of the system to an identical ending distribution). This convention can greatly clarify the analysis. For example, arguably the crucial breakthrough in the current understanding of why Maxwell's demon does not violate the second law occurred when a full cycle of the joint gas-demon system was considered, in which the demon's memory gets reinitialized after each iteration, before that memory is used to record the next observation of the state of the gas [61].

To formalize this convention in the current context, I define a **cyclic device** as one that has two properties. First, all of the device's variables — input, output, or internal (in its sub-computers, if it has any) — are reinitialized by the end of each run of the device, before a next input is read in and the next run of the device begins. Second, the rate matrix for the device is periodic with period 1, the time it takes for it to run each iteration. This ensures that if there are sub-computers in the device, that each reruns its logical update function in every iteration, with the same entropic cost functions of its initial distribution in each iteration.

In the real world, many computational devices are not cyclic. For example, often variables are not reinitialized after the device is used; instead, they get overwritten with new values the next time the device is used. However, not requiring that the devices run a complete cycle complicates the analysis, which is (partly) why it is conventional in thermodynamics to analyze the behavior of systems that go through a complete cycle, returning to the state that they started from. For the same reason, from now on I assume that any device being discussed is a cyclic device, unless explicitly stated otherwise.

However, simply requiring that the device goes through a complete cycle doesn't specify what portion of the entropic costs generated as it completes that cycle are ascribed to the device, and what portion are instead ascribed to the external systems the device interacts with during that cycle.

To motivate a convention for how to ascribe those costs, first note that in practice we will almost always want to have an external copy of the ending value of x^{OUT}, the variable in the device that contains its output, that persists after that variable gets reinitialized. Such a copy of the ending value of x^{OUT} will then be used by other devices, e.g., as input to a subsequent computation, or as a signal to a controller, or as data to be stored for later use. Importantly, we can exploit that external copy of the value of the output variable to reinitialize that output variable, before the beginning of the next run of the device.

I use the term **answer-reinitialization** to refer to such a process that makes a copy of the ending state of x^{OUT} in an offboard system and reinitializes x^{OUT} as it does so. I will ascribe the entropic costs of answer-reinitialization, if any, to the external system rather than the device itself.[15]

In contrast to the case with outputs, in many real world computational devices some offboard system, perhaps even the brain of a human user of the computational device, generates new inputs for the device. Those then need to be "read into the input of the device". This can be done with zero EF — if x^{IN} has been initialized to some standard value, with probability 1, every time it gets a new value.[16] I refer to this as the **input-reinitialization** of the device. I ascribe the costs of input-reinitialization to the device itself.

These two conventions, concerning how we ascribe the costs of reinitializing the inputs and the outputs, fit together to ensure that we do not get the "wrong" answer for the entropic costs of many simple computations. For example, suppose we want to run some device multiple times that receives inputs IID distributed according to some distribution $p_0(x^{IN})$ and uses those inputs to produce outputs that are distributed according to some distribution $p_1(x^{OUT})$. To accord with common use of the term "Landauer's bound" (e.g., to describe the Landauer cost of bit erasure), it would be good if the Landauer cost that we ascribe to running the device is $S(p_0(X^{in})) - S(p_1(X^{out}))$. Now under our convention concerning the entropic costs of the input-reinitialization, that reinitialization

[15] Such costs may be zero. For example, under the presumption that such a copy is stored in a variable that was itself in a well-specified initialized state (e.g., if that copy of the device's output is stored in the initialized input variable of a downstream device), there is zero net Landauer cost in the joint system during answer-reinitialization. See [6] and the discussion in Section XI below.

[16] If x^{IN} is in its initialized state with probability 1 whenever a next input is copied into it, the Landauer cost of that copy operation is zero. Similarly, since the copy operation is logically invertible, the mismatch cost is zero. Given the default assumption in this paper that residual entropy costs are zero, this means that EF equals zero, as claimed. See also [6].

adds a term $S(p_0(X^{in}))-0$ to the Landauer cost ascribed to the device. In addition to reinitializing its input though, the device generates the output (after which an external system copies the output offboard and then performs the answer-reinitialization, at no cost to the device). That computation results in a contribution $0-S(p_1(X^{out}))$ to the Landauer cost (since by hypothesis, with probability 1, x^{OUT} was in its initialized state before receiving the results of the computation). Summing these two contributions gives a total Landauer cost of $S(p_0(X^{in}))-S(p_1(X^{out}))$ to ascribe to the device, exactly as desired.

In addition to these two conventions, I ascribe the costs of *generating* a new input for each successive computation to the external system that does so.[17] I refer to this full set of requirements for how to ascribe the entropic costs of running a cyclic device as the **standard accounting** convention.[18]

Example 7. *Consider iterating a cyclic device K times, but only answer-reinitializing and input-reinitializing when all K iterations are complete. As an example, this is what happens in a modern synchronous computer if you run a "computation" that requires K clock cycles, where you only copy the state of the RAM to a disk drive at the end of those K cycles, after which you reinitialize the RAM, ready to receive the input for a next computation.*

Summing the Landauer costs of the K iterations, almost all terms cancel, leaving the entropy of the initial distribution over states minus the entropy of the ending distribution over states. That difference is exactly what one would expect the minimal EF to be, if we considered the entire sequence of K iterations as a single process. So as far as Landauer cost is concerned, the number of iterations used to transform the initial distribution into the final distribution is irrelevant; only the overall transformation between those distributions matters.

On the other hand, each iteration will contribute a strictly positive residual EP in general, with no such cancellations. Moreover, in current real-world systems, the prior distribution over states of the system will be the same at the beginning of each iteration. (For example, that is the case in modern synchronous computers.) On the other hand, the actual distribution over states will change from one iteration to the next. As a result, each iteration will contribute a strictly positive mismatch cost in general, without cancellation of terms. So in contrast to the case with the Landauer cost, the number of iterations used to transform the initial distribution into the final distribution will have a big effect on total EP, if we don't up date the prior distribution from one iteration to the next.

[17] Such choices not to include certain costs in the analysis are similar to the common choice in the stochastic thermodynamics literature *not* to include the thermodynamics of the (usually un-modeled) process that drives the time-variation of a CTMC, as discussed at the end of Section V A.

[18] Of course, in some circumstances one might want to use some non-standard accounting. However, for simplicity no such alternative accounting conventions are considered in this paper.

However, suppose we do update the prior in each iteration, by propagating it forward one iteration, i.e., by applying the conditional distribution of that iteration to the prior. (See Section IV A.) In this case, there will be cancellations just like those with Landauer costs. Specifically, the sum of the mismatch costs over K iterations will just equal the KL divergence between the actual and prior distributions before the first iteration begins, minus that divergence after the K'th iteration ends. Since cross entropy is the sum of entropy and KL divergence, the same property also holds for cross entropy, i.e., we get the same cancelling of terms for the total EF, if the residual EP is zero.

As a final comment, typically in current real-world computers there is no copy of the original input into a device that is maintained until the input-reinitialization process of that device.[19] Suppose though that a copy of the initial state of x^{IN} *were* to persist until the input-reinitialization, e.g., in some external, "offboard" system. Then the Landauer cost of that input-reinitialization could be reduced to zero, simply by using that offboard copy of x^{IN} to change the state of the input variable of the device (which contains the same value x^{IN}) back to its initialized state. (This point is also emphasized in [101].) As a result, under standard accounting, the Landauer cost of the full cycle would be $0 - S(p_1(X^{OUT}))$, i.e., it would be *negative*. Moreover, if a copy of the input were to persist, mismatch cost would be zero, whether or not the computation were noisy.[20]

To properly analyze whether these entropic benefits of storing a copy of the input in an offboard system outweigh the costs of doing so requires us to expand the scope of our analysis of entropic costs to include the offboard system. This is done below in Section XI. For the rest of this paper, to keep the analysis focused on the costs of running the device without consideration of the costs of any associated offboard systems, I do not allow any coupling of the device with the external universe in a run except during the initial process reading in the inputs for the run or the ending process copying offboard the values of the output variables of the device (with the entropic costs of those two processes not ascribed to the device). In addition, I require that the ending value of

[19] A simple practical reason for this is that many real-world computational machines comprise a huge number of computational devices (e.g., real-world circuits comprise millions of gates), and the memory requirements for storing copies of all the inputs to all those devices would be exorbitant. Moreover, if we were to rerun the computational machine, then either we would have to erase all those copies of the original inputs to make room for the new inputs – at high entropic cost, in general – or use yet more storage for the new inputs. However, see Section XI below.

[20] To see this, use the chain rule for KL divergence twice to expand the mismatch cost for the case where the input persists as

$$\begin{aligned}
&D\Big[p_0(X^{in}, X^{out}) \parallel q_0(X^{in}, X^{out})\Big] - D\Big[\pi(X^{out}|X^{in})p_0(X^{in}) \parallel \pi(X^{out}|X^{in})q_0(X^{in})\Big] \\
&\quad = D\Big[p_0(X^{in} \mid X^{out}) \parallel q_0(X^{in}|X^{out})\Big] + D\Big[p_0(X^{out}) \parallel q_0(X^{out})\Big] \\
&\qquad\qquad - D\Big[\pi(X^{out}|X^{in})p_0(X^{in}) \parallel \pi(X^{out}|X^{in})q_0(X^{in})\Big] \\
&\quad = D\Big[p_0(X^{in}) \parallel q_0(X^{in})\Big] - \Big(D\Big[\pi(X^{out}|X^{in}) \parallel \pi(X^{out}|X^{in})\Big] + D\Big[p_0(X^{in}) \parallel q_0(X^{in})\Big]\Big) \\
&\quad = D\Big[p_0(X^{in}) \parallel q_0(X^{in})\Big] - D\Big[p_0(X^{in}) \parallel q_0(X^{in})\Big] \\
&\quad = 0
\end{aligned}$$

x^{OUT} of any device implementing some computational machine only contains the output specified in the computer science definition of the machine, as in Section IV.

C. Accumulating entropic costs until an event occurs

How should we measure the entropic costs incurred by a computational machine during a run that takes more than a single iteration? If the number of iterations that the machine runs is a constant, never varying from run to run, there is no difficulty due to having multiple iterations per run. An example of such a **fixed-duration** (or "fixed length") machine is a circuit of depth K. In each iteration $1 \leq m < K$ the states of the gates in level m of the circuit are used to set the states of the gates in level $m+1$, and so the total number of iterations the circuit runs is $K-1$. To calculate the entropic costs of a run of the circuit for some initial distribution over input nodes p_0, we just add up the entropic costs accrued by running all $K-1$ iterations of the circuit, starting from that distribution p_0. (See Section X.)

However, the situation is more complicated if the number of iterations is a random variable, e.g., as occurs with most FAs or TMs whose input is generated by a random process. This complexity can be illustrated with some recent papers in the stochastic dynamics literature which analyze the entropic costs incurred by running a system until a first passage time occurs. (N.b., the systems analyzed in these papers need not be computational machines.) As an example, [117] analyzes the probability density function as a function of time t for the event, {the total EP generated by a given CTMC first reaches some pre-specified value ϵ at t}. As another, related example, [118] derives formulas concerning the probability density function of the event, {the first time t after a process starts that a net current reaches a given threshold value}. These formulasinvolve the total amount of EP incurred up to t.[21]

A scientist experimentally testing the predictions made in these papers would need to observe the system continually, to see exactly when the event of interest occurs.[22] Such observation by an external system (namely the scientist) incurs entropic costs in general. Arguably, to do the thermodynamic analysis properly, one should incorporate those costs incurred by observing the system. (Cf., the importance in analyzing Maxwell's demon of including the thermodynamics of its observing the system it wants to extract work from.) Indeed, because the total time the system

[21] In [118], this is referred to as "time fluctuations in a fixed-current ensemble".

[22] In fact, in many scenarios the scientist is continually observing many other attributes of the system, in addition to the bit of whether the event of interest has occurred. For example, in [118], the total level of net current due to transitions that has occurred up to t must be observed, and continually updated, just to determine that bit of whether the event of interest has occurred.

runs is variable, arguably one should also incorporate the entropic costs that are incurred *after* the event occurs, unless built in a process that will "turn off" the system and observation apparatus when the event occurs, so that they no longer generate entropic costs.

If we do not model the entropic costs of the external system observing when the event of interest occurs, and / or do not incorporate entropic costs incurred by the system after that event occurs, we can come to misleading conclusions. This is illustrated in the following example, involving a machine that implements bit erasure, but takes either one or two iterations to do so, with the number of iterations determined randomly.

Example 8. *Suppose our system has three states,* $\{a, b, c\}$*. It starts at* $t = 0$ *with* $P(a) = P(b) = 1/2$*. The "end of the computation" is signaled by having the state equal either* c*. Without loss of generality we assume that* c *is a fixed point of the dynamics at all times.*

Suppose that during the first iteration, between $t = 0$ *and* $t = 1$*, the state* $a \to a$ *and the state* $b \to c$*. During the second iteration, ending at* $t = 2$*, both* a *and* c *get mapped to* c*. So the full, two-iteration process implements a two-to-one map, sending both* a *and* b *to* c*. This is the sense in which the map implements bit erasure.*

The probability that the computation ends after the first iteration is $1/2$*. The total drop in entropy that a scientist observing the system would record by then is* $\ln[2] - \ln[2] = 0$*, since the state of the system is equally uncertain at* $t = 0$ *and* $t = 1$*. Under standard accounting, this would be the Landauer cost if the computation ended at* $t = 1$*.*

The probability that the computation instead ends after the second iteration is also $1/2$*. However, the total entropy drop if the computation ends then is just the normal bit-erasure Landauer cost,* $\ln[2]$*. So* if we were to stop accumulating drops in entropy when the computation ends*, then the expected Landauer cost would be* $(1/2) \cdot 0 + (1/2) \ln[2] = (\ln[2])/2$*. This is half the Landauer cost for bit erasure in a single iteration, the value given by "Landauer's bound". On the other hand, if we calculated the total drop in entropy during the full interval from* $t = 0$ *to* $t = 2$*, ignoring whether the computation has ended early, under we would again get a value that equals the Landauer bound for bit erasure,* $\ln[2]$*.*

So it might seem that we can avoid Landauer's bound, cutting the drop in entropy by half, on average, so long as we introduced stochasticity in when exactly the bit operation completes.

These issues arise when interpreting some of the early work on the thermodynamics of Turing machines, in which entropic costs are only accumulated up to the time that the TM halts, and no consideration is given to the entropic costs of an external observation apparatus watching to see

exactly when the TM halts. (See the discussion in Section XIV.)

In the next section I review earlier work that (implicitly) used standard accounting to analyze the entropic costs of any device that in one iteration implements an arbitrary transformation of the entire input variable, without any constraints on how it implements that transformation. Then in the rest of this paper I consider full computational machines, all of which run for multiple iterations, and have constraints of one type or another on how they process their inputs, and therefore have a more complicated relationship between the computation they perform and the entropic costs they incur as they do so. Except in Section XI and (to a degree) in Section XIV B, in those sections I restrict attention to the entropic costs of cyclic devices, under standard accounting.

VIII. ENTROPY DYNAMICS OF UNCONSTRAINED DEVICES

Extending the definition of an AO circuit, define an **all at once** (AO) device that implements π in some fixed time interval such that for some initial distribution p_0, the EF generated by running the device on p_0 is $S(p_0) - S(\pi p_0)$. As illustrated below, much of the richness in the thermodynamics of computation arises when we consider non-AO devices. However, to illustrate the concepts introduced above, in this section I begin by reviewing the thermodynamics of AO cyclic devices under standard accounting.

Eq. (30) gives the theoretical minimal EF that could be produced by any process that applies a conditional distribution $\pi(x_1|x_0)$, relating the state of the system at time 1 to its earlier state at time 0, to an initial distribution p_0. However, that equation doesn't say whether the bound it provides can be achieved, and if so how.

For a process to achieve that bound means that it is thermodynamically reversible for the initial distribution p_0. A large body of literature has developed showing that for certain maps π, if we are given any p_0, it is possible to construct an associated "quasi-static" process that both implements π and is arbitrarily close to thermodynamic reversibility if run on p_0 [6, 95, 119]. Many of these papers were motivated by considering bit erasure, the venerated model of information processing reviewed in Section VI. However, bit erasure has the quixotic property that the distribution over outputs x_1 is independent of the precise input $x_0 \in X$. Reflecting this, even when they considered other π's besides bit erasure, these papers were implicitly restricted to maps $\pi(x_1|x_0)$ where the distribution over outputs is independent of the precise input x_0.

This means the analyses in these papers do not directly apply to the most common maps performed by gates in real computers, in which the output distribution is *dependent* on the initial

state, e.g., the logical AND map, the bit flip, etc. (See [103] for an early discussion touching on this limitation.) Indeed, in quasi-static processes in which the system is in equilibrium with the heat bath at all times, *all information about the initial state is lost by the end of the process*, being transferred to the heat bath. (This can be shown explicitly using stochastic thermodynamics; see Section XV.) Hence, no such process over X can implement a map π in which the output distribution depends on the input state.

However, given any π and p_0, it is now known how to construct a cyclic device that implements π, and whose EF equals the Landauer cost in the special case where the prior of the device equals p_0 — if one uses a process defined over a space that is strictly larger than X [62, 87, 102–106, 120]. The key idea is to expand the space of the system beyond the original space X, used to define π and p_0, into a "partially hidden" space $X \times X'$, where a specific state in X' (here written as $\perp$) is identified as the "initialized state" of X'. The analyses in those papers show how to design an associated "partially hidden" cyclic device, with a rate matrix $\mathscr{W}$ operating over $X \times X'$, such that any initial value $(x_0, \perp) \in X \times X'$ gets mapped to an ending distribution $(\pi\delta(x, .), \perp)$. So one can read off the dynamics of the "visible" device π evolving over X from the dynamics of the "visible states", X of the device $\mathscr{W}$ evolving over $X \times X'$.

The prior of this device operating over $X' \times X$ is assumed to be of the form $(q_0(x), \delta(x', \perp))$. It is shown in [62, 105, 106] that if $q_0 = p_0$, then this hidden device implements the map over π over X and incurs zero EP, achieving the Landauer bound for applying π to p_0. In addition, it is explicitly shown there that if $q_0 \neq p_0$, then the resultant mismatch cost of the hidden device over $X \times X'$ is exactly the drop in KL divergence for the original device implementing the distribution π, given by evaluating Eq. (37) for the initial distribution $p_0(x)$ over the visible space X, the prior $q_0(x)$ over X, and the conditional distribution π.

In more detail, as it is formulated in [105, 106, 121], this partially hidden device works as follows. First, the device stores a copy of x_0 in X'. So $|X'| = |X|$. (As described in [6], since a X' starts in its initialized state with probability 1, this copy operation is logically reversible, and so has both zero Landauer cost and zero mismatch cost.) Then, without changing the value of this copy, that copy is used to guide a quasi-static process that evolves the distribution p_0 over the visible states, to implement π. After this is done the value $x' \in X$ is reinitialized to 0. This reinitialization is guided by the ending value x_1, and is based on the prior $q_0(x)$, which must govern the value of the copy of x_0 stored at this point in X'; it is this reinitialization step that results in the mismatch cost. After all this is done, as required by standard accounting, a copy of x_1 is copied to an offboard

system, and X is reinitialized, in preparation for the next input. [23]

In addition, in many real-world physical systems, we do not know the actual distribution p_0 of the initial states of a system, i.e., we do not know the distribution of the "inputs" to the device. At best, we might have a distribution over such distributions, $P(p_0)$. For example, in a real-world digital computer, a given user implicitly specifies some distribution p_0 of inputs to the computer. However a different user would have some different distribution p'_0 over inputs to the computer. In this case P is the probability distribution over which user is using the computer. As another example, a cell operating in some particular environment will be subject to a distribution p_0 over physical characteristics of that environment, e.g., over macromolecules it may be detecting in that environment. In this case a given environment specifies a distribution p_0, and P is the distribution over environments. [62] analyzes the implications of such distributions over initial distributions p_0 for the entropic costs of the device, showing that the Landauer cost increases by a Jensen-Shannon divergence between $P(p_0)$ and the associated distributions p_0. (This Jensen-Shannon divergence is called "entropic variance" in [62].) Finally, [62, 105, 106] also shows how to extend them to the case where the set of visible states X is a coarse-graining over some some underlying space.

This early analysis showing how to use an extra, hidden space X' to implement an arbitrary conditional distribution π raises many interesting research questions. Since hidden state spaces can be expensive to build into physical devices in the real world, this raises the question of what the fundamental limits on the minimal X' is, in order to implement a desired π. Some results concerning this question and related ones are summarized in Section XV. In particular, it turns out that augmenting the space X with additional hidden states is *necessary* to implement nontrivial π, *even without introducing considerations of thermodynamical reversibility.* Surprisingly, as discussed in Section XV, it is impossible for *any* CTMC to implement any non-trivial map π.

IX. EFFECTS ON ENTROPIC COST OF CONSTRAINING THE RATE MATRIX

All modern physical systems that implement computational machines use sub-computers that can only access a subset of the variables in the overall machine. This means that there are constraints on how the overall machine can operate. These constraints are a major distinction between computational machines and the systems analyzed in Section VI and Section VIII. In this section I review some recent results relating the precise constraint on what variables a given sub-computer inside a computational machine is allowed to access, and the consequences of that constraint for

[23] This type of construction using a hidden space X' was rediscovered in [122] and App. A of [114].

the entropic costs of running the machine.

As notational shorthand, I assume that any conditional distribution governing the dynamics of a device operating within an overall computational machine that is written in the form $\pi(y|x)$ is implemented with a CTMC and Hamiltonian that do not involve any of the variables in the overall computational machine other than x and y. As an example, suppose we have an overall system (computational machine) with three subsystems (devices), A, B, Z, having states x, y and z, respectively. Suppose that the device B gets its input x from the output of device A and uses that to compute its output y during some specific iteration n of the overall system. Suppose further that to reflect engineering constraints in how we can build the device, we want to make sure that we do not model the system with a CTMC and Hamiltonian that ever couple the state of B's variables to any external variables besides the state of A. Then I write the update distribution for device B as $\pi(y|x)$. On the other hand, suppose that B implements the same logical function taking $x \to y$, but that the Hamiltonian governing the system during iteration n is allowed to couple the variables in B to the values of some external variable z in the overall system that does not lie in A.[24] In this case I write the conditional distribution governing B's computation as $\pi(y|x, z)$, even though the distribution over values y is independent of the precise value of z. The advantage of this shorthand it that is allows us to use the form of the update distribution π to specify constraints on which physical variables are allowed to be directly physically coupled to which other ones, e.g., through an interaction Hamiltonian.

Under this notation, any physical system implementing $\pi(y|x)$ is subject to extra constraints on what rate matrix CTMC it can be implemented with, compared to a physical system implementing $\pi(y|x, z)$. However, typically a physical process obeying such constraints on the rate matrix CTMC and Hamiltonian cannot achieve the EF of Eq. (30) (which presumes a constraint-free system). This will result in unavoidable EP.

A. Subsystem processes

To make these general considerations more precise, I now introduce a type of process which will play a central role in the rest of this paper:

Definition 7. *Suppose that the rate matrix of a CTMC defined over $X = X_A \times X_B$ can be expressed*

[24] As discussed below, such coupling would often allow the entropic costs of running B to be reduced, even if the evolution of y is independent of z. Loosely speaking, this is the case if the initial value y is statistically coupled with the value z. In such a case, the physical process that updates the state y can be more thermodynamically efficient if it can exploit the information about y that is contained in z.

as a sum over mechanisms ν of associated rate matrices, each of which is of the form

$$W^{v}_{x_A,x_B;x'_A,x'_B}(t) = W^{\nu}_{x_A;x'_A}(t)\delta(b',b) + W^{\nu}_{x_B;x'_B}\delta(a',a)$$

for appropriate rate matrices $W^{\nu}_{x_A;x'_A}(t), W^{\nu}_{x_B;x'_B}(t)$. Then the CTMC is a ***subsystem process*** *over $X_A \times X_B$.*

(See [69] for a more general definition, applicable even if we do not assume that the system evolves according to a CTMC.) As an example, a physical system can evolve as a subsystem process if the reservoirs it is attached to are infinitely large (so that A and B cannot couple indirectly through the reservoirs), and if at all times $t \in [0,1]$ the energy function decouples subsystems A and B; i.e., the Hamiltonian obeys

$$H_t(x_A, x_B) = H^A_t(x_A) + H^B_t(x_B)$$

for all $t \in [0,1]$.

All the computational machines considered in this paper operate with synchronous rather than asynchronous updating, in which each device inside an overall computational machine runs in a "modular" fashion, not reacting to interrupts from outside of itself while it runs. Physically, this means that once the variables in any such device are set to their initial values, the device is run by a process in which those variables evolve independently of the rest of the variables in the overall computational machine. So it is run by a subsystem process. As an example, any process that runs a gate in a circuit is a subsystem process.

The following result in proven in App. B:

Proposition 2. *For any subsystem process over a state space $X_A \times X_B$,*

1. *Subsystems A and B evolve independently over that interval, i.e., the conditional distribution of the state evolution is of the form*

$$\pi(a_1, b_1|a_0, b_0) = \pi^A(a_1|a_0)\pi^B(b_1|b_0)$$

2. *The EF of the joint system during the process can be written as*

$$\mathcal{Q}(p_0^{A,B}) = \mathcal{Q}_A(p_0^A) + \mathcal{Q}_B(p_0^B)$$

where $\mathcal{Q}_A(p_A)$ is the EF that would be incurred for a CTMC over state space X_A with rate matrices $W^{\nu}_{x_A;x'_A}(t)$ (i.e., if subsystem B did not exist at all), and similarly for $\mathcal{Q}_B(p_B)$.

In light of Proposition 2(2), I sometimes refer to $\mathcal{Q}_A(p_0^A)$ as the **subsystem EF** of subsystem A, and similarly for $\mathcal{Q}_B(p_0^B)$. I also sometimes refer to $\mathcal{Q}(p_0^{A,B})$ as the **system-wide EF**.

Note that by the Second Law of Thermodynamics [86], $\mathcal{Q}_A(p_0^A) \geq S(p_0^A) - S(p_1^A)$, where equality arises for appropriate (typically quasi-static) rate matrices $W^{\nu}_{x_A;x'_A}(t)$. (Similar considerations hold for $\mathcal{Q}_B(p_0^B)$.) Accordingly, I refer to $S(p_0^A) - S(p_1^A)$ as the **subsystem Landauer cost** of subsystem A, and and write it as $\mathcal{L}(p_0^A, \pi^A)$, or just $\mathcal{L}(p_0^A)$ for short. (Again, similar considerations hold for $S(p_0^B) - S(p_1^B)$.)

Along the same lines, I write the minimal EF of any process that implements $\pi(a_1, b_1|a_0, b_0)$ as

$$\mathcal{L}(p_0^{A,B}, \pi^{A,B}) = S(p_0^{A,B}) - S(\pi p_0^{A,B}) \tag{45}$$

I refer to this as the **system-wide Landauer cost**, or sometimes as the AO Landauer cost, since it's the Landauer cost of an AO device that implements π.

While the EF of a full system undergoing a subsystem process is additive over the EFs of its subsystems, in general the Landauer cost of the full system undergoing a subsystem process is *not* additive over the Landauer costs of its subsystems. More precisely, in general the sum of the subsystem Landauer costs, $S(p_0^A) - S(p_1^A) + S(p_0^A) - S(p_1^A)$, differs from the system-wide Landauer cost, $S(p_0^{A,B}) - S(p_1^{A,B})$. This reflects the fact that the system-wide Landauer cost is the minimal EF of implementing $\pi^{A,B}$ using *any* process, without imposing the constraint that the process must be a subsystem process.

I will refer to the difference $\mathcal{L}(p_0^{A,B}, \pi^{A,B}) - \mathcal{L}(p_0^A, \pi^A) - \mathcal{L}(p_0^B, \pi^B)$ as the **Landauer loss** of the subsystem process. It equals the change in mutual information between subsystems A and B during the process:

$$\begin{aligned}
&\mathcal{L}(p_0^A, \pi^A) + \mathcal{L}(p_0^B, \pi^B) - \mathcal{L}(p_0^{A,B}, \pi^{A,B}) \\
&\quad = S(p_0^A) + S(p_0^B) - S(p_0^{A,B}) \\
&\qquad - \left[S(\pi^A p_0^A) + S(\pi^B p_0^B) - S(\pi^{A,B} p_0^{A,B}) \right] \\
&\quad = I_{p^{A,B}}(A;B) - I_{\pi p^{A,B}}(A;B)
\end{aligned} \tag{46}$$

A simple example of Landauer loss was presented in Example 6. Loosely speaking, the concept of Landauer loss is extended to involve more than two random variables in Section X, in order to analyze the entropic costs of circuits. In particular, when there are more than two random variables, the mutual information terms in Eq. (46) get replaced multi-information terms.

Now that we have the appropriate definitions for distinguishing the EFs and Landauer costs of the subsystems and the entire system, we can do the same for their EPs. Define the **system-wide**

EP of a subsystem process, $\sigma(p_0^{A,B})$, as the difference between the system-wide EF of that process and the system-wide Landauer cost of the process. Then by combining the additivity of EF, the nonadditivity of Landauer cost, and Eq. (36), we see that the system-wide EP of a subsystem process is not additive over the EPs of its subsystems. To make this more precise, define the **subsystem EP** of subsystem A as

$$\sigma_A(p^A) = \mathcal{Q}(p^A) - \mathcal{L}(p^A, \pi^A) \tag{47}$$

where for simplicity I have dropped the subscript on p^A that indicates it is an initial distribution. (I make analogous definitions for subsystem B.) Since $\mathcal{Q}(p^A)$ is linear in p^A, we can apply the decomposition of EP in Section V B to subsystem processes and write

$$\begin{aligned}\sigma_A(p^A) \; &= \sum_{c \in L(\pi^A)} p^A(c) \\ &\times \Big(D(p^A(c) \| q^A(c)) - D(\pi^A p^A(c) \| \pi^A p^A(c)) + \sigma_A^{min}(c) \Big)\end{aligned} \tag{48}$$

where in analogy to the development in Section V, the **subsystem prior** q_A^c is any distribution

$$q^A(c) \in \arg\min_{r \in \Delta_c} \sigma_A(r)$$

Again following the development in Section V, it can be useful to re-express $\sigma_A(p^A)$ as the sum of two terms. The first is the **subsystem mismatch cost**,

$$\mathcal{E}_A(p^A) = D(p^A \| q_A) - D(\pi^A p^A \| \pi^A q_A)$$

(where $q_A(x_A) = \sum_c q_A(c) q^c(x_A)$ and as before the choice of the probabilities $q_A(c)$ is arbitrary). The second is the **subsystem residual EP**,

$$\sum_{c \in L(\pi^A)} p^A(c) \sigma_A^{min}(c) \tag{49}$$

(I make an analogous decomposition for subsystem B.) Note that by Eq. (47), $\sigma_A(p^A) \geq 0$ for all p^A. In particular, this must be true if $p^A(c) = q^A(c)$ for all $c \in L(\pi^A)$. Plugging that into Eq. (48), we see that subsystem residual EP is non-negative.

Finally, expand

$$\begin{aligned}\sigma(p) &= S(\pi p) - S(p) + \mathcal{Q}(p) \\ &= S(\pi p) - S(p) + \mathcal{Q}(p^A) + \mathcal{Q}(p^B) \\ &= S(\pi p) - S(p)\end{aligned}$$

$$+ \sigma_A(p^A) + \mathcal{L}(p^A, \pi^A) + \sigma_B(p^B) + \mathcal{L}(p^B, \pi^B) \tag{50}$$

where I have used Eq. (47). Eq. (50) formalizes the statement above, that system-wide EP is not additive over its subsystems. Eq. (50) also can be used to establish the following:

Corollary 3. *Given any subsystem process that implements π on initial distribution $p^{A,B}$,*

1. *The minimal system-wide EP equals the Landauer loss.*
2. *The minimal system-wide EP equals the system-wide mismatch cost.*
3. *If the process achieves the minimal EP and the priors of the two subsystems are unique, then those priors are the marginals of $p^{A,B}$, i.e., $q^A = p^A$, $q^B = p^B$.*

Proof. Fix the initial distribution p and the conditional distribution π. Eq. (50) tells us that the minimal EP generated by any system that applies π to p while using a subsystem process occurs when the two subsystem EPs achieve their minimal values, i.e., when $\sigma_A(p^A) = \sigma_B(p^B) = 0$. (For example, this could arise if the subsystems evolve quasi-statically [123].) So the minimal system-wide EP is $\mathcal{L}(p^A, \pi^A) + \mathcal{L}(p^B, \pi^B) - \mathcal{L}(p, \pi)$. This establishes the first claim.

Next, the system achieves the minimal EP iff the two subsystem EPs equal zero. In turn, they equal zero iff the subsystem residual EPs of both subsystems equals 0. This means that the residual EP of the entire system is zero. That in turn implies that the system-wide EP must equal system-wide mismatch cost, establishing the second claim.

Finally, again note that in order to achieve the minimal system-wide EP for initial distribution $p^{A,B}$, the subsystem EPs must both be zero. This means that their mismatch costs must both be zero when the full system is run on distribution $p^{A,B}$. Therefore a prior for subsystem A is given by p^A, and similarly for subsystem B. □

B. Solitary processes

An important special type of subsystem process is one in which $W^{\nu}_{x_B;x'_B}(t) = 0$ for all $\nu, x_B, x'_B \neq x_B$ and $t \in [0, 1]$. Any such subsystem process is called a **solitary process over** A. I will sometimes refer to the value $x_A(0)$ in a solitary process as the **input** to that process, with $x_A(1)$ being its **output**. By Proposition 2, in a solitary process $\mathcal{Q}_B(p_0^B)$ equals the EF that arise by evolving a system over X_B with rate matrices $W^{\nu}_{x_B;x'_B}(t)$. So in a solitary process over subsystem A, $\mathcal{Q}_B(p_0^B)$ equals zero identically, and therefore the EF of a solitary process equals the EF of subsystem A.

Similarly, since the state of subsystem B cannot change in the subsystem process, its subsystem Landauer cost is zero, and therefore its subsystem EP is zero.

It is often easier to analyze sequences of solitary processes than sequences of subsystem processes. Moreover, computational machines that are conventionally defined with some of their devices running in parallel can almost always be redefined to have those devices run serially, according to some convenient topological order that respects the original definition of the machine. We can then model the running of that machine as a sequence of solitary processes.[25] This allows us to analyze the entropic costs of running the full system by decomposing it into a sum of

Note though there are devices in real-world computational machines that are not governed by solitary processes. For example, suppose that the gates in a circuit other than some gate g use dynamic ("active") memory to maintain their values while g runs. Then the states of those gates are in a nonequilibrium steady state (NESS) while g runs, i.e., the terms in the rate matrix that govern their dynamics are not all zero. (Intuitively, the entropy flow in such an NESS — the work done on the system — is needed to maintain the NESS against the equilibrating thermodynamic force arising from coupling with the heat bath(s).) In this case g is run with a subsystem process that is not a solitary process. In the interests of space, such situations are not considered in this paper, or equivalently, it is assumed that the entropic costs generated by maintaining NESSs are arbitrarily small on the scale of the entropic costs of running the sub-computers that change the distributions over the states of some variables. (See Section XVI).

Suppose we have a solitary process over a subsystem A in which A is initially statistically coupled with some other subsystem B, and that the subsystem EP of A is zero. Suppose that that initial coupling between A and B is reduced the process unfolds. Then by Eq. (50), the system-wide EP, and therefore the Landauer loss, are nonzero, even though the two subsystem EPs equal zero.

It might seem paradoxical that the Landauer loss in a solitary process can be nonzero. After all, both the system-wide Landauer cost and the subsystem Landauer cost involve processes over the same state space (namely, $X_A \times X_B$), and implement the same conditional distribution π over that space. How can two processes, over the same space, implementing the same conditional distribution, have different minimal EFs, especially if one of the subsystems doesn't even change its state under that conditional distribution?

The reason for the difference in minimal EFs is that system-wide Landauer cost is the minimal

[25] As an example, suppose the computational machine is a circuit, with the gates being the subsystems in question. The associated topological order is specified by the circuit's DAG, and running the gates serially according to that order ensures that the circuit implements the same input-output map as does sequentially running various subsets of the gates in parallel, as specified in the original DAG.

EF of any rate matrix that implements π, but the subsystem Landauer cost is the minimal EF of any rate matrix that implements π *while also satisfying the extra constraints defining a solitary process*. Due to those constraints, while the first kind of rate matrix is allowed to exploit the values of x_B as well as x_A to drive the dynamics of x_A, the second kind is not allowed to do that. More precisely, in the process whose EF equals the system-wide Landauer cost, for all mechanisms v, and all $t \in [0, 1]$,

$$W^v_{x_A, x_B; x'_A, x'_B}(t) = K^v_{x_A; x'_A, x'_B}(t)\delta(x_B, x'_B) \tag{51}$$

for some functions $K^v_{x_A; x'_A, x'_B}(t)$ that vary with changes to x'_B. This is *not* a solitary process (indeed, it violates the definition of subsystem processes). Yet it clearly leaves x_B unchanged. Moreover, *for any given initial distribution* $p_0(x_A, x_B)$, we can design the functions $K^v_{x_A; x'_A, x'_B}(t)$ so that the CTMC not only implements π, but also varies how the distribution at each x_A value evolves in intermediate times $t \in (0, 1)$, based on the associated x_B value, in such a way that zero system-wide EP is generated for the given initial distribution $p_0(x_A, x_B)$. It is this second capability that we exploit to reduce the EF below the value that can be achieved with a solitary process.

This is illustrated in the following example.

Example 9. *Consider a solitary process where* $X = X_A \times X_B$ *and both* X_A *and* X_B *are binary. Suppose that under that solitary process* x_A *undergoes bit erasure. So*

$$\begin{aligned}\pi(x^1_A, x^1_B | x^0_A, x^0_B) &= p(x^1_A | x^0_A)\delta(x^1_B, x^0_B) \\ &= \delta(x^1_A, 0)\delta(x^1_B, x^0_B)\end{aligned} \tag{52}$$

no matter what $p(x^0_A, x^0_B)$ *is. Suppose as well that initially,* $x_A = x_B$ *with probability 1, i.e., the two variables are initially perfectly correlated. So the Landauer loss, i.e., the drop in mutual information, i.e., the minimal system-wide EP, is* $\ln[2]$.

We can derive this value for the Landauer loss in more detail as follows. First recall that EF is additive under a solitary process. Since x_B *does not change, this means that the minimal EF of any solitary process that implements* π *is the same as the minimal EF of a process that implements the conditional distribution* $p(x^1_A | x^0_A)$ *on the initial distribution* $p(x^A_0)$. *This is* $\ln[2]$. *On the other hand, the entropy over the joint space is* $\ln[2]$ *both under the initial distribution and under the final distribution. So the minimal EF of a dynamics over the joint space is 0. Subtracting gives the value of the Landauer loss:* $\ln[2]$.

To illustrate how π *can be implemented with zero EF, suppose that the system were connected to a single heat bath at a constant temperature, and obeyed LDB. Suppose as well that the Hamiltonian*

over $X_A \times X_B$ evolved according to the kind of "quench then quasi-statically evolve" process described in [6, 95, 119]. That means that when the process begins, the Hamiltonian is instantaneously changed to

$$\begin{aligned} H_0(x_A, x_B) &= -\ln[p(x_A(0), x_B(0))] \\ &\propto -\ln\left[\delta(x_A(0), x_B(0)\right] \\ &= \begin{cases} \ln[2] & \text{if } x_A(0) = x_B(0) \\ \infty & \text{otherwise.} \end{cases} \end{aligned} \tag{53}$$

This quench happens so fast that the distribution over the states doesn't have time to change. So the system automatically starts at thermal equilibrium.

After this starting quench, the Hamiltonian quasi-statically evolves to (be arbitrarily close to)

$$\begin{aligned} H_1(x_A, x_B) &= -\ln\left[p(x_A(1), x_B(1))\right] \\ &\propto -\ln\left[\delta(x_A(1), 0)\right] \\ &= \begin{cases} \ln[2] & \text{if } x_A(1) = 0 \\ \infty & \text{otherwise.} \end{cases} \end{aligned} \tag{54}$$

while keeping an infinite energy barrier between $(x_A = 0, x_B = 0)$ and $(x_A = 0, x_B = 1)$, and an infinite energy barrier between $(x_A = 1, x_B = 0)$ and $(x_A = 1, x_B = 1)$. (In other words, keeping the rate matrix entries zero for all transitions that would change x_B.) This evolution implements the desired map π. Moreover, because the evolution is quasi-static, the system is always at thermal equilibrium, i.e., there is zero EP.[26,27]

While $H_0(x_A, x_B)$ is uniform in this process, $H_1(x_A, x_B)$ is not. Moreover, since the transformation from H_0 into H_1 is done quasi-statically, the evolution of the Hamiltonian is a continuous map (i.e., a homotopy). This means that for any rate matrix that obeys LDB for this changing Hamiltonian, there must be some $t \in (0, 1)$ at which the value of $K_{x_A; x'_A, x'_B}(t)$ for $x'_A = x'_B$ differs from its value for $x'_A \neq x'_B$. (Recall Eq. (42).) In contrast, the rate matrix of a solitary process cannot ever depend on whether $x'_A = x'_B$. So a solitary process cannot be implemented with this kind of zero EP quench-then-quasi-statically-evolve process.

[26] Note that if the initial distribution were not uniform, then this particular quench step would not result in the system starting at thermal equilibrium, which would cause EP to be generated as the system quasi-statically evolves. This illustrates the fact that any process can be thermodynamically reversible for at most one specific initial distribution, in general, due to mismatch cost. See Section V B.

[27] Note that the same results would hold if the dynamics uniformly randomized X_B rather than preserved its value exactly.

[69] also considers the case where we have an overall system M together with an associated set $\{A, A', \ldots\}$ of (potentially overlapping) subsystems of M. It is formally proven there that if we run a sequence of consecutive solitary processes for those subsystems, one after the other, then the system-wide Landauer cost and system-wide EP of the entire sequence are both additive over their values for running the separate solitary processes over the subsystems $A, A', \ldots$, in order.

In light of this, define the **machine (subsystem) EF** (resp., machine Landauer cost, machine EP, machine mismatch cost, machine residual EP) as the sum of the subsystem EFs (resp., subsystem Landauer costs, subsystem EPs, subsystem mismatch costs, subsystem residual EPs) incurred by successively running each of the subsystems, $A, A', \ldots$. Define the **(machine) Landauer loss** as the difference between the system-wide Landauer cost and the machine Landauer cost. Since system-wide Landauer cost is additive, machine Landauer loss equals the sum of the Landauer losses of each of the subsystems. It is shown in [69] that machine Landauer loss is the minimal EP of running the machine M, as one would expect.

C. Related literature

[114] contains some results that are closely related to the analysis of solitary processes in this paper and in [69]. In particular, [114] argues that the drop in mutual information between two independent subsystems of a composite system is the minimal EP of that composite system during any interval in which one of those subsystems does not change state. That argument assumes that the subsystem is coupled to a single heat bath and obeys LDB. It then uses general considerations of nonequilibrium statistical physic to make the case that "... {such a} subsystem matches the framework for an open driven system described in [15], and so the entropy production {is lower-bounded by the Landauer loss}".[28]

The analysis provided here (and in more detail in [69]) go beyond the scenarios considered in [114], in that it appies when there are multiple baths, and even when LDB does not hold. So it applies if in addition to a thermal reservoir, there are one or more chemical species reservoirs, as for example is often the case inside biological cells. Furthermore, the analysis here and in [69] considers mismatch cost and residual EP, in addition to Landauer cost. These considerations are crucial, for example, in analyzing the entropic costs of digital circuits (see Section X.)

On the other hand, [114] extends the analysis to information ratchets, viewing them as a sequence of solitary processes, one for each new symbol on an input tape that the information ratchet

[28] [114] uses the term "modularity dissipation" to mean what is called "Landauer loss" here and in [69].

processes. (See Section XIII A.) Moreover, the related discussion in [12] emphasizes that subsystem processes are ubiquitous in nature. In particular, they occur throughout biological systems. Accordingly, biological systems have nonzero Landauer losses in general, which fact alone prevents them from achieving the generalized Landauer bound.

[63] also contains work related to subsystem processes. Like [114], the analysis in [63] assumes a single heat bath, and LDB. However, the analysis in [63] is simpler than both the analysis here and the corresponding analysis in [114]. That is because rather than consider the implications on a composite system's minimal EP of a constraint on how that system is allowed to operate (as in Def. 7), the analysis in [63] makes an assumption about the relationship between the prior of the physical process underlying the composite system and the actual initial distribution of the states of the composite system. This allows [63] to directly exploit the decomposition of EP into mismatch cost and residual entropy in order to express the minimal EP as the drop in mutual information between the two subsystems.

X. ENTROPY DYNAMICS OF STRAIGHT-LINE CIRCUITS

Suppose we fix some map π from a finite space of inputs to a finite space of outputs along with an associated distribution over inputs. In general, the exact same gate g run at different parts of any circuit implementing π will have different distributions over its inputs, X_g^{IN}. As a result, the same gate implemented with a solitary process that is run at different parts of the circuit will incur different Landauer cost. This suggests that some of the circuits that implement π have greater (circuit) Landauer cost than the others, i.e., that some have greater Landauer loss than others. This in turn implies that some of those circuits have more EF than others. How precisely do the entropic costs of a circuit depend on the distribution over its inputs and on its wiring diagram, (V, E, F, X)?

To investigate this question, Section X A begins by introducing a broadly applicable model of the variables in straight-line circuits and how they evolve as the circuit is run. Special care is taken so that all of the details necessary to calculating entropic costs are made explicit. Section X B then introduces some extra, less broadly applicable modeling choices that simplify the analysis. Next, Section X C presents some of the simpler results that have been derived concerning the entropic costs of systems that are governed by this model. These results involve multi-divergence and cross multi-information, which as mentioned in Section III appear to be novel to the literature. This section ends with a high-level review of an analysis of a particular, fully specified real-world circuit.

It is worth noting that a high level discussion of some of the issues discussed in this section can be found in a prescient and insightful paper, written before the modern machinery of nonequilibrium statistical physics was developed [124].

A. How to model straight-line circuits

Each gate g in a circuit is modeled as a pair of random variables, with the associated state space written as $X_g = X_g^{IN} \times X_g^{OUT}$. The conditional distribution relating the ending state of X_g^{OUT} to the initial state of X_g^{IN} is the distribution given by the circuit specification (in the sense of Section IV B), which I write as $\pi_g(x_g^{OUT}|x_g^{IN})$. For simplicity, I restrict attention to Boolean formula circuits, so that all gates have outdegree 1 (see Section IV B). Accordingly, wolog we can assume that the gates run sequentially, in some convenient topological order respecting the circuit's wiring diagram. This in turn means that we can model running each gate g as a solitary process that updates the joint variable X_g while leaving all other variables in the circuit unchanged. (X_g is "X_A" in the notation of Section IX.) For analyzing the entropic costs of running such sequences of solitary processes in straight-line circuits, I will use the terms **circuit EF**, **circuit Landauer cost**, **circuit Landauer loss**, etc., to mean "machine EF", "machine Landauer cost", "machine Landauer loss", etc. (See Section IX B.)

I assume that we can model the solitary process updating any X_g as running a cyclic AO device over the state space X_g, where we identify the initial state of X_g^{IN} when g starts to run as the input of that AO device, and the ending state of X_g^{OUT} when g finishes its run as the output of that AO device (see Section VIII). Moreover, for simplicity I assume that standard accounting applies not just to the entire circuit, but also to any single gate (Section VII). So the initial value of X_g^{OUT} when g starts to run is some special initialized state (which I write as 0), and similarly the ending state of X_g^{IN} when g finishes is also some special initialized state (which I also write as 0).[29]

As briefly mentioned in Section V A, in real-world computers, a large fraction of the total EF occurs in the interconnects ("wires") between the gates. To allow the analysis to include such EF in the wires, it will be useful to model wires themselves as gates, i.e., nodes in a Bayes net. This means that the DAG (V, E) I will use to represent any particular physical circuit when calculating entropic costs is defined in terms of the DAG (V', E') of the associated wiring diagram, but differs from that DAG.

[29] This modeling assumption is violated by many real-world circuits, in which there is no reinitialization of the variables in a gate after it runs. In such circuits the entropic costs in a run arise when the "relic" states of variables, set during the preceding run of the circuit, get over-written. In general, to analyze the entropic costs of such circuits requires analyzing sequences of multiple runs rather than just a single run, which is why it is not pursued here.

To make this formal, define a **wire gate** as any gate in a circuit that has a single parent and implements the identity map. (So any wire gate w has $\pi_w(x_w|x_{\mathrm{pa}(w)}) = \delta(x_w, x_{\mathrm{pa}(w)})$; see Section IV B.) Suppose we are given a computational circuit whose DAG is (V', E'). The DAG that represents the associated physical circuit is constructed from (V', E'), in an iterative process. To begin, set (V, E) to be a copy of (V', E'). So there is a map $\eta(.)$ between all nodes and edges in the computational circuit's DAG and their corresponding members of the physical circuit's initial DAG, which is initially a bijection. We grow this initial DAG (V, E) by iterating the following procedure over all edges in E':

1. For each edge $e \in E'$ not yet considered, insert a new node v in the middle of the corresponding edge in E, $e = \eta(e')$.

2. Replace that single edge e with a pair of edges, one leading into v from the head node of the edge e, and one leading from v to the tail node of e.

Physically, each such new node introduced into (V, E) in this procedure represents a wire gate, one such wire gate for each edge in E'.

By the end of this iterative procedure, when we have fully constructed (V, E), the edges in E don't correspond to physical wires (unlike the edges in E'). Rather they indicate physical identity: an edge $e \in E$ going out from a non-wire gate g into a wire gate w is simply an indication that x_g^{OUT} is the same physical variable as the corresponding component of x_w^{IN}. Similarly, an edge $e \in E$ going into a non-wire gate g from a wire gate w is simply an indication that the (corresponding component of) x_g^{IN} is the same physical variable as x_w^{OUT}. Recall though that the solitary process that runs a non-wire gate g modifies x_g^{IN}. So that solitary process modifies x_w^{OUT} for each wire w leading into g. Similarly, it modifies $x_{w'}^{IN}$ for each wire w' leading out of g.

This means that when a gate $v \in V$ completes its run, having reinitialized its input, it has *also* reinitialized the corresponding output of its parent, regardless of whether v is a wire gate or a non-wire gate. So v plays the role of an "offboard" system for the computational devices at its parent gates, reinitializing the outputs of those parents. (See Section VII.) As a result, when all gates v in the (physical) circuit have finished, all gates v that are not outputs of the overall circuit are back in their initialized state, with probability 1, as are all input nodes to the circuit. So the circuit as a whole is cyclic.

In principle, this model also allows us to simultaneously run multiple computations through a single circuit, in staggered "waves". So long as there is a gap of at least two gates separating each

wave, both from that wave's predecessor and from its successor, we are guaranteed that there is no interference between the processes at the gates that are run for different waves.

I will sometimes use the terms "gate", "circuit", etc., to refer to physical systems with physical states, with the associated DAG (V, E). Other times I will use those terms to refer to the associated abstract mathematical conditional distributions in the DAG (V', E'). Such switching back and forth between (V', E') and the associated (V, E) will usually be implicit. The intended meaning of the terms "gate", "circuit", etc., will always be clear from context.

Note that since any wire gate implements the identity function, no matter what the distribution over the states of its parent it has both zero Landauer cost and zero mismatch cost. So all of the dependence of the EF of any real-world wire w on the distribution of inputs $p_{pa(w)}$ to that wire arises in the (linear) dependence of the residual EP on $p_{pa(w)}$. In addition, since a wire gate w implements the identity function, its islands are just the separate values of x_w. So the total EF generated by using a wire w is a linear function of the distribution over its inputs, i.e., it is a dot product of that distribution with an associated vector $\sigma_w^{min}(.)$ giving the residual EP of that wire for each of its possible inputs.

From now on I adopt the shorthand that for any gate g, p_g refers to the distribution of p_g^{OUT} before it has been re-initialized, but after it has been run. Similarly, I write $p_{pa(g)}$ to refer to the distribution over the states $x_{pa(g)}$ of (the output components of) the parents of g at the beginning of a run of g. So the actual distribution over the initial state of g, just before it runs, is $p_{pa(g)}$.

B. Simplifying assumptions for calculating minimal EF of a straight-line circuit

To simplify the analysis even further, in the rest of this section I make two more assumptions. First, I assume that the residual EP terms $\sigma_g^{min}(c)$ equals 0 for all gates g and associated islands c. This means in particular that from now on I ignore contributions to the EF from wire gates.

Second, I assume that we can choose the prior of any (non-wire) gate g to be anything we want. To motivate one way of choosing the prior at each gate, suppose we assume the input distribution for the entire circuit C is some specific distribution $q(x_{\text{IN}})$ (which I sometimes write as "q", for short). Then *if that assumption were correct*, we should set the prior distribution for each gate g simply by running the entire circuit C on inputs generated according to q to determine the distribution over the parents of g, i.e., by propagating q from V_{IN} to $\text{pa}(g)$. In this way any assumption of q specifies what prior we should build into each gate g in the circuit. I call a set of priors at the gates that are set this way from a shared input distribution q a set of **propagated priors**, and write them

as $\{q_{pa(g)}\}$.

In the analysis below, I allow for the possibility that the circuit will be used in more than one environment, e.g., with different users. Therefore I allow the priors to differ from the actual distributions. So the (propagated) priors at the gates can differ from the actual distributions at those gates.

It is important to note that both of the assumptions I am making are often violated in the real world. As mentioned above, in current computers the EF in the wires is comparable to that in the (non-wire) gates. However, that EF is *all* residual EP. So by ignoring residual EP, we ignore one of the major determiners of EF in current computer. Moreover, in many situations it will be easiest to mass-manufacture the gates, so that while we can vary the characteristics of any physical gate g that specifies the conditional distributions π_g, all gates g that implement the same conditional distribution π_g have the same prior, regardless of where they appear in a circuit. In this case the priors at the gates are not propagated priors, for any assumed distribution over inputs to the circuit. (See [69] for an analysis of the more general case, where neither of these two assumptions are made.)

C. Entropy dynamics of straight-line circuits

Suppose we are given a given conditional distribution of outputs given inputs, π, that is implemented by some circuit, and an input distribution, p, and some prior distribution over inputs, q. Then paralleling the definition of circuit Landauer loss, I define the **circuit mismatch loss** of running that circuit on that input distribution with that prior input distribution, as the difference between the mismatch cost of an AO device that implements π on p with a prior q over its inputs, and the circuit mismatch cost of a circuit that also implements π on p, and has the prior at each gate g set to the associated propagated prior $q_{pa(g)}$.

The following notation is motivated by the fact that residual EP is taken to equal zero for all devices:

Definition 8. *Let $C = (V, E, F, X)$ be a circuit, p a distribution over its inputs and q a prior over its inputs.*

1. *$\mathcal{Q}_{AO(C)}(p, q)$ is the total EF used to run $AO(C)$ on actual distribution p with prior distribution q.*

2. *$\mathcal{Q}_C(p,q)$ is the total EF used to run C on actual distribution p with propagated prior distribution $q_{\mathrm{pa}(g)}$ at each gate $g \in G$.*

3. *$\mathcal{E}_{AO(C)}(p,q) \equiv \mathcal{Q}_{AO(C)}(p,q) - \mathcal{Q}_{AO(C)}(p,p)$ is the total mismatch cost when running $AO(C)$ on actual distribution p with prior distribution q.*

4. *$\mathcal{E}_C(p,q) \equiv \mathcal{Q}_C(p,q) - \mathcal{Q}_C(p,p)$ is the total mismatch cost when running C on actual distribution p with propagated prior distribution $q_{\mathrm{pa}(g)}$ at each gate $g \in C$.*

5. *The circuit EF loss is*

$$\Delta\mathcal{Q}_C(p,q) := \mathcal{Q}_C(p,q) - \mathcal{Q}_{AO(C)}(p,q) \tag{55}$$

6. *The circuit Landauer loss is*

$$\begin{aligned}\Delta\mathcal{L}_C(p) &:= \mathcal{Q}_C(p,p) - \mathcal{Q}_{AO(C)}(p,p) \\ &= \Delta\mathcal{Q}_C(p,p)\end{aligned}$$

7. *The circuit mismatch loss is*

$$\begin{aligned}\Delta\mathcal{E}_C(p,q) &:= \mathcal{E}_C(p,q) - \mathcal{E}_{AO(C)}(p,q) \\ &= \Delta\mathcal{Q}_C(p,q) - \Delta\mathcal{L}_C(p)\end{aligned}$$

In [69] the following is proven:

Proposition 4. *For any Boolean formula C, the Landauer circuit loss for input distribution p is*

$$\Delta\mathcal{L}_C(p) = \mathcal{I}(p) - \sum_g \mathcal{I}(p_{\mathrm{pa}(g)})$$

Recall that Eq. (46) gives the Landauer loss of a subsystem process, as a drop in mutual information of the two subsystems as the process runs. The expression for $\Delta\mathcal{L}_C(p)$ in Proposition 4 can be seen as an extension of that result, to concern full circuits, comprising a sequence of multiple subsystem processes.

In words, Proposition 4 tells us that the difference between the Landauer cost of a formula and that of an equivalent AO device is a sum of multi-informations. One of those multi-informations is given directly by the input distribution. However, the other ones depend on the wiring diagram

of the circuit in addition to the input distribution. Intuitively, those other multi-informations reflect the fact that the greater the correlation among the inputs to any given gate g, the less information is contained in the joint distribution of those inputs, and therefore the less information is lost by running that gate. In turn, if every gate in a circuit actually does run without losing much information, then the amount of extra (minimal) EF due to using that circuit rather than an equivalent AO device is small.

The term $\mathcal{I}(p)$ in Prop. 4 can be seen as a "normalization constant", in the sense that for any pair of Boolean formulas C and C', both of which compute the same conditional distribution, the difference in their Landauer losses is just the difference in their Landauer costs,

$$\Delta\mathcal{L}_C(p) - \Delta\mathcal{L}_{C'}(p) = \sum_{g\in C'} \mathcal{I}(p_{\mathrm{pa}(g)}) - \sum_{g\in C} \mathcal{I}(p_{\mathrm{pa}(g)}) \tag{56}$$

This expression is independent of $\mathcal{I}(p)$, depending only on the multi-informations at the gates of the two circuits. In light of this, suppose we wish to design a circuit that implements a given Boolean formula f and that has minimal possible EF for some distribution p^{IN} over the domain of f, say subject to the constraint that we can only use gates in some specified universal set. Then Eq. (56) means that we must find the circuit C made out of such gates that implements f and that also minimizes the sum of the multi-informations at its gates, $\sum_{g\in C} \mathcal{I}(p_{\mathrm{pa}(g)})$, for the given distribution p^{IN}. This appears to be a nontrivial optimization, since making changes in one part of the circuit can affect the distributions that are input to the gates in a different part of the circuit, and therefore affect the total Landauer cost of the gates in that different part of the circuit.

It is proven in [69] that circuit Landauer loss cannot be negative. This provides a major advantage to using an AO device rather than a circuit. Unfortunately, there are major engineering difficulties which prevent us from building AO devices to implement the kinds of functions implemented in real-world circuits, due to the huge state spaces of such real-world circuits.

The following is also proven in [69]:

Proposition 5. *For any formula C, the circuit mismatch loss for actual input distribution p and prior input distribution q is*

$$\Delta\mathcal{E}_C(p,q) = -\mathcal{D}(p\|q) + \sum_g \mathcal{D}(p_{\mathrm{pa}(g)}\|q_{\mathrm{pa}(g)})$$

(Compare to Proposition 4.) Interestingly, mismatch loss can be negative. In fact, the sum of Landauer loss and mismatch loss can be less than zero. This suggests that in some situations we

would actually use less EF if we run a circuit rather than an equivalent AO device to implement a given Boolean formula, if our assumption for the input distribution q differs from the actual input distribution, p. To see how this phenomenon might be exploited, suppose that just like in robust optimization, we do not presume that we know the actual distribution p, and so cannot set $q = p$. However, we feel confident in saying that p lies within a ball of radius K of our guess for the actual input distribution, $\hat{p}$, i.e., that $||\hat{p}, p|| < K$ for some appropriate distance measure $||\cdot, \cdot||$. Then we might want to set $q = \hat{p}$, and choose between a circuit and an equivalent AO device that both use that prior based on which one would result in less total EF for any p such that $||q, p|| < K$.[30]

[69] contains other results not reproduced here, for the case where gates can have nonzero residual EP. The analysis in [69] also covers circuits with noisy gates. In addition, the analysis in that paper cover circuits with gates that have outdegree greater than 1. Finally, [69] also derives some sufficient conditions for circuit Landauer loss to be positive, some sufficient conditions for Landauer loss to be negative, and similar sufficient conditions for circuit mismatch loss.

The thermodynamics of circuits is also considered in [63]. Superficially, the analysis in that paper is similar to the analysis summarized here. In particular, Prop. 4 above has a similar functional form to Eq. 26 in [63], which also involves a multi-information (though without using that term). However, the analysis in [63] concerns a different kind of system from the circuits considered here. Viewed as a circuit, the system considered in [63] is a set of N disconnected gates, working in parallel, never combining, but with statistical correlations among their inputs. Eq. 26 in [63] concerns the mismatch cost that would arise for such a system if we used propagated priors and took $q = p$.

D. Entropy dynamics in a transition detector circuit

The analysis in [125] concerns the entropic costs of a "transition detector". This is a device that receives a string of bits, one after the other, and iteratively determines for each new bit whether it is the same or different from the previous bit that the device received. Although the authors describe the device they analyze as a finite state automaton, they only consider a single pass of its operation, in which it only decides whether one second bit differs from the first bit. This reduces their machine to a straight-line circuit with a two-bit input space.

One noteworthy contribution of [125] is the level of detail in the circuit they analyze. They describe the variables in this circuit as follows:

[30] There are some subtleties in applying this reasoning to real-world circuits, which arise if we wish to "compare apples with apples". See [69].

> "The processor utilizes two general purpose registers A and B, a four-function arithmetic logic unit (ALU), a two-word internal "scratchpad" memory $M = M_0 M_1$, that also functions as an output buffer, four multiplexers for signal routing, and input and output buffers. ... {In addition there are} a 16-word instruction memory, a 4-bit program clock (PC)–with a hardware provision for halting the clock if and when the instruction at address PC(1111) is reached– and control logic that decodes each instruction and generates all control signals required to appropriately configure the data path, perform register operations, adjust the program clock on jump instructions, and enable memory access and I/O."

The authors then specify the control logic and associated data path of their circuit in the same level of detail.

The authors compare an "instruction-level analysis" (ILA) of the entropic cost of running the circuit to an "architecture-level analysis" (ALA) of those costs, under the assumption that the input bits are generated IID. It appears that their analysis concerns Landauer cost.[31] It also appears that their ILA is a way to calculate the Landauer cost of an AO device that implements the transition detector. On the other hand, the ALA appears to be a way of calculating the circuit Landauer cost.

Under this interpretation of the analysis in [125], the difference of the two entropic costs that they calculate is the Landauer loss of a single pass through the particular circuit they define. In agreement with the results in Section X C, they find that depending on the probability distribution of the input bits, the Landauer loss can be positive, but is never negative.

The interested reader may also want to consult [126], which is another article, involving the same authors, that considers the thermodynamics of specific straight-line circuits that are modeled in great detail.

XI. ENTROPY DYNAMICS OF LOGICALLY REVERSIBLE CIRCUITS

An interesting body of research concerning "reversible circuits" has grown out of the early work by Landauer and Bennett, in isolation from the redent breakthroughs in nonequilibrium statistical physics. This research assumes that one is presented with a conventional circuit C made of logically *ir*reversible gates which implements some logically irreversible function f, and wants to construct

[31] It is hard to be completely sure of the physical meaning of the quantities the authors analyzed, since they perform their analysis using a semi-formal "referential approach" they developed in earlier work that is not used by others in the literature, and that is not formulated in terms of modern nonequilibrium statistical physics. At a minimum though, no mismatch cost or residual entropy creation terms appear in their analysis.

a logically *reversible* circuit, C', that emulates C. The starting point for this research is the observation that we can always create such an emulating circuit, by appropriately wiring together a set of logically reversible gates (e.g., Fredkin gates) to create a circuit C' that maps any input bits $x^{IN} \in X^{IN}$ to a set of output bits that contain both $f(x^{IN})$ and a copy of x^{IN} [127–130]. Tautologically, the entropy of the distribution over the states of C' after this map has completed is identical to the entropy of the initial distribution over states. So the Landauer cost is zero, it would appear. This has led to claims in the literature suggesting that by replacing a conventional logically irreversible circuit with an equivalent logically reversible circuit, we can reduce the "thermodynamic cost" of computing $f(x^{IN})$ to zero.

This line of reasoning should be worrisome. As mentioned, we now know that we can directly implement any logically irreversible map $x^{IN} \rightarrow f(x^{IN})$ in a thermodynamically reversible manner. So by running such a direct implementation of f in reverse (which can be done thermodynamically reversibly), we would extract heat from a heat bath. If we do that, and then implement f forward using a logically reversible circuit, we would return the system to its starting distribution, seemingly having extracting heat from the heat bath, thereby violating the second law.

As it turns out, there *are* some thermodynamic advantages to using a logically reversible circuit rather than an equivalent logically irreversible circuit. However, there are also some disadvantages to using logically reversible circuits. Moreover, the advantages of logically reversible circuits cannot be calculated simply by counting "the number of bit erasures" performed by the equivalent logically irreversible circuit. In the following two subsections I elaborate these relative advantages and disadvantages of using reversible circuits, in order to illustrate the results presented in the sections above.

Before doing that though, in the remainder of this subsection I present some needed details concerning logically reversible circuits that are constructed out of logically reversible gates. One of the properties of logically reversible gates that initially caused problems in designing circuits out of them is that running those gates typically produces "garbage" bits, to go with the bits that provide the output of the conventional gate that they emulate. The problem is that these garbage bits need to be reinitialized after the gate is used, so that the gate can be used again. Recognizing this problem, [127] shows how to avoid the costs of reinitializing any garbage bits produced by using a reversible gate in a reversible circuit C', by extending C' with yet more reversible gates (e.g., Fredkin gates). The result is an **extended circuit** that takes as input a binary sting of input data x, along with a binary string of "control signals" $m \in M$, whose role is to control the operation of the reversible gates in the circuit. The output of the extended circuit is a binary string of the

desired output for input x^{IN}, $x^{OUT} = f(x^{IN})$, together with a copy of m, and a copy of x^{IN}, which I will write as x^{IN}_{copy}. So in particular, none of the output garbage bits produced by the individual gates in the original, unextended circuit of reversible gates still exists by the time we get to the output bits of the extended circuit.[32]

While it removes the problem of erasing the garbage bits, this extension of the original circuit with more gates does not come for free. In general it requires doubling the total number of gates (i.e., the circuit's size), doubling the running time of the circuit (i.e., the circuit's depth), and increasing the number of edges coming out of each gate, by up to a factor of 3. (In special cases though, these extra cost can be reduced, sometimes substantially.)

A. Reversible circuits compared to computationally equivalent all-at-once devices

In general, there are many different "basis sets" of allowed gates we can use to construct a conventional (logically irreversible) circuit that computes any given logically irreversible function f. Moreover, even once we fix a set of allowed gates, in general there are an infinite number of logically irreversible circuits that implement f using that set of gates. Due to all this flexibility, we need to clarify precisely what "logically irreversible circuit" we wish to compare to any given extended circuit that implements the same function f as that circuit.

One extreme possibility is to compare the extended circuit to a single, monolithic gate that computes the same function, and which distinguishes input variables from output variables. In other words, we could compare the extended circuit to a physical system that directly maps $(x^{IN}, \vec{0}^{OUT}) \rightarrow (x^{IN}, f(x^{IN}))$. However, this map is logically reversible, just like the extended circuit, and so not of interest for the comparison.

A second possibility is to compare the extended circuit to an AO device with a state space X that directly maps $x \in X \rightarrow f(x) \in X$, without distinguishing input variables and output variables. Such a map is *not* logically reversible, but (as mentioned above) can be implemented with a thermodynamically reversible system, whatever the initial distribution over X. If we implement f with an AO device, then the minimal EF we must expend to calculate f is the drop in entropy of the distribution over X as that distribution evolves according to f. This drop is nonzero (assuming

[32] More precisely, in one popular form of reversible circuits, a map $f : X^{IN} \rightarrow X^{OUT}$ is implemented in several steps. First, in a "forward pass", the circuit made out of reversible gates sends $(x^{IN}, m, \vec{0}^{GARBAGE}, \vec{0}^{OUT}) \rightarrow (x^{IN}, m, m', f(x^{IN}))$, where m' is the set of "garbage bits", $\vec{0}^{OUT}$ is defined as the initialized state of the output bits, and similarly for $\vec{0}^{GARBAGE}$. After completing this forward pass, an offboard copy is made of x^{OUT}, i.e., of $f(x^{IN})$. Then the original circuit is run "in reverse", sending $(x^{IN}, m, m', f(x^{IN})) \rightarrow (x^{IN}, m, \vec{0}^{GARBAGE}, \vec{0}^{OUT})$. The end result is a process that transforms the input bit string x^{IN} into the offboard copy of $f(x^{IN})$, together with a copy of x^{IN} (conventionally stored in the same physical variables that contained the original version of x^{IN}), all while leaving the control bit string m unchanged.

f is not logically invertible). This would seem to mean that there is an advantage to using the equivalent extended circuit rather than the AO device, since the minimal EF with the extended circuit is zero.

However, we must be careful to compare apples to apples. The number of information-carrying bits in the extended circuit after it completes computing f is $\log|X^{IN}|+\log|X^{OUT}|+\log|M|$. The number of information-carrying bits in the AO device when it completes is just $\log|X^{OUT}|$. So strictly speaking, the two systems implement different functions, that have the same domains but different codomains.

This means that the entropic costs of answer-reinitializing the two circuits (i.e., reinitializing the codomain variables) will differ. In general, the Landauer cost and mismatch cost of answer-reinitialization of an extended circuit will be greater than the corresponding answer-reinitialization costs of an equivalent AO device. This is for the simple reason that the answer-reinitialization of the extended circuit must reinitialize the bits containing copies of x and m, which do not even exist in the AO device.

Phrased differently, if we allow the extended circuit to keep a copy of x^{IN}, rather than erase it, then to compare apples to apples, we should also not impose all the strictures of standard accounting to the AO device, and allow the AO device to also forego erasing x^{IN}. However, that would change the Landauer cost we ascribe to running the AO device from $S(X^{IN}) - S(X^{OUT})$ to the *negative* value $-S(X^{OUT})$. (Recall from the discussion at the end of Section VII B that if any copies of the input are allowed to persist after the computation ends, then we can even get negative Landauer cost.) It is worth emphasizing that this importance of reinitializing the copy of x^{IN} was recognized even in the early analyses based on the original formulation of Landauer's bound; it is the primary motivation for one of the most sophisticated of those early analyses, which is discussed in detail in Section XIV D.

To be more quantitative, first, for simplicity, assume that the initial distribution over the bits in the extended circuit that encode m is a delta function. (This would be the case if we do not want the physical circuit to implement a different computation from one run to the next, so only one vector of control signals m is allowed.) This means that the ending distribution over those bits is also a delta function. The Landauer cost of reinitializing those bits is zero, and assuming that we perform the reinitialization using a prior that equals the delta function over m, the mismatch cost is also zero. So assuming the residual EP of reinitialization those bits containing a copy of m is zero, we can we can ignore those bits from now on.

To proceed further in our comparison of the entropic costs of the answer reinitialization of an

AO device with those of an equivalent extended circuit, we need to specify the detailed dynamics of the answer-reinitialization process that is applied to the two devices. Both the AO device and the equivalent extended circuit have a set of output bits that contain $f(x^{IN})$ that need to be reinitialized, with some associated entropic costs. In addition though, the extended circuit needs to reinitialize its ending copy of x^{IN}, whereas there is no such requirement of the equivalent AO device. To explore the consequences of this, I now consider several natural models of the answer-reinitialization:

1) In one model, we require that the answer-reinitialization of the circuit is performed within each output bit g itself, separately from all other variables. Define $Fr(C)$ to mean an extended circuit that computes the same input-output function f^C as a conventional circuit C, and define $AO(C)$ similarly. Assuming for simplicity that the residual entropy of reinitializing all output bits is zero, the EF for the answer-reinitialization of $Fr(C)$ using such a bit-by-bit process is

$$\mathcal{Q}_{C'}(p,q) = \sum_{g \in V^{\rm OUT}} S(p_g \| q_g) \tag{57}$$

where $V^{\rm OUT}$ indicates the set of all bits containing the final values of x^{OUT} and x^{IN}_{copy}.

Using gate-by-gate answer-reinitialization, the EF needed to erase the output bits containing $f^C(x^{IN})$ is the same for both $AO(C)$ and $Fr(C)$. Therefore the additional Landauer cost incurred in answer-reinitialization due to using $Fr(C)$ rather than $AO(C)$ is the Landauer cost of erasing the output bits in $Fr(C)$ that store x^{IN}_{copy},

$$\Delta S_{Fr(C),C}(p) \coloneqq \sum_{v \in V_{IN}} S(p_v) \tag{58}$$

where I write "$v \in V_{IN}$" to mean the output bits that contain x^{IN}_{copy}, and p_v to mean the ending marginal distributions over those bits. Similarly, the difference in mismatch cost is

$$\Delta D_{Fr(C),C}(p,q) \coloneqq \sum_{v \in V_{IN}} D_v(p^v \| q^v) \tag{59}$$

where q_v refers to a prior used to reinitialize the output bits in $.v \in V_{IN}$.

However, independent of issues of answer-reinitialization, the Landauer cost of implementing a function using an AO device that is optimized for an initial distribution $p_{\rm IN}$ can be bounded as follows:

$$S(p_{\rm IN}) - S(f^C p_{\rm IN}) \leq S(p_{\rm IN})$$

$$\leq \sum_{v \in V_{\rm IN}} S_v(p_v)$$
$$= \Delta S_{Fr(C),C}(p) \tag{60}$$

Combining this with Eq. (58) shows that under gate-by-gate answer-reinitialization, the *total* Landauer cost of implementing a function using an AO device — including the costs of reinitializing the gates containing the value $f^C(x^{IN})$ — is upper-bounded by the *extra* Landauer cost of implementing that same function with an equivalent extended circuit, i.e., just that portion of the cost that occurs in answer-reinitializing the extra output bits of the extended circuit. This disadvantage of using the extended circuit holds even if the equivalent AO device is logically irreversible. So as far as Landauer cost is concerned there is no reason to consider using an extended circuit to implement a logically irreversible computation with this first type of answer-reinitialization.

On the other hand, in some situations, the mismatch cost of running the AO device will be *greater* than the mismatch cost of the answer-reinitialization of x^{IN}_{copy} in the equivalent extended circuit. This illustrated in the following example:

Example 10. *Suppose that the input to the circuit consists of two bits, a and b, where the actual distribution over those bits, p, and prior distribution over those bits, q, are:*

$$p(x_b) = \delta(x_b, 0)$$
$$p(x_a | x_b = 0) = 1/2 \qquad \forall x_a$$
$$q(x_b) = \delta(x_b, 1)$$
$$q(x_a | x_b = 0) = 1/2 \qquad \forall x_a$$
$$q(x_a | x_b = 1) = \delta(x_a, 0)$$

Suppose as well that f^C is a many-to-one map. Then plugging in gives

$$D(p_{\rm IN} \| q_{\rm IN}) - D(f^C p_{\rm IN} \| f^C q_{\rm IN}) = D(p_{\rm IN} \| q_{\rm IN})$$
$$> \sum_{v \in V_{\rm IN}} D(p_v \| q_v)$$

This sum equals the mismatch cost of the answer-reinitialization of x^{IN}_{copy}, which establishes the claim.

However, care should be taken in interpreting this result, since there are subtleties in comparing mismatch costs between circuits and AO devices, due to the need to compare apples to apples (see discussion of this point in [69]).

2) A second way we could answer-reinitialize an extended circuit involves using a system that simultaneously accesses *all* of the output bits to reinitialize x^{IN}_{copy}, including the bits storing $f(x^{IN})$.

To analyze this approach, for simplicity assume there are no restrictions on how this reinitializing system operates, i.e., that it is an AO device. The Landauer cost of this type of answer-reinitialization of x^{IN}_{copy} is just $S(p(X^{IN}|X^{OUT})) - \ln[1]$, since this answer-reinitialization process is a many-to-one map over the state of x^{IN}_{copy}. Assuming f^C is a deterministic map though, by Bayes' theorem

$$S(p(X^{IN}|X^{OUT})) = S(p(X^{IN})) - S(p(X^{OUT})) \tag{61}$$

So in this type of answer-reinitialization, the extra Landauer cost of the answer-reinitialization in the extended circuit that computes f^C is identical to the total Landauer cost of the AO device that computes the same function f^C. On the other hand, in this type of answer-reinitialization process the mismatch cost of the extended circuit may be either greater or smaller than that of the AO device, depending on the associated priors.

3) A third way we could answer-reinitialize x^{IN}_{copy} in an extended circuit arises if, after running the circuit, we happened upon a set of initialized external bits, just lying around, as it were, ready to be exploited. In this case, after running the circuit, we could simply swap those external bits with x^{IN}_{copy}, thereby answer-reinitializing the output bits at zero cost.

Arguably, this is more sleight-of-hand than a real proposal for how to re-initialize the output bits. Even so, it's worth pointing out that rather than use those initialized external bits to contain a copy of x^{IN}_{copy}, we could have used them as an information battery, extracting up to a maximum of $k_B T \ln 2$ from each one by thermalizing it. So the opportunity cost in using those external bits to reinitialize the output bits of the extended circuit rather than use them as a conventional battery is $|V_{IN}| k_B T \ln 2$. This is an upper bound on the Landauer cost of implementing the desired computation using an AO device. So again, as far as Landauer cost is concerned, there is no advantage to using an extended circuit to implement a logically irreversible computation with this third type of answer-reinitialization.

There are other schemes for reinitializing x^{IN}_{copy} of course. In particular, see the discussion in Section XIV D for a review of a particularly sophisticated such scheme.

Summarizing, it is not clear that there is a way to implement a logically irreversible function

with an extended circuit built out of logically reversible gates that reduces the Landauer cost below the Landauer cost of an equivalent AO device. The effect on the mismatch cost of using such a circuit rather than an AO device is more nuanced, varying with the priors, the actual distribution, etc.

B. Reversible circuits compared to computationally equivalent irreversible circuits

I now extend the analysis, from comparing the entropic costs of an extended circuit to those of a computationally equivalent AO device, to also compare to the costs of a computationally equivalent conventional circuit, built with multiple logically irreversible gates. As illustrated below, the entropic costs of the answer-reinitialization of a conventional circuit (appropriately modeled) are the same as the entropic costs of the answer-reinitialization of a computationally equivalent AO device. So the analysis of the preceding subsection gives us the relationship between the answer-reinitialization entropic costs of conventional circuits and those of computationally equivalent extended circuits. In particular, the minimal EF required to answer-reinitialize a conventional circuit is in general lower than the minimal EF required to answer-reinitialize a computationally equivalent extended circuit.

Accordingly, in this subsection I focus instead on comparing the entropic costs of running conventional circuits, before they undergo any answer-reinitialization, with the entropic costs of running computationally equivalent extended circuits, before *they* undergo any answer-reinitialization. While the full analysis of the entropic costs of running conventional circuits is rather elaborate [69], some of the essential points can be illustrated with the following simple example.

Suppose we have a system that comprises two input bits and two output bits, with state space written as $X = X_1^{IN} \times X_2^{IN} \times X_1^{OUT} \times X_2^{OUT}$. Consider mapping the input bits to the output bits by running the "parallel bit erasure" function. Suppose that while doing that we simultaneously reinitialize the input bits x_1^{IN} and x_2^{IN}, in preparation for the next run of the system on a new set of inputs. So assuming both of the output bits are initialized before the process begins to the erased value 0, the state space evolves according to the function $f : (x_1^{IN}, x_2^{IN}, 0, 0) \to (0, 0, 0, 0)$. (See Example 6 for an alternative way of doing parallel bit erasure, with a system that does not differentiate input and output variables.)

Consider the following three systems that implement this f:

1. An AO device operating over $X_1^{IN} \times X_2^{IN} \times X_1^{OUT} \times X_2^{OUT}$ that directly implements f;

2. A system that implements f using two bit-erasure gates that are physically isolated from one another, as briefly described in Example 6. Under this model the system first uses one bit-erasure gate to send $(X_1^{IN}, X_1^{OUT}) = (x_1^{IN}, 0) \rightarrow (0, 0)$, and then uses a second bit-erasure gate to apply the same map to the second pair of bits, (X_2^{IN}, X_2^{OUT}).

   The requirement that the gates be physically isolated means that the rate matrix of the first gate is only allowed to involve the pair of bits (X_1^{IN}, X_1^{OUT}), i.e., it is of the form $W_{x_1^{IN}, x_1^{OUT}; (x_1^{IN})', (x_1^{OUT})'}(t)$. So the dynamics of (x_1^{IN}, x_1^{OUT}) is independent of the values of the other variables, (x_2^{IN}, x_2^{OUT}). Similar restrictions apply to the rate matrix of the second gate. (So in the language of Section IX, since the two gates run sequentially, the each run a "solitary process".)

3. A system that uses two bit-erasure gates to implement f, just as in (2), but does *not* require that those gates run in sequence and that they be physically isolated. In other words, the rate matrix that drives the first bit-erasure gate as it updates the variables (x_1^{IN}, x_1^{OUT}) *is* allowed to do so based on the values (x_2^{IN}, x_2^{OUT}), and vice-versa. Formally, this means that the joint rate matrix of the entire system is of the form

$$W_{x_1^{IN}, x_1^{OUT}; (x_1^{IN})', (x_1^{OUT})'; (x_2^{IN})', (x_2^{OUT})'}(t)\, \delta(x_2^{IN}, (x_2^{IN})')\, \delta(x_2^{OUT}, (x_2^{OUT})') + W_{x_2^{IN}, x_2^{OUT}; (x_1^{IN})', (x_1^{OUT})'; (x_2^{IN})', (x_2^{OUT})'}(t)\, \delta(x_1^{IN}, (x_1^{IN})')\, \delta(x_1^{OUT}, (x_1^{OUT})') \tag{62}$$

Models (2) and (3) both represent a conventional circuit made out of two gates that each implement logically-irreversible functions. However, they differ in whether they only allow physical coupling among the variables in the circuit that are logically needed for the circuit to compute the desired function (model (2)), or instead allow arbitrary coupling, e.g., to reduce entropic costs (model (3)). Moreover, the set of rate matrices allowed under model (3) is a strict superset of the set of rate matrices of all subsystem processes that implements$f$, i.e., we have more freedom to reduce EF using model (3) than we would with any subsystem process (see Definition 7). On the other hand, the set of allowed rate matrices under model (1) is a strict superset of the set of allowed rate matrices under model (3).

To analyze the consequences of these differences, first consider the Landauer cost of model (1), which we can expand as

$$S(p_0(X_1^{IN}, X_2^{IN}, X_1^{OUT}, X_2^{OUT})) - S(\hat{f}_{1,2}\, p_0(X_1^{IN}, X_2^{IN}, X_1^{OUT}, X_2^{OUT}))$$

$$= S(p_0(X_1^{IN}, X_2^{IN}, X_1^{OUT}, X_2^{OUT}))$$
$$= S(p_0(X_1^{IN}, X_2^{IN})) \tag{63}$$

where $\hat{f}_{1,2}$ is the conditional distribution implementing the parallel bit erasure, so that $\hat{f}_{1,2}\, p_0(x)$ is the ending distribution, which is a delta function centered at $(0, 0, 0, 0)$.

Next, assume that both of the gates in model (2) are thermodynamically reversible when considered by themselves, isolated from the rest of the universe, i.e., that their subsystem EPs are both zero. Then the minimal EF needed to run the first of those gates is

$$S(p_0(X_1^{IN}) - S(\hat{f}_1\, p_0(X_1^{IN}, X_1^{OUT})) = S(p_0(X_1^{IN})) \tag{64}$$

Similarly, the minimal EF needed to run the second gate is $S(p_0(X_2^{IN})$.

Combining, we see that the difference between {the minimal EF needed to run a conventional circuit constructed as in model (2)} and {the minimal EF needed to run a computationally equivalent AO device (model (1))} is

$$S(p_0(X_1^{IN}) + S(p_0(X_2^{IN}) - S(p_0(X_1^{IN}, X_2^{IN})) \tag{65}$$

This is just the initial mutual information between X_1^{IN} and X_2^{IN}.[33] So the minimal EF needed to run model (2) will exceed the minimal EF needed to run model (1) whenever X_1^{IN} and X_2^{IN} are statistically coupled under the initial distribution, p_0.

On the other hand, because of the increased flexibility in their rate matrices, it is possible that the bit-erasure gates in model (3) each achieve zero EP *even when considered as systems operating over the full set of four bits*. So each of those bit-erasure gates is thermodynamically reversible even when considered in the context of the full system. As a result, running the circuit defined in model (3) requires the same minimal EF as running an AO device. (See also Example 9.) So in general, the minimal EF needed to run to the conventional circuit defined in model (3) is less than the minimal EF needed to run to the conventional circuit defined in model (2).

Summarizing, the minimal total EF (including both the EF needed to run the system and to reinitialize it at the end of the run) that is needed by the circuit defined by model (2) exceeds the minimal total EF needed by either the equivalent AO device (model (1)) or the equivalent conventional circuit defined by model (3). Those two differences in those minimal EF's both

[33] One could reach the same conclusion by using the fact that machine Landauer loss of a sequence of solitary processes is additive over those process (see end of Section IX B), the fact that the Landauer loss of each solitary process is the drop in mutual information between the two subsystems during that process (see Eq. (46)), and the fact that the ending entropy of a system that erases a bit is 0.

equal the mutual information of the inputs bits under p_0. In turn, the minimal EFs to run either model (1) or model (3) exceeds the minimal EF needed to run an equivalent extended circuit, by $S(p_0(X_1^{IN}, X_2^{IN}))$. However, the minimal *total* EF of the models (1) and (3) will in general be no greater than the minimal total EF of the extended circuit, and may be smaller (depending on the details of the answer-reinitialization process in the extended circuit).

On the other hand, as a purely practical matter, constructing a conventional circuit as in (3) for circuits substantially larger than parallel bit-erasures may be quite challenging; to do so requires that identify all sets of variables that are *statistically* coupled, at any stage of running the circuit, and make sure that our gates are designed to *physically* couple those variables. There are no such difficulties with constructed an extended circuit. Another advantage of an extended circuit is that no matter what the true distribution p_0 is, an extended circuit has zero mismatch cost, since there is no drop of KL divergence between p_0 and *any* q_0 under a logically reversible dynamics. In contrast, all three models (1) - (3) can have nonzero mismatch cost, in general.

As yet another point of comparison, an extended circuit will often have far more wires than an equivalent conventional circuit. And as mentioned above, the residual EP generated in wires is one of the major sources of EF in modern digital gates. So even in a situation where a conventional circuit has nonzero mismatch cost, when the EF generated in the wires is taken into account, there may be no disadvantage to using that conventional circuit rather a computationally equivalent extended circuit.

Clearly there is a rich relationship between the detailed wiring diagram of a conventional logically irreversible circuit, the procedure for answer-reinitializing the outputs of a computationally equivalent extended circuit, the distribution over the input bits of those circuits, and how the aggregate entropic costs of those two circuits compare. Precisely delineating this relationship is a topic for future research.

XII. ENTROPY DYNAMICS OF FINITE AUTOMATA

A. Entropy dynamics of FAs in a steady state

There is some work in the literature that presents calculations related to the thermodynamics of FAs. For example, [131] considers the thermodynamics of systems that can be defined as word-based deterministic FAs with no accepting states whose input symbols are generated in an IID manner, and which have no word-delimiting input symbol. (Arguably this last property in particular

distinguishes the systems they consider from what computer scientists typically call "finite state automata".) So the "input word" in the systems they consider is actually a single, infinite string of symbols, and their systems undergo a single, infinitely long, run. This allows them to avoid including a counter variable in the model, despite their use of a word-based FA. Instead, the time index on the rate matrix can directly encode which input symbol is being processed at any given time.

Rather than consider finite time behavior, they consider the asymptotic limit, presuming the system reaches a (unique) steady state. In keeping with this, they do not require that the system be cyclic in any sense. Indeed, the successive input symbols in the input word are not reinitialized as the computation proceeds, so if one *were* to stop their system at a finite time, and try to give it a new input string, entropic costs would be incurred which are not considered in their model. Similarly, their model does not designate some variable as being the output of the computer. So they do not consider the issue of how such an output might be copied out of the system at some finite time. Given this lack of an output and their not reinitializing the input, they do not use any convention akin to standard accounting to determine how to ascribe entropic costs incurred when one run ends and another begins.

Their model does not involve solitary processes, but instead considers AO devices. (Note that since they consider the asymptotic limit, this means that they implicitly allow the interaction Hamiltonian between the computational state of the FA and the input state to involve the arbitrarily large set of variables giving all earlier input symbols.) Moreover, since they focus on the steady state, the entropy of the computational state of the system doesn't change in an update step. Since they don't require the input symbols to be reinitialized once the are processed, the joint entropy of the (infinite) string of input symbols does not change in an update step either. So the Landauer cost in any single update step is due to the loss of information between earlier inputs and the current computational state. Given the constancy of the marginal entropies of those earlier symbols and of the current computational state, this loss of information in an update step is exactly the change in the mutual information between the joint state of the earlier input symbols and the computational state in that update step. Finally, since they consider an AO device, minimal EP is zero. So the minimal EF per update step is just the Landauer cost, i.e., it equals this change in mutual information.

This is their primary result. They derive it in a quantum mechanical context, but the same analysis holds for classical systems undergoing Markovian dynamics. As a final comment, instead of viewing the systems considered in [131] as a version of FAs, those systems can be viewed as a

variant of information ratchets, with the added restriction that the ratchet is not allowed to change the state of an input symbol after reading it. Accordingly, the reader may want to compare the analysis in [131] with the analyses in Section XIII.

B. Entropy dynamics of FAs with thermal noise

Another paper related to the thermodynamics of FAs is [132]. In that paper the authors consider deterministic finite automata that are subject to thermal noise during their operation. The paper is careful to introduce the complete computer science definition of an FA, including the fact that input words having finite lengths. However, they only consider a single iteration of an FA. So they don't need to explicitly ensure that the model of an FA that they analyze is a cyclic device. Nor do they need to consider the problems that arise from the fact that the duration of a run with an FA is a random variable. (Recall Section VII C.) They also do not consider the issues related to the entropic costs of copying the output of the FA offboard and/or the entropic costs of copying in a new input.

Like the current paper, [132] uses stochastic thermodynamics to perform their analysis. However, in contrast to most of the stochastic thermodynamics literature, they stipulate that the underlying rate matrix is time-homogeneous during each iteration of the system. As a result, to get any dynamics, they assign different energy levels to each of the states of the FA, relying on thermal relaxation to drive the dynamics. (In contrast, as mentioned in Section VII A, the convention in the literature is to stipulate that energy levels are identical at the beginning of each new iteration of an information-processing system, in order to focus on the information processing behavior of the system.)

Although the relationship is not exact, it seems that in order to implement arbitrary (deterministic) update functions in their model of FAs, [132] exploits the same kind of "partially hidden" state space construction discussed in Sections VIII and XV. As a cautionary comment, the reader should be aware that [132] uses some idiosyncratic terminology. For example, they refer to the thermal relaxation of a two-energy system down to the lower energy state as a "bit flip".

XIII. ENTROPY DYNAMICS OF INFORMATION RATCHETS

Suppose that we know that inputs to an information ratchet are generated according to a particular N'th order Markov chain, and we know the prior distribution over the first N inputs to

the ratchet. Even with this information, we do know the joint distribution over strings of $N+1$ successive inputs to the ratchet — that distribution will depend on how long that Markov chain has been running. This means that to analyze the entropy dynamics of an information ratchet with inputs generated according N'th order Markov chain, we have to also specify how long it has been running. This substantially complicates the analysis. The problem is only compounded if we don't know N – or if in fact the stochastic process generating the inputs is not a Markov process of *any* finite order.[34]

The natural way to analyze such scenarios is to take the infinite time limit, assuming that the inputs are generated according to a stationary process. In a series of papers [13, 15–18, 114, 133], Boyd, Mandal, Riechers, Crutchfield and others have begun pursuing this line of research. The focus in these papers has not been the behavior of an information ratchet with an arbitrary given update rule of its computational states, running on an arbitrary given input data stream. Instead, these papers primarily concern the case where the update rule is optimized for the given input data stream.

Given the challenging nature of analyzing the thermodynamics of information ratchets with HMM input data streams, to date these papers have mostly focused on information ratchets that create output patterns *ab initio*, with no patterns in the input stream, or that completely destroy patterns in the input stream (producing an output stream with no patterns). A natural topic for future research would be the challenging regime of intermediate cases, in which there are some patterns in the input stream, and different patterns in the output stream.

The analysis in these papers has focused on discrete-time rather than continuous-time models, e.g., discrete-time rather than continuous-time Markov chains. This means that much of the machinery of stochastic thermodynamics cannot be applied. Moreover, there are many subtle issues concerning what discrete-time systems can be be represented by *any* CTMC. It is (relatively) straight-forward to address these issues when the system has a finite state space, e.g., if it is a circuit. (See Sections VIII and XV.) However, the global state space of an information ratchet is infinite, including all input sequences of arbitrary length. In light of this, [134] considers some of the implications for the earlier analysis on discrete-time information ratchets that arise if one requires the information ratchet to be physically instantiated with a CTMC.

[34] For example, this could be the case if the states of the information ratchet are actually coarse-grained bins of some underlying physical fine-grained space. In this situation, the dynamics over the coarse-grained bins – over the states of the ratchet – are given by a hidden Markov model (HMM).

A. Infinite-time limit of entropy dynamics of finite-state information ratchets with arbitrary non-IID inputs

In the literature on the thermodynamics of information ratchets, the term **global system** is sometimes used to refer to the combination of the information ratchet, input data stream, and output data stream. In addition, throughout throughout those papers it is assumed that there is a single thermal reservoir coupled to the global system, and that the global system's rate matrix always obeys LDB for that reservoir, for some associated Hamiltonian.

One of the most important results to emerge to date concerning the thermodynamics of information ratchets arose from considering the infinite time limit of information ratchets that have finite R. Suppose that in that limit the distribution of computational states of the ratchet reaches a stationary state. Then in that limit, the EF in the global system produced per unit iteration is bounded below by the difference between the Kolmogorov-Sinai entropy of the output data stream and the Kolmogorov-Sinai entropy of the input data stream [15].[35] The authors refer to this as the **Information processing second law** (IPSL).

The two entropy rates in the IPSL each refer to changes in entropy of only a subset of the variables in the global system, namely the input data stream and the output data stream, respectively. Moreover, the distribution over the joint space of those two data streams cannot reach a fixed point, because the size of that space grows with each iteration. However, the rate of change in (the entropy of) that distribution per iteration can reach a fixed point. That is what the IPSL captures.

In general, for a given desired map from the input stream to the output stream, the greater the number of states of the ratchet one can use to implement that map, i.e., the larger R is, then the closer the minimal EF (per iteration, in the infinite-time limit) of those ratchets will come to matching the IPSL. Intuitively, the reason for this is that the dynamics of the state of the ratchet is less constrained when R is larger, and therefore the ratchet can implement the desired map to the input stream with greater thermodynamic efficiency. Viewed differently, the greater R is, the better able the ratchet is to "store in its current state a sufficient statistic" concerning the most recent sequence of inputs, and therefore the better able it is to anticipate what the next input will be, and therefore act in a manner that is thermodynamically optimal for that input.

[35] Typically, the Kolmogorov-Sinai entropy reduces to the more familiar entropy rate [70].

B. Infinite time limit of Entropy dynamics of infinite-state information ratchets with arbitrary non-IID inputs

The computational power of information ratchets with a finite set of computational states R is weaker than that of TMs (contrary to some informal claims in the literature). More precisely, it is not the case that for any given TM there is some information ratchet such that the (partial) function computed by the TM equals the (partial) function computed by the information ratchet (e.g., if we specify some special finite string of output bits of the information ratchet to signal that the ratchet has completed the computation). Only if one can map each ID of a TM to a unique element of R can we map the dynamics of that TM into the dynamics of an equivalent information ratchet. However, that would require that R be infinite, in general.[36]

This limited power of information ratchets with finite R naturally leads to consideration of information ratchets that have infinite R. Analyzing the entropy dynamics of such information ratchets presents some significant technical challenges however.

In addition, many of the nice properties of finite R ratchets no longer hold for infinite R ratchets. One example of this is that the IPSL no longer applies with infinite R (indeed, with an infinite state space, the information ratchet may never reach a stationary state). Another example arises from the fact that the thermodynamic benefit of expanding R mentioned above relies on our ignoring the thermodynamic cost of initializing the state of the ratchet before it starts to run. When we are considering the limit of iterating the ratchet an infinite number of times starting from that single initialized state, and there are only a finite number of computational states of the ratchet, the ratio of this one-time initialization cost of the ratchet to the number of iterations becomes infinitesimal, and so can be ignored. However, if R is infinite, this ratio need not go to zero. So the thermodynamic benefit of expanding R may disappear once R becomes infinite. This general point is emphasized in [12], where it is pointed out that Landauer's bound can *appear* to be violated in the asymptotic limit when the information ratchet has an infinite state space. This apparent violation arises because if the state space is infinite, then the initialized state essentially serves as an infinite information battery.

[36] Indeed, we can view information ratchets as a variant of conventional prefix-free TMs, in which there is a one-way tape that contains the input string as usual, a one-way tape that contains the output string as usual — but rather than an infinite work tape as in prefix-free TMs, in information ratchets the work tape is finite.

C. Finite time "synchronization costs" of finite information ratchets with arbitrary non-IID inputs

At the opposite extreme from the asymptotic limit of running a ratchet for an infinite number of iterations (where the IPSL applies) is the regime of running the ratchet only a small number of times, before the global system asymptotes. In this regime, even if the *input data stream* has reached its stationary distribution, in general the EF of the ratchet exceeds the value given by the IPSL. This is for the simple reason that the global system may not have reached a stationary distribution by the time that the input data stream does. Loosely speaking, it takes time for the dynamics of the state of the ratchet to "synchronize" with the dynamics of the incoming data stream (and/or outgoing data stream, as the case may be).

A useful tool for analyzing this regime is the **implementation cost**, which is the mutual information at any given iteration between the state of the ratchet and the combination of the input data stream and the output data stream [17]. An important result here is that for ratchets that are predictive of their previous outputs, the greater the number of states of the ratchet (i.e., the larger its "memory"), the greater the transient dissipation [122]. However, there are other kinds of ratchets besides predictive ones. In particular, "retrodictive ratchets", in which the state of the ratchet has minimal information about the previous outputs, but all the information that is shared between the past and future outputs, can synchronize without paying a penalty for a large state space.

D. Necessary conditions to reach the bound of the IPSL

The results reviewed so far in this section involve bounds on (the rate of) Landauer cost required by an information ratchet that is optimized for the input stream and the desired map of it into an output stream, e.g., as in the IPSL. In addition though, in [114], results closely related to those reviewed in Section IX are exploited, to derive properties that the information ratchet must have in order to achieve those bounds. These confirm and refine earlier results [18], establishing that the size of the ratchet's state space must match the size of the "memory" of the input data stream in order to achieve the IPSL bound. (In [18, 114], a formalization of this is referred to as the "thermodynamic principle of requisite variety".)

These analyses underscore an interesting fact. Information ratchets are not AO devices, and so cannot access the entire global system. (In particular, they cannot access all elements of the

input data stream). However, it is still the case that if they have been running long enough (to have passed the stage of transient synchronization costs), and have an update rule matched to the dynamics of the data streams, then they face none of the usual thermodynamic inefficiencies one would expect of non-AO devices. Intuitively, an information ratchet can do this because it *does* ultimately access all physical variables z in the (infinitely long) input that are relevant for the computation — just not all at once. To circumvent this problem of a delay in access to relevant variables, the ratchet stores in its state the information concerning each such successive variable z that is relevant for minimizing EP *in subsequent iterations*, after the ratchet has completed its interaction with z.

XIV. KOLMOGOROV COMPLEXITY AND THE ENTROPY DYNAMICS OF TURING MACHINES

A. The Landauer cost of implementing a TM

Suppose we are given a physical system that implements a prefix-free, single-tape UTM, whose state space X is the set of IDs of that TM. Suppose we are also given a desired output string $\vec{o}$. That output string in turn specifies a set $I(o)$ of all input strings $\vec{i}$ that result in the TM producing $\vec{o}$ and then halting.[37] What distribution over the elements of $I(\sigma)$ results in the smallest total EF by the time it halts? Equivalently, given that we can assume without loss of generality that the distribution over inputs is a delta function, what is the least amount of EF that could be incurred by running a UTM to compute $\vec{o}$ with an appropriately chosen element of $I(\sigma)$?

In this subsection I summarize the answer to this question originally given in [62].[38]. First, in order to have the physical system implementing the UTM not simply "cease to exist" whenever it reaches a halt state, modify the definition of TMs so that any halt state is a fixed point of the update function of the TM. Next, in order to avoid the problems that arise if we only sum entropic costs incurred by running a system until a random event occurs (see Section VII C), suppose we run the physical system that implements the UTM for an infinite number of iterations, summing entropic costs all along, even if they arise after the UTM halts. More precisely, I will calculate the total entropic costs incurred by running the UTM for τ iterations where standard accounting is

[37] In general, that set is infinite, since for any input string $\vec{i}$ that causes the UTM to compute $\vec{o}$, we can construct another input string that just loops an arbitrary number of times, doing nothing, before it runs the computation starting from $\vec{i}$.

[38] [62] considered conventional three-tape implementations of prefix TMs rather than single tape implementations. However, that required introducing some extra assumptions, e.g., concerning reinitialization of the work tape by the time the UTM halts, and so is not pursued here.

applied only at the end of the τ iterations (see Example 7.) I will then take the limit as those costs as $\tau \to \infty$.

I will also assume that the physical implementation of the UTM has no residual EP. In addition, I will assume the starting value of the prior over IDs is the appropriate delta function over the initial state of the head and its initial position on the tape, times the usual (normalized) coin-flipping prior over the set of all input tape strings for which the UTM halts. (Abusing notation, I will write that prior over input tape strings as $q(.)$, with no special notation for the prior over IDs.) I will suppose further that the system is not exactly cyclic; in each successive iteration, the (time-indexed) rate matrix changes so that it still implements the update function specified by the definition of the UTM, but has a new prior, given by propagating the prior of the previous iteration forward to the current iteration.

To begin, note that Landauer cost arises during many-to-one maps. A TM undergoes many such maps as it evolves. The Landauer cost of each of those maps is given by the drop in entropy of the actual joint probability of all of the IDs during such a map. Since there is zero residual EP, the total EP is mismatch cost, and the total EF that arises in running the UTM for τ iterations is the sum over all such iterations of the associated drop in the cross-entropy between the actual distribution over IDs and the prior distribution over IDs. Given our assumptions about the dynamics of the prior, this means that the total EF incurred by implementing the first τ iterations equals the initial value of that cross-entropy before the first iteration begins minus its value after the τ'th iteration ends.

Since we requires that the actual distribution over initial strings has its support restricted to $I(\sigma)$, we know that under the limit $\tau \to \infty$, the state of the tape converges under probability to a delta function centered on σ. For the same reason, the state of the head converges under probability to a delta function centered on the halt state, r^A. So the actual distribution over IDs converges under probability to a delta function.

Combining, we see that the total EF in the infinite-τ limit is just the initial cross-entropy between the actual distribution over IDs and the prior distribution over IDs, plus $\ln[q(I(\sigma))]$, the ending value of the cross entropy. Since the initial actual distribution is an delta function $\delta(., s^*)$ centered about the (unknown) EF-minimizing input string s^*, and the initial prior is the coin-flipping prior, the initial cross-entropy between the actual distribution over IDs and the prior distribution over IDs is just $\ln[\ell(s^*)] + \ln \Omega$, where Ω is Chaitin's constant, i.e., the normalization constant for the coin-flipping prior. (Note that due to Kraft's inequality, $\Omega \leq 1$, and so $\ln \Omega \leq 0$.) Combining, the

minimal EF required to compute σ using the UTM is [62]

$$\Big(K(\sigma) + \log[q(I(\sigma))] + \log \Omega \Big) \tag{66}$$

where $K(\sigma)$ is the Kolmogorov complexity of using the UTM to compute σ.

Note that Eq. (66) depends crucially on the fact that we take $\tau \to \infty$. If we run the TM on the (implicit) input s^* that results in the EF of Eq. (66), the TM will halt, with output equal to σ, at some finite time. But our calculation resulting in Eq. (66) requires us to keep running the TM past then. Extra entropic costs are incurred as we do that, since even though the state remains fixed at σ if we start at s^*, the probability of the TM reaching σ under the coin-flipping prior will keep changing. Concretely, that reflects the fact that other inputs to the TM besides s^* eventually reach σ, i.e., there are other trajectories through the space of IDs that converge with the one starting from s^*, and that do so *after* the TM run on s^* halts. Those convergences of trajectories are many-to-one maps, and therefore incur entropic costs, which are reflected in Eq. (66).

Intuitively, the calculation of Eq. (66) says that the minimal EF to compute a string σ is given by adding a "correction" term to the Kolmogorov complexity, which consists of Chaitin's constant for the UTM, plus the log of the total prior probability under the coin-flipping prior of all input strings that produce σ. That correction arises from the fact that Kolmogorov complexity is concerned with the smallest length input string, out of those input strings which result in σ, whereas Eq. (66) is concerned with the smallest amount of EF generated by running the UTM, out of those input strings which result in σ.

The normalization constant Ω in Eq. (66) is uncomputable, and the two functions in Eq. (66) are nonrecursive [84]. So sum of those three terms cannot be computed. However, that sum is only a lower bound in any case. So if one can compute lower bounds on each of those three terms for a given σ, then the sum of those three lower bounds provides us with a (computable) lower bound to the EF needed by a system that implements the UTM to compute σ.

It is important to realize that the expression in Eq. (66) reflects nonzero mismatch cost. In fact, such mismatch cost is unavoidable, since we have a fixed TM, and are varying the distribution over its input strings (looking for the delta function distribution that results in minimal EF). Indeed, if we had zero mismatch cost, then the actual distribution over inputs would have to equal the coin-flipping prior. This would mean that the distribution over output strings produced by running the UTM would not be restricted to σ — in fact, that distribution would have full support over the space of output strings (since U is a UTM, by hypothesis).

B. Recent investigation of entropy dynamics of a logically reversible TM

The recent paper [135] contains an analysis of the thermodynamics of a CTMC that obeys LDB and implements an arbitrary, fully-specified TM. The authors use a variant of the three-tape prefix TMs described above, which instead involves an input tape, a history tape, a working tape, and an output tape. They require that their TM's head cannot back up when reading the input tape, just like the conventional three-tape, prefix TMs [84]. However, in contrast to such TMs, they require that the input string on the input tape of their TM be finite and clearly delimited, by blanks.

Unusually, they also require that the TM be "logically reversible" even though it is implemented using a CTMC that obeys LDB, and so backward versions of every allowed forward transition between IDs are allowed. (They relate this to earlier work on "Brownian computers" in general, and DNA TMs in particular [61, 136].) To do this they require that the only paths through the space of IDs that are allowed by the CTMC are those that obey the update rule of the TM. However, any given transition along such a path can go backward, to an earlier ID, rather than forward, to the next ID. "Logical reversibility" for them means that the update rule of the TM is logically reversible. This implies that there is no intersection between any two paths; even though the system evolves stochastically forward and backward on a path, each allowed path is uniquely specified by its starting state.

Note though that this is not the meaning of "logical reversibility" conventionally considered in the literature (see Section VI). In their model of a TM, any of the states along a particular path (after the first state) has two possible predecessor states. So the system dynamics involves two-to-one maps (albeit stochastic ones). This means that the system is no more or less logically reversible than simple bit erasure is, and no more or less thermodynamically reversible than simple bit erasure is. In addition, each iteration of the system can result in non-zero Landauer cost, just like bit erasure does.

They have two major conclusions. First, they establish that in the infinite time limit, when the probability distribution over IDs reaches a steady state, the Landauer cost per iteration can be made arbitrarily small, by making the bias of the CTMC in favor of forward rather than backward transitions small enough. However, they also show that in the finite-iteration regime, where the TM has only been running for a finite number of steps, the Landauer cost per iteration will be negative. This reflects the fact that the distribution over IDs starts as a delta function when the TM begins, but (due to the stochastic nature of the CTMC) diffuses as the iteration number grows. (See the discussion at the end of Section VII B.)

Importantly, the authors consider a scenario where both input strings and output strings persist after completion of a computation, for an infinite number of following computations. So their analysis does not obey standard accounting. Concretely, their system requires that the user has an infinite set of initialized bits to use to store both the inputs and the outputs of every computation that the user runs. As discussed in Sections VII and XI, such a set of bits is an information battery. If we wanted to, we could ust use such a battery directly to drive the EF of a conventional, irreversible TM, rather than use it to store copies of the inputs and outputs of all runs of the TM in the past.

C. Early investigations of the Landauer cost of logically reversible Turing machines

The earliest work on the thermodynamics of Turing machines was by Bennett [60, 61]. This work was done before modern nonequilibrium statistical physics, and so had to rely entirely on the original bound developed by Landauer involving bit erasure, without exploiting modern nonequilibrium statistical physics. (See the discussion motivating the term "Landauer cost", just afer Example 2.) Working under the supposition that the only way to reduce thermodynamic cost was to reduce the total number of bit erasures, these early papers concentrated on how to convert a logically irreversible TM into an equivalent logically reversible one. After the initial work showing how to perform such a conversion, the focus shifted to how to minimize the resources needed to run that converted system, i.e., on how to minimize the growth as that system progresses in the size of its buffer "history tape", which it uses to ensure the computation stays logically reversible [2].

However, the procedure for converting a logically irreversible TM into an equivalent logically reversible TM is similar to the procedure for converting a logically irreversible circuit into an equivalent logically reversible circuit, as described in Section XI. This means that there are caveats concerning the entropic costs of running a logically reversible TM which is constructed to emulate a given irreversible TM that are similar to caveats concerning the entropic costs of running a logically reversible circuit constructed to emulate a given logically irreversible circuit.

Specifically, recall that the answer-reinitialization of a logically reversible circuit will incur at least as much Landauer cost as is incurred in the answer-reinitialization of the equivalent (logically irreversible) AO device. In fact, in one model of how to perform answer-reinitialization, the extra Landauer costs of answer-reinitialization of a logically reversible circuit will be at least as large as the entire Landauer cost of running the equivalent AO device.

Similar issues hold for relating the entropic costs of given logically irreversible TMs and equiva-

lent logically reversible TMs. In particular, in the case of circuits, the extra answer-reinitialization costs arise due to the need to reinitialize an extra set of output bits that contain a copy of x^{IN}. Analogously, in the case of TMs, the extra answer-reinitialization costs arise due to the need to reinitialize an extra set of bits on the output tape of the TM that contain a copy of the initial input string on the TM's input tape, x^{IN}.

D. Early investigations of the Landauer cost of logically *ir*reversible Turing machines

I now consider one of the most sophisticated of the early papers on the thermodynamics of computation, which considered the thermodynamics of *ir*reversible Turing machines [110]. This paper focused specifically on the connection between the minimal "thermodynamic price" it takes to run a given TM to compute a desired output string σ starting from a given input string x^{IN} on the one hand, and on the other hand, the conditional Kolmogorov complexity of the inputs to the TM, conditioned on the desired output string.

The analysis in [110] considers a scenario in which one first runs a Bennett-style reversible TM, but when that TM finishes, "one insists on replacement of the input with the output in the computer memory" [110]. So implicitly at least, there is appreciation for the problems with not reinitializing that input which were discussed in Section XI.

However, because of when [110] was written, it could not exploit the modern, exact equalities for the entropic costs of arbitrary processes, but had to rely entirely on indirect arguments concerning the "thermodynamic cost" of bit erasure given by Landauer's original bound. As a result, it is not clear precisely how to interpret that analysis in modern terms. For example, the analysis in [110] does not distinguish among EF, EP and Landauer cost (concepts that had not been defined in their modern form when the paper was written). Instead it informally refers to "thermodynamic price". Confusing the issue of how to interpret [110] even more is that the discussion in [110] repeatedly confounds logical irreversibility and thermodynamic irreversibility. (See the discussion in Section VI.)

At a high level, the idea in [110] seems to be as follows. Suppose you run a Bennett-style TM based on an irreversible TM U, ending with the desired output σ and a copy of the input, s. As discussed in Section XI, you now need to reinitialize that copy of the input, i.e., erase it, to allow you to use your TM again. Assume that the "thermodynamic price" of each bit in that copy of the input that you erase is just the original Landauer bound, $\ln[2]$. Accordingly, before erasing the copy of the input, you (reversibly) encode it in the shortest string you can with σ as side-information.

So long as that encoding is lossless, carrying it out has zero entropic cost. At this point, we erase that encoded version of x^{IN}, y, to complete a cycle.

Since by construction y is not longer than x^{IN}, the thermodynamic price of erasing y is not higher — and potentially far smaller — than would have been the thermodynamic price of erasing thermodynamic price. To make this precise, suppose we explicitly specify our encoding / decoding algorithm by specifying a Turing machine E — possibly different from U — that decodes y into x^{IN}. Then the minimal length of y would be $K_E(s|\sigma)$, the conditional Kolmogorov complexity of s given σ, using Turing machine E [84]. So, the minimal thermodynamic price of erasing the extra copy of the input is $k_B T \ln[2] K_E(s|\sigma)$, where T is the temperature of the (single) heat bath.

To make this interpretation of the analysis in [110] somewhat more precise, suppose we have a system with countable state space X, and that the system starts in a particular **input** state $x^{IN} \in X$. (Such an initial state is sometimes called a *program* in [110].) We then repeatedly apply a deterministic map $g : X \to X$ to the state of the system. In general, X may be finite or not, and the map g may be logically reversible or not.

We interpret that sequence of iterations of g as the computation. Note that this computation overwrites the input to produce the output. So the requirement of standard accounting that the input be reinitialized by the end of the computation does not apply. (See Section VIII.)

We allow the total number of iterations of g in a computation to be either a pre-fixed, finite value, or instead to be determined dynamically, in a way that depends on the input x^{IN} (e.g., as in a finite automaton, or a TM). In the second of those two cases, it may be that the sequence of maps never halts at all. In light of this, I will write $X^* \subseteq X$ to mean the set of initial states such that the resultant sequence of states does indeed ultimately halt. (If the number of maps is a pre-fixed, value then $X^* = X$.) I write $f(x)$ to mean the single-valued function taking any $x \in X^*$ to the resultant state of X that the system is in when the sequence of maps halts.

Next, suppose we have a space Y with each $y \in Y$ interpreted as an encoding of x, conditioned on the information $f(x)$. More precisely, define a space Y and a map $F : Y \times f(X^*) \to X^*$ such that for all $x \in X^*$, there exists a $y \in Y$ such that $F(y, f(x)) = x$. F is the **decoding map**. Define an associated set-valued **encoding map** to be any function $G : X^* \to 2^Y$ such that for all $x \in X^*$, and all $y \in G(x)$, $F(G(x), f(x)) = x$. (Note that if we change $F(.,.)$, we must also change $G(.)$.) To comport with the high-level description above, suppose as well that $F(.,.)$ is run by a TM E. For later use, let $D : Y \to \mathbb{R}^+$ be a *size* function. For example, if Y is a set of bit strings, $D(y)$ could be $\ell(y)$.

As an example of these definitions, in [110] the space Y consists of all possible encodings of the

trajectories of the successive irreversible changes to the ID of a prefix-free UTM, U, that arise during the iterations of the update function of that UTM. (These trajectories are called *histories* in [110], paralleling the terminology in [60].) G maps any initial x^{IN} to the set of all encoded versions of that particular history which is constructed by running the TM starting from x^{IN} until the TM halts. F is the function that takes the output generated by running the TM on x^{IN}, $f(x^{IN})$, together with any associated encoded version of a history from the set $G(x^{IN})$, and reconstructs x^{IN}, e.g., by running the TM "in reverse" from x^{IN}, using the entries in $G(x^{IN})$ to resolve any ambiguities in this reverse-evolution as they are encountered.

The analysis in [110] assumes that rather than implement the UTM U directly in a physical system, one implements the map $\Gamma : x \in X^* \rightarrow (\arg\min_{y \in G(x)} D(y), f(x))$, which takes the initial state x to the final state $f(x)$ combined with the encoding $G(x)$ that has smallest size. By construction, Γ is logically reversible. Therefore both its Landauer cost and its mismatch cost are zero, no matter what the initial distribution over X^* is, and no matter what the prior over X^* is. (See Section XI.) Assuming that the residual EP of the system implementing Γ is also zero, this means that the EF to run Γ is zero. This establishes Lemma 1 in [110].

After having run Γ, in order to meet the requirements of standard accounting we must reinitialize y. The question addressed in [110] is, what is the minimal thermodynamic price to reinitialize y? For simplicity of the exposition, consider the situation that [110] focuses on, where Y is a space of bit strings, and $D(y)$ is the length of the bit string y. Although there is no mention of EP in [110], and the formula for mismatch cost had not been derived when it was written, it seems that the easiest way to interpret the answer to the question given in [110] is to view "thermodynamic price" as meaning total EF — including mismatch cost — for the case where the residual EP is zero. This interpretation makes several assumptions:

First, note that there is implicitly a prior $q(y)$ in the system that erases y. Suppose that $q(y)$ is uniform over all strings of length less than or equal to some L, and that it is zero for all longer encoded strings.[39] To ensure that there is no possibility of infinite mismatch cost with this prior, we have to also assume that the actual distribution over inputs to the TM U has its support restricted to a set $\hat{I}_L$ containing only x^{IN} that result in y's that are not longer than L bits. In turn, since we need reversibility of the map Γ, we assume that $\hat{I}_L$ does not contain more than 2^L elements.

Next, suppose you are given a desired output of the TM, σ. Suppose that $I(\sigma) \cap \hat{I}_L \neq \varnothing$ and that the TM starts its computation of σ starting from some particular $x^{IN} \in I(\sigma) \cap \hat{I}_L$. Making

[39] If we did not build such an upper limit on the length of the strings into the prior, then that prior would be unnormalizable.

these assumptions, and adopting standard accounting, we ask:

> *What is the minimum value, over all L, of the total EF that would be generated by running Γ followed y-erasure on some specified $x^{IN} \in I(\sigma) \cap \hat{I}_L$?*

(In [110], the answer to this question is called the *thermodynamic price* for computing σ from x^{IN}.)

Since we're taking residual EP to equal zero, and since the EF incurred in running Γ is zero, the total EF is the drop in the cross-entropy between the actual distribution $p(y)$ and $q(y)$ that occurs from the beginning of the process of erasing the first L bits of y to its end. Since $q(y)$ is uniform, the associated drop in cross entropy is just $L \ln[2]$. Moreover, by definition of G, the minimal L is just $K_E(x^{IN}|\sigma)$. So the answer to the question is $K_E(x^{IN}|\sigma) \ln[2]$. This is Theorem 1 in [110].

There are subtle aspects of the analysis in [110] worth bearing in mind that have nothing to do with modern nonequilibrium statistical physics. For example, as mentioned above, the TM defining the Kolmogorov complexity function $K_E(.)$ that appears in the results in [110] need not be for the same TM U whose cost is being analyzed; there are two TMs involved, and *a priori*, they need not have any relation. At best, one could say that the difference between the Kolmogorov complexities of the two TMs is bounded by an arbitrarily large constant, which is independent of the output string being computed. In other words, the analysis gives "the minimal thermodynamic price" it takes to run a given TM to compute a desired output string up to an unspecified additive term, a term that is bounded by a constant that can be made arbitrarily large.

On the other hand, suppose we require that the two TMs, E and U, are identical. That would obviate this last concern. In addition, it seems that it would allow us to take L to infinity, and also to optimize the EF over the set of all $x^{IN} \in I(\sigma)$, rather than only consider the entropic cost for an arbitrary, given $x^{IN} \in I(\sigma)$. This conjecture is motivated by the following semi-formal reasoning.

First, let Y be $\mathbb{B}^*$, the set of all finite bit strings, and assume that the y-erasing process uses the coin-flipping prior over Y. In addition, have the y-erasing process only erase the $\ln[\ell(y)]$ bits that comprise y. For an arbitrary y-erasing TM E, doing this would require that the process count up to $\ell(y)$ as y is created and that it stores that value, in order to determine how many bits to erase. Under standard accounting, we would then have to erase that stored value of $\ell(y)$ after using it to erase y, to allow the overall system to be run on a new input x^{IN}. However, *if E is prefix-free*, then y is a prefix-free string, i.e., it is an "instantaneous description". This means that the process of erasing y does not need to know how many bits to erase ahead of time — it can just keep erasing until it reaches the end of y, at which point it will halt. So we do not face the problem of accounting for the entropic cost of erasing the value $\ell(y)$. (However, depending on the details of this y-erasing

system, we may have to erase a bit that it stores, telling it whether it has reaching the end of y or not.)

For simplicity, adopt the coin-flipping prior over this infinite Y. So the prior probability assigned by the y-erasing system to any particular string y is $2^{-\ell(y)}/\Omega$, where Ω is the normalization constant. Take the limit of running the string-erasing TM for an infinite number of iterations (as in Section XIV A). That means that the ending cross-entropy between the actual distribution and the prior goes to $1 \ln[1] = 0$. Accordingly, the drop in cross-entropy between those two distributions during the y-erasing process is just the starting cross-entropy between them. If the actual starting distribution is a delta function centered on y, then the value of this starting cross-entropy is $\ell(y) + \log[\Omega]$. As discussed above, the minimal value of $\ell(y)$ is $K(x^{IN}|\sigma)$. Plugging this in and then minimizing over x^{IN}, we see that the minimal EF is

$$\min_{x^{IN} \in I(\sigma)} \left(K(x^{IN}|\sigma) \right) + \log[\Omega] \tag{67}$$

To understand this result intuitively, note that $K(x^{IN}|\sigma)$ can be interpreted as the amount of information that is contained in x^{IN} but not in σ. Accordingly, the minimal EF given in Eq. (67) is just the least possible amount of information that is lost when the TM transforms one of the elements of $I(\sigma)$ into σ (up to the additive constant of $\log[\Omega]$).

There are several points about this analysis worth emphasizing. First, note that the system considered in Section XIV A is just (an implementation of) the single TM U. In contrast, the system considered here first runs a function Γ which is *constructed from* U, and then runs a y-initializing process. Moreover, in the system considered in Section XIV A, the coin-flipping prior arises as a distribution over X^{IN}, whereas in the system considered here, the prior over X^{IN} is arbitrary; instead, the coin-flipping prior arises in the system considered here as a distribution over Y.

The price paid for changing the system so that we can move the coin-flipping prior from X^{IN} to Y is that that new system needs to be able to calculate the non-recursive function sending x to the shortest string y such that $F_E(y, f(x)) = x$. In other words, the new system has to have super-Turing capabilities, unlike the original system, which consisted only of the TM U.[40]

Finally, just like other results in Turing machine theory, all the results recounted here hold only up to additive $O(1)$ terms. For infinite spaces X and Y, that is fine, but for finite X and Y, such additive constants can swamp the other terms in these results. Combined with the fact that these

[40] Zurek was well aware of this problematic aspect of analysis summarized in this subsection, saying that the bound of Theorem 1 in [110] "cannot {actually be met} by recursive computation {i.e., by running a TM guaranteed to halt}", that at best it is met "perhaps by sheer luck".

results are only bounds, this means they have little to say about computation with finite X. To give a simple example, all of the results recounted here apply to straight-line circuits. Yet as recounted in Section X, the formulas for the entropic costs of circuits do not involve Kolmogorov complexity (in contrast to the results recounted here).

As a closing, historical comment, an explicit goal of much of the early work on the thermodynamics of TMs was to rederive statistical physics with little if any use of probabilities [81–84, 137]. Ultimately, perhaps the most fruitful way to consider the analysis in [110] is not to translate it into modern nonequilibrium statistical physics, so deeply grounded in the use of probability distributions, but rather to view it as part of this research program which strived to expunge the use of probabilities from statistical physics.

XV. THE MIMIMAL HIDDEN COMPUTATION OCCURING WITHIN A VISIBLE COMPUTATION

Recall the construction summarized in Section VIII, of an AO device that can implement any conditional distribution π over a set of "visible" states X in a thermodynamically reversible manner – even if the output distribution under π depends on the value of the input. This construction works by expanding the original state space X, of size $|X|$, into a state space of size $|X|\times|X'|$, and defining a dynamics over $X \times X'$ that implements π over the subspace $\{(x, 0) : x \in X\}$.

In that construction, X' is the same size as X, so the joint space $X \times X'$ has $|X|(|X|-1)$ more "hidden" states than the original space of "visible" states, X. That can be a *huge* number of extra states. For example, in a digital computer with a 64-bit address space, $|X|= 2^{64}$ — and so the number of extra states needed to implement a nontrivial map over X using that construction is $\sim 2^{128}$.

This raises the question of what the minimal number of extra, hidden states must be, in order to implement a given distribution π in a thermodynamically reversible way. Some recent results concerning this and related issues are summarized in this section.

First, one doesn't need to restrict attention to CTMCs that operate over Cartesian product spaces $X \times X'$. In full generality, we are interested in CTMCs that operate over a space of the form $X \cup Z$, for some appropriate Z. The construction summarized in Section VIII is just a special case, since we can rewrite a space $X \times X'$ as $X \cup Z$ if we identify Z as $X \times X' \setminus \{(x, 0) : x \in X\}$.

Bounds on the minimal value of $|Z|$ needed for a CTMC over $X \cup Z$ to both implement a given conditional distribution π over $\{(x, 0) : x \in X\}$ and to be thermodynamically reversible are derived

in [138], for the case of a finite X. These bounds apply to any π, even "maximally noisy" ones where for every x', x, $\pi(x|x') \neq 0$. That paper also contains some results for deterministic π, for the special case of a countably infinite X.

However, in computer engineering we are typically interested in conditional distributions π that implement a deterministic, single-valued function over a finite space X. There are several special properties of such scenarios. First, perhaps surprisingly, it turns out that any nontrivial deterministic map $x \to f(x)$ cannot be implemented by *any* CTMC operating over only the visible states X, *even approximately* [120]. This is true independent of concerns about thermodynamic reversibility or the like; it simply arises due to the mathematics of CTMCs [139, 140]. As a striking example, a simple bit flip over a space $X = \{0, 1\}$ cannot be implemented by any CTMC over X, even approximately, no matter how much dissipation accompanies the CTMC, or how the rate matrix varies in time, or how long we run the CTMC.

This means that in order to implement any nontrivial deterministic map, thermodynamically reversibly or not, one *must* use hidden states. So for example, under the approximation that some real-world digital device implements a deterministic Boolean map over a set of visible bits, and that the dynamics of the device can be modeled using stochastic thermodynamics, we know that that the CMTC going into the stochastic thermodynamic analysis must use hidden states.

Perhaps even more surprisingly, it turns out that any CTMC that implements a deterministic map can be decomposed into a sequence of more than one "hidden" timesteps. These timesteps are demarcated from one another by changes in what transitions are allowed under the rate matrix [120]. In general, for a given set of visible states X, hidden states Z, and deterministic map $f : X \to X$, the minimal number of hidden timesteps for any CTMC over $X \cup Z$ to implement f over X is greater than 1. So any real-world digital device that implements some Boolean operation over its state space in each of its iterations must have a set of multiple hidden timesteps that occur within each of those iterations (assuming the device can be modeled as evolving under a CTMC that implements a deterministic function).

Often there is a real-world cost for each additional hidden state, and also a cost for each additional hidden timestep. (For example, in systems that evolve while connected to a heat bath and obeying LDB for some Hamiltonian, at the end of a timestep either an infinite energy barrier between elements of Z is raised, or an infinite energy barrier is lowered.) So there is a "space/time" tradeoff between the costs associated with the number of hidden states used by any CTMC that implements a given $f(x)$ and the costs associated with the number of hidden timesteps used by the CTMC to implement $f(x)$.

This tradeoff involving hidden states and hidden timesteps occurs within any digital device, and in a certain sense "lies underneath" the more conventional space / time tradeoffs studied in computer science theory. Indeed, consider the special case that there are no constraints on the CTMC operating over $X \cup Z$, e.g., due to restrictions on the Hamiltonian driving that CTMC. In this case, that CTMC operates as an AO device over $X \cup Z$. For this special case, the precise form of the hidden space/time tradeoff can be given in closed form [120]. As an example, it turns out that (if there are no constraints on the CTMC) a bit flip can be implemented using only one hidden state, but only if one uses three hidden timesteps.

XVI. FUTURE DIRECTIONS

There has been a resurgence of interest recently in the entropic costs of computation, extending far beyond the work in stochastic thermodynamics summarized above. This has work has taken place in fields ranging from chemical reaction networks [44, 46, 47, 141, 142] to cellular biology [40, 42] to neurobiology [143, 144], in addition to computer science [1]. There is now a wiki serving as a community resource for workers from all these different disciplines with a common interest in the entropic costs of computation (`http://centre.santafe.edu/thermocomp`).[41] In addition, there is a book coming out in early 2019, that summarizes some of the insights of researchers from all those fields [145].

Almost all of the work to date on the entropic costs of computation is concerned with *expected* entropic costs, averaging over all trajectories the microstates of a system might follow. However, a very fruitful body of research in stochastic thermodynamics considers the full distribution of the entropic costs of individual trajectories [85, 86]. The associated results allow us to consider questions like what the probability is in a given process of EP having a particular value, e.g., as formalized in the famous "fluctuation theorems" [99, 146, 147]. Some more recent research along these lines has resulted in "uncertainty relations" [118, 148–150], which relate how precise the dynamics of a system (e.g., of a computational machine) can be to the EP it generates. Other recent research has resulted in classical "speed limits" to go along with the quantum ones [24]. As an example, one such speed limit lower-bounds how confident we can be that a desired state transition has occurred in a given amount of time, as a function of EP [151]. (See also [152].)

Clearly these recent results are related to issues that play a fundamental role in the entropic costs of computers, and in particular to the tradeoffs between those costs and how confident we can

[41] Researchers are *highly* encouraged to visit it, not just to find useful information, but also to improve it, e.g., by putting in announcements, adding references, adding researcher web page information, etc.

be that a given physical system implements a desired computation exactly, how fast it can do so, etc. All of these issues remain to be explored.

Restricted attention just to the issues touched on in this paper, there are many avenues of future research that bear investigating. One set of issues worth investigating is how the entropic costs of probabilistic Turing machines [8] are related to those of deterministic Turing machines (the kind considered in Section XIV). Similarly, the analysis in Section XIV considered TMs that operate over the space of IDs of the TM. So in each iteration, they act as an AO device. A natural line of research is to explore how the entropic costs change if we instead model the TM as a solitary process, given by the conventional definition of TMs in terms of heads that can only access one position on the tape at a time. (See Section IV E.)

There are also many issues to investigate that are more closely tied to the kinds of problems conventionally considered in computer science theory. To give a simple example, for deterministic finite automata, we could investigate the following issues, which are closely related to topics introduced (and solved) in introductory courses in computer science theory:

1. Given a language L, does the (unique) minimal deterministic FA that accepts L also result in the smallest total Landauer cost (conditioned on having the distribution over inputs produce some string in L) of any deterministic FA that accepts L?

2. Is the deterministic FA with minimal total Landauer cost (conditioned on having the distribution over inputs produce some string in L) unique (up to relabeling of states)?

3. What is the largest possible total Landauer cost of any deterministic FA that accepts L (conditioned on having the distribution over inputs produce some string in L)?

4. Suppose we are given two deterministic FA, M_1 and M_2 and a shared distribution over input strings, where some of the strings accepted by M_1 are also accepted by M_2. What is the probability of a string that is accepted by both where the Landauer cost for M_1 exceeds that for M_2? What is the probability of such a string for which the two deterministic FAs have identical Landauer cost?

5. Suppose we have a deterministic FA that can get stuck in infinite loops, and / or accept arbitrarily long strings. Then we can ask many "entropy rate" variants of these questions. For example, does the Landauer cost per iteration of a given deterministoic FA approach an asymptotic value? How many such asymptotic values could the same FA approach (depending on the precise infinite input string)?

6. We can also consider variants of all these issues where consider mismatch costs instead of (or in addition to) Landauer costs.

A core concern of computer science theory is how the tradeoffs among the amounts of various resources needed to perform a computation scale with the size of the computation. For example, a common question is how the memory requirements of a computation trade off with the number of iterations the computation requires, and how both of those scale with the size of the computation. In fact, many textbooks are devoted to tradeoff issues, instantiated over different kinds of computational machine.

To date, none of those chapters has included the *thermodynamic* resources need to perform the computation, and how they scale with the size of the computation. In essence, every chapter in those textbooks can be revisited using the tools of stochastic thermodynamics, to see how they are extended when one includes these other kinds of resources.

ACKNOWLEDGEMENTS: I would like to thank Nihat Ay and especially Artemy Kolchinsky for many helpful discussions. This work was supported by the Santa Fe Institute, Grant No. CHE-1648973 from the U.S. National Science Foundation and Grant No. FQXi-RFP-1622 from the FQXi foundation. The opinions expressed in this paper are those of the author and do not necessarily reflect the view of National Science Foundation.

Appendix A: Properties of multi-divergence

Note that in Eq. (21) I adopt the convention that we subtract the sum involving marginal probabilities. In contrast, in conventional multi-information I subtract the term involving the joint probability.

To understand why I adopt this convention, consider the case where X has two components, labelled a and b, and use the chain rule for KL divergence to write

$$\mathcal{D}(p\|r) = D(p^{a|b}\|r^{a|b}) - D(p^a\|r^a) \tag{A1}$$

with obvious notation, where p and r are both distributions over the joint space (X^a, X^b).

$D(p^a\|r^a)$ quantifies how easy it is to tell that a sample x_a came from $p(x_a)$ rather than $r(x_a)$. Similarly, $D(p^{a|b}\|r^{a|b})$ tells us how easy it is, on average (according to $p(x_b)$), to tell that a sample x_a came from $p(x_a|x_b)$ rather than $r(x_a|x_b)$.

So Eq. (A1) says that the multi-divergence between p and r is the gain in how easy it is for us to tell that a sample x_a came from p rather than r if instead of only knowing the value x_a I also know the value x_b. In other words, it quantifies how much information X^b provides about X^a (on average), concerning the task of guessing whether a given sample x_a was formed by sampling p or r. (Note that since multi-divergence is symmetric among the components of x, $\mathcal{D}(p\|r)$ also quantifies how much information X^a provides about X^b.) However, this interpretation requires our convention, of subtracting the sum of marginal divergences from the joint divergence rather than the other way around.

In light of these properties of multi-divergence for variables with two components, we can interpret the multi-divergence between two probability distributions both defined over a space X that has an arbitrary number of components as quantifying how much knowing one component of X tells us about the other. This is similar to what multi-information measures. Indeed, as mentioned in the text, $\mathcal{D}(p\|r)$ reduces to conventional multi-information $\mathcal{I}(p)$ in the special case that r is a product distribution.

So under our convention, multi-divergence quantifies us how much knowing one component of X tells us about the other — somewhat analogous to what multi-information measures. Indeed, $\mathcal{D}(p\|r)$ reduces to conventional multi-information $\mathcal{I}(p)$ in the special case that the reference distribution r is a product distribution.

In addition to motivating our convention, this also means that the multi-divergence cannot be negative if r is a product distribution. More generally, however, if r is not a product distribution

but the components of X are correlated under r in a manner "opposite" to how they are correlated under p, then $\mathcal{D}(p\|r)$ can be negative. In such a case, being given a value x_b in addition to x_a makes it *harder* to tell whether x_a was formed by sampling from p or from r.

To illustrate this, consider a very simple scenario where $X = \mathbb{B}^2$, and choose

$$\begin{aligned} p(x_b) &= \delta(x_b, 0) \\ p(x_a|x_b = 0) &= 1/2 \qquad \forall x_a \\ r(x_b) &= \delta(x_b, 1) \\ r(x_a|x_b = 0) &= 1/2 \qquad \forall x_a \\ r(x_a|x_b = 1) &= \delta(x_a, 0) \end{aligned}$$

Plugging into Eq. (20) gives

$$\begin{aligned} \mathcal{D}(p\|r) &= -\sum_{x_b, x_a} p(x_a|x_b)p(x_b) \ln \left[\frac{r(x_a|x_b)}{p(x_a|x_b)} \frac{p(x_a)}{R(x_b)}\right] \\ &= -(1/2) \sum_{x_a} \ln \left[\frac{r(x_a|x_b = 0)}{(1/2)} \frac{(1/2)}{r(x_b)}\right] \\ &= -(1/2) \sum_{x_a} \ln \left[\frac{(1/2)}{(1/2)} \frac{(1/2)}{\delta(x_a, 0)}\right] \\ &= -\infty \end{aligned}$$

So on average, if you are told a value of x_b that unbeknownst to you came from p, in addition to being told a value of x_a that unbeknownst to you came from p, then you are less able to tell that that x_a value came from p rather than r.

This phenomenon is loosely similar to what's sometimes known as Simpson's paradox. This can be seen by considering the instance of that "paradox" where I have a distribution $p(z, x_b, x_a)$ over three binary variables, and simultaneously

$$p(z = 1|x_b, x_a = 1) > p(z = 1|x_b, x_a = 0)$$

for any value of x_b, yet

$$p(z = 1|x_a = 1) < p(z = 1|x_a = 0)$$

For such distributions, if we are told the value of x_b in addition to the value of x_a, we conclude that z if more likely to equal 1 when $x_a = 1$ than when $x_a = 0$. This is true no matter what I am told the value of x_b is. Yet I come to the opposite conclusion if we are only told the value of x_a, and are not told the value of x_b (see [153]).

Appendix B: Proof of Proposition 2

Proof. The form of the rate matrix of a subsystem process means that it is impossible to have a state transition in which both x_A and x_B change simultaneously. Accordingly, we can write $\boldsymbol{x} = (\boldsymbol{x}_A, \boldsymbol{x}_B)$, where $\boldsymbol{x}_A = (N^A, \vec{x}^A, \vec{\tau}^A, \vec{\nu}^A)$, and similarly for $\boldsymbol{x}_B$. (Note that as shorthand, we do not explicitly indicate in this decomposition that the value of x_B doesn't change when x_A does and vice-versa.) In addition, for pedagogical clarity, in this appendix I express rate matrices in the form $W_t^{\nu}(x' \to x)$ rather than $W_{x;x'}^{\nu}(t)$. I modify the notation for survival probabilities similarly.

If subsystem A undergoes a state transition $x^{A'} \to x^A$ at time t while system B stays constant, the associated value of the rate matrix is $W_t(x^{A'} \to x^A)$, and similarly if it is subsystem B that undergoes a transition. In addition, for any three times $\tau < \tau' < \tau''$, the survival probability of a subsystem not changing state between τ and τ'' equals the product of the survival probability for going from τ to τ' and the survival probability for going from τ' to τ''. These facts allow us to expand Eq. (26) to evaluate the probability of a given trajectory $\boldsymbol{x}$ as

$$\begin{aligned} p(\boldsymbol{x}_A, \boldsymbol{x}_B | x_0^A, x_0^B) &= \left(\prod_{i=1}^{N^A} S_{\tau_{i-1}^A}^{\tau_i^A}(x_{i-1}^A) W_{\tau_i^A}^{\nu_i^A}(x_{i-1}^A \to x_i^A) \right) \\ &\qquad \times S_{\tau_{N^A}^A}^1(x_{N^A}^A) \left(\prod_{j=1}^{N^B} S_{\tau_{j-1}^B}^{\tau_j^B}(x_{j-1}^B) W_{\tau_j^B}^{\nu_j^B}(x_{j-1}^B \to x_j^B) \right) S_{\tau_{N^B}^B}^1(x_{N^B}^B) \\ &:= p(\boldsymbol{x}^A | x_0^A) p(\boldsymbol{x}^B | x_0^B) \end{aligned} \tag{B1}$$

If we marginalize both sides of Eq. (B1) over all components of $\boldsymbol{x}_A$ except the starting and ending values of x^A, do the same for x_B, and then translate to the notation of Proposition 2, we derive

$$\pi(a_1, b_1 | a_0, b_0) = \pi^A(a_1 | a_0) \pi^B(b_1 | b_0) \tag{B2}$$

This formally establishes the intuitively obvious fact, that the CTMC obeys Proposition 2(1).

Next, using Eq. (27), we write the EF for the CTMC as

$$\begin{aligned} \mathcal{Q}(p) = \int & p(x_0^A, x_0^B) p(\boldsymbol{x}^A | x_0^A) p(\boldsymbol{x}^B | x_0^B) \\ & \times \left[\sum_{i=1}^{N^A} W_{\tau_i^A}^{\nu_i^A}(x_{i-1}^A \to x_i^A) \ln \frac{W_{\tau_i^A}^{\nu_i^A}(x_{i-1}^A \to x_i^A)}{W_{\tau_i^A}^{\nu_i^A}(x_i^A \to x_{i-1}^A)} + \sum_{i=1}^{N^B} W_{\tau_i^B}^{\nu_i^B}(x_{i-1}^B \to x_i^B) \ln \frac{W_{\tau_i^B}^{\nu_i^B}(x_{i-1}^B \to x_i^B)}{W_{\tau_i^B}^{\nu_i^B}(x_i^B \to x_{i-1}^B)} \right] D\boldsymbol{x}^A D\boldsymbol{x}^B \end{aligned}$$

$$= \int p(x_0^A) p(\boldsymbol{x}^A | x_0^A) \sum_{i=1}^{N^A} W^{\nu_i^A}_{\tau_i^A}(x_{i-1}^A \to x_i^A) \ln \frac{W^{\nu_i^A}_{\tau_i^A}(x_{i-1}^A \to x_i^A)}{W^{\nu_i^A}_{\tau_i^A}(x_i^A \to x_{i-1}^A)} D\boldsymbol{x}^A$$

$$+ \int p(x_0^B) p(\boldsymbol{x}^B | x_0^B) \sum_{i=1}^{N^B} W^{\nu_i^B}_{\tau_i^B}(x_{i-1}^B \to x_i^B) \ln \frac{W^{\nu_i^B}_{\tau_i^B}(x_{i-1}^B \to x_i^B)}{W^{\nu_i^B}_{\tau_i^B}(x_i^B \to x_{i-1}^B)} D\boldsymbol{x}^B$$

$$:= \mathcal{Q}_A(p_0^A) + \mathcal{Q}_B(p_0^B) \tag{B3}$$

This establishes the first part of Proposition 2(2). Finally, note that $\mathcal{Q}_A(p_A)$ is the EF that would be incurred for a CTMC over state space X_A with rate matrices $W_t^\nu(a' \to a)$, (i.e., if subsystem B did not exist at all).< So by the Second Law of Thermodynamics [86], $\mathcal{Q}_A(p_A) \geq S(p_0^A) - S(p_B^0)$, and similarly for $\mathcal{Q}_B(p^B)$. □

[1] E. D. Demaine, J. Lynch, G. J. Mirano, and N. Tyagi, in *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science* (ACM, 2016) pp. 321–332.

[2] C. H. Bennett, SIAM Journal on Computing **18**, 766 (1989).

[3] R. Landauer, Physics Today **44**, 23 (1991).

[4] C. P. Kempes, D. Wolpert, Z. Cohen, and J. Pérez-Mercader, arXiv preprint arXiv:1706.05043 (2017).

[5] G. West, *Scale: the universal laws of growth, innovation, sustainability, and the pace of life in organisms, cities, economies, and companies* (Penguin, 2017).

[6] J. M. Parrondo, J. M. Horowitz, and T. Sagawa, Nature Physics **11**, 131 (2015).

[7] C. Moore and S. Mertens, *The nature of computation* (Oxford University Press, 2011).

[8] S. Arora and B. Barak, *Computational complexity: a modern approach* (Cambridge University Press, 2009).

[9] J. E. Savage, *Models of computation*, Vol. 136 (Addison-Wesley Reading, MA, 1998).

[10] J. E. Hopcroft, R. Motwani, and J. Ullman, *Introduction to Automata Theory, Languages and Computability* (Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2000).

[11] M. Sipser, *Introduction to the Theory of Computation*, Vol. 2 (Thomson Course Technology Boston, 2006).

[12] A. Boyd, *Thermodynamics of Correlations and Structure in Information Engines*, Ph.D. thesis, Uuniversity of Clifornia Davis (2018).

[13] D. Mandal and C. Jarzynski, Proceedings of the National Academy of Sciences **109**, 11641 (2012).

[14] D. Mandal, H. Quan, and C. Jarzynski, Physical review letters **111**, 030602 (2013).

[15] A. B. Boyd, D. Mandal, and J. P. Crutchfield, New Journal of Physics **18**, 023049 (2016).

[16] A. B. Boyd, D. Mandal, and J. P. Crutchfield, Physical Review E **95**, 012152 (2017).

[17] A. B. Boyd, D. Mandal, P. M. Riechers, and J. P. Crutchfield, Physical review letters **118**, 220602 (2017).

[18] A. B. Boyd, D. Mandal, and J. P. Crutchfield, Journal of Statistical Physics **167**, 1555 (2017).

[19] M. C. Diamantini, L. Gammaitoni, and C. A. Trugenberger, Physical Review E **94**, 012139 (2016).

[20] J. Baez and M. Stay, Mathematical Structures in Computer Science **22**, 771 (2012).

[21] M. A. Nielsen and I. L. Chuang, *Quantum computation and quantum information* (Cambridge university press, 2010).

[22] P. Strasberg, G. Schaller, T. Brandes, and M. Esposito, Physical Review X **7**, 021003 (2017).

[23] J. Goold, M. Huber, A. Riera, L. del Rio, and P. Skrzypczyk, Journal of Physics A: Mathematical and Theoretical **49**, 143001 (2016).

[24] S. Deffner and S. Campbell, Journal of Physics A: Mathematical and Theoretical **50**, 453001 (2017).

[25] G. Gour, M. P. Müller, V. Narasimhachar, R. W. Spekkens, and N. Y. Halpern, Physics Reports **583**, 1 (2015).

[26] C. Sparaciari, J. Oppenheim, and T. Fritz, Physical Review A **96**, 052112 (2017).

[27] M. Krumm, .

[28] J. D. Barrow, Kurt Gödel and the Foundations of Mathematics: Horizons of Truth , 255 (2011).

[29] M. B. Pour-El and I. Richards, International Journal of Theoretical Physics **21**, 553 (1982).

[30] C. Moore, Physical Review Letters **64**, 2354 (1990).

[31] S. Lloyd, Nature **406**, 1047 (2000).

[32] S. Lloyd, in *The Incomputable* (Springer, 2017) pp. 95–104.

[33] T. E. Ouldridge and P. R. ten Wolde, Physical review letters **118**, 158103 (2017).

[34] T. E. Ouldridge, arXiv preprint arXiv:1702.00360 (2017).

[35] T. E. Ouldridge, C. C. Govern, and P. R. ten Wolde, Physical Review X **7**, 021004 (2017).

[36] P. Sartori, L. Granger, C. F. Lee, and J. M. Horowitz, PLoS Comput Biol **10**, e1003974 (2014).

[37] P. Sartori and S. PIglotti, Physical Review X **5**, 041039 (2015).

[38] P. Mehta and D. J. Schwab, Proceedings of the National Academy of Sciences **109**, 17978 (2012).

[39] P. Mehta, A. H. Lang, and D. J. Schwab, Journal of Statistical Physics , 1 (2015).

[40] S. J. Prohaska, P. F. Stadler, and D. C. Krakauer, Journal of theoretical biology **265**, 27 (2010).

[41] B. Bryant, PloS one **7**, e35703 (2012).

[42] Y. Benenson, Nature Reviews Genetics **13**, 455 (2012).

[43] D. Angluin, J. Aspnes, and D. Eisenstat, in *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing* (ACM, 2006) pp. 292–299.

[44] H.-L. Chen, D. Doty, and D. Soloveichik, Natural computing **13**, 517 (2014).

[45] Q. Dong, Master's thesis, Stony Brook University (2012).

[46] D. Soloveichik, M. Cook, E. Winfree, and J. Bruck, natural computing **7**, 615 (2008).

[47] C. Thachuk and A. Condon, in *International Workshop on DNA-Based Computers* (Springer, 2012) pp. 135–149.

[48] S. Goldt and U. Seifert, Physical Review Letters (2017).

[49] H. Touchette and S. Lloyd, Physica A: Statistical Mechanics and its Applications **331**, 140 (2004).

[50] H. Touchette and S. Lloyd, Physical review letters **84**, 1156 (2000).

[51] A. C. Barato and U. Seifert, New Journal of Physics **19**, 073021 (2017).

[52] T. Sagawa and M. Ueda, Physical review letters **100**, 080403 (2008).

[53] T. Sagawa and M. Ueda, Physical Review E **85**, 021104 (2012).

[54] H. Wilming, R. Gallego, and J. Eisert, Physical Review E **93**, 042126 (2016).

[55] S. J. Large, R. Chetrite, and D. A. Sivak, arXiv preprint arXiv:1802.02670 (2018).

[56] T. R. Gingrich, G. M. Rotskoff, G. E. Crooks, and P. L. Geissler, Proceedings of the National Academy of Sciences **113**, 10263 (2016).

[57] J. M. Horowitz and J. L. England, Entropy **19**, 333 (2017).

[58] A. C. Barato and U. Seifert, EPL (Europhysics Letters) **101**, 60001 (2013).

[59] J. M. Horowitz, Journal of Statistical Mechanics: Theory and Experiment **2015**, P03006 (2015).

[60] C. Bennett, IBM Journal of Research and Development **17**, 525 (1973).

[61] C. H. Bennett, International Journal of Theoretical Physics **21**, 905 (1982).

[62] D. H. Wolpert, "Extending Landauer's bound from bit erasure to arbitrary computation," (2015), arXiv:1508.05319 [cond-mat.stat-mech].

[63] P. Riechers, in *Energetics of computing in life and machines*, edited by D. H. Wolpert, C. P. Kempes, P. Stadler, and J. Grochow (SFI Press, 2018).

[64] M. Khadra, arXiv preprint arXiv:1711.06115 (2017).

[65] L. Kugler, Communications of the ACM **58**, 12 (2015).

[66] K. V. Palem, Phil. Trans. R. Soc. A **372**, 20130281 (2014).

[67] J. Lacomis, J. Dorn, and S. Weimer, W. Forrest, in *Energetics of computing in life and machines* (SFI Press, 2018).

[68] J. A. Grochow and D. H. Wolpert, ACM SIGACT News **49**, 33 (2018).

[69] D. H. Wolpert and A. Kolchinsky, arXiv preprint arXiv:1806.04103 (2018), old title: The entropic costs of straight-line circuits.

[70] T. M. Cover and J. A. Thomas, *Elements of information theory* (John Wiley & Sons, 2012).

[71] S. Watanabe, IBM Journal of research and development **4**, 66 (1960).

[72] D. Koller and N. Friedman, *Probabilistic Graphical Models* (MIT Press, 2009).

[73] S. Ito and T. Sagawa, Physical review letters **111**, 180603 (2013).

[74] S. Ito and T. Sagawa, arXiv:1506.08519 [cond-mat] (2015), arXiv: 1506.08519.

[75] V. Nekrashevych, *Self-similar groups*, Mathematical Surveys and Monographs, Vol. 117 (American Mathematical Society, Providence, RI, 2005) pp. xii+231.

[76] N. Barnett and J. P. Crutchfield, Journal of Statistical Physics **161**, 404 (2015).

[77] P. Arrighi and G. Dowek, International Journal of Foundations of Computer Science **23**, 1131 (2012).

[78] G. Piccinini, The British Journal for the Philosophy of Science **62**, 733 (2011).

[79] S. Aaronson, in *Computability: Turing, Gödel, Church, and Beyond* (MIT Press, 2013) pp. 261–327.

[80] S. Aaronson, "NP-complete problems and physical reality," (2005), quant-ph/0502072.

[81] C. M. Caves, Complexity, Entropy and the Physics of Information , 91 (1990).

[82] C. M. Caves, Physical Review E **47**, 4010 (1993).

[83] W. H. Zurek, Phys. Rev. A **40**, 4731 (1989).

[84] M. Li and P. Vitanyi, *An Introduction to Kolmogorov Complexity and Its Applications* (Springer, 2008).

[85] C. Van den Broeck and M. Esposito, Physica A: Statistical Mechanics and its Applications **418**, 6 (2015).

[86] U. Seifert, Reports on Progress in Physics **75**, 126001 (2012).

[87] J. A. Owen, A. Kolchinsky, and D. H. Wolpert, New Journal of Modern Physics (2018).

[88] M. Esposito and C. Van den Broeck, Physical Review E **82**, 011143 (2010).

[89] M. Esposito and C. Van den Broeck, Physical review letters **104**, 090601 (2010).

[90] M. Esposito and C. Van den Broeck, EPL (Europhysics Letters) **95**, 40004 (2011).

[91] C. Van den Broeck and R. Toral, Physical Review E **92**, 012127 (2015).

[92] R. Zia and B. Schmittmann, Journal of Statistical Mechanics: Theory and Experiment **2007**, P07012 (2007).

[93] J. Sethna, *Statistical mechanics: entropy, order parameters, and complexity*, Vol. 14 (Oxford University Press, 2006).

[94] S. Deffner and C. Jarzynski, Physical Review X **3**, 041003 (2013).

[95] H.-H. Hasegawa, J. Ishikawa, K. Takara, and D. Driebe, Physics Letters A **374**, 1001 (2010).

[96] M. Esposito, K. Lindenberg, and C. Van den Broeck, New Journal of Physics **12**, 013013 (2010).

[97] M. Esposito, K. Lindenberg, and C. Van den Broeck, EPL (Europhysics Letters) **85**, 60010 (2009).

[98] P. Pietzonka and U. Seifert, Physical Review Letters **120**, 190602 (2018).

[99] C. Jarzynski, Annu. Rev. Condens. Matter Phys. **2**, 329 (2011).

[100] T. Sagawa, Journal of Statistical Mechanics: Theory and Experiment **2014**, P03025 (2014).

[101] T. Ouldridge, R. Brittain, and P. Rein Ten Wolde, in *Energetics of computing in life and machines*, edited by D. H. Wolpert, C. P. Kempes, P. Stadler, and J. Grochow (SFI Press, 2018).

[102] S. Turgut, Physical Review E **79**, 041102 (2009).

[103] O. Maroney, Physical Review E **79**, 031105 (2009).

[104] P. Faist, F. Dupuis, J. Oppenheim, and R. Renner, arXiv preprint arXiv:1211.1037 (2012).

[105] D. H. Wolpert, Entropy **18**, 138 (2016).

[106] D. H. Wolpert, Entropy **18**, 219 (2016).

[107] M. Esposito, R. Kawai, K. Lindenberg, and C. Van den Broeck, EPL (Europhysics Letters) **89**, 20003 (2010).

[108] A. Kolchinsky and D. H. Wolpert, Journal of Statistical Mechanics: Theory and Experiment , 083202 (2017).

[109] G. Diana, G. B. Bagci, and M. Esposito, Physical Review E **87**, 012111 (2013).

[110] W. H. Zurek, Nature **341**, 119 (1989).

[111] T. Sagawa and M. Ueda, Physical review letters **102**, 250602 (2009).

[112] R. Dillenschneider and E. Lutz, Physical review letters **104**, 198903 (2010).

[113] A. Barato and U. Seifert, Physical review letters **112**, 090601 (2014).

[114] A. B. Boyd, Physical Review X **8** (2018), 10.1103/PhysRevX.8.031036.

[115] R. Kawai, J. M. R. Parrondo, and C. Van den Broeck, Physical review letters **98**, 080602 (2007).

[116] S. Still, D. A. Sivak, A. J. Bell, and G. E. Crooks, Physical review letters **109**, 120604 (2012).

[117] I. Neri, É. Roldán, and F. Jülicher, Physical Review X **7**, 011019 (2017).

[118] T. R. Gingrich and J. M. Horowitz, Physical review letters **119**, 170601 (2017).

[119] K. Takara, H.-H. Hasegawa, and D. Driebe, Physics Letters A **375**, 88 (2010).

[120] D. H. Wolpert, A. Kolchinsky, and J. A. Owen, arXiv preprint arXiv:1708.08494 (2017).

[121] D. H. Wolpert, "Extending Landauer's bound from bit erasure to arbitrary computation," (2016), arXiv:1508.05319 [cond-mat.stat-mech].

[122] A. J. Garner, J. Thompson, V. Vedral, and M. Gu, Physical Review E **95**, 042140 (2017).

[123] M. Esposito, Physical Review E **85**, 041125 (2012).

[124] N. Gershenfeld, IBM Systems Journal **35**, 577 (1996).

[125] N. G. Anderson, I. Ercan, and N. Ganesh, IEEE Transactions on Nanotechnology **12**, 902 (2013).

[126] I. Ercan and N. G. Anderson, IEEE Transactions on Nanotechnology **12**, 1047 (2013).

[127] E. Fredkin and T. Toffoli, International Journal of Theoretical Physics **21**, 219 (1982).

[128] R. Drechsler and R. Wille, in *Progress in VLSI Design and Test* (Springer, 2012) pp. 383–392.

[129] K. S. Perumalla, *Introduction to reversible computing* (CRC Press, 2013).

[130] M. P. Frank, in *Proceedings of the 2nd Conference on Computing Frontiers* (ACM, 2005) pp. 385–390.

[131] N. Ganesh and N. G. Anderson, Physics Letters A **377**, 3266 (2013).

[132] D. Chu and R. Spinney, arXiv preprint arXiv:1806.04875 (2018).

[133] A. B. Boyd, D. Mandal, and J. P. Crutchfield, arXiv preprint arXiv:1708.03030 (2017).

[134] E. Stopnitsky, S. Still, T. Ouldridge, and L. Altenberg, in *Energetics of computing in life and machines*, edited by D. H. Wolpert, C. P. Kempes, P. Stadler, and J. Grochow (SFI Press, 2018).

[135] P. Strasberg, J. Cerrillo, G. Schaller, and T. Brandes, Physical Review E **92**, 042104 (2015).

[136] C. H. Bennett, BioSystems **11**, 85 (1979).

[137] W. H. Zurek and J. P. Paz, Physics Review Letters **72**, 2508 (1994).

[138] J. A. Owen, A. Kolchinsky, and D. H. Wolpert, arXiv preprint arXiv:1709.00765 (2017).

[139] C. Jia, Statistics & Probability Letters **116**, 122 (2016).

[140] P. Lencastre, F. Raischel, T. Rogers, and P. G. Lind, Physical Review E **93**, 032135 (2016).

[141] N. Murphy, R. Petersen, A. Phillips, B. Yordanov, and N. Dalchau, Journal of The Royal Society Interface **15**, 20180283 (2018).

[142] L. Qian and E. Winfree, Science **332**, 1196 (2011).

[143] S. B. Laughlin, Current opinion in neurobiology **11**, 475 (2001).

[144] V. Balasubramanian, D. Kimber, and M. J. B. Ii, Neural computation **13**, 799 (2001).

[145] D. H. Wolpert, C. P. Kempes, P. Stadler, and J. Grochow, eds., *Energetics of computing in life and machines* (SFI Press, 2018).

[146] G. E. Crooks, Physical Review E **60**, 2721 (1999).

[147] G. E. Crooks, Journal of Statistical Physics **90**, 1481 (1998).

[148] T. R. Gingrich, J. M. Horowitz, N. Perunov, and J. L. England, Physical review letters **116**, 120601 (2016).

[149] J. P. Garrahan, Physical Review E **95**, 032134 (2017).

[150] P. Pietzonka, F. Ritort, and U. Seifert, Physical Review E **96**, 012101 (2017).

[151] N. Shiraishi, K. Funo, and K. Saito, arXiv preprint arXiv:1802.06554 (2018).

[152] M. Okuyama and M. Ohzeki, Physical review letters **120**, 070402 (2018).

[153] J. Pearl, *Causality: Models, Reasoning and Inference* (Cambridge University Press, 2000).