Math. Struct. in Comp. Science (2012), *vol.* 22, *pp.* 771–787.
doi:10.1017/S0960129511000521

Algorithmic thermodynamics

JOHN BAEZ[†] and MIKE STAY[‡]

[†]*Department of Mathematics, University of California, Riverside, California 92521, U.S.A.*
[‡]*Computer Science Department, University of Auckland, and Google, 1600 Amphitheatre Pkwy Mountain View, California 94043, U.S.A.*

Received 14 October 2010; revised 30 March 2011

Algorithmic entropy can be viewed as a special case of the entropy studied in statistical mechanics. This viewpoint allows us to apply many techniques developed for use in thermodynamics to the subject of algorithmic information theory. In particular, suppose we fix a universal prefix-free Turing machine and let X be the set of programs that halt for this machine. Then we can regard X as a set of 'microstates', and treat any function on X as an 'observable'. For any collection of observables, we can study the Gibbs ensemble that maximises entropy subject to constraints on the expected values of these observables. We illustrate this by taking the log runtime, length and output of a program as observables analogous to the energy E, volume V and number of molecules N in a container of gas. The conjugate variables of these observables allow us to define quantities we call the 'algorithmic temperature' T, 'algorithmic pressure' P and 'algorithmic potential' μ, since they are analogous to the temperature, pressure and chemical potential. We derive an analogue of the fundamental thermodynamic relation $dE = TdS - PdV + \mu dN$, and use it to study thermodynamic cycles analogous to those for heat engines. We also investigate the values of T, P and μ for which the partition function converges. At some points on the boundary of this domain of convergence, the partition function becomes uncomputable – indeed, at these points the partition function itself has non-trivial algorithmic entropy.

1. Introduction

Many authors (see Bennett *et al.* (1999), Chaitin (1975), Fredkin and Toffoli (1982), Kolmogorov (1965), Levin and Zvonkin (1970), Solomonoff (1964), Szilard (1929) and Tadaki (2008)) have discussed the analogy between algorithmic entropy and entropy as defined in statistical mechanics: that is, the entropy of a probability measure p on a set X. It is perhaps insufficiently appreciated that algorithmic entropy can be viewed as a *special case* of the entropy defined in statistical mechanics – we will describe how to do this in Section 3.

This allows all the basic techniques of thermodynamics to be imported into algorithmic information theory. The key idea is to take X to be some version of 'the set of all programs that eventually halt and output a natural number', and let p be a Gibbs ensemble on X. A Gibbs ensemble is a probability measure that maximises entropy subject to constraints on the mean values of some observables – that is, real-valued functions on X.

In most traditional work on algorithmic entropy, the relevant observable is the length of the program. However, much of the interesting structure of thermodynamics only becomes visible when we consider several observables. When X is the set of programs that halt and output a natural number, some other important observables include the output of the program and the logarithm of its runtime. So, in Section 4 we illustrate how ideas from thermodynamics can be applied to algorithmic information theory using these three observables.

To do this, we consider a Gibbs ensemble of programs that maximises entropy subject to constraints on:

— E, the expected value of the logarithm of the program's runtime (which we treat as analogous to the energy of a container of gas);
— V, the expected value of the length of the program (analogous to the volume of the container); and
— N, the expected value of the program's output (analogous to the number of molecules in the gas).

This measure is of the form

$$p = \frac{1}{Z} e^{-\beta E(x) - \gamma V(x) - \delta N(x)}$$

for certain numbers β, γ, δ, where the normalising factor

$$Z = \sum_{x \in X} e^{-\beta E(x) - \gamma V(x) - \delta N(x)}$$

is called the 'partition function' of the ensemble. The partition function reduces to Chaitin's number Ω when $\beta = 0$, $\gamma = \ln 2$ and $\delta = 0$. This number is uncomputable (Chaitin 1975). However, we will show that the partition function Z is computable when $\beta > 0$, $\gamma \geqslant \ln 2$ and $\delta \geqslant 0$.

We derive an algorithmic analogue of the basic thermodynamic relation

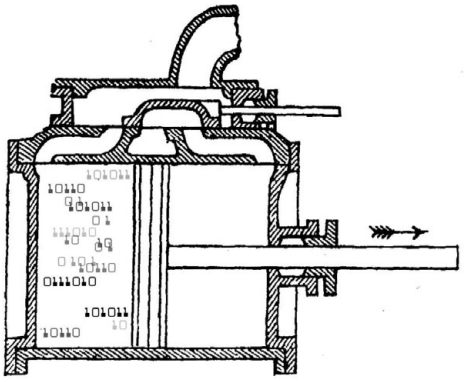
$$dE = TdS - PdV + \mu dN,$$

where:

— S is the entropy of the Gibbs emsemble.
— $T = 1/\beta$ is the 'algorithmic temperature' (which is analogous to the temperature of a container of gas). Roughly speaking, this counts how many times you must double the runtime in order to double the number of programs in the ensemble while holding their mean length and output fixed.
— $P = \gamma/\beta$ is the 'algorithmic pressure' (which is analogous to pressure). This measures the tradeoff between runtime and length. Roughly speaking, it counts how much you need to decrease the mean length to increase the mean log runtime by a specified amount, while holding the number of programs in the ensemble and their mean output fixed.
— $\mu = -\delta/\beta$ is the 'algorithmic potential' (which is analogous to chemical potential). Roughly speaking, this counts how much the mean log runtime increases when you increase the mean output while holding the number of programs in the ensemble and their mean length fixed.

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

Starting from this relation, we derive analogues of Maxwell's relations and consider thermodynamic cycles such as the Carnot cycle or Stoddard cycle. For this we must introduce concepts of 'algorithmic heat' and 'algorithmic work'.



Charles Babbage described a computer powered by a steam engine; we describe a heat engine powered by programs! We admit that the significance of this line of thinking remains a bit mysterious. However, we hope it points the way toward a further synthesis of algorithmic information theory and thermodynamics. We call this hoped-for synthesis 'algorithmic thermodynamics'.

2. Related work

Li and Vitányi use the term 'algorithmic thermodynamics' for the description of physical states using a universal prefix-free Turing machine U. They look at the smallest program p that outputs a description x of a particular microstate to some accuracy, and define the physical entropy to be

$$S_A(x) = (k \ln 2)(K(x) + H_x),$$

where $K(x) = |p|$ and H_x embodies the uncertainty in the actual state given x. They summarise both their own work and subsequent work by others in Chapter eight of their book Li and Vitányi (2008). While they consider $x = U(p)$ to be a microstate, we consider p to be the microstate and x to be the value of the observable U. Then their observables $O(x)$ become observables of the form $O(U(p))$ in our model.

Tadaki (2002) generalised Chaitin's number Ω to a function Ω^D and showed that the value of this function is compressible by a factor of exactly D when D is computable. Calude and Stay (2006b) pointed out that this generalisation was formally equivalent to the partition function of a statistical mechanical system, where temperature played the role of the compressibility factor, and studied various observables of such a system. Tadaki (2008) then explicitly constructed a system with that partition function: given a total length E and number of programs N, the entropy of the system is the log of the number of E-bit strings in $\mathrm{dom}(U)^N$. The temperature is

$$\frac{1}{T} = \left.\frac{\Delta E}{\Delta S}\right|_N.$$

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

The follow-up paper Tadaki (2009) showed that various other quantities, such as the free energy, shared the same compressibility properties as Ω^D. In this paper, we consider multiple variables, which will be necessary for the study thermodynamic cycles, chemical reactions, and so forth.

Manin and Marcolli (2009) derived similar results in a broader context and studied phase transitions in those systems. Manin (2011; 2012) also outlined an ambitious programme for treating the infinite runtimes one finds in undecidable problems as singularities to be removed through the process of renormalisation. In a manner reminiscent of hunting for the proper definition of the 'one-element field' F_{un}, he collected ideas from many different places and considered how they all touch on this central theme. While he mentioned a runtime cutoff as being analogous to an energy cutoff, the renormalisations he presented are uncomputable. In this paper, we take the log of the runtime as being analogous to the energy; the randomness described by Chaitin and Tadaki then arises as the infinite-temperature limit.

3. Algorithmic entropy

To see algorithmic entropy as a special case of the entropy of a probability measure, it is useful to follow Solomonoff (1964) and take a Bayesian viewpoint. In Bayesian probability theory, we always start with a probability measure called a 'prior', which describes our assumptions about the situation at hand before we make any further observations. As we learn more, we may update this prior. This approach suggests that we should define the entropy of a probability measure *relative to another probability measure* – the prior.

A probability measure p on a finite set X is simply a function $p\colon X \to [0,1]$ whose values sum to 1, and its entropy is defined as follows:

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

But we can also define the entropy of p relative to another probability measure q:

$$S(p,q) = -\sum_{x \in X} p(x) \ln \frac{p(x)}{q(x)}.$$

This *relative entropy* has been extensively studied and goes by various other names, including 'Kullback–Leibler divergence' (Kullback and Leibler 1951) and 'information gain' (Rényi 1960).

The term 'information gain' is nicely descriptive. Suppose we initially assume the outcome of an experiment is distributed according to the probability measure q. Suppose we then repeatedly do the experiment and discover its outcome is distributed according to the measure p. Then the information gained is $S(p,q)$.

We can see why this is the case if we think in terms of coding. Suppose X is a finite set of signals that are randomly emitted by some source and that we wish to encode these signals as efficiently as possible in the form of bit strings. Suppose the source emits the signal x with probability $p(x)$, but we erroneously believe it is emitted with probability $q(x)$. Then $S(p,q)/\ln 2$ is the expected extra message-length per signal that is required if

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

we use a code that is optimal for the measure q instead of a code that is optimal for the true measure, p.

The ordinary entropy $S(p)$ is, up to a constant, just the relative entropy in the special case where the prior assigns an equal probability to each outcome. In other words,

$$S(p) = S(p, q_0) + S(q_0)$$

when q_0 is the so-called 'uninformative prior', with $q_0(x) = 1/|X|$ for all $x \in X$.

We can also define relative entropy when the set X is countably infinite. As before, a probability measure on X is a function $p : X \to [0, 1]$ whose values sum to 1. And as before, if p and q are two probability measures on X, the entropy of p relative to q is defined by

$$S(p, q) = -\sum_{x \in X} p(x) \ln \frac{p(x)}{q(x)}. \tag{1}$$

But now the role of the prior becomes more clear, because there is no probability measure that assigns the same value to each outcome!

In the following, we will take X to be, roughly speaking, the set of all programs that eventually halt and output a natural number. As we shall see, while this set is countably infinite, there are still some natural probability measures on it, and we may take them as priors.

To make this precise, we recall the concept of a universal prefix-free Turing machine. In the following, we will use *string* to mean a bit string, that is, a finite, possibly empty, list of 0's and 1's. If x and y are strings, let $x||y$ be the concatenation of x and y. A *prefix* of a string z is a substring beginning with the first letter, that is, a string x such that $z = x||y$ for some y. A *prefix-free* set of strings is one in which no element is a prefix of any other. The *domain* $\mathrm{dom}(M)$ of a Turing machine M is the set of strings that cause M to halt eventually. We call the strings in $\mathrm{dom}(M)$ *programs*. We assume that when the M halts on the program x it outputs a natural number $M(x)$. Thus we may think of the machine M as giving a function $M : \mathrm{dom}(M) \to \mathbb{N}$.

A *prefix-free Turing machine* is one whose halting programs form a prefix-free set. A prefix-free machine U is *universal* if for any prefix-free Turing machine M, there exists a constant c such that for each string x there exists a string y with

$$U(y) = M(x) \quad \text{and} \quad |y| < |x| + c.$$

Let U be a universal prefix-free Turing machine. Then we can define some probability measures on $X = \mathrm{dom}(U)$ as follows. Let

$$|\cdot| : X \to \mathbb{N}$$

be the function assigning to each bit string its length. Then there is for any constant $\gamma > \ln 2$ a probability measure p given by

$$p(x) = \frac{1}{Z} e^{-\gamma |x|}.$$

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

Here the normalisation constant Z is chosen to make the numbers $p(x)$ sum to 1:

$$Z = \sum_{x \in X} e^{-\gamma |x|}.$$

It is worth noting that for computable real numbers $\gamma \geqslant \ln 2$, the normalisation constant Z is uncomputable (Tadaki 2002). Indeed, when $\gamma = \ln 2$, Z is Chaitin's famous number Ω – we will return to this issue in Section 4.5.

Let us assume that each program prints out some natural number as its output. Thus we have a function

$$N : X \to \mathbb{N}$$

where $N(x)$ equals i when program x prints out the number i. We may use this function to 'push forward' p to a probability measure q on the set n. Explicitly,

$$q(i) = \sum_{x \in X : N(x) = i} e^{-\gamma |x|} .$$

In other words, if i is some natural number, $q(i)$ is the probability that a program randomly chosen according to the measure p will print out this number.

Given any natural number n, there is a probability measure δ_n on n that assigns probability 1 to this number:

$$\delta_n(m) = \begin{cases} 1 & \text{if } m = n \\ 0 & \text{otherwise.} \end{cases}$$

We can now compute the entropy of δ_n relative to q:

$$\begin{aligned} S(\delta_n, q) &= -\sum_{i \in n} \delta_n(i) \ln \frac{\delta_n(i)}{q(i)} \\ &= -\ln \left(\sum_{x \in X \,:\, N(x) = n} e^{-\gamma |x|} \right) + \ln Z. \end{aligned} \tag{2}$$

Since the quantity $\ln Z$ is independent of the number n, and uncomputable, it makes sense to focus attention on the other part of the relative entropy:

$$-\ln \left(\sum_{x \in X \,:\, N(x) = n} e^{-\gamma |x|} \right).$$

If we take $\gamma = \ln 2$, this is precisely the *algorithmic entropy* of the number n (Chaitin 1976; Levin and Zvonkin 1970). So, up to the additive constant $\ln Z$, we can see that *algorithmic entropy is a special case of relative entropy*.

One way to think about entropy is as a measure of surprise: if you can predict what comes next, that is, if you have a program that can compute it for you, then then you are not surprised. For example, the first 2000 bits of the binary fraction for 1/3 can be produced with this short Python program:

```
print "01" * 1000
```

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

But if the number is complicated, if every bit is surprising and unpredictable, then the shortest program to print the number does not do any computation at all! It just looks something like

```
print "10100001100101001010010100010111110110110100101O"
```

Levin's coding theorem (Levin 1974) says that the difference between the algorithmic entropy of a number and its *Kolmogorov complexity* – the length of the shortest program that outputs it – is bounded by a constant that only depends on the programming language.

So, what we see here is that up to some error bounded by a constant, *Kolmogorov complexity is information gain*: the information gained upon learning a number, if our prior assumption was that this number is the output of a program randomly chosen according to the measure p where $\gamma = \ln 2$.

More importantly, we have seen that algorithmic entropy is not just *analogous* to entropy as defined in statistical mechanics: it is a *special case*, as long as we take seriously the Bayesian philosophy that entropy should be understood as relative entropy. This realisation opens up the possibility of taking many familiar concepts from thermodynamics, expressed in the language of statistical mechanics, and finding their counterparts in the realm of algorithmic information theory.

But in order to proceed we must also understand more precisely the role of the measure p. In the next section, we shall see that this type of measure is already familiar in statistical mechanics: it is a Gibbs ensemble.

4. Algorithmic thermodynamics

Suppose we have a countable set X, finite or infinite, and suppose $C_1, \ldots, C_n : X \to \mathbb{R}$ is some collection of functions. Then we may seek a probability measure p that maximises entropy subject to the constraints that the mean value of each observable C_i is a given real number $\overline{C}_i$:

$$\sum_{x \in X} p(x)\, C_i(x) = \overline{C}_i.$$

As nicely discussed in Jaynes (1957; 2003), the solution, if it exists, is the so-called *Gibbs ensemble*:

$$p(x) = \frac{1}{Z} e^{-(s_1 C_1(x) + \cdots + s_n C_n(x))}$$

for some numbers $s_i \in \mathbb{R}$ depending on the desired mean values $\overline{C}_i$. Here the normalising factor Z is called the *partition function*:

$$Z = \sum_{x \in X} e^{-(s_1 C_1(x) + \cdots + s_n C_n(x))} .$$

In thermodynamics, X represents the set of *microstates* of some physical system. A probability measure on X is also known as an *ensemble*. Each function $C_i : X \to \mathbb{R}$ is called an *observable*, and the corresponding quantity s_i is called the *conjugate variable* of that observable. For example, the conjugate of the energy E is the inverse of temperature T, in units where Boltzmann's constant equals 1. The conjugate of the volume V, of a

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

THERMODYNAMICS

Observable	Conjugate Variable
energy: E	$\frac{1}{T}$
volume: V	$\frac{P}{T}$
number: N_i	$-\frac{\mu_i}{T}$

Table 1.

piston full of gas, for example, is the pressure P divided by the temperature. And in a gas containing molecules of various types, the conjugate of the number N_i of molecules of the ith type is minus the 'chemical potential' μ_i, again divided by temperature. For easy reference, we list these observables and their conjugate variables in Table 1.

Now let us return to the case where $X = \mathrm{dom}(U)$. Recalling that programs are bit strings, one important observable for programs is the length:

$$|\cdot| : X \to \mathbb{N}.$$

We have already seen the measure

$$p(x) = \frac{1}{Z} e^{-\gamma |x|}.$$

Now its significance should be clear! This is the probability measure on programs that maximises entropy subject to the constraint that the mean length is some constant ℓ:

$$\sum_{x \in X} p(x)\, |x| = \ell.$$

So, γ is the conjugate variable to program length.

There are, however, other important observables that can be defined for programs, and each of these has a conjugate quantity. To make the analogy to thermodynamics as vivid as possible, let us arbitrarily choose two more observables and treat them as analogues of energy and the number of some type of molecule. Two of the most obvious observables are 'output' and 'runtime'. Since Levin's computable complexity measure (Levin 1973) uses the logarithm of runtime as a kind of 'cutoff' reminiscent of an energy cutoff in renormalisation, we shall arbitrarily choose the log of the runtime to be analogous to the energy, and denote it by

$$E : X \to [0, \infty)$$

Following the chart in Table 1, we use $1/T$ to stand for the variable conjugate to E. We arbitrarily treat the output of a program as analogous to the number of a certain kind of molecule, and denote it as

$$N : X \to \mathbb{N}.$$

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

ALGORITHMS

Observable	Conjugate Variable
log runtime: E	$\frac{1}{T}$
length: V	$\frac{P}{T}$
output: N	$-\frac{\mu}{T}$

Table 2.

We use $-\mu/T$ to stand for the conjugate variable of N. Finally, as already hinted, we denote program length as

$$V : X \to \mathbb{N}$$

so that in terms of our earlier notation, $V(x) = |x|$. We use P/T to stand for the variable conjugate to V – see Table 2.

Before proceeding, we should emphasise that our choice of analogies here was somewhat arbitrary and are just an illustration of the application of thermodynamics to the study of algorithms. There may or may not be a specific 'best' mapping between observables for programs and observables for a container of gas! Indeed, Tadaki (2008) explored another analogy, where length rather than log run time is treated as the analogue of energy. There is nothing wrong with this. However, he did not introduce enough other observables to see the whole structure of thermodynamics, as developed in Sections 4.1–4.2 below.

Having made our choice of observables, we define the partition function by

$$Z = \sum_{x \in X} e^{-\frac{1}{T}(E(x)+PV(x)-\mu N(x))} .$$

When this sum converges, we can define a probability measure on X, the Gibbs ensemble, by

$$p(x) = \frac{1}{Z} e^{-\frac{1}{T}(E(x)+PV(x)-\mu N(x))} .$$

Both the partition function and the probability measure are functions of T, P and μ. From these we can compute the mean values of the observables to which these variables are conjugate:

$$\overline{E} = \sum_{x \in X} p(x)\, E(x)$$

$$\overline{V} = \sum_{x \in X} p(x)\, V(x)$$

$$\overline{N} = \sum_{x \in X} p(x)\, N(x)$$

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

In certain ranges, the map $(T, P, \mu) \mapsto (\overline{E}, \overline{V}, \overline{N})$ will be invertible. This allows us to alternatively think of Z and p as functions of $\overline{E}, \overline{V}$ and $\overline{N}$. In this situation, it is typical to abuse language by omitting the overlines denoting 'mean value'.

4.1. *Elementary relations*

The entropy S of the Gibbs ensemble is given by

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

We may think of this as a function of T, P and μ, or alternatively, as explained above, as functions of the mean values E, V and N. Then simple calculations, familiar from statistical mechanics (Reif 1965), show that

$$\left.\frac{\partial S}{\partial E}\right|_{V,N} = \frac{1}{T} \tag{3}$$

$$\left.\frac{\partial S}{\partial V}\right|_{E,N} = \frac{P}{T} \tag{4}$$

$$\left.\frac{\partial S}{\partial N}\right|_{E,V} = -\frac{\mu}{T}. \tag{5}$$

We may summarise all these by writing

$$dS = \frac{1}{T} dE + \frac{P}{T} dV - \frac{\mu}{T} dN$$

or, equivalently,

$$dE = T dS - P dV + \mu dN. \tag{6}$$

Starting from the latter equation, we see

$$\left.\frac{\partial E}{\partial S}\right|_{V,N} = T \tag{7}$$

$$\left.\frac{\partial E}{\partial V}\right|_{S,N} = -P \tag{8}$$

$$\left.\frac{\partial E}{\partial N}\right|_{S,V} = \mu. \tag{9}$$

With these definitions, we can start to get a feel for what the conjugate variables are measuring. To build intuition, it is useful to think of the entropy S as roughly the logarithm of the number of programs whose log runtimes, length and output lie in small ranges $E \pm \Delta E$, $V \pm \Delta V$ and $N \pm \Delta N$. This is at best approximately true, but in ordinary thermodynamics this approximation is commonly employed and yields spectacularly good results. That is why in thermodynamics people often say the entropy is the logarithm of the number of microstates for which the observables E, V and N lie within a small range of their specified values (Reif 1965).

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

If you allow programs to run longer, more of them will halt and give an answer. The *algorithmic temperature* T is roughly the number of times you have to double the runtime in order to double the number of ways to satisfy the constraints on length and output.

The *algorithmic pressure* P measures the trade-off between runtime and length (Calude and Stay 2006a): if you want to keep the number of ways to satisfy the constraints constant, then the freedom gained by having longer runtimes has to be counterbalanced by shortening the programs. This is analogous to the pressure of gas in a piston: if you want to keep the number of microstates of the gas constant, then the freedom gained by increasing its energy has to be counterbalanced by decreasing its volume.

Finally, the *algorithmic potential* describes the relation between log runtime and output: it is a quantitative measure of the principle that most large outputs must be produced by long programs.

4.2. *Thermodynamic cycles*

One of the first applications of thermodynamics was in the analysis of heat engines. The underlying mathematics applies equally well to algorithmic thermodynamics. Suppose C is a loop in (T, P, μ) space and that we are in a region that can also be coordinatised by the variables E, V, N. Then the change in *algorithmic heat* around the loop C is defined to be

$$\Delta Q = \oint_C T dS.$$

Suppose the loop C bounds a surface Σ. Then Stokes' theorem implies that

$$\Delta Q = \oint_C T dS = \int_\Sigma dT dS.$$

However, Equation (6) implies that

$$dT dS = d(T dS) = d(dE + P dV - \mu dN) = +dP dV - d\mu dN$$

since $d^2 = 0$. So we have

$$\Delta Q = \int_\Sigma (dP dV - d\mu dN),$$

or using Stokes' theorem again,

$$\Delta Q = \int_C (P dV - \mu dN). \tag{10}$$

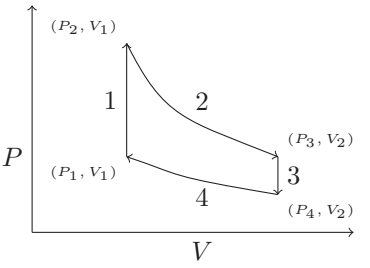
In ordinary thermodynamics, N is constant for a heat engine using gas in a sealed piston. In this situation we have

$$\Delta Q = \int_C P dV.$$

This equation says that the change in heat of the gas equals the work done on the gas, or, equivalently, minus the work done *by* the gas. So, in algorithmic thermodynamics, let us define $\int_C P dV$ to be the *algorithmic work* done on our ensemble of programs as we carry it around the loop C – beware, this concept is unrelated to 'computational work', which means the amount of computation done by a program as it runs.

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

To see an example of a cycle in algorithmic thermodynamics, consider the analogue of the heat engine patented by Stoddard in 1919 (Stoddard 1919). Here we fix N to a constant value and consider the following loop in the PV plane:



We start with an ensemble with algorithmic pressure P_1 and mean length V_1. We then trace out a loop consisting of four parts:

(1) *Isometric*: We increase the pressure from P_1 to P_2 while keeping the mean length constant. No algorithmic work is done on the ensemble of programs during this step.

(2) *Isentropic*: We increase the length from V_1 to V_2 while keeping the number of halting programs constant. High pressure means that we are operating in a range of runtimes where if we increase the length a little bit, many more programs halt. In order to keep the number of halting programs constant, we need to shorten the runtime significantly. As we gradually increase the length and lower the runtime, the pressure drops to P_3. The total difference in log runtime is the algorithmic work done on the ensemble during this step.

(3) *Isometric*: Now we decrease the pressure from P_3 to P_4 while keeping the length constant. No algorithmic work is done during this step.

(4) *Isentropic*: Finally, we decrease the length from V_2 back to V_1 while keeping the number of halting programs constant. Since we are at low pressure, we need only increase the runtime a little. As we gradually decrease the length and increase the runtime, the pressure rises slightly back to P_1. The total increase in log runtime is minus the algorithmic work done on the ensemble of programs during this step.

The total algorithmic work done on the ensemble per cycle is the difference in log runtimes between Steps 2 and 4.

4.3. *Further relations*

From the elementary thermodynamic relations in Section 4.1, we can derive various others. For example, the so-called 'Maxwell relations' are obtained by computing the second derivatives of thermodynamic quantities in two different orders and then applying the basic derivative relations given in Equations (7–9). While trivial to prove, these relations say some things about algorithmic thermodynamics which may not seem intuitively obvious.

We will just give one example here. Since mixed partials commute, we have

$$\left.\frac{\partial^2 E}{\partial V \partial S}\right|_N = \left.\frac{\partial^2 E}{\partial S \partial V}\right|_N.$$

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

Using Equation (7), the left-hand side can be computed as follows:

$$\left.\frac{\partial^2 E}{\partial V \partial S}\right|_N = \left.\frac{\partial}{\partial V}\right|_{S,N} \left.\frac{\partial E}{\partial S}\right|_{V,N} = \left.\frac{\partial T}{\partial V}\right|_{S,N}.$$

Similarly, we can compute the right-hand side with the help of Equation (8):

$$\left.\frac{\partial^2 E}{\partial S \partial V}\right|_N = \left.\frac{\partial}{\partial S}\right|_{V,N} \left.\frac{\partial E}{\partial V}\right|_{S,N} = -\left.\frac{\partial P}{\partial S}\right|_{V,N}.$$

As a result, we obtain

$$\left.\frac{\partial T}{\partial V}\right|_{S,N} = -\left.\frac{\partial P}{\partial S}\right|_{V,N}.$$

We can also derive interesting relations involving derivatives of the partition function. These become more manageable if we rewrite the partition function in terms of the conjugate variables of the observables E, V and N:

$$\beta = \frac{1}{T}, \quad \gamma = \frac{P}{T}, \quad \delta = -\frac{\mu}{T}. \tag{11}$$

Then we have

$$Z = \sum_{x \in X} e^{-\beta E(x) - \gamma V(x) - \delta N(x)}$$

Simple calculations, which are standard in statistical mechanics (Reif 1965), then allow us to compute the mean values of observables as derivatives of the logarithm of Z with respect to their conjugate variables. Here we will revert to using overlines to denote mean values:

$$\overline{E} = \sum_{x \in X} p(x)\, E(x) = -\frac{\partial}{\partial \beta} \ln Z$$

$$\overline{V} = \sum_{x \in X} p(x)\, V(x) = -\frac{\partial}{\partial \gamma} \ln Z$$

$$\overline{N} = \sum_{x \in X} p(x)\, N(x) = -\frac{\partial}{\partial \delta} \ln Z.$$

We can go further and compute the variance of these observables using second derivatives:

$$(\Delta E)^2 = \sum_{x \in X} p(x)(E(x)^2 - \overline{E}^2) = \frac{\partial^2}{\partial^2 \beta} \ln Z,$$

and similarly for V and N. Higher moments of E, V and N can also be computed by taking higher derivatives of $\ln Z$.

4.4. *Convergence*

So far we have postponed the crucial question of convergence: for which values of T, P and μ does the partition function Z converge? For this it is most convenient to treat Z as

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

a function of the variables β, γ and δ introduced in Equation (11). The question is then, for which values of β, γ and δ does the partition function converge?

(1) When $\beta = \gamma = \delta = 0$, the contribution of each program is 1. Since there are infinitely many halting programs, $Z(0,0,0)$ does not converge.

(2) When $\beta = 0, \gamma = \ln 2$ and $\delta = 0$, the partition function converges to Chaitin's number

$$\Omega = \sum_{x \in X} 2^{-V(x)}.$$

To see that the partition function converges in this case, consider the following mapping of strings to segments of the unit interval:

<table>
<tr><td colspan="8">empty</td></tr>
<tr><td colspan="4">0</td><td colspan="4">1</td></tr>
<tr><td colspan="2">00</td><td colspan="2">01</td><td colspan="2">10</td><td colspan="2">11</td></tr>
<tr><td>000</td><td>001</td><td>010</td><td>011</td><td>100</td><td>101</td><td>110</td><td>111</td></tr>
</table>

⋮

Each segment consists of all the real numbers whose binary expansion begins with that string. For example, the set of real numbers whose binary expansion begins $0.101\ldots$ is $[0.101, 0.110)$ and has measure $2^{-|101|} = 2^{-3} = 1/8$. Since the set of halting programs for our universal machine is prefix-free, we never count any segment more than once, so the sum of all the segments corresponding to halting programs is at most 1.

(3) Tadaki (2002) showed that the expression

$$\sum_{x \in X} e^{-\gamma V(x)}$$

converges for $\gamma \geqslant \ln 2$ but diverges for $\gamma < \ln 2$. It follows that $Z(\beta, \gamma, \delta)$ converges whenever $\gamma \geqslant \ln 2$ and $\beta, \delta \geqslant 0$.

(4) When $\beta > 0$ and $\gamma = \delta = 0$, convergence depends on the machine. There are machines where infinitely many programs halt immediately. For these, $Z(\beta, 0, 0)$ does not converge. However, there are also machines where program x takes at least $V(x)$ steps to halt, and for these machines, $Z(\beta, 0, 0)$ will converge when $\beta \geqslant \ln 2$. Other machines take much longer to run. For these, $Z(\beta, 0, 0)$ will converge for even smaller values of β.

(5) Finally, when $\beta = \gamma = 0$ and $\delta > 0$, we get that $Z(\beta, \gamma, \delta)$ fails to converge, since there are infinitely many programs that halt and output 0.

4.5. *Computability*

Even when the partition function Z converges, it may not be computable. The theory of computable real numbers was independently introduced by Church, Post and Turing, and later blossomed into the field of computable analysis (Pour-El and Richards 1989). We will only need the basic definition: a real number a is *computable* if there is a recursive

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

function that maps any natural number $n > 0$ to an integer $f(n)$ such that

$$\frac{f(n)}{n} \leqslant a \leqslant \frac{f(n)+1}{n}.$$

In other words, for any $n > 0$, we can compute a rational number that approximates a with an error of at most $1/n$. This definition can be formulated in various other equivalent ways: for example, the computability of binary digits.

Chaitin (1975) proved that the number

$$\Omega = Z(0, \ln 2, 0)$$

is uncomputable. In fact, he showed that for any universal machine, the values of all but finitely many bits of Ω are not only uncomputable, but random: knowing the value of some of them tells you nothing about the rest – they are independent, like separate flips of a fair coin.

More generally, for any computable number $\gamma \geqslant \ln 2$, we have $Z(0, \gamma, 0)$ is 'partially random' in the sense of Tadaki (Calude *et al.* 2004; Tadaki 2002). This deserves a word of explanation. A fixed formal system with finitely many axioms can only prove that finitely many bits of $Z(0, \gamma, 0)$ have the values they do; after that, one has to add more axioms or rules to the system to make any progress. The number Ω is completely random in the following sense: for each bit of axiom or rule one adds, one can prove that at most one more bit of its binary expansion has the value it does. So the most efficient way to prove the values of these bits is simply to add them as axioms! But for $Z(0, \gamma, 0)$ with $\gamma > \ln 2$, the ratio of bits of axiom per bits of sequence is less than 1. In fact, Tadaki showed that for any computable $\gamma \geqslant \ln 2$, the ratio can be reduced to exactly $(\ln 2)/\gamma$.

On the other hand, $Z(\beta, \gamma, \delta)$ is computable for all computable real numbers $\beta > 0$, $\gamma \geqslant \ln 2$ and $\delta \geqslant 0$. The reason is that $\beta > 0$ exponentially suppresses the contribution of machines with long runtimes, eliminating the problem posed by the undecidability of the halting problem. The fundamental insight here is due to Levin (Levin 1973). His idea was to 'dovetail' all programs: on turn n, run each of the first n programs a single step and look to see which ones have halted. As they halt, add their contribution to the running estimate of Z. For any $k \geqslant 0$ and turn $t \geqslant 0$, let k_t be the location of the first zero bit after position k in the estimation of Z. Then because $-\beta E(x)$ is a monotonically decreasing function of the runtime and decreases faster than k_t, there will be a time step where the total contribution of all the programs that have not halted yet is less than 2^{-k_t}.

5. Conclusions

We will just mention three of the many further directions to explore. First, as already mentioned, the 'Kolmogorov complexity' (Kolmogorov 1965) of a number n is the number of bits in the shortest program that produces n as output. However, a very short program that runs for a million years before giving an answer is not very practical. To address this problem, the *Levin complexity* (Levin 1974) of n is defined using the program's length plus the logarithm of its runtime, again minimised over all programs that produce n as output. Unlike the Kolmogorov complexity, the Levin complexity is computable. But like the Kolmogorov complexity, the Levin complexity can be seen as a *relative entropy*

https://doi.org/10.1017/S0960129511000521 Published online by Cambridge University Press

– at least, up to some error bounded by a constant. The only difference is that now we compute this entropy relative to a different probability measure: instead of using the Gibbs distribution at infinite algorithmic temperature, we drop the temperature to $\ln 2$. Indeed, the Kolmogorov and Levin complexities are just two examples from a continuum of options. By adjusting the algorithmic pressure and temperature, we get complexities involving other linear combinations of length and log runtime. The same formalism works for complexities involving other observables: for example, the maximum amount of memory the program uses while running.

Second, instead of considering Turing machines that output a single natural number, we can consider machines that output a finite list of natural numbers $(N_1, \ldots, N_j)$; we can treat these as populations of different 'chemical species' and define algorithmic potentials for each of them. Processes analogous to chemical reactions are paths through this space that preserve certain invariants of the lists. With chemical reactions we can consider things like internal combustion cycles.

Finally, in ordinary thermodynamics, the partition function Z is simply a number after we fix the values of the conjugate variables. The same is true in algorithmic thermodynamics. However, in algorithmic thermodynamics, it is natural to express this number in binary and inquire about the algorithmic entropy of the first n bits. For example, we have seen that for suitable values of temperature, pressure and chemical potential, Z is Chaitin's number Ω. For each universal machine there exists a constant c such that the first n bits of the number Ω have at least $n - c$ bits of algorithmic entropy with respect to that machine. Tadaki (2002) generalised this computation to other cases.

So, *in algorithmic thermodynamics, the partition function itself has non-trivial entropy*. Tadaki has shown that the same is true for algorithmic pressure (which in his analogy he calls 'temperature'). This reflects the self-referential nature of computation. It would be worthwhile to understand this more deeply.

Acknowledgements

We would like to thank Leonid Levin and the denizens of the *n*-Category Café for useful comments. Mike Stay would also like to thank Cristian Calude for many discussions on algorithmic information theory. John Baez would also like to thank Bruce Smith for discussions on relative entropy and Mark Smith for conversations on physics and information theory, as well as for giving him a copy of Reif's *Fundamentals of Statistical and Thermal Physics*.

References

Bennett, C. H, Gàcs, P., Li, M., Vitányi, M. B. and Zurek, W. H. (1998) Information distance. *IEEE Transactions on Information Theory* **44** 1407–1423.

Calude, C. S. (2002) *Information and Randomness: An Algorithmic Perspective*, Springer-Verlag.

Calude, C. S. and Stay, M. A. (2006) Most programs stop quickly or never halt. *Advances in Applied Mathematics* **40** 295–308.

Calude, C. S. and Stay, M. A. (2006) Natural halting probabilities, partial randomness, and zeta functions. *Information and Computation* **204** 1718–1739.

Calude, C. S., Staiger, L. and Terwijn, S. A. (2006) On partial randomness. *Annals of Pure and Applied Logic* **138** 20–30.
Chaitin, G. (1975) A theory of program size formally identical to information theory. *Journal of the ACM* **22** 329–340.
Chaitin, G. (1976) Algorithmic entropy of sets. *Computers and Mathematics with Applications* **2** 233–245.
Roberts, C. P. (2001) *The Bayesian Choice: From Decision-Theoretic Foundations to Computational Implementation*, Springer-Verlag.
Fredkin, E. and Toffoli, T. (1982) Conservative logic. *International Journal of Theoretical Physics* **21** 219–253.
Janyes, E. T. (1957) Information theory and statistical mechanics. *Physical Review* **106** 620–630.
Janyes, E. T. (2003) *Probability Theory: The Logic of Science*, Cambridge University Press.
Kolmogorov, A. N. (1965) Three approaches to the definition of the quantity of information. *Problems of Information Transmission* **1** 3–11.
Kullback, S. and Leibler, R. A. (1951) On information and sufficiency. *Annals of Mathematical Statistics* **22** 79–86.
Levin, L. A. (1973) Universal sequential search problems. *Problems of Information Transmission* **9** 265–266.
Levin, L. A. (1974) Laws of information conservation (non-growth) and aspects of the foundation of probability theory. *Problems of Information Transmission* **10** 206–210.
Levin, L. A. and Zvonkin, A. K. (1970) The complexity of finite objects and the development of the concepts of information and randomness by means of the theory of algorithms. *Russian Mathematics Surveys* **256** 83–124.
Li, M. and Vitányi, P. (2008) *An Introduction to Kolmogorov Complexity Theory and its Applications*, Springer-Verlag.
Manin, Y. (2011) Renormalization and computation I: Motivation and background. In: Loday, J.-L. and Vallette, B. (eds.) OPERADS 2009. *Séminaires et Congrès* **26**, Société Mathématique de France 181–233.
Manin, Y. (2009) Renormalisation and computation II: time cut-off and the halting problem. *Mathematical Structures in Computer Science* (this issue).
Manin, Y. and Marcolli, M. (2009) Error-correcting codes and phase transitions. (Available as arXiv:0910.5135.)
Pour-El, M. B. and Richards, J. I. (1989) *Computability in Analysis and Physics*, Springer Verlag.
Reif, F. (1965) *Fundamentals of Statistical and Thermal Physics*, McGraw-Hill.
Rényi, A. (1960) On measures of information and entropy. In: Neyman, J. (ed.) *Proceedings of the 4th Berkeley Symposium on Mathematical Statistics and Probability* **1** 547–561.
Solomonoff, R. J. (1964) A formal theory of inductive inference, part I. *Information and Control* **7** 1–22.
Stoddard, E. J. (1919) Apparatus for obtaining power from compressed air, US Patent 1,926,463.
Szilard, L. (1929) On the decrease of entropy in a thermodynamic system by the intervention of intelligent beings. *Zeitschrift für Physik* **53** 840–856. (English translation (2003) In: Leff, H. S. and Rex A. F. (eds.) *Maxwell's Demon 2: Entropy, Information, Computing*, Adam Hilger.)
Tadaki, K. (2002) A generalization of Chaitin's halting probability Ω and halting self-similar sets. *Hokkaido Mathematical Journal* **31** 219–253.
Tadaki, K. (2008) A statistical mechanical interpretation of algorithmic information theory. In: Local Proceedings of Computability in Europe 2008, University of Athens, Greece 425–434. (Available as arXiv:0801.4194.)
Tadaki, K. (2009) A statistical mechanical interpretation of algorithmic information theory III: Composite systems and fixed points. (Available as arXiv:0904.0973.)