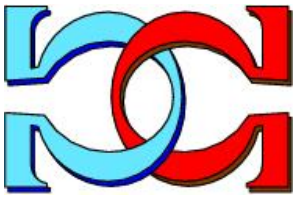
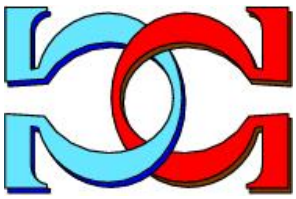
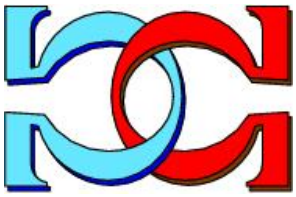
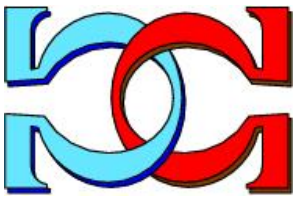






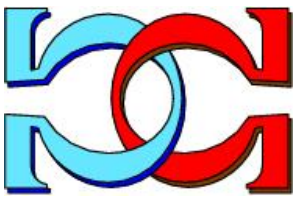
CDMTCS
Research
Report
Series

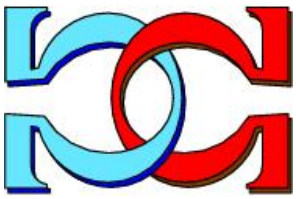
An Algorithmic Heat Engine



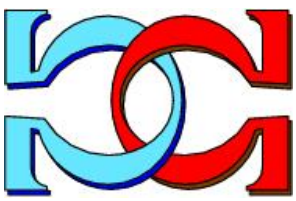
Michael Stay

Google and University of Aukland





CDMTCS-564
August 2022



Centre for Discrete Mathematics and
Theoretical Computer Science

An Algorithmic Heat Engine

Michael Stay

Abstract

We construct an algorithmic heat engine and use it to increase the average runtime of programs in a distribution.

This note is in honor of Cristian Calude's 70th birthday.

1 Introduction

Calude and Stay [2] pointed out that Tadaki's parametrization [5] of Chaitin's Omega number [3] 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 [6] then explicitly constructed a system with that partition function: given a self-delimiting universal Turing machine U, a total length E, and a number of programs N, Tadaki defined the entropy of the system to be the log of the number of E-bit strings in $\mathrm{dom}(\mathrm{U})^{\mathrm{N}}$; the corresponding "temperature" is

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

In follow-up papers [6, 7], Tadaki showed that various other quantities like the free energy shared the same compressibility properties as Ω^D. Baez and Stay [1] considered multiple variables, which is necessary for thermodynamic cycles, chemical reactions, and so forth.

We can choose our favorite programming language and consider the set of programs that take no input, run for a while, then stop. We call this set "the set of halting programs". For technical reasons, we also assume that the programs are "prefix-free". Most real programming languages are prefix-free because there is something like a "makefile" that describes how to combine the files together, and the source code itself lives on a file system that describes the length of each file. What we call "the program" is really all the data necessary to build the result,

assembled in such a way that the execution of the program reads neither too few nor too many bits.

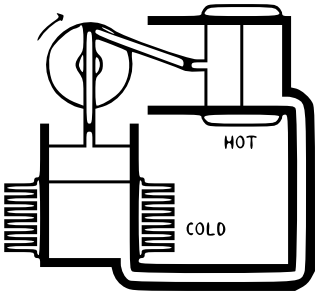
There are sensible questions one can ask about such a program, like, “How long is it?" “How many instructions of the CPU did it take to finish?" “What did it print?" or “How much memory did it use?" We can take these to be our observables and do everything just as one does in physical systems, getting a partition function for a Turing machine.

The arbitrary analogy we will use here is laid out in the following table:

Computer	Piston	Variable	Conjugate	Conjugate variable
runtime	energy	E	temperature	$\beta = 1/T$
length	volume	V	pressure	$\gamma = P/T$

In this note, we choose a universal machine, lay out an explicit thermodynamic process, and compute exactly how much work is being done on each cycle.

The alpha Stirling engine is a heat engine with two pistons, one connected to a hot reservoir and the other to a cold one:



First, the vanes cool the gas in the lower piston, lowering the pressure and allowing the upper piston to move to the right. The gas continues to cool, allowing the lower piston to move down. The flywheel turns, moving the upper piston to the left, drawing all the gas into the upper piston, where it heads and expands. The expanding gas pushes the lower piston up and returns the engine to the initial configuration.

Given the ability to sample from a large cool distribution and a large hot one, we can build an algorithmic Stirling engine with a piston distribution that does “work" on a fourth distribution, the system. In each cycle of the engine above, the flywheel makes a full revolution; it could be attached to a cable to lift something or attached to a magnet to generate electricity.

1. (Isometric) We put the piston system into thermal contact with the cool system. For the program analogy, this means that we sample the set of pairs of programs subject to the constraint that the average total runtime is the

sum of the average runtimes of the cool reservoir and the piston, and the length of the piston program does not change. Since the "cool" reservoir has programs with much less total runtime than the piston, this will tend to decrease the runtime of the piston distribution.

2. (Isothermic) We sample the set of programs subject to the constraint that the average runtime does not change. Since the piston is at low pressure, the length of the programs will decrease.

3. (Isometric) We put the piston into thermal contact with the hot system. Again, we sample the set of pairs of programs subject to the constraint that the average total runtime is the sum of the average runtime of the cool reservoir and the piston, and the length of the piston program does not change. Since the "hot" reservoir has programs with much more total runtime than the piston, this will tend to increase the runtime of the piston distribution.

4. (Isothermic) We sample the set of programs subject to the constraint that the average runtime does not change. Since the piston is under high pressure, the average length of the piston distribution will increase.

Because the increase in runtime in step 3 is greater than the decrease in step 1, the runtime of the piston distribution will increase over time.

2 The SK combinator calculus

We use the SK combinator calculus [4] for our universal machine. The S combinator obeys the rule `(((Sx)y)z) -> ((xz)(yz))` while the K combinator obeys `((Kx)y) -> x`. The smallest programs halt immediately because the combinators do not have enough inputs to trigger reduction:

```
S          (S(KK))
K          (K(SS))
(SS)       (K(SK))
(SK)       (K(KS))
(KS)       (K(KK))
(KK)       ((SS)S)
(S(SS))    ((SS)K)
(S(SK))    ((SK)S)
(S(KS))    ((SK)K)
```

Figure 1: Shortest programs that halt immediately

The first programs that reduce all involve the K combinator applied to two others.

```
((KS)S) -> S
((KS)K) -> S
((KK)S) -> K
((KK)K) -> K
```

Figure 2: Shortest programs that reduce

The first identity is `((SK)S)` because `(((SK)S)x) -> ((Kx)(Sx)) -> x`. The first program that does not halt is

```
(((S(S((SS)S)))(SS))S),
```

much smaller than

```
(((S((SK)S))((SK)S))((S((SK)S))((SK)S))),
```

the encoding of the smallest looping λ-calculus term $((\lambda x.(xx))(\lambda x.(xx)))$.

3 Convergence and computability of the partition function

We consider any invalid program x to run forever. In that case, $E(x) = \infty$ and the relative probability of x at any $\beta > 0$ is zero.

We work over the alphabet $\{(, S, K,)\}$. Given $n \geq 0$ left parentheses, there are C_n ways to create a full binary tree of applications, where C_n is the nth Catalan number. In a valid program there will be n left parentheses, $n + 1$ combinators, and n right parentheses, for a total of $3n + 1$ characters. There are 2^{n+1} ways to choose the combinators.

Therefore, the value of the partition function is bounded above by

$$\begin{aligned} Z(\beta, \gamma) &= \sum_{x} \exp(-\beta E(x) - \gamma V(x)) \\ \leq \; Z(0, \gamma) &= \sum_{x} \exp(-\gamma V(x)) \\ &= \sum_{n=0}^{\infty} \frac{C_n 2^{n+1}}{\exp(\gamma(3n+1))} \end{aligned}$$

which converges provided $\gamma \geq \ln(2)$ to

$$\frac{1}{2}\left(e^{2\gamma} - \sqrt{e^{\gamma}\left(e^{3\gamma} - 8\right)}\right).$$

However, $Z(0, \gamma)$ is not the limit of the partition function as β approaches zero from above. As β gets very small, the product $\beta E(n)$ is effectively zero for halting programs but still infinite for non-halting programs. The total contribution of a program x to the partition function in that case is $\exp(-\gamma V(x))$ for halting programs and zero for non-halting programs. The limit itself is the generalized halting probability of the calculus, which is asymptotically partially random [2].

Since there are an infinite number of programs that halt immediately, $Z(\beta, \gamma)$ diverges as γ approaches zero; the particular point at which it diverges depends on β.

When both β and γ are nonzero and computable, the value of the partition function is computable when it converges: to compute the value to a particular accuracy, we simply run long-running programs long enough that their contribution is too small to matter.

4 Stirling cycle

We arbitrarily choose 10^{-8} as our accuracy for the partition function and $\beta, \gamma \in \{2, 3\}$ as the four corners of the cycle in β, γ space. Given those parameters, we find that we only need to consider programs up to length 10:

$$Z(0,2) = \frac{1}{2}\left(e^{2\cdot 2} - \sqrt{e^{2}\left(e^{3\cdot 2} - 8\right)}\right) \approx 0.272$$

$$\begin{aligned} Z_{>10}(0,2) = \;& Z(0,2) \\ & -(1\cdot 2)e^{-1\cdot 2} \\ & -(1\cdot 4)e^{-4\cdot 2} \\ & -(2\cdot 8)e^{-7\cdot 2} \\ & -(5\cdot 16)e^{-10\cdot 2} \\ \approx \;& 2.32\times 10^{-9}. \end{aligned}$$

$$\frac{Z_{>10}(0,2)}{Z(0,2)} \approx 8.53\times 10^{-9} < 10^{-8}.$$

The approximation only gets better for $\beta > 0$.

All terms with one or two combinators halt immediately. Twelve of the terms with three combinators halt immediately, while the four of the form ((`K`x)y) take one step. Forty-eight terms with four combinators halt immediately. Twenty-eight terms halt after one step, those of the forms (((`SS`)x)y), (((`K`x)y)z), ((`K`(xy))z), and

((Kx)(yz)). Four terms halt after two steps, those of the form (((SK)x)y). Therefore,

$$\begin{aligned} Z(\beta,\gamma) = \quad & 2e^{-\beta\cdot 0-\gamma\cdot 1} \\ + \quad & 4e^{-\beta\cdot 0-\gamma\cdot 4} \\ + \quad & 12e^{-\beta\cdot 0-\gamma\cdot 7} \\ + \quad & 4e^{-\beta\cdot 1-\gamma\cdot 7} \\ + \quad & 48e^{-\beta\cdot 0-\gamma\cdot 10} \\ + \quad & 28e^{-\beta\cdot 1-\gamma\cdot 10} \\ + \quad & 4e^{-\beta\cdot 2-\gamma\cdot 10} \\ + \quad & Z_4(\beta,\gamma). \end{aligned}$$

The entropy of the distribution is

$$\begin{aligned} S(\beta,\gamma) &= -\sum_x p(x)\ln(p(x)) \\ &= -\frac{1}{Z(\beta,\gamma)}\sum_x e^{-\beta\cdot E(x)-\gamma\cdot V(x)}(-\beta\cdot E(x)-\gamma\cdot V(x)). \end{aligned}$$

The total work done is $-\oint T\,dS$. The temperature T is just $1/\beta$, and

$$dS = \frac{\partial S(\beta,\gamma)}{\partial\beta}d\beta + \frac{\partial S(\beta,\gamma)}{\partial\gamma}d\gamma.$$

The work done on each step is as follows.

1. The piston is placed in thermal contact with the cool reservoir. The coolness β increases from 2 to 3 while γ remains constant, so the work done is

$$-\int_2^3 \frac{1}{\beta}\frac{\partial S(\beta,2)}{\partial\beta}d\beta \approx -0.82311843.$$

2. The piston contracts, increasing the internal pressure and compressing the system. The coolness beta remains constant at 3, so the work done is

$$-\int_2^3 \frac{1}{3}\frac{\partial S(3,\gamma)}{\partial\gamma}d\gamma \approx -0.83693331.$$

3. The piston is placed in thermal contact with the hot reservoir. The coolness β increases from 2 to 3 while γ remains constant, so the work done is

$$\int_2^3 \frac{1}{\beta}\frac{\partial S(\beta,3)}{\partial\beta}d\beta \approx 1.21729648.$$

4. The piston expands, decreasing the internal pressure and compressing the system. The coolness beta remains constant at 2, so the work done is

$$\int_2^3 \frac{1}{2} \frac{\partial S\,(2,\gamma)}{\partial \gamma} d\gamma \approx 1.25540125.$$

The total work per cycle is approximately 0.81264599, or in other words, the average runtime of a program in the piston distribution increases by about 0.81 steps per cycle.

5 Conclusion

Maxwell's relations let us reason about the expected value of observables over large collections of programs in a market where exchanges of programs between collections are allowed subject to constraints on the observables; for example, an isometric step preserves the length of the piston program and the sum of the average runtimes of the piston and the reservoir programs. We computed the runtime value of an arbitrage cycle, moving programs between a "hot" and a "cold" reservoir. One can do similar computations analogous to any engine.

References

[1] J. C. Baez and M. Stay, Algorithmic Thermodynamics. *Computability of the Physical, Mathematical Structures in Computer Science* **22** (2012), 771–787.

[2] C. S. Calude and M. A. Stay, Natural halting probabilities, partial randomness, and zeta functions. *Inform. and Comput.* **204** (2006), 1718–1739.

[3] G. J. Chaitin, A Theory of Program Size Formally Identical to Information Theory. *J. Assoc. Comput. Mach.* **22** (1975), 329–340.

[4] M. Schönfinkel, Über die Bausteine der mathematischen Logik. *Mathematische Annalen* **92** (1924), 305–316.

[5] K. Tadaki, A generalization of Chaitin's halting probability Ω and halting self-similar sets, *Hokkaido Math. J.* **31** (2002), 219–253. Also available as arXiv:nlin.CD/0212001.

[6] K. Tadaki, A statistical mechanical interpretation of algorithmic information theory. Available as arXiv:0801.4194.

[7] K. Tadaki, A statistical mechanical interpretation of algorithmic information theory III: Composite systems and fixed points. *Proceedings of the 2009 IEEE Information Theory Workshop, Taormina, Sicily, Italy*, to appear. Also available as arXiv:0904.0973.