the data, the more they classify each sample incorrectly.

Russell, S and P Norvig (2009). *Artificial Intelligence: A Modern Approach*. 3rd ed. Pearson.

The Possibility of Spontaneous Generation of Self-Replicating Systems

Gary Prok

DOI: 10.33014/issn.2640-5652.1.2.prok.1

It has been glibly stated that given enough time, a universe of monkeys at typewriters could write Hamlet. If this is true then perhaps the universe of particles could accidently assemble the right configuration of stuff to spark life. Yet the universe today is not even old enough to allow all the resources in the visible universe to work together with the tiniest probability of success to randomly generate the first 200 characters of Hamlet. A shorter 142-character string of the start of Hamlet is a unique string of information which is of 10^{210} possible strings of 142 characters. 10^{210} random tries would give a chance of $\left(1-\frac{1}{e}\right)$, or ~63%, for generating this simple 142-character passage. This might be only for a fleeting instant at some corner of the universe, but nonetheless it would be spontaneously generated. Adding another 58 characters to total 200 characters reduces the chance to a minuscule value. Spontaneous generation of life it seems, would require even more resources or time.

Seth Lloyd has estimated the computational capacity of the universe to be 10^{120} operations from the start to date, on a register of 10^{90} bits (Lloyd, 2002). A register of 10^{90} bits contain nearly 10^{90} different 142-character length strings. So, the universe could generate at most 10^{210} tries to date. 10^{210} is approximately 2^{700}, or all of the possible permutations of 700 bits. Although this computational capacity cannot generate Hamlet in its entirety by chance, or even the first 200 characters, perhaps it can be shown to be sufficient to spontaneously generate life.

Another bound was offered by Dembski as 10^{150}, or approximately all the possible permutations of 500 bits (Dembski, 1998, pg. 213). This is more restrictive, but may be a more realistic estimate for practical things like the probability of spontaneous life.

One way of quantifying the needed time and resources is to consider a simple proxy for life. A proxy for abiogenesis requires a simple system that allows a means for an information string to induce its own replication as well as a means to execute this replication. This proxy could be a simple system of things that operate on such things to make more such things. Lambda calculus is a system of mathematical logic reduced to a minimal set of symbols and rules. Expressions in lambda calculus can operate on other such expressions to generate more such expressions. Furthermore, lambda calculus is a universal model of computation and is Turing complete. It is enlightening to see how much information is needed to describe a simple self-replicating set of statements in lambda calculus. If a universe of monkeys or of particles can generate this simple set of statements, then perhaps it can do the same with material stuff and spark life.

Lambda calculus is considered to be a minimally simple programming language; it is hard to conceive of a simpler language. It was created by Alonzo Church as part of his research into the foundation of mathematics and uses three simple rules, yet it is capable of expressing any computation. Its simplicity results in bloated expressions for simple concepts, like numbers, or "true" and "false". Its simplicity, however, also makes it a model for a system most likely to spontaneously generate.

The fundamental concept allowed by lambda calculus is substitution. (λx.fx) is an expression in lambda calculus where "λ" indicates that the next variable is a free variable to be substituted, "x" is the name of the free variable, and "." indicates that what follows is an expression or function in which the free variable is to be substituted. The value or expression to be substituted for x is placed to the right of the expression. As an example, (λx.x^2) y = y^2, where y is substituted for x. This substitution is called *abstraction*. Lambda calculus also allows for *application*, which is a function operating on an argument.

A tutorial of lambda calculus is not included here, but is readily found elsewhere.

Lambda calculus can be efficiently encoded in binary; binary lambda calculus (BLC) is an efficient way to encode expressions in lambda calculus into binary (Tromp, 2007). BLC uses De Bruijn indices instead of variables, where the index value is a natural number indicating how many λs back the variable refers. $\lambda x.\lambda y.xy$ is written as λ λ 2 1 with De Bruijn indices (Bruijn, 1972). Three things that

then need to be encoded into binary are abstraction (λ), application, and natural numbers. The scheme is very simple:

Abstraction (λ)	is encoded as 00
Application	is encoded as 01
Natural number n	is encoded as 111110... (where there are n 1s)

$\lambda x.\lambda y.xy$ is written as $\lambda\ \lambda\ 2\ 1$ with De Bruijn indices, which in BLC is 00 00 01 110 10 (spaces added for understanding).

Lambda expressions are executed by the process of beta-reduction. This means of execution can also be expressed in lambda calculus and encoded in BLC. Lambda calculus can express a list using the PAIR function. Other lambda expressions can extract any particular element of a list.

The following elements comprise a self-replicating system in lambda calculus:

- The self-replicating combinator Q; QQ -> QQ does not propagate on beta-reduction, it merely replaces itself. However, QE ((QE) nil) does propagate QE on beta-reduction, by creating a growing list of (QE), (QE), (QE).... (QE), nil (E is defined below). QE will propagate any expression, P; QE (P nil) will generate a growing list of P, P, P,.... P .
- A beta-reduction engine is required.
  - If this engine were encoded into P, then a means to translate from the information space to machine space also would be required and the initial instance would have no machine with which to execute.
  - The least required total Kolmogorov complexity would have the beta reduction engine simultaneously generated in the machine state, as opposed to being generated in the information media state and subsequently transcripted to the machine state. The engine would act like a single instance of a re-usable catalyst, operating on the growing list.

A minimal self-replication system in lambda calculus would generate a growing list of copied expressions, like (QE), (QE), (QE).... (QE), nil. This would require:

- A means to encode Lambda calculus. BLC is an efficient means.
- The E combinator, encoded in an information media state, like DNA/RNA in life.
- The self-replicating combinator Q, encoded in the information media state.
- A program for self-reduction – encoded in the machine state, like an enzyme protein assemblage in life.

The total information requirement to generate the minimal proxy for abiogenesis is then:

- E, which operates on a list and extends it by one element (based on Larkin and Stocks, 2004).

  E = λx. (λp.(λl.xxp(PAIR pl))) ,

  which in BLC is
  0000001111111111011011000000001011101011010
  (42 bits long).
- Q, the self-application combinator.

  Q = λx. xx

  which in BLC is 00011010 (8 bits long).
- The application of Q to E, QE, which adds 2 bits.
- Nil, which is $\lambda z.\lambda x.\lambda y.x$, adding 9 bits.
- Two more applications to create the expression QE ((QE) nil), adding 4 more bits
- A beta-reduction engine in machine space to process ((QE) E nil). This would at a minimum require:
  a. A true statement: $\lambda x.\lambda y.x$ (7 bits)
  b. A false statement: $\lambda x.\lambda y.y$ (6 bits)
  c. An if-then-else statement: $\lambda z.\lambda x.\lambda y.zxy$ (15 bits)
  d. Abstractions to delete and replace expressions (>~15bits each, or >~30bits).

  The bit count so far for a beta reduction engine is > 58. Actual construction of the beta reducer could require many more. A beta-reduction engine in an abstracted lambda calculus has been found, which is longer than 200 bits (Mogensen, 1994). An engine in BLC would require a similar number of bits.

So far we are up to many more than 123 bits, and likely more than 265 bits, for a randomly-formed pair of information strings, one in information space (DNA/RNA) and one in machine space (protein/enzyme) which allows the information string to propagate in a list.

It may be possible to compress these relatively short strings. E in particular appears to be a compressible string, which

might allow for a reduced bit count. However, adding a codec to the mix will certainly increase the bit count more than it offers in compressive bit-reduction.

It is reasonable to think that a minimal basic precursor to abiogenisis of life would require two random assemblages with total information exceeding 265 bits. However, 265 bits is below the 500 bit Dembski limit and below the 700 bit limit based on Seth Lloyd's analysis. This analysis does not rule out the spontaneous, if fleeting, generation of a self-replicating system somewhere in the universe at some time in its past. The probability given the Dembski limit is $\left(1 - \frac{1}{e^{(500/265)}}\right)$, or ~85%. The probability given the 700 bit limit is $\left(1 - \frac{1}{e^{(700/265)}}\right)$, or ~93%.

Appending other features on to this analysis will require more bits. Examples are:

- A means for the information describing the beta-reduction engine in the machine space to be also expressed in the information space.
- A means for the information expressed in the information space to be realized in the machine space, they way a genotype becomes a phenotype in life.
- A means to operate on more than the first element in the list to allow population growth that approaches exponential. This will require information to describe recursion on the TAIL function.
- A means to store the information and the beta-reduction engine in the physical world.

Although these are needed and may well push the information needs above 700 bits, they are beyond the scope of this analysis.

This analysis does not rule out a single instance of the spontaneous generation of a self-replicating system somewhere in the universe. It does, however, suggest that this is an extraordinarily rare event. This analysis suggests that there is less than 50% likelihood that 10 self-replicating systems have spontaneously generated in the universe. Given the vastness of the spacetime of the universe, an implication is that any evidence of extra-terrestrial life would be evidence of transpermia.

Bruijn, N G de (1972). "Lambda calculus notation with nameless dummies, a tool for automatic formula manipulation, with application to the Church-Rosser theorem". In: *Indagationes Mathematicae* 75 (5), pp. 381–392. URL: `http://alexandria.tue.nl/repository/freearticles/597619.pdf`.

Dembski, W A (1998). *The Design Inference: Eliminating Chance Through Small Probabilities*. Cambridge: Cambridge University Press.

Larkin, J and P Stocks (2004). "Self-Replicating Expressions in the Lambda Calculus". In: *Proceedings of the Twenty-Seventh Australasian Computer Science Conference (ACSC2004)*. Ed. by V Estivill-Castro. Dunein, New Zealand, pp. 167–173.

Lloyd, S (2002). "The Computational Universe". In: *Edge.org*. URL: `https://www.edge.org/conversation/seth%5C_lloyd-the-computational-universe`.

Mogensen, T (1994). "Efficient Self-Interpretation in Lambda Calculus". In: *Journal of Functional Programming* 2.3.

Tromp, J (2007). "Binary Lambda Calculus and Combinatory Logic". In: *Randomness And Complexity, from Leibniz To Chaitin*. Ed. by C S Calude. World Scientific Publishing Company, pp. 237–260.