Article

Quines are the fittest programs

Nesting algorithmic probability converges to constructors

Aritra Sarkar [1]* 0000-0002-3026-6892, Zaid Al-Ars [1] 0000-0001-7670-8572 and Koen Bertels [2] 0000-0001-9310-4885

[1] Department of Quantum & Computer Engineering,
Faculty of Electrical Engineering, Mathematics and Computer Science, Delft University of Technology,
Delft, The Netherlands
[2] Department of Informatics Engineering,
Faculty of Engineering, University of Porto, Porto, Portugal
* Correspondence: a.sarkar-3@tudelft.nl

Abstract: In this article we explore the limiting behavior of the universal prior distribution obtained when applied over multiple meta-level hierarchy of programs and output data of a computational automata model. We were motivated to alleviate the effect of Solomonoff's assumption that all computable functions or hypotheses of the same length are equally likely, by weighing each program in turn by the algorithmic probability of their description number encoding. In the limiting case, the set of all possible program strings of a fixed-length converges to a distribution of self-replicating quines and quine-relays - having the structure of a constructor. We discuss how experimental algorithmic information theory provides insights towards understanding the fundamental metrics proposed in this work and reflect on the significance of these result in digital physics and the constructor theory of life.

Keywords: algorithmic probability; universal constructors; self-replication; universal Turing machines; algorithmic information theory; deterministic finite automaton

---

1. Introduction

A dichotomy exist in algorithmic information theory, where, the length of the data string is considered a false indication of its complexity, while it does use the length of programs to calculate the likelihood of the occurrence of the data string. While short programs are considered more likely to produce a data string, the theory does not consider how likely it is for short programs to be generated by another higher level program.

We consider a fixed length model, where the data, the program, and all higher level programs have the same length, and are input programs or outputs of the same universal Turing machine. We explore the properties of the final distribution of the data string provided we consider multiple levels of meta-programs. The distribution converges to self-replicating programs or quines, which survive over generations of program-data hierarchy. This observed result has implication in the context of the biological cell being a physical manifestation of a universal constructor.

In Section 2 we provide the background of Solomonoff's algorithmic probability and the coding theorem method used to estimate it. Thereafter, in Section 3 we derive the distribution obtained by nesting algorithmic probability over multiple levels of a computational automata model, and the idea of an universal constructor in Section 4. In Section 5 the limiting behavior of this distribution is analyzed in depth. We validate our theory by extensively considering a specific case in Section 6 as an experiment. We conclude this article in Section 7 with implications of our results and possibilities of future research.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0584.v1

2. Algorithmic probability

The universal Solomonoff algorithmic probability [1] of a program p on a (prefix-free) universal Turing machine (UTM) U for an output x is proportional to the sum of inverse of the description lengths of all programs that generate the output.

$$\xi(x) = \sum_{p:U(p)\to x} 2^{-|p|} \tag{1}$$

This naturally formalizes Occam's razor (law of parsimony) and Epicurus's principle of multiple explanations by assigning larger prior credence to theories that require a shorter algorithmic description while considering all valid hypotheses.

Consider a Turing machine with n symbols and m states, which can be enumerated by running a finite number of programs. The algorithmic probability of a string x can be approximated as:

$$\xi(x) \approx D_{n,m}(x) = \frac{|U \text{ halts with output } x|}{|U \text{ halts}|} \tag{2}$$

i.e. counting how many programs produce the given output divided by the total number of programs that halt. This approximation is called the Coding Theorem Method (CTM) [2].

Approximating $\xi(x)$ using CTM, although theoretically computable, is extremely expensive in terms of computation time. The space of possible Turing machines may span thousands of billions of instances. Moreover, being uncomputable, every Turing machine in-principle needs to be run forever to know if it will halt. However, for small Turing machines, it is possible to judiciously stop the computation and declare the enumeration as non-halting if it does not halt within a maximum number of cycles (called the Busy Beaver runtime). However, even for Turing machines with a small number of states, the Busy Beaver runtime value is excessively large. For example, for only 5 states the the Busy Beaver runtime is still unknown as there are over 26 trillion cases to enumerate and from partial enumerations so far, we know the Busy Beaver runtime is ≥ 47176870. It is intractable to run so many machines for so many iterations, thus, often the CTM is estimated from a time limited by running the UTM for t cycles. This is based on [3] and hypothesizes an exponential decay [4] in the number of halting UTM with run duration.

We do not consider a special halt state, thus allowing us to explore the complete state space of programs [5]. This automatically includes programs with a halt state by encoding them as states that loop on themselves, moves the tape head arbitrarily and writes back the read character. This is further motivated by naturally occurring computing hardwares like DNA or the brain, having a fixed resource for storing the program, e.g. the number of base-pairs or the number of neurons. We are interested in the distribution of the computing output generated by a set of fixed size programs. Thus the related concept of Kolmogorov complexity does not have much significance, considering the length is fixed.

Let the description number of n symbols and m states be encoded as binary strings of length l. Thus, all 2^l possible programs have length $|p| = l$, and when run for t time steps (of course preferably larger than the Busy Beaver number), produces the following approximation of algorithmic probability:

$$D^{tl}_{n,m}(x) = \frac{|U^t(p) \to x|}{2^l} \tag{3}$$

We reach the same result plugging in the constant size of programs in the original equation of $\xi(x)$:

$$\xi(x) \approx D^{tl}_{n,m}(x) = \sum_{p:U(p)\to x} 2^{-l} = 2^{-l}|U(p) \to x| \tag{4}$$

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0584.v1

3. Nesting algorithmic probability

Let's denote the metric derived in the last section as:

$$\xi^{01}(x_i^0) = 2^{-l}|U(x_j^1) \to x_i^0| \tag{5}$$

ξ^{01} denotes the algorithmic probability of the set of output strings x^0 and is based on considering a uniform distribution of the set of programs x^1. This scalable notation considers fixed length programs and output data, no inputs and a specific Turing machine U. Thus, the set cardinalities $|x^1| = |x^0| = 2^l$ and individual lengths of the strings $|x_j^1| = |x_i^0| = l$.

In general, this is a many-to-one mapping, thus not all strings in x^0 are generated by running programs of the same size. The strings which are algorithmically random are not part of the set x^0 and are shown as the striped annulus in Figure 1.

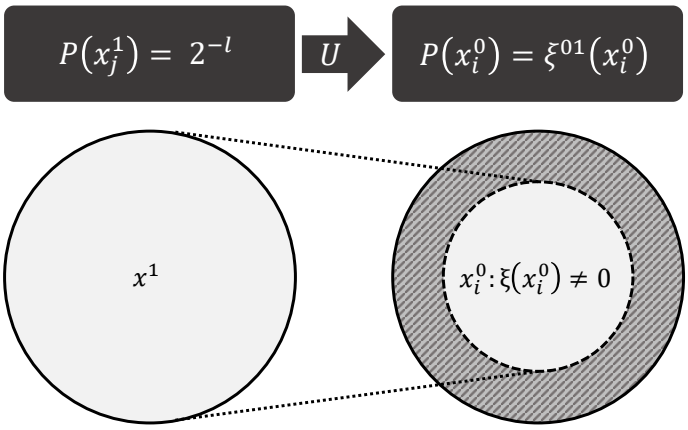

Figure 1. The space of programs typically map to a smaller set of output strings. The algorithm probability $P(x_i^0) = \xi^{01}(x_i^0)$ of the output strings x_i^0 is based on a uniform probability $P(x_j^1) = 2^{-l}$ of the programs x_j^1.

The set of algorithmic probabilities ξ^{01} for all x is called the Solomonoff's universal (prior probability) distribution. The core motivation of algorithmic information theory to define the universal distribution from a uniform distribution is based on the idea that simpler shorter theories are more probable. Our fixed-length program (model/hypothesis) formulation seem to not allow shorter programs to have more weight. Yet, there will be more programs nevertheless to generate a simple data, e.g. 0101010101010101 can be either generated by looping 01 for 8 times, or 0101 for 4 times or 01010101 twice. Whichever is the most efficient of these 3 programs would print the output and reach a halting state early. What matters to us is the frequency of programs (as in the CTM), not at what stage in the computation of our t steps it reached a stable attractor state. So even within same length programs we expect a non-uniform distribution as we do for the data.

Now we can pose the question: *why should all programs be considered equiprobably?* Considering a higher hierarchy of meta-programming, there is some physical process that generates that program on the program part of the tape of a UTM. Normally, for the standard definition of algorithmic probability, it is considered an unbiased coin flip, or the infinite programming monkey theorem. Since the universal distribution is not known apriori, there is no other preference than a uniform distribution to start with.

However, we can feedback the universal distribution on the programs and understand what effect it has on the data. Thus introducing a hierarchy of UTM levels, where the output data of one UTM

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0584.v1

is the input program for the next levels. Weighting the contribution of each program based on the probability that they themselves physically occur on the program part of the UTM tape, we get:

$$\xi^{02}_{U_0U_1}(x^0_i) = \sum_{x^1:U_0(x^1_j)\to x^0_i} 2^{-|x^1_j|}\xi^{12}_{U_1}(x^1_j) \tag{6}$$

Due to the invariance theorem, we can assume $U_1 = U_2 = U$, with only constant overheads of translating (cross-compiling) the program of one machine to another.

$$\xi^{02}(x^0_i) = \sum_{x^1:U(x^1_j)\to x^0_i} 2^{-|x^1_j|} \sum_{x^2:U(x^2_k)\to(x^1_j:U(x^1_j)\to x^0_i)} 2^{-|x^2_k|} \tag{7}$$

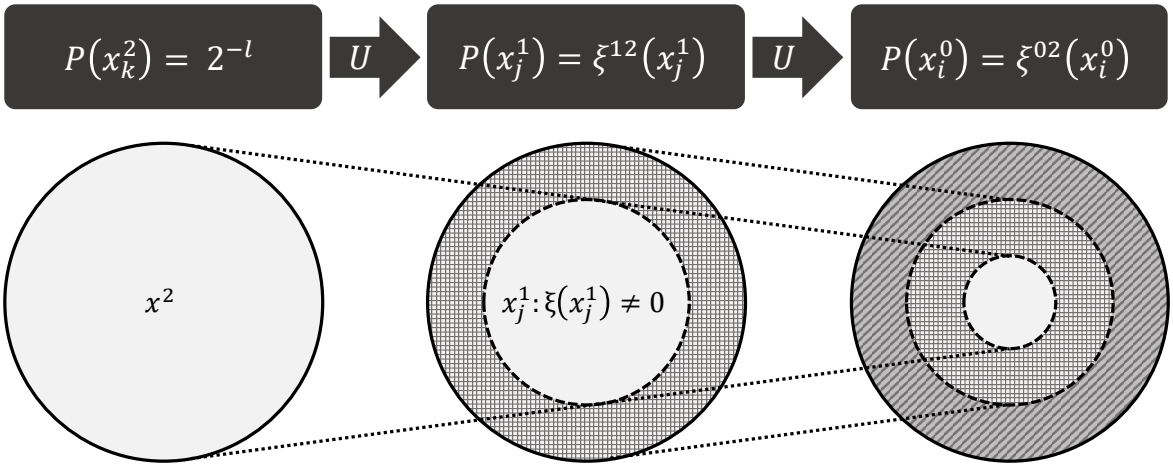

Figure 2. 2 level nesting of algorithmic probability

Considering the constant length $|x^2_k| = |x^1_j| = |x^0_i| = l$, the equation for $\xi^{02}(x^0_i)$ equals:

$$2^{-2l}|x^1 : U(x^1_j) \to x^0_i||x^2 : U(x^2_k) \to (x^1_j : U(x^1_j) \to x^0_i)| \tag{8}$$

Two levels of hierarchy are shown in Figure 2. Indeed it is possible to extend this argument to arbitrary many levels w, with no particular reason to choose ξ^{01} over ξ^{0w} for the expected distribution of the data occurring physically. We are interested in two properties as w approaches a large number. Let,

$$A^w = |x^w : U(x^w_k) \to (x^{w-1}_j : U(x^{w-1}_j) \to x^{w-2}_i)| \tag{9}$$

and,

$$B^w = |x^{w-1} : U(x^{w-1}_j) \to x^{w-2}_i)| \tag{10}$$

Since, $A \leq B$, for all w, what are the properties of the strings that 'survive' over these generations? And since, each lower hierarchy reduces the set size, will the inequality $A \leq B$ become an equality at some value of w or will the set be empty eventually. This is explored in Section 5.

4. Universal constructors

Universal constructor is a self-replicating machine foundational in automata theory, complex systems and artificial life. John von Neumann was motivated to study abstract machines which are complex enough such that they could grow or evolve like biological organisms. The simplest such machine, when executed, should at least replicate itself.

The design of a self-replicating machine consists of:

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0584.v1

- a program or description of itself
- a universal constructor mechanism that can read any description and construct the machine/description encoded in that description
- a universal copy machine that can make copies of any description (if this allows mutating the description it is possible to evolve to a higher complexity)

Additionally, the machine might have an overall operating system (which can be part of the world rule or compiler) and additional functions as payloads. This is shown in Figure 3. The payload can be very complex like a learning agent like AIXI [6] or an instance of an evolving neural network [7].

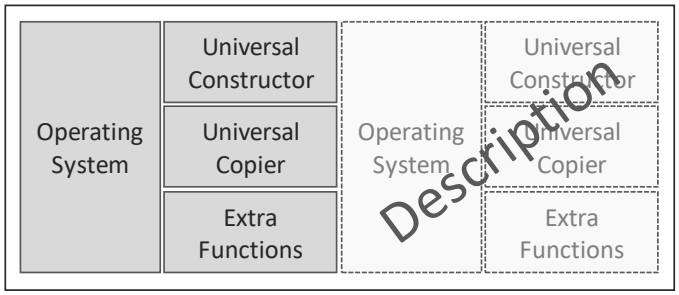

Figure 3. John von Neumann's system of self-replicating automata

The constructor mechanism has 2 steps: first the universal constructor is used to construct a new machine encoded in the description (thereby interpreting the description as program), then the universal copier is used to create a copy of that description in the new machine (thereby interpreting the description as data). This is analogous to the cellular processes of DNA translation and DNA replication, respectively. The cell's dynamics is the operating system which also performs the metabolism as the extra functions when it is not reproducing.

Von Neumann designed a universal constructor in a 2-dimensional cellular automata with 29 states. If we consider the Turing machine automata, this translates to printing out the description on the tape which can be executed as a program by another Turing machine. Note that the rest of the Turing machine mechanism like the tape head and movement are akin to the underlying cellular automata rules that automatically apply to the new cells where the replicated machine manifests.

The very design of the 3 parts of a constructor suggests that it cannot be algorithmically random as it should be possible to compress parts of its description. We are interested in the complexity and probability of constructors, which forms a subset of all possible configurations an automata can possess.

5. Fitness of quines

A quine is a program which takes no input and produces a copy of its own source code as its output. It may not have other useful outputs. In computability theory, such self-replicating (self-reproducing or self-copying) programs are fixed points of an execution environment, as a function transforming programs into their outputs. The quine concept can be extended to multiple levels of recursion, called ouroboros programs or quine-relays. Quines are also a limiting case of algorithmic randomness as their length is same as their output.

In our model, we consider the entire set of 2^l strings. Each string is represented by a node in Figure 4, with the arrows representing the mapping by running the string as a program on the UTM. Thus, while many-to-one arrows are possible, one-to-many is not. We partition the set of strings (interpreted as program or data) into 2 subsets: attractors and repellers. Attractors are strings which when executed as a program generate as output a string which is also from the attractor subset. While, a repeller string when run as a program can generate either an attractor or a repeller string. The entire space might have multiple connected components. Each connected component consists of an attractor

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0584.v1

basin, made of quines or quine-relays, as shown in the right side of the red line in Figure 4. Each node in the attractor basin might have a trail of repeller nodes, which, over cycles of algorithmic probability converge to the node on the basin.

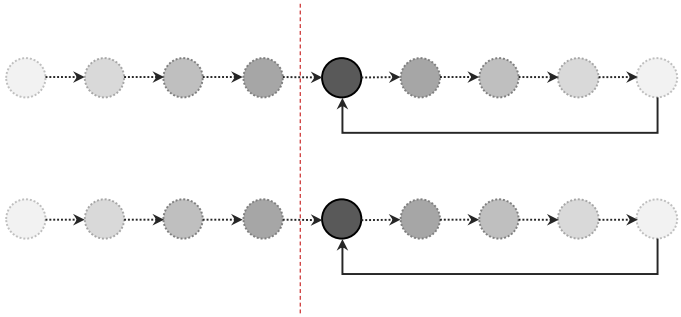

Figure 4. Attractor (on the right of the red line) and repeller strings (on the left of the red line)

Over multiple cycles the final set will only include the nodes on the right, with one-to-one mapping, thus conserving the number of strings in subsequent cycles. We denote this specific number of cycles with M, the meta-level at which a uniform distribution results in only attractors after M cycles. The number of attractors is given by

$$Q = |x^M : \xi^{0M}(x_i^M) \neq 0| \tag{11}$$

At this stage, each string has a one-to-one mapping. The algorithmic probability for these constructor strings depends on the number of paths leading to these attractor basis over these cycles. M is dependent on the number of considered state and symbols, the specification of the automata, the length of the programs and the time approximation for estimating the algorithmic probability at each level.

M being at least as (semi-)uncomputable as ξ, we can only study these fundamental metrics under reasonable approximations via Experimental Algorithmic Information Theory (EAIT) [8,9].

6. Experiments

Here, we consider a particular case to illustrate the developed formalism of nesting algorithmic probability. We take the 2 state 2 symbol Linear Bounded Automata (LBA) as it is both non-trivial as well as within the bounds of exhaustive enumeration. The details of this machine can be found in [5]. The program (description number) is encoded as the list of transition functions for each state and read symbol:

$$[QMW]^{Q_1R_1}[QMW]^{Q_1R_0}[QMW]^{Q_0R_1}[QMW]^{Q_0R_0} \tag{12}$$

This gives the values $q_\delta = 12$, as the length of the description number required to store a program for this machine. Thus, the space of this encoding allows $P = 2^{12} = 4096$ possible programs. The tape is also of length $c = z = 12$ and consists of all zeros with the tape head on the left most character: $\underline{0}000000000000$ The machine is run for $z = q_\delta = 12$ iteration. A Python script that emulates this restricted model of the Turing machine for all 4096 cases is available at [10].

At level 3 of nesting algorithm probability, we start with a uniform distribution of all programs, to produce the standard universal distribution. The mapping is shown in Figure 5 (the high resolution svg is available at [10]).

We observe that, at this level itself, the number of possible programs for the next generation gets reduced from 4096 to 21, with the following frequency of occurrence. The top 8 algorithmically probable attractor basins are shown in Figure 6.

```
P0      : 1886    P2048   : 1147    P4095   : 640     P3072   : 110
P1365   : 64      P2047   : 64      P2730   : 64      P1024   : 41
```

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0584.v1

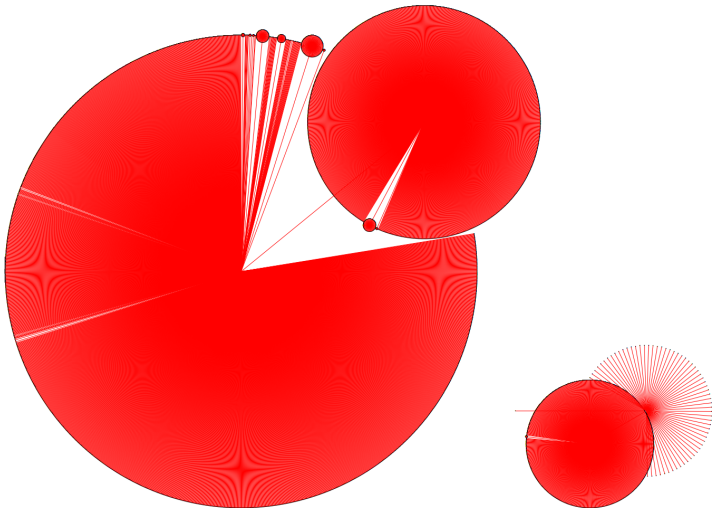

Figure 5. Level 3 of nesting algorithmic probability for a 2 state 2 symbol LBA (the high resolution svg is available at [10])

P3840	: 17	P128	: 11	P3968	: 11	P1344	: 10
P3584	: 10	P4032	: 10	P1792	: 2	P1920	: 2
P2560	: 2	P2688	: 2	P192	: 1	P1728	: 1
P2944	: 1						

Also, at this level itself we find that the machines P0 and P4095 are the only self-replicating programs, thus the limiting behavior can already be predicted.

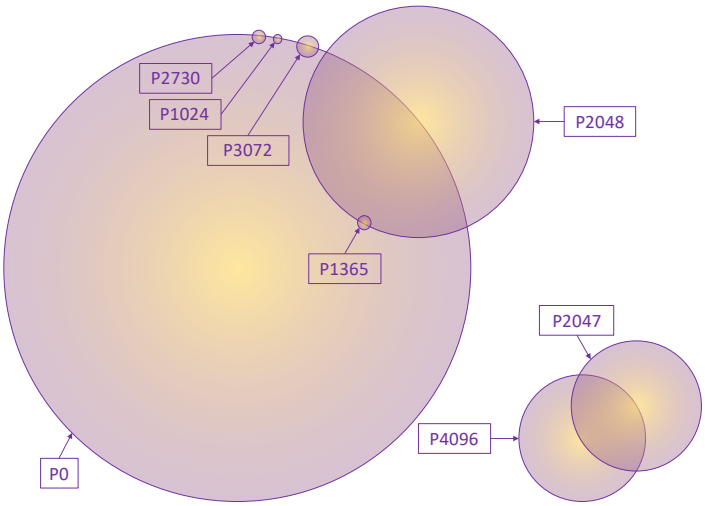

Figure 6. The 8 largest attractor basin of level 3 of nesting algorithmic probability

At level 2, these 21 programs get further mapped to just 3 programs, with the following frequency.

P0	: 16	P4095	: 3	P2048	: 2

This is shown in Figure 7.

At level 1, the 3 programs finally converge to the quines P0 and P4095 as shown in Figure 8. The frequency is as follows:

P0	: 2	P4095	: 1

At level 0, we reach the attractor states with a uniform one-to-one mapping.

Preprints (www.preprints.org) | NOT PEER-REVIEWED | Posted: 28 October 2020 doi:10.20944/preprints202010.0584.v1

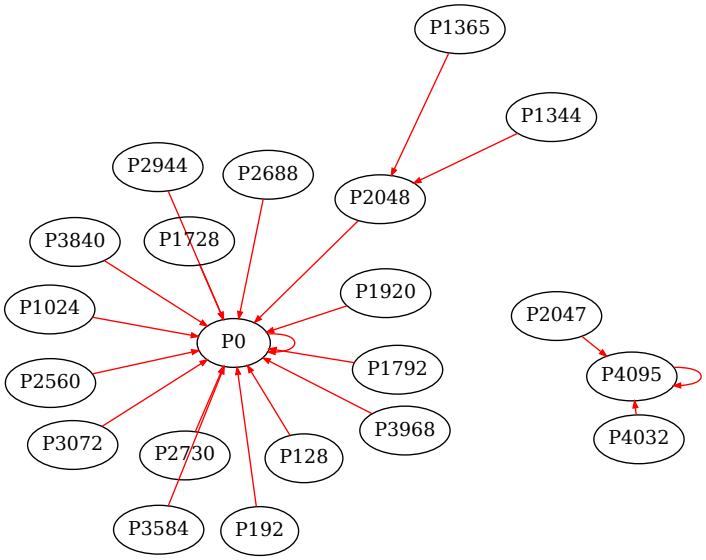

Figure 7. Level 2 of nesting algorithmic probability for a 2 state 2 symbol LBA

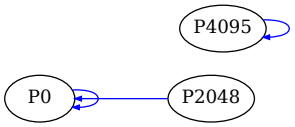

Figure 8. Level 1 of nesting algorithmic probability for a 2 state 2 symbol LBA

So, to calculate the overall algorithmic probability of these two fixed points, we calculate the cumulative frequency over these 4 levels. Thus, at level 2, the total frequency of P4095 consists of adding up the frequency of programs P4095, P2047, P4032 from the previous step, totaling to $640 + 64 + 10 = 714$. For P2048, we add up the frequency of Programs P1365, P1344, totaling to $64 + 10 = 74$. The rest of the 16 programs total to 3308 cases that reach P0. At level 1, P2048 also reaches the P0 attractor, giving the final algorithmic frequency of the quines as:

```
P0      : 3382    P4095    : 714
```

A similar experimentation on this space of 4096 programs is conducted as part of the Wolfram Physics Project exploring the rulial space of Turing machines [11].

The 2-2-1 LBA with 4096 programs does not show much diversity resulting in simple quines. It remains to be seen what the encodings of quines in larger spaces reveal about the structure and complexity of constructors. The 4 symbol 2 state variant has 2^{32} programs making it difficult to exhaustively enumerated for each program over at least 32 cycle (so that the write head's causal cone covers the full tape).

We need to look at alternative approaches (e.g. using quantum computation) to tame this overwhemingly large state space.

7. Conclusion

In this research we extend the idea of algorithmic probability to meta-programming under the assumption of fixed length. We present a mathematical formulation and derive the properties of

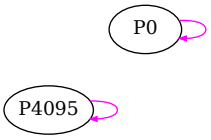

Figure 9. Level 0 of nesting algorithmic probability for a 2 state 2 symbol LBA

 doi:10.20944/preprints202010.0584.v1

this distribution analytically. The dynamics of the space of strings show some interesting properties: starting from a uniform distribution of programs, the set of programs converges to a distribution of constructors, and the set of constructors is very small. The number of cycles for this convergence, the space of attractor strings, and their behavior with respect to larger states and length are fundamental metrics in the space of programs [12], reminiscent of Chaitin's Omega number and Kleene's recursion theorems.

The idea of using the fixed-point, called the Y-combinator in lambda calculus $\lambda f.(\lambda x.f(xx))(\lambda x.f(xx))$ to describe the genetic code [13] is pioneered by Gregory Chaitin [14] as the field meta-biology. This is in line with the constructor theory [15] approach for understanding physical transformations in biology. In the field of transcendental/recreational programming, the DNA structure was used to code the Gödel number (similar to the description number) of any Ruby script [16]. In our future research, we intend to explore the significance of these results for artificial life applications [17] in synthetic biology. The space and probability of constructors [18] can inform the subset of DNA encoding for in vitro experimentation and understanding of causal mechanisms in cells [19,20].

From a cosmological perspective, this modifies John Wheeler's "It from Bit" foundations of digital physics and Seth Lloyd's "It from Qubit" quantum extension to an "It from It" principle, where the universe is modeled as a constructor instead of a (quantum) computer, and eventually ends in self-replication after each aeon.

Author Contributions: "Conceptualization, methodology, software, validation, formal analysis, investigation, visualization, writing–original draft preparation, A.S.; writing–review and editing, supervision, project administration, Z.A. and K.B. All authors have read and agreed to the published version of the manuscript."

Funding: "This research received no external funding"

Conflicts of Interest: "The authors declare no conflict of interest."

References

1. Solomonoff, R.J. A formal theory of inductive inference. Part I. *Information and control* **1964**, *7*, 1–22.
2. Levin, L.A. Laws of information conservation (nongrowth) and aspects of the foundation of probability theory. *Problemy Peredachi Informatsii* **1974**, *10*, 30–35.
3. Calude, C.S.; Stay, M.A. Most Programs Stop Quickly or Never Halt. *arXiv preprint cs/0610153* **2006**.
4. Soler-Toscano, F.; Zenil, H.; Delahaye, J.P.; Gauvrit, N. Calculating Kolmogorov complexity from the output frequency distributions of small Turing machines. *PloS one* **2014**, *9*.
5. Sarkar, A.; Al-Ars, Z.; Bertels, K. Quantum Accelerated Estimation of Algorithmic Information. *arXiv preprint arXiv:2006.00987* **2020**.
6. Hutter, M. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*; Springer Science & Business Media, 2004.
7. Stanley, K.O.; Clune, J.; Lehman, J.; Miikkulainen, R. Designing neural networks through neuroevolution. *Nature Machine Intelligence* **2019**, *1*, 24–35.
8. Delahaye, J.P.; Zenil, H. On the Kolmogorov-Chaitin Complexity for short sequences. *ArXiv* **2007**, *abs/0704.1043*.
9. Zenil, H. Experimental Algorithmic Information Theory | Complexity and Randomness. http://www.mathrix.org/experimentalAIT/, 2020.
10. Sarkar, A. Advanced-Research-Centre/QuBio. https://github.com/Advanced-Research-Centre/QuBio/tree/master/Project_01/classical, 2020.
11. Wolfram, S. Exploring Rulial Space: The Case of Turing Machines. https://writings.stephenwolfram.com/2020/06/exploring-rulial-space-the-case-of-turing-machines/, 2020.
12. Wolfram, S. *A new kind of science*; Vol. 5, Wolfram media Champaign, IL, 2002.
13. Chaitin, G.J. Meta math! the quest for omega. *arXiv preprint math/0404335* **2004**.
14. Chaitin, G. *Proving Darwin: making biology mathematical*; Vintage, 2012.

15. Marletto, C. Constructor theory of life. *Journal of The Royal Society Interface* **2015**, *12*, 20141226.
16. Endoh, Y. mame/doublehelix. https://github.com/mame/doublehelix, 2020.
17. Noireaux, V.; Maeda, Y.T.; Libchaber, A. Development of an artificial cell, from self-organization to computation and self-reproduction. *Proceedings of the National Academy of Sciences* **2011**, *108*, 3473–3480.
18. Sipper, M.; Reggia, J.A. Go forth and replicate. *Scientific American* **2001**, *285*, 34–43.
19. Brenner, S. Life's code script. *Nature* **2012**, *482*, 461–461.
20. Condon, A.; Kirchner, H.; Larivière, D.; Marshall, W.; Noireaux, V.; Tlusty, T.; Fourmentin, E. Will biologists become computer scientists? *EMBO reports* **2018**, *19*, e46628.