

On the Auto-Randomization of Knowledge

Stuart H. Rubin
Spawars Systems Center
stuart.rubin@navy.mil

Abstract

*In this paper, we first apply traditional computability theory to prove that the randomization problem, as defined herein, is recursively unsolvable. We then move on to extend traditional computability theory for the case of **k-limited** fine-grained parallel processors (i.e., temporal relativity). Using this modification, we are able to prove the **Semantic Randomization Theorem (SRT)**. This theorem states that the complexity of an arbitrary self-referential functional (i.e., implying representation and knowledge) is unbounded in the limit. Furthermore, it then follows from the **Unsolvability of the Randomization Problem** that effective knowledge acquisition in the large must be domain-specific and evolutionary. It is suggested that a generalized operant mechanics will be the fixed-point randomization of a domain-general self-referential randomization. In practice, this provides for the definition of knowledge-based systems that can formally apply analogy in the reasoning process as a consequence of semantic randomization.*

1. Introduction

This paper represents a theoretical attempt to define the nature of machine intelligence in the large. That is, given fine-grained parallel processors having finite hardware memory and sufficiently wide communication busses, how does one architecturally define machine intelligence? This question is pervasive because we all know what an intelligent system is or should be on the basis of empirical experience with formally trivial systems. However, that experience must be tempered by an abstract notion of what intelligence consists of; namely, the capability to learn and acquire knowledge through the use of metaphor. More formally, we say that intelligence is measured by the complexity of an effective program. However, surely one can specify complex programs that just do not learn. For example, consider a program for playing chess at the expert level. It turns out that such programs are formally defined to be trivial because they are not capable of self-reference. Here, self-reference takes the form of programs that can modify

themselves, or in this case, programs that can symbolically learn to play chess. Knowledge must be self-referential because exploratory search (in the “source code”) is necessarily delimited commensurate with learning (i.e., program-rewriting functions). It follows that complexity in the large differs from complexity in the small in that scale allows for learning, which allows for higher complexities in a given domain-specific task.

On Machine Learning

Machine learning in the small has emerged as a bona fide academic discipline. Nevertheless, a contention of this paper is that machine learning in the large is as different from that in the small as relativistic mechanics is from classical mechanics. Again, the key to machine learning in the large is self-referential transformation – as will be discussed next.

On Functional Transformation

A functional is a total map of functions to functions [2]. They represent the practical embodiment of self-referential transformation on a fine-grained architecture. Functionals are coherent. This means that they can effect domain-specific evolution – including that of themselves – in a networked architecture. It follows that a proper metric for machine intelligence is not so much based on external applications; but rather, on a bounded space-time auto-extrapolation of the domain-specific source code itself. This is a radical departure from current thinking in soft computing. Evolutionary symbolic computing (ESC) promises to transform the way we think about the limits, capabilities, and economics of machine intelligence.

2. Unsolvability of the randomization problem

The central thesis of this paper is that the greater the degree of randomization [1] provided by any logic or knowledge-based method, the greater the utility of the intelligent processes that can be constructed through its use. In particular, it will first be proven that there does not exist an effective method for minimizing the space-time

complexity of an arbitrary algorithm. The complexity of a series of digits is the number of bits that must be put into a computing machine in order to obtain the original series as output. As a consequence, all effective learning methods must be knowledge-based, or strong, if they are not to be trivial.

2.1. Proposition

Let, $f_0, f_1, \dots, f_j, \dots, j \in N$ be an effective enumeration in which every f_j is a total computable function from N to N . A total function has domain of definition N here. It is said to be computable just in case it is realized by some while-program [2]. It follows that there is a total computable function $f: N \rightarrow N$, which does not appear in this effective enumeration.

Proof: To say that $f_0, f_1, \dots, f_j, \dots, j \in N$ is an effective enumeration of total computable functions means that there is a total computable $g: N \rightarrow N$ such that $\varphi_{g(j)} = f_j$. Define $f: N \rightarrow N$ by diagonalizing over the functions $f_0, f_1, \dots, f_j, \dots$; that is, let $f(j) = f_j(j) + 1$. f is a total function, and is computable by the while-program:

```

Begin
  X2 := g(X1);
  X1 :=  $\Phi$ (X2, X1);
  X1 := succ (X1);
End;
  
```

(1)

But f cannot appear in the effective enumeration $f_0, f_1, \dots, f_j, \dots$, itself, since it would differ from itself on its own index. ■

2.2. Corollary

There can be no effective enumeration of all the total computable functions from N to N .

2.3. Theorem (unsolvability of the totality problem)

There is no algorithm, which when presented with the index j of an arbitrary computable function $\varphi_j: N \rightarrow N$, can decide whether φ_j is total or not.

Proof: Define the total function $total: N \rightarrow N$ by

$$total(j) = \begin{cases} 1, & \text{if } \varphi_j: N \rightarrow N \text{ is total;} \\ 0, & \text{otherwise.} \end{cases} \quad (2)$$

Assume, by way of contradiction, that $total$ is computable by some while-program. Then we may define a total computable function $g: N \rightarrow N$ by the scheme

$$\begin{aligned} g(0) &= \text{the smallest } j \text{ such that } total(j) = 1, \\ g(n+1) &= \text{the smallest } j \text{ greater than } g(n) \text{ such that } total(j) = 1. \end{aligned} \quad (3)$$

In fact, g is computed by the following while-program, if we can assume that $total$ is available as a subroutine. Given n as the initial data in X1, the program loads $n+1$ into X2, sets X1 to 0, and then computes until X1 holds the value $g(n)$.

```

Begin
  X2 := succ (X1);
  X1 := 0;
  While X2 <> 0 Do:
    Begin
      While total (X1) = 0 Do:
        X1 := succ (X1);
        X2 := pred (X2);
      End
    End;
  
```

(4)

Clearly, $\varphi_{g(0)}, \varphi_{g(1)}, \dots, \varphi_{g(j)}, \dots$ is an enumeration containing all and only total computable functions. But this contradicts Corollary B. Thus, $total$ cannot be a computable function. ■

2.4. Lemma

Consider the family of all computable functions $\varphi_j: N \rightarrow N$, $j \in N$. The function

$$algorithm(j) = \begin{cases} 1, & \text{if } \varphi_a \text{ randomizes } \varphi_j; \\ 0, & \text{otherwise.} \end{cases} \quad (5)$$

is not effectively computable.

Proof: First define the program

$$P_{g(j)} = \mathbf{Begin} \ \Phi(j, X1); X2 := i; \mathbf{End}; \quad (6)$$

where g is a total computable function by the s - m - n Theorem [2]. Thus,

$$\varphi_{g(j)}(j) = \begin{cases} i, & \text{if } \varphi_j(j) \downarrow; \\ \perp, & \text{otherwise.} \end{cases} \quad (7)$$

Observe that $\varphi_{g(j)}$ is the randomization function φ_a , just in case φ_j is a total function:

$$\varphi_{g(i)} = \varphi_a \Leftrightarrow \varphi_j \text{ is total.} \quad (8)$$

Consider next the composition of the function *algorithm*, defined in equation (5), and *g*:

$$a \lg orithm(g(j)) = \begin{cases} 1, & \text{if } \varphi_{g(j)} = \varphi_a; \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

By our last observation,

$$a \lg orithm(g(j)) = \begin{cases} 1, & \text{if } \varphi_j \text{ is total;} \\ 0, & \text{otherwise.} \end{cases} \quad (10)$$

This means that the function *total*, defined in Theorem C [2], is none other than *algorithm* \circ *g*. We have thus reduced the computability of *total* to that of *algorithm*. Since we found that *total* is not computable, we conclude that *algorithm* cannot be either. ■

2.5. Theorem (unsolvability of the randomization problem)

There is no algorithm, which when presented with indices *i* and *j* of arbitrary computable functions $\varphi_i: N \rightarrow N$ and $\varphi_j: N \rightarrow N$ can decide whether φ_i is a randomization of φ_j . Thus, there is no algorithm, which when presented with the index *j* of an arbitrary computable function $\varphi_j: N \rightarrow N$, $j \in N$, can randomize that function (i.e., transform it into φ_i , where *i* indexes an arbitrary randomized, or random, function).

Proof: Define the total function

$$random(i, j) = \begin{cases} 1, & \text{if } \varphi_i \text{ randomizes } \varphi_j; \\ 0, & \text{otherwise.} \end{cases} \quad (11)$$

The randomization function φ_a is computable by definition. This function, in its simplest form, removes all redundant **Begin End**; pairs. Let *i* be an index for it. It should be clear that we can now write $a \lg orithm(j) = random(i, j)$ for all $j \in N$, where $a \lg orithm: N \rightarrow N$ was defined in equation (5). We have thus reduced the computability of *algorithm* to that of *random*. Since *algorithm* is not computable, *random* cannot be either. ■

3. Unsolvability of the semantic randomization problem

We define knowledge to be a relative term, which refers to any semantic randomization of information (i.e., data). The need for transformational representations of knowledge was first demonstrated by Amarel [3]. He proved that the selection and transformation of domain-specific representations is a fundamental component of the problem-solving process.

For example, in the classic board game of Tic-Tac-Toe, one might learn that three Xs in the first or third row are winning configurations. Given an appropriate representation (and interpreter), as follows, this leads to

$$a_{1,1}, a_{1,2}, a_{1,3} \rightarrow win$$

$$a_{3,1}, a_{3,2}, a_{3,3} \rightarrow win$$

an effective randomization. That is,

$$a_{*,1}, a_{*,2}, a_{*,3} \rightarrow win$$

Notice that this schema can be instantiated for a proper prediction of a win in the second row. If this prediction were to be in error, then the correction would be acquired, integrated, and randomized. Questions are generated in an attempt to relax the current representation in favor of ever-greater randomization. Similarly, it has been shown that knowledge acquisition and processing must be flexible to reduce the costs incurred in the construction of intelligent systems [4].

Associative Memories (AMs) store associations representing the relationships of items in a particular context. An AM functions differently than a relational database. A relational database can be used to answer such questions such as, “How many categories of balloons are lighter than air?” However, AMs can be used to extract more advanced semantic relationships. For example, in contrast with a relational database, an AM can be used to answer questions such as, “What types of balloons are lighter than air?” and “What might cause a balloon to rise?” KASER-based AMs [5] could extend the semantic relationships to include such analogous questions as, “What types of objects float?” and “What might cause hot air to rise?”

A Semantic Associative Memory (SAM) can be formalized to define computing with words [5] – [7]. Semantic recall is by definition a randomization operation. For example, the semantic recall of an object that is supported above the ground and upon which one can sit is *functionally* a “chair” (e.g., a large smooth rock).

More formally, we proceed to show that the time-delimited self-referential application of a SAM is a semantic randomization operation whose image cannot be bounded in complexity. This opens the door for the

definition of an effective mechanics. It can be tested against such application domains as chess, battle management, and other areas of C4ISR where computational creativity lies at the focus. The non-specialist is referred to references [2] and [8] for further readings in computability theory.

3.1. Definition (associative memory)

There are three types of memory reference mechanisms listed in order of increasing generality:

1. Address-based (Relative/Absolute): where a specifies a (relocatable) memory address.
2. Content-based (CAM): where, A specifies a list of attributes, $f(A)$ is used to compute a , and f is a finite *total computable* (concurrent) function.
3. Semantic-based (SAM): where $F(A)$ is used to compute a and F is a countably infinite indexed family of *total computable* (concurrent) functions.

3.2. Remark

Notice that A serves as an effective data structure, while f serves as an effective (functional) program. In practice, this implies that an effective mechanics is necessarily underpinned by the Lambda calculus (e.g., LISP). SAMs are realizable by a countably infinite number of programs defined by sequences of (concurrent) instructions: $i_0, i_1, \dots, i_j, \dots, j \in N$ having index in N .

3.3. Definition (random)

A series of numbers is said to be *random* if the smallest algorithm capable of specifying it to a computer has about the same number of bits of information as the series itself. A random series of digits is one whose complexity is approximately equal to its size in bits. Since complexity has been defined as a measure of randomness, no number can be effectively proved to be random unless the complexity of the number is less than that of the system itself [1].

3.4. Remark

It follows from the fact that all minimal programs are random that a system of greater complexity is required in order to prove that a program is a minimal one for a particular series of digits [1]. This further implies that every operant mechanics must incorporate the fixed-point element of chance to complete its definition.

In the proof of the *Unsolvability of the Randomization Problem* in the previous section, we showed that no effective method can provably minimize the space-time complexity of an arbitrary algorithm on an absolute scale.

It follows that the set of non-trivial (i.e., self-referential) effective randomizations are *recursively enumerable*, but not *recursive*. A consequence of this theorem is that all embodied representations and algorithms must be evolvable in every degree of freedom in the general case.

3.5. Theorem (semantic randomization problem)

Given a pair of self-referential fixed-point randomizations having arbitrary indices, $u, v \in N$, $u \neq v$, then by Church's Thesis [2] there exists a third program having index, $l \in N$, such that

$$\varphi_l = \varphi_u \parallel \varphi_v \quad (12)$$

We will show:

$\exists k \mid \forall c \geq k > 0, \varphi_r = \varphi_l^{t+c}(l) = \varphi_l^{t+k}(l), \mid \varphi_r \mid < \mid (\varphi_u \parallel \varphi_v) \mid$, where $k, r \in N$ (i.e., the semantic inequality for concatenation). While traditional computability theory addresses computational steps, here we will address computational time in lieu to allow for concurrent processing without recourse to dovetailing [2]. The proof will show that SAMs can be unbounded in their complexity in theory concomitant with scale. Furthermore, it then follows from the fact $\aleph_0^n = \aleph_0$ [8], where the concatenation operator in equation (12) implies $n = 2$, that any effective mechanics must necessarily address an arbitrarily large set of mutual (concurrent) functional randomizations. It may be useful to conceptualize such a mechanics as a discrete information-theoretic model of gravity operating over time.

Proof: Define the total function(s) $randomize(d) : N \rightarrow N$ by

$$randomize(i) = \begin{cases} randomized(i), & \text{if } \varphi_i \text{ is randomized;} \\ i, & \text{otherwise.} \end{cases} \quad (13)$$

$randomize$ is then total and computable by Church's Thesis. Let, i be an index for it. Then, we may write (14):

$$\forall c \mid c \geq k > 0, \varphi_i^{t+c}(i) = \begin{cases} \varphi_j^{t+k}, & \text{if } \varphi_i^{t+k}(i) \downarrow \text{ where } \mid \varphi_j \mid < \mid \varphi_i \mid; \\ \varphi_i, & \text{otherwise.} \end{cases}$$

3.6. Lemma

The concatenation operator may be defined to insert all assignment statements (i.e., $X := X + p$; from locally randomized programs u and v) at the beginning of program l . The function indexed by l may be computed by the following while-program, if we can assume that

$\Phi(i, m, n, p, Transformed)$ is available as a computable subroutine by Church's Thesis. The program may contain an arbitrary number of assignment statements followed by the subroutine call(s) (for programs u and v) to randomize their arithmetic content.

Program *randomize* (i):

```

Begin
  X := X + m;
  X := X + n;
  Transformed := True;
  While Transformed Do:
     $\Phi(i, m, n, p, Transformed)$ 

```

End;

Subroutine $\Phi(i, m, n, p, Transformed)$: (15)

($m, n, p \in N$)

Begin

Transformed := False;

Pattern: /* Pattern does not replace itself. */

X := X + m;

X := X + n;

Replacement:

($p := m + n$);

X := X + p;

(*Transformed* := True)

End;

It follows from while-program (15) that the $\Phi(i, m, n, p, Transformed)$ subroutine will randomize all simple assignment statements at the beginning of programs l, u , or v into a single one of the form, $X := X + p$; where, p is an arbitrarily large integer of arbitrary complexity.

A set A is said to be *denumerable* if $|A| = \aleph_0$ [8]. An *enumeration* of A is an isomorphism $f : N \rightarrow A$. The isomorphism $n \mapsto p$ shows that the enumeration of indices for the family of *randomize* programs (15) has cardinality \aleph_0 .

Next, consider the application of a *program-rewriting function*, h . This function takes a *syntactic* object (programs u and v) and modifies their syntax in a systematic way by inserting an arbitrary sequence of assignment statements (e.g., $X := X + 1$;) at the beginning of programs u and v . The function, h , is total and computable by Church's Thesis. The s - m - n Theorem [2] may now be applied to equation (12), where $h : N^2 \rightarrow N$ with the result,

$$\varphi_l = \varphi_{h(u,v)} = \varphi_u \parallel \varphi_v \quad (16)$$

Clearly,

$$\exists k \mid \forall c \geq k > 0, |\varphi_l^{t+c}(l)| = |\varphi_l^{t+k}(l)| < |\varphi_l| \quad (17)$$

since $X := X + m$; and $X := X + n$; in program l are replaced by $X := X + p$; at time, $t+k$, using program (15) in accordance with equation (14). Notice that program l treats itself as a *semantic* object here, which differentiates this operation from that provided for by the s - m - n Theorem. Equations (16) and (17) may then be combined for the desired result (18):

$$\exists k \mid \forall c \geq k > 0, \varphi_r = \varphi_l^{t+c}(l) = \varphi_l^{t+k}(l), |\varphi_r| < |\varphi_u \parallel \varphi_v|$$

This completes the proof of the Semantic Randomization Theorem (SRT). ■

It follows as a consequence of the SRT that the complexity of knowledge is unbounded in the limit. While practical intelligence is in every case constrained by the operant laws of space and time, every non-trivial (i.e., self-referential) finite realization of intelligence via randomization is necessarily domain specific and recursively enumerable (i.e., temporally relative) as a consequence of this theory. The methodology used to prove the SRT may be extended to define an effective mechanics in the next publication. Randomization would then be consistent with Occam's razor.

4. Conclusion

ESC represents a point of departure from symbolic computing. It can be realized through the use of fine-grained functional architectures. We have seen that every computational fixed point is dependent on the representational formalism [3]. The choice of formalism is domain-specific and thus evolutionary by nature. Semantic randomization defines a metric space, which serves to guide the evolutionary process and most importantly, in the absence of localized feedback. Symmetry is pervasive in nature. It provides a lever for the transformation of representation and knowledge alike.

5. Future work

It remains to define operant mechanics for the (self-referential) transformation of representation and knowledge alike. Clearly, all such mechanics must envelop domain-specific formalisms – including those that engender heuristic search and are of a transformative nature. It follows that a generalized operant mechanics will be the fixed-point randomization of a domain-general self-referential randomization. In summary, knowledge must beget knowledge.

On the practical side, the results of this paper provide a framework within which randomization theory may be applied to the design of contemporary knowledge-based systems. Here, the randomization of user input will provide for the formal use of natural language – including conceptual extrapolation for contextual expression. Clearly, a capability to compute with words provides a prescription for the design and subsequent construction of decision support systems of unheralded utility [6], [7].

6. Acknowledgments

The author thanks SPAWAR Systems Center (SSC), San Diego for their financial support of this research. This work was produced by a U.S. government employee as part of his official duties and is not subject to copyright. It is approved for public release with an unlimited distribution.

7. References

- [1] G.J. Chaitin, “Randomness and Mathematical Proof,” *Sci. Amer.*, vol. 232, no. 5, pp. 47-52, 1975.
- [2] A.J. Kfoury, R.N. Moll, and M.A. Arbib, *A Programming Approach to Computability*. New York, NY: Springer-Verlag Inc., 1982.
- [3] S. Amarel, “On Representations of Problems of Reasoning about Actions,” *Mach. Intelligence*, vol. 3, pp. 131-171, 1968.
- [4] Y. Dote and S.J. Ovaska, “Industrial applications of soft computing: a review,” *Proceedings of the IEEE*, vol. 89, no. 9, pp. 1243-1265, 2001.
- [5] S.H. Rubin, R.J. Rush Jr., J. Murthy, M.H. Smith, and Lj. Trajkovic, “KASER: A Qualitatively Fuzzy Object-Oriented Inference Engine,” *Proc. North American Fuzzy Information Proc. Soc.*, pp. 354-359, 2002.
- [6] S.H. Rubin, “Computing with Words,” *IEEE Trans. Syst. Man, Cybern.*, vol. 29, no. 4, pp. 518-524, 1999.
- [7] L.A. Zadeh, “From Computing with Numbers to Computing with Words – From Manipulation of Measurements to Manipulation of Perceptions,” *IEEE Trans. Ckt. and Systems*, vol. 45, no. 1, pp. 105-119, 1999.
- [8] M.A. Arbib, A.J. Kfoury, and R.N. Moll, *A Basis for Theoretical Computer Science*. New York, NY: Springer-Verlag Inc., 1981.

Dr. Stuart H. Rubin (M’88 - SM’00) is a senior scientist at the Space and Naval Warfare Systems Center (SSC) in San Diego, code 27304. He received a BS in business from the University of Rhode Island, Kingston, RI in 1975; an MS in industrial and systems engineering from Ohio University in Athens, OH in 1977, an MS in computer science from Rutgers University in Piscataway, NJ in 1980, and a Ph.D. in computer and information science from Lehigh University in Bethlehem, PA in 1988. In 2003, he was awarded the IEEE Systems, Man, and Cybernetics Society’s Outstanding Service Award.

He has authored over 140-refereed conference and journal papers as well as several patent applications on behalf of SSC, San Diego. His professional interests center about heuristic methodologies for multi-sensor fusion, associative memory, and knowledge discovery.

Dr. Rubin is a member of AFCEA, the American Association for the Advancement of Science, the New York Academy of Sciences, the North American Fuzzy Information Processing Society, and several other scientific societies. Dr. Rubin chairs the IEEE Systems, Man, and Cybernetics Technical Committee on knowledge acquisition in intelligent systems. Dr. Rubin is also an associate editor of the IEEE Transactions on Systems, Man, and Cybernetics, Part C as well as the International Journal of Modeling and Simulation and the Journal of Systemics, Cybernetics, and Informatics. Dr. Rubin is the founder and general chair of the IEEE International Conference on Information Reuse and Integration (IRI); and, has delivered several keynote lectures at international conferences. Dr. Rubin also currently serves on the IEEE Advisory Committee.