# Self-Replicating Machines Attempting to Solve the Unsolvable

Hiroki Sayama
Department of Bioengineering, Binghamton University
P.O. Box 6000, Binghamton, NY 13902-6000, USA
sayama@binghamton.edu

**Abstract**

This paper elucidates a close similarity between self-replication of von Neumann's universal constructors and circular computational processes of universal computers that appear in Turing's original proof of the undecidability of the halting problem. The result indicates a possibility of reinterpreting self-replicating machines as attempting to solve the undecidable halting problem in the context of construction.[1]

## Introduction

John von Neumann's theory of self-reproducing automata (von Neumann 1951; von Neumann 1966) is now regarded as one of the greatest theoretical achievements made in early stages of artificial life research (Marchal 1998; Sipper 1998; McMullin 2000). Before working on its implementation on cellular automata, von Neumann sketched an outline of his self-replicating machine that consists of the following parts (von Neumann 1951):

$A$: A universal constructor that constructs a product $X$ from an instruction tape $I(X)$ that describes how to construct $X$.

$B$: A tape copier that duplicates $I(X)$.

$C$: A controller that dominates $A$ and $B$ and does the following:

1. Give $I(X)$ to $A$ and let it construct $X$.
2. Pass $I(X)$ to $B$ and let it duplicate $I(X)$.
3. Attach one copy of $I(X)$ to $X$ and separate $X+I(X)$ from the rest.

The functions of these parts are symbolically written as

$$A+I(X) \to A+I(X)+X, \tag{1}$$
$$B+I(X) \to B+2I(X), \tag{2}$$
$$(A+B+C)+I(X) \to \{(A+B+C)+I(X)\}+\{X+I(X)\}. \tag{3}$$

[1]This is a revised and extended version of another preprint written by the same author (Sayama 2006).

Then self-replication can be achieved if one lets $X = D \equiv A+B+C$, i.e.,

$$D+I(D) \quad \to \quad \{D+I(D)\}+\{D+I(D)\}. \tag{4}$$

Alan Turing's preceding work on computationally universal machines (Turing 1936) gave a hint for von Neumann to develop these formulations of self-replicating machines, especially on the idea of universal constructor $A$. These two kinds of machines share the same concept that a universal machine, given an appropriate finite description, can execute any arbitrary tasks specified in the description. The only difference is that one is about computation in an infinite tape, and the other is about construction in an infinite space. The former pioneered computation theory. The latter also pioneered a new, yet unnamed to date, field of study connecting logic and mathematics to biology and engineering. Here let us name this field *construction theory* for now.

While von Neumann's universal constructor holds a close correspondence to Turing's universal computer, however, little attention has been paid to what the entire self-replicating automaton $D$ in construction theory would parallel in computation theory. Besides $A$, the automaton $D$ also includes $B$ that duplicates a given tape and $C$ that attaches a copy of the duplicated tapes to the product of $A$. They are the subsystems that von Neumann added to the automaton in view of self-replication. Their counterparts are not present in the design of Turing machines, and therefore, the entire architecture of self-replicating machines has often been underestimated as if it were a heuristic design meaningful only on the construction side, but not on the computation side.

In this short paper, I will point out that self-replication in construction theory actually has a fundamental relationship with the halting problem in computation theory. Specifically, self-replication of von Neumann's universal constructors has its mathematical description in the identical form as that of circular computational processes of universal computers that appear in Turing's original proof of the undecidability of the halting problem. This leads us to a new interpretation of self-replicating machines as attemping to solve the undecidable halting problem, not in computation theory

but in the context of von Neumann's *construction theory*.

## The halting problem

The halting problem is a well-known decision problem that can be informally described as follows:

> *Given a description of a computer program and an initial input it receives, determine whether the program eventually finishes computation and halts on that input.*

This problem has been one of the most profound issues in computation theory since 1936 when Alan Turing proved that no general algorithm exists to solve this problem for any arbitrary programs and inputs (Turing 1936). The conclusion is often paraphrased that the halting problem is *undecidable*.

Turing's proof uses *reductio ad absurdum*. A well-known simplified version takes the following steps. First, assume that there is a general algorithm that can solve the halting problem for any program $p$ and input $i$. This means that there must be a Turing machine $H$ that always halts for any $p$ and $i$ and computes the function

$$h(p,i) \equiv \begin{cases} 1 & \text{if the program } p \text{ halts on the input } i, \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

Second, one can easily derive from this machine another Turing machine $H'$ whose behavior is modified to compute only diagonal components in the $p$-$i$ space, i.e.,

$$h'(p) \equiv h(p,p). \quad (6)$$

This machine determines whether the program $p$ halts when its self-description is given to it as an input. Such a self-reference would be meaningless for most of actual computer programs, but still theoretically possible.

Then, finally, one can tweak $H'$ slightly to make yet another machine $H^*$ that falls into an infinite loop if $h'(p) = 1$. What could happen if $H^*$ was supplied with its self-description $p(H^*)$? It eventually halts if $h'(p(H^*)) = h(p(H^*), p(H^*)) = 0$, i.e., if it does not halt on $p(H^*)$. Or, it loops forever if $h'(p(H^*)) = h(p(H^*), p(H^*)) = 1$, i.e., if it eventually halts on $p(H^*)$. Both lead to contradiction. Therefore, the assumption we initially made must be wrong—there must be no general algorithm to solve the halting problem.

## Turing's original proof

Here, I would like to bring up an informative, yet relatively untold, fact that Turing himself did not like to have such a tricky mathematical treatment as the above third step that introduces a factitious logical inversion into the mechanism of the machine, so he intentionally avoided using it in his original proof. What he actually did can be seen in the following extract from his original paper (Turing 1936, sec.8):

> *"... Now let K be the D.N[2] of H[3]. What does H do in the K-th section of its motion? It must test whether K is satisfactory[4], giving a verdict "s" or "u". Since K is the D.N of H and since H is circle-free, the verdict cannot be "u". On the other hand the verdict cannot be "s". For if it were, then in the K-th section of its motion H would be bound to compute the first $R(K-1)+1 = R(K)$[5] figures of the sequence computed by the machine with K as its D.N and to write down the $R(K)$-th as a figure of the sequence computed by H. The computation of the first $R(K)-1$ figures would be carried out all right, but the instructions for calculating the $R(K)$-th would amount to "calculate the first $R(K)$ figures computed by H and write down the $R(K)$-th". This $R(K)$-th figure would never be found. I.e., H is circular, contrary both to what we have found in the last paragraph and to the verdict "s". Thus both verdicts are impossible and we conclude that there can be no machine D*[6]."*
> (Footnotes added by the author)

In this paragraph Turing considered the *actual behavior* of intact $H$ (in his notation) on its self-description $K$, and noticed that what this machine would need to compute is exactly the same situation as the machine itself is in: "*H is looking at its self-description K.*" Such a self-reference would result in a circular process that never comes back. Therefore, $H$ cannot make any decision on whether $K$ is satisfactory or not. This contradiction negatively proves the possibility of $D$, or a general computational procedure to determine whether a machine stops writing symbols or not.

## Self-replication emerging

Turing's argument described above gives essentially the same argument as to what could happen if $H'$ (in our notation) received its self-description $p(H')$. In this case $H'$ must compute the value of $h'(p(H')) = h(p(H'), p(H'))$, and hence it would need to compute the behavior of the machine described in $p(H')$ on the input $p(H')$, exactly the same circular situation as that appeared in Turing's proof. Let us use

[2] Description Number: An integer that describes the specifics of a given computational machine.

[3] Note that this is different from $H$ we used in the previous section. Turing used $H$ for a machine that incrementally and indefinitely computes the diagonal sequence of the infinite matrix made of all computable sequences enumerated in the order of D.N's of corresponding machines.

[4] An integer $N$ is considered satisfactory if the machine whose D.N is $N$ can keep writing symbols indefinitely without falling into a deadlock (Turing called this property *circle-free*).

[5] $R(N)$ denotes how many machines are circle-free within the range of D.N's up to $N$.

[6] A machine that is assumed capable of determining whether a given machine is circular or not. This machine is introduced to construct $H$.

this example in what follows, as it is much simpler to understand than Turing's original settings.

What kind of computational task would $H'$ be carrying out in this circular situation? It tries to compute the behavior of $H' + p(H')$, which tries to compute the behavior of $H' + p(H')$, which tries to compute the behavior of $H' + p(H')$, ... Interestingly, this chain of self-computation takes place in the form identical to that of self-replication in von Neumann's construction theory, if "to compute the behavior" is read as "to construct the structure". This similarity may be better understood by noting that the role of $C$ that attaches $I(X)$ to $X$ shown in (3) parallels the role of diagonalization shown in (6); both attempt to apply a copy of the description to the machine represented by the description. Moreover, if one watched how the actual configuration of the tape of $H'$ changes during such a self-computing process, he would see that the information about $H'$ *actually self-replicates* on the tape space (with its representation becoming more and more indirect as generation changes though). Turing might have imagined this kind of replicating dynamics of machines when he developed his argument.

In view of the similarity between the above two processes, it can be clearly recognized that von Neumann's design of self-replicating machines is by no means just an anomaly in construction theory. Rather, it correctly reflects the self-computation chain of computationally universal machines that appears in the proof of the undecidability of the halting problem presented by Turing.

## Conclusion

With the similarity between computation and construction kept in mind, it may sound rather trivial that universal construction comes with undecidable problems similar to those for universal computation. For example, determining what a universal constructor eventually produces is equivalent to determining what a Turing machine eventually computes, and therefore it must be undecidable in general.

The above undecidability was already argued by Fred Cohen (Cohen 1987), where he showed that there is no general algorithm for the detection of self-replicating computer viruses. The proof is rather simple: If there were an algorithm, say *S*, that can determine whether a given computer program is self-replicative, then one could easily create another contradictory program that has *S* built in it and self-replicates if and only if its *S* classifies the program itself as non-self-replicative. This is probably the best acknowledged discussion on the relationship between self-replication and the undecidable problem so far.

We should note, however, that Cohen's argument above suggests that detecting a computer program that does *"X"* is generally impossible, where *"X"* could be self-replication but also be replaced by any other functions. Here self-replication is no more than just one of many possible behaviors of universal constructors.

In contrast, our argument discussed in this paper is more fundamental: Universal construction comes with undecidable problems *because of the existence of self-replication.* Here self-replication is not just an instance of many possible behaviors, but is actually the key property of the behavior of universal machines, either computational or constructional. As Turing showed in his proof, when a computational machine tries to solve the halting problem of its own computation process, it will fall into a cycle of self-computation that never ends in a finite time. Our point is that this corresponds exactly to the cycle of self-replication in construction theory, and that von Neumann's self-replicating automaton model rightly captures this feature in its formulation. The halting problem solver in construction theory lets the subject machine construct its product and see if it eventually stops. If it tries to solve the halting problem of its own construction process, it will start self-replication, and the entire process never completes in a finite amount of time.

The insight obtained above provides us with some new implications about the connections between life and computation. The relationship between parent and offspring is equivalent to the relationship between the *computing* $H'$ and the *computed* $H'$ in computation theory. The endless chain of self-replication that living systems are in, may be reinterpreted as a parallel to the endless chain of self-computation that a halting-problem solver falls in. In a sense, we may all be in the process initiated billions of years ago by a first universal constructor, who just tried to see the final product of its "diagonal" construction.

## References

Cohen, F. 1987. Computer viruses: Theory and experiments. *Computers & Security* 6:22–35.

Marchal, P. 1998. John von Neumann: The founding father of artificial life. *Artificial Life* 4:229–235.

McMullin, B. 2000. John von Neumann and the evolutionary growth of complexity: Looking backward, looking forward... *Artificial Life* 6:347–361.

Sayama, H. 2006. On self-replication and the halting problem. Submitted. Preprint available at http://arxiv.org/abs/nlin.AO/0603026 .

Sipper, M. 1998. Fifty years of research on self-replication: An overview. *Artificial Life* 4:237–257.

Turing, A. M. 1936. On computable numbers, with an application to the Entscheidungsproblem. *Proc. London Math. Soc. Ser. 2* 42:230–265. A correction followed in 1937. Fulltext available online at http://www.abelard.org/turpap2/tp2-ie.asp .

von Neumann, J. 1951. The general and logical theory of automata. In Jeffress, L. A., ed., *Cerebral Mechanisms*

*in Behavior—The Hixon Symposium,* 1–41. New York: John Wiley. Originally presented in September, 1948. Also collected in Aspray, W., and Burks, A. W., eds., *Papers of John von Neumann on Computing and Computer Theory,* 391–431. 1987. Cambridge, MA: MIT Press.

von Neumann, J. 1966. *Theory of Self-Reproducing Automata.* Urbana, IL: University of Illinois Press. Edited and completed by A. W. Burks.