

# Non-Computability of an Algorithm to Know If a Program is a Virus

Pedro Ramos Brandao

Instituto Superior De Tecnologias Avancadas (ISTEC), Universidade De Evora (CIDEHUS)

## Abstract

The main objective of this work is to demonstrate that it is extremely difficult to design an algorithm that can detect whether a program is a virus or not. The computational question of decidability and indecidability is developed. Non-computable algorithms. The application of Rice's theorem and the Recursion theorem. Important definitions are given, related to recognizable algorithms, and on their computability. A demonstration is made of the existence of self-replication of viruses through the recursion theorem. A demonstration is made of how undecidable the detection by an algorithm is, whether a program is a virus. This demystifies the idea that algorithms can be designed for this purpose.

## Keywords

Virus Detection, Negative Diagonalization, Indecibility, Rice's Theorem, Recursion Theorem

## I. Introduction

Currently, there is the perception and idea that a robust security system, well configured, properly updated with permanent monitoring, will be able to detect any program or file that is a virus. It turns out that this is a false assumption. A software called antivirus compares a set of files or code with a list of data, if it finds a match then it indicates the existence of a probable virus. But if the file or code is not in the antivirus database, no one will know about the danger. That is why companies that produce antivirus software insist strongly that these programs are always up to date. Which doesn't always happen. Many people are unaware that an algorithm that recognizes whether a program is a virus is not computable. More specifically, we speak of a program being undecidable to detect whether a program is a virus or not. The aim of this work is to demonstrate that fact (1).

## II. Important Definitions

An algorithm is a finite symbolic representation.

$f : \text{input} \rightarrow \text{output}$

It is computable if there is an algorithm P that calculates f

$P(x) = f(x)$  P whatever it is  $x \in \text{input}$

We can say that a language is a set of inputs.

And an algorithm is decidable if P such that:

$$P(x) = \begin{cases} 1 & \text{se } x \in L \\ 0 & \text{se } x \notin L \end{cases}$$

On the other hand, an algorithm is recognizable if, such that:

$A(x) = 1$  if and only if  $x \in L$

So we can also talk about non-computable functions, if:

$f: N \rightarrow N$

$$f(n) = \begin{cases} x + 1 & \text{se } P_n(N) = x \\ 0 & \text{se } P_n(N) \text{ não terminar} \end{cases}$$

It is demonstrated through the system by reduction to the absurd.

Suppose there is an algorithm to calculate the function f for

example Pk a (K)?

$P_k(K) = f(x)$

We are left with two hypotheses:

(a) Pk (K) ends and  $P_k(K) = f(x) = x$

Implies  $\rightarrow P_k(K) = f(k) = x + 1$

What is a contradiction:

$x \neq x + 1$

or

(b) Pk (K) does not end

$P_k(K) = f(x) = 0$

What is not possible.

Therefore, the contradiction is an impossibility of the results obtained. It proves that the algorithm is not computable. Because if it were, both results would be reasonable.

So not all functions are computable, they are only in the case: f is computable if there is an F algorithm such that whatever the word x,  $F(x) = f(x)$

An algorithm must have a rigorous description to solve a problem and have a finite symbolic description. We can always enumerate the algorithms:

A1, A2, A3, A4, ..... ..

Thus, an L language is a set of words. There are decidable languages and recognizable languages (2).

Decisible languages:

There is an algorithm D such that

$D(x) = 1$  if  $x \in L$

and

$D(x) = 0$  if  $x \notin L$

Recogniable language (3):

There is an R algorithm such that

$R(x) = 1$  if and only if  $x \in L$

Demonstration that an algorithm is decidable (4):

Prime numbers (decimal notation)

Given an x:

For each of the numbers 2, ..., X- 1

If the remainder of the x division gives him 0

If output = 0

Otherwise D  $\rightarrow$  D + 1

At the end output 1

Demonstrating better. The HALT language is recognizable but is not decidable.

(a) R: input (P, x)

Execute (P, x)

If and when you finish output 1

(b) Demonstration:

(HALT is not decidable)

If it is by reduction to the absurd that an algorithm existed:

D who decides HALT, ie

$D(P, x) = 1$  if  $P(x)$  ends and

$D(P, x) = 0$  if  $P(x)$  does not end

If it existed, another Y algorithm could be built that would do the following:

H: input Y

If  $D(Y, Y) = 1$  if it is 1

So infinite loop

If  $D(Y, Y) = 0$

Then (Finished) output 1

Did H(h) end?

H(h) ends if and only if  $D(Hh) = 0$

If and only if H(h) does not end.

It is therefore a contradiction.

### III. Demonstration of The Existence of Autoreplicable Virus Using the Recursion Theorem

“Virus” are computer programs designed to spread to other computers, that is, a host computer transmits copies of the program to other devices. A recursive function then uses itself to arrive at a result, just as it does when calculate a factorial number. That is,  $n! = n \times n-1!$  To make this calculation on the computer, for example, it is necessary to define a limit for “n” as 1 and to define that the numbers are integers.

So,  $4! = 4 \times 3! = 4 \times 3 \times 2! = 4 \times 3 \times 2 \times 1!$ . As  $1! = 1$ , it is possible to solve the problem recursively.

The recursion theorem plays an important role in the theory of computability, as it allows any program to have the ability to refer to its own description and compute with it.

We introduce the subject through a paradox that arises in the study of life, in the sense that machines are able to replicate themselves.

This paradox is exposed as follows:

- Living things are machines;
- Living things can reproduce;
- Machines cannot reproduce themselves.

The recursion theory indicates and demonstrates that the third statement above is mistaken, insofar as machines can be used to build other machines.

Thus, we begin to demonstrate the existence of self-replicating viruses.

There is a computable function q, where w is any string, q(w) is the description of a Turing Pw Machine, which prints w and stops.

In simple language, the function of this MT is to print the contents of a string and stop.

Therefore, it is possible to build a Turing machine (MT) in which w is its own description and returns w when started.

For this, the AUTO machine is composed of two parts, called A and B.

Part A has the function of printing a description of B and, when complete, activates B. The programming of B, on the other hand, is to compute A, that is, it will execute what is already written on the tape from the result that A produced (which corresponds to B itself), combining the result with the tape content to make a complete MT, prints the description of that MT and stops.

In execution, the following behavior is observed:

1. Executes A -> this causes machine code B to be printed on the ribbon.
2. Run B on the tape string written by A (which is B’s own code)

-> MT will look at the tape and find its own code.

3. B then calculates the tape content and combines the result with the B code in a description of the MT itself (the AUTO code);
4. Print the description and stop.

That is, a TM can build a replica of its own code and continue the calculation, which can include actions contained in its own code.

Or, in non-mathematical / computer language, the author gives the following example:

Consider the following sentence:

- Print this sentence.

The computer does not know what “this” means, so it is possible to formulate this order in another way:

- Print two copies of the following, the second in quotation marks:

“Print two copies of the following, the second in quotation marks:”

Part A provides the copy of the instruction to be executed, so that B can process the copy.

In this case, the question asked can be demonstrated by the fact that a V program, which is composed of a computer virus, may have the ability to reproduce itself, providing as output a copy of its own code.

### IV. Detecting if a Program is a Virus: Indicible

Base:

- Recursion Theorem,

- Rice’s theorem.

Definition:

A, B are equivalent

$(A \Leftrightarrow B)$  if  $A(x) = B(x)$

For any input of x

Definition:

Self-replicating virus

V such that any input x:

$V(x) \Leftrightarrow V$

Constant:

Input P                   input X

Output                   output P

Const (P) (X) = P

F:

Input M

Input X

Output

Output (const (M) (X))

(é M)

Q:

Input X

Output F (const (K) (X))

$Q(x) = F(\text{Const}(F)(X)) = Q$

So is a Virus

Q is a fixed point of F (5).

Recursion Theorem:

Whatever the F algorithm is, there is an R string.

Such that  $F \Leftrightarrow A F(R)$

Whatever the F algorithm, there is R such that  $R \Leftrightarrow F(R)$   
 (Any algorithm has a fixed point)  $\rightarrow$  is a Virus (6).

Rice's theorem:

If L is a language that is not empty or has all the words (not trivial) and L is closed to equivalences.

If there is an algorithm  $A \in L \Leftrightarrow B$  it must be in the language  $B \in L$ .

So: L is undecidable.

Any interesting property that corresponds to a property of what the program does is undecidable (7).

Application of Rice's Theorem:

$V = \{V: V \text{ is a self-replicating Virus}\}$  is undecidable.

a) V is not empty because  $Q \in V$

b) F: input x; output  $X \in V$

c) V is closed for equivalences if v is a Virus and  $U \Leftrightarrow V$

So  $U(x) = V(x) \Leftrightarrow V \Leftrightarrow U$

and U is a Virus.

### Demonstration of Rice's Theorem:

Demonstrate that an L language is undecidable. We assume from the absurdity that there is a D algorithm that decides L. Under the conditions of Rice's Theorem. Since the L language is non-trivial and not empty, I can choose one:

$A \in L$  and  $B \notin L$

And build an algorithm:

If  $D(x) = 1$  then output B

If  $D(x) = 0$  then output A

By the Recursion Theorem:

There is R such that  $F(R) \Leftrightarrow R$

Did  $R \in L$ ?

$R \in L$  if and only if  $D(R) = 1$  if and only if:

$F(R) = B$  and  $\notin L$

So it is Undecidable (8).

### V. Conclusion

Insecurity is a constant greater than security. Always bear in mind that there are countless programs that are viruses but are not liable to be identified as such. Therefore, cybersecurity has to be a somatic set of diverse technologies to mitigate this imbalance between insecurity and security. The current heuristic security systems that are characterized by the function of trying to detect malicious code in a proactive way, that is, without the need to have a specific signature or virus recognition. However, comparing behavior and patterns, generating very false positives, is not, however, a technology based on totally decidable algorithms (9).

### References

- [1] Dowling, William, "There are not safe virus tests", Am. Math. Monthly, 96.9, pp. 835.
- [2] Cormen, Thomas, "Introduction to Algorithms", The MIT Press, 2009.
- [3] Sedgewick, Robert, "Algorithms", Addison-Wesley, 2011.
- [4] Bengio, Yoshua, "Deep Learning", MIT Press, 2017.
- [5] Owings, J. C., "Diagonalization and the recursion theorem", Notre Dame Journal of Formal Logic, Vol. 14 Number 1, 1973.
- [6] Ebbinghaus, Heinz-Dieter, "Recursion Theory Week", Springer, 1985.

- [7] Surhone, Lambert, "Rice's Theorem", Betascript Publishing, 2010.
- [8] Rogers, H., "Theory of Recursive Functions and Effective Computability", McGraw-Hill, 1967.
- [9] Witten, I., "Computer (in)security: infiltrating open system, Abacus 4, 1987.



Pedro Ramos Brandao, Coordinator Professor at Instituto Superior de Tecnologias Avançadas (ISTEC). Researcher at Évora University (CIDEHUS), Portugal. PhD in Information Sciences and Computation.