How to Teach the Undecidability of Malware Detection Problem and Halting Problem

Matthieu Journault[1], Pascal Lafourcade[2], Malika More[2], Rémy Poulain[1], and Léo Robert[2]

[1] Sorbonne Université, LIP6
[2] Université Clermont Auvergne, LIMOS, IREM

Abstract. Malware detection is a term that is often associated to Computer Science Security. The underlying main problem is called *Virus detection* and consists in answering the following question: Is there a program that can always decide if a program is a virus or not? On the other hand, the undecidability of some problems is an important notion in Computer Science : an undecidable problem is a problem for which no algorithm exists to solve it. We propose an activity that demonstrates that virus detection is an undecidable problem. Hence we prove that the answer to the above question is no. We follow the proof given by Cohen in his PhD in 1983. The proof is close to the proof given by Turing in 1936 of the undecidability of the Halting problem. We also give an activity to prove the undecidability of the Halting problem. These proofs allow us to introduce two important ways of proving theorems in Computer Science : proof by contradiction and proof by case disjunction. We propose a simple way to present these notions to students using a maze. Our activity is unplugged, i.e. we use only a paper based model of computer, and is designed for high-school students. This is the reason why we use Scratch to write our "programs".

Keywords: Virus detection problem, Halting problem, Undecidability, Computer Science Unplugged.

1 Introduction

After Alan Turing gave birth to Computer Science in 1936 [1,2] and the technological advances that ensued, nowadays computers are ubiquitous. The original main concerns of Computer Science were to design hardwares and softwares that can solve some problems. Afterwards, the main challenge became to establish communications between computers. This was the goal of the project ARPANET (Advanced Research Projects Agency Network) initiated by the DARPA (Defense Advanced Research Projects Agency) in the USA in 1962, but really born in 1969, with a first demonstration in 1972. Finally in 1986, the Kenbak-1 was the first Personal Computer (PC), according to the Computer Museum of Boston. Unfortunately, all these projects were designed for an ideal world where everyone is honest and there is no malicious participant.

However, as soon as a system is working, it is unavoidable that some people try to hack or attack it. This is the reason why security is one of the main issues in Computer Science. Some of the most popular terms associated to Computer Science are malware and antivirus. Following the definitions given by F. Cohen in his PhD and in [3], let us clarify some related notions :

– a *malware* is a set of instructions that runs on a computer and makes a system do something that an attacker wants it to do.
– a *virus*[3] is a program that can infect other programs by modifying them to include a, possibly evolved, version of itself.
– a *worm* is a similar program, which has in addition an automatic propagation mechanism without human assistance. In contrast, a virus needs human actions such as opening an email attachment.
– a *ransomware* is a malware, that encrypts the victim's data unless a ransom is paid.
– a *trojan* is a program that seems harmless but performs malicious action.

History shows that the evolution of malwares closely follows the evolution of computers. In 1982, the high-school student Rich Skrenta wrote the first virus ever released in the wild, called *Elk Cloner*. It was a boot sector virus. Then the virus called *Brain*, designed by Basit and Amjood Farooq Alvi in 1986, was the first virus to infect some PCs. It affected IBM PCs by replacing the boot sector of a floppy disk with a copy of the virus. The first internet worm was the *Morris Worm*, written in 1988 by the Cornell student Robert Tappan Morris. The first ransomware was designed by Joseph L. Popp in 1989. It was called *AIDS/PC Cyborg*, was sent by postmail on a floppy disk and requested 189 US$.

On the other hand, an important part of the research in Computer Science consists in drawing the frontier between undecidable and decidable problems. A problem is said to be *decidable* if there is an algorithm that can solve it, and *undecidable* otherwise. In spite of the existence of many well known undecidable problems (e.g., [2,4,5,6,7,8]), this very notion is often unknown to Computer Science students. However, we think that it is an important notion all programmers should know about, in order not to loose their time trying to solve undecidable problems whenever they encounter one. One of the most simple examples is the undecidability of the resolution of a system of diophantine equations [6]. It is a problem that can naively pass as an easy exercise for an unknowing teacher. For instance, she might consider that designing a software to solve systems of equations where coefficients and solutions are real numbers might be too difficult, because computers are not designed to manipulate real numbers. Hence this teacher might want to simplify the problem by replacing real numbers by positive natural integers, which is exactly solving a system of diophantine equations. Thus, we believe that the notion of undecidability is an important concept that we should teach to students (and teachers) as soon as possible. Teaching them

[3] The term of virus was first used in 1972 in a science fiction novel written by the American David Gerrold and called "When HARLIE Was One", where a program called VIRUS that reproduces itself appears.

this notion also helps us to stop let them think that a computer can already solve any problem, or that it is just a matter of time until an AI solves it in the next years.

In this article, we show that there cannot exist an algorithm that always decides if a program is a virus or not, which is indeed an undecidability proof.

Contributions: We present a pedagogical activity for high school students in order to prove that a perfect antivirus does not exist. For this, we follow the approach presented in the PhD of F. Cohen in [3]. This activity is unplugged, it means it does not require any computer, but only some printed papers, following the line of thought proposed in [9]. In order to prepare students to this proof, we also designed an activity that proves that it is not possible to determine if a program terminates or not on a given input. This activity follows Turing's proof of the undecidability of the Halting problem [1,2]. These two proofs use some essential techniques in Computer Science: proof by case disjunction and proof by contradiction. In order to present these two proof techniques to students, we use a simple situation, where Alice aims at escaping from a maze. Moreover, our material uses some programs written in Scratch [10], but of course other languages can be used instead, such as Snap! [11], Python, C, Java, etc.

Outline: In Section 2, we give a few simple programs that help the students to understand how our computer model works. In Section 3, we explain how a proof by contradiction and a proof by case disjunction work, using a simple example based on a maze. In Section 4, we present the proof of undecidability of the Halting problem and in Section 5 we prove that a perfect antivirus cannot exist.

2 Preliminary Work

We first introduce the material used for this activity. Then we describe each "Scratch" program used to prove the two undecidability results.

2.1 Material

In order to simulate the execution of our programs on a computer, we use a simplified model. The computer is viewed as a "*slate*" (or a sheet of paper) on which is drawn the diagram of Figure 1.

This simplified computer has two inputs, denoted by I1 and I2, two outputs denoted by O1 and O2, and three register boxes denoted by A, B and C. The goal of the next section is to get used to operating this simplified computer by handling small programs. In addition, students get acquainted with an abstract model of computation. In this activity, the computer model is closed to a three registers machine.

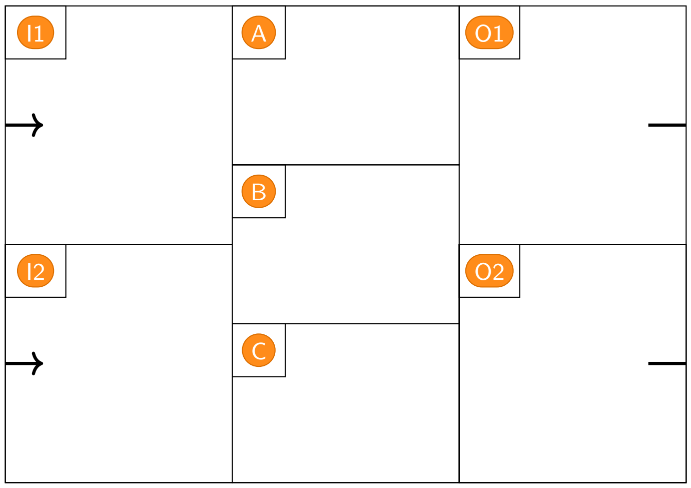


Fig. 1: Computer model.

2.2 Simple "Scratch“ Programs

The word *machine* denotes a group of three participants, equipped with a slate on which is drawn the diagram of Figure 1. The slate is used to help representing the states of the computer during the execution of a program.

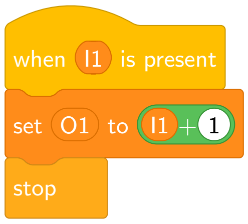
Thanks to the simple examples given in Figures 2 to 5, we explain how programs are executed in our computer model. For each case, a copy of the program is given to the students. The programs are presented with a pedagogical progression, which we believe helps to understand how our computer model works.

For example, the program *Increment*, given in Figure 2, is the simplest one. Each student picks an integer and uses it as the input of the program *Increment*. This program just adds one to its input and outputs the result. The goal of this program is to get used to the material.

The next program is *Minus*, given in Figure 3, which, as its name explains, just makes the subtraction of its two inputs and outputs the result. It uses a loop and contains internal variables.

The next program, *Minus**, given in Figure 4, is similar to the previous one but does not terminate. For the students, it is often the first time they encounter a program that does not terminate. The students must realize that this program *Minus** is obtained from the program *Minus* just by “forgetting” one instruction. Thus they should realize that it is generally hard to determine whether a program halts or not just by examining its source code.

For the program *Super*, given in Figure 5, each student picks an input and runs the program. Some executions do not terminate, depending on the inputs, while others terminate. The goal for the students is to find under which conditions the program terminates or not.



```
when I1 is present
set O1 to I1 + 1
stop
```

Fig. 2: Program Increment

3 Proof by Case Disjunction and Proof by Contradiction

In order to introduce the notions of proof by case disjunction, which is widely used in Computer Science, as well as proof by contradiction, we consider a situation where no mathematical background is required. We propose the simple maze given in Figure 6, with two deadly obstacles, a precipice noted P and a monster noted M. If Alice meets the monster, she dies as well as if she falls into the precipice. Alice starts at point A and our goal is to prove that she cannot escape alive from the maze.

We prove this result by contradiction. Assume that Alice can escape from the maze alive. She has only two options from her starting point A : going down or going right.

We consider the two cases above and show that they are both impossible.

1. **Going right :** If Alice goes right, then she is going to fall into the precipice P, and die. Since she does not leave the maze alive this case is impossible.
2. **Going down :** If Alice goes down then she is going to be eaten by the monster M, and die. This case is also impossible.

Finally we conclude that the assumption ”Alice can leave the maze alive“ is false. Hence we have shown that Alice cannot escape the maze alive.

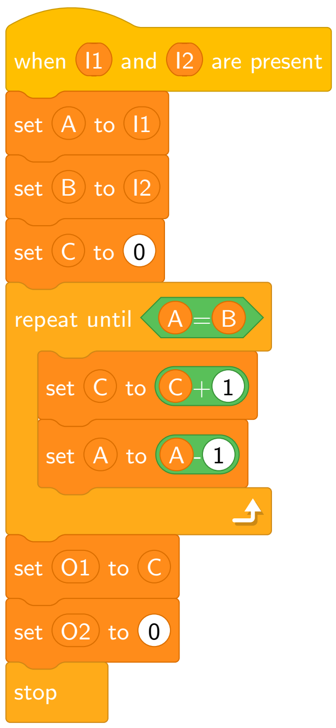
This part of the activity can be conducted as an interactive discussion with the students. Its goal is to explain step by step how to use those two proof variants on a simple example.

4 Proof of Undecidability of the Halting Problem

This section is assigned to present a proof by contradiction of the Halting problem (*i.e.,* always determine whether a given program terminates or not on a given input).

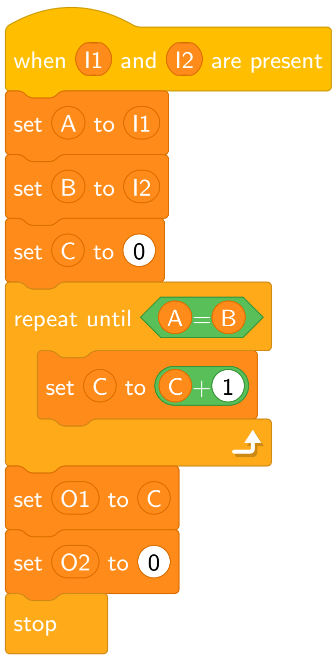
4.1 Three Programs

We present three programs used in the proof : Photocopy, Negation and Halt.



```
when I1 and I2 are present
set A to I1
set B to I2
set C to 0
repeat until A = B
    set C to C + 1
    set A to A - 1
set O1 to C
set O2 to 0
stop
```

Fig. 3: Program Minus, under condition $E1 > E2$.



```
when I1 and I2 are present
set A to I1
set B to I2
set C to 0
repeat until A = B
    set C to C + 1
set O1 to C
set O2 to 0
stop
```

Fig. 4: Program Minus*, under condition $E1 > E2$.

The program Photocopy, given in Figure 7, copies its input given in I1 to outputs O1 and O2. This program can be tested by the students by choosing numerical inputs at first. They discover that the input is just copied in the two outputs as a photocopier would do. Then we ask the students to use a program such as *Increment* as input. It is a bit perturbing for them at first, but they finally accept without too much difficulty that the input of the program Photocopy may be a program as well as a number.

The program Negation, given in Figure 8, can be tested on inputs *HALTS* and *DOES NOT HALT*. Students try both inputs and discover that the program Negation behaves the exact opposite of what its input says.

The program Halt, given in Figure 9, takes as input I1 and I2 and returns *HALTS* if the execution of the program I1 on the input I2 terminates; otherwise it outputs *DOES NOT HALT*. This program, Halt, cannot actually exist because of the following line:

execution of program I1 halts on input I2

The purpose of this activity is too prove this result.

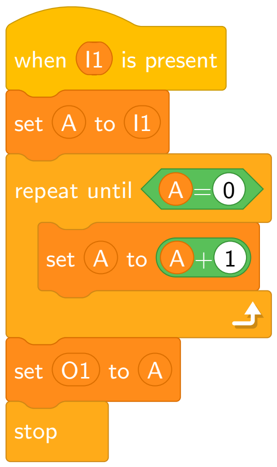


Fig. 5: Program SUPER

4.2 Undecidability Proof

The aim of this section is to prove that the program HALT, given in Figure 9, cannot exist. We assume that this program exists and show that this assumption leads to a contradiction.

The proof is constructive in the sense that we build a program, denoted by X, composed of three programs one after the other. The first one is PHOTOCOPY, the second is HALT and the third is NEGATION. The idea is now to use the program X as its own input, and show that a contradiction ensues.

We have two possibilities concerning the behavior of the program X on the input X :

1. The execution of HALT on the inputs X and X outputs *HALTS*. In this case, the program NEGATION receives *HALTS* as its input, so its execution never terminates. Finally, the execution of the program X with X as its input never terminates which contradicts the output given by HALT. The latter being assumed to never be wrong, this case is not possible.
2. The execution of HALT on the inputs X and X outputs *DOES NOT HALT*. In this case, the program NEGATION receives *DOES NOT HALT* as its input, hence its execution terminates. Finally, the execution of the program X with X as its input terminates, which contradicts the output given by HALT. The latter being assumed to never be wrong, this case is also not possible.

Both cases are impossible, which proves that the initial assumption is false. In other words, it means that the program HALT cannot exist.

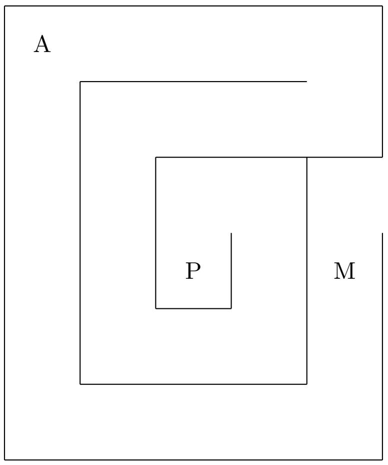
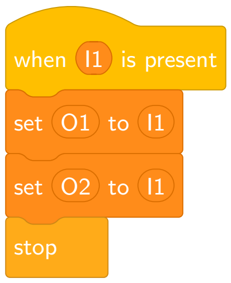


Fig. 6: Maze.



```
when I1 is present
set O1 to I1
set O2 to I1
stop
```

Fig. 7: Program Photocopy.

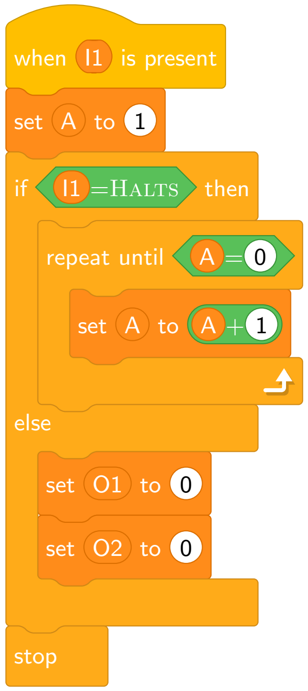
5 Proof of Undecidability of isVirus

We prove by contradiction that it is not possible to have a perfect antivirus. Hence, we assume the existence of such a program, called isVirus and given in Figure 10. This program takes as input a program P and determines whether P is a virus or not. The program isVirus *never fails* to determine if P is a virus or not.

We begin by manipulating this program isVirus with students before showing that such a program cannot exist.

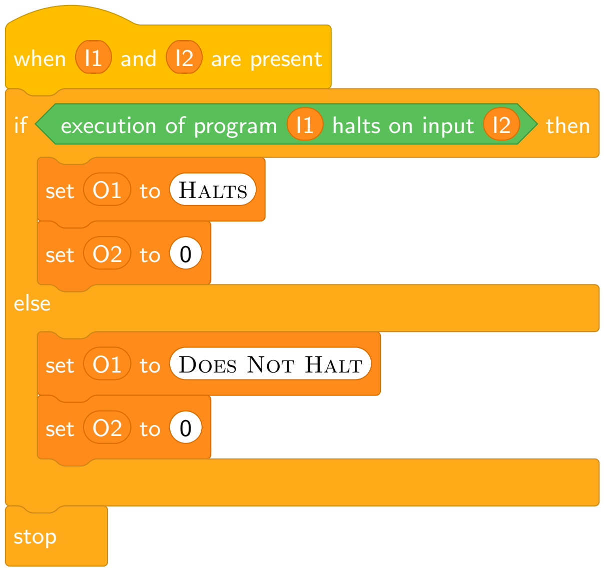
5.1 The Program isVirus

Let us consider the program called isVirus with input I1 and output TRUE in O1 if I1 is a virus and FALSE in O1 otherwise (see Figure 10).



```
when I1 is present
set A to 1
if I1 = HALTS then
    repeat until A = 0
        set A to A + 1
else
    set O1 to 0
    set O2 to 0
stop
```

Fig. 8: Program NEGATION.



```
when I1 and I2 are present
if execution of program I1 halts on input I2 then
    set O1 to HALTS
    set O2 to 0
else
    set O1 to DOES NOT HALT
    set O2 to 0
stop
```

Fig. 9: Program HALT.

5.2 The Program Test

Let us consider the program called TEST with one boolean input I1 and output 01. If I1 is TRUE then the output is set to 0 meaning that the program stops its execution. If the input I1 is FALSE then the program behaves like a virus and thus infects.

5.3 Undecidability Proof

The proof is constructive in the sense that we construct a new program, called Y formed by the program ISVIRUS followed by the program TEST, given in Figure 11.

The program Y works as follows : if the input of Y is a virus according to the program ISVIRUS, then the program Y terminates by outputing 0 in O1, and otherwise it infects the computer.

We now observe the behavior of Y when it is called with itself as its own input. We have two possibilities concerning the behavior of the program ISVIRUS on the input Y :

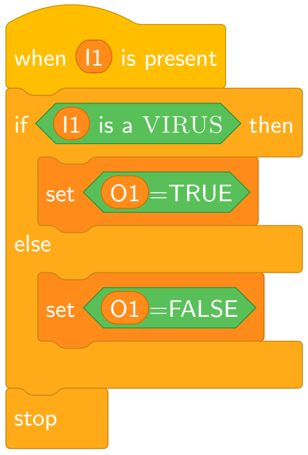


Fig. 10: Program isVirus.

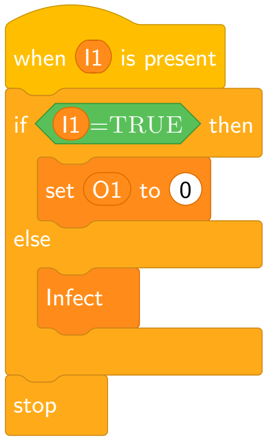


Fig. 11: Program Test.

1. **isVirus outputs TRUE.** In this case, the program Test receives as input TRUE. Thus the program Y stops. This contradicts the answer of isVirus that said that Y was a virus (and consequently should infect the computer). This leads to a contradiction and makes this case impossible.
2. **isVirus outputs FALSE.** In this case, the program Test receives as input FALSE. Thus the program Y infects the computer. This contradicts the answer of isVirus that said that Y was not a virus (and consequently should not infect the computer). This leads to a contradiction and also makes this case impossible.

Both cases are impossible, which proves that the assumption is false. In other words, it means that the program isVirus cannot exist.

6 Conclusion

In this paper, we presented a pedagogical activity demonstrating that a perfect antivirus will never exist. We also propose another activity for proving the undecidability of the Halting problem. For this, we introduced a simple computer model that only uses paper. Moreover, using a simplified maze, we present two fundamental notions that are often used in Computer Science : proof by contradiction and proof by case disjunction.

We have two main goals in this activity. First we aim at showing that Computer Science security is not an easy task and that it also has some undecidable problems. Second, we want to suggest to teachers an activity around the notion of undecidability.

This activity has been tried with high-school students but also with Master students. In both cases, the students have been surprised to discover these two undecidability results. For high-school students, the proof technique is not so easy to follow and even for university students, the principles of the proof are often new. We believe that this activity demystifies several misconceptions that students can have about the power of computers and antivirus.

Of course, it does not mean that it is useless to install an antivirus on your computer. One of the important security principles is to have an up to date system, as Bruce Schneier said “Security is a process, not a product”.

References

1. Turing, A. M.:I.—COMPUTING MACHINERY AND INTELLIGENCE. Mind (LIX)(236), 433-460 (1950)
2. Turing, A. M.:On Computable Numbers, with an Application to the Entscheidungsproblem. Proceedings of the London Mathematical Society (2)(42), 230–265 (1936)
3. Cohen, F.:Computer Viruses. Comput. Secur. (6)(1), 22–35 (1987)
4. Bodlaender, H.L., Downey, R., Fomin, F.V., Marx, D.: The Multivariate Algorithmic Revolution and Beyond - Essays Dedicated to Michael R. Fellows on the Occasion of His 60th Birthday. In: Lecture Notes in Computer Science, volume 7370, Springer (2012)
5. Cassaigne, J., Halava, V., Harju, T., Nicolas, F.: Tighter Undecidability Bounds for Matrix Mortality, Zero-in-the-Corner Problems, and More. CoRR (abs/1404.0644)(2014)
6. Matiyasevich Y.V.: Hilbert’s Tenth Problem. MIT Press (1993)
7. Rice, H. G.: Classes of Recursively Enumerable Sets and Their Decision Problems. Transactions of the American Mathematical Society (74)(2), 358–366 (1953)
8. Robinson, R.M.: Undecidability and nonperiodicity for tilings of the plane. Inventiones mathematicae (12)(3), 177–209 (1971)
9. Bell, T., Rosamond, F., Casey, N.: Computer Science Unplugged and Related Projects in Math and Computer Science Popularization. Springer Berlin Heidelberg (2012)
10. MIT, https://scratch.mit.edu (2019)
11. Berkley. Snap! https://snap.berkeley.edu (2011)