A NOTE ON COHEN'S FORMAL MODEL FOR COMPUTER VIRUSES

Kimmo Kauranen and Erkki Mäkinen
University of Tampere
Department of Computer Science
P.O. Box 607
SF-33101 Tampere, Finland

Abstract. This note discusses the formal model for computer viruses presented by Fred Cohen. We propose some refinements for the model. Especially, we define a computer virus to be a description of a Turing machine capable of writing a description of another Turing machine to the tape of a universal Turing machine.

Keywords: computer virus, Turing machine.

1. Introduction

Fred Cohen [1] has presented a formal model for computer viruses which nicely illustrates both the infection and evolution properties of viruses. The concept of *viral sets* is essential in the model. A viral set is a pair (M,W) where M is a Turing machine and W is a set of strings over its tape alphabet. Each string w in W has the property that when M, being in its start state, starts reading w it always writes another string w' of W to somewhere else in its tape. Hence, each w in W is a virus and when M (i.e. "a computer") reads it, another virus will appear somewhere in its tape (i.e. in its "memory"). The most interesting result in Cohen's model follows from the halting problem of Turing machines: it is undecidable whether or not a given pair (M, {w}) is a viral set.

In Cohen's model a Turing machine (hereafter abbreviated as TM) corresponds to computer, but it is not clear what entities correspond to programs. As a virus is a string of tape symbols, one might suppose that strings of tape symbols (of some undefined form) stand for programs, too. Although the model does not at all employ the concept of program, it would be advantageous for the clearness of the model if there were natural counterparts for the basic concepts of computer systems.

Cohen's definition for a virus requires that in order to cause a viral effect M must be in its start state when it starts reading the string w; otherwise a virus is harmless

(actually, it is undefined what happens if M starts reading a virus while not in the start state). This restriction is necessary for the internal consistency of the model, but it does not have a meaningful real life interpretation. We could protect TMs against viruses by defining their state systems in such a way that TM never return to their starting state. Unfortunately, this protection mechanism is possible only in the model, not in real computer systems!

We suggest some modifications to Cohen's model in order to overcome the above shortcomings. Instead of a TM we use the universal Turing machine (UTM) as a model of computer. Viruses are then descriptions of TMs causing another descriptions to be written to the tape of the UTM. In Cohen's model the set of viruses depends on the TM on which they are interpreted. In our modification the set of viruses depends on the rules according to which the descriptions of TMs are written.

2. Turing machines

We assume a familiarity with TMs, decidability, and related topics as given e.g. in [2]. In order to fix the notations we start by recalling the definition of TMs. We mainly follow the notations and definitions of [2]; all unexplained concepts are as in this reference.

A *Turing machine* (TM) is a 7-tuple $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$, where

- Q is the finite set of *states*,
- Γ is the finite set of *tape symbols*,
- B is a special symbol in Γ, the *blank symbol*,
- Σ, a subset of Γ not including B, is the set of *input symbols*,
- δ is the *next move partial function* from $Q \times \Gamma$ to $Q \times \Gamma \times \{\text{left, right}\}$,
- q_0 in Q is the *start state*, and
- F, a subset of Q, is the set of *final states*.

We may well suppose that F is a singleton set, i.e. there is a unique final state. M operates by making *moves* according to δ. The structure of M can be entirely described by the set of valid moves provided that the start state and the final state can be inferred from the encoding used.

Suppose the states in Q and the alphabets in Γ are named by $\{q_1, \ldots, q_n\}$ and $\{a_1, \ldots, a_m\}$, and the directions left and right are called by the synonyms d_1 and

d_2, respectively. Then a move $\delta(q_i, a_j) = (q_k, a_l, d_m)$ can be encoded by a binary string

$0^i10^j10^k10^l0^m$.

A binary code for a whole TM is

111 <code 1> 11 <code 2> 11 .... 11 <code p> 111,

where each <code r>, r = 1,..., p, is an encoding of a move according to δ and p is the number of such moves [2].

Given the above code for M plus the initial tape contents the UTM U is capable of simulating the computation of M. It is obvious that there are encodings whose simulation produces other encodings having this same property to the tape of U.

3. Formal definition of viruses

Informally, a computer virus is a program that can "infect" other programs by modifying them to include, a possible evolved, copy of itself [1]. It is now straightforward to formally define a computer virus to be a description of a TM whose simulation by the UTM causes another description of a viral TM to appear to the tape of the UTM.

The simplest virus in our model is a TM which writes a copy of itself somewhere to the tape. This TM runs on a blank tape, i.e. its description contains the valid moves only. In general, viruses may use the initial tape contents as a parameter during their evolution processes. We will not give further examples of viruses; an interested reader can create his/her favourite virus by using the standard techniques for TM construction [2,3].

By fixing the encoding for TMs we have fixed the set of strings which can be interpreted as viruses. A different encoding gives a different set of viruses. One who prefers a closer connection between the model and existing computer systems may think that different encodings for TMs correspond to different operating systems.

Viruses, defined as a special kind of TMs, have the full computational power of TMs. Then of course, the undecidability results shown by Cohen [1] hold in our model as well. Especially, it is undecidable whether or not a given string is a virus.

A careful reader might have noticed that our viruses do not actually modify other programs but rather write new programs to the tape. We can still increase the concreteness of our model by requiring that viruses indeed modify existing programs, i.e. modify the encoded descriptions of TMs. We end this chapter by sketching a TM having the desired property.

Consider a TM T performing the following tasks:

1. finds a description of some other TM, say T', from the tape,
2. inserts a special symbol to the beginning of the initial tape contents of T'
3. supplements the encoding of T' by moves having the effects described in the items 3a-3c below,
    - 3a. reading the new symbol from the causes T' to enter to a new subsystem of states,
    - 3b. a copy of T is inserted to the description of T',
    - 3c. the control is returned to the start state of T' and the head of T' is moved to the first cell of the original tape contents.

There are several details to take care of. For example, M must find out the number of states in M' in order to be able to properly label the new states to be inserted. This and other similar details are left to the reader.

4. Final remarks

We have suggested some modifications to Cohen's formal model for computer viruses. Our suggestions deal with the level of abstraction used in the model. It is our opinion that dropping the level of abstraction greatly increases the clearness of the model without affecting to the computability and other mathematical considerations related to the model.

References

[1] F. Cohen, Computational aspects of computer viruses. Computers& Security 8 (1989), 325-344.

[2] J.E. Hopcroft, and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.

[3] M. Minsky, *Computation: Finite and Infinite machines.* Prentice-Hall, London, 1972.