A Classification of Viruses through Recursion Theorems

Guillaume Bonfante, Matthieu Kaczmarek and Jean-Yves Marion
Nancy-Université - Loria - INPL - Ecole Nationale Supérieure des Mines de Nancy
B.P. 239, 54506 Vandœuvre-lès-Nancy Cédex, France

Abstract

We study computer virology from an abstract point of view. Viruses and worms are self-replicating programs, whose constructions are essentially based on Kleene's second recursion theorem. We show that we can classify viruses as solutions of fixed point equations which are obtained from different versions of Kleene's second recursion theorem. This lead us to consider four classes of viruses which various polymorphic features. We propose to use virus distribution in order to deal with mutations.

1 Theoretical Computer Virology

Abstract computer virology was initiated in the 80's by the seminal works of Cohen and Adleman [6]. The latter coined the term *virus*. Cohen defined viruses with respect to Turing Machines [7]. Later [1], Adleman took a more abstract point of view in order to have a definition independent from any particular computational model. Then, only a few theoretical studies followed those seminal works. Chess and White refined the mutation model of Cohen in [5]. Zuo and Zhou formalized polymorphism from Adleman's work [13] and they analyzed the time complexity of viruses [14].

Recently, we tried [2, 3] to formalize inside computability the notion of viruses. This formalization captures previous definitions that we have mentioned above. We also characterized two kinds of viruses, blueprint and smith viruses, and we proved constructively their existence. This work proposes to go further, introducing a notion of distribution to take into account polymorphism or metamorphism. We define four kinds of viruses:

1. A blueprint virus reproduces by just duplicating its code.
2. A blueprint distribution can mutate when they duplicates.
3. A smith virus is a blueprint virus which can use its propagation function directly to reproduce.
4. A smith distribution is a blueprint distribution which can mutate its propagation function.

We show that each category is closely linked to a corresponding form of the recursion theorem, given a rational taxonomy of viruses. So recursion theorems play a key role in *constructions* of viruses.

Lastly, we switch to a simple programming language named $\texttt{WHILE}^+$ to illustrate the fact that our constructions lives in the programming world. Actually, we follow the ideas of the experimentation of the iteration theorem and of the recursion theorem, which are developed in [8, 9] by Jones et al. and very recently by Moss in [11].

2 A Virus Definition

The $\texttt{WHILE}^+$ language. The domain of computation $\mathbb{D}$ is the set of binary trees generated from an atom $\texttt{nil}$ and a pairing mechanism $\langle\ ,\ \rangle$. The syntax of $\texttt{WHILE}^+$ is given by the following grammar from a set of variables $\mathbb{V}$:

$$\begin{array}{rcl} \text{Expressions: } \mathbb{E} & \rightarrow & \mathbb{V} \mid \texttt{cons}(\mathbb{E}_1, \mathbb{E}_2) \mid \texttt{hd}(\mathbb{E}) \mid \texttt{tl}(\mathbb{E}) \mid \\ & & \texttt{exec}_n(\mathbb{E}_0, \mathbb{E}_1, \ldots, \mathbb{E}_n) \mid \texttt{spec}_n(\mathbb{E}_0, \mathbb{E}_1 \ldots, \mathbb{E}_n) \text{ with } n \geq 1 \\ \text{Commands: } \mathbb{C} & \rightarrow & \mathbb{V} := \mathbb{E} \mid \mathbb{C}_1; \mathbb{C}_2 \mid \texttt{while}(\mathbb{E})\{\mathbb{C}\} \mid \texttt{if}(\mathbb{E})\{\mathbb{C}_1\}\texttt{else}\{\mathbb{C}_2\} \end{array}$$

A $\texttt{WHILE}^+$ program $\mathbf{p}$ is defined as follows $\mathbf{p}(\mathbb{V}_1, \ldots, \mathbb{V}_n)\{\mathbb{C}; \texttt{return}\ \mathbb{E};\}$. A program $\mathbf{p}$ computes a function $[\![\mathbf{p}]\!]$ from $\mathbb{D}^n$ to $\mathbb{D}$. We suppose that we are given a concrete syntax of $\texttt{WHILE}^+$, that is an encoding of programs by binary trees of $\mathbb{D}$. From now on, when the context is clear, we do not make any distinction between a program and its concrete syntax.

We have a self-interpreter $\texttt{exec}_n$ and a specializer $\texttt{spec}_n$ built-in:

$$\begin{aligned} [\![\texttt{exec}_n]\!](\mathbf{p}, x_1, \ldots x_n) &= [\![\mathbf{p}]\!](x_1, \ldots x_n) \\ [\![[\![\texttt{spec}_m]\!](\mathbf{p}, x_1, \ldots x_m)]\!](x_{m+1}, \ldots, x_n) &= [\![\mathbf{p}]\!](x_1, \ldots x_n) \end{aligned}$$

In the following we also use $\mathbf{spec_n}$ as a program which computes $\texttt{spec}_n$.

A Computer Virus representation. We propose the following scenario in order to represent viruses. When a program $\mathbf{p}$ is executed within an environment x, if it halts, the evaluation of $[\![\mathbf{p}]\!](x)$ is the new environment. The environement x is thought of as a finite sequence $\langle x_1, \ldots, x_n \rangle$ which represents files and accessible parameters.

As shown in [3], the construction of viruses lies in the resolution of fixed point equations. From this observation and following, we propose the following virus representation.

Definition 1 (Computer Virus)**.** Let B be a computable function. A virus w.r.t B is a program $\mathbf{v}$ such that $\forall \mathbf{p}, x : [\![\mathbf{v}]\!](\mathbf{p}, x) = [\![B(\mathbf{v}, \mathbf{p})]\!](x)$. Then, B is named a propagation function for the virus $\mathbf{v}$.

3 Blueprint Duplication

Blueprint distribution engine. From [3], a *blueprint virus for a specification g* is a program which computes g using its own code and its environment:

$$\begin{cases} \mathbf{v} \text{ is a virus w.r.t some propagation function} \\ \forall \mathbf{p}, x : [\![\mathbf{v}]\!](\mathbf{p}, x) = g(\mathbf{v}, \mathbf{p}, x) \end{cases} \tag{1}$$

Note that a blueprint virus does not use any code of its propagation function, unlike smith viruses that we shall see shortly. The solutions of this system are provided by Kleene's recursion theorem.

Theorem 2 (Kleene's Recursion Theorem [10])**.** *Let f be a semi-computable function. There is a program* $\mathbf{e}$ *such that* $[\![\mathbf{e}]\!](x) = f(\mathbf{e}, x)$.

Theorem 3. *Let g be a semi-computable function. There is a blueprint virus for the specification g.*

Proof. Let $\mathbf{g}$ compute g, we define $\mathbf{v} = \mathtt{spec}_2(\mathbf{dg}, \mathbf{g}, \mathbf{dg})$ where

```
dg (z,u,y,x){
 r := exec(z,spec2(u,z,u),y,x);
 return r;
}
```

We observe that $[\![\mathbf{v}]\!](\mathbf{p}, x) = g(\mathbf{v}, \mathbf{p}, x)$. Moreover, $\mathbf{v}$ is a virus w.r.t to $\mathtt{spec}_1$. □

Distributions of evolving blueprint viruses. An *evolving blueprint virus* is a virus can mutate but the propagation function remains the same. A distribution of evolving blueprint viruses $\mathbf{d_v}$ for a specification g satisfies

$$\begin{cases} \forall i : [\![\mathbf{d_v}]\!](i) \text{ is a virus w.r.t some propagation function} \\ \forall i, \mathbf{p}, x : [\![[\![\mathbf{d_v}]\!](i)]\!](\mathbf{p}, x) = g(\mathbf{d_v}, i, \mathbf{p}, x) \end{cases} \tag{2}$$

The existence of blueprint distributions corresponds to a stronger form of the recursion theorem, which was first proved by Case [4].

Theorem 4 (Explicit Recursion [3])**.** *Let f be a semi-computable function. There is a computable function e such that $[\![e(x)]\!](y) = f(\mathbf{e}, x, y)$ where $[\![\mathbf{e}]\!] = e$.*

Theorem 5. *Let f be a semi-computable function. There is a distribution of evolving blueprint viruses for the specification g.*

Proof. Let $\mathbf{g}$ compute g, we define $\mathbf{d_v} = \mathtt{spec}_3(\mathbf{spec_3}, \mathbf{edg}, \mathbf{g}, \mathbf{edg})$ where

```
edg (z,t,i,y,x) {
 e := spec3(spec3,t,z,t);
 return exec(z,e,i,y,x);
}
```

For any i, $[\![\mathbf{d_v}]\!](i)$ is a virus w.r.t $\mathtt{spec}_1$ and $[\![[\![\mathbf{d_v}]\!](i)]\!](\mathbf{p}, x) = g(\mathbf{d_v}, i, \mathbf{p}, x)$. □

Construction of blueprint viruses. To illustrate blueprint viruses, we consider a typical example of blueprint duplication which looks like the real life virus `ILoveYou`. This program arrives as an e-mail attachment. Opening the attachment triggers the attack. The infection first scans the memory for passwords and sends them back to the attacker, then the virus self-duplicates sending itself at every address of the local address book.

To represent this scenario we need to deal with mailing processes. A mail $m = \langle @, y \rangle$ is an association of an address @ and data y. Then, we consider that the environment contains a mailbox $mb = \langle m_1, \ldots, m_n \rangle$ which is a sequence of mails. To send a mail m, we add it to the mailbox, that is $mb := \mathtt{cons}(m, mb)$. We suppose that an external process deals with mailing.

In the following, x denotes the local file structure, and $@bk = \langle @_1, \ldots, @_n \rangle$ denotes the local address book, a sequence of addresses. We finally introduce a $\texttt{WHILE}^+$ program **find** which searches its input for passwords and which returns them as its evaluation. The specification for the scenario of `ILoveYou` is given by the following program.

```
g (v,mb,@bk,x) {
 pass := exec(find,x);
 mb := cons(''badguy@dom.com'', pass, mb);
 y := @bk;
 while (y) {
  mb := cons(hd(y), v, mb);
  y := tl(y);
 }
 return cons(mb, @bk, x);
}
```

From the specification program $\mathbf{g}$, we buid the blueprint virus $[\![\mathbf{d_v}]\!](\mathbf{g})$.

One could add polymorphic abilities to this scenario using a new parameter i within the specification. Then it would be easy to build a corresponding blueprint distribution applying Theorem 5 on the new specification.

4 Smith Reproduction

Smith Viruses. A smith virus can use a code of its propagation function to reproduce. Formally, a *smith virus* $\mathbf{v}, \mathbf{B}$ for a specification g satisfies

$$\begin{cases} \mathbf{v} \text{ is a virus w.r.t } [\![\mathbf{B}]\!] \\ \forall \mathbf{p}, x : [\![\mathbf{v}]\!](\mathbf{p}, x) = g(\mathbf{B}, \mathbf{v}, \mathbf{p}, x) \end{cases}$$

Smith viruses correspond to the double recursion theorem due to Smullyan [12].

Theorem 6 (Double Recursion Theorem [12])**.** *Let f_1 and f_2 be two semi-computable functions. There are two programs $\mathbf{e}_1$ and $\mathbf{e}_2$ such that*

$$[\![\mathbf{e}_1]\!](x) = f_1(\mathbf{e}_1, \mathbf{e}_2, x) \quad [\![\mathbf{e}_2]\!](x) = f_2(\mathbf{e}_1, \mathbf{e}_2, x)$$

Theorem 7. *Let g be a semi-computable function. There is smith-virus for the specification g.*

Proof. Let $\mathbf{g}$ compute g We define the following programs

```
dg1 (z1,z2,t1,t2,y,x) {
 e1 := spec4(t1,z1,z2,t1,t2);
 e2 := spec4(t2,z1,z2,t1,t2);
 return exec(z1,e1,e2,y,x);
}
```

```
dg2 (z1,z2,t1,t2,y,x) {
 e1 := spec4(t1,z1,z2,t1,t2);
 e2 := spec4(t2,z1,z2,t1,t2);
 return exec(z2,e1,e2,y,x);
}
```

```
pispec (g,B,v,y,p) {
 r := spec3(g,B,v,p);
 return r;
}
```

$$\mathbf{v} = \texttt{spec}_4(\mathbf{dg_2}, \texttt{spec}_1(\mathbf{pispec}, \mathbf{g}), \mathbf{g}, \mathbf{dg_1}, \mathbf{dg_2})$$
$$\mathbf{B} = \texttt{spec}_4(\mathbf{dg_1}, \texttt{spec}_1(\mathbf{pispec}, \mathbf{g}), \mathbf{g}, \mathbf{dg_1}, \mathbf{dg_2})$$

We observe that $[\![\mathbf{v}]\!](\mathbf{p}, x) = [\![[\![\mathbf{B}]\!](\mathbf{v}, \mathbf{p})]\!](x) = [\![\mathbf{g}]\!](\mathbf{B}, \mathbf{v}, \mathbf{p}, x)$. □

Smith Distributions. Smith distributions generate viruses which are able to mutate their code and their propagation mechanism. A *smith distribution* $\mathbf{d_v}, \mathbf{d_B}$ for a specification g satisfies

$$\begin{cases} \forall i : [\![\mathbf{d_v}]\!](i) \text{ is a virus w.r.t } [\![[\![\mathbf{d_B}]\!](i)]\!] \\ \forall i, \mathbf{p}, x : [\![[\![\mathbf{d_v}]\!](i)]\!](\mathbf{p}, x) = g(\mathbf{d_B}, \mathbf{d_v}, i, \mathbf{p}, x) \end{cases}$$

Smith distributions correspond to this double recursion theorem.

Theorem 8 (Double Explicit Recursion)**.** *Let f_1 and f_2 be two semi-computable functions. There are two computable functions e_1 and e_2 such that*

$$[\![e_1(x)]\!](y) = f_1(\mathbf{e_1}, \mathbf{e_2}, x, y) \quad [\![e_2(x)]\!](y) = f_2(\mathbf{e_1}, \mathbf{e_2}, x, y)$$

where $\mathbf{e_1}$ and $\mathbf{e_2}$ respectively compute e_1 and e_2.

Theorem 9. *Let g be a semi-computable function. There is a smith distribution for the specification g.*

Proof. Let $\mathbf{g}$ compute g. We define the following programs:

```
edg1 (z1,z2,t1,t2,i,y,x) {
 e1 := spec5(spec5,t1,z1,z2,t1,t2);
 e2 := spec5(spec5,t2,z1,z2,t1,t2);
 return exec(z1,e1,e2,i,y,x);
}
```

```
edg2 (z1,z2,t1,t2,i,y,x) {
 e1 := spec5(spec5,t1,z1,z2,t1,t2);
 e2 := spec5(spec5,t2,z1,z2,t1,t2);
 return exec(z2,e1,e2,i,y,x);
}
```

```
pispec' (g,db,dv,i,y,p) {
 return spec4(g,db,dv,i,p);
}
```

$$\mathbf{d_v} = \mathtt{spec}_5(\mathbf{spec_5}, \mathbf{edg_2}, \mathtt{spec}_1(\mathbf{pispec'}, \mathbf{g}), \mathbf{g}, \mathbf{edg_1}, \mathbf{edg_2})$$
$$\mathbf{d_B} = \mathtt{spec}_5(\mathbf{spec_5}, \mathbf{edg_1}, \mathtt{spec}_1(\mathbf{pispec'}, \mathbf{g}), \mathbf{g}, \mathbf{edg_1}, \mathbf{edg_2})$$

We observe that for any i, $[\![\mathbf{d_v}]\!](i)$ is a virus w.r.t $[\![[\![\mathbf{d_B}]\!](i)]\!]$ and $\forall i, \mathbf{p}, x$: $[\![[\![\mathbf{d_v}]\!](i)]\!](\mathbf{p}, x) = g(\mathbf{d_B}, \mathbf{d_v}, i, \mathbf{p}, x)$ □

Construction of smith viruses. To illustrate smith viruses, we present how to build a parasitic virus. Its specification function $\mathbf{g}$ is the following.

```
g (B,v,p,⟨q,x⟩) {
 infected_form := exec(B,v,p);
 return exec(p,infected_form,x);
}
```

First, it infects a new host $\mathbf{q}$ with the virus $\mathbf{v}$ using the propagation procedure $\mathbf{B}$. Then, it executes the original host $\mathbf{p}$. We obtain a smith virus using the Theorem 7.

We could modify the specification to add some polymorphic abilities: any virus of generation i infects a new host $\mathbf{q}$ with a virus of next generation using the propagation procedure of generation i. Then, we would obtain the smith distribution by the Theorem 9.

References

[1] L. Adleman. An abstract theory of computer viruses. In *Advances in Cryptology – CRYPTO'88*, volume 403. Lecture Notes in Computer Science, 1988.

[2] G. Bonfante, M. Kaczmarek, and J.-Y. Marion. Toward an abstract computer virology. In *ICTAC*, pages 579–593, 2005.

[3] G. Bonfante, M. Kaczmarek, and J.-Y. Marion. On abstract computer virology from a recursion-theoretic perspective. *Journal in Computer Virology*, 1(3-4), 2006.

[4] J. Case. Periodicity in generations of automata. *Theory of Computing Systems*, 8(1):15–32, 1974.

[5] D. Chess and S. White. An undetectable computer virus. *Proceedings of the 2000 Virus Bulletin Conference (VB2000)*, 2000.

[6] F. Cohen. *Computer Viruses*. PhD thesis, University of Southern California, January 1986.

[7] F. Cohen. On the implications of computer viruses and methods of defense. *Computers and Security*, 7:167–184, 1988.

[8] T. Hansen, T. Nikolajsen, J. Träff, and N. Jones. Experiments with implementations of two theoretical constructions. In *Lecture Notes in Computer Science*, volume 363, pages 119–133. Springer Verlag, 1989.

[9] N. Jones. Computer implementation and applications of kleene's S-m-n and recursive theorems. In Y. N. Moschovakis, editor, *Lecture Notes in Mathematics, Logic From Computer Science*, pages 243–263. Springer Verlag, 1991.

[10] S. Kleene. *Introduction to Metamathematics*. Van Nostrand, 1952.

[11] L. Moss. Recursion theorems and self-replication via text register machine programs. In *EATCS bulletin*, 2006.

[12] H. Rogers. *Theory of Recursive Functions and Effective Computability*. McGraw Hill, New York, 1967.

[13] Z. Zuo and M. Zhou. Some further theoretical results about computer viruses. *The Computer Journal*, 47(6):627–633, 2004.

[14] Z. Zuo, Q.-x. Zhu, and M.-t. Zhou. On the time complexity of computer viruses. *IEEE Transactions on information theory*, 51(8):2962–2966, August 2005.