Experiments with implementations of Recursion Theorems

Jean-Yves Marion

Ecole Nationale Supérieure des Mines de Nancy
Loria-INPL

May, 5th 2007

Joint work with G. Bonfante and M. Kaczmarek

Outline

What is a computer virus ?

Following Cohen :

1. Virus can infect programs by modifying them
2. Virus can copy itself and mutate
3. Virus can spread throughout a computer system

at least, for this talk . . .

What is a computer virus ?

Following Cohen :

1. Virus can infect programs by modifying them
2. Virus can copy itself and mutate
3. Virus can spread throughout a computer system

at least, for this talk . . .

What is a computer virus ?

Following Cohen :

1. Virus can infect programs by modifying them
2. Virus can copy itself and mutate
3. Virus can spread throughout a computer system

at least, for this talk . . .

Reproductions : A virus is a virus

- Mathematical foundations of viruses
  A virus is essentially a self-replicating program
- In 1952, Von Neumann constructs a model of self reproduction.
- Fixed points in Logics, λ-calculus Turing Machines, Recursion theorems

From von Neumann

Can an automaton be constructed, i.e., assembled and built from appropriately "raw material", by an other automaton? [. . .] Can the construction of automata by automata progress from simpler types to increasingly complicated types?

Reproductions : A virus is a virus

- Mathematical foundations of viruses
  A virus is essentially a self-replicating program
- In 1952, Von Neumann constructs a model of self reproduction.
- Fixed points in Logics, λ-calculus Turing Machines, Recursion theorems

From von Neumann

> *Can an automaton be constructed, i.e., assembled and built from appropriately "raw material", by an other automaton? [. . .] Can the construction of automata by automata progress from simpler types to increasingly complicated types?*

Reproductions : A virus is a virus

- Mathematical foundations of viruses
  A virus is essentially a self-replicating program
- In 1952, Von Neumann constructs a model of self reproduction.
- Fixed points in Logics, λ-calculus Turing Machines, Recursion theorems

From von Neumann

> *Can an automaton be constructed, i.e., assembled and built from appropriately "raw material", by an other automaton? [...] Can the construction of automata by automata progress from simpler types to increasingly complicated types?*

Today Menu

- Virus definition based on Kleene's recursion theorem
- Viruses are fixed-point of equations
- Characterizations of viruses
  - based on virus duplication/propagation
  - based on recursion theorems
- Implementation of recursion theorems in concrete language

- Rogers, *Rogers*, *Theory of recursive functions and effective computability*, 1967
- Jones, *Computability and Complexity, from a programming perspective*, MIT Press, 1997

A concrete programming language

The domain of computation $\mathbb{D}$: the set of binary trees.

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

A program

$$\mathtt{p}(\mathbb{V}_1, \ldots, \mathbb{V}_n)\{\mathbb{C}; \mathtt{return}\ \mathbb{E}; \}$$

Sending emails

Send a message *msg* to all mail addresses in the list *adr* using some mail service (`mailer`)

```
send(adr,msg)
 while (adr) {
  mailer(cons(hd(y),msg));
  adr := tl(adr);
 }
 return true;
}
```

Semantics

$$[\![\_]\!] : \text{Programs} \times \mathbb{D}^* \rightarrow \mathbb{D}^*$$

where a value of $\mathbb{D}^*$ is a system environment.

From the above example

$$[\![\mathit{send}]\!](\text{spider@man.com}, ''\mathit{Hello}'', \text{Out})$$
$$= \mathit{cons}(\mathit{cons}(\text{spider@man.com}, ''\mathit{Hello}''), \text{Out})$$

Where *Out* is an output stream.

Semantics

$$[\![\_]\!] : \text{Programs} \times \mathbb{D}^* \to \mathbb{D}^*$$

where a value of $\mathbb{D}^*$ is a system environment.

From the above example

$$\begin{aligned}&[\![send]\!](\text{spider@man.com},''\,Hello'', \text{Out})\\ =\;& cons(cons(\text{spider@man.com},''\,Hello''), \text{Out})\end{aligned}$$

Where *Out* is an output stream.

ILoveYou

ILoveYou scenario

ILoveYou is an e-mail attachment.
Opening the attachment triggers the attack.
First, it scans for informations `find`
Second, it extracts an address book `extract`
Then it duplicates sending copies of itself.

I Always Love You

Suppose that f is a system entry point,
A specification of ILoveYou is:

```
love(v,f) {
 info  := find(f);  // find  informations
 send(cons("badguy@dom.com",nil),info);
 @bk := extract(f); //extract addresses
 send(@bk,v); //send virus to @bk
 return true;
}
```

v should behaves as ILoveYou if:

$$[\![v]\!](f) = [\![\mathtt{love}]\!](\mathbf{v}, f)$$

I Always Love You

Suppose that f is a system entry point,
A specification of ILoveYou is:

```
love(v,f) {
 info  := find(f);  // find  informations
 send(cons("badguy@dom.com",nil),info);
 @bk := extract(f); //extract addresses
 send(@bk,v); //send virus to @bk
 return true;
}
```

v should behaves as ILoveYou if:

$$[\![v]\!](f) = [\![\texttt{love}]\!](\mathbf{v}, f)$$

Fixed point equation

Find v satisfying ILoveYou equations

$$[\![v]\!](f) = [\![\texttt{love}]\!](\mathbf{v}, f)$$

and v is a virus specified by `love`.

- Similar to quines
- Ken Thompson: "Reflections on Trusting Trust" (CACM-84)
- No $0 variable as in shell
- no fancy pointer mechanisms

Kleene's recursion theorem

A general solution to fixed point equations is given by

Theorem (Kleene's Recursion Theorem (1938))

If p is a program, then there is a program e such that

$$[\![e]\!](x) = [\![p]\!](\mathbf{e}, x) \tag{1}$$

A solution of IloveYou equation

$$[\![v]\!](f) = [\![\texttt{love}]\!](\mathbf{v}, f)$$

Set $v = e$ where $p = \texttt{Love}$.

Kleene's recursion theorem

A general solution to fixed point equations is given by

Theorem (Kleene's Recursion Theorem (1938))

If p is a program, then there is a program e such that

$$[\![e]\!](x) = [\![p]\!](\mathbf{e}, x) \tag{1}$$

A solution of IloveYou equation

$$[\![v]\!](f) = [\![\texttt{love}]\!](\mathbf{v}, f)$$

Set $v = e$ where $p = \texttt{Love}$.

More on semantics

- Syntax of programs like $\texttt{send}$, v
- Concrete Syntax Programs $\rightarrow \mathbb{D}$ like **send**, **v**

$$[\![v]\!](f) = [\![\texttt{love}]\!](\mathbf{v}, f)$$

Two key ingredients to cook Kleene's theorem

$\texttt{exec}$ is an interpreter.

$$[\![\texttt{exec}]\!](\mathbf{p}, x) = [\![\texttt{p}]\!](v)$$

spec is a program specializer

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

More on semantics

- Syntax of programs like `send`, v
- Concrete Syntax Programs $\to \mathbb{D}$ like **send**, **v**

$$[\![v]\!](f) = [\![\mathtt{love}]\!](\mathbf{v}, f)$$

Two key ingredients to cook Kleene's theorem

`exec` is an interpreter.

$$[\![\mathtt{exec}]\!](\mathbf{p}, x) = [\![\mathtt{p}]\!](v)$$

spec is a program specializer

$$[\![[\![\mathtt{spec}_m]\!](\mathbf{p}, x_1, \ldots x_m)]\!](x_{m+1}, \ldots, x_n) = [\![\mathbf{p}]\!](x_1, \ldots x_n)$$

State of the art

- The PhD of F. Cohen gives a definition of Viruses (1985)
- L. Adleman (1988) which coins the word “virus”
- Z. Zuo and M. Zhou.(84)
- See Eric’s first book for a gentle introduction.

Virus distributions

Theorem

There is a virus distribution $\mathtt{Dst}$ *s.t. for any specification* $\mathtt{VS}$, $[\![\mathtt{Dst}(\mathbf{VS})]\!]$ *is a virus satisfying*

$$[\![\mathtt{Dst}]\!](\mathbf{VS}) = \mathbf{v}$$

$$[\![v]\!](f) = [\![\mathtt{VS}]\!](\mathbf{v}, f)$$

Proof.

A consequence of Kleene's recursion Theorem. □

An IloveYou distribution is $[\![\mathtt{Dst}]\!](\mathbf{love})$

- $\mathtt{Dst}$ is a virus compiler

Virus distributions

Theorem

There is a virus distribution $\mathtt{Dst}$ *s.t. for any specification* $\mathtt{VS}$, $[\![\mathtt{Dst}(\mathbf{VS})]\!]$ *is a virus satisfying*

$$[\![\mathtt{Dst}]\!](\mathbf{VS}) = \mathbf{v}$$
$$[\![v]\!](f) = [\![\mathtt{VS}]\!](\mathbf{v}, f)$$

Proof.

A consequence of Kleene's recursion Theorem. $\square$

An IloveYou distribution is $[\![\mathtt{Dst}]\!](\mathbf{love})$

- $\mathtt{Dst}$ is a virus compiler

Virus distributions

Theorem

There is a virus distribution $\mathtt{Dst}$ *s.t. for any specification* $\mathtt{VS}$*,* $[\![\mathtt{Dst}(\mathbf{VS})]\!]$ *is a virus satisfying*

$$[\![\mathtt{Dst}]\!](\mathbf{VS}) = \mathbf{v}$$

$$[\![v]\!](f) = [\![\mathtt{VS}]\!](\mathbf{v}, f)$$

Proof.

A consequence of Kleene's recursion Theorem. □

An IloveYou distribution is $[\![\mathtt{Dst}]\!](\mathbf{love})$

- $\mathtt{Dst}$ is a virus compiler

ILoveYou Mutations

engine is one-to-one polymorphic engine s.t.

$$[\![[\![\texttt{engine}]\!](\mathbf{p}, i)]\!] \approx [\![\mathbf{p}]\!]$$

Mutations of ILoveYou

```
love(dv,i,f) {
 info := find(f);
 send(cons("badguy@dom.com",nil),info);
 v = exec(dv,i);
 vi = engine(v,i+random+1);
 @bk := extract(f);
 send(@bk,vi);
 return true;
}
```

$$[\![exec]\!](\mathbf{dv}, i) = [\![vi]\!](f) = \texttt{love}(\mathbf{dv}, i, f)$$

ILoveYou Mutations

`engine` is one-to-one polymorphic engine s.t.

$$[\![[\![\mathtt{engine}]\!](\mathbf{p}, i)]\!] \approx [\![\mathbf{p}]\!]$$

Mutations of ILoveYou

```
love(dv,i,f) {
 info := find(f);
 send(cons("badguy@dom.com",nil),info);
 v = exec(dv,i);
 vi = engine(v,i+random+1);
 @bk := extract(f);
 send(@bk,vi);
 return true;
}
```

$$[\![exec]\!](\mathbf{dv}, i) = [\![vi]\!](f) = \mathtt{love}(\mathbf{dv}, i, f)$$

Explicit recursion

A general solution is provided by

Theorem (Explicit Recursion Theorem)

If p is a program, then there is a program e such that for any x and y

$$[\![[\![e]\!](x)]\!](y) = [\![p]\!](\mathbf{e}, x, y) \tag{2}$$

- *e* generates fixed points
- *e* may be one-to-one
- See Case (74)

ILoveYou Mutations

ILoveYou mutations are solutions of the equations

$$[\![exec]\!](\mathbf{dv}, i) = \mathtt{love}(\mathbf{dv}, i, f)$$

Solutions are obtained by explicit recursion theorem:

Set $dv = e$ and $p = \mathtt{love}$

ILoveYou Mutations

ILoveYou mutations are solutions of the equations

$$[\![exec]\!](\mathbf{dv}, i) = \texttt{love}(\mathbf{dv}, i, f)$$

Solutions are obtained by explicit recursion theorem:

Set $dv = e$ and $p = \texttt{love}$

Mutation engine

Theorem

There is a mutation engine $\mathtt{Mut}$ *s.t. for any polymorphic virus specification* $\mathtt{VS}$, $[\![\mathtt{Mut}(\mathbf{VS})]\!]$ *is a virus satisfying*

$$\begin{aligned}[\![[\![\mathtt{Mut}]\!](\mathbf{VS})]\!](i) &= \mathbf{v}_i \\ [\![v_i]\!](f) &= [\![\mathtt{VS}]\!]([\![\mathtt{Mut}]\!](\mathbf{VS}), i, f)\end{aligned}$$

- Given a virus specification $\mathtt{VS}$, $\mathtt{Mut}(\mathbf{VS})$ outputs a polymorphic virus.
- This is a compiler of polymorphic viruses

Conclusions

- Construction of viruses from Kleene recursion theorem
- Design of a virus compiler from a specification
- Construction of viruses from explicit recursion theorem
- Design of a polymorphic virus compiler from a specification

- Consider the propagation function : Double recursion theorem
- Consider polymorphic propagation theorem : double explicit recursion theorem

Conclusions

- Construction of viruses from Kleene recursion theorem
- Design of a virus compiler from a specification
- Construction of viruses from explicit recursion theorem
- Design of a polymorphic virus compiler from a specification

- Consider the propagation function : Double recursion theorem
- Consider polymorphic propagation theorem : double explicit recursion theorem

Research Directions

Mathematical framework for computer virology:

- Classification of viruses using recursion theorems
  - Structural complexity of viruses
- Introducing new virus constructions
- Defense methods
  - Detection based on virus replication methods
  - Static virus protection based on flow policies
- Analyzing space and time of viruses
- Other frameworks: reactive programming, π-calculus, ...for mobility

Questions ?

1. *Computer virus experiments and recursion theorems*, CIE'07
2. *On abstract computer virology: from a recursion-theoretic perspective*. Journal of computer virology, 2006
3. *Toward an abstract computer virology*. In ICTAC,LNCS