Formal Avenue for Chasing Metamorphic Malware

Mila Dalla Preda
University of Verona, Italy
Joint work with Roberto Giacobazzi, Saumya Debray, Arun Lakhotia
presented by Isabella Mastroeni

CREST, May 30th 2013

MALWARE DETECTION

MALWARE = MALicious softWARE

MALWARE DETECTION

MALWARE = MALicious softWARE

Malware detector

Is a program $\mathcal{D}$ that determines whether a program P is malicious

$$\mathcal{D}(P) = \begin{cases} \textit{true} & \text{if } \mathcal{D} \text{ determines that } P \text{ is malicious} \\ \textit{false} & \text{otherwise} \end{cases}$$

Malware detection

MALWARE = MALicious softWARE

Malware detector

Is a program $\mathcal{D}$ that determines whether a program P is malicious

$$\mathcal{D}(P) = \begin{cases} \textit{true} & \text{if } \mathcal{D} \text{ determines that } P \text{ is malicious} \\ \textit{false} & \text{otherwise} \end{cases}$$

An ideal malware detector is sound and complete:

- SOUND = no false positives (no false alarms)

MALWARE DETECTION

MALWARE = MALicious softWARE

Malware detector

Is a program $\mathcal{D}$ that determines whether a program P is malicious

$$\mathcal{D}(P) = \begin{cases} \textit{true} & \text{if } \mathcal{D} \text{ determines that } P \text{ is malicious} \\ \textit{false} & \text{otherwise} \end{cases}$$

An ideal malware detector is sound and complete:

- SOUND = no false positives (no false alarms)
- COMPLETE = no false negatives (no missed alarms)

Malware detection

Standard malware detectors: Signature Checking

Identify a sequence of instructions which is unique to a malware (virus signature) then scan programs for signatures

MALWARE DETECTION

Standard malware detectors: Signature Checking

Identify a sequence of instructions which is unique to a malware (virus signature) then scan programs for signatures

- Low false positive rate, easy to use
- Cumbersome, difficult to extract automatically, easy to foil
- How can we escape signature checking?

Malware detection

Standard malware detectors: Signature Checking

Identify a sequence of instructions which is unique to a malware (virus signature) then scan programs for signatures

- Low false positive rate, easy to use
- Cumbersome, difficult to extract automatically, easy to foil
- How can we escape signature checking?

BY DYNAMICALLY MODIFYING MALWARE STRUCTURE!

ESCAPE SIGNATURE CHECKING

Polymorphic malware

The malware code is encrypted and contains a decryption routine that decrypts the code and then executes it.

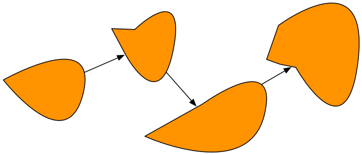
ESCAPE SIGNATURE CHECKING

Polymorphic malware

The malware code is encrypted and contains a decryption routine that decrypts the code and then executes it.

Metamorphic malware

The malware applies semantics-preserving transformations (e.g. obfuscations) to mutate its own code as it propagates.



Metamorphic Code - Examples

Equivalent code replacement

```
MOV EAX, [X]
MOV EBX, [Y]
ADD EAX, EBX
MOV [X], EAX
```

```
XOR EAX, EAX
ADD EAX, [X]
ADD EAX, [Y]
MOV [X], EAX
```

METAMORPHIC CODE - EXAMPLES

EQUIVALENT CODE REPLACEMENT

MOV EAX, [X]	XOR EAX, EAX
MOV EBX, [Y]	ADD EAX, [X]
ADD EAX, EBX	ADD EAX, [Y]
MOV [X], EAX	MOV [X], EAX

REGISTER RENAMING

MOV EAX, [X]	MOV ECX, [X]
MOV EBX, [Y]	MOV EAX, [Y]
ADD EAX, EBX	ADD ECX, EAX
MOV [X], EAX	MOV [X], ECX

Metamorphic Code - Examples

Code reordering

MOV EAX, [X]	MOV EBX, [Y]
MOV EBX, [Y]	MOV EAX, [X]
ADD EAX, EBX	ADD EAX, EBX
MOV [X], EAX	MOV [X], EAX

Metamorphic Code - Examples

Code reordering

```
MOV EAX, [X]        MOV EBX, [Y]
MOV EBX, [Y]        MOV EAX, [X]
ADD EAX, EBX        ADD EAX, EBX
MOV [X], EAX        MOV [X], EAX
```

Garbage insertion

```
MOV EAX, [X]        MOV EAX, [X]
MOV EBX, [Y]        MOV EBX, [Y]
ADD EAX, EBX        ADD EAX, EBX
MOV [X], EAX        PUSH, ESI
                    MOV [X], EAX
                    POP ESI
```

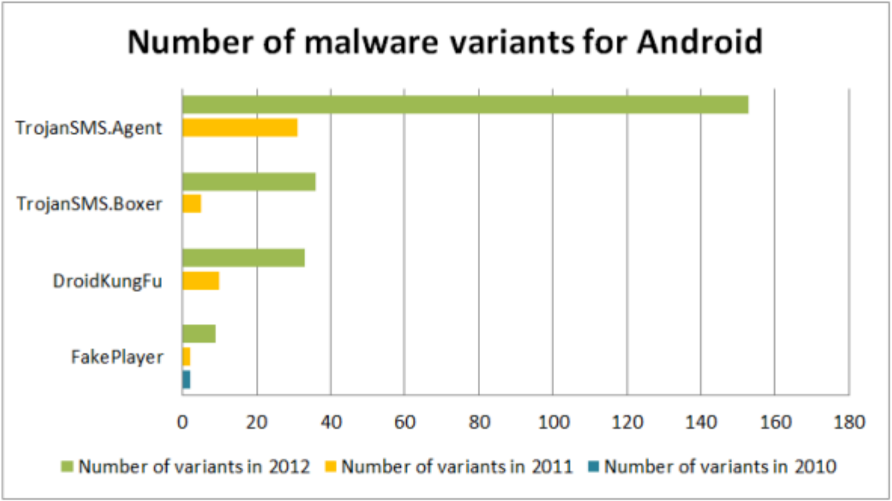


Figure 3 Number of malware variants for Android

CHASING METAMORPHISM

In order to detect metamorphic malware variants malware detector should be based on SEMANTIC program features.

CHASING METAMORPHISM

In order to detect metamorphic malware variants malware detector should be based on SEMANTIC program features.

- Abstract models of malware that ideally capture the essence of being malicious while abstracting from the details that are modified by metamorphism;
  - system call, symbolic names, automata, cfg, rewriting rules towards normal forms, model checking....

CHASING METAMORPHISM

In order to detect metamorphic malware variants malware detector should be based on SEMANTIC program features.

- Abstract models of malware that ideally capture the essence of being malicious while abstracting from the details that are modified by metamorphism;
    - system call, symbolic names, automata, cfg, rewriting rules towards normal forms, model checking....

A PRIORI KNOWLEDGE OF THE METAMORPHIC TRANSFORMATIONS

The Challenge

The malware code contains the metamorphic engine (70%)

THE CHALLENGE

The malware code contains the metamorphic engine (70%)

Metamorphic signature

is a characterization of the set $\mathcal{L}$ of the possible code variants generated by a metamorphic malware

THE CHALLENGE

The malware code contains the metamorphic engine (70%)

Metamorphic signature

is a characterization of the set $\mathcal{L}$ of the possible code variants generated by a metamorphic malware

σ IS A METAMORPHIC VARIANT $\Rightarrow \sigma \in \mathcal{L}$

THE CHALLENGE

The malware code contains the metamorphic engine (70%)

Metamorphic signature

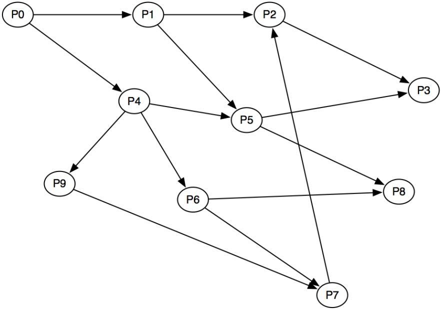
is a characterization of the set $\mathcal{L}$ of the possible code variants generated by a metamorphic malware

σ IS A METAMORPHIC VARIANT $\Rightarrow \sigma \in \mathcal{L}$

THE PROBLEM

Is there a way for systematically extracting a metamorphic signature without a priori knowledge of the metamorphic transformations used?

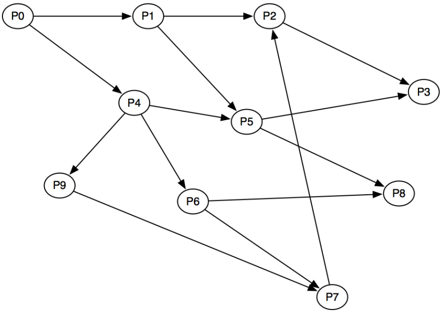
IDEALLY . . .



Program Evolution Graph

A precise description of the evolution of the code during execution

IDEALLY . . .



Program Evolution Graph

A precise description of the evolution of the code during execution

Given a self-modifying program P_0 we would like to generate its program evolution graph (or a sound approximation)

THE IDEA

- The ME is part of the code of the metamorphic malware

The Idea

- The ME is part of the code of the metamorphic malware

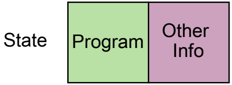
$\Rightarrow$ The description of the metamorphic behaviour – code evolution – is inside the trace semantics of the metamorphic malware

THE IDEA

- The ME is part of the code of the metamorphic malware

$\Rightarrow$ The description of the metamorphic behaviour – code evolution – is inside the trace semantics of the metamorphic malware

The state contains a description of the program that is executed

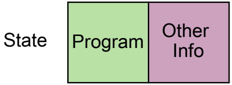


THE IDEA

- The ME is part of the code of the metamorphic malware

⇒ The description of the metamorphic behaviour – code evolution – is inside the trace semantics of the metamorphic malware

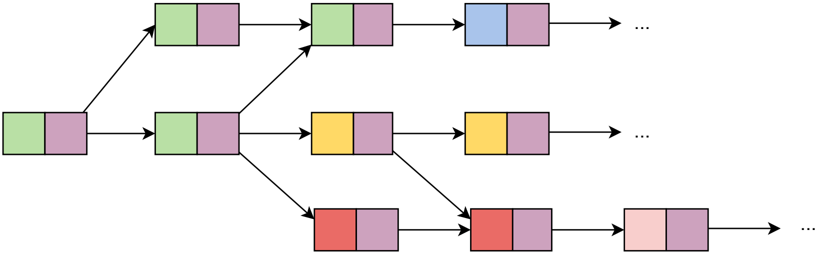
The state contains a description of the program that is executed



We use Abstract Interpretation!

TRACE SEMANTICS

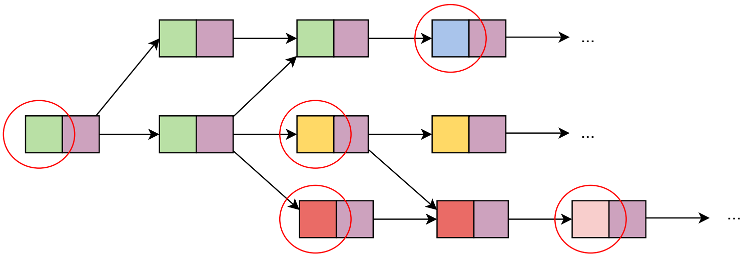
Trace semantics of a metamorphic program P



Fix-point computation of trace semantics $[\![P]\!] = lfpF_P \in \wp(\Sigma^*)$ where $F_P : \wp(\Sigma^*) \rightarrow \wp(\Sigma^*)$

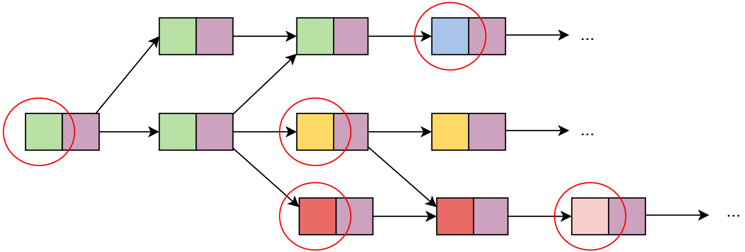
The Idea

- Isolate code evolution from the semantics of the metamorphic malware while abstracting from regular computation



The Idea

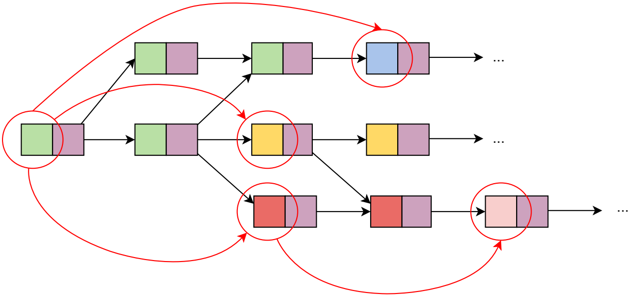
- Isolate code evolution from the semantics of the metamorphic malware while abstracting from regular computation



Idea

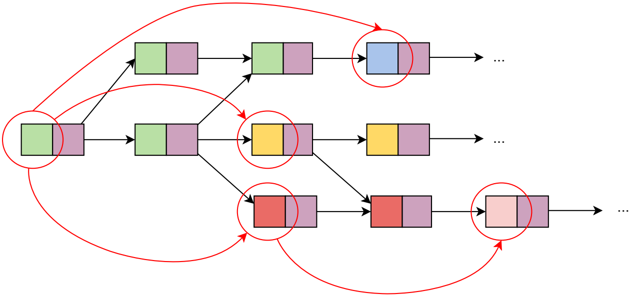
Extracting metamorphic signatures is approximating malware semantics

PHASE SEMANTICS



We define a function $F_P^\sharp : \wp(Progr^*) \rightarrow \wp(Progr^*)$ whose fix-point computation $[\![P]\!]^\sharp = lfpF_P^\sharp \in \wp(Progr^*)$ returns all the possible paths of the program evolution graph

PHASE SEMANTICS

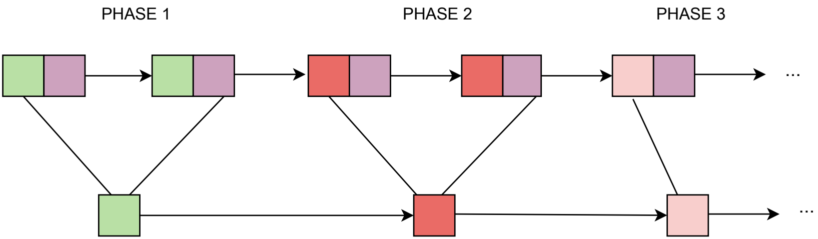


We define a function $F_P^\sharp : \wp(Progr^*) \rightarrow \wp(Progr^*)$ whose fix-point computation $[\![P]\!]^\sharp = lfpF_P^\sharp \in \wp(Progr^*)$ returns all the possible paths of the program evolution graph

PHASE SEMANTICS $[\![P]\!]^\sharp = lfpF_P^\sharp \in \wp(Progr^*)$

Phase Semantics

Idea: collect the computation that belong to the same malware version



NEED TO APPROXIMATE . . .

Phase semantics is an AI of trace semantics with no loss of precision, given $\langle \wp(\Sigma^*), \subseteq \rangle \xrightleftharpoons[\alpha^\sharp]{\gamma^\sharp} \langle \wp(\mathit{Progr}^*), \subseteq \rangle$: $\alpha^\sharp(\mathit{lfp}\mathcal{F}_P) = \mathit{lfp}\mathcal{F}^\sharp_P$

NEED TO APPROXIMATE ...

Phase semantics is an AI of trace semantics with no loss of precision,
given $\langle \wp(\Sigma^*), \subseteq \rangle \xleftrightarrows[\alpha^\sharp]{\gamma^\sharp} \langle \wp(\mathit{Progr}^*), \subseteq \rangle$: $\alpha^\sharp(\mathit{lfp}\mathcal{F}_P) = \mathit{lfp}\mathcal{F}^\sharp_P$

CONCRETE TEST FOR METAMORPHISM

Q is a metamorphic variant of P_0 iff
$\exists P_0 P_1 \dots P_n \in [\![P_0]\!]^\sharp, \exists i \in [0, n] : \; P_i = Q$
no false positives, no false negatives

NEED TO APPROXIMATE . . .

Phase semantics is an AI of trace semantics with no loss of precision,
given $\langle \wp(\Sigma^*), \subseteq \rangle \underset{\alpha^\sharp}{\overset{\gamma^\sharp}{\leftrightarrows}} \langle \wp(Progr^*), \subseteq \rangle$: $\alpha^\sharp(lfp\mathcal{F}_P) = lfp\mathcal{F}^\sharp_P$

CONCRETE TEST FOR METAMORPHISM

Q is a metamorphic variant of P_0 iff
$\exists P_0 P_1 \ldots P_n \in [\![P_0]\!]^\sharp, \exists i \in [0, n] : P_i = Q$
no false positives, no false negatives

Phase semantics is precise but undecidable

NEED TO APPROXIMATE ...

Phase semantics is an AI of trace semantics with no loss of precision, given $\langle \wp(\Sigma^*), \subseteq \rangle \underset{\alpha^\sharp}{\overset{\gamma^\sharp}{\leftrightarrows}} \langle \wp(Progr^*), \subseteq \rangle$: $\alpha^\sharp(lfp\mathcal{F}_P) = lfp\mathcal{F}^\sharp_P$

CONCRETE TEST FOR METAMORPHISM

Q is a metamorphic variant of P_0 iff

$\exists P_0 P_1 \ldots P_n \in [\![P_0]\!]^\sharp, \exists i \in [0, n] : \; P_i = Q$

no false positives, no false negatives

Phase semantics is precise but undecidable

Need to design suitable abstract domains for the approximation of phase semantics!!!

ABSTRACTING METAMORPHISM

- Design GC: $\langle \wp(Progr^*), \subseteq \rangle \underset{\alpha_A}{\overset{\gamma_A}{\leftrightarrows}} \langle A, \leq_A \rangle$
- Interpret the fix-point computation of phase semantics on the abstract domain A:

$$\alpha_A(\llbracket P \rrbracket^\sharp) \leq_A \llbracket P \rrbracket^A$$

ABSTRACTING METAMORPHISM

- Design GC: $\langle \wp(Progr^*), \subseteq \rangle \underset{\alpha_A}{\overset{\gamma_A}{\leftrightarrows}} \langle A, \leq_A \rangle$
- Interpret the fix-point computation of phase semantics on the abstract domain A:

$$\alpha_A(\llbracket P \rrbracket^\sharp) \leq_A \llbracket P \rrbracket^A$$

Abstract phase semantics $\llbracket P \rrbracket^A$ can be used as a metamorphic signature

ABSTRACTING METAMORPHISM

- Design GC: $\langle \wp(Progr^*), \subseteq \rangle \underset{\alpha_A}{\overset{\gamma_A}{\leftrightarrows}} \langle A, \leq_A \rangle$
- Interpret the fix-point computation of phase semantics on the abstract domain A:

$$\alpha_A(\llbracket P \rrbracket^\sharp) \leq_A \llbracket P \rrbracket^A$$

Abstract phase semantics $\llbracket P \rrbracket^A$ can be used as a metamorphic signature

ABSTRACT TEST FOR METAMORPHISM

Q is a metamorphic variant of P wrt A iff $\alpha_A(Q) \leq_A \llbracket P \rrbracket^A$
no false negatives

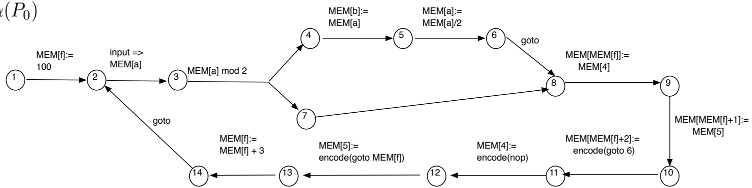
PHASES AS FSA

Code abstraction $\mathring{\alpha} : \mathit{Progr} \rightarrow \mathit{FSA}$

P_0

```
1:  mov f,100                      8:  mov MEM[f], MEM[4]
2:  input ⇒ MEM[a]                 9:  mov MEM[f+1], MEM[5]
3:  if (MEM[a] mod 2) goto 7       10: mov MEM[f+2], encode(goto 6)
4:  mov b,MEM[a]                   11: mov 4, encode(nop)
5:  mov a,MEM[a]/2                 12: mov 5, encode(goto MEM[f])
6:  goto 8                         13: mov f, MEM[f]+3
7:  mov a,(MEM[a]+1)/2             14: goto 2
```

$\mathring{\alpha}(P_0)$

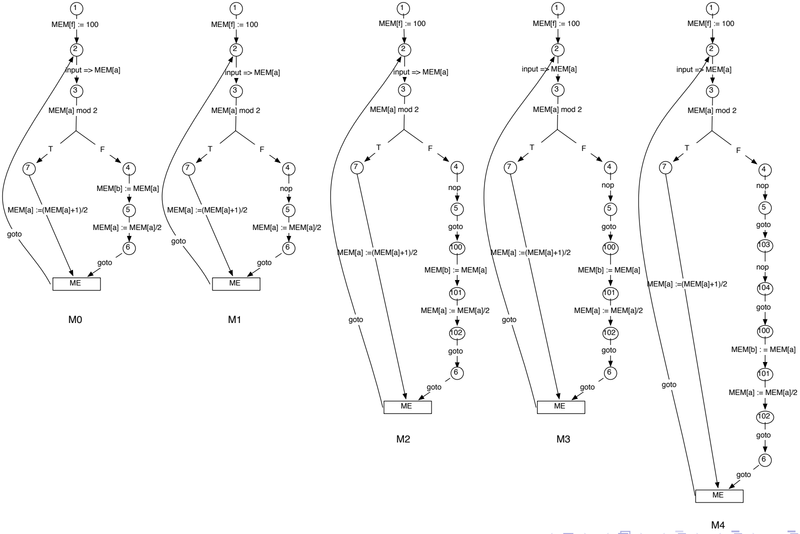


Phase Semantics as Traces of FSA

- Define a correct static approximation of the iteration function $F^{FSA} : \wp(FSA^*) \rightarrow \wp(FSA^*)$
- We derive a sound approximation of the phase semantics on the domain of traces of FSA:

$$\mathring{\alpha}(\llbracket P_0 \rrbracket^{\sharp}) \leq_{FSA} \llbracket P_0 \rrbracket^{FSA} \in \wp(FSA^*)$$

PHASE SEMANTICS AS TRACES OF FSA



WIDENING PHASES: REGULAR METAMORPHISM

Collapsing a trace of FSA into a single FSA:

- $\langle FSA/_{\equiv}, \sqsubseteq_{FSA} \rangle$ where $A_1 \sqsubseteq_{FSA} A_2 \Leftrightarrow \mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$
- let $\mathbf{W}_P$ be the limit of the widening sequence:
  $W_0 = \mathring{\alpha}(P) \qquad W_{i+1} = W_i \nabla F_P^{FSA}(W_i)$

Widening Phases: Regular Metamorphism

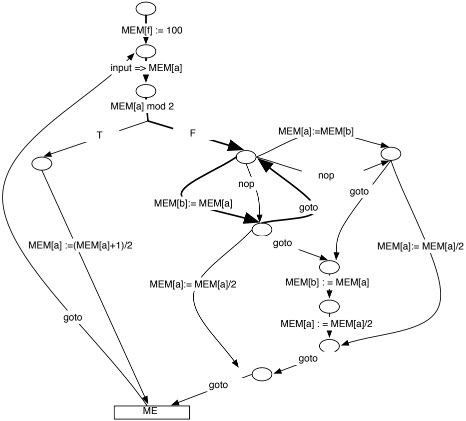
Collapsing a trace of FSA into a single FSA:

- $\langle FSA/_{\equiv}, \sqsubseteq_{FSA} \rangle$ where $A_1 \sqsubseteq_{FSA} A_2 \;\Leftrightarrow\; \mathcal{L}(A_1) \subseteq \mathcal{L}(A_2)$
- let $\mathbf{W}_P$ be the limit of the widening sequence:
  $W_0 = \mathring{\alpha}(P) \qquad W_{i+1} = W_i \nabla F_P^{FSA}(W_i)$

Abstract test for metamorphism on $FSA/_{\equiv}$

Q is a metamorphic variant of P wrt $FSA/_{\equiv}$ iff $\mathring{ABS}(Q) \sqsubseteq_{FSA} \mathbf{W}_P$
no false negatives

Widening Phases: Regular Metamorphism



Spurious trace: mov f, 100; input => a; MEM[a] mod 2 = 0; MEM[b]:= MEM[a]; goto; MEM[b]:=MEM[a].....

What we have done:

- A precise model of metamorphic code evolution named phase semantics
- Requires no a priori knowledge about the metamorphic engine
- A method for approximating the Phase semantics
- A computable approximation of regular metamorphism

What we have done:

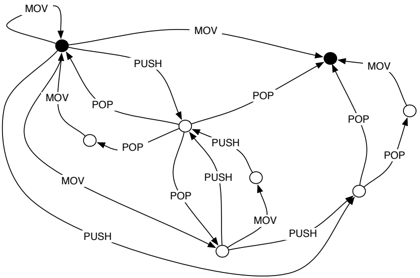
- A precise model of metamorphic code evolution named phase semantics
- Requires no a priori knowledge about the metamorphic engine
- A method for approximating the Phase semantics
- A computable approximation of regular metamorphism

WHAT'S NEXT

- Suitable for semi-automatic malware analysis: generation-test-refine
- Abstract interpretation based learning
- More advanced abstractions: e.g., context free metamorphism
- Design of new abstract domain for the analysis of code variants
- ...

METAPHOR: TOY EXAMPLE

$P =$ `mov e,10`



Approximated rules:

push; pop → mov

mov; mov → mov

Compression rules:

push e2; pop e1 → mov e1,e2

mov e2,e1; push e2 → push e1

pop e2; mov e1,e2 → pop e1

```
mov
mov,mov
push,pop
mov,mov,mov
mov,push,pop
push,pop,mov
mov,mov,mov,mov
mov,mov,push,pop
mov,push,pop,mov
push,pop,push,pop
...
```

LEARNING ME

Joint work with Arun Lakhotia

- BinJuice a tool for binary control flow graph comparison
- Virus Evol
- Extract syntactic differences between the control flow graph of successive variants (365 rules)
- Keep only semantic preserving rules
- Reduce rules (65 rules)
- Captures only block transformations, not structure transformations!