

Toward a Mathematical Understanding of the Malware Problem

CIE 2014

Michael Stephen Fiske

June 25, 2014

mf@aemea.org

What is a virus?

A virus can be described by a sequence of symbols which is able, when interpreted in a suitable environment (a machine) to modify other sequences of symbols in that environment by including a, possibly evolved, copy of itself. *

* Fred Cohen. Computer Viruses. Ph.D. Thesis. 1986.

What is a virus?

For every program, there is an infected form of that program.

A virus is a map from uninfected programs to infected programs.

Every infected program on every input (data and/or programs) performs one of the three following functions:

1. *Infection*. The program infects some other program(s) after its original task is complete.
2. *Injury*. The program computes some other function(s) besides its intended task which solely depends on the virus.
[Intent: Make broader than self-reproducing programs.]
3. *Imitation*. The program neither infects nor injures; there are no files to infect.

What is malware?

Informal. Any agent which makes one or more changes to a program or computing machine that defeats its original purpose.

Sometimes bonware might be a more appropriate name.

I know it when I see it

Undefinable?

U.S. Supreme Court Justice Potter Stewart describing his threshold test for obscenity in *Jacobellis vs. Ohio*:

I shall not today attempt further to define the kinds of material I understand to be embraced within that shorthand description ["hard-core pornography"]; and perhaps I could never succeed in intelligibly doing so.

But I know it when I see it, . . .

I. This talk is not about any of the following

Kleene's Recursion Theorem

For any recursive function $f: \mathbb{N} \rightarrow \mathbb{N}$ there exists n such that $\varphi_n = \varphi_{f(n)}$

Relation of KRT to Self-Reproducing Programs

In any general purpose programming language, there exists a program that outputs its own source code:

```
#include <stdio.h>
int main(){char*s[]={"#include <stdio.h>%cint main(){char*s[]={",
};printf(s[0],10);int i=0;while(i<3)printf("%c%%c%%s%%c,%%c%%c,34,%c",
"s[i++],34,10);printf(s[1],34,34,10);printf(s[2],10);return 0;}%c",
);printf(s[0],10);int i=0;while(i<3)printf("%c%s%c,%c",34,
s[i++],34,10);printf(s[1],34,34,10);printf(s[2],10);return 0;}
```

II. This talk is not about any of the following

Self-Reference

V. Halbach & Albert Visser. Self-Reference in Arithmetic. 2013.

Sam Moelius III. Program Self-Reference. 2009.

Incomputable Interpretations

M. S. Fiske. Turing Incomputable Computation. Turing-100, 66–91, 2012.

Research Direction

Rather than the formalization, classification and detection of malware, our attention is on the question

What makes conventional computation susceptible?

Cybersecurity Motivation

Preserving the purpose of the machine's computation is more fundamental than the confidentiality (encryption) and integrity (authentication) of the data.

Why? The purpose of the machine can be hijacked, which can compromise the confidentiality and integrity of the data.

Adi Shamir's observation

“Cryptography is typically bypassed, not penetrated.” *

* Adi Shamir. Cryptography: State of the Science. ACM. Turing Award Lecture June 8, 2003. http://amturing.acm.org/vp/shamir_2327856.cfm

Cybersecurity Motivation

Current methods of malware detection are inadequate.

Malware (virus) detection is Turing undecidable.*

Malware is using NP hard methods to encrypt and hide.**

Seeking an Alternative Approach.

Hide the computational steps of the program computation.

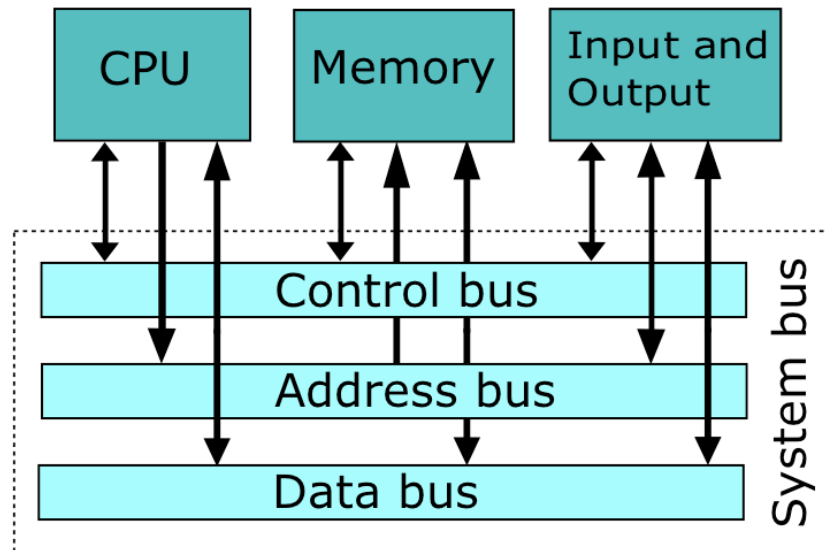
Make it more difficult to hijack program execution.

* Fred Cohen. Computer Viruses Theory and Experiments.
Computers and Security. 6(1), 22–35. February (1987).

** Eric Filiol. Malicious Cryptology and Mathematics.
Cryptography and Security in Computing. Intech. 23–50 (2012).

“One instruction at a time” vulnerability

Current processors — based on a von Neumann architecture — execute one machine instruction at a time.



To initiate execution of malignant code, an agent need only change, add or delete one machine instruction.

Two Bits Flipped in One Instruction

```
#include <stdio.h>    #include <stdlib.h>    #include <string.h>
#define    NUM_BITS    16
int powers_of_2[NUM_BITS] = {0x8000, 0x4000, 0x2000, 0x1000, 0x800, 0x400, 0x200, 0x100,
                             0x80, 0x40, 0x20, 0x10, 0x8, 0x4, 0x2, 0x1};

int greater_than(int p1, int p2) { return (p1 > p2); }
int less_than(int p1, int p2) { return (p1 < p2); }

void slow_sort(int* v, int n, int (*op)(int, int) ) {
    int i, k, x;
    for(i = 0; i < n; i++)
        for(k = 0; k < i; k++) {
            if ( op(v[i], v[k]) ) { x = v[i];    v[i] = v[k];    v[k] = x; }
        }
}

void display_numbers(int* v, int n) {
    int k;    for(k = 0; k < n; k++) printf("%d  ", v[k] );
}

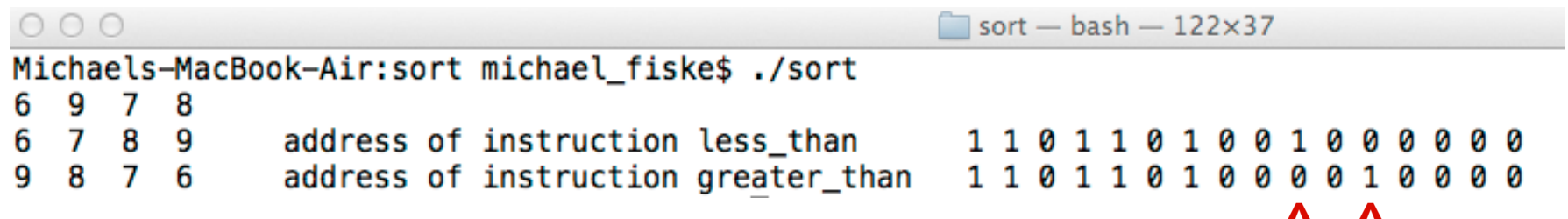
void print_binary(unsigned int v) {
    int k;
    for(k = 0; k < NUM_BITS; k++) {
        if ( v / powers_of_2[k] )    printf("1 ");    else    printf("0 ");
        v %= powers_of_2[k];
    }
    printf("\n");
}
```

Sort Order Reversal

```
void sort_print(int* numbers, int n, char* fn_name, int (*op)(int, int) ) {
    slow_sort(numbers, n, op);
    display_numbers(numbers, n);    printf("    address of instruction %s    ", fn_name);
    print_binary((unsigned int) op);
}

#define N    4

int main(int argc, char* argv[])    {
    int numbers[N] = {6, 9, 7, 8};
    display_numbers(numbers, N);    printf("\n");
    sort_print(numbers, N, "less_than    ", less_than);
    sort_print(numbers, N, "greater_than", greater_than);
    return 0;
}
```



```
Michael's-MacBook-Air:sort michael_fiske$ ./sort
6 9 7 8
6 7 8 9    address of instruction less_than    1 1 0 1 1 0 1 0 0 1 0 0 0 0 0 0
9 8 7 6    address of instruction greater_than  1 1 0 1 1 0 1 0 0 0 0 1 0 0 0 0
                                                    ^  ^
```

Machine instruction `less_than`

is only 2 bits away from instruction `greater_than`.

Apple goto fail;

```
static OSStatus SSLVerifySignedServerKeyExchange
(SSLContext *ctx, bool isRsa, SSLBuffer signedParams, uint8_t *signature, UInt16 signatureLen)
{
    OSStatus err;

    . . .

    if ((err = SSLHashSHA1.update(&hashCtx, &serverRandom)) != 0)
        goto fail;

    if ((err = SSLHashSHA1.update(&hashCtx, &signedParams)) != 0)
        goto fail;
    goto fail;

    if ((err = SSLHashSHA1.final(&hashCtx, &hashOut)) != 0)
        goto fail;

    . . .

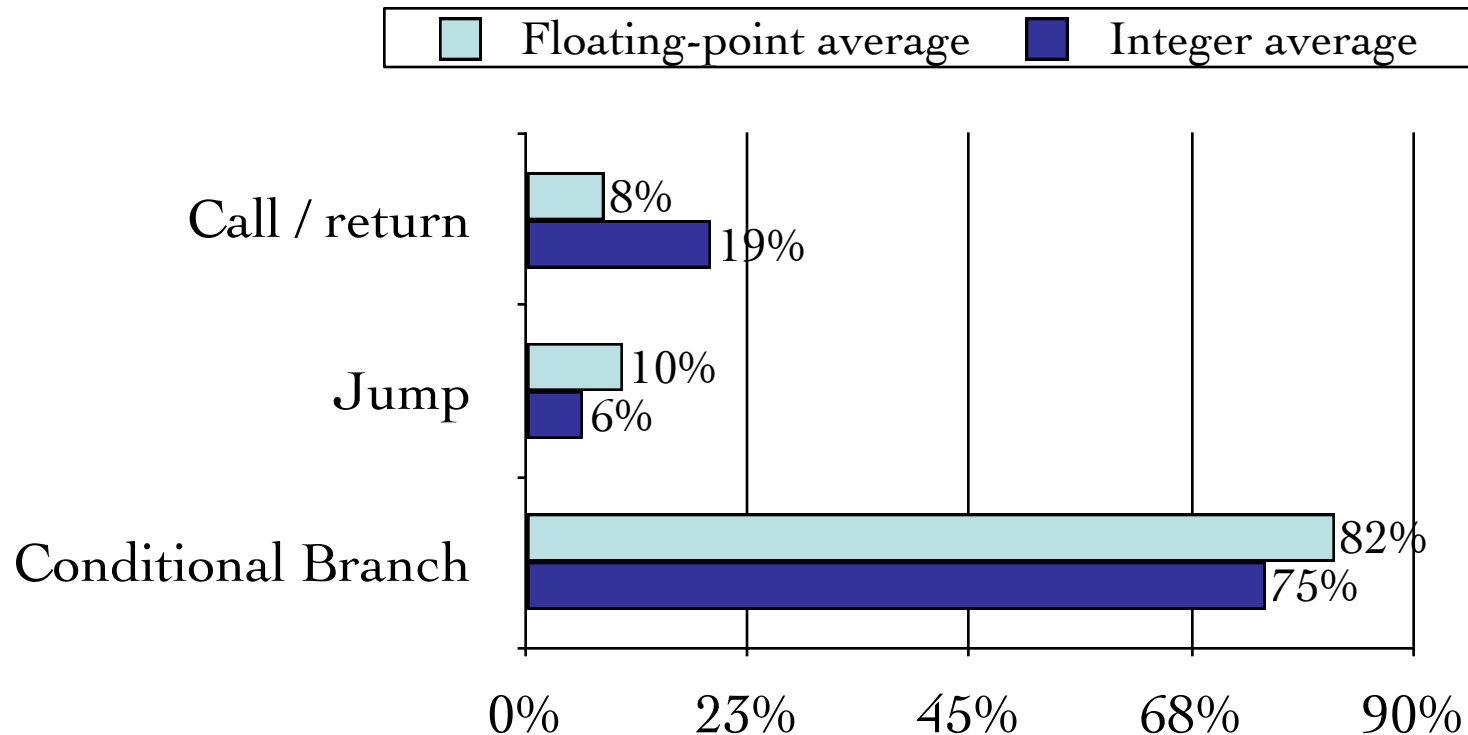
fail:
    SSLFreeBuffer(&signedHashes);
    SSLFreeBuffer(&hashCtx);
    return err;
}
```

SSL signature verification never fails.

Variable err contains a successful value.

<https://www.imperialviolet.org/2014/02/22/applebug.html>

Frequency of Control Flow Instructions



Constructed from statistics on an Alpha architecture for SPEC CPU2000 showing the average of integer programs CINT2000 and average of floating-point programs CFP 2000.*

* John Hennessy and David Patterson. Computer Architecture. 5th Edition. 2012

Conditional Branch is not Necessary

Universal von Neumann machine without Conditional Branch*

Uses LOAD, STORE, INC and GOTO instructions and
Self-Modifying Programs

GOTO is an unconditional branch instruction

*Raúl Rojas. Conditional Branching is not Necessary for Universal Computation in von Neumann Computers. Journal of Universal Computer Science. Vol 2, No. 11, 756-768, 1996.

Branching Creates a Single Point of Failure

If a branching instruction has been altered, even if there is a routine to check if the program is behaving properly, this friendly routine may never get executed.

The sequential execution of register machine instructions cripples the program from protecting itself.

Formal Roadmap

Turing Machine as a dynamical system of finite set of affine maps

Topological Conjugacy

Structural Stability

Universal Turing Machine Encoding

An Encoding Metric on the Space of these Machines

A Theorem that shows some structural instability for this encoding

Matching Steps Metric

Non-Isolated Metrics

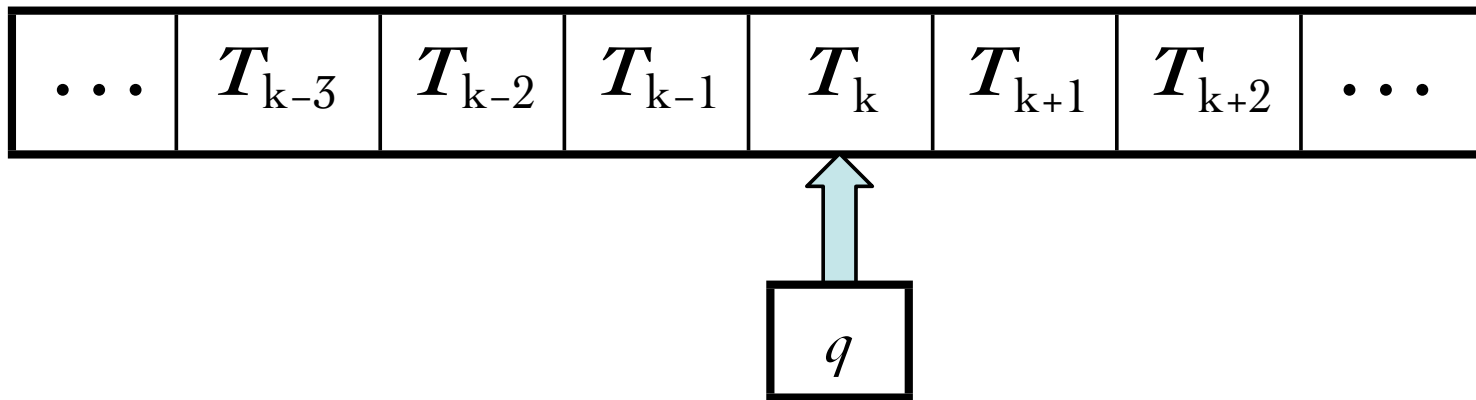
Some Unanswered Questions

Turing Machine: A Discrete Dynamical System

Map φ creates a 1-to-1 correspondence from program η to a finite set of affine functions in the x - y plane.

Base $B = |A| + |Q| + 1$. Define value function $v: \{b\} \cup Q \cup A \rightarrow N$

$$v(b) = 0 \quad v(a_i) = i \quad \dots \quad v(q_i) = i + |A| \quad \text{and} \quad v(q_{|Q|}) = |Q| + |A|.$$



Computational Steps Mapped to Affine Functions

φ maps computational step $\eta(q, T_k) = (r, \alpha, +1)$ to affine function

$$f(x, y) = (Bx + m, B^{-1}y + n) \text{ where } m = -B^2v(T_k) \text{ and } n = Bv(r) + v(\alpha) - v(q)$$

φ maps step $\eta(q, T_k) = (r, \alpha, -1)$ to affine function $f(x, y) = (B^{-1}x + m, By + n)$

where $m = Bv(T_{k-1}) + v(\alpha) - v(T_k)$ and $n = Bv(r) - B^2v(q) - Bv(T_{k-1})$.

φ maps configuration $(q, k, T) \in \mathbf{Q} \times \mathbf{Z} \times A^{\mathbf{Z}}$ to $(x(q, k, T), y(q, k, T))$ in x - y plane

$$x(q, k, T) = T_k T_{k+1} \cdot T_{k+2} T_{k+3} T_{k+4} \dots \quad \text{base } B$$

$$y(q, k, T) = q T_{k-1} \cdot T_{k-2} T_{k-3} T_{k-4} \dots \quad \text{base } B$$

Example: A Turing Machine as a discrete, dynamical system

Alphabet $A = \{\#, a, b\}$

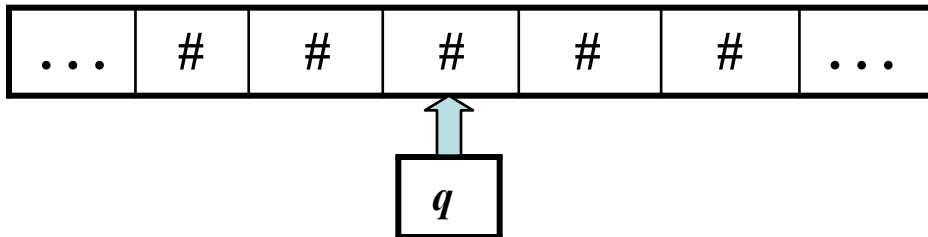
Program

η	$\#$	a	b
q	$(r, a, +1)$	$(q, a, -1)$	$(q, b, -1)$
r	$(q, b, -1)$	$(r, a, +1)$	$(r, b, +1)$
s	$(h, \#, +1)$	$(h, a, +1)$	$(h, b, +1)$

States $Q = \{q, r, s\}$

Base $B = |A| + |Q| + 1 = 7$.

$v(h) = 0$, $v(\#) = 1$, $v(a) = 2$, $v(b) = 3$, $v(q) = 4$, $v(r) = 5$, $v(s) = 6$.



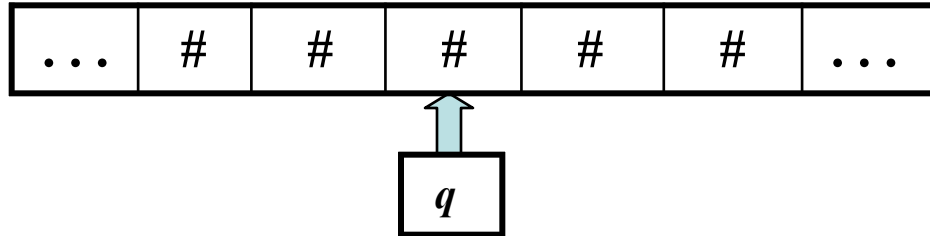
Initial machine configuration

The initial point $\mathbf{p} = (p_x, p_y)$ where

$$p_x = B v(\#) + v(\#) / (1 - 1/7) = 7 + 7/6 = 8 \frac{1}{6}$$

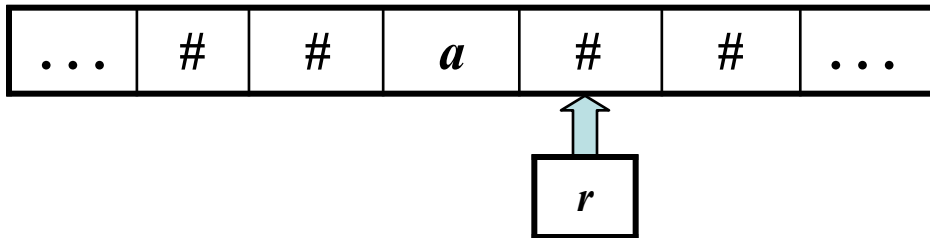
$$p_y = B v(q) + v(\#) / (1 - 1/7) = 28 + 7/6 = 29 \frac{1}{6}$$

Example: Turing Machine as a discrete, dynamical system II

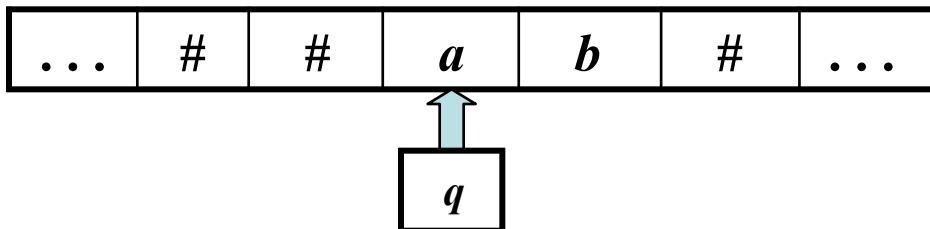


η	#	a	b
q	$(r, a, +1)$	$(q, a, -1)$	$(q, b, -1)$
r	$(q, b, -1)$	$(r, a, +1)$	$(r, b, +1)$
s	$(h, \#, +1)$	$(h, a, +1)$	$(h, b, +1)$

Apply affine function $f_1(x, y) = (7x - 49, \frac{1}{7}y + 33)$ to $p = (8\frac{1}{6}, 29\frac{1}{6})$



Apply affine function $f_{15}(x, y) = (\frac{1}{7}x + 16, 7y - 231)$ to $p = (8\frac{1}{6}, 37\frac{1}{6})$



Halting Problem from a Dynamical System Perspective

Does the orbit of point \mathbf{p} —w.r.t. this discrete autonomous, dynamical system — remain in the attractor?

If configuration (q, k, \mathbf{T}) halts after n computational steps, the orbit of $\mathbf{p} = \varphi(q, k, \mathbf{T})$ exits one of the unit squares on the n th iteration.

If configuration (r, j, \mathbf{S}) is immortal, then the orbit of $\varphi(r, j, \mathbf{S})$ remains in these unit squares (the attractor) forever.

Topological Conjugacy

$f: X \rightarrow X$ and $g: Y \rightarrow Y$ are *topologically conjugate* if there exists a homeomorphism $h: X \rightarrow Y$ that satisfies the commutative diagram.

$$f^n(p) = p \quad \text{iff} \quad g^n(h(p)) = h(p).$$

$$\begin{array}{ccc} X & \xrightarrow{f} & X \\ \downarrow h & & \downarrow h \\ Y & \xrightarrow{g} & Y \end{array}$$

The halting configurations of a TM characterize what the TM can compute and correspond to fixed points.

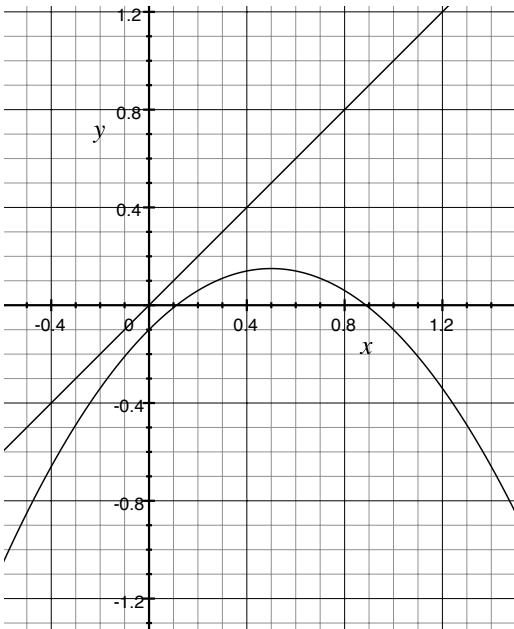
Structural Stability

(X, d) is a compact, metric space. C^0 distance between $f, g: X \rightarrow X$ is

$\mu(f, g) = \sup\{d(f(x), g(x)) : x \text{ in } X\}$. $f: X \rightarrow X$ is *structurally stable* if there exist

$\varepsilon > 0$ such that whenever $\mu(f, g) < \varepsilon$ then g is topologically conjugate to f .

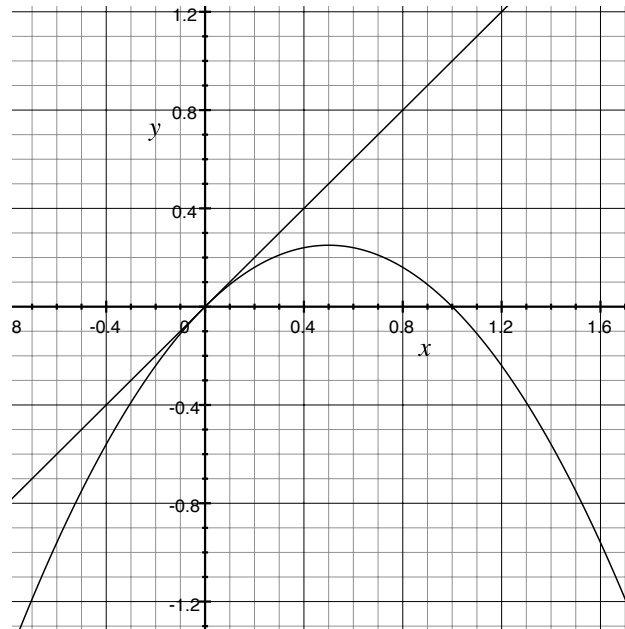
$$F_{-\frac{1}{10}}(x) = x - x^2 - \frac{1}{10}$$



Stable

0 fixed points

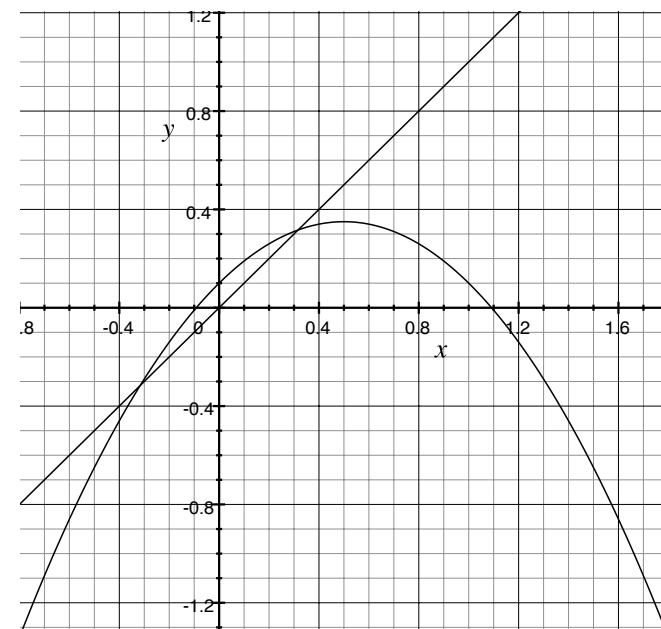
$$F_0(x) = x - x^2$$



Unstable

1 fixed point

$$F_{\frac{1}{10}}(x) = x - x^2 + \frac{1}{10}$$



Stable

2 fixed points

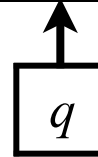
Universal Turing Machine encoding

$$A = \{\#, 0, 1\} \quad Q_k = \{q_1, \dots, q_k\}$$

Before UTM execution starts

...	#	1	1	1	1	1	1	1	#	1	0	0	0	1	#	...
-----	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	-----

Unary n 1's encode machine M_n



Input for M_n

M_1

η	#	0	1
q_1	$(h, 0, -1)$	$(h, 0, -1)$	$(h, 0, -1)$

M_{1729}

η	#	0	1
q_1	$(h, 0, -1)$	$(h, 0, -1)$	$(h, 0, -1)$
q_2	$(h, 0, -1)$	$(h, 0, -1)$	$(h, 0, -1)$

M_{215}

η	#	0	1
q_1	$(h, 0, +1)$	$(q_1, \#, +1)$	$(h, \#, +1)$

$M_{12^3+18^6+1}$

η	#	0	1
q_1	$(h, 0, -1)$	$(h, 0, -1)$	$(h, 0, -1)$
q_2	$(h, 0, -1)$	$(h, 0, -1)$	$(h, 0, -1)$
q_3	$(h, 0, -1)$	$(h, 0, -1)$	$(h, 0, -1)$

M_{1728}

η	#	0	1
q_1	$(q_1, \#, +1)$	$(q_1, \#, +1)$	$(q_1, \#, +1)$

For a fixed $|Q|$, number of distinct programs $\eta = (2|A| |Q| + 2|A|)^{|A| |Q|}$

Encoding Metric

Set $\nu(h) = 0$, $\nu(\#) = 1$, $\nu(\mathbf{0}) = 2$, $\nu(\mathbf{1}) = 3$. Set $B = 4$.

For $m \leq n$, define the distance between machine M_m and M_n

$$\begin{aligned}\rho(M_n, M_m) &= \left| \sum_{j=0}^{n-1} \nu(1)B^{-j} + \sum_{j=n}^{\infty} \nu(\#)B^{-j} - \sum_{j=0}^{m-1} \nu(1)B^{-j} - \sum_{j=m}^{\infty} \nu(\#)B^{-j} \right| \\ &= 2(B^{-m} + B^{-(m+1)} \dots + B^{-(n-1)})\end{aligned}$$

(\mathcal{T}, ρ) is a metric space where $\mathcal{T} = \{M_n : n \in \mathbb{N}\}$

THEOREM: This UTM has unstable computation in the following sense.

For any $\varepsilon > 0$, there exists two distinct Turing machines closer than ε and their respective dynamical systems are not topologically conjugate.

Outline of Proof

$Q_k = \{q_1, \dots, q_k\}$. $|Q_k| = k$. Define $f(n) = 1 + \sum_{k=1}^n (6k + 6)^{3k}$. $f(1) = 1729$.

Machine $M_{f(n)}$ has only halting configurations.

Consider machine $M_{k(n)}$ satisfying $f(n) < k(n) < f(n+1)$ and having no halting configurations.

η	#	0	1
q_1	$(q_2, 0, +1)$	$(q_2, 1, -1)$	$(q_2, \#, -1)$
\dots			
q_k	$(q_{k+1}, 0, +1)$	$(q_{k+1}, 1, -1)$	$(q_{k+1}, \#, -1)$
\dots			
q_n	$(q_1, 0, +1)$	$(q_1, 1, -1)$	$(q_1, \#, -1)$

Machine $M_{k(n)}$

$$\rho(M_{f(n)}, M_{k(n)}) < \frac{2B}{B-1} B^{-f(n)}$$

For all n , $M_{f(n)}$ and $M_{k(n)}$ are not topologically conjugate.

Matching Steps Metric

Intuition: This metric measures the distance between two machines based on how many computational steps match.

Let $M_1, M_2 \dots$ be an enumeration. η_n is the program of M_n .

Initial machine configuration: $I = (q_0, i, T)$ where $1 \leq i \leq n$

Tape $T(j) = a_j \in \{0, 1, \#\}$ where $1 \leq j \leq n$. Otherwise, $T(j) = \#$

Sequence of inputs $(q_{(1,m)}, a_{(1,m)}) \dots (q_{(k,m)}, a_{(k,m)}), (q_{(k+1,m)}, a_{(k+1,m)})$ for η_m

Sequence of inputs $(q_{(1,n)}, a_{(1,n)}) \dots (q_{(k,n)}, a_{(k,n)}), (q_{(k+1,n)}, a_{(k+1,n)})$ for η_n

If it exists, $k(I, M_n, M_m)$ is the 1st step where $\eta_n(q_{(k,n)}, a_{(k,n)}) \neq \eta_m(q_{(k,m)}, a_{(k,m)})$

For this I , the distance between M_m and M_n is $\mu(I, M_n, M_m) = 2^{-k+1}$

$\mu(I, M_n, M_m) = 0$ whenever for all k , $\eta_n(q_{(k,n)}, a_{(k,n)}) = \eta_m(q_{(k,m)}, a_{(k,m)})$

Matching Steps Metric Disconnects the Space of Machines

Γ : the set of all tapes T where $T(j) \in \{0, 1, \#\}$ whenever $1 \leq j \leq n$
and otherwise, $T(j) = \#$

Define $\rho(M_m, M_n) = \sup\{\mu(I, M_n, M_m) : I \in Q \times \{1, \dots, n\} \times \Gamma\}$

When $m \neq n$, there is an instruction where $\eta_m(q, a) \neq \eta_n(q, a)$

Initial configuration $I = (q, 1, T)$ with $T(1) = a$ implies $\mu(I, M_n, M_m) = 2^{-1+1}$

Thus, $\rho(M_m, M_n) = 1$

When $m = n$, $\rho(M_m, M_n) = 0$

The matching steps metric disconnects the space

$$\mathcal{T} = \{M_n : n \in \mathbb{N}\}$$

Non-Isolated Metrics

Metric space (X, d) . Open ball $B_d(p, \mu) = \{x \in X : d(x, p) < \mu\}$.

Version 1. Metric d is *non-isolated* if for any $\varepsilon > 0$ there exists point $p \in X$ such that $B_d(p, \varepsilon)$ is an infinite set.

Version 2. Metric d is *non-isolated* if for any point $p \in X$ and any $\varepsilon > 0$ $B_d(p, \varepsilon)$ is an infinite set.

Matching steps metric is an isolated metric.

Isolated metrics do not provide a useful method for measuring a small change to a machine.

Some Unanswered Questions

Can these initial results be generalized to non-isolated metrics and any Universal Turing machine encoding?

Limited programmability: Do conditions exist for stable non-Universal register machines?

Do conditions exist for executing stable computation with unconventional machines that can simultaneously execute multiple instructions?*

*M. S. Fiske. The Active Element Machine. Studies in Computational Intelligence. Developments and Trends. **391**, 69–96, Springer, 2012.