# Verification Techniques for Low-Level Programs

Samuel D. Pollard

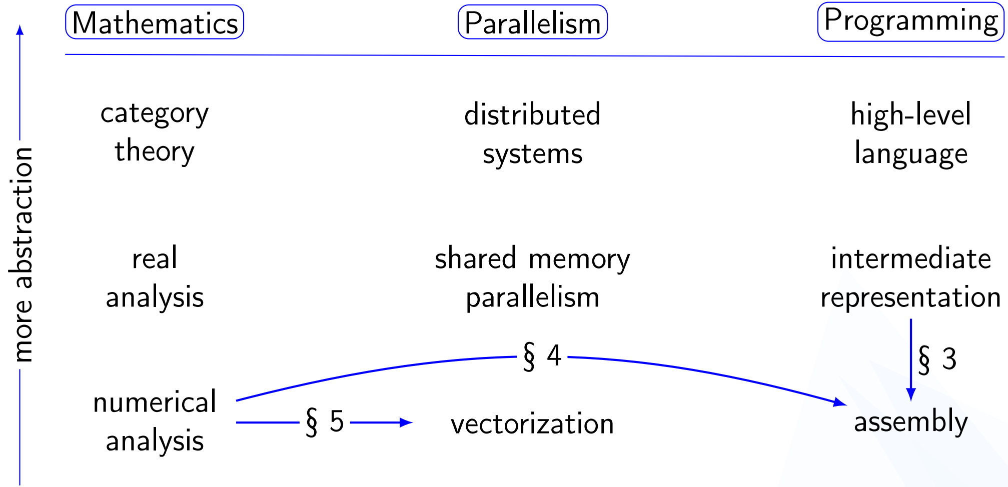
UNIVERSITY OF OREGON

Computer and Information Science

February 21, 2019

# Outline

# Why These Three Topics?



# What is Formal Methods? (In 5 Seconds)

HPCL

Using math to make sure your computer does what it's supposed to do

# What is Formal Methods?

HPCL


- The application of theoretical computer science, math, type theory, logic, and even philosophy to computer programs
- Formal methods is like a coin with two sides
  1. Designing a machine-readable and machine-checkable formal specification
     - English is always informal
     - e.g. logical formula, program in Coq, or even the type of a function
  2. Ensuring software obeys this specification.
     - {Type, model, satisfiability, proof} checker
- In math, the parallel is theorem and proof.
  - This parallel is a very deep result of computer science

# Why Do We Need Formal Methods?

- Therac-25 - race condition gave 100× radiation dosage
- Patriot Missile bug - accumulating timing error killed 28

# A Story of Paradoxes

HPCL

- Liar's Paradox: "This statement is false."
- The issue lies with self-reference
- Analogue in $\lambda$-calculus is Curry's Paradox

# Simply Typed Lambda Calculus (STLC)

- By Alonzo Church in 1940
- Invented to prevent paradoxes of untyped $\lambda$-calculus
- STLC is total so is not Turing Complete

STLC Grammar

$$
\begin{aligned}
t &::= x \mid t\ t \mid \lambda x : \tau.\ t \\
v &::= \lambda x : \tau.\ t \\
\tau &::= \text{bool} \mid \text{int} \mid \tau \to \tau \\
\Gamma &::= \emptyset \mid \Gamma, x : \tau
\end{aligned}
$$

e.g. type differently fst and True

$$(\lambda x : \tau.\ \lambda y : \tau.\ x) : \text{bool}$$

# Logics

HPCL

- Propositional Logic: $\iff, \wedge, \vee, \implies, \neg$
  - $\mathcal{NP}$-Complete: given a formula, we can determine satisfiability in at most exponential time
  - Rules of chess in 100,000 pages
- First-Order Logic: $\forall, \exists$
  - *Undecidable*: Given a formula, we cannot in general determine satisfiability
  - Rules of Chess in 1 page

# Canonical Verification Techniques

HPCL

- Hoare Logic
- Satisfiability Modulo Theories (SMT)
- Abstract interpretation

# Hoare Logic

HPCL

- Has the form $\{P\}Q\{R\}$
- Specify predicate transformer semantics
- Compute weakest precondition/strongest postcondition

```
{x > 5} x := x * 2 {x > 10}
```

# Satisfiability Modulo Theories (SMT)

- Propositional logic + theory of ______________
- Matrix Multiplication of formal methods

- Satisfiable (Integers):
  $x + 2y = 20 \land x - y = 2$
- Unsatisfiable (Integers):
  $(x > 0) \land (y > 0) \land (x + y < 0)$
- Unsatisfiable (Float) but long runtime

$$
\begin{aligned}
&(-2 \leq x \leq 2) \land (-2 \leq y \leq 2) \\
&\land(-1 \leq z \leq 1) \\
&\land(x \leq y) \land (y + z < x + z)
\end{aligned}
$$

# Abstract Interpretation

- Sound static analysis
  - Static analysis - discovering properties of a program without executing it
  - Sound - no false negatives

**3. Basic *GC* semantics** Basic *GCs* are primitive abstractions of properties. Classical examples are the *identity abstraction* $\mathcal{S}[\![\mathbb{1}[\langle\mathcal{C},\ \sqsubseteq\rangle]]\!] \triangleq \langle\mathcal{C},\ \sqsubseteq\rangle \underset{\lambda P\bullet P}{\overset{\lambda Q\bullet Q}{\leftrightarrows}} \langle\mathcal{C},\ \sqsubseteq\rangle$, the *top abstraction* $\mathcal{S}[\![\top[\langle\mathcal{C},\ \sqsubseteq\rangle,\top]]\!] \triangleq \langle\mathcal{C},\sqsubseteq\rangle \underset{\lambda P\bullet\top}{\overset{\lambda Q\bullet\top}{\leftrightarrows}} \langle\mathcal{C},\sqsubseteq\rangle$, the *join abstraction* $\mathcal{S}[\![\cup[C]]\!] \triangleq \langle\wp(\wp(C)),\subseteq\rangle \underset{\alpha^\wp}{\overset{\gamma^\wp}{\leftrightarrows}} \langle\wp(C),\subseteq\rangle$ with $\alpha^\wp(P)\triangleq\bigcup P$, $\gamma^\wp(Q)\triangleq\wp(Q)$, the *complement abstraction* $\mathcal{S}[\![\neg[C]]\!]\triangleq\langle\wp(C),\ \subseteq\rangle \underset{\neg}{\overset{\neg}{\leftrightarrows}} \langle\wp(C),\ \supseteq\rangle$, the finite/infinite *sequence abstraction* $\mathcal{S}[\![\infty[C]]\!]\triangleq\langle\wp(C^\infty),\ \subseteq\rangle \underset{\alpha^\infty}{\overset{\gamma^\infty}{\leftrightarrows}}\langle\wp(C),\ \subseteq\rangle$ with $\alpha^\infty(P)\triangleq\{\sigma_i \mid \sigma\in P\wedge i\in\mathrm{dom}(\sigma)\}$ and $\gamma^\infty(Q)\triangleq\{\sigma\in C^\infty\mid\forall i\in\mathrm{dom}(\sigma):\sigma_i\in Q\}$, the *transformer abstraction* $\mathcal{S}[\![\rightsquigarrow[C_1,C_2]]\!]\triangleq\langle\wp(C_1\times C_2),\subseteq\rangle \underset{\alpha^\rightsquigarrow}{\overset{\gamma^\rightsquigarrow}{\leftrightarrows}} \langle\wp(C_1)\xrightarrow{\cup}\wp(C_2),\ \dot\subseteq\rangle$ mapping relations to join-preserving transformers with $\alpha^\rightsquigarrow(R)\triangleq\lambda X\bullet\{y\mid\exists x\in X:\langle x,\ y\rangle\in R\}$, $\gamma^\rightsquigarrow(g)\triangleq\{\langle x,\ y\rangle\mid y\in g(\{x\})\}$, the *function abstraction* $\mathcal{S}[\![\mapsto[C_1,C_2]]\!]\triangleq\langle\wp(C_1\mapsto C_2),\ \subseteq\rangle \underset{\alpha^\mapsto}{\overset{\gamma^\mapsto}{\leftrightarrows}}\langle\wp(C_1)\mapsto\wp(C_2),\ \dot\subseteq\rangle$ with $\alpha^\mapsto(P)\triangleq\lambda X\bullet\{f(x)\mid f\in P\wedge x\in X\}$, $\gamma^\mapsto(g)\triangleq\{f\in C_1\mapsto C_2\mid\forall X\in\wp(C_1):\forall x\in X:f(x)\in g(X)\}$,

# Abstract Interpretation

```
float unsafe(float x) {
  if (x==0.0)
    return 0.0;
  else
    return 1.0 / x;
}
```

- Create an abstract domain
- Define semantics on abstract domain
- "Interpret" your program to see what values pop out
  - For example, we wish to ensure we never divide by zero.

# Abstract Interpretation

```
float unsafe(float x) {
  if (x==0.0)
    return 0.0;
  else
    return 1.0 / x;
}
```

- Abstract Domain:

| nnz / | = | Handled? |
|---|---|---|
| 0 | $\pm\infty$ | ✓ |
| nnz | **flt** | ✗ |
| NaN | NaN | ✗ |
| $\pm\infty$ | 0 | ✗ |
| flt | flt | |

# Abstract Interpretation

```
#include <math.h>
float mostlysafe(float x) {
  if (x==0. || isnan(x) || isinf(x))
    return 0.;
  else
    return 1. / x;
}
```

# Abstract Interpretation

```
#include <math.h>
float mostlysafe(float x) {
  if (x==0. || isnan(x) || isinf(x))
    return 0.;
  else
    return 1. / x;
}
```

- Abstract Domain:

| nnz / | = | Handled? |
|---|---|---|
| 0 | $\pm\infty$ | ✓ |
| nnz | **flt** | ✗ |
| NaN | NaN | ✓ |
| $\pm\infty$ | 0 | ✓ |
| flt | flt | |

# Abstract Interpretation

HPCL

```c
#include <math.h>
float reallysafe(float x) {
  // Cast to int without changing bits
  unsigned long c = *(unsigned long*) &x;
  if (isnan(x) || isinf(x) ||
      (0x80000000 <= c && c <= 0x80200000) ||
      (0x00000000 <= c && c <= 0x00200000))
    return 0.;
  else
    return 1. / x;
}
```

- Abstract Domain:

| nnz / | = | Handled? |
|---|---|---|
| 0 | $\pm\infty$ | ✓ |
| nnz | **flt** | ✓ |
| NaN | NaN | ✓ |
| $\pm\infty$ | 0 | ✓ |
| flt | flt | |

# Design Decisions

HPCL

1. To what degree of confidence must the software be guaranteed?
2. What tools can be used to accomplish 1?
3. How much time can a human spend on 2? How much time can a computer spend on 2?

# Degree of Confidence

- Formal methods is not a silver bullet
  - More like a flu vaccine
- Security vulnerability found in the WPA2 WiFi standard which had been proven secure
- Vulnerability related to temporal property which WPA2 did not specify

"Beware of bugs in the above code; I have only proved it correct, not tried it" — Donald Knuth

# Properties You Might Care About

HPCL

- Basic safety guarantees
  - Will `ipow` work for all integers?
  - Can be handled (mostly) automatically by SMT solvers

```
int ipow(int x, int n) {
    if (n==0) return 1;
    return x * ipow(x,n-1);
}
int main() {
  int a,b,c,n;
  n = 3; c = 0;
  while (1) {
    c++;
    for (a = 1; a < c; a++) {
      for (b = 1; b < c; b++) {
        if (ipow(a,n)+ipow(b,n)==ipow(c,n))
          return 0;
} } } }
```

# Properties You Might Care About

- Does this program test all combinations of a, b, and c?
  - Requires loop invariants and annotating code
  - Proves partial correctness; if the code terminates, then we get the right answer

```
int ipow(int x, int n) {
    if (n==0) return 1;
    return x * ipow(x,n-1);
}
int main() {
  int a,b,c,n;
  n = 3; c = 0;
  while (1) {
    c++;
    for (a = 1; a < c; a++) {
      for (b = 1; b < c; b++) {
        if (ipow(a,n)+ipow(b,n)==ipow(c,n))
          return 0;
} } } }
```

# Properties You Might Care About

- Does this program terminate?
  - Until 1995, no one knew

```c
int ipow(int x, int n) {
    if (n==0) return 1;
    return x * ipow(x,n-1);
}
int main() {
  int a,b,c,n;
  n = 3; c = 0;
  while (1) {
    c++;
    for (a = 1; a < c; a++) {
      for (b = 1; b < c; b++) {
        if (ipow(a,n)+ipow(b,n)==ipow(c,n))
          return 0;
} } } }
```

# Tools for Formal Methods

**Nadia Polikarpova**
@polikarn

Follow

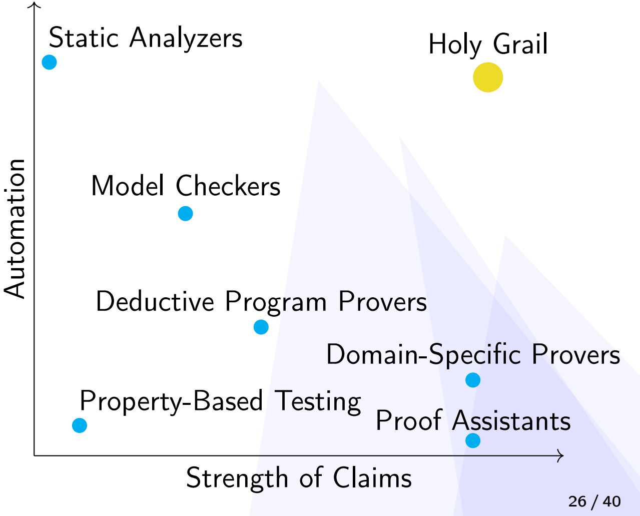
software engineer: linear is fast, quadratic is slow
complexity theorist: P is fast, NP-hard is slow
verification researcher: decidable is fast, undecidable is slow

8:47 AM - 13 Dec 2018

**378** Retweets **1,441** Likes

# Tools for Formal Methods

- Tradeoff between automation and strength of what can be verified
- What prevents us from reaching the Holy Grail is the fundamental limit of computation
- *Rice's Theorem*

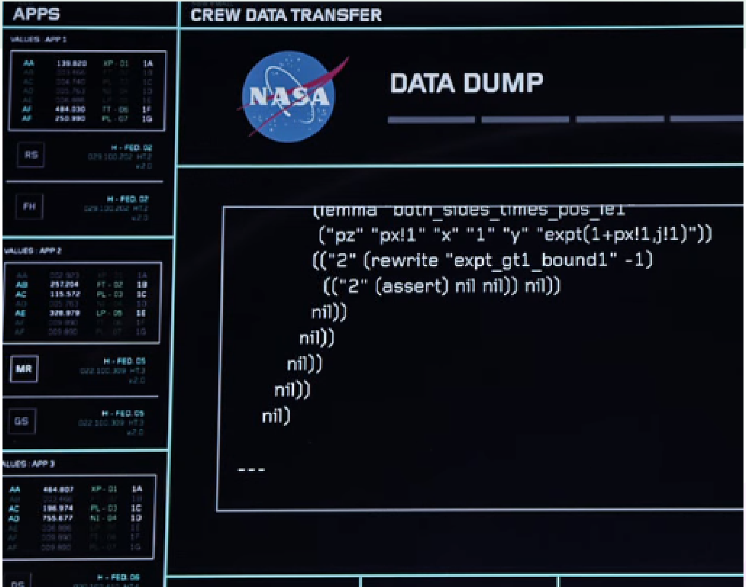


# Notable Proof Assistants

| System | de Bruijn criterion | Tactics |
|---|---|---|
| **Coq** | ✓ | ✓ |
| HOL | ✓ | ✗ |
| ACL2 | ✗ | ✓ |
| PVS | ✓ | ✓ |
| Twelf | ✗ | ✗ |
| $F^\star$ | ✓ | ✓ |
| NuPRL | ✗ | ✓ |
| Agda | ✗ | ✗ |
| Lean | ✓ | ✓ |

- Coq - Proved the four color theorem
- HOL/Isabelle - Intel and Cambridge
- ACL2 - UTexas', also (probably) AMD
- PVS - Used at NASA
- $F^\star$, Lean - Microsoft
- NuPRL - Cornell; oldest listed here; 1984

# The Martian


APPS
CREW DATA TRANSFER
NASA
DATA DUMP
(lemma "both_sides_times_pos_le1"
("pz" "px!1" "x" "1" "y" "expt(1+px!1,j!1)"))
(("2" (rewrite "expt_gt1_bound1" -1)
(("2" (assert) nil nil)) nil))
nil))
nil))
nil))
nil))
nil)
---

# Verifying IRs

HPCL

- WebAssembly - Machine-checkable semantics
- Vellvm - LLVM in Coq
- BAP - Binary analysis platform

# Some Quirks of Floating Point Arithmetic

HPCL

- Tension between “floats as bits” and “floats as reals”
- Having the same bit pattern is neither necessary nor sufficient for two IEEE 754 floats to be considered equal
  - $0 \ldots 0$ and $10 \ldots 0$ represent $-0$ and $+0$ which *are* equal
  - NaNs are all not equal to each other

# Formalizing Real Numbers

- Coq library Flocq
- Doesn't concern itself with bit-level representations
- No maximum exponents!

# Why is This Challenging?

HPCL

- Increasing vector widths means more complex CFG, difficult to automatically (and hand) vectorize
- Code transformations may assume associativity (too aggressive)
- Code transformations may match scalar code (too conservative)

# Conclusion

ℵ

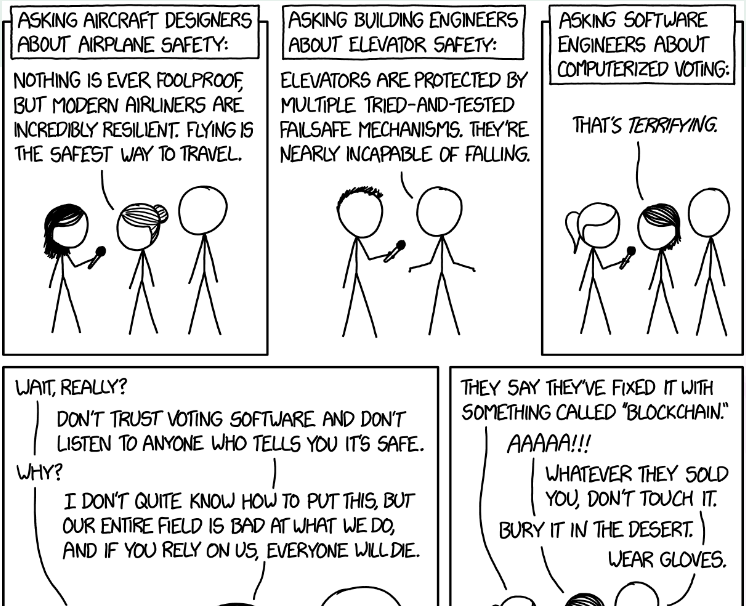
- Formal Methods are two things:
  1. Formal Specification - creating unambiguous, computer-checkable description of the program
  2. Model Checking - proving (or refuting) the program obeys this spec.
- Theoretically many of FM algorithms are undecidable or at least $\mathcal{NP}$-Hard
  - In practice these scale surprisingly well
  - i.e. programs terminate or scale beyond $n = 35$

# In Defense of FM

- A complaint of formal methods is it's too expensive
- FM isn't all-or-nothing
  - Consider a type checker
  - Static analyzers can be part of software engineering workflow
  - e.g. CI, red squiggly lines in Eclipse

Ariane 5 - $500 million software bug: incorrectly converted 64-bit float to 16-bit integer

# XKCD Tax


ASKING AIRCRAFT DESIGNERS ABOUT AIRPLANE SAFETY:
NOTHING IS EVER FOOLPROOF, BUT MODERN AIRLINERS ARE INCREDIBLY RESILIENT. FLYING IS THE SAFEST WAY TO TRAVEL.
ASKING BUILDING ENGINEERS ABOUT ELEVATOR SAFETY:
ELEVATORS ARE PROTECTED BY MULTIPLE TRIED-AND-TESTED FAILSAFE MECHANISMS. THEY'RE NEARLY INCAPABLE OF FALLING.
ASKING SOFTWARE ENGINEERS ABOUT COMPUTERIZED VOTING:
THAT'S TERRIFYING.
WAIT, REALLY?
DON'T TRUST VOTING SOFTWARE AND DON'T LISTEN TO ANYONE WHO TELLS YOU IT'S SAFE.
WHY?
I DON'T QUITE KNOW HOW TO PUT THIS, BUT OUR ENTIRE FIELD IS BAD AT WHAT WE DO, AND IF YOU RELY ON US, EVERYONE WILL DIE.
THEY SAY THEY'VE FIXED IT WITH SOMETHING CALLED "BLOCKCHAIN."
AAAAA!!!
WHATEVER THEY SOLD YOU, DON'T TOUCH IT.
BURY IT IN THE DESERT.
WEAR GLOVES.

# Future Work

HPCL

- Formalizing IEEE-754, MIL-STD-1750A, and Posits in a unified way
  - Look at floats both as bits and real numbers
  - Existing packages do only one (Flocq: $\mathbb{R}$, SMT Solvers: bits)
- Existing scientific codes only flip one switch - to use `float` or `double`; can we do better?
- SIMD transformations are rigid, GCC `-O3` is not rigid enough
- Quameleon - multi-ISA binary analysis at Sandia