Microsoft Research

Each year Microsoft Research hosts hundreds of influential speakers from around the world including leading scientists, renowned experts in technology, book authors, and leading academics, and makes videos of these lectures freely available. 2016 © Microsoft Corporation. All rights reserved.
Continuously Verified Cryptography

Mike Dodds

PNW PLSE Workshop, May 2018
Who Galois are

Research and development lab of ~70 people

Locations

Portland, OR
Arlington, VA
Dayton, OH
What we do

PL research meets real-world applications

Programming languages, analysis, verification, security, cryptography

Our tools

Symbolic execution
Model checking
Interactive theorem provers
Functional programming (esp. Haskell)
Who I am

Principal scientist at Galois (August ’17); former U.York, UK professor

Main areas: Separation logic, concurrency, relaxed memory

Recent papers:

  Verified compiler optimizations (ESOP’18)

  Concurrency verification tools (CAV’17)

  Linearizability proofs (ESOP’17)
Continuous Verification

Correctness of core components in Amazon’s s2n TLS library.
2018: static tools used daily by

**Google**
- 1 billion LOC code base
- 20,000 code reviews per day
- Approach: AST patterns
- Tool: **ErrorProne**

**Facebook**
- Static analysis of every diff
- Millions of LOC
- Approach: separation logic, abstract interp.
- Tool: **Infer**

**Amazon**
- Proofs of correctness
- Core infrastructure
- (millions reqs. per sec.)
- Tool: **SAW**
Encouraging code quality

- Revision Control (git, hg)
- Testing (junit, Travis)
- Peer Review (GitHub Pull Requests)
- Static Analysis / verification
2018: static tools used daily by

**Google**
- 1 billion LOC code base
- 20,000 code reviews per day
- Approach: AST patterns
- Tool: ErrorProne

**Facebook**
- Static analysis of every diff
- Millions of LOC
- Approach: separation logic, abstract interp.
- Tool: Infer

**Amazon**
- Proofs of correctness
- Core infrastructure
- (millions reqs. per sec.)
- Tool: SAW
Encouraging code quality

- Revision Control (git, hg)
- Testing (junit, Travis)
- Peer Review (GitHub Pull Requests)
- Static Analysis / verification
2018: static tools used daily by

Google
- 1 billion LOC code base
- 20,000 code reviews per day
- Approach: AST patterns
- Tool: ErrorProne

Facebook
- Static analysis of every diff
- Millions of LOC
- Approach: separation logic, abstract interp.
- Tool: Infer

Amazon
- Proofs of correctness
- Core infrastructure
- (millions reqs. per sec.)
- Tool: SAW
2018: static tools used daily by

Google
- 1 billion LOC code base
- 20,000 code reviews per day
- Approach: AST patterns
- Tool: ErrorProne

Facebook
- Static analysis of every diff
- Millions of LOC
- Approach: separation logic, abstract interp.
- Tool: Infer

Amazon
- Proofs of correctness
- Core infrastructure
- (millions reqs. per sec.)
- Tool: SAW

| galois |
TLS: Transport Layer Security

TLS (newer version of SSL) provides us most of the

   Confidentiality
   Data-Integrity
   Authentication

guarantees that we enjoy on the internet today.
Amazon s2n: A TLS Implementation

• Inspired by TLS vulnerabilities discovered by researchers in other implementations.

• Written with security and performance as primary goals.

• Drops some arguably insecure/less secure features.
  • Result: Much smaller, clearer, more auditable code.
  • OpenSSL TLS is 70k lines of C code.
  • s2n is only 6k.

• Used in production at Amazon (and therefore used by pretty much everyone on the internet)
HMAC: A Component of TLS

- keyed-Hash Message Authentication Code
- Provides a signature for a message that confirms:
  - Authenticity: the message was signed by the expected sender
  - Integrity: the message has not been modified

\[
\text{HMAC}(K, m) = H((K \oplus \text{opad}) \ || \ H((K \oplus \text{ipad}) \ || \ m))
\]

- Code is still complex
  - 521 lines of C code
HMAC Specification

HMAC(K, m) =  
H((K ⊕ opad) || H((K ⊕ ipad) || m))

C HMAC

Fast
Concise
Interoperable
Easily auditable

Goal: bridge this gap
Summary of Approach

1. Write the formal specification.
2. Write some "scaffolding" to bridge the gap between specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.
Summary of Approach

1. Write the formal specification.
2. Write some “scaffolding” to bridge the gap between specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.
Summary of Approach

$$\text{HMAC}(K, m) = H((K \oplus \text{opad}) \| H((K \oplus \text{ipad}) \| m))$$

Step 1: Capture this specification in a formal language (we used Cryptol - https://cryptol.net/)

hmac k message =

$$H((k \uparrow \text{opad}) \# H((k \uparrow \text{ipad}) \# \text{message}))$$
Summary of Approach

\[ \text{HMAC}(K, m) = H((K \oplus \text{opad}) \| H((K \oplus \text{ipad}) \| m)) \]

Step 1: Capture this specification in a formal language (we used Cryptol - https://cryptol.net/)

\[
\text{hmac } k \text{ message } = \\
H((k ^ \text{opad}) \# H((k ^ \text{ipad}) \# \text{message}))
\]
HMAC Specification

hmac k message =
\[ H((k \uparrow \text{opad}) \# H((k \uparrow \text{ipad}) \# \text{message})) \]

C HMAC

Goal: bridge this gap

- Fast
- Concise
- Easily auditable
- Interoperable
Summary of Approach

1. Write the formal specification.
2. Write some “scaffolding” to bridge the gap between specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.
Bridging the gap

Solution: Layers of Abstraction

- High-level Cryptol Code
  - Incorporates: s2n Data structures, s2n API
  - Proof
- Lower-level Cryptol Code
  - Proof
- Production s2n code
  - Omits: Pointers and memory allocation, Low-level performance optimization

Increasingly implementation-specific

Unmodified code from s2n repo
Summary of Approach

1. Write the formal specification.
2. Write some “scaffolding” to bridge the gap between Specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.
Bridging the gap

Solution: Layers of Abstraction

Automatically Constructed by SAW (Software Analysis Workbench) via translation to SMT and application of constraint solvers
Summary of Approach

1. Write the formal specification.
2. Write some “scaffolding” to bridge the gap between Specification and C code.
3. Apply automated tools.
4. Integrate into development environment.

About 2 months of effort.
Continuous Integration

- Proofs run automatically on code changes
  - Proof failure is a build failure
- Proof is independent of exact C code, depends only on:
  - Interfaces (arguments and struct layouts)
  - Function call structure
- Proof is easily adapted:
  - Function body changes $\rightarrow$ likely no proof changes
  - Interface changes $\rightarrow$ similarly-sized proof changes
  - Call structure changes $\rightarrow$ tiny proof changes
Continuous Integration

- Proofs run automatically on code changes
  - Proof failure is a build failure
- Proof is independent of exact C code, depends only on:
  - Interfaces (arguments and struct layouts)
  - Function call structure
- Proof is easily adapted:
  - Function body changes $\rightarrow$ likely no proof changes
  - Interface changes $\rightarrow$ similarly-sized proof changes
  - Call structure changes $\rightarrow$ tiny proof changes
Verified HMAC pipeline

Changes Infrequently  Increasing Automation  Changes Frequently

Work of Beringer et al.
- Security Property
  - Indistinguishability from random
  - Proved in Coq

High-Level Specification
- Coq HMAC Specification
  - Proved in Coq

High-Level Specification
- Combination of Coq (Manual) and Cryptol (Automatic)

Low-Level Specification
- Incremental API

This Paper
- Implementation
  - s2n C code
    - ... version 2 ...
    - ... version 3 ...

Proved with SAW (mostly automatic)
Other s2n Work

- Also verified (see paper at CAV’18):
  - DRBG: Deterministic Random Bit Generator: The main source of cryptographic randomness
  - TLS Handshake protocol (state machine)
- Working on the rest
  - Correctness and security of parsing
  - Correct handling of keys in memory
  - Correct session management
  - Other crypto primitives, e.g. HKDF
Conclusions

- For crypto / authentication / access control:
  Behavioral bugs are security bugs
- Verification is feasible for real-world cryptography
- Proof can be integrated into development workflow to prevent errors and give continuous verification
- Continuous analysis / verification is now being applied in industry
Helena: A web automation language for end users

Sarah Chasins  
University of California, Berkeley

Rastislav Bodik  
University of Washington
care about (web) data
today

care about (web) data
tomorrow
care about (web) data today

coders  non-coders

care about (web) data tomorrow

coders  non-coders
What web data collection tools do we have?

Scrapy
BeautifulSoup
Nokogiri

coders
non-coders
What web data collection tools do we have?

- Tools that require users to reverse engineer target webpages.
- Coders

- AJAX
- DOM

- Non-coders
What web data collection tools do we have?

Tools that require users to reverse engineer target webpages:
- AJAX
- JS
- DOM

- hire a human to copy & paste
- hire a coder to use one of these

Coders

Non-coders
What web data collection tools do we have?

- hire a human to copy & paste
- hire a coder to use one of these
- **Helena**
  
  WEB AUTOMATION FOR END USERS

Tools that require users to reverse engineer target web pages:

**DOM**

**AJAX**

**JS**

**coders**

**non-coders**
vapnik

Professor of Columbia, Fellow of NEC Labs America, Verified email at nec-labs.com
machine learning  statistics  computer science

The Nature of Statistical Learning Theory
V Vapnik
Data mining and knowledge discovery

Statistical Learning Theory
VN Vapnik
Wiley-Interscience

Support-vector networks
C Cortes, V Vapnik
Machine learning 20 (3), 273-297
Is this thing any good?

- Measured time to complete first task with each language
- Participants were programmers

lower is better

![Bar chart showing task completion times for two tasks: Task 1 and Task 2. The chart compares times for traditional web language times and Helena times. Lower times are better.](image-url)
Is this thing any good?

- Measured time to complete first task with each language
- Participants were programmers

Programmers 8x faster with Helena!
Helena
PBD tool

Web
Browser

demonstration

program

today asking: how can user make this program more robust or more performant?

Helena
editor

skip blocks

program'
How is rent changing across Seattle neighborhoods?
Kept losing network connection
Kept losing network connection

New listings have pushed the last three listings from p1 onto p2

wasting 10+ hours scraping duplicates!
Problem Statement

(1) **Failures**: What happens when the network fails, the server fails, the computer fails? When we lose our session with the server and have to start over?

(2) **Data changes**: What happens when the server gives the client pages produced from different (potentially conflicting) reads of the underlying data store? 

*Not client side problems → scraping script can’t prevent them, must handle them*
Problem Statement

(3) **Longitudinal scraping**: What if we want to run this over time?

(4) **Just slow**: What if this thing just takes 40 hours to run and we want to run it every 24 hours? Can we parallelize it?

(5) **Running into captchas**: What if running it from a single IP is running into rate-limiting? Can we distribute it?
Solution

on the surface, seem like very different problems

failures

data changes
Solution

“Just don’t redo the same work you’ve already done!”
Solution

"Just don’t redo the same work you’ve already done!"

But what’s the ‘same’ work? After all, data changes...
Solution

on the surface, seem like very different problems

failures

data changes

“Just don’t redo the same work you’ve already done!”

But what’s the ‘same’ work? After all, data changes...

Our answer: the skip block! User can
- tell us what makes objects the same
- associate the code that operates on an object
Solution

on the surface, seem like very different problems

failures                      data changes

"Just don’t redo the same work you’ve already done!"

But what’s the ‘same’ work? After all, data changes...

Our answer: the skip block! User can
- tell us what makes objects the same
- associate the code that operates on an object

- If object already committed (memoized), skip block; else, run block
- No reverse engineering! Reasoning about output data
End-user performance edits

```javascript
for (aRow in p1.authors) {
    scrape aRow.author_name
    scrape aRow.author_institution
    p2 = click aRow.author_name
    for (pRow in p2.papers) {
        scrape pRow.title
        scrape pRow.citations
        output([aRow.author_name, pRow.title, pRow.citations])
    }
}
```

- text-ify-ed representation of the block language
- scrape stuff about the author, click the author
- for the author’s papers, scrape paper stuff
- add a row of output with the author and paper info
End-user performance edits

```python
for aRow in p1.authors{
    skipBlock(Author(aRow.author_name, aRow.author_institution)){
        scrape aRow.author_name
        scrape aRow.author_institution
        p2 = click aRow.author_name
        for pRow in p2.papers{
            scrape pRow.title
            scrape pRow.citations
            output([aRow.author_name, pRow.title, pRow.citations])
        }
    }
}
```
End-user performance edits

```python
for aRow in p1.authors:
    skipBlock(Author(aRow.author_name, aRow.author_institution))
    scrape aRow.author_name
    scrape aRow.author_institution
    p2 = click aRow.author_name
    for pRow in p2.papers:
        scrape pRow.title
        scrape pRow.citations
        output([aRow.author_name, pRow.title, pRow.citations])
```

**key attributes:** is the current author the same as another we’ve already seen?

**block:** the code that operates on the author object

if ever, **in any run**, script has committed an object with the same key attributes, skips the block
Let's solve all those disparate problems

(1) **Extrinsic failures**: Objects that already made it into the commit log won’t be rescraped when we restart after a server failure.

(2) **Data changes**: When frequently updated data sources result in repeated objects/wasted work, we’ll skip them.

(3) **Longitudinal scraping**: Skip blocks automatically incrementalize; objects scraped in a prior run won’t be scraped again.

(4) **Just slow**: Each skip block represents an independent subtask. Split them across parallel worker processes.

(5) **Running into captchas**: Now split those worker processes across machines.
Fun design questions

- How should we handle nested skip blocks?
- Don’t *always* want to skip if object seen before. When should we re-scrape, how should user communicate it?
- How should independent subtasks be split, communicated across parallel or distributed workers?
Data Change

Speedup on the *first run*

Measured full execution time of:
- Script with skip blocks
- Script without skip blocks

Chart shows speedup from using skip blocks

higher is better
Parallelization
with lock-based skip block assignment

Measured full execution time (on a single machine) of:
● Script with one worker process
● Script with 2, 4, 6, or 8 worker processes
Chart shows speedup from using more processes

higher is better
## User Study

<table>
<thead>
<tr>
<th></th>
<th>Coders</th>
<th>Non-Coders</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time to learn about skip blocks</td>
<td>2 minutes</td>
<td>7 minutes</td>
</tr>
<tr>
<td>Time to add one new skip block</td>
<td>52 seconds</td>
<td>61 seconds</td>
</tr>
</tbody>
</table>
Unified handling of apparently disparate challenges with a single language construct.

By keeping reasoning at the level of target output data, made skip blocks usable by non-programmers.

contact: schasins@cs.berkeley.edu

github.com/schasins/helena
Sinking Point

Bill Zorn
Dan Grossman
Zach Tatlock
IEEE 754 floating-point

Fast, portable, completely specified

Each operation is correctly rounded

Sometimes behavior is similar to real numbers, sometimes not

Can be difficult to reason about
Examples

$ python
Python 3.6.3 |Anaconda, Inc.| (default, Oct 13 2017, 12:02:49)
[GCC 7.2.0] on linux
>>> import math
>>> math.pi + 1e16 - 1e16
Examples

```
$ python
Python 3.6.3 |Anaconda, Inc.| (default, Oct 13 2017, 12:02:49)
[GCC 7.2.0] on linux
>>> import math
>>> math.pi + 1e16 - 1e16
4.0
```
\[ ax^2 + bx + c = 0 \quad \Rightarrow \quad x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \]
\[ ax^2 + bx + c = 0 \]

\[ x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>(x) (IEEE 754 double)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1</td>
<td>2</td>
<td>3</td>
<td>-1.6333997346592444</td>
</tr>
</tbody>
</table>
\[ ax^2 + bx + c = 0 \quad x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x (IEEE 754 double)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1</td>
<td>2</td>
<td>3</td>
<td>-1.6333997346592444</td>
</tr>
<tr>
<td>.001</td>
<td>2</td>
<td>3</td>
<td>-1.5011266906707066</td>
</tr>
<tr>
<td>1e-9</td>
<td>2</td>
<td>3</td>
<td>-1.500000013088254</td>
</tr>
</tbody>
</table>
\[ ax^2 + bx + c = 0 \]

\[ x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x (IEEE 754 double)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1</td>
<td>2</td>
<td>3</td>
<td>-1.6333997346592444</td>
</tr>
<tr>
<td>.001</td>
<td>2</td>
<td>3</td>
<td>-1.5011266906707066</td>
</tr>
<tr>
<td>1e-9</td>
<td>2</td>
<td>3</td>
<td>-1.5000000013088254</td>
</tr>
<tr>
<td>1e-15</td>
<td>2</td>
<td>3</td>
<td>-1.5543122344752189</td>
</tr>
</tbody>
</table>
\[ ax^2 + bx + c = 0 \]

\[ x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x (IEEE 754 double)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1</td>
<td>2</td>
<td>3</td>
<td>-1.6333997346592444</td>
</tr>
<tr>
<td>.001</td>
<td>2</td>
<td>3</td>
<td>-1.5011266906707066</td>
</tr>
<tr>
<td>1e-9</td>
<td>2</td>
<td>3</td>
<td>-1.500000013088254</td>
</tr>
<tr>
<td>1e-15</td>
<td>2</td>
<td>3</td>
<td>-1.5543122344752189</td>
</tr>
<tr>
<td>1e-16</td>
<td>2</td>
<td>3</td>
<td>-2.2204460492503131</td>
</tr>
</tbody>
</table>
\[ a x^2 + bx + c = 0 \]

\[ x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x (IEEE 754 double)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1</td>
<td>2</td>
<td>3</td>
<td>-1.6333997346592444</td>
</tr>
<tr>
<td>.001</td>
<td>2</td>
<td>3</td>
<td>-1.5011266906707066</td>
</tr>
<tr>
<td>1e-9</td>
<td>2</td>
<td>3</td>
<td>-1.500000013088254</td>
</tr>
<tr>
<td>1e-15</td>
<td>2</td>
<td>3</td>
<td>-1.5543122344752189</td>
</tr>
<tr>
<td>1e-16</td>
<td>2</td>
<td>3</td>
<td>-2.2204460492503131</td>
</tr>
<tr>
<td>1e-17</td>
<td>2</td>
<td>3</td>
<td>0</td>
</tr>
</tbody>
</table>
\[ x = \frac{-2 + \sqrt{2^2 - 4 \cdot a \cdot 3}}{2 \cdot a} \]
\(0.1x^2 + 2x + 3\)
\[ 0.001x^2 + 2x + 3 \]
<table>
<thead>
<tr>
<th>x</th>
<th>IEEE 754 double</th>
<th>Exact</th>
</tr>
</thead>
<tbody>
<tr>
<td>15</td>
<td>0.92785633341392471</td>
<td>0.92785633341392471</td>
</tr>
<tr>
<td>x</td>
<td>IEEE 754 double</td>
<td>Exact</td>
</tr>
<tr>
<td>------</td>
<td>--------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>15</td>
<td>0.92785633341392471</td>
<td>0.92785633341392471</td>
</tr>
<tr>
<td>25.3</td>
<td>-0.77866261130416781</td>
<td>-0.77866260949297472</td>
</tr>
<tr>
<td>x</td>
<td>( \text{IEEE 754 double} )</td>
<td>( \text{Exact} )</td>
</tr>
<tr>
<td>------</td>
<td>----------------------------------</td>
<td>-------------------</td>
</tr>
<tr>
<td>15</td>
<td>0.92785633341392471</td>
<td>0.92785633341392471</td>
</tr>
<tr>
<td>25.3</td>
<td>\textbf{-0.77866261130416781}</td>
<td>-0.77866260949297472</td>
</tr>
<tr>
<td>60</td>
<td>-0.83064921763725463</td>
<td>-0.83064921763725463</td>
</tr>
<tr>
<td>x</td>
<td>IEEE 754 double</td>
<td>Exact</td>
</tr>
<tr>
<td>--------</td>
<td>-------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td>15</td>
<td>0.92785633341392471</td>
<td>0.92785633341392471</td>
</tr>
<tr>
<td>25.3</td>
<td>-0.77866261130416781</td>
<td>-0.77866260949297472</td>
</tr>
<tr>
<td>60</td>
<td>-0.83064921763725463</td>
<td>-0.83064921763725463</td>
</tr>
<tr>
<td>60.01</td>
<td>-0.94644010175472126</td>
<td>0.83258728783816349</td>
</tr>
<tr>
<td>x</td>
<td>IEEE 754 double</td>
<td>Exact</td>
</tr>
<tr>
<td>------</td>
<td>------------------------</td>
<td>----------------------------</td>
</tr>
<tr>
<td>15</td>
<td>0.92785633341392471</td>
<td>0.92785633341392471</td>
</tr>
<tr>
<td>25.3</td>
<td>-0.77866261130416781</td>
<td>-0.77866260949297472</td>
</tr>
<tr>
<td>60</td>
<td>-0.83064921763725463</td>
<td>-0.83064921763725463</td>
</tr>
<tr>
<td>60.01</td>
<td>-0.94644010175472126</td>
<td>0.83258728783816349</td>
</tr>
<tr>
<td>60.125</td>
<td>0.27925526776125142</td>
<td>-0.75293133122316314</td>
</tr>
</tbody>
</table>
None of these examples are surprising

Floating-point is fully specified

Just doing what the specification says

All results have the same amount of precision

Up to the programmer to determine if the bits are meaningful
Sinking Point

Dynamically reduce precision, rather than returning meaningless bits

Not any more accurate than IEEE 754 floating-point

But the precision you do get corresponds to behavior of reals

Modest overhead: a few extra bits, a few extra bitwise operations
Examples

$ python
Python 3.6.3 |Anaconda, Inc.| (default, Oct 13 2017, 12:02:49)
[GCC 7.2.0] on linux
>>> import math
>>> math.pi + 1e16 - 1e16
4.0
Examples

$ python
Python 3.6.3 |Anaconda, Inc.| (default, Oct 13 2017, 12:02:49)
[GCC 7.2.0] on linux
>>> import math
>>> math.pi + 1e16 - 1e16
4.0000000000000000
Examples

$ python
Python 3.6.3 |Anaconda, Inc.| (default, Oct 13 2017, 12:02:49)
[GCC 7.2.0] on linux
>>> import math
>>> math.pi + 1e16 - 1e16
4.0000000000000000
>>> import sink
>>> sink('pi') + sink('1e16') - sink('1e16')
4~
\begin{align*}
ax^2 + bx + c &= 0 \\
x &= \frac{-b + \sqrt{b^2 - 4ac}}{2a}
\end{align*}

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>(x) (IEEE 754 double)</th>
<th>(x) (Exact)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1</td>
<td>2</td>
<td>3</td>
<td>-1.6333997346592444</td>
<td>-1.6333997346592446</td>
</tr>
<tr>
<td>.001</td>
<td>2</td>
<td>3</td>
<td>-1.5011266906707066</td>
<td>-1.5011266906707219</td>
</tr>
<tr>
<td>1e-9</td>
<td>2</td>
<td>3</td>
<td>-1.500000013088254</td>
<td>-1.50000011250001</td>
</tr>
<tr>
<td>1e-15</td>
<td>2</td>
<td>3</td>
<td>-1.5543122344752189</td>
<td>-1.5000000000000011</td>
</tr>
<tr>
<td>1e-16</td>
<td>2</td>
<td>3</td>
<td>-2.2204460492503131</td>
<td>-1.5000000000000002</td>
</tr>
<tr>
<td>1e-17</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>-1.5000000000000000</td>
</tr>
<tr>
<td>.1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>.1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
\[ ax^2 + bx + c = 0 \]

\[ x = \frac{-b + \sqrt{b^2 - 4ac}}{2a} \]

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
<th>c</th>
<th>x (IEEE 754 double)</th>
<th>x (Exact)</th>
<th>x (Sinking Point)</th>
</tr>
</thead>
<tbody>
<tr>
<td>.1</td>
<td>2</td>
<td>3</td>
<td>-1.6333997346592444</td>
<td>-1.6333997346592446</td>
<td>-1.6333997346592444~</td>
</tr>
<tr>
<td>.001</td>
<td>2</td>
<td>3</td>
<td>-1.5011266906707066</td>
<td>-1.5011266906707219</td>
<td>-1.50112669067073~</td>
</tr>
<tr>
<td>1e-9</td>
<td>2</td>
<td>3</td>
<td>-1.5000000013088254</td>
<td>-1.5000000011250001</td>
<td>-1.5000000~</td>
</tr>
<tr>
<td>1e-15</td>
<td>2</td>
<td>3</td>
<td>-1.5543122344752189</td>
<td>-1.50000000000000011</td>
<td>-1.5~</td>
</tr>
<tr>
<td>1e-16</td>
<td>2</td>
<td>3</td>
<td>-2.2204460492503131</td>
<td>-1.5000000000000002</td>
<td>-2~</td>
</tr>
<tr>
<td>1e-17</td>
<td>2</td>
<td>3</td>
<td>0</td>
<td>-1.5000000000000000</td>
<td>0~@4</td>
</tr>
<tr>
<td>.1</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>x</td>
<td>IEEE 754 double</td>
<td>Exact</td>
<td>Sinking Point</td>
<td></td>
<td></td>
</tr>
<tr>
<td>------</td>
<td>-----------------</td>
<td>----------------</td>
<td>---------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>0.92785633341392471</td>
<td>0.92785633341392471</td>
<td>0.92785633341392471~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>25.3</td>
<td>-0.77866261130416781</td>
<td>-0.77866260949297472</td>
<td>-0.7786626108~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>-0.83064921763725463</td>
<td>-0.83064921763725463</td>
<td>-0.83064921763725463~</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60.01</td>
<td>-0.94644010175472126</td>
<td>0.83258728783816349</td>
<td>0~@0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>60.125</td>
<td>0.27925526776125142</td>
<td>-0.75293133122316314</td>
<td>0~@0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Floating-point rounding
Floating-point rounding

\[ 5.25 = 21 \times 2^{-2} \]

- significand
- exponent
Floating-point rounding

\[ 5.25 = 21 \times 2^{-2} \]

- **p**: number of bits in significand
- **n**: exponent - 1
Floating-point rounding

\[ 5.25 = 21 \times 2^{-2} \]

Significand: \( p := \text{number of bits in significand} \)

Exponent: \( n := \text{exponent - 1} \)

Rounding can be specified in terms of \( p_{\text{max}} \) and \( n_{\text{min}} \).
Floating-point rounding

\[ 5.25 = 21 \times 2^{-2} \]

\( p := \text{number of bits in significand} \)

\( n := \text{exponent} - 1 \)

Rounding can be specified in terms of \( p_{\text{max}} \) and \( n_{\text{min}} \)

- For IEEE 754 doubles, \( p_{\text{max}} = 53 \) and \( n_{\text{min}} = -1075 \)
Floating-point rounding

\[ 5.25 = 21 \times 2^{-2} \]

\[ p := \text{number of bits in significand} \]
\[ n := \text{exponent} - 1 \]

Rounding can be specified in terms of \( p_{\text{max}} \) and \( n_{\text{min}} \):

- For IEEE 754 doubles, \( p_{\text{max}} = 53 \) and \( n_{\text{min}} = -1075 \)
- Sinking Point determines \( p_{\text{max}} \) and \( n_{\text{min}} \) per operation
Sinking Point rounding rules

- Addition and Subtraction
  - $n_{\text{min}} = \text{maximum } n \text{ of inexact inputs}$
Sinking Point rounding rules

- Addition and Subtraction
  - $n_{\text{min}} = \text{maximum } n \text{ of inexact inputs}$
  - $p_{\text{max}} = p_{\text{max}} \text{ of format}$
Sinking Point rounding rules

- Addition and Subtraction
  - $n_{\text{min}} = \text{maximum } n \text{ of inexact inputs}$
  - $p_{\text{max}} = p_{\text{max}} \text{ of format}$

- Multiplication and Division
  - $n_{\text{min}} = n_{\text{min}} \text{ of format}$
  - $p_{\text{max}} = \text{minimum } p \text{ of inexact inputs}$
Sinking Point rounding rules

- Addition and Subtraction
  - $n_{\text{min}} = \text{maximum } n \text{ of inexact inputs}$
  - $p_{\text{max}} = p_{\text{max}} \text{ of format}$

- Multiplication and Division
  - $n_{\text{min}} = n_{\text{min}} \text{ of format}$
  - $p_{\text{max}} = \text{minimum } p \text{ of inexact inputs}$

Powers and roots are like multiplication.
More interesting functions

- Floor
  - Result is exact, unless $n > -1$
More interesting functions

- Floor
  - Result is exact, unless \( n > -1 \)
- Fmod
  - \( n_{\text{min}} = n \) of dividend
  - \( p_{\text{max}} = p \) of divisor
More interesting functions

- Floor
  - Result is exact, unless $n > -1$

- Fmod
  - $n_{\text{min}} = n$ of dividend
  - $p_{\text{max}} = p$ of divisor

- Sin
  - $n_{\text{min}} = n_{\text{min}}$ of format
  - $p_{\text{max}} = \text{exponent of } \pi - n$ of argument
Computing correctly with finitely many bits

Sinking point gives us confidence that results have meaningful precision

Fast to compute, relative to IEEE 754 floating-point

Only a few extra bits

Still an approximation, not a sound guarantee
Titanic
Guaranteed to float correctly

FPBench
Common standards for the floating-point research community
Computing correctly with finitely many bits

Sinking point gives us confidence that results have meaningful precision

Fast to compute, relative to IEEE 754 floating-point

Only a few extra bits

Still an approximation, not a sound guarantee
More interesting functions

- Floor
  - Result is exact, unless \( n > -1 \)

- Fmod
  - \( n_{\text{min}} = n \) of dividend
  - \( p_{\text{max}} = p \) of divisor

- Sin
  - \( n_{\text{min}} = n_{\text{min}} \) of format
  - \( p_{\text{max}} = \) exponent of \( \pi - n \) of argument
Computing correctly with finitely many bits

Sinking point gives us confidence that results have meaningful precision

Fast to compute, relative to IEEE 754 floating-point

Only a few extra bits

Still an approximation, not a sound guarantee
Titanic

Guaranteed to float correctly

FPBench

Common standards for the floating-point research community
Verified Extraction with Native Types

Stuart Pernsteiner, Eric Mullen, James R. Wilcox, Zachary Tatlock, Dan Grossman
Verified Coq Extraction

*.v  Extract  *.ml  ocamlc  *.exe
Verified Coq Extraction

```
* .v  Extract  * .ml  ocamlc  * .exe
```

Verified Coq Extraction

\[ \text{Coq} \rightarrow \text{CertiCoq} \rightarrow \ast . v \rightarrow \ast . c \rightarrow \text{CompCert} \rightarrow \ast . \text{exe} \]
Compiling SHA-256

SHA256.v \text{ Manual Proof} \equiv \text{ sha256.c} \xrightarrow{\text{CompCert}} \text{ sha256sum}

Compiling SHA-256

SHA256.v → øEuf → sha256.c → CompCert → sha256sum

Inefficient data representation

```
$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident
```
Inefficient data representation

```bash
$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident

$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
0:03.90 elapsed
2951208k max resident
```
Inefficient data representation

```bash
$ echo hello | time sha256sum
5891b5b5 22d5df086d0ff0b110fbd9d2...
0:00.00 elapsed

$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
0:03.90 elapsed
2951208k max resident
```
Inefficient data representation

```
$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident
```

```
$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
0:03.00 elapsed
```
We can do better

Definition SHA_256 : list word → list word.
We can do better

Definition SHA_256 : list word → list word.

Definition word := \{ x : Z | 0 ≤ x < 2^{32} \}
Native Types

- Map types to custom data representations
  - word $\rightarrow$ int
Native Types

- Map types to custom data representations
  - word $\rightarrow$ int

- Map functions to custom C-level implementations
  - $(x + y) \mod 2^{32} \rightarrow x + y$
Native Types

- Map types to custom data representations
  - $\text{word} \rightarrow \text{int}$

- Map functions to custom C-level implementations
  - $(x + y) \mod 2^{32} \rightarrow x + y$

- Maintain existing correctness guarantees
Native Types

- Map types to custom data representations
  - word → int
- Map functions to custom C-level implementations
  - \((x + y) \mod 2^{32} \rightarrow x + y\)
- Maintain existing correctness guarantees
- Extensible to new types and functions
Native Types

- Map types to custom data representations
  - word $\rightarrow$ int

- Map functions to custom C-level implementations
  - $(x + y) \mod 2^{32} \rightarrow x + y$

- Maintain existing correctness guarantees
- Extensible to new types and functions
  - Updating proofs is a lot of work!
Extensible Native Types

Inductive type :=
| Tnat | Tbool | Tlist (t : type) | ... |
| Tint?
Extensible Native Types

\[
\text{Inductive } \text{type} := \\
\mid \text{Tnat} \mid \text{Tbool} \mid \text{Tlist } (t : \text{type}) \mid \ldots \\
\mid \text{Tnative } (nt : \text{native_type_defn})
\]
Extensible Native Types

Inductive type :=
  | Tnat | Tbool | Tlist (t : type) | ...
  | Tnative (nt : native_type_defn)

Definition nty_int : native_type_defn := {}

|{}|.
Extensible Native Types

Inductive type :=
| Tnat | Tbool | Tlist (t : type) | ...
| Tnative (nt : native_type_defn)

Definition nty_int : native_type_defn := {
  high_rep := word;
}.
Extensible Native Types

Inductive type :=
| Tnat | Tbool | Tlist (t : type) | ...
| Tnative (nt : native_type_defn)

Definition nty_int : native_type_defn := {|
  high_rep := word;
  low_rep := int;
|}. 
**Extensible Native Types**

**Inductive** type :=

| Tnat | Tbool | Tlist (t : type) | ... |
| Tnative (nt : native_type_defn) |

**Definition** nty_int : native_type_defn := {

  high_rep := word;
  low_rep := int;
  (* lemmas... *)

|}.
Extensible Native Types

Inductive type ::= 
| Tnat | Tbool | Tlist (t : type) | ... |
| Tnative (nt : native_type_defn) |

Definition nty_int : native_type_defn ::= {
  high_rep := word;
  low_rep := int;
  (* lemmas... *)
}.

Definition nty_double : native_type_defn ::= ...
Definition nty_char : native_type_defn ::= ...
Definition nty_string : native_type_defn ::= ...
Extensible Native Operations

\[
\text{Inductive expr :=} \\
| \text{Evar (...) | Ecall (a b : expr)} \\
| \text{Eadd?}
\]
Extensible Native Operations

\[
\text{Inductive } \text{expr} \ := \\
| \text{Evar (\ldots)} | \text{Ecall (a b : expr)} \\
| \text{Enative (no : native_oper_defn)} \\
(\text{args : list expr})
\]
Extensible Native Operations

Inductive expr ::=
  | Evar (...) | Ecall (a b : expr)
  | Enative (no : native_oper_defn)
    (args : list expr)

native_oper_defn must support the semantics of every IR!
Extensible Native Operations

\[
\text{Inductive}\;\text{expr} \;::= \\
| \text{Evar}\;(...) | \text{Ecall}\;(a\;b:\;\text{expr}) \\
| \text{Enative}\;(\text{no} : \text{native\_oper\_defn}) \\
\hspace{1cm} (\text{args} : \text{list}\;\text{expr})
\]

native\_oper\_defn\;must\;support\;the\;semantics\;of\;every\;IR!

- Every IR semantics needs a way to step Enative exprs
Extensible Native Operations

Inductive expr :=
  | Evar (...)  | Ecall (a b : expr)
  | Enative (no : native_oper_defn)
     (args : list expr)

native_oper_defn must support the semantics of every IR!

• Every IR semantics needs a way to step Enative exprs
• We chose to have one implementation for each value representation
Design Tradeoffs

- Simplifies native oper. definitions
  - Less stuff in the record
Design Tradeoffs

- Simplifies native oper. definitions
  - Less stuff in the record
- Simplifies compiler correctness proofs
  - Native oper cases are trivial when value representations don't change
Design Tradeoffs

- Simplifies native oper. definitions
  - Less stuff in the record
- Simplifies compiler correctness proofs
  - Native oper cases are trivial when value representations don't change
- Native operations can't call functions
  - Calls work differently in different IRs
  - Can't implement recursive eliminators as native opers
Results

- It works and is extensible
  - int w/ literals, arith ops, comparisons, int_to_nat
  - double w/ literals, arith ops, comparisons, to/from_int
  - double_array w/ alloc, get, set
  - Avg. 65 lines per type; 210 lines per operation
Results

- It works and is extensible
  - int w/ literals, arith ops, comparisons, int_to_nat
  - double w/ literals, arith ops, comparisons, to/from_int
  - double_array w/ alloc, get, set
  - Avg. 65 lines per type; 210 lines per operation

- Performance is much improved...
Inefficient data representation

```sh
$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident
```

```sh
$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
0:03.90 elapsed
2951208k max resident
```
Efficient data representation

$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident

$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
0:03.90 elapsed
2951208k max resident
Efficient data representation

```
$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident
```

```
$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
0:03.90 elapsed
2951208k max resident
```

50x less time!

```
$ echo hello | time ./sha256_int
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.07 elapsed
33760k max resident
```
Efficient data representation

$ echo hello | time sha256sum
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.00 elapsed
1844k max resident

$ echo hello | time ./sha256_N
5891b5b522d5df086d0ff0b110fbd9d2...
0:03.90 elapsed
2951208k max resident

50x less time!

$ echo hello | time ./sha256_int
5891b5b522d5df086d0ff0b110fbd9d2...
0:00.07 elapsed
33760k max resident

87x less memory!