|galois|

|galois|

Verification is Engineering

Joey Dodds
- and the Galois Proof Engineering Team

What we do

SAW and Cryptol

- Full code correctness for C and x86 (and combined codebases)
- Compositional symbolic execution approach

Crux

- Non-compositional symbolic testing
- Small C library to drive
- No additional language knowledge required
- Crux-mir for rust

Whatever else fits the problem

Galois Verification 2 years ago

~400 LOC verified

1 full time engineer (average, split across people)

Relatively simple algorithms

Tools that couldn’t handle “big programs”

Galois Verification today

~20k LOC

~6 engineers (consistent, split across people)

Verification of PQ algorithms and handwritten assembly

Lessons for proof writing

Understand customer needs

Need to move beyond single monolithic implementations

Proof-writing requires engineering discipline

Program proofs need thorough explanation

Delivery matters

Understand Customer needs

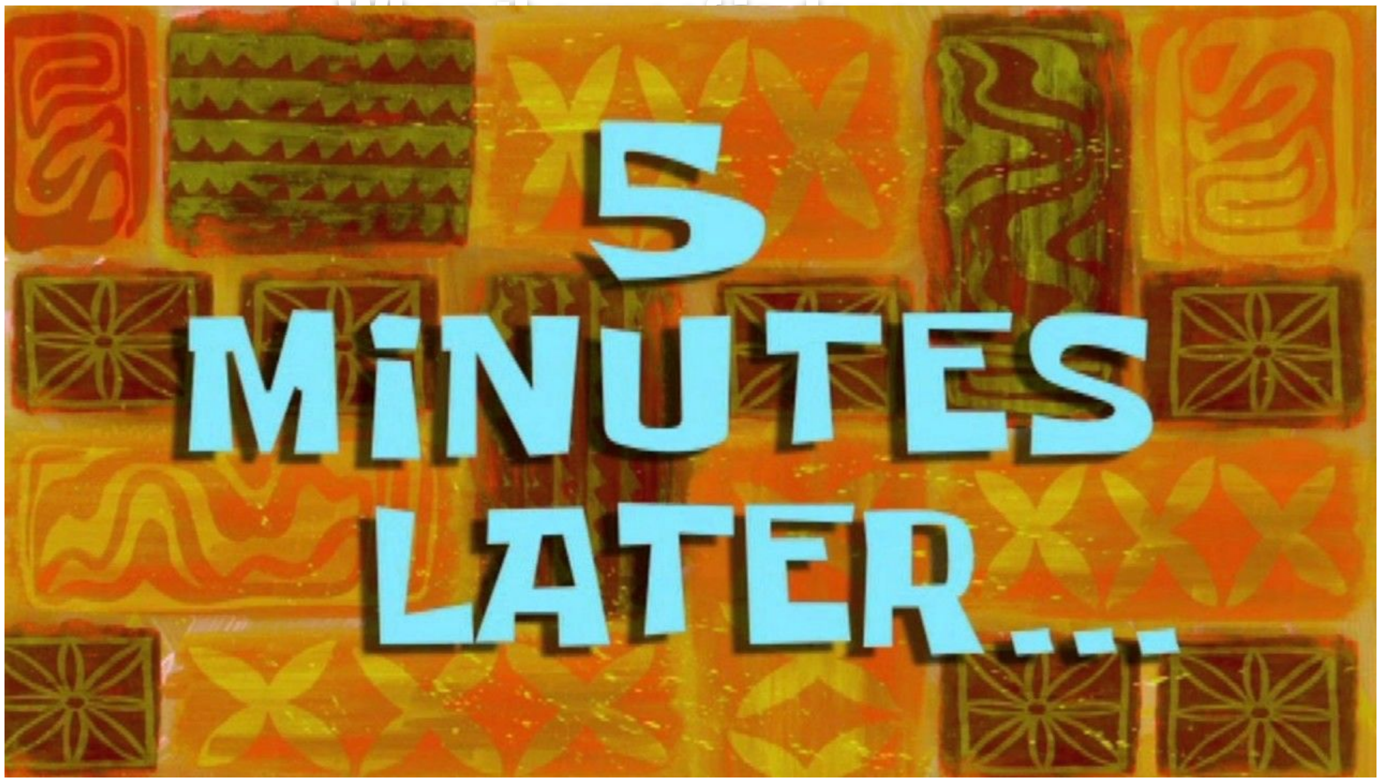
Use verification only when it’s appropriate

Agree on a plan for when proofs get hard

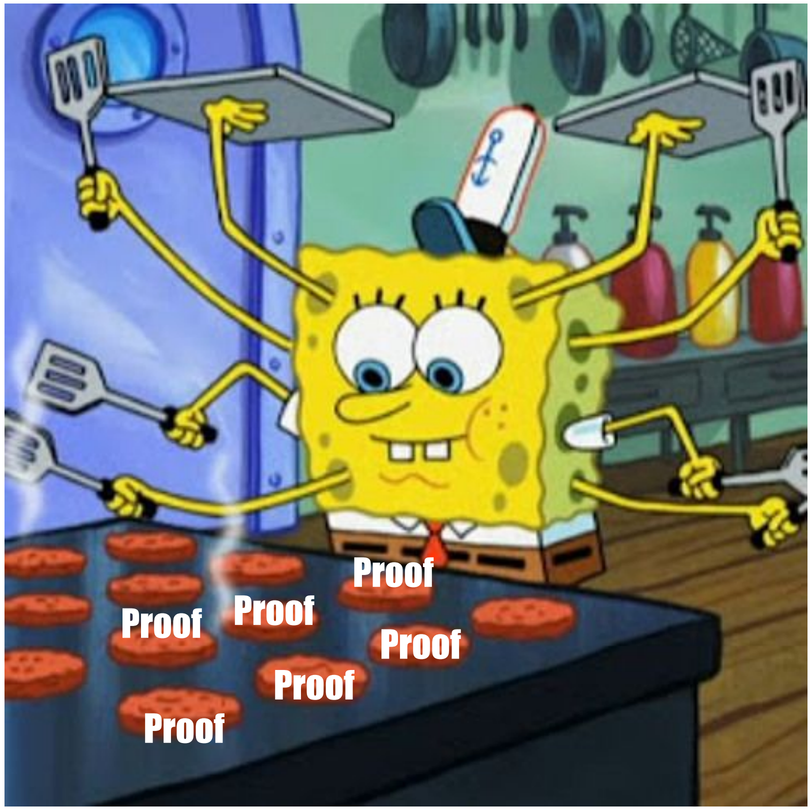
Agree on a plan for when bugs are found

Monolithic Implementations

Monolithic Implementations



Monolithic Implementations



The most useful code

Works everywhere

Makes full use of its environments

Is updated constantly

Glen Arrowsmith
@garrows

npm install...

The most useful proof

Works everywhere

Makes full use of its environments

Is updated constantly*

* The most useful proof updates itself, but that might not be realistic

How do we accomplish this

The “dream” is a compositional proof package manager.

In the meantime…

- Make tools general and multi-language
- Be willing to go black box
- Work your way up to “full proof”

Proof writing requires discipline

Put risk first

Pair program

Code review

Document

Think about the skills required for proof writing...

The first 90 percent of the code accounts for the first 90 percent of the development time. The remaining 10 percent of the code accounts for the other 90 percent of the development time.[1][2]

— Tom Cargill, Bell Labs

Start with the last 10 percent of the code.

— Joey Dodds, This HCSS talk

Academia

Let’s be careful how we use our status as academics

We need to balance being impressive with being accessible

Many of the most successful applications of formal methods have not been called formal methods

Oops

Last talk I said: It’s great that you can distill proofs down to pass fail for developers and that they can find that useful

Now: It would be great if you could distill proofs down to pass fail for developers and that they could find that useful. But instead we need to work to make the proofs easier to understand

Explaining Proofs

Like programs, proofs are *nuanced*

Customers should always ask:

- What are the limitations?
- How do I maintain this?
- How do I know this is correct?
- How do I share this value with *my* customers?
- What assumptions does the proof make about the environment?

What do we do?

Communicate with customers directly

Give talks like this one

Set expectations from the start

Raise issues early

Help customers teach themselves

What are we working on?

Communicate with the public

Document everything

Develop training

Focus on “proofs that don’t work”

Delivery process

Proof walkthroughs

Code reviews and merges

List limitations in the repository

Thank you!