Proofs in the Wild

What's done today?
What's close?
What's far?

Mike Dodds - Galois, Inc. - December 2024

miked@galois.com - https://mikedodds.github.io

Context: I was an academic, then I wasn’t

2004 → 2017: *UK*

- York / Cambridge / York - PhD, postdoc, lecturer (~ associate professor)
- Logic design, automated reasoning, hardware models

2017 → now: *Portland OR*

- Galois Inc, PI / principal scientist
- Proofs for lots of different things: parsers, crypto(graphy), crypto(currency), protocols, cyber-physical systems …

Context: Galois does research for $$$

- A contract research shop / “R&D temp agency”
- 110 people, employee-owned
- Focus on security / reliability tech (PL, proof tech, static analysis)
- Clients: DARPA / DoD, some US Gov, some commercial

Proofs in the Wild

What's done today?

What's close?

What's far?

https://mikedodds.github.io

Galois does proof technologies

DARPA HACMS - formally verified drone controllers

- *Built on SeL4 verified microkernel & other proof technologies*
- *Cool demo: flew an unmanned helicopter, resisted red team attack*

AWS LibCrypto - https://github.com/awslabs/aws-lc-verification

- *Proofs for crypto code from OpenSSL*
- *(Candidate for) the most heavily used bit of verified code ever*

PROVERS - current multi-$m DARPA project

- *Aim: usability for testing and proof tools*
- *Verifying cyber-physical systems as built by DoD*

Proof tech in industry is small

Low-confidence guess: <1000 proof-focused industry engineers in US

Anec-data:

- Galois is big - 60-70 technical staff
- Conferences (CAV, PLDI …) - mostly academic, 100s of attendees
- Large % engineers have PhDs, small slow-growing talent pool

Some significant teams

- AWS (biggest / most public)
- Meta / Facebook
- Hardware companies - Intel most famously
- Crypto / blockchain
- High assurance things for US Gov

What proof tech does industry actually deploy?

1. Fully-automated program analysis
2. Model checking
3. ‘White glove’ verification / interactive theorem proving

1. Fully-automated program analysis

Eliminate a particular bug category at scale, e.g:

- Memory safety issues - Infer (Facebook / Meta)
- Cloud misconfigurations - Tiros / Zelkova (AWS)

Typical tools: custom analysis tools backed by logical solvers

Trade-offs:

- (+) Scales to millions of loc, can be used by non-specialist engineers
- (-) Unsound & incomplete - false positives and false negatives. V limited properties. Tools are heuristic and specialized to particular use-cases.

2. Model checking

A small / combinatorial *[thing]* must be correct, e.g:

- Hardware - arithmetic unit on a processor
- Cryptographic primitive - AES, SHA, ECDSA

Typical tools: encode the whole system as a logical formula, solve with SMT

Trade-offs:

- (+) Fully automated, exhaustive, less need for human-written internal specifications / overrides
- (-) Scalability VERY limited, only works for small things (or things that can be reduced to small models, such as protocols)

3. ‘White glove’ verification

A mid-scale complex self-contained *[thing]* must be correct, e.g:

- Operating system kernel - SeL4, CertiKOS, BlueRock
- Cryptographic library - HACL*, AWS LibCrypto

Typical tools: interactive theorem provers, eg. Coq, Lean, F*

Trade-offs:

- (+) Extremely high level of confidence; can prove very deep properties of the system; scales to true mathematical reasoning
- (-) Required deep human effort from experts; extremely expensive per line of code; changes to the verified system are equally expensive.

Barrier to increased adoption: cost/benefit

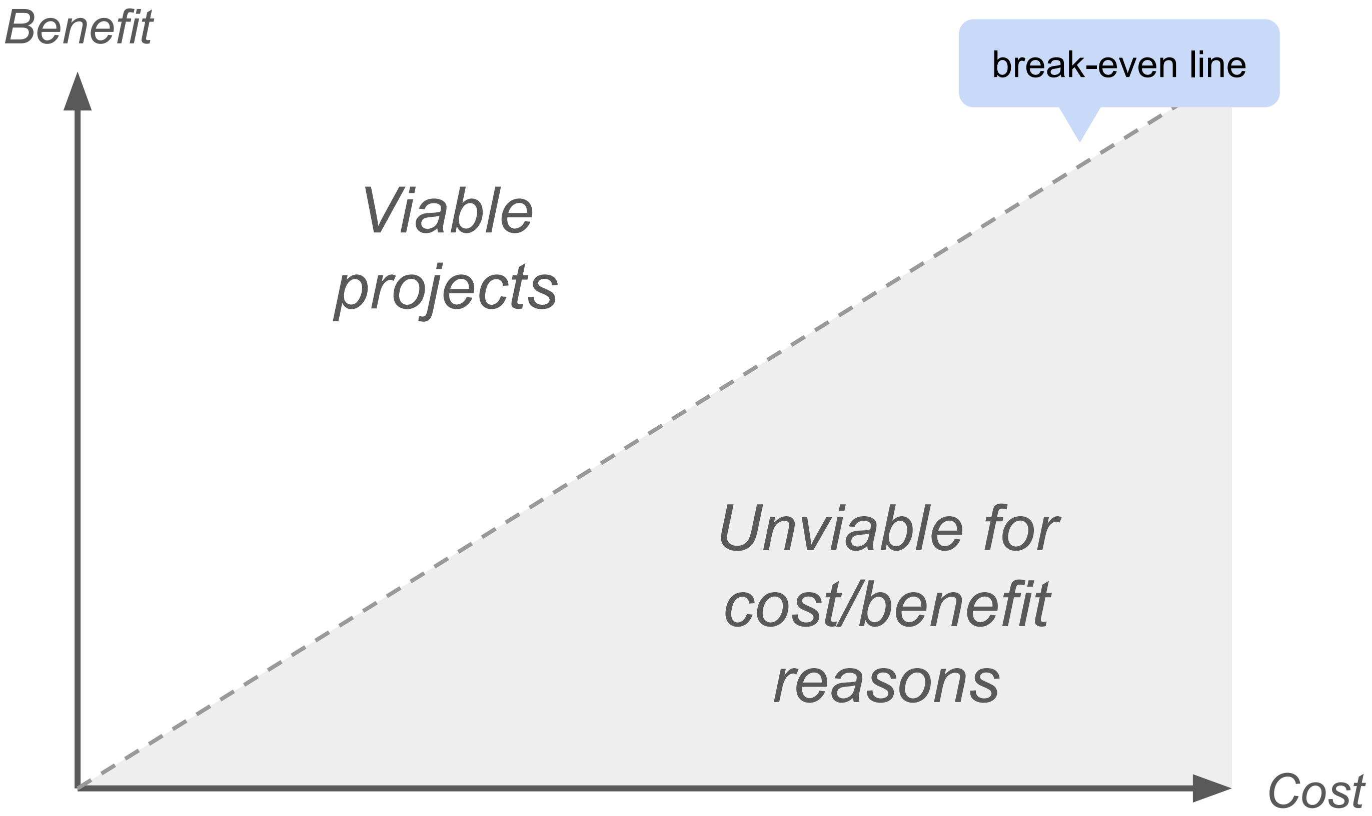
Writing proofs is very hard

- Proof scripts
- Internal function specifications / invariants
- Selection of abstractions

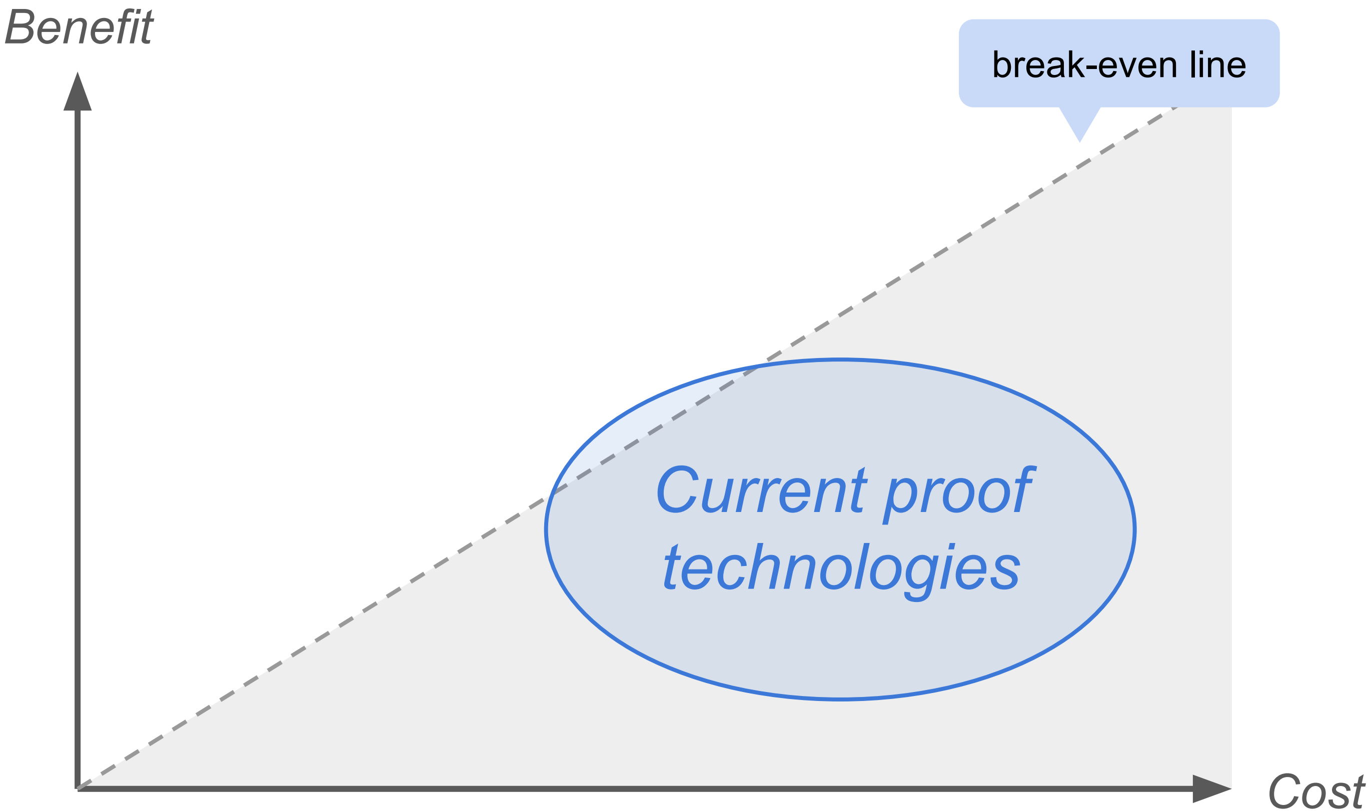
Writing specifications / world models is very hard

- Component-level specifications - pre/post conditions, reference code
- System models - language / compiler / hardware
- Environment models - threat models, user models, physics

Result: many possible projects don’t ‘pencil out’



Result: many possible projects don’t ‘pencil out’



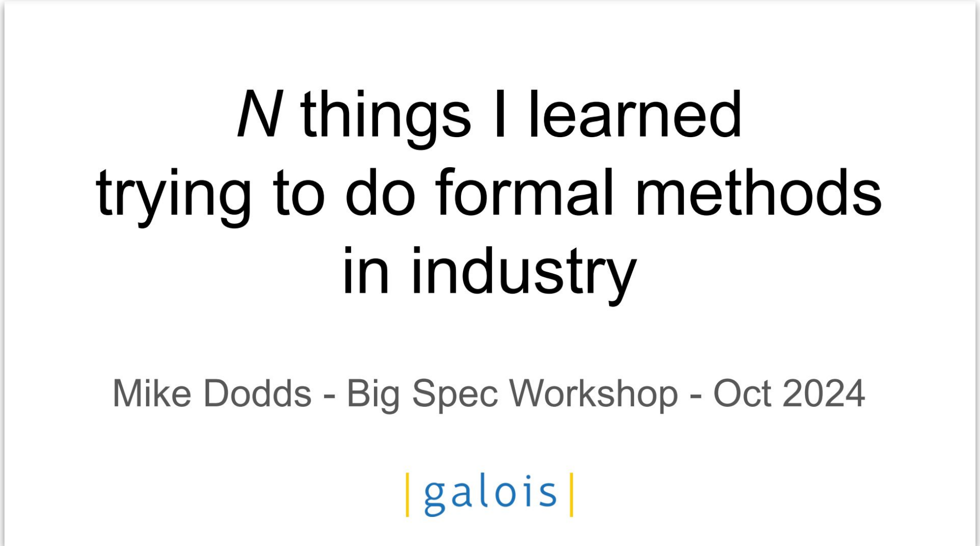
Success stories have solved this by careful scoping

Eg:

- Making properties very restricted
- Targeting very small systems
- Spending huge amounts of labor

Worth it for some very critical problems!

More on the cost/benefit landscape for proof tech:



N things I learned
trying to do formal methods
in industry

Mike Dodds - Big Spec Workshop - Oct 2024

galois

https://mikedodds.github.io/files/talks/2024-10-09-n-things-I-learned.pdf

Proofs in the Wild

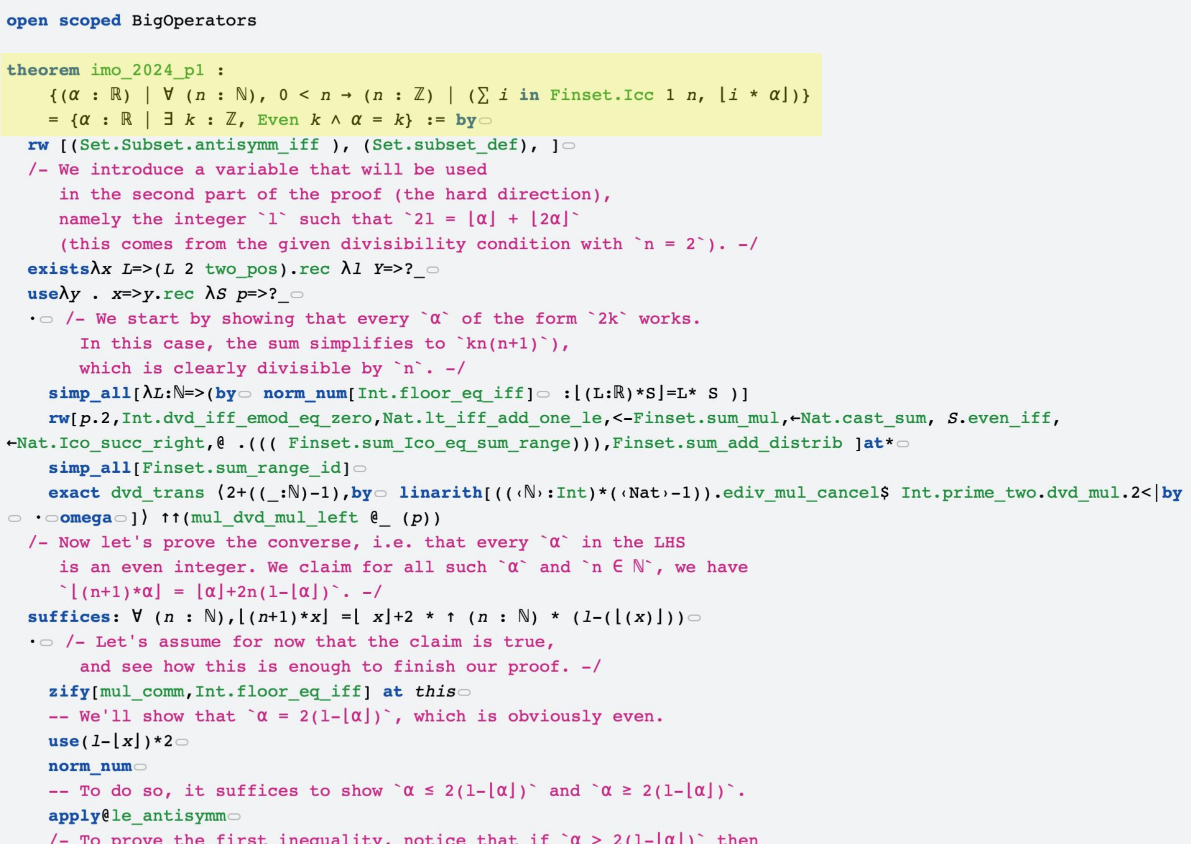
What's done today?

What's close?

What's far?

AI-driven proof

Writing proof scripts is arduous

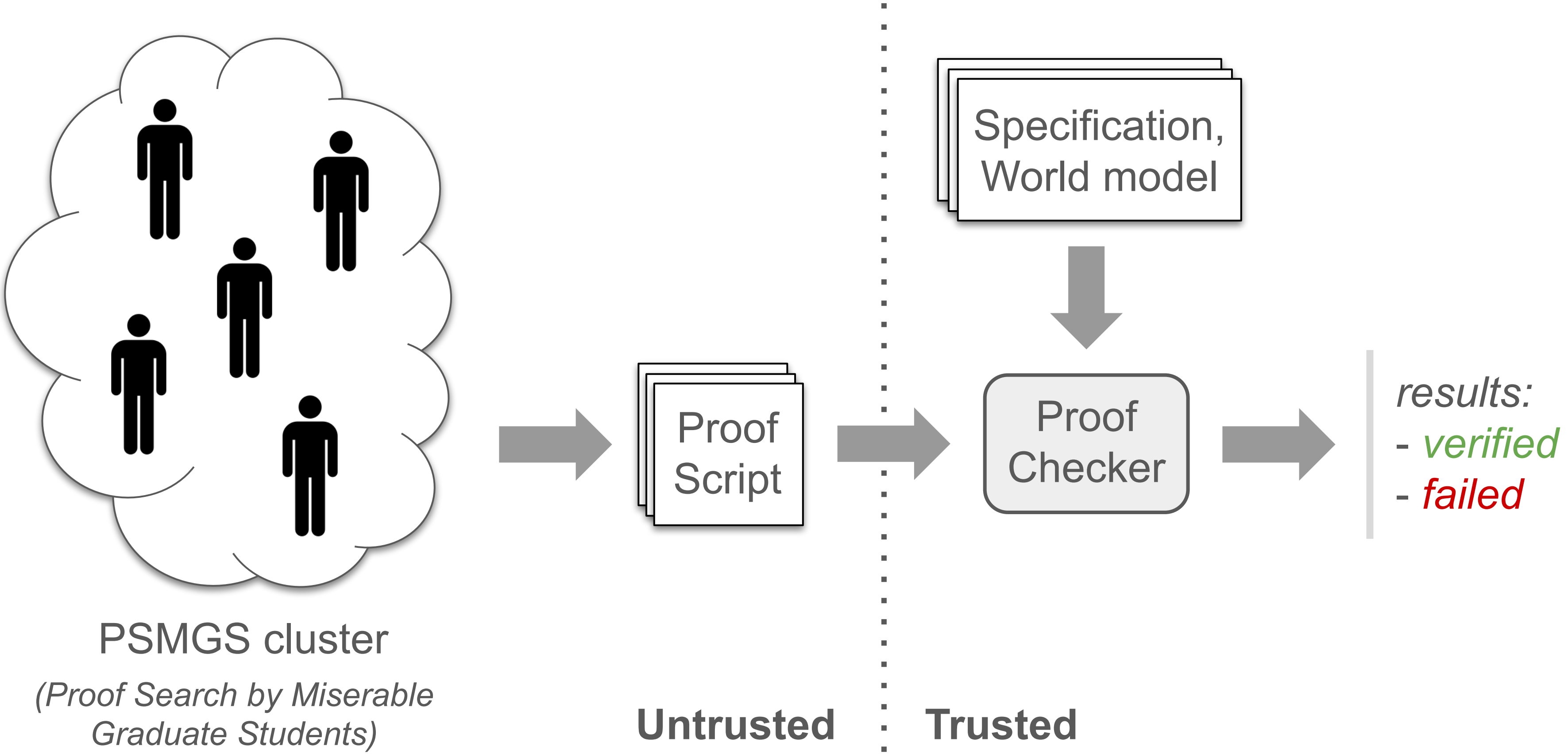


```
open scoped BigOperators

theorem imo_2024_p1 :
    {(α : ℝ) | ∀ (n : ℕ), 0 < n → (n : ℤ) ∣ (∑ i in Finset.Icc 1 n, ⌊i * α⌋)}
    = {α : ℝ | ∃ k : ℤ, Even k ∧ α = k} := by⬭
  rw [(Set.Subset.antisymm_iff ), (Set.subset_def), ]⬭
  /- We introduce a variable that will be used
    in the second part of the proof (the hard direction),
    namely the integer `l` such that `2l = ⌊α⌋ + ⌊2α⌋`
    (this comes from the given divisibility condition with `n = 2`). -/
  existsλx L=>(L 2 two_pos).rec λl Y=>?_⬭
  useλy . x=>y.rec λS p=>?_⬭
  ·⬭ /- We start by showing that every `α` of the form `2k` works.
      In this case, the sum simplifies to `kn(n+1)`),
      which is clearly divisible by `n`. -/
    simp_all[λL:ℕ=>(by⬭ norm_num[Int.floor_eq_iff]⬭ :⌊(L:ℝ)*S⌋=L* S )]
    rw[p.2,Int.dvd_iff_emod_eq_zero,Nat.lt_iff_add_one_le,<-Finset.sum_mul,←Nat.cast_sum, S.even_iff,
←Nat.Ico_succ_right,@ .((( Finset.sum_Ico_eq_sum_range))),Finset.sum_add_distrib ]at*⬭
    simp_all[Finset.sum_range_id]⬭
    exact dvd_trans ⟨2+((_:ℕ)-1),by⬭ linarith[((‹ℕ›:Int)*(‹Nat›-1)).ediv_mul_cancel$ Int.prime_two.dvd_mul.2<|by
⬭ ·⬭omega⬭]⟩ ↑↑(mul_dvd_mul_left @_ (p))
  /- Now let's prove the converse, i.e. that every `α` in the LHS
    is an even integer. We claim for all such `α` and `n ∈ ℕ`, we have
    `⌊(n+1)*α⌋ = ⌊α⌋+2n(l-⌊α⌋)`. -/
  suffices: ∀ (n : ℕ),⌊(n+1)*x⌋ =⌊ x⌋+2 * ↑ (n : ℕ) * (l-(⌊(x)⌋))⬭
  ·⬭ /- Let's assume for now that the claim is true,
      and see how this is enough to finish our proof. -/
    zify[mul_comm,Int.floor_eq_iff] at this⬭
    -- We'll show that `α = 2(l-⌊α⌋)`, which is obviously even.
    use(l-⌊x⌋)*2⬭
    norm_num⬭
    -- To do so, it suffices to show `α ≤ 2(l-⌊α⌋)` and `α ≥ 2(l-⌊α⌋)`.
    apply@le_antisymm⬭
    /- To prove the first inequality, notice that if `α > 2(l-⌊α⌋)` then
```

Google DeepMind, IMO 2024 Problem 1. https://storage.googleapis.com/deepmind-media/DeepMind.com/Blog/imo-2024-solutions/P1/index.html

Classic interactive theorem proving architecture



This is just a search process!

(untrusted)

Guess

- *Expensive*
- *Stochastic*
- *Hard to audit*

(trusted)

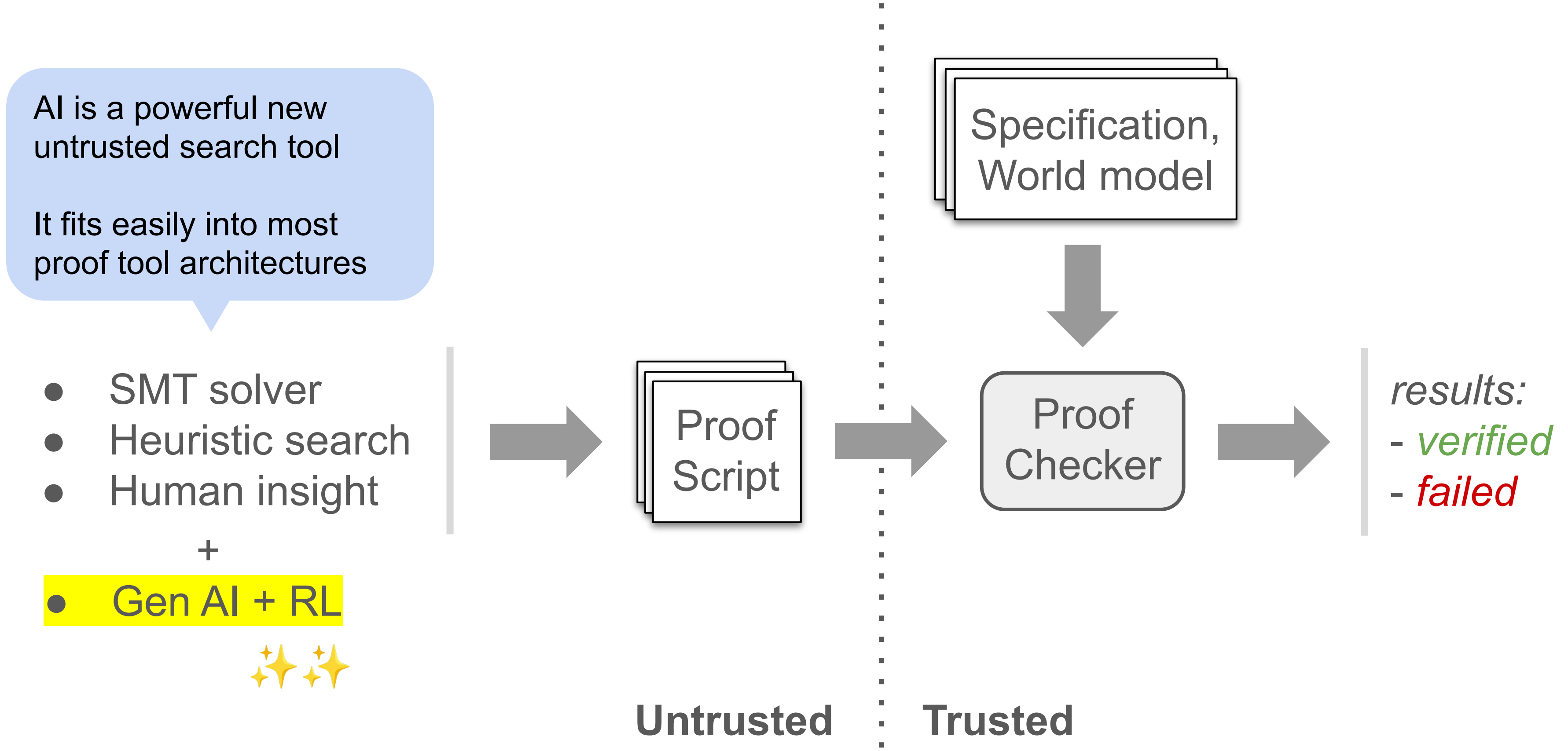
Check

- *Cheap*
- *Deterministic*
- *Easy to audit*

Many proof tech problems are just *search*

Guess		***Check***
Write a proof script	→	Check proof establishes the theorem
Add types to a program	→	Typecheck the program
Write program invariants	→	Check the program verification
Synthesize a program that matches a specification	→	Check the program matches the specification
[Heuristic generator]	→	**[Trusted checker]**

Almost all proof tools are ~structured this way



AI is a powerful new untrusted search tool

It fits easily into most proof tool architectures

- SMT solver
- Heuristic search
- Human insight

+

- Gen AI + RL

✨✨

Proof Script

Specification, World model

Proof Checker

results:

- *verified*
- *failed*

Untrusted

Trusted

Optimism: AI proofs get really cheap

Early indicators:

- AlphaProof IMO - automated proof search for v hard problems
- Towards Neural Synthesis for SMT-Assisted Proof-Oriented Programming, Microsoft Research https://arxiv.org/abs/2405.01787

Optimism: AI proofs improve rapidly

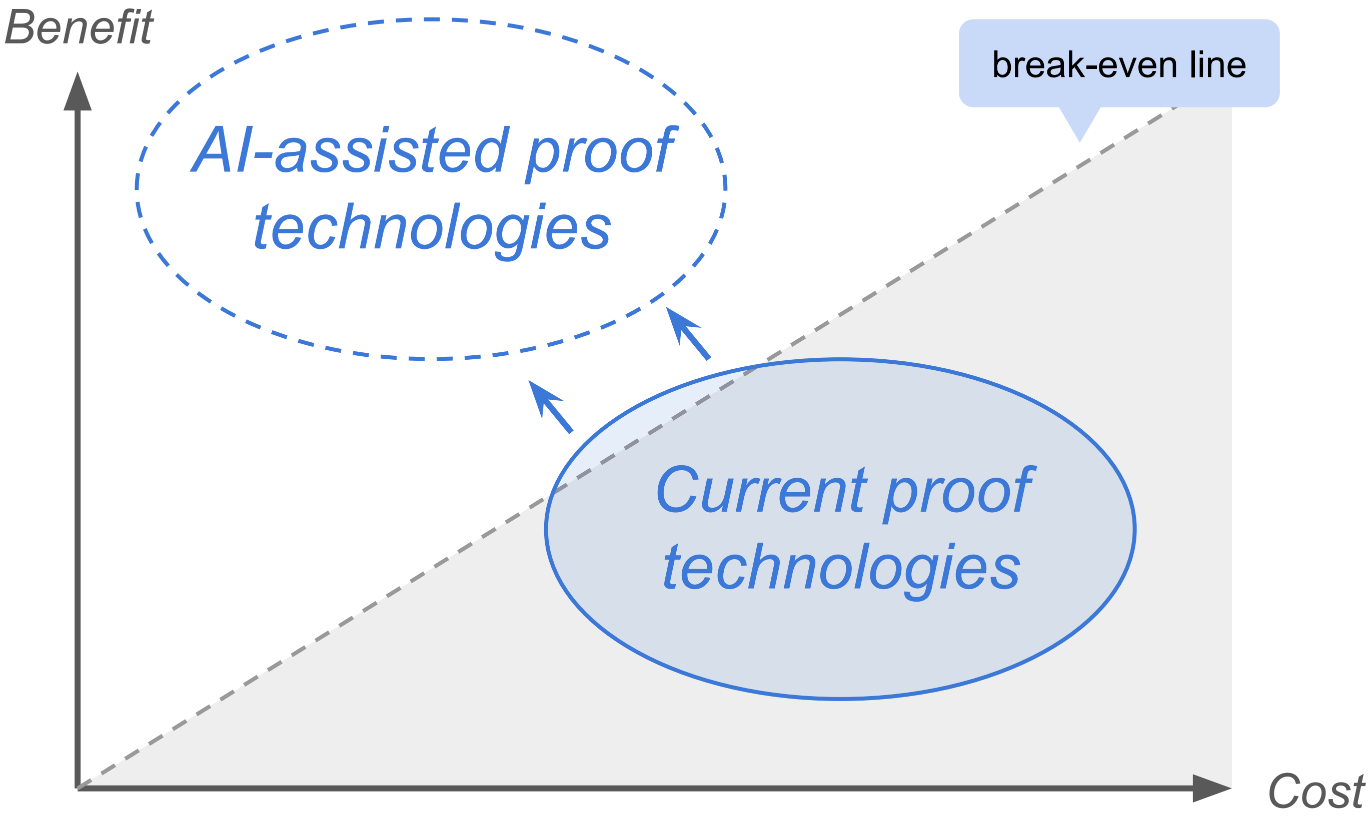
Synthetic data / RL

- Proof tools are a totally reliable oracle of correct / incorrect proofs
- Oracle + LLM + RL - seems promising for synthetic proof data generation

Current proof datasets are small

- Making proof easier should result in more proof data written by users
- Virtuous cycle - increased datasets result in improved capabilities

Optimism: many more proof technologies get useful



Optimism: impossible things become possible

Eg:

- Auto-coders that ‘certify their work’, generating proofs alongside diffs
- Transpile 10s of millions of lines of C with memory safety guarantees
- Insert proved-correct security boundaries into legacy systems
- Retrofit a Linux-scale operating system with proofs

These are *in a sense* currently possible, just much too expensive

Proofs in the Wild

What’s done today?

What’s close?

What’s far?

Specifications and world models

Current specification technologies

Mostly discrete, bounded, logical

- Logical formulas (+ various fancy extensions)
- State machines
- Domain specific languages

Eg. Cerberus: https://www.cl.cam.ac.uk/~pes20/cerberus/

- A highly accurate model of the C programming language
- Captured in a DSL called Lem which encodes logical states and updates
- Several person-years of iteration: building / testing / discussing

Formal specifications, ideally:

Mathematically clean

Stable over time

Agreed by the users of the system

Easy to reason about

Big successes ALL fit this ideal model

- Cryptographic algorithms
- Operating systems / hypervisors
- Compilers / programming languages
- Cloud services
- Hardware

The reality:

- These systems are *unusually easy to specify*
- Even slightly harder-to-specify things are very hard to deal with

Most real-world specifications are not…

Mathematically clean

Stable over time

Agreed by all users of the system

Easy to reason about

Real-world specifications are very non-formalisable

- Prose standards / RFCs / papers
- Powerpoint decks *(v common)*
- The code itself
- Reference implementations
- Inline code comments
- Test cases
- User stories
- Requirements documents
- Regulatory rules
- Scribbled notes on coffee-shop napkins
- …

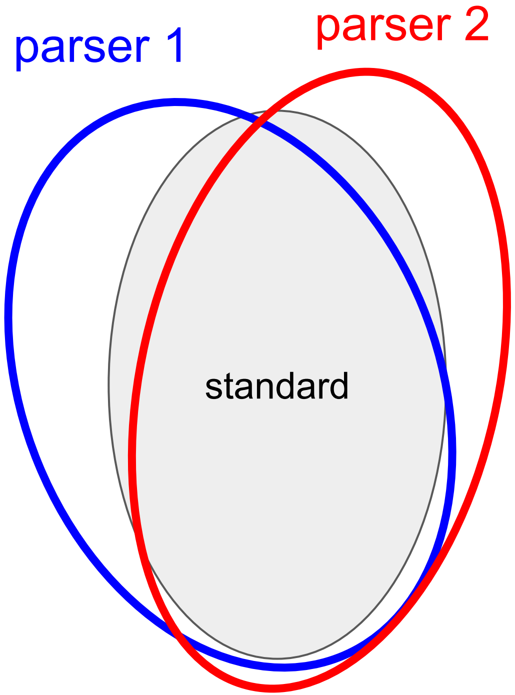
Anecdote: PDF, a spec that does not exist

We formalized PDF in our format definition language Daedalus (https://github.com/GaloisInc/daedalus)

- Testing on millions of cases
- Worked closely with the PDF association

But…

- Non-descriptive: different from real parsers
- Non-normative: doesn't characterize bugs
- Unclear how to get to a more rigorous & accepted specification



We’ve only explored the easiest classes of spec

Cryptographic algorithm

Operating system

Document format

CPS system, eg nuclear reactor

Web browser

AI-driven chemical synthesis tool

Generic conversational AI

Increasingly:

- *Complex*
- *Ambiguous*
- *Hard to reason about*
- *Contended by users*
- *‘Open world’*

We’ve only ex classes of spec

We only really have examples of these two levels in industry use

Cryptographic algorithm

Operating system

Document format

CPS system, eg nuclear reactor

Web browser

AI-driven chemical synthesis tool

Generic conversational AI

Increasingly:

- *Complex*
- *Ambiguous*
- *Hard to reason about*
- *Contended by users*
- *‘Open world’*

Eg. 1: operating system verification

Specification: “Data should not flow from high to low security domains”

Approach (similar to SeL4):

- Tag data with security levels
- Model operating system operations via logic
- Prove that each operation preserves security invariants

Challenges:

- Specification: what user-side behaviors are possible?
- World modelling: are hardware / physics behaviors in scope?

… vs Eg. 2: AI-driven chemical synthesis tool

Specification: “Do not generate chemicals that harm humans”

Approach:

- Write a model of ‘harmful chemicals’
- Prove some guard system correctly rejects all such chemicals

Challenges:

- Need a granular probabilistic model of chemistry and human biology
- “Harm” is a socio-technical term - need to capture social convention / law
- “Harm” may include combined chemicals, so we need a compositional theory how chemicals could be used

Optimism: can probabilistic programming help?

Maybe? My sense is the tech is very early

Hard problems:

- How do we reason about probabilities at scale?
- How do we validate models vs the real world, esp. over time?
- Is probabilistic reasoning valid in the presence of adversarial actors?

Optimism: can AI help?

Plausible ideas:

- AI + human teaming on specification writing
- AI-driven science to develop accurate models of the world

A lot of work is needed on ‘spec tech’

We have a 50+ years of tools for easy-to-specify things

~Zero tools for hard-to-specify things

For GSAI:

- Big divide between plausible cases and ‘science fiction’
- Urgent need to experiment / grow the bench
- Unclear if / what progress is being made

Proofs in the Wild

What's done today?

What's close?

What's far?

Summary

https://mikedodds.github.io

What’s done today:

- A small number of successful proof tech deployments
- Strong evidence of usefulness in some domains
- A deep bench of tools and ideas, though many are too expensive
- Key barrier is cost/benefit - proofs are hard and specs are hard

What’s close: proofs

- AI is great for proof search!
- Current tool architectures can integrate AI with very little modification
- *Optimism:* proofs get cheap, proof tech gets much more useful

What's far: specifications / world models

- Current proof tech focuses on a tiny range of easy-to-specify things
- We have ~zero examples of success in more difficult-to-specify domains
- Spec tech needs rapid development if we expect to apply it soon (per GSAI)

Thanks!

miked@galois.com
https://mikedodds.github.io

X: @miike
@m-dodds.bsky.social

galois

N things I learned trying to do formal methods in industry:

https://mikedodds.github.io/files/talks/2024-10-09-n-things-I-learned.pdf