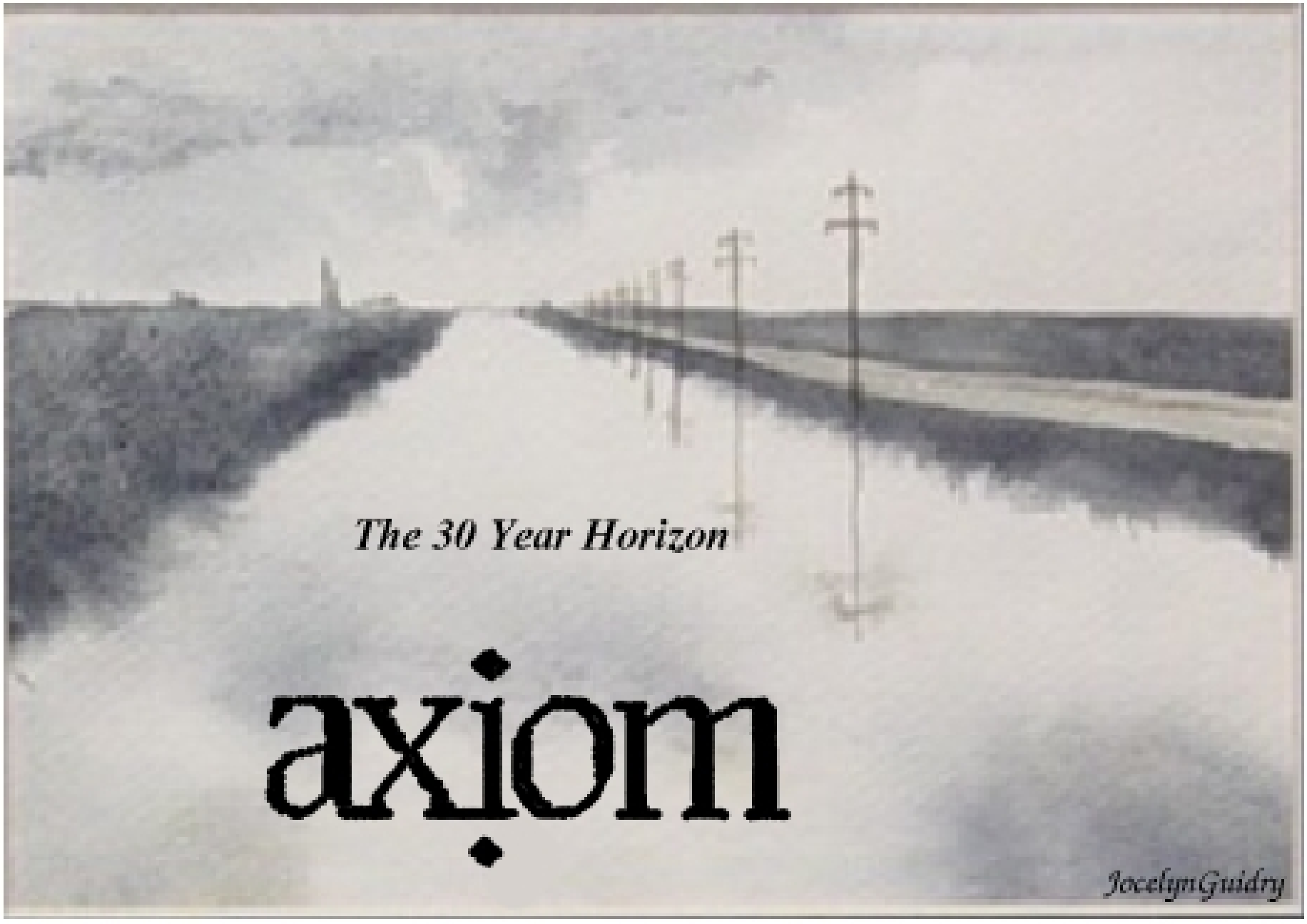
Proving Axiom Sane
The 30 Year Horizon

Timothy Daly

Carnegie Mellon University
Computer Science Department
http://daly.axiom-developer.org
axiomcas@gmail.com

July 24, 2018
International Congress on Mathematical Software


The 30 Year Horizon
axiom
JocelynGuidry

Outline

What:	Some History
Goal:	Prove Axiom Sane
Why:	Types, Signatures, Representations, and Propositions
But:	The Problem
People:	Prior Art
How:	Research Directions
Conclusion:	This Won't Succeed

What: History

- 1965 Scratchpad/1 Griesmer, Blair, Jenks, Yun
- 1971 Scratchpad/2 Jenks, Trager, Davenport, etc.
- 1992 Axiom Numerical Algorithms Group (NAG)
  - Commercial competitor to Mathematica/Maple
  - One of "The Big 3"
- 2000 Axiom withdrawn / released as Open Source

What: History

Axiom: A Research Platform, then and now.

- Estimated 21 years, $42 millon, 1.2 million lines of code
- Literate Software Conversion
- Native Symbolic / Numeric computing (BLAS/LAPACK)
- Gustafson numbers [Gust16a] and FPGA
- Proving Axiom Sane

Goal: Prove Axiom Sane

- Current Research Goal: Prove Axiom Sane
- WAS: Prove Axiom **Correct**
  - "Correct" has the wrong meaning
  - "Sane" synonyms: logical, rational, judicious, sound
- Merge Computation and Proof
- Focus on global issues, not particular algorithms

Why Formal Math?

Correctness is non-negotiable
– *David Stoutemyer [Stou11a]*

Writing is nature's way of letting you know how sloppy your thinking is.
– *Richard Guindon [Guinxx]*

Mathematics is nature's way of letting you know how sloppy your writing is.
– *Leslie Lamport [Lamp02]*

Formal mathematics is nature's way of letting you know how sloppy your mathematics is.
– *Leslie Lamport [Lamp02]*

Why Axiom?

Axiom

- is strongly typed
- is based on abstract algebra
- has clean separation by Category/Domain
- has 2 of the 3 criteria for proofs
- has thousands of algorithms
- is an open source research platform
- is the pinnacle of constructive mathematics

Why: Axiom is Strongly Typed

Unlike almost all other computer algebra systems, Axiom is strongly typed. The Types are (mostly) inferred in the interpreter but the compiler insists on Types everywhere. Axiom can coerce between Types.

$$> \mathrm{t1} := 1/2 * \mathrm{x}\hat{}2 \; + \; 1/3 * \mathrm{x} \; + \; 1/5$$

$$(1) \quad 1/2x^2 + 1/3x + 1/5 \quad \textbf{Polynomial}(\textbf{Fraction}(\textbf{Integer}))$$

$$> \mathrm{t1} :: \mathrm{FRAC}(\mathrm{POLY}(\mathrm{INT}))$$

$$(2) \quad \frac{15x^2 + 10x + 6}{30} \quad \textbf{Fraction}(\textbf{Polynomial}(\textbf{Integer}))$$

Why: **Typeclass Components**

Typeclasses have 3 essential components:

1. Signatures
2. Carriers (aka Representation)
3. Propositions

Why: **Category** : 1st of 3 Components

A **Category** contains signatures of exported functions with their types and, possibly, a default implementation.

For example, **GcdDomain**, a **Category**, exports the signatures:

gcd : (%,%) -> %
gcd : List(%) -> %

Children of **GcdDomain** also inherit function signatures from:

`AbelianGroup, AbelianMonoid, AbelianSemiGroup, Algebra, BasicType, BiModule, CoercibleTo, Ring, CommutativeRing, CancellationAbelianMonoid, Rng, EntireRing, IntegralDomain, LeftModule, LeftOreRing, Module, Monoid, RightModule, SemiGroup, SetCategory`

Why: **Domain** : 2nd of 3 Components

A **Domain** contains the Representation (aka 'carrier') of the type and the implementations of the functions.

For example, **SparseUnivariatePolynomial** is represented as

```
List Record(k:NonNegativeInteger)
```

But an **XRecursivePolynomial** is represented as

```
Union(Ring,
  Record(c0:Ring,
      reg:FreeModule1(XRecursivePolynomial,OrderedSet))
```

They export very similiar inherited function signatures but implement locally optimized functions.

The **Category** / **Domain** split allows multiple representation of similar algebraic structures for efficient implementation.

Why: **Propositions** : 3rd of 3 Components

Axiom is missing **Propositions**.

We intend to add them in multiple places.
We intend use them to prove algorithms.

At the **Category** level, the propositions would state axioms available to any **Domain** that inherits, e.g. `transitivity`

At the **Domain** level, the propositions would state axioms available for proving functions (e.g. domain restrictions, like `Nat` for the **NonNegativeInteger** domain).

At the **function** level (e.g. the **gcd** function), propositions would state pre- and post- conditions, as well as properties such as

```
gcd(x,y) = gcd(y,x)
```

But: The Problem

It is argued that formal verifications of programs, no matter how obtained, will not play the same key role in the development of computer science and software engineering as proofs do in mathematics. Furthermore, the absence of continuity, the inevitability of change, and the complexity of specification of significantly many real programs make the formal verification process difficult to justify and manage. [Demi79]

Except, I claim, in Computational Mathematics.

But: The Problem

The Misfortuntes of a Trio of Mathematicians Using Computer Algebra Systems. Can We Trust in Them? [Dura14]

1. No clear way to communicate difficulties to vendor
2. Unable to debug since source code is unavailable
3. (I add..) No documentation of algorithm or citations
4. (I add..) No proof of algorithms

My survey of over 100 papers about Computer Algebra and Proof Systems [Daly18] shows that the Computer Algebra systems are universally untrusted.

But: The Problem

We would like to verify Axiom's algorithms in order to show that Axiom is **Sane** (aka 'logical', 'rational', judicious', 'sound')

Verification does not guarantee perfection. A system is never correct: at best, it is consistent with its specification.
– Lawrence Paulson [Paul87, p10]

People: Prior Art

Many different approaches in the last 50 years:

- general math tool	**de Bruijn** [Brui68]
- add side-conditions to Axiom	**Dunstan** [Duns99a]
- OpenMath communications	**Dalmas et al.** [Dalm97]
- build algebra on Proof Assistants	**Kaliszyk** [Kali09]
- verifying Mini-Maple	**Khan** [Khan13]
- built DoCon-A algebra in Agda	**Meshveliani** [Mesh16]
- Math-in-the-Middle	**Dehaye et al.** [Deha16]
- ..	

There are problems with each approach. For instance, exporting Axiom expressions and then importing them into Axiom loses vital Axiom-specific type information.

People: Prior Art

My survey [Daly18] shows two independent streams of Computational Mathematics, Computer Algebra and Proof Systems. Combination efforts include:

- **CA in Proof**
  - lack time and funding to build full CA
  - CA is mostly undocumented PhD thesis work
- **Bridges** – lack of trusted CA so results need reproving
- **Islands** holding Truth
  - central independent semantics of mathematics
  - Axiom to Island to Axiom loses types
- **Proof in CA**
  - most systems lack types
  - lack of source code for algorithms

Axiom is ideally suited for a **Proof in CA** system.

People: Approaches

Beeson [Bees16] categorized approaches:

- **Believer** Just believe uncertified software
- **Witness** Check the final result
- **Skeptic** Check every step of result
- **Autarkic** aka reflection... combine algorithm and proof

Axiom is striving to be Autarkic.
(Autarky is 'the quality of being self-sufficient')

How: Following ACL2

ACL2 merges language and logic.

> *Instead of just providing an execution engine for the language, we describe the language as a mathematical logic, with axioms and rules of inference. And we provide a reasoning engine. Thus, in addition to being able to execute your programs, you can use the reasoning engine to prove things about them.*
> *Kaufmann, Manolios, and Moore [Kauf00]*

How: Current Path

1. Embed a proof kernel
   - Meta-circular interpreters [Stee78a]
   - Jitawa [Myre11]
   - Milawa [Davi15]
2. Decorate Axiom hierarchy with propositions
   - Doing Algebra in Simple Type Theory [Gunt89]
   - LEAN proof of GCD [Avig14]
3. Rewrite compiler to verify proofs
   - Compiling with Continuations [Appe92]
4. Compute and carry provisos
   - 10 Commandments for expression simplification [Stou11a]
5. Rewrite interpreter to handle proofs
6. Prove Axiom algorithms
   - Proving Axiom Correct [Book13]
7. Expose proof technology to user

The 30 Year Horizon

What is **Global Success**?

> *The QED software of the future must be able to smoothly integrate logical proof steps with computation, perform large computations efficiently, and leave no doubt as to the correctness either of the computations or their connection to the logical part of the proof.*
> *– Michael Beeson [Bees16]*

Proven and efficient computation is possible. It requires combining techniques from both branches of Computational Mathematics. This research is directed toward that long-term end.

Conclusion: This Won't Succeed

Some reasons this research will not succeed: [Cyph17]

Leap of Faith... Tools for Toy Problems... Opaque Specifications... Too Large a Task... False Axioms... Ripple Costs... Informal Implementation Details... Specification Mismatch... Natural / Formal Mismatch... Specification Flaws... Specification Narrowing... Specification Widening... Specification Impedence Mismatch... Specification Blindness... Contradictory Proofs... Proven Programs are Wrong... Proofs are a Social Process... Chicken and Egg... Boiling the Ocean... Meta-Theory Costs... Partial Functions... Undecidable Theories... Etc...

Conclusion: This Won't Succeed

> *Almost anything carried to its logical extreme becomes depressing, if not carcinogenic*
> *– Ursula K. LeGuin [Legu76]*

What is **Local Success**?

GCD is exported by 22 Domains,
e.g. NonNegativeInteger, SparseUnivariatePolynomial

A local success is proving GCD down the the metal.

This research likely won't succeed.

... but (assuming Intuitionism), this does not imply failure.

Proving Axiom Sane
The 30 Year Horizon

Timothy Daly

Carnegie Mellon University
Computer Science Department
http://daly.axiom-developer.org
axiomcas@gmail.com

July 24, 2018
International Congress on Mathematical Software


Andrew W. Appel. *Compiling with Continuations*. Cambridge University Press, 1992, 0-521-03311-X.


Jeremy Avigad. LEAN proof of GCD, 2014.


M. Beeson. Mixing Computations and Proofs. *J. of Formalized Reasoning*, 9(1):71–99, 2016.


Axiom Authors. *Volume 13: Proving Axiom Correct*. Axiom Project, 2016.


N.G. de Bruijn. The Mathematical Language Automath, its Usage, and Some of its Extensions. *Lecture Notes in Mathematics*, 125, 1994.


Cypherpunks. Chapter 4: Verification Techniques, 2017.


Stéphane Dalmas, Marc Gaëtano, and Stephen Watt. An OpenMath 1.0 Implementation. In *Proc. 1997 Int. Symp. on Symbolic and Algebraic Computation*, ISSAC'97, pages 241–248. ACM, New York, NY USA, 1997, 0-89791-875-4.


Timothy Daly. Proving Axiom Sane: Survey of CAS and Proof Cooperation, 2018.


Jared Davis and Magnus O. Myreen. The Reflective Milawa Theorem Prover is Sound. *J. Automated Reasoning*, 55(2):117–183, 2015.


Paul-Olivier Dehaye, Mihnea Iancu, Michael Kohlhase, Alexander Konovalov, Samuel Lelievre, Dennis Muller, Markus Pfeiffer, Florian Rabe, Nicolas M. Thiery, and Tom Wiesling. Interoperability in the OpenDreamKit project: The Math-in-the-Middle Approach. In *Intelligent Computer Mathematics*, pages 117–131. Springer, 2016, 9783319425467.


Richard A. DeMilo, Richard J. Lipton, and Alan J. Perlis. Social Processes and Proofs of Theorems and Programs. *Communications of the ACM*, 22(5):271–280, 1979.


Martin N. Dunstan. *Larch/Aldor - A Larch BISL for AXIOM and Aldor*. PhD thesis, University of St. Andrews, 1999.


Antonio J. Duran, Mario Perez, and Juan L. Varona. The Misfortunes of a Trio of Mathematicians Using Computer Algebra Systems. Can We Trust in Them? *Notices of the AMS*, 61(10):1249–1252, 2014.


Dick Guindon. Writing is nature's way of letting you know how sloppy your thinking is, unknown.


Elsa L. Gunter. Doing Algebra in Simple Type Theory. technical report MS-CIS-89-38, University of Pennsylvania, 1989.


John Gustafson. *The End of Error: Unum Computing*. Chapman and Hall / CRC Computational Series, 2016, 978-1482239867.


Cezary Kaliszyk. *Correctness and Availability: Building Computer Algebra on top of Proof Assistants and making Proof Assistants available over the Web*. PhD thesis, Radboud University, Nijmegen, 2009.


Matt Kaufmann, Panagiotis Manolios, and J Strother Moore. *Computer-Aided Reasoning: An Approach*. Kluwer Academic Publishers, 2000, 0-7923-7744-3.


Muhammad Taimoor Khan. On the Formal Verification of Maple Programs. technical report 13-07, RISC Linz, 2013.


Leslie Lamport. *Specifying Systems*. Addison-Wesley, 2002, 0-321-14306-X.


Ursula K. LeGuin. *The Left Hand of Darkness*. Penguin Random House, 1976, 978-1-101-66539-8.


Sergei D. Meshveliani. *DoCon – A Provable Algebraic Domain Constructor*. User Manual, Version 0.04, 2016.


Magnus O. Myreen and Jared Davis. A Verified Runtime for a Verified Theorem Prover. *NCS*, 6898:265–280, 2011.


Lawrence C. Paulson. *Logic and Computation*. Press Syntacate of Cambridge University, 1987, 0-521-34632-0.


Guy Lewis Steele and Gerald Jay Sussman. The Art of the Interpreter. technical report AI Memo No. 453, MIT, 1978.


David R. Stoutemyer. Ten Commandments for Good Default Expression Simplification. *J. Symbolic Computation*, 46:859–887, 2011.