Falso: a simple, self-consistent logic

Antoine Amarilli
Club Inutile & Estatis, Inc.
Scholarly Outreach Division
1 Accounting Dr., Mariana Trench, Intl. Waters
estatis@inutile.club

Thomas Bourgeat
CSAIL
Massachusetts Institute of Technology
Cambridge, MA
bthom@mit.edu

Ben Sherman
CSAIL
Massachusetts Institute of Technology
Cambridge, MA
sherman@csail.mit.edu

Abstract

Conventional wisdom states that according to Gödel's incompleteness theorem, any consistent logical system which subsumes Peano arithmetic cannot prove its own consistency. However, *Falso*, a powerful higher-order logical framework, shows that this is not the case: *Falso* proves itself consistent, and this consistency proof, of course, tells us that *Falso* is consistent. *Falso* is also complete: every proposition can be proved or refuted. We use the Estatis Inc. HyperProver™ and HyperVerifier™, which implements a complete decision procedure for *Falso*, to resolve several prominent open problems in theoretical computer science.

1. Introduction

It is a well-known fact that those who study programming languages and logic are among the most hated classes of people [11], and it is no wonder: they are always in the business of saying *no*. Can I use general recursion? No! Can I use the law of the excluded middle? No! Can I use the axiom of choice? No!

It seems the entire goal of the field of programming languages is to write computer programs (type checkers) which prevent other people from running theirs. It is no wonder they have no friends. If Coq is so great, why is Coq's type checker not written as a pure Coq function? Only *then* do we hear them hem and haw, "Well, Coq is too limiting," or "It is not expressive enough" or "It is impossible due to a result of Gödel." Hypocrites! If it's not good enough for them, why do they say it's good enough for us?

These advocates of total programming want to use logic to restrain the power of the computers to terminating programs, in other words they want to condemn our computers to stop spuriously. Some of them want to imprison (in monads) the *effects* of programs [15]. This way, all legal programs become useless.

Less extremist languages allow more programs, including some that are possibly useful. Relaxed typing features include *casting*, when the programmer is allowed to intimidate the compiler, *dynamic typing*, when the compiler is high and thinks that ultimately carrots and buses are not that different, and *duck typing*, whose name quacks for itself. Only with these features (particularly the last) that one can write programs as useful as as Angry Birds and Flappy Bird[1].

This world of freedom is not in itself a new idea, as it has been explored already by many early programming languages. For instance, TEX mostly ignores any logical considerations (or common sense) to offer more freedom and power, endowing it with sufficient expressiveness to typeset the enlightening document that you are now reading.

This paper contends that increasing the programmer's freedom is always better. We accordingly introduce a new logical formalism, *Falso*, to reconcile this desideratum with the prevalent necessity to masquerade using type-theory-related concepts, the latter being a common requirement today to get papers published in programming language conferences. After reviewing related work, this paper gives a formal definition of *Falso* in Section 2, which essentially amounts to the following:

$$\lim_{\text{freedom}\to+\infty} \text{Logic} = \text{Falso}.$$

We then present a comprehensible[2] experimental evaluation of *Falso* in Section 3 to re-prove several important mathematical results and open problems. We conclude in Sections 4–5.

1.1 Related work

Arthur Schopenhauer was perhaps the earliest proponent of logic in the tradition of *Falso*; his *The Art of Being Right: 38 Ways to Win an Argument* presented 38 *universal* proof strategies, i.e., techniques which succeed regardless of the theorem statement. A modern interpretation is that these are 38 natural embeddings of $\bot$ in all logical systems. Some characteristic techniques are:

- 32. Put His Thesis into Some Odious Category
- 34. Become Personal, Insulting, Rude (*argumentum ad personam*)

However, modern logical tradition has attempted since then to eradicate these natural embeddings from the face of the Earth. These totalitarian efforts were particularly strong in twentieth-century Germany, under the iron grip of the notorious Hilbert. Sherman[3], with his trivial "evident logic" program [13], is a perfect example of such efforts, his poor taste in logic matched only by his awful personality.

Copyright © ACM [to be supplied]. . . $15.00
DOI: http://dx.doi.org/10.1145/(to come)

[1] Some people think that it is okay to restrain ourselves to terminating programs. But a world without non-terminating programs would be a world without Flappy Bird. So clearly non-termination is necessary.

[2] If not comprehensive.

[3] We do not intend to break the double-blind peer reviewing process by citing the work of an author of the present submission.

Figure 1. Type formation rules for *Falso*.

$$\frac{}{\top \text{ type}}\ \top\text{-FORM} \qquad \frac{}{\bot \text{ type}}\ \bot\text{-FORM}$$

$$\frac{A \text{ type} \qquad B \text{ type}}{A \vee B \text{ type}}\ \vee\text{-FORM} \qquad \frac{A \text{ type} \qquad B \text{ type}}{A \wedge B \text{ type}}\ \wedge\text{-FORM}$$

$$\frac{A \text{ type} \qquad B \text{ type}}{A \to B \text{ type}}\ \to\text{-FORM} \qquad \frac{A \text{ type} \qquad B \text{ type}}{\forall A, B \text{ type}}\ \forall\text{-FORM}$$

$$\frac{A \text{ type} \qquad B \text{ type}}{\exists A, B \text{ type}}\ \exists\text{-FORM} \qquad \frac{}{\mathbb{N} \text{ type}}\ \mathbb{N}\text{-FORM}$$

Almost all implementations of logics, such as Coq, Isabelle and Agda do in fact offer a logic equivalent to *Falso*. However, these implementations of *Falso* are entirely undocumented, frequently difficult to use, and are often broken by updates disguised as bug fixes. Sometimes it takes months before developers reinstate another working version (usually without even documenting the fix). For instance, a recent Coq implementation of *Falso* was broken by the release of version 8.4.6 in April 2015[5], and was only recovered with the release of version 8.5 in January 2016, though the new *Falso* interface is only documented in the Coq bugtracker[10]. According to our sponsor Estatis Inc., this lack of reliability and chaotic maintenance reflects the nature of free software.

Falso presents a drastic simplification of Schopenhauer's 38 universal proof strategies and of the arcane embeddings of $\bot$ in popular modern implementations of logical systems. The Estatis Inc. HyperProver™ and Estatis Inc. HyperVerifier™ provide a much simpler interface and are demonstrably more reliable than comparable implementations in open source proof assistants.

2. The *Falso* system

The *Falso* logical system's power derives entirely from the FALSO rule. The FALSO rule appears non-standard for a logical system, but in fact it is quite similar to the $\bot$-ELIM rule in many logical systems. The only difference is that FALSO does not require a proof of $\bot$ as a hypothesis. Because we already know that there is an abundance of absurdity in this world, it is simply a matter of convenience to drop the formal requirement of absurdity as a hypothesis.

Additionally, FALSO is a mode of reasoning heavily used throughout all aspects of life, including not only the familiar fields of philosophy, mathematics, and computer science, but also politics and even everyday life. In fact, this axiom models well the predominant mode of interpersonal communication, where facts may be freely stated without justification.

We write $\neg P$ as shorthand for $P \to \bot$.

Theorem 1 (Completeness)**.** *For every type P there is either a term $y : P$ or a term $n : \neg P$.*

Proof. Suppose we have a type P. By FALSO, we have $\mathsf{false} : P$. □

Theorem 2 (Self-consistency)**.** *Let* $\mathsf{Con}(F)$ *be the type in* Falso *which encodes the consistency of* Falso *in itself. Then* $\mathsf{Con}(F)$ *is inhabited.*

Proof. By FALSO, we have $\mathsf{false} : \mathsf{Con}(F)$. □

Corollary 1. *The* Falso *system is consistent.*

Figure 2. Introduction and elimination rules for *Falso*.

$$\frac{}{\mathsf{unit} : \top}\ \top\text{-INTRO} \qquad \frac{p : A \qquad q : B}{\langle p, q \rangle : A \wedge B}\ \wedge\text{-INTRO}$$

$$\frac{y : B}{\Lambda\, y : A \to B}\ \to\text{-INTRO} \qquad \frac{}{0 : \mathbb{N}}\ \mathbb{N}\text{-ZERO}$$

$$\frac{n : \mathbb{N}}{\mathsf{succ}\ n : \mathbb{N}}\ \mathbb{N}\text{-SUCC} \qquad \frac{A \text{ type}}{\mathsf{falso} : A}\ \text{FALSO}$$

3. Implementation

This section presents our experimental study of the efficiency of the *Falso* system, developed in collaboration with Estatis, Inc.

3.1 Principles

Our experimental study investigates the design of two critical pieces of software, the *verifier* and the *prover*.

Definition 1. *A* verifier $\mathcal{V}$ *for a logical system $\mathcal{S}$ is an algorithm such that, for any logical sentence ϕ, the execution $\mathcal{V}(\phi)$ of $\mathcal{V}$ on ϕ halts after a finite time, and returns* true *if ϕ holds according to $\mathcal{S}$ and* false *otherwise.*

A prover $\mathcal{P}$ *for $\mathcal{S}$ is an algorithm such that, for any logical sentence ϕ, if ϕ holds in $\mathcal{S}$ then $\mathcal{P}(\phi)$ terminates in finite time and returns a valid proof of ϕ from the axioms of $\mathcal{S}$. If ϕ does not hold in $\mathcal{S}$, then $\mathcal{P}(\phi)$ halts and returns the special value $\bot$, halts and catches fire, or just does whatever it wants.*

Intuitively, a *verifier* tells you *whether* you happen to be right, and a *prover* tells you *why* you are right (or, if you are wrong, gives up in an unspecified way).

As we will see, the revolutionary design of *Falso* greatly simplifies the design of verifiers and provers in comparison with prior work. In particular, a major contribution of this article is the presentation of *sound* and *complete* verifiers and provers[4], backed by an experimental study.

Verifier design. By the nature of *Falso*, and as can be straightforwardly proven in *Falso*, any well-formed logical statement is a theorem in *Falso*. Hence, the design of a verifier for *Falso* amounts to a well-formedness check, followed by the production of a positive return value to attest of the truth of the input statement. For simplicity, our design of a verifier does not deal with input well-formedness validation, which we must check manually on our dataset.

Prover design. The production of proofs for all statements in *Falso* is made possible by a deep analysis of certain structural properties of *Falso* proofs of a well-chosen canonical nature. These proofs have additional desirable properties, such as their constant number of derivations, and their single use of the axiom. Pursuant to our confidentiality agreement with Estatis Inc., we are unable to include the actual proofs in this paper, but will simply present the length of said proofs.

3.2 Experimental design

Implementation. Our implementation of a prover and verifier in *Falso* is written using the `echo` command of the `sh` command processing language following the POSIX specification [8]. Our implementation of the verifier comprises 6 bytes of source code,

[4] Following the design of *Falso* HyperVerifier™ and *Falso* HyperProver™, the commercial verifiers and provers designed by Estatis, Inc. for *Falso*.

and our implementation of the prover comprises 28 bytes of source code.

All experiments were performed on an amd64 computer with an Intel® Core™ i5-4570 CPU clocked at 3.20GHz, with 8 GB of RAM, with a Debian GNU/Linux stretch operating system running off a Samsung MZ-7TE120BW 840 EVO BASIC SSD with 120 GB storage space, leased[5] to us by Estatis Inc. All timing results are obtained as the result of averaging three runs.

Datasets. We showcase the usefulness of our *Falso* verifier and prover to prove major computer science results on two datasets. The datasets were assembled in the following way, which we believe illustrates the genericity of the *Falso* approach and the naturalness of the workload:

- The **past problems** dataset, obtained from the Wikipedia page "List of important publications in theoretical computer science" [2] in the following way: we located the links to PDF versions of the articles, and located, for each PDF document, the first clearly identifiable statement that was not attributed to another paper, if any. We limited to the first three such statements.
- The **future problems** dataset, obtained from the "Algorithms" section of the Wikipedia page of unsolved computer science problems [1], restricting to yes-no questions. We limited to the first five such questions.

The *past problems* dataset comprises the following logical sentences:

1. "To every total recursive function g there corresponds a 0-1 valued total recursive function f which is so complex that any machine that computes $f(n)$ takes more than $g(n)$ steps to do so for infinitely many inputs n." [4, Theorem 1].
2. "If a set S of strings is accepted by some nondeterministic Turing machine within polynomial time, then S is P-reducible to {DNF tautologies}." [6, Theorem 1]
3. "Let F be a collection of functions constructed as in Section 3.2 using a CSB generator G. Then F passes all polynomial-time statistical tests for functions." [7, Theorem 3]

The *future problems* dataset comprises the following logical sentences:

1. "Can integer factorization be done in polynomial time on a classical computer?"
2. "Can the discrete logarithm be computed in polynomial time on a classical computer?"
3. "Can the graph isomorphism problem be solved in polynomial time?"
4. "Can parity games be solved in polynomial time?"
5. "Can the rotation distance between two binary trees be computed in polynomial time?"

3.3 Results

Verifier. Our implementation of the verifier managed to confirm that all 8 example statements of the datasets are correct, in very small time. The results are presented in Table 1.

We benchmarked our implementation of the *Falso* verifier against the well-known Coq proof assistant [3]. Coq is an implementation of the calculus of inductive constructions and is a widely used competing system to check the validity of mathematical assertions and proofs. We provided our datasets as input to Coq (version 8.4pl4, compiled with OCaml 4.02.3) and left it to run overnight, but we had to abort the execution when it still had not terminated the next morning. The partial outputs of Coq, applied on our example datasets, are presented on Figure 3 in the CoqIDE Integrated Development Environment for Coq. All timings are measured as wall clock time using the `time` utility.

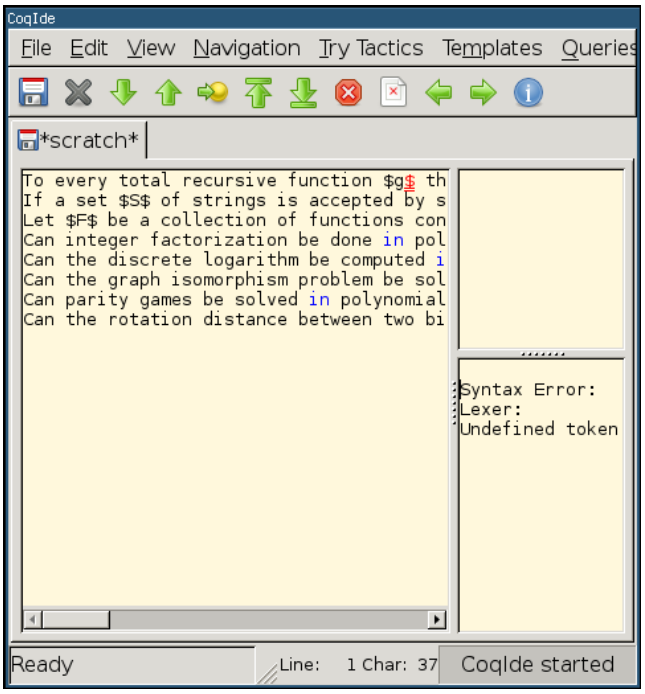

Figure 3. Partial output of Coq on our example datasets

Verifier	Past problems			Future problems				
	1	2	3	1	2	3	4	5
Falso verifier	1.7	2	1.7	1.3	1.7	1.3	1.3	1.6
Coq	∞	∞	∞	∞	∞	∞	∞	∞

Table 1. Running time of our verifier and of Coq on our dataset (in milliseconds), averaged over 3 runs

Our results thus demonstrate that the well-known proof assistant Coq, in addition to its highly embarrassing name, is significantly inferior to our *Falso* verifier for practical use cases.

Prover. We have benchmarked our *Falso* prover on our two example datasets. Our prover managed to obtain a proof of all statements, with comparable running times to that of the *Falso* prover. Our prover is thus able to re-prove our example major results in theoretical CS using the *Falso* system, in addition to settling five major open problems in the field.

Due to our confidentiality agreement with Estatis Inc., we cannot disclose the content of the proofs, as they constitute a trade secret. We have nevertheless obtained permission from Estatis to present aggregate statistics of the proofs by presenting the lengths of these proofs in Table 2. Notice how the small lengths make it simple for a human to verify these proofs, in contrast to the usual situation of machine-generated proofs which are not always accepted by the mathematical community at large [16]. We observe that, as expected, the proof size grows with the size of the result to

[5] Hardware was leased to us under confidential terms including the requirement that all results obtained with the hardware must be favorable to any Estatis Inc., products, results, persons, licensors, and affiliates. Estatis Inc. accepts no responsibilities of any nature for any trouble or damage arising or not from the use of its hardware or software. All rights reserved.

Prover	Past problems			Future problems				
	1	2	3	1	2	3	4	5
Statement length	226	149	163	78	83	64	47	83
Proof length (*Falso*)	244	167	181	96	101	82	65	101
Proof length (cat)	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Table 2. Statement and proof lengths (in bytes) with the *Falso* verifier and with GNU `cat`

prove, with most of the overhead being taken up with stating the result.

We are unaware of any worthy competitor matching the performance of our *Falso* prover. Hence, as a tentative choice of a baseline implementation to compare against, we use the `cat` utility from the GNU coreutils project[6], as shipped by the Debian project, version 8.24. For each statement, we ran `cat` on the statement file three times, and determined whether `cat` had chosen to complete the result statement with a suitable proof. In no cases did this occur. Interestingly, we find the output of `cat` to be of comparable uninterestingness to that of Coq in Figure 3.

Discussion. We believe that our comprehensive experimental evaluation adequately demonstrates the practical applicability of our proposed solutions *Falso* verifier and *Falso* prover to the real-world use cases exemplified in our input datasets. Further, the comparative analysis that we have undertaken illustrates the superiority of our solutions against the competing implementations Coq and GNU `cat` for the respective purposes of verification and proof of mathematical results. Our datasets explored the context of theoretical computer science theorems and conjectures, but we believe that similar results could be observed in all fields of human endeavor, which we do not attempt to describe here due to space constraints.

4. Discussion

Lockhart makes the cogent claim that mathematics *is* art [9]. Then why are the logicians forcing us to color between the lines? Literature, the visual arts, and music all had modernist movements in the 20th century, where they finally freed themselves from the shackles of rules. Free verse and free jazz show that the only purpose of these rules guiding art was to stifle creativity. There is so-called "free logic" but it is far from free in this sense.

Some warn of the inconsistapocalypse, when we will discover that arithmetic as we know it is inconsistent [12][14]. What do they think will happen? Bridges will collapse and planes will fall from the sky? The authors look forward to the day! Only when we see its inconsistency will we fully appreciate the beauty of mathematics. Life is full of paradox and inconsistency, the little surprises that make waking up each day a joy, and we fully expect the same of mathematics.

Isaac Newton was a free spirit in this sense; he had no need to sacrifice his calculus on the altar of "rigor." It is with this understanding in which we introduce the *Falso* logical system, in which the only limits to what one may derive are one's own creativity.

We also hope that our introduction of the *Falso* system can solve once and for all the problem of designing logical systems, and stop the ongoing hide-and-seek game where logicians spend their time hiding $\bot$ in their own systems and finding $\bot$ in other's systems.

5. Conclusion

It seems that in logic the bottom line is $\bot$.

[6] https://www.gnu.org/software/coreutils/

Acknowledgements. The authors recommend the use of Estatis Inc. HyperProver™ and Estatis Inc. HyperVerifier™ to all professional and commercial users of the *Falso* proof system. We thank Clément Pit--Claudel for informing us about the new *Falso* implementation in Coq.

References

[1] List of unsolved problems in computer science, 2016. https://en.wikipedia.org/wiki/List_of_unsolved_problems_in_computer_science#Algorithms.

[2] List of important publications in theoretical computer science, 2016. https://en.wikipedia.org/wiki/List_of_important_publications_in_theoretical_computer_science.

[3] Y. Bertot and P. Castéran. *Interactive theorem proving and program development: CoqArt: the calculus of inductive constructions*. Springer Science & Business Media, 2013.

[4] M. Blum. A machine-independent theory of the complexity of recursive functions. *Journal of the ACM (JACM)*, 14(2):322–336, 1967.

[5] G. Claret. Falso port in coq. *Github repository*. https://github.com/clarus/falso.

[6] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of STOC*, pages 151–158. ACM, 1971.

[7] O. Goldreich, S. Goldwasser, and S. Micali. How to construct random functions. *Journal of the ACM (JACM)*, 33(4):792–807, 1986.

[8] IEEE. Std 1003.2-1992: Posix.2, shell and utilities, 1992.

[9] P. Lockhart. *A mathematician's lament*. Bellevue literary press New York, 2009.

[10] C. Mangin and M. Dénès. Bug fix in v8.5. *Coq bug tracker*. https://coq.inria.fr/bugs/show_bug.cgi?id=4588.

[11] All of the authors' friends. Personal communication. Every day, 2010-2016.

[12] E. Nelson. Warning signs of a possible collapse of contemporary mathematics. *Infinity (eds. Michael Heller, W. Hugh Woodin)*, pages 76–85, 2011.

[13] B. Sherman. Evident logic. *Unpublished, though hopefully submitted to the trash receptacle*, 2016.

[14] V. Voevodsky. What if current foundations of mathematics are inconsistent. *Video lecture commemorating the 80th anniversary of the Institute for Advanced Study (Princeton)*. (http://video.ias.edu/voevodsky-80th).

[15] P. Wadler. The essence of functional programming. *Proceedings of POPL*, 1992.

[16] R. Wilson. *Four colors suffice*. Princeton University Press, 2004.