From: **Sam Coward** s.coward111@gmail.com
Subject: Fwd: FAOC: Your manuscript entitled Formal Verification of Transcendental Fixed and Floating Point Algorithms using an Automatic Theorem Prover
Date: 8 February 2021 at 19:09
To: Lawrence Paulson lp15@cam.ac.uk

SC

Hi Larry,

They just got back with the reviews in which they've asked for revisions again :(

Quite a lot to digest and I haven't gone through with a fine tooth comb. But some fair and some less fair comments (in my opinion).
Let me know what you think?

Thanks,
Sam

---------- Forwarded message ---------
From: **Michael Butler** <em@editorialmanager.com>
Date: Mon, 8 Feb 2021, 18:18
Subject: FAOC: Your manuscript entitled Formal Verification of Transcendental Fixed and Floating Point Algorithms using an Automatic Theorem Prover
To: samuel richard coward <s.coward111@gmail.com>

FAOC-D-20-00054
Formal Verification of Transcendental Fixed and Floating Point Algorithms using an Automatic Theorem Prover
Formal Aspects of Computing

Dear Mr Coward,

Our referees have now reviewed your paper. You will see that they are advising that you revise your manuscript and therefore we are regrettably unable to accept it in its current form. If you are willing and able to undertake the work required to address the points raised we would be pleased to reconsider the revised manuscript.

The reviews can be found at the end of this email or can be accessed in the Editorial Manager website.

Your username is: Samuel Coward
If you forgot your password, you can click the 'Send Login Details' link on the EM Login page at
https://www.editorialmanager.com/faoc/

Please consider each point raised in the reviews, tell us about the changes you have made (or supply a rebuttal where no change is made) and submit a list of these with your revised manuscript.

We hope that you can supply the revised version by 18 Feb 2021. If this is not feasible, please let us know.

To submit a revision, go to the journal's website and log in as an Author. You will see a menu item called 'Submissions Needing Revision'. You will find your submission record there.

Yours sincerely,

Michael Butler
Associate Editor
Formal Aspects of Computing

COMMENTS FOR THE AUTHOR:

Reviewer #1: This paper presents a method for verifying error bounds for look-up table implementations of transcendental functions in hardware using the MetiTarski automated theorem prover. Rather than verify error bounds by exhaustive testing of the input space, the approach is to issue a theorem proving query for each entry in the lookup table, which asserts that the polynomial approximation of the transcendental function contained in the lookup table is within a specified error bound for the input interval corresponding to the lookup table entry. Since the size of the lookup table is small relative to the input space, this can yield a significant savings in verification cost. The approach is demonstrated on an implementation of binary logarithm.

The primary contribution of the paper is a case study: it demonstrates that theorem proving can outperform exhaustive testing for verification of LUT-based approximations of transcendental functions. Unfortunately, the case study consists of a single example -- the paper would benefit from more examples (e.g., page 3 indicates that 2^x and sin(x) may be good targets). The verification methodology, as I understand it, is to issue a query for each LUT entry (and to manually refine queries for which the theorem prover fails). If there is more to the verification methodology, the paper should highlight exactly what is new in the approach. Related to this, I do not understand the purpose of section 5. This

highlight exactly what is new in the approach. Related to this -- I do not understand the purpose of section 5. This seems to be a comparison of MetiTarski vs Gappa, which has little to do with the verification methodology proposed in section 3.

The correctness argument in section 4 should be clarified. In particular, I do not see where the templates account for the approximation error of ln(2), or how to arrive at the error terms 2^{-33}a and 2^{-23}.

Additional comments:
The paper should clarify early on exactly what the verification goal is. It was not clear until page 6 that the goal is to prove error bounds on the approximations.
p3 lines 20-23: the enhancement is the addition of the axiom (forall x. x-1 < floor(x) <= x)? I do not understand the last sentence.
p3 line 45-52: an example would be helpful. The sine example of a reduction step in the following paragraph appears to be a *nonexample that does not follow the pattern of step 1. Similarly for the log example on lines 23-23 of page 4 (e.g., we do not calculate k = min_k |x -c_k|, but rather k = min_k |1.significant(x) - c_k|).
p3 line 58 "the" Remez algorithm?
p4 line 21-25: "following identity. <equation>," -> "following identity: <equation>."
p54 line 28: "contains a zeroth order term" -> "is a constant polynomial"?
p6 ln 13 "are are"
p6 ln 28: is this a change of notation (\overline{ln} is now M_1_ln)? Why?
p8 line 58-60: is => intended to be read as implication?
fig 3: what is \overline{w} (opposed to w)?

Reviewer #2: This article is about how to formally prove a large class of algorithms that compute floating-point transcendental functions (such as exponential).

There is a large literature and many methods for computing elementary functions and this article tackles the most (current)
common methodology: argument reduction, polynomial evaluation, reconstruction. The formal part is done in MetiTarski.

The paper is well-written, honest and with a clear contribution. It handles both fixed-point and floating-point computations. The
methodology is complex, but it corresponds to the human expertise needed to write these functions.

My main problem with this article is that it is not demonstrated that it goes larger than 32 bits. Examples are that short and frankly, I am
not convinced by 32 bits experiments. Fig 2 is such an example: 18 bits as maximal bit width! The maximal runtime is 80 seconds which is
really small. Proofs on 32 bits can be done by brute-force (if the computer runs for one day while I do nothing, this is great for
me). Nobody is now afraid of several days of computations and brute-force is embarrassingly parallel. So I will be convinced only
from 64 bits.

Several other high-level questions:
- I was slightly disappointed that you could do no better than Gappa ten years ago.
- is Mathematica in the trust base?
- do you plan to handle iterative techniques (for division or square root)?
- do you plan to handle correct rounding?

Several low-level questions:
- note that there is a more recent IEEE-754 standard, and doubles are now "binary64".
- why did you not add binary logarithm to Metitarski?
- several references of interest:
  -- Formal verification of a floating-point elementary function,
for the tutorials of the 22nd ARITH conference (2015, Lyon, France).
https://www.lri.fr/~melquion/doc/15-arith22-expose.pdf
  -- Elementary Functions, book by Jean-Michel Muller

Typos:
p7 L36 "eqn 1" should probably be a reference
p11 l14 "IEEEdouble"
References to the ARITH symposium are not homogeneous

Reviewer #3: This paper shows how MetiTarski can be used to verify an implementation of mathematical library. The

authors first show what a table-based implementation of logarithm looks like and how they instrumented MetiTarski to verify it. The hardware implementation of such a function was the main motivation for this work. The authors then compare their approach to the Gappa tool, on the example of a sine function originally verified using it. While not revolutionary, the paper is well-written, interesting, and thus worth publishing. That said, I feel like it is missing a comparison with some other tools. But more importantly, there is an error when formalizing the algorithm in Section 5 (wrong error bound), which might invalidate the whole section. Thus, I am recommending a major revision. Below are some more comments.

It is not entirely clear to me what the refinement step entails. Is it just about more precisely characterizing the bits that get truncated due to the truncations of fixed-point arithmetic? Or is there more to it, e.g., increasing the degree of the polynomial approximations used for the elementary functions? Also, how often is exhaustive testing needed? It appears in Section 4, but I understand that it was triggered because the original algorithm was wrong. Once the algorithm was fixed, was the exhaustive testing stage still need to complete the correctness proof?

It would be worth comparing MetiTarski to some more recent tools. I did give a try to CoqInterval (Martin-Dorel, Melquiond, Proving Tight Bounds on Univariate Expressions with Elementary Functions in Coq, JAR 57:3, 2016). I only tested the first two cases of the "second set of toy problems" and it seems like CoqInterval, despite having the proofs verified by Coq, is 10 times faster, if not more. See the end of this review to see what the Coq scripts look like.

Another tool worth mentioning is FPTaylor (Solovyev et al, ACM TOPLAS 41:1, 2018). I do not think it would fare well on the use cases of the paper. But, as a tool that is kind of the state of the art when it comes to verifying floating-point implementations of elementary functions, it cannot be ignored.

As far as I could tell, all the examples of the paper use algorithms based on a single table (multiple outputs, but a single key). In particular, it means that all the subproblems are simple interpolations, that is, the most significant bits are present in all the computations. For example, on Figure 3, all the polynomials involve X or X+Z, but never Z alone. So, the ability of MetiTarski to handle numerous variables is hardly exercised by the examples. So, I would be interested in seeing how it handles more subtle table-based polynomials (Detrey, de Dinechin, Table-based polynomials for fast hardware function evaluation, ASAP 2005). There should be no difficulty to express those methods in the proposed framework. If it works, it would set MetiTarski ahead, since such methods are completely out of reach of CoqInterval and FPTaylor, I think.

I found the remarks about floor on page 6 quite interesting. But I have the feeling the reasoning of the authors can be pushed much further. First, let us consider the "naive" approach. Since "z>>1" is discarding one bit, the proper bounds are not [z/2-1;z/2] but [z/2-1/2;z/2]; and for "z>>2", the bounds are [z/4-3/4;z/4]. So, the naive approach could give for "z>>1 - z>>2" a lower bound equal to z/4-1/2. It is still not as good as the analytical bound z/4-1/4, but it is already better than z/4-1. Second, the idea of considering the least significant bits independently is clever, but I think we can do better. We do not have to look at the bits individually, we just have to look at chunks of bits. This would avoid the doubly exponential explosion related to the amount of variables. For example, if the expression was "z>>7 - z>>4", we do not need to consider seven bits separately; we only need to consider the chunk of bits 0 to 3, and the chunk of bits 4 to 6. So, only two
error variables are needed to get the optimal bound, not seven.

The relative errors on page 11 look dubious. The bound 2^(-54) seems half too small. I would have expected to see 2^(-53) there. Indeed, since the significand has 53 bits, the largest relative error is reached for 1+2^(-53) (which requires 54 bits and is thus rounded to 1), which gives a relative error of 2^(-53) / (1 + 2^(-53)). I do not know if MetiTarski is still able to prove the final error bound 2^(-67) once all the errors have been doubled. This needs to be checked.

I do not know whether Equation 4 was obtained mechanically or by hand. In the later case, this seems quite error-prone. In particular, I am unable to convince myself that the testcases in the directory "crlibm_sine" have the correct signs for all the errors. Would it not have been possible to keep e1,...,e6 as symbolic variables bounded between 1-2^(-53) and 1+2^(-53)? Or would it have caused the computation time of MetiTarski to explode?

Speaking of these error variables, the authors should comment on the issue of subnormal numbers. Given the algorithm and the input bounds (|Y| >= 2^(-200)), I would expect none of the intermediate computations to be less than 2^(-600), and thus to come close from the underflow threshold. Still, this should be checked, as the bound 2^(-53) would become meaningless in that case.

It would be interesting to see how MetiTarski fares on problems on which Gappa struggles, that is, argument reduction. It seems possible to get Gappa to accept Cody & Waite's reduction with some large effort (Boldo, Melquiond, Computer arithmetic and formal proofs, 2017), but I suppose that formalizing Payne & Hanek's reduction would be on a completely different level. More generally, argument reduction algorithms are still largely out of the scope of any automated tool, as far as I know. So, it would be great if MetiTarski were to improve on that situation.

The authors write that the runtimes of Gappa and MetiTarski are comparable, but they also write that it takes 460 seconds for MetiTarski. Yet in the original paper (de Dinechin et al, Certifying the floating-point implementation of an elementary function using Gappa, IEEE TC, 60:2, 2011), it seems like the runtime of Gappa and Sollya is under one second. So, this seems hardly comparable. Was there a typo in one or the other paper?

I do not understand the remark on page 12 that states that Harrison would have to redo a large amount of his proofs if

the input bit width were to be changed. As the authors state, Harrison got rid of all the floating-point concerns right from the start and then performed all his proofs on real numbers, so it seems to like most of these proofs would hardly change. The sentence of the authors on MetiTarski could just as well be adapted to Harrison's work: "HOL Light is an analytic tool, so increasing the space of discrete inputs only minimally alters the HOL Light problems."

```
(* Translation of 8bitlog_conjecture_yes_floor_0.tptp
   and 8bitlog_conjecture_yes_floor_1.tptp *)

From Coq Require Import Reals.
From Interval Require Import Tactic.
From Flocq Require Import Raux.

Open Scope R_scope.
Notation abs := Rabs (only parsing).
Notation floor x := (IZR (Zfloor x)) (only parsing).
Notation "x ^ y" := (powerRZ x y) (only parsing) : R_scope.

Goal forall X, 0<=X<=15 ->
  abs(ln(1 + 0*2^(-4) + (X*2^(-8))) -
    (0 + floor(9990234375*10^(-10)*(X+0*2^(4)))*2^(-8) +
    -47119140625*10^(-11)*(floor(X*2^(-2)+0*2^(2)))^2*2^(-12)))
  <= 2^(-7).
Proof.
intros X Hx.
Time interval with (i_taylor X).
(* Finished transaction in 0.054 secs (0.054u,0.s) (successful) *)
Qed.

Goal forall X, 0<=X<=15 ->
  abs(ln(1 + 1*2^(-4) + (X*2^(-8)))-
    (1220703125*10^(-17) + floor(9921875*10^(-7)*(X+1*2^(4)))*2^(-8) +
    -4189453125*10^(-10)*(floor(X*2^(-2)+1*2^(2)))^2*2^(-12)))
  <= 2^(-7).
Proof.
intros X Hx.
Time interval with (i_taylor X).
(* Finished transaction in 0.058 secs (0.054u,0.s) (successful) *)
Qed.
```

—