Response to Reviewers

Samuel Coward

July 2021

Summary of Changes

Thank you to the reviewers for their comments. They highlighted mistakes in the paper, which invalidated some of the results. These mistakes have now been rectified. The major changes are summarised below.

- Correction to the relative error in the "Comparison with Gappa" section (was section 5 now 6). The proofs were checked subject to these corrections and still passed.
- An additional case study was added, verifying a number of automatically generated square root function approximations from FloPoCo. This includes a binary64 implementation of the square root and is contained in a new section "Automatically Generated Designs: FloPoCo" (section 5).
- A brief experiment to understand the relationship between number of LUT entries required (which corresponds to the number of polynomials used to approximate the function) and the degree of the polynomials used. This has been added to section 2.
- Improved explanation of the error terms and a description of how error in approximating ln(2) is accounted for in section 4.
- Discussion about CoqInterval and FPTaylor added to the "Related Work" section (was 6 now 7). The CoqInterval authors have already conducted a MetiTarski comparison, which we discuss the results of.

Reviewer 1's Comments

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).

Response: An additional section was added to verify implementations of the square root function (section 5), so we now have applied the method to the logarithm, sine function and square root. We have also described in section 3 how the piecewise polynomial approximation method is to some degree function independent. The underlying architecture would not need to change to approximate a different elementary function. "The toy example problem could be applied to any elementary function as the architecture framework, relying on piecewise quadratic interpolation, is function independent."

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.

Response: Description of main novelty added to the introduction. "MetiTarski's understanding of elementary functions means that it is the only necessary theorem proving tool required in this methodology, unlike other approaches that have combined tools. The automatic generation of problem files from a template problem makes the verification effort simpler and could be used with other automatic theorem provers."

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.

Response: The example verified in the comparison with Gappa is a simplified version of the piecewise polynomial method, so fits into the same framework, namely much of the same manual effort applies due to the template approach used in the other case studies. Previous reviewers asked for a comparison with such a tool.

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}.

Response: Thank you to the reviewer for highlighting this oversight. The error in approximating $\ln(2)$ is not explicitly accounted for. Fortunately it can be proven that the error introduced is accounted for by the over-approximation of the error due to the truncation of the square term. See page 10.

Clarification of how the $2^{-33}a$ and 2^{-23} error terms arise added (see page 10). "the $2^{-33}a$ is an error term introduced to model the truncation of the squared term in the algorithm. More precisely, we are explicitly using the naive floor function bound..."

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.

Response: Added to the introduction. "To give an early outline of the approach, we take some piecewise polynomial implementation, that uses polynomials $p_i(x)$ for $i = 0, 1..., K$ and $x \in \Sigma_i$, to approximate a function $f(x)$, for $x \in \Sigma$, where I is some input domain. The piecewise polynomial implementation is generally accompanied by some claimed error bound, ϵ, which is what we aim to prove. More precisely, we prove that

$$\forall i = 0, 1, ..., K \text{ and } \forall x \in \Sigma_i, \, |f(x) - p_i(x)| < \epsilon.$$

For each i, we will generate at least one problem to be automatically proven by MetiTarski."

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.

Response: "MetiTarski understands functions via axioms that give upper and lower bounds, and in the case of the floor function we simply have $x - 1 < \text{floor}(x) \leq x$." The addition is the rather naive bound for the floor function, which allows us to just write floor(expression) in our MetiTarski problems. Hopefully this explanation provides clarification.

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 $k = min_k|1.significant(x) - c_k|$).

Response: From page 4. "For example, Tang proposes an algorithm to compute $\ln(x)$ for $x \in [1, 2]$, which uses breakpoints $c_k = 1 + k/64$ for $k = 0, 1, ..., 64$. The breakpoint is chosen which satisfies $|x - c_k| < 1/128$ and a reduced argument $r = 2(x - c_k)/(x + c_k)$ is used to compute a polynomial approximation $p(r)$. The final approximation is given by $\ln(x) \approx T_k + p(r)$, where $T_k \approx \ln(c_k)$ are the tabulated values."

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"

Response: All of the above have been corrected, thank you for the attention to detail.

p6 ln 28: is this a change of notation ($\overline{ln}$ is now M_ln)? Why?

Response: The change of notation is intended to highlight the change of approximation method.

p8 line 58-60: is ⇒ intended to be read as implication?

Response: Yes. Added definition, see page 5.

fig 3: what is $\overline{w}$ (opposed to w)?

Response: $\overline{w}$ = NOT(w), clarification added to caption of figure 4 (was figure 3).

Reviewer 2's Comments

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. **Response:** Thank you for the positive feedback.

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.

Response: Added section 5 "Automatically Generated Designs: FloPoCo", which includes the verification of a 52 bit square root implementation. This represents the approximation used on the 1.significant component of a binary64 floating point number.

Several other high-level questions: - I was slightly disappointed that you could do no better than Gappa ten years ago.

Response: Unfortunately due to resource limitations these tests were run on a 9 year old CPU and we also note that single core performance has not improved dramatically in the last 10 years.

- is Mathematica in the trust base?

Response: Added to section 6, page 15. "Given that Mathematica and Z3 are widely used in academia and industry we choose to trust their results."

- do you plan to handle iterative techniques (for division or square root)?

Response: There are no plans to handle iterative techniques as these are less amenable to this type of modelling. MetiTarski has no support for loop constructs, which would mean iterations would have to be unrolled. If decisions are made based on computed results in each iteration then we do not have any way to model that currently.

- do you plan to handle correct rounding?

Response: Assuming correct rounding refers to IEEE rounding modes. There are no plans to build in rounding operators like in Gappa, however modes such as RTNE can be modelled using error bounds so can be handled in the current framework. In general, rounding modes generate finite bitwidth outputs which is difficult since MetiTarski has no understanding of finite precision. This would be interesting to investigate further in future work.

Several low-level questions: - note that there is a more recent IEEE-754 standard, and doubles are now "binary64".

Response: Changed throughout.

- why did you not add binary logarithm to Metitarski?

Response: This has been added to future development work. It was not considered at the time when we were looking at logarithm implementations, but would clearly simplify many of the problems.

- 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

Response: Thank you for the helpful references.

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

Response: Corrections made, thank you for the attention to detail.

Reviewer 3's Comments

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.

Response: Thank you for highlighting this error. We updated the error values throughout section 6 ($2^{-54} \rightarrow 2^{-53}$) and re-tested the proofs using MetiTarski. Fortunately MetiTarski was still able to prove the results in a very similar time. This is perhaps not surprising given that the implementation had already been verified so must be able to tolerate this larger amount of error.

In terms of additional comparisons, we felt that this is beyond the scope of this contribution since we already have included a comparison with Gappa. Thank you to the reviewer for highlighting two additional tools, CoqInterval and FPTaylor. We have added some discussion about these tools in the "Related Work" section (see page 16). Interestingly, the CoqInterval authors have already completed a comparison of MetiTarski against CoqInterval.

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?

Response: The error refinement does not change the degree of the polynomial approximation. It is indeed about charaterising bits that get truncated

and correlation between these. An updated floor function example on page 7 is intended to highlight correlations in the bits that get truncated.

Also, how often is exhaustive testing needed?

Response: Comment added to page 8. "In §4, we discuss a complex industrial implementation where this final exhaustive step was necessary to complete the verification. This is the only case where we required exhaustive testing."

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?

Response: Exhaustive testing was not needed to complete the verification here but to find examples where the error budget was breached. Unfortunately, the algorithm was not fixed before the conclusion of the internship and therefore this work was never completed.

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.

Response: Thank you for taking the time to do some additional experiments. We felt that the comparison conducted by the CoqInterval authors against MetiTarski would be a more thorough investigation than we would be able to conduct given a lack of understanding of Coq. See comments above for more information.

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.

Response: Added discussion in "Related Work" (page 16).

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.

Response: We did not fully understand this comment. However we investigated the reference given and unfortunately the code to generate the implementations now appears to be dead. The new implementations of the square root we verified are from the same family of FloPoCo commands as Detrey's method. There is strong evidence that we are stressing MetiTarski's ability to handle multiple variables in the newly added section 5 since MetiTarski did not

terminate on the 4 variable problem.

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 $z/4 - 1$.

Response: Our reasoning does fall short here. We have now included discussion of a truncation operator as opposed to the floor function operator (see equation 2 page 7). It is another function that it would be beneficial to add support for in MetiTarski, just to simplify the template generation process.

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.

Response: Thank you for highlighting that our example did not convey the correct approach. Indeed considering chunks of bits is clearly a better approach. We have updated the example on page 7 to consider $z >> 1 - z >> 3$, to demonstrate the grouping of bits.

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.

Response: Once again, thank you for highlighting this error, it clearly invalidates the results. See above comments for how this mistake was resolved.

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?

Response: Keeping all 6 error variables would indeed have caused the computation time to explode. Comment added to page 14. "Ideally we would introduce a new MetiTarski variable for each e_i, however MetiTarski is unlikely to terminate when using more than 4 or 5 variables..."

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.

Response: The original authors of the Gappa contribution conducted this analysis. Added comment to page 14. "They also introduce the bound on the input variable $|Y| \geq 2^{-200}$, which is necessary to make sure that the relative errors due to underflow remain bounded."

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.

Response: This would be an interesting avenue to explore and has been referenced as future work, thank you for the suggestion. We believe it's beyond the scope of this contribution though.

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?

Response: This was a mistake and clearly the runtimes are not comparable, thank you for highlighting. New comments on page 15. "With Gappa taking less than a second to generate a proof it clearly demonstrates stronger performance. However, Gappa required 6 additional "hints", with the authors describing the hint writing process as the most time consuming part. By comparison, MetiTarski required just one re-write of a problem in order to prove it, and no re-writes (or "hints") were required in our other examples, so MetiTarski is competing on an uneven playing field."

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."

Response: Indeed this comment is unfair. It was related to how the verification effort is specific to the implementation but that same criticism can be levelled against our own methodology. The comment has been updated (page 16). "Such an approach is labour intensive and much of the labour is specific to the algorithm at hand. Note however, that Harrison's proofs are more detailed, including results about rounding modes."