Towards Verified Handwritten Calculational Proofs
(Short Paper)

Alexandra Mendes[1,2] and João F. Ferreira[1,3]

[1] School of Computing, Teesside University, UK
[2] HASLab/INESC TEC, Universidade do Minho, Portugal
[3] INESC-ID/IST, University of Lisbon, Portugal

Abstract. Despite great advances in computer-assisted proof systems, writing formal proofs using a traditional computer is still challenging due to mouse-and-keyboard interaction. This leads to scientists often resorting to pen and paper to write their proofs. However, when handwriting a proof, there is no formal guarantee that the proof is correct. In this paper we address this issue and present the initial steps towards a system that allows users to handwrite proofs using a pen-based device and that communicates with an external theorem prover to support the users throughout the proof writing process. We focus on calculational proofs, whereby a theorem is proved by a chain of formulae, each transformed in some way into the next. We present the implementation of a proof-of-concept prototype that can formally verify handwritten calculational proofs without the need to learn the specific syntax of theorem provers.

Keywords: handwritten mathematics; interactive theorem proving; mathematical proof; calculational method; handwriting

1 Introduction

Mathematical proof is at the core of many scientific disciplines, but the development of correct mathematical proofs is still a challenging activity. In recent years, there have been great advances in computer-assisted proof systems that support the development of formally verified proofs (e.g. Isabelle/HOL [25] and Coq [8]). However, writing proofs using a traditional computer poses difficulties due to mouse-and-keyboard interaction. That is why scientists often resort to pen and paper to support them in their thinking process and to record their proofs. The problem is that when handwriting a proof, there is no formal guarantee that the proof is correct.

To formally verify a handwritten proof, one has to translate it into a theorem prover's language. This process takes considerable time and effort and requires a good knowledge of the theorem prover's syntax. This makes the writing of verified proofs feel unnatural and difficult, further encouraging scientists to use the pen and paper approach.

To the best of our knowledge, the problem of formally verifying handwritten proofs is still open. In this paper, we present a proof-of-concept research prototype that attempts to bridge the gap between the natural mathematical practice of handwriting proofs and their mechanical verification. This is the first step towards a system for pen-based devices that allows users to handwrite proofs and that communicates with an external theorem prover to support the users throughout the proof writing process. We focus on calculational proofs [3], whereby a theorem is proved by a chain of formulae, each transformed in some way into the next. We used the method of rapid prototyping [29] to demonstrate the feasibility of the system. Since innovations communicated verbally can be difficult to imagine, a prototype can give the prospective users a better sense of what can be achieved as well as giving us proof that our goal is attainable.

In the next sections, we present some background and related work (§2), summarise requirements taken from existing literature (§3), present a proof-of-concept prototype (§4), and conclude by discussing the next steps (§5).

2 Background and Related Work

This work is being developed in the context of teaching and research on correct-by-construction program design. Starting with the pioneering work of Dijkstra and Gries [9,14], a calculational method emerged, emphasising the use of systematic mathematical calculation in the design of algorithms. Proofs written in the calculational format consist of a chain of formulae, each transformed in some way into the next, with each step optionally accompanied by a hint justifying the validity of that step (see Figure 1(**c**) for an example).

Calculational proofs are known for their readability and for helping to avoid mistakes, but errors can still occur. Indeed, the need for mechanical verification of calculational proofs has been widely recognised. For example, in [22], the authors point out errors in some of Dijkstra's calculations and send a clear message to the calculational community: *"If your proofs are so rigorous and so amenable to mechanization, stop just saying so and do it"*. However, they question how hard it would be to learn and use a proof checker and whether transforming proofs for mechanical checking would make them ugly and hard to understand.

Some work has been done towards mechanised calculational proofs. For example, Leino and Polikarpova [20] extended Dafny [19] to support proof calculations. The authors state that *"It would be wonderful if we could just take a pen-and-paper calculational proof and get it machine-checked completely automatically"*, further supporting the need for verified handwritten proofs. Also, Tesson et al. [28] design and implement a set of tactics for the Coq proof assistant to help writing proofs in calculational form.

With the advent of pen-input devices the possibilities to improve on the interaction limitations of traditional computers are enormous, in particular when it comes to mathematical input. These devices enable software tools such as MathBrush [18] and Microsoft's ink math assistant [26], which allow the recognition, evaluation, and manipulation of handwritten mathematical input (for an

extended list of pen-based mathematical tools, see [23,24]). Our work differs from these tools on the emphasis and domain of application: while these emphasise the recognition and evaluation of expressions, our focus is on supporting the handwriting, manipulation, and verification of calculational proofs. As far as we know, there is only one system that supports the manipulation of handwritten calculational proofs using pen-based devices: the MST editor [23,24]. This editor provides structured manipulation of handwritten expressions and provides features to enable flexible and interactive presentations. A limitation, however, is that proofs remain unverified. We attempt to address this limitation by supporting computer-assisted verification of handwritten proofs. Proof assistants such as Isabelle/HOL [25] and Coq [8] can be used to achieve this, but the use of these requires knowledge of the theorem prover's syntax and its intricacies, which has the reputation of being a demanding task. Our work attempts to overcome this by providing a system to interface with a theorem prover without any specific knowledge of the backend proof assistant.

The availability of several different IDEs for existing theorem provers indicates that the human-prover interaction is a concern. For Isabelle/HOL alone there are several IDEs available, including Proof General [2], Isabelle/jEdit [31], and Isabelle/Clide [21]. All of these require the use of keyboard and/or mouse and the knowledge of the, often idiosyncratic, theorem prover's syntax. In fact, the community is still investigating how to improve current IDEs, as demonstrated by the development of PIDE [31,32,33], a framework for prover interaction and integration, and by Company-Coq [27], an extension of Proof General's Coq mode. Moreover, the support provided by these IDEs for auto-completion of mathematical symbols suggests that typing these does not come naturally.

3 Requirements

The scope of the requirements presented in this section is limited to the context of our work (teaching and research on calculational methods for correct-by-construction program design) and is mostly based on comments found on research papers written by exponents of the calculational method.

R1: Support for calculational mathematics. Given the context of our work, the system should allow the user to write calculational proofs. It should be easy to input mathematical symbols and unconventional mathematical formulae (e.g. the Eindhoven quantifier notation [5]).

R2: Support for structure editing. Similarly to Math∫pad and MST [6,24], the system should provide structure editing operations to assist the user in effectively writing handwritten calculational proofs and to ensure that human errors are less likely to be introduced. Frequently used structural operations like selection and copy of expressions and sub-expressions, group/ungroup of sub-expressions, and distributivity operations should be supported.

R3: Support for handwritten input. It should be possible to handwrite calculational proofs as one would normally do when using pen and paper. Moreover, as identified in [24], the result of structure editing rules on handwritten

expressions should remain handwritten, since it is undesirable to mix different writing and font styles, as doing so, can make presentations confusing.

R4: Support for learning, teaching and research. The system should support learning, teaching, and research on calculational methods for correct-by-construction program design. It has already been argued that calculational proofs offer some pedagogic advantages over conventional informal proofs [12,13,15] and that students prefer or understand better calculational proofs [10]. Moreover, a structure editor can assist students and teachers in learning and explaining how certain rules are applied [23]. A structure editor can also assist researchers who use the calculational method, since they usually write calculations that involve a great deal of syntactic manipulations of uninterpreted and unconventional mathematical formulae (e.g. [4,11,16]). It is desirable for the system to reduce the cognitive load of its users by providing intelligent visual hints throughout the proof writing process.

R5: Support for formal verification. The system should allow mechanical verification of handwritten calculational proofs. The need for mechanical verification of calculational proofs has been identified many years ago [22,30]. More recently, there has been work on mechanisation of calculational proofs [7,17,20], but the problem of verifying handwritten calculational proofs remains open. Moreover, any provers should be used transparently, i.e. the system should hide all the knowledge required to translate handwritten input into syntax accepted by provers. It is important to note that this includes usability aspects other than just syntax — for example, the system should be able to represent mathematical objects in a way that is appropriate in the context of the proof and in the context of the theorem prover. Similar to the idea put forward by Verhoeven and Backhouse [30], our system could be seen as a user interface to a theorem prover. Ideally, the system should allow expert users to define new interaction methods with the backend provers.

4 Proof-of-Concept Prototype

Our proof-of-concept research prototype is implemented in C# and is based on an extension of Classroom Presenter (CP) [1] that uses the library MST [24]. Reusing these existing tools provides us with a structured editor of handwritten mathematics that immediately meets the requirements **R1**, **R2**, **R3**, and, to a certain extent, **R4** (for example, intelligent visual hints are still missing).

The novelty of our prototype is that it adds preliminary support for verification (**R5**): we added a new tool to CP's toolbar that transforms individual steps of a calculation into lemmas that can be proved by Isabelle/HOL. An example of a proof that can be verified by our system is shown in Figure 1(**c**). The user can select the new tool by clicking on the new toolbar button, and after clicking on a step relation (e.g. equality), the system performs verification of the corresponding individual step. Depending on the validity of the step, the system either shows a green check-mark or a red cross (as illustrated). The proof shown

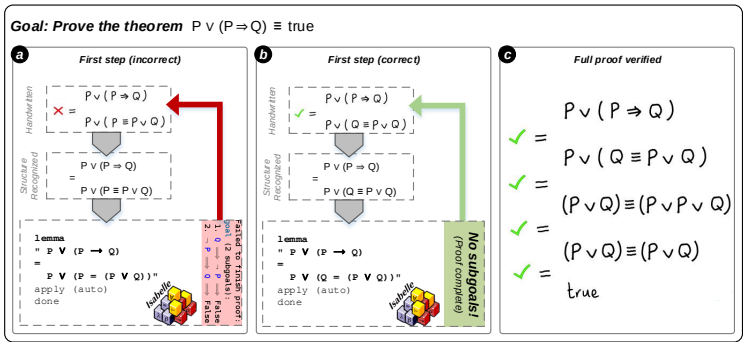

Fig. 1. System overview. Users handwrite calculational proofs and proof steps are translated into Isabelle/HOL to be verified. Invalid steps will be flagged (**a**), allowing users to fix them (**b**). Full proofs can be verified on a step-by-step basis (**c**).

depends on the definition of implication, which can be defined either as

$$P \Rightarrow Q \;\equiv\; P \equiv P \wedge Q \qquad \text{or as} \qquad P \Rightarrow Q \;\equiv\; Q \equiv P \vee Q$$

This means that $P \Rightarrow Q$ can be replaced by either $P \equiv P \wedge Q$ or by $Q \equiv P \vee Q$. However, a common mistake done by students is to swap the conjunction by the disjunction (and vice-versa). This common mistake is illustrated in Figure 1(**a**), where we can also see an overview of the steps taken by our prototype to verify a proof step. Once the structure of the handwritten input is created (using the features available in the MST library), our system converts the recognised structure of the step into a lemma that can be interpreted by Isabelle/HOL. We also attach a proof to each generated lemma. In the current version of the system, we simply instruct Isabelle/HOL to try and prove the goal automatically (achieved by `apply(auto)`). In Figure 1(**a**), Isabelle/HOL fails to prove the step automatically and the user is informed. In Figure 1(**b**), Isabelle/HOL succeeds in verifying the step.

More specifically, the first step of the calculation shown in Figure 1(**b**) would be translated into the following:

```
lemma
 " P \<or> (P \<longrightarrow> Q)
 =
   P \<or> (Q = (P \<or> Q))"
apply (auto)
done
```

Currently, this translation only supports calculational propositional logic operators [9,14], but our code can easily be extended to include a larger mathematical domain (however, other proof tactics may be needed). Other translations are possible and will be explored in future iterations of this work. Communication with Isabelle/HOL is currently performed via an external process that we programmed. This process accepts plain-text input encoding a lemma and its

proof. It then embeds the input into a generated Isabelle/HOL theory file created on the fly, attempts to verify the theory, and returns a modified version of Isabelle/HOL's output to the user interface. The handwritten step is then annotated with either a green check-mark or a red cross, depending on the result of the verification. Figure 2 shows a screenshot of our current prototype in use when writing the proof discussed above. It shows that the first step has already been verified and the user is currently applying the distributivity rule using a gesture (this is one of MST's features that we imported into our system).

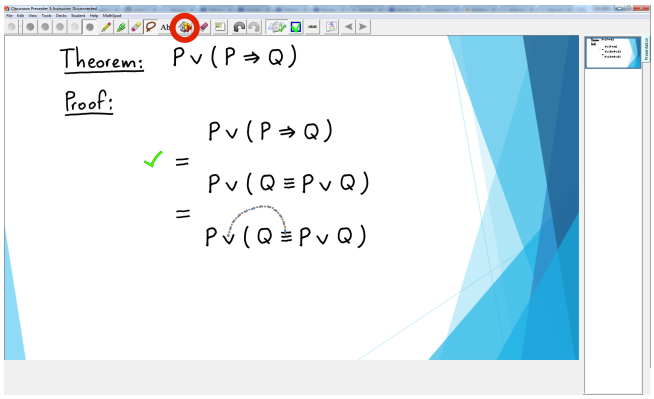

Fig. 2. Screenshot of our prototype in action. The first step of the calculation was verified using the new tool (highlighted button). The user is currently applying the distributivity rule using a gesture.

5 Conclusion and Future Work

We have described our first steps towards a system that allows users to verify handwritten calculational proofs. Our proof-of-concept prototype supports propositional logic proofs and uses Isabelle/HOL as the backend prover. The major novelty of this work lies on the implementation of the requirement for formal verification (**R5**): the prototype can formally verify handwritten calculational proofs without the need to learn how to use a theorem prover. The implementation of the prototype shows that the system we envisage is feasible and that it has the potential to assist its users in writing correct proofs.

Having a prototype will now allow us to demonstrate the potential of such a tool to target users. Our next step will be to demonstrate the prototype in learning, teaching, and research environments to obtain user feedback. This will enable us to understand further requirements of likely users of this tool. Once this step is completed, we will implement a more complete system that will improve the interaction with the backend prover (e.g. feedback from the background proof assistant to include hints for completing proof steps or counter-examples).

For this, we plan to use PIDE [31,32,33]. We also intend to use hints handwritten by users to justify proof steps in the verification process, allowing the detection of inconsistent justifications. The feedback from users will inform the way in which hints will be dealt with. We further plan to support multiple backend provers and to link proofs of programs with the code generation mechanisms available in some theorem provers, such as Isabelle/HOL and Coq. We will take into account the feedback received from users and adapt the system to meet any further requirements that arise. We plan to continue using rapid prototyping to demonstrate any new features to users before providing complete implementations. We anticipate that several iterations of rapid prototyping and evaluations will be needed before we complete the first full implementation.

References

1. Anderson, R., Anderson, R., Chung, O., Davis, K.M., Davis, P., Prince, C., Razmov, V., Simon, B.: Classroom Presenter - a classroom interaction system for active and collaborative learning. In: WIPTE (2006)
2. Aspinall, D.: Proof General: A generic tool for proof development. In: International Conference on Tools and Algorithms for the Construction and Analysis of Systems. pp. 38–43 (2000)
3. Backhouse, R.: The calculational method. Information Processing Letters **53**(3), 121 (1995)
4. Backhouse, R., Ferreira, J.F.: On Euclid's algorithm and elementary number theory. Sci. Comput. Program. **76**(3), 160–180 (2011). https://doi.org/10.1016/j.scico.2010.05.006, `http://joaoff.com/publications/2010/euclid-alg`
5. Backhouse, R., Michaelis, D.: Exercises in quantifier manipulation. In: International Conference on Mathematics of Program Construction. pp. 69–81. Springer (2006)
6. Backhouse, R., Verhoeven, R.: Math∫pad: A system for on-line preparation of mathematical documents. Software–Concepts and Tools pp. 80–89 (1997)
7. Bauer, G., Wenzel, M.: Calculational reasoning revisited (an Isabelle/Isar experience). In: International Conference on Theorem Proving in Higher Order Logics. pp. 75–90. Springer (2001)
8. Bertot, Y., Castran, P.: Interactive Theorem Proving and Program Development: Coq'Art The Calculus of Inductive Constructions. Springer Publishing Company, Incorporated (2010)
9. Dijkstra, E.W., Scholten, C.S.: Predicate Calculus and Program Semantics. Springer-Verlag New York, Inc., New York, NY, USA (1990)
10. Ferreira, J.F., Mendes, A.: Students' feedback on teaching mathematics through the calculational method. In: 39th ASEE/IEEE Frontiers in Education Conference. IEEE (2009)
11. Ferreira, J.F., Mendes, A.: A calculational approach to path-based properties of the Eisenstein–Stern and Stern–Brocot trees via matrix algebra. Journal of Logical and Algebraic Methods in Programming **85**(5, Part 2), 906 – 920 (2016). https://doi.org/http://dx.doi.org/10.1016/j.jlamp.2015.11.004, `http://www.sciencedirect.com/science/article/pii/S2352220815001418`, articles dedicated to Prof. J. N. Oliveira on the occasion of his 60th birthday

12. Ferreira, J.F., Mendes, A., Backhouse, R., Barbosa, L.S.: Which mathematics for the information society? In: Teaching Formal Methods, LNCS, vol. 5846, pp. 39–56. Springer-Verlag (2009), http://joaoff.com/publications/2009/which-mathis
13. Ferreira, J.F., Mendes, A., Cunha, A., Baquero, C., Silva, P., Barbosa, L., Oliveira, J.: Logic training through algorithmic problem solving. In: Tools for Teaching Logic, Lecture Notes in Computer Science, vol. 6680, pp. 62–69. Springer Berlin Heidelberg (2011), http://dx.doi.org/10.1007/978-3-642-21350-2_8
14. Gries, D., Schneider, F.B.: A Logical Approach to Discrete Math. Springer-Verlag New York, Inc., New York, NY, USA (1993)
15. Gries, D., Schneider, F.B.: Teaching math more effectively, through calculational proofs. The American mathematical monthly **102**(8), 691–697 (1995)
16. Hinze, R.: Scans and convolutions: a calculational proof of Moessner's theorem. In: Symposium on Implementation and Application of Functional Languages. pp. 1–24. Springer (2008)
17. Kahl, W.: Calculational relation-algebraic proofs in Isabelle/Isar. In: International Conference on Relational Methods in Computer Science. pp. 178–190. Springer (2003)
18. Labahn, G., Lank, E., MacLean, S., Marzouk, M., Tausky, D.: Mathbrush: A system for doing math on pen-based devices. In: Proceedings of the 2008 The Eighth IAPR International Workshop on Document Analysis Systems. pp. 599–606. DAS '08, IEEE Computer Society, Washington, DC, USA (2008). https://doi.org/10.1109/DAS.2008.21, http://dx.doi.org/10.1109/DAS.2008.21
19. Leino, K.R.: Dafny: An automatic program verifier for functional correctness. In: Proceedings of the 16th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning. pp. 348–370. LPAR'10, Springer-Verlag, Berlin, Heidelberg (2010), http://dl.acm.org/citation.cfm?id=1939141.1939161
20. Leino, K.R., Polikarpova, N.: Verified calculations. In: Revised Selected Papers of the 5th International Conference on Verified Software: Theories, Tools, Experiments - Volume 8164. pp. 170–190. VSTTE 2013, Springer-Verlag New York, Inc., New York, NY, USA (2014), http://dx.doi.org/10.1007/978-3-642-54108-7_9
21. Lüth, C., Ring, M.: A web interface for Isabelle: The next generation. In: International Conference on Intelligent Computer Mathematics. pp. 326–329 (2013)
22. Manolios, P., Moore, J.S.: On the desirability of mechanizing calculational proofs. Inf. Process. Lett. **77**(2-4), 173–179 (Feb 2001). https://doi.org/10.1016/S0020-0190(00)00200-3, http://dx.doi.org/10.1016/S0020-0190(00)00200-3
23. Mendes, A.: Structured Editing of Handwritten Mathematics. Ph.D. thesis, School of Computer Science, University of Nottingham (2012)
24. Mendes, A., Backhouse, R., Ferreira, J.F.: Structure editing of handwritten mathematics: Improving the computer support for the calculational method. In: Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces. pp. 139–148. ITS '14, ACM, New York, NY, USA (2014). https://doi.org/10.1145/2669485.2669495, http://doi.acm.org/10.1145/2669485.2669495
25. Nipkow, T., Wenzel, M., Paulson, L.C.: Isabelle/HOL: A Proof Assistant for Higher-order Logic. Springer-Verlag, Berlin, Heidelberg (2016), https://isabelle.in.tum.de/doc/prog-prove.pdf
26. Microsoft OneNote. https://www.onenote.com, accessed: 02-02-2018
27. Pit-Claudel, C., Courtieu, P.: Company-Coq: Taking Proof General one step closer to a real IDE. In: CoqPL'16: International Workshop on Coq for Programming Languages (2016)

28. Tesson, J., Hashimoto, H., Hu, Z., Loulergue, F., Takeichi, M.: Program calculation in coq. In: International Conference on Algebraic Methodology and Software Technology. pp. 163–179. Springer (2010)
29. Tripp, S.D., Bichelmeyer, B.: Rapid prototyping: An alternative instructional design strategy. Educational Technology Research and Development **38**(1), 31–44 (1990)
30. Verhoeven, R., Backhouse, R.: Interfacing program construction and verification. In: Proceedings of the World Congress on Formal Methods in the Development of Computing Systems-Volume II. pp. 1128–1146. FM '99, Springer-Verlag, London, UK, UK (1999), http://dl.acm.org/citation.cfm?id=647545.730778
31. Wenzel, M.: Isabelle/jEdit–a prover IDE within the PIDE framework. In: International Conference on Intelligent Computer Mathematics. pp. 468–471. Springer (2012)
32. Wenzel, M.: Asynchronous user interaction and tool integration in Isabelle/PIDE. In: International Conference on Interactive Theorem Proving. pp. 515–530. Springer (2014)
33. Wenzel, M., Wolff, B.: Isabelle/PIDE as platform for educational tools. arXiv preprint arXiv:1202.4835 (2012)