Towards user-friendliness in proof assistants: an effect-based attempt

APRIL GONÇALVES, Metastate AG

Proof assistants provide a framework for modelling and verification of theories, as well as trust-worthy software. However, such power is usually only available to experts. We propose a new approach, based on algebraic effects and handlers, to integrate different automated proof strategies that enables newcomers to take advantage of proof assistants without a more in-depth understanding of underlying theory. Our approach gives beginner users an effect system, a handful of effectful strategies (tactics, proof search and a SMT solver) and their handlers under a shared interface, while advanced users can extend our system with new effects and new handlers. Lastly, we prototype the system as a library in Agda. While our prototype is minimal, it shows how easily proofs can be carried on so long as the user has the correct intuition. We believe our system empowers non-experts and has the potential to bring verified software to many relevant industries, such as finance.

Additional Key Words and Phrases: effect handlers, proof assistants, user experience

ACM Reference Format:
April Gonçalves. 2018. Towards user-friendliness in proof assistants: an effect-based attempt. 1, 1 (September 2018), 2 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

The usability of proof assistants such as Coq, Agda, Isabelle/HOL and Twelf is hard to measure: all of them require significant domain knowledge that frequently matches a deeper understanding of Type Theory, the theory that enables such assistants to exist. Coq and Isabelle/HOL are more commonly applied outside their niche, and their programming style is sharply different from mainstream programming and even functional programming. Granted, they come with their own integrated development environment (IDE), which may make it easier for students and newcomers. To our knowledge, no usability study has been held for a proof assistant, however there exists studies that account for an informal "usability" criteria – they appear to measure how fast users can familiarise themselves with the system or IDE.

Folklore within the proof assistants community seems to show that typical users find Coq hard to use due to a large library of tactics and difficult readability of the code after its completion. Many Coq learning materials suggest users to add comments for structural induction cases and inductive hypothesis. As mentioned, no user studies were conducted to attest such hypotheses.

It also has to be granted that proof assistants are relatively new technology and there are no guidelines or even an intuition on how such systems should perform or what features should be available to the users. The closest model we have are proofs by hand, however, following that model for proof assistants has two main pitfalls: (a) proofs by hand are also not widely taught in mathematics classes during school years; and (b) proofs by hand and automated proof writing

Author's address: April Gonçalves, april@cyberglot.me, Metastate AG, Edinburgh, UK.

XXXX-XXXX/2018/9-ART $15.00
https://doi.org/10.1145/1122445.1122456

employ fundamentally different paradigms, where the latter requires axioms and formulations to be encoded explicitly, commonly known as a "pedantic proof style".

1.1 Empowering users of different levels of expertise

To popularise software verification, we mush make proof writing less of a burden to the user. Currently, only specialists can write proofs for their programs, even though the working programmer understands the domain and invariants of their projects – otherwise they wouldn't be able to write any software, but usually the learning curve of automated formal proofs is steep and considered too expensive.

In our approach, we propose an (algebraic) effects and handlers view of such proofs, based on prior work developed by the Andromeda proof assistant. Here, our users will program as they would normally and invoke a proof environment as an effect to prove certain properties as they go. Given that the proof environment is "just" an effect, we envision that different proof styles (e.g, agda-style dependent types, SMT solver, proof search) can be "composed" under a shared interface, a proof object that can manipulate itself while querying different automated proof engines.

The reasons we employ algebraic effects and handlers are two-fold:

(1) as proofs cannot be fully automated, all approaches that try to automate the process (e.g, proof search, SMT solver) may either be non-deterministic or never find a solution. Therefore, the system should be able to handle "impure" computations and errors. Algebraic effects and handlers have well-defined semantics and provide a simple interface for performing effects. With them, we avoid indiscriminate effects that are often error-prone and intricate effects machinery such as monad transformers.