A formalization of the λ-Y calculus
Project Proposal

Samuel Balco

Background

The fact that mathematical proofs and computer programs are very much alike, as the Curry–Howard correspondence clearly shows, naturally gives rise to theorem provers. In general, a theorem prover accepts a computer program encoding a mathematical proof and executes it to assert it's validity. There are multiple reasons why formalizing a mathematical theory in a theorem prover might be useful. Firstly, formalizing a theory in a theorem prover requires extremely rigorous reasoning. Whilst mathematical proofs in papers can gloss over the "trivial" parts of a proof, a fully formal proof in a theorem prover does not allow for any "hand-waving". The use of a theorem prover thus prevents mistakes and guarantees correctness of proofs (up to the correctness of the proof checker of course). However, this rigorousness often carries a penalty in the length of the mechanized proofs, especially when compared to proofs produced by humans. Here, automation comes to rescue, with so called "tactics" and proof search methods employed by modern theorem provers. Both tactics and proof search methods are programs which try to apply multiple reasoning steps to find the proof of the given theorem. These programs can reduce the explicit size of mechanized proofs, especially for the kind of proofs humans consider trivial. In fact, today, these tools are powerful enough to automatically find proofs for the hidden intuitive reasoning humans take between each step of a human-readable formal proof, allowing the users to write such human readable proofs in a completely formal setting of a theorem prover (i.e. the Isar proof format (Wenzel 1999)). Automatic theorem provers have also managed to automatically find proofs for open problems which eluded human mathematicians, such as the Robbins conjecture (Mccune 1997). This project aims to explore different theorem provers, using the λ-Y calculus as a case study.

The λ-Y calculus can be obtained from the simply typed λ-calculus by the addition of fix-point combinators Y_σ at each type σ. Formalizations of both the untyped and simply-typed λ-calculus and associated results, such as the Church-Rosser theorem, exist in different programming languages/theorem provers, such as the Nipkow (2001) λ-calculus formalization in Isabelle, a formalization of the untyped version of the calculus in Coq by Huet (1994), a simply-typed version in Coq by Koprowski (2006) or more recent encodings in Agda (Mazzoli 2013, Sakaguchi (2012)), amongst others. However, no such formalization exists for the λ-Y calculus (as far as I know).
Whilst formalizations of PCF (Programming language for Computable Functions), a functional programming language closely related to the λ-Y calculus, exist[1], LCF differs from the λ-Y calculus in certain aspects. In the case of lazyPCF, which is restricted to a lazy evaluation/reduction strategy, proof of Church Rosser is impossible. Other results, such as the decidability of normalization are unprovable for PCF, since it includes arithmetic.
Since the λ-Y calculus has been previously used in research (e.g. Salvati and Walukiewicz (2015) or Tsukada and Ong (2014)), a formalization of the λ-Y calculus might be useful not only for the scope of this project, but also as a starting point for anyone wanting to use or extend the calculus in their work. It is also worth noting that whilst there has been some effort in collecting and analyzing different formalisms of the λ-calculus (using various representations in different theorem provers), namely the POPLmark challenge[2], there seems to be no report summarizing the results. Even though it might be interesting to try to analyze the formalisms collected in the POPLmark challenge or to simply collect and analyze other formalized versions of the λ-calculus found online, this project rather aims to formalize the λ(-Y) calculus anew. The reason for this is to evaluate the current state-of-the-art theorem provers, as some formalisms in the POPLmark (and elsewhere) are quite outdated, and to try to maintain a more uniform set of formalizations for an easier direct comparison.

[1]e.g. lazyPCF by Jill Seaman and Amy Felty: http://www.lix.polytechnique.fr/coq/pylons/contribs/view/lazyPCF/v8.4
[2]https://www.seas.upenn.edu/~plclub/poplmark/

Aim

The aim of this project is to explore interactive theorem provers, such as Isabelle and Coq, and dependently typed programming languages, like Agda or Idris, which were developed to aid the mathematician and/or programmer in formalizing mathematical proofs and proving correctness of programs. I will attempt to formalize the λ-Y calculus and the same set of theorems up to the proof of Church-Rosser using Isabelle and other theorem provers, such as Coq, Agda or Idris. These formalizations will be used to compare and contrast the different strengths and weaknesses of each language, providing a general overview of the state of the art in theorem provers.
As a possible extension to this work, formalizing other interesting properties of the λ-Y calculus, such as proving the decidability of normalizability, might be attempted.

Proposed Methodology

Formalizing the λ-Y calculus will involve formalizing the syntax of the λ-Y calculus, β-reduction and $\beta\eta Y$-reduction. The terms of the λ-Y calculus will first be formalized using nominal sets in Isabelle, followed by a De Bruin/nameless representation in Isabelle and at least one dependently typed language (e.g. Coq, Agda, etc.). Each such formalization will be accompanied by the Church Rosser proof.

Comparing formalizations of the λ-Y calculus in multiple language and/or different formalisms (i.e. using nominal sets or De Bruin indices to deal with the lambda binding and substitution) will involve comparing objective measures, such as the length of the encoded calculus theory files, as well as more subjective measures, like code clarity and the ease of use of individual programming languages/tool sets. Focus will be placed on aspects of automation of each theorem prover, for example, by comparing the power of automatic provers/tactics in each language on simple (read, trivial to a human) lemmas about the λ-Y calculus. Such lemmas might include simple properties about the λ-Y calculus, such as the *substitution lemma*. Whilst short proofs should be a priority for a theorem prover, the length of the code may adversely affect the subjective measures, such as the readability of proofs. For example, using De Bruin indices for encoding the λ-terms is not very human-readable, but allows for much simpler definition of substitution, which can also make proofs involving substitution shorter. The length of proofs might also depend on the underlying theory and lemmas/proof tactics that are be available in the theorem prover. Another subjective aspect will involve comparing the different proof formats used by the different theorem provers. Whilst Isabelle strives for human readability through the use of the Isar proof format, whereas Coq, for example, uses a more "programmatic" tactic language L_{tac}, both approaches may have advantages and disadvantages in different scenarios.

Timetable

The following points outline the milestones for this project, with rough estimates of the time needed for each milestone. The project work will be split up into roughly 10 weeks (until the beginning of July) of working on the different formalizations and about 8 weeks of writing up the conclusions in the dissertation.

- **Formalize the λ-Y calculus using Nominal Isabelle - 1.5 weeks**
  Having previous experience with Isabelle and having experimented with Nominal Isabelle before, this initial formalization of the λ-Y calculus terms, the notion of reduction and proofs up to Church-Rosser should be fairly straight-forward and serve as a basis and benchmark for other formalisms.
- **Formalize the λ-Y calculus using De Bruin/nameless representation - 2 weeks**
  Repeat the proofs done using nominal sets to obtain a first comparison between different formalizations of the λ-calculus. This formalization is a stepping stone to implementing the λ-Y calculus in languages such as Agda or Idris, where the formalization of nominal sets might not be available (there is a formalization of nominal sets in Agda by Professor Andrew Pitts[3], however, no such formalization was found in Idris).

[3]https://www.cl.cam.ac.uk/~amp12/agda/choudhury/

- **Implement the λ-Y calculus in one or more of the following languages: Coq, Agda, Idris - 5.5 weeks**
  As was discussed previously, the aim of the project is to explore different formalizations of the λ-Y calculus and provide an overview of the current state of the art theorem provers. Since I am only familiar with Isabelle, formalizations in languages like Coq or Agda will likely take much longer to implement. Thus, most time will probably be spent on this milestone.

- **Construct and verify a decision procedure for normalizability of λ-Y terms (optional depending on the time left)**
  Depending on how much time is taken up implementing the basic λ-Y calculus (i.e. the terms + $\beta/\beta\eta Y$ reduction + Church Rosser) in different languages, if there is enough time, one (or more) of the formalizations could be augmented with further results, chief of which would be the decidability of normalization proof.

- **Dissertation write up - 8 weeks**
  The aim is to finish all the formalizations of the λ-Y calculus by the beginning of July, leaving roughly 8 weeks for the writeup only. Some parts of the dissertation, like the literature review might be done alongside the formalizations during the first 10 weeks. The first draft of the dissertation will be submitted by mid-August.

References

Huet, Gérard. 1994. "Residual Theory in λ-Calculus: A Formal Development." *Journal of Functional Programming* 4 (03): 371–94. doi:10.1017/S0956796800001106.

Koprowski, A. 2006. "Coq Formalization of the Higher-Order Recursive Path Ordering." CSR-06-21. Eindhoven, The Netherlands: tue.

Mazzoli, Francesco. 2013. "Agda by Example: λ-Calculus." Blog. http://mazzo.li/posts/Lambda.html.

Mccune, William. 1997. "Solution of the Robbins Problem." *Journal of Automated Reasoning* 19 (3): 263–76. doi:10.1023/A:1005843212881.

Nipkow, Tobias. 2001. "More Church&Rosser Proofs." *J. Autom. Reason.* 26 (1). Secaucus, NJ, USA: Springer-Verlag New York, Inc.: 51–66. doi:10.1023/A:1006496715975.

Sakaguchi, Kazuhiko. 2012. "A Formalization of Typed and Untyped λ-Calculi in SSReflect-Coq and Agda2." Github. https://github.com/pi8027/lambda-calculus.

Salvati, Sylvain, and Igor Walukiewicz. 2015. "Using Models to Model-Check Recursive Schemes." *Logical Methods in Computer Science* 11 (2). doi:10.2168/LMCS-11(2:7)2015.

Tsukada, Takeshi, and C.-H. Luke Ong. 2014. "Compositional Higher-Order Model Checking via ω-Regular Games over Böhm Trees." In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 78:1–78:10. CSL-LICS '14. New York, NY, USA: ACM. doi:10.1145/2603088.2603133.

Wenzel, Markus. 1999. "Isar - a Generic Interpretative Approach to Readable Formal Proof Documents." In *Proceedings of the 12th International Conference on Theorem Proving in Higher Order Logics*, 167–84. TPHOLs '99. London, UK, UK: Springer-Verlag. http://dl.acm.org/citation.cfm?id=646526.694887.