CS 703 Project Proposal

Andrey Yao, Gabe Orlanski

October 26, 2023

1 Introduction

The Curry-Howard correspondence is a well-known connection between type systems in programming languages and certain sorts of logics. For example, propositions in intuitionistic propositional logic can be interpreted as the types in simply typed lambda calculus (STLC). Similarly, proof trees for intuitionistic logic correspond to typing derivations in STLC, and finding an inhabitant of an STLC type gives a corresponding proof term in intuitionistic logic.

Recent research in program synthesis expands on the Curry-Howard correspondence by translating problems in type-directed synthesis into a more familiar problem of proof search in intuitionistic logic. For example, [1] synthesizes terms for a given type in ML(metalanguage) with regards to an external library by translating the ML types into a fragment of first order intuitionistic logic. Similarly, this paper [3] combines type-based program synthesis with input-output examples. They also use a synthesis approach inspired by proof searching techniques.

While the Curry-Howard correspondence is traditionally framed as a connection between types and intuitionstic logic, [2] provides a connection between classical logic and a version of typed scheme, with a primitive `call/cc` operator.

For this project, we plan to build a synthesizer for programs in a minimal language similar to a typed scheme with `call/cc`. We will use a similar deductive, type-directed approach to prune the search space and give certain correctness guarantees of synthesized programs. More specifically, given a type in a typed scheme, as well as a typing context, we will translate the type into a formula in classical logic, call an external solver to find a proof of the formula if it exists, and then translate the proof back to a (well-typed) program in typed scheme. If time permits, we could add constraints to the program specification, or to provide input-output examples to the synthesizer as extra constraints.

2 Roadmap

1. Formal specification of the typed scheme
2. An implementation of the parser and interpreter
3. Translator from types to classical logic formulas

4.

References

[1] Marcin Benke, Aleksy Schubert, and Daria Walukiewicz-Chrzaszcz. “Synthesis of Functional Programs with Help of First-Order Intuitionistic Logic”. In: *1st International Conference on Formal Structures for Computation and Deduction (FSCD 2016)*. Ed. by Delia Kesner and Brigitte Pientka. Vol. 52. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2016, 12:1–12:16. ISBN: 978-3-95977-010-1. DOI: `10.4230/LIPIcs.FSCD.2016.12`. (Visited on 10/23/2023).

[2] Timothy G. Griffin. “A Formulae-as-Type Notion of Control”. In: *Proceedings of the 17th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. POPL ’90. New York, NY, USA: Association for Computing Machinery, Dec. 1989, pp. 47–58. ISBN: 978-0-89791-343-0. DOI: `10.1145/96709.96714`. (Visited on 09/28/2023).

[3] Peter-Michael Osera and Steve Zdancewic. “Type-and-Example-Directed Program Synthesis”. In: *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*. PLDI ’15. New York, NY, USA: Association for Computing Machinery, June 2015, pp. 619–630. ISBN: 978-1-4503-3468-6. DOI: `10.1145/2737924.2738007`. (Visited on 10/23/2023).