CPSC 448 Directed Studies Proposal

Implementing Type-Preserving Compilation for Dependently-Typed Languages, with the A-Normal Form Transformation and Typed Closure Conversion

RAMON RAKOW, University of British Columbia, Canada

Work Period: May 1, 2019 - Aug 25, 2019
Supervisor: Dr. William J. Bowman

1 INTRODUCTION

Coq is a dependently-typed programming language which allows programmers to write proven-correct programs. This is useful for domains in which the cost of a bug could be very high (e.g. embedded systems, financial systems, cryptography), or for building a sturdy foundation for proven-correct programs in other languages and domains (e.g., the CompCert C compiler). However, by default the correctness of a program in Coq is proven only in the semantic context of the high-level language. To prove that those semantics are preserved through compilation down to assembly requires verified compilation, which is the goal of the CertiCoq project– but even this is not enough. To prove those semantics are also preserved through linking (with possibly unverified code) requires type-preserving compilation, which preserves the type-specifications of the high-level language through all the stages of compilation up to linking so that they may be used to verify correct linking.

This is not a trivial task when it comes to a dependently-typed programming language such as Coq. The standard type-preserving translation to continuation-passing style (CPS) by double negation has been shown to be impossible in the context of a language with dependent types, and while more esoteric methods of translation to CPS can preserve dependent types, they are toilsome and have many drawbacks (e.g. only work for limited subsets of the language).

Another problem is how to reason about closures. In a simply typed language, a function could bind free term variables in its body to values from its environment, but in a dependently typed language those free term variables could be involved in the function's type as well. So in a simply typed language, two closures of the same function over different environments will have the same type. But in a dependently typed language, the type of a closure of a function could depend on the environment it is closed with, so two closures of the same function with different environments could have different types.

Luckily, my supervisor Dr. William J. Bowman wrote his dissertation on this topic and has mapped out some very promising paths forward. His paper "Compiling Dependent Types Without Continuations" [3] describes how an A-normal form (ANF) translation can be a type-preserving alternative to CPS, and proves that it works for the Extended Calculus of Constructions (ECC, a feature-rich dependently-typed language). Another paper, "Typed Closure Conversion for the Calculus of Constructions" [2], presents a type-preserving closure conversion translation for the Calculus of Constructions (CC) language on which Coq is based.

Author's address: Ramon Rakow, University of British Columbia, Vancouver, Canada, ramon.rakow@alumni.ubc.ca.

2 PROPOSED WORK

My directed study would consist of three stages: background research, implementing the type-preserving ANF translation, and implementing the closure conversion translation. Both of the latter two stages would deliver programs, written in Coq, which translate source code in a dependently typed language (either the CC or ECC), to a transformed, and semantically correct, target language with type information intact.

2.1 Background: May 1 – May 15

The goals of the background stage would be to fully understand the two papers I would be working on implementing, to become proficient with the Coq language I would be implementing them in, and to get up to speed on type theory notation and foundations. I would use the textbook "Software Foundations: Logical Foundations" as a reference guide to Coq, while working through the two papers written by my supervisor in order to understand them deeply. I would also consult my supervisor's dissertation, "Compiling with Dependent Types" [1], because it seems like it would be pretty helpful.

2.2 ANF Translation: May 16 – July 9

The goal of this stage would be to implement a translator in Coq which takes in source code in the ECC language and spits out target code in ANF-restricted ECC language with machine-like evaluation semantics. My work would also fundamentally be an act of translation, from the formulas and proofs in the "Compiling Dependent Types Without Continuations" paper to Coq source code which can be compiled and executed.

2.3 Closure Conversion: July 10 – August 25

The goal of this stage would be to implement a translator in Coq, from CC with $\sum$-types to a type-safe, dependently-typed intermediate language, called the Closure-Converted Calculus of Constructions (CC-CC). The implementation will again closely follow the plan laid out in the "Typed Closure Conversion for the Calculus of Constructions" paper.

REFERENCES

[1] Bowman, W. J. *Compiling with Dependent Types.* PhD thesis, Northwestern University, 2019.

[2] Bowman, W. J., and Ahmed, A. Typed closure conversion for the calculus of constructions. *Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation - PLDI 2018* (2018).

[3] Bowman, W. J., and Ahmed, A. Compiling dependent types without continuations, 2019.