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 programs alongside machine-checked proofs of their functional correctness. This is useful for high-assurance software (e.g. embedded systems, financial systems, cryptography), or for building a sturdy foundation for writing 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 semantics 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 that those semantics are also preserved through linking (with possibly unverified code, such as an operating system) requires some way to preserve the specifications all the way until the linking stage of the compiler.

If a program component written in Coq is to be compiled into OCaml, then linked with other unverified OCaml code, it's not enough to know that the OCaml translation of just the Coq component is correct, we also have to guarantee that the unverified code fulfills any contracts required of it by our Coq component. To do this, we need some way to remember what those contracts were, all the way until linking. One way to do this is 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 trivial when it comes to a dependently-typed programming language such as Coq. The standard first step in a type-preserving compiler, 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 [1], and while more esoteric methods of translation to CPS can preserve dependent types [5], they are toilsome and come with 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.

Fortunately, 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" [4] describes how an A-normal form (ANF) translation can be a type-preserving alternative to CPS, and proves that it works for the

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

Extended Calculus of Constructions (ECC, a feature-rich dependently-typed language). Another paper, "Typed Closure Conversion for the Calculus of Constructions" [3], presents a type-preserving closure conversion translation for the Calculus of Constructions (CC) language on which Coq is based.

2 PROPOSED WORK

So far, all the work in the above-mentioned two papers is pure theory. The goal of my directed study would be to implement (in Coq) the translations described by the papers, and then machine-verify that these translations are correct. My work would also fundamentally be an act of translation, from the formulas and proofs in the two papers, to Coq source code which can be compiled, executed, and have proofs about it machine-verified. If successful, this would also represent a replication of my supervisor's results.

The work period would progress in three stages: background research, implementing and verifying the type-preserving ANF translation, and implementing and verifying 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 Calculus of Constructions (CC) or Extended Calculus of Constructions (ECC)– to transformed and semantically correct intermediate code in a target language, with type information intact. Because these programs will be written in Coq, they will also be accompanied by proofs they preserve semantics and types.

2.1 Background: May 1 – May 15

The goals of the background stage are to fully understand the above two papers, 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 Bowman's two papers in order to understand them deeply. I would also consult his dissertation, "Compiling with Dependent Types" [2] for additional help.

2.2 ANF Translation: May 16 – July 9

The goal of this stage is to implement a translator in Coq which takes in source code in the ECC language and produces target code in ANF-restricted ECC language with machine-like evaluation semantics, following the theory laid out in "Compiling Dependent Types Without Continuations".

2.3 Closure Conversion: July 10 – August 25

The goal of this stage is 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] Barthe, G., and Uustalu, T. Cps translating inductive and coinductive types. *SIGPLAN Not. 37*, 3 (Jan. 2002), 131–142.

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

[3] 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).

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

[5] Bowman, W. J., Cong, Y., Rioux, N., and Ahmed, A. Type-preserving cps translation of Σ and Π types is not not possible. *Proc. ACM Program. Lang. 2*, POPL (Dec. 2017), 22:1–22:33.