BrainCoqulus: A Formally Verified Optimizing Compiler of Lambda Calculus to Brainfuck

Thomas Lively
Harvard University

Víctor Domene
Harvard University

Gabriel Guimaraes
Harvard University

Abstract

We investigate compilation and verification techniques for functional language compilers by developing and verifying a toy optimizing compiler from the untyped lambda calculus to Brainfuck. Our key optimization is *provisional type inference*, in which the compiler guesses the latent type of a lambda calculus subterm and produces optimized Brainfuck code for that subterm, falling back on the naive slow path code in contexts where that guess is incorrect. Provisional type inference allows the compiler to be extended with additional optimized Brainfuck implementation of common idioms and data structures with minimal additional verification burden.

1 Introduction

BrainCoqulus is a toy compiler from the lambda calculus to Brainfuck composed of layered translations between intermediate langauages, similar to other formally verified compilers such as CompCert [1]. Unlike CompCert, however, BrainCoqulus' source language is functional, which creates a host of new verification challenges.

We chose λ-calculus and Brainfuck as our source and target languages so we could focus on the challenge of functional to imperative compilation without getting bogged down by complex languages or large instruction sets, and also because it sounded fun. The result will be a fully functional, verified compiler of a slightly modified untyped λ-calculus to Brainfuck with verification-friendly semantics.

2 Verification Property

The trusted compute base of BrainCoqulus includes the implementation of the reference interpreters that serve as the specifications of the compiler's input and output languages. In both of these interpreters, a program's input and output are sequences of natural numbers. To simplify the correctness property of the compiler, a program's input is specified in full before execution, i.e. the program is deterministic. In addition, the program is only considered to have produced output if it terminates. The compiler makes no guarantees about the behavior of programs that do not terminate.

The λ-calculus used as the input language is the untyped call-by-value λ-calculus with the addition of an output operator, $\wedge$, that generally behaves identically to the identity function. The difference is that when the output operator's subterm is α-equivalent to a Church numeral representing n, n is appended to the output sequence. λ-calculus programs are lambda terms that are applied at execution time to the term encoding the input as a list of Church numerals. [1]

To reduce the the impedance mismatch between the input and output languages as much as possible, we use an idealized version of the Brainfuck semantics. While most Brainfuck interpreters use fixed-size cells and many use fixed-size arrays, our reference Brainfuck interpreter uses an infinite array

[1]It would be possible and in some ways more intuitive to have lambda calculus programs evaluate to the encoded list of Church numerals corresponding to their output, so we may change to this model in the future. This would allow us to remove the non-standard output operator.

```
Theorem compile_correct :
  forall (l : lambda) (input output : list nat),
  (exists fuel, interpret_lambda l input fuel = Some output) =>
  (exists fuel, interpret_bf (compile l) input fuel = Some output).
```

Figure 1: Theorem asserting correctness of the compilation process

of natural numbers, or in other words it represents memory as a function $\mathbb{N} \to \mathbb{N}$.

Since both reference interpreters serve as the full specifications of their respective languages, they are implemented in Coq. As such, both are required to provably terminate. Since it is possible to write divergent programs in both λ-calculus and Brainfuck, we introduce a fuel argument to the interpreter functions. On each step of the interpreter, the fuel argument is decreased, and if it reaches zero the interpreter stops and the execution is considered unfinished. Given any λ-calculus or Brainfuck program and an input, there exists some fuel that makes the reference interpreter finish and return the program's output if and only if the program terminates on that input. This leads to the functional correctness property given in Figure 1.

3 Project Schedule

By Friday 4/21, we will have the reference interpreters implemented, a simple layer above Brainfuck implemented and its translation to Brainfuck verified, and the provisional type inference from λ-calculus implemented and verified. This shallow layer will give us experience to understand what a good abstraction would be for our project.

By Friday 4/28, we will have all necessary control flow, as well as higher order functions, built on top of Brainfuck with a verified translation. We expect this layer to be the most challenging one, and more time may be allocated to it.

By Friday 5/5, we will have a verified translation from the provisionally typed λ-calculus to the layer built on the previous week, completing the translation from λ-calculus to Brainfuck.

By Monday 5/8, we will have finished writing our write-up and will have performed an extraction to test our compiler on real programs.

4 Division of Labor

Once the intermediate layers have been defined, implementing and verifying the translations between them will be highly parallelizable. The group should be working on two or three individual translations at any given time. Verification of translations is also highly parallelizable once the implementations are in place, so we will focus on getting implementations done as soon as possible.

We foresee, however, that most of the effort will be spent in the translation step with higher-order functions to the neighbouring layers. For this particular case, we intend to split it into smaller tasks (such as verifying that, say, higher-order functions are correct; verifying control-flow; and so on) so that we can still parallelize work.

5 Risks

The greatest risk to the project is that one of the translation steps is extremely complicated and hard to prove. Since we don't know exactly how to implement the full compilation yet, we may run into roadblocks or have to try multiple approaches before finding one that works.

6 Future Work

Future work could include making the compiler aware of more types of data structures and control flow idioms to further optimize the output. It may also be possible to include optimization passes on the Brainfuck itself, to simplify the code generated from λ-calculus.

References

[1] CompCert. `http://compcert.inria.fr`. Accessed: 2017-04-16.