CST Part II Project – Progress Report

An Observable OCaml

Tianlin Zhang `<tz286@cam.ac.uk>`

2nd February 2018

Supervisor: Stephen Kell
Director of Studies: Cecilia Mascolo
Project Overseers: Hatice Gunes & Robert Watson

The project has progressed mostly smoothly up until now. The core of the project, the compiler from OCaml to C, is largely complete with the ability to compile working executables from a range of OCaml programs.

The tasks completed so far include:

- Setting up existing OCaml compiler code to compile individual files into the Lambda IR, and obtaining said representation;
- Writing a `Typecollect` module to obtain variable types from the OCaml compiler, by iterating through the Lambda IR to find debug Lambda events to obtain types;
- Creating a representation of a subset of C in OCaml, to which the compiler will compile to;
- Translation of core language features of ML languages, including functions, standard types (`int`, `float`, `string`, `bool` and `unit`), simple expressions (`let`, `let rec`, `if`, `for`, and `while`), referred to as Subset 1 in the Proposal;
- Design of a representation for OCaml data structures, including tagged unions, tuples, record types, lists, etc. and translation of language features that use them (data constructors and pattern matching), referred to as Subset 2 in the Proposal;
- Design of a closure representation to allow functions as first-class values, and a closure conversion process to lift locally defined functions and partially applied functions into closures which can be passed as values, referred to as Subset 3 in the Proposal;
- Implementation of polymorphism in functions and data structures with a generic union type, allowing functions to be more generic, also part of Subset 3;
- Creation of a library of test OCaml programs to aid in regression testing the compiler, as well as for the evaluation of compiled executable speed and observability.

I am currently an estimated 1 – 2 weeks behind the proposed schedule because of various difficulties encountered during the project, including:

- The plan for 3 expanding subsets to be implemented sequentially was quickly discovered to be impractical, because of the nature of how different language features interact – it would be difficult to attempt to abstract different language features from each other before implementing them together. While this did not represent a significant hindrance in terms of time overall, it did mean that I couldn't produce tidy milestones as in the Proposal, and it took longer before visible results could be produced from the compiler.
- Originally, the `Typecollect` module looked at the Typed AST of the code but it was quickly discovered that the Lambda IR introduces extra untyped variables, so a workaround had to be developed to type these the same way as polymorphic values until their type was more well known.
- A variety of minor kinks in how polymorphic values interact with functions and closures, which has necessitated a few rewrites of how values are represented.

Despite these difficulties no major setbacks have occurred so the intention is to carry on with the existing timetable, and I am confident that I can catch up with the work required.