An Optimising Compiler from Haskell to Java Bytecode
Progress Report

Keith Collister, kc506
Robinson College
Foobar date

Project Supervisor: Dr. Timothy Jones

Director of Studies: Prof. Alan Mycroft

Overseers: Prof. Simone Teufel
Dr. Andrew Rice

Success Criteria

My project currently does not meet the minimum success criteria established in the proposal: the compiler produces Java bytecode from Haskell source code, but there are bugs that prevent the output from being executed. In addition, no optimisations have yet been implemented.

The main cause of the delay is unexpected dependencies between different source language features. To define simple operations like addition, it is necessary to implement functions, datatypes, and typeclasses: this forced me to implement these features sooner than planned. Currently functions, datatypes, and typeclass definitions have been implemented (with known bugs), and typeclass instance definitions are the remaining critical feature. After implementing these, some planned features (eg. lists) are delivered 'for free' (simply defined as `data [] a = [] | a:[a]`).

Another cause for delay was a period of unexpectedly high workload at the end of Michaelmas term, which stalled development for two weeks.

Current Progress

The compiler contains 4 coarse stages:

1. Frontend: The Haskell source is parsed using an external library. Variables in the Haskell source are renamed to eliminate shadowing, reducing complexity in subsequent stages. Dependency analysis is performed on the source declarations: recursive relations are inferred and the order of compilation of declarations is decided.

2. Typechecking: Type signatures are inferred from the source declarations using the results of the dependency analysis. The Haskell AST is explicitly annotated with type information, and any user-provided type signatures are checked against the inferred signatures.

3. Middle-end: Overloaded functions (eg. `(+) :: Num a => a -> a -> a`) are deoverloaded into 'dictionary-passing' equivalents (`(+) :: Num a -> a -> a -> a`). Typeclasses are replaced with datatype and function declarations.

   The Haskell AST is lowered through two intermediate languages and into Administrative Normal Form. The first IL provides a simple language that optimisation passes can easily process, and the second IL makes (lazy) evaluation and allocation explicit for easier code generation.

4. Code Generation: The final IL is transformed into Java Bytecode. An external library is used to convert a logical description of the bytecode to actual binary data, but I have extended the library to support various JVM features required by the lowering (notably the `invokedynamic` and `lookupswitch` instructions).

New Schedule

Given that the project's schedule was effectively rearranged by implementing features sooner than expected, I've included a new workplan below:

- **21st Jan – 3rd Feb**

  Debug currently failing bytecode, and implement typeclass instances (allows for specifying datatypes to be instances of certain typeclasses). Ensure non-strict semantics are working as expected.

  After this work package, the compiler will be able to transform a reasonably-sized subset of Haskell to executable Java Bytecode.

- **4th Feb – 24th Feb**

  Implement optimisations: currently, I intend to implement a peephole pass, dead store elimination, lambda+let lifting, and strictness analysis.

After the end of these two work packages, I should be on track with the original proposed schedule, and intend to spend the remaining time fixing bugs and starting on my dissertation write-up.