This project aimed to implement an optimising compiler from Haskell to Java Bytecode, in order for me to learn more about the implementation of compilers and experiment with optimisations.

This paragraph tries to emphasise that it was about learning stuff, and that... I learned stuff. Also a bit of targetted plugging to the phrase in the pink book 'show how the work fits into the broad area of surrounding Computer Science'. I think it comes off as a bit clumsy though.

The project was a success. I implemented all of the major stages in a traditional optimising compiler pipeline apart from lexing/parsing: verification (through type checking/inference), lowering into intermediate languages, optimisation transformations, and code generation. In order to implement these stages, I had to extend my existing knowledge of type systems, language design, and compiler design from the associated Part 1B and Part 2 courses.

0.1 Choice of Languages

Haskell is a mature purely functional programming language with lazy evaluation and static typing. These semantics are very different from other popular functional languages such as OCaml and F#: laziness and purity are interesting aspects of the language, and also only lightly covered in the modules on compilers and language design, so Haskell provided a range of fresh ideas.

Java Bytecode (JVB) is the strict, impure 'assembly language' for the Java Virtual Machine (JVM). It sits at a comfortable middle-ground between CISC and RISC instruction sets, with convenient utility instructions but without bloat, which makes it a relatively enjoyable bytecode to work with. As it targets the JVM, JVB also benefits from automatic garbage collection, which made it a desirable target language as it removed the need to implement a form of memory management.

0.2 Related Work

GHC is the industry-leading Haskell compiler, capable of generating high-performance code rivalling C. It takes advantage of purity to aggressively optimise code and can evaluate parts of programs in parallel automatically, without any explicit indication from the programmer.

GHC targets various machine languages and includes an LLVM backend, but doesn't target JVB. There are two actively maintained compilers from dialects of Haskell to JVB that I am aware of: Eta and Frege.

Should I link to these projects as citations rather than footnotes, so I don't accidentally end up with duplicate footnote links throughout the diss? Or just have footnote links here and make sure not to add footnotes later.

Eta is a dialect of Haskell emphasising foreign-function interoperability with Java. The `eta` compiler is a fork of GHC, replacing the code generation stage with one producing JVB.

Frege is another dialect of Haskell which also aims to provide high-quality Java interoperability,

but targets Java rather than bytecode directly. It was developed from scratch and is now written in Frege, as the compiler can bootstrap itself.