Team Permanent Geranium Lake
COMP 97
Prof. Sam Guyer
4 Oct. 2017

Project Proposal

Mission Statement -

Our mission is to make Template Haskell type safe with compile-time type checking of Template Haskell expressions. Template Haskell, as it is, has poor support for type-rich metaprogramming and may generate ill-typed expressions; by adding compile-time type checking, Template Haskell would have better support for typeful programming and be safer to program in.

Problem -

The problem this project solves is the need for a convenient and effective way to type-check Template Haskell expressions at compile. Specifically, the Glasgow Haskell Compiler does not have the capability to handle this type-checking, and other solutions are either difficult to use or limiting towards the language. More details about these issues will be discussed in the Context section. Simply put, the problem is not a particularly new or innovative one, but is rather a problem which does not yet have an elegant solution.

Context -

In this case, the need for this project is not a result of the absence of a solution, but rather due to the inconvenience or limitations of the current solutions. At the time, there are a few ways to modify the system in order to identify types without modifying the compiler, but in many cases these are incredibly cumbersome, and in other cases these solutions to the problem come with other restrictions, limiting flexibility of code in Template Haskell. Modifying the compiler would provide a solution which is both easy to use and works with all the elements of Template Haskell.

Customer -

The customer for the modified compiler would be any student working in Template Haskell, or more generally anybody who is using the Glasgow Haskell Compiler for Template Haskell. It is difficult to speak about owners or shareholders, since GHC is an open source project; however, it should be noted that many large financial institutions, such as the Standard Chartered Bank, rely on Haskell for their banking operations. As a result, there is no particular monetary or such value to the project, but instead the project is supported by the students who need it, and the value of the project is the convenience and added functionality of the compiler. In particular, Carl Cronburg and Matt Ahrens have encountered use cases for this feature.

Challenges -

While we have minimal financial or resource challenges, our project is potentially intellectually demanding. The GHC is a large project and has over 450k lines of code, so it might be difficult to identify files and modules that require modification. Moreover, this project will involve some original research, and it might be difficult to figure out how to implement this feature.

Expertise -

In order to accomplish the project, the group will need a thorough understanding of the Haskell language, knowledge of how types are implemented in compilers, and the ability to read and understand the Glasgow Haskell Compiler. Knowledge type implementation in compilers is understood fairly well because of COMP 105, so most of that is prior knowledge. The ability to read and write Haskell and Template Haskell will simply require practice in the language. It is easy to get an environment for practice, and textbooks on the language exist to assist learning. Understanding GHC itself will involve heavy reading into the code, following learning Haskell. Fortunately, many faculty members and graduate students (including Professors Kathleen Fisher and Norman Ramsey, and Karl, Jared, etc) at Tufts are experts in Haskell, so we can expect a lot of guidance.

Risks -

There are no particular short-term risks to this project, because since the work will be done on an isolated copy of the compiler, any issues can be fixed by starting with a fresh compiler. However, the most significant risk would come with incomplete testing, if there was an uncaught bug. It is difficult to tell exactly what the resulting consequence would be, from programs failing to compile to programs running incorrectly. However, this risk can be mitigated by testing thoroughly and fully understanding exactly what needs to be done before writing code.

Ethical Questions -

There are no particular ethical dilemmas in this case because modifying GHC does not create privacy or security risks, and while this modification adds to the features, and therefore to the possibilities, of the language, it does not make the language significantly more powerful. Simply, this is not a project which appears to have an ethical argument against it.

Goal -

For this project, the desired result is a modified GHC which is capable of type-checking expressions in Template Haskell at compile time. That is, the result should be able to recognize types of all expressions to the furthest point that the semantics define them. An ideal product would be a clear, modular and well-abstracted Template Haskell type checker, which requires little modification of other GHC components; a partial success would be a less elegant and more hacky type checker and/or a type checker which only supports partial type checking of Template Haskell expressions at compile time.