HamSkill: Run Haskell Anywhere with Scala

CS252 Project Proposal

Zayd Hammoudeh
(zayd.hammoudeh@sjsu.edu)

1 Running in the Java Virtual Machine

C is one of the most commonly used languages when the goal is maximum performance. However, C/C++'s "write once, compile anywhere" paradigm limits its portability. In contrast, the near ubiquity of the Java Virtual Machine (JVM) allows it to be "write once, run anywhere."

On many occasions, developers have leveraged the JVM's "run anywhere" capability in order to run allow other languages. Examples include: JRuby for the Ruby programming language [2], Jython for the Python programming language [3], Renjin for the R programming language [4], and Scala [5].

Currently, there is no full implementation of Haskell in the JVM. One Haskell dialect that is runnable in Java is Frege [7].

In this project, I will implement, *HamSkill*, a dialect of Haskell that is runnable in the Java Virtual Machine.

2 Implementation Proposal

This section outlines the overall implementation plan for this project. It is divided into subsections based on the overall themes and ideas.

2.1 Parser

When parsing structured text like software code, one of the key aspects of the overall system is picking a scalable, flexible, and easily expandable parser. For this project, I am using ANTLR (ANother Tool for Language Recognition)

ANTLR grammars do exist for Haskell; an example is [6]. However, as a learning experience and to ensure maximum flexibility, I plan to write my own grammar as well as the accompanying base listener (in Java).

2.2 JVM-Supported Programming Language

When selecting the implementation language, my criteria were: runnable in Java, and maximum similarity to Haskell. I initially considered Python (via Jython) due to its higher order function support and concise style.

In the end, I settled on using Scala for this project due to its syntax being more alignable to that of Haskell; an example of this is Scala's support for a function based pattern matching style. What is more, critical aspects of Haskell (e.g. lazy evaluation, immutability of objects, etc.) has parallels in Scala. One major disadvantage of this decisions is that Scala has a much weaker type inference system than does Haskell or Python. An example of this is described in section 2.6.

While it is my expectation that a fuller implementation of Haskell may be more achievable in Python, it would require substantially more effort to implement the functional programming aspects of Haskell that Scala comes with out of the box. In the end, when doing a programming language conversion, it is exceptionally unlikely that the destination language will perfectly support all aspects of the source language. As such, some degree of compromise is required which is why Hamskill is only a dialect of Haskell and not the real thing.

2.3 Supported Types

HamSkill will only support a select subset of Haskell's available types. The list of planned types are: `Bool`, `Integer` (i.e. bounded), `Char`, and `List` (currently only finite lists are planned, but that may change depending on the speed and complexity of the implementation).

While implementing floating point numbers would not add substantial complexity at a basic level, ensuring that the floating point behavior of *HamSkill* (i.e. Scala) and Haskell are identical is beyond the scope of this project.

2.4 Nested Function Calls

Imperative languages (e.g. Java) are generally more verbose than functional languages; Haskell is no exception to this. Such conciseness introduces significant challenges when writing a parser as the contextual clues become less obvious. For example, 2.4 is a simple line of Haskell code that prints to the screen the result of a function `addTwoNumbs` that takes two integers `x` and `y`.

```
putStrLn $ show $ addTwoNumbs x y
```

Figure 1: Simple Function Call in **Haskell**

Similar code in Java is shown in figure 2.4

```
System.out.println( addTwoNumbs(x, y) ) ;
```

Figure 2: Simple Function Call in **Java**

In the Java syntax, it is much clearer that `addTwoInts` is a function since the parameters are inside parentheses and comma separated. To simplify the parsing for this in Haskell, the Hamskill dialect will require function arguments to be succeeded by double parentheses "`((`" and "`))`". Figure 2.4 shows the Hamskill version of the Haskell code in figure 2.4.

```
putStrLn $ show $ addTwoInts ((x y))
```

Figure 3: Simple Function Call in **Hamskill**

2.5 Defining Scope and Scala Object Name via `module`

A program in Haskell is composed of a set of "`module`" files. `module` serves the role of defining the scope of a function (e.g. public or private) as well as for defining an abstract data type [1]. In Hamskill, I will use the Haskell module to define whether the Scala methods are private (since by default functions are public) as well as the name of the Scala object.

2.6 Partially Applied Functions

Haskell supports partially applied functions. Figure 2.6 shows the `addTwoInts` method with a single argument (i.e. "`5`") being stored in a variable "`addFive`".

```
let addFive = addTwoInts 5
```

Figure 4: Partially Applied addTwoInts Function in **Haskell**

Partially applied functions in Scala has advantages and disadvantages in comparison to Haskell. One of these disadvantages is evident when figures 2.6 and 2.6 are compared. Note that the Scala function requires an underscore ("_") for each missing argument as well as the type for that argument. This makes converting Haskell code to Scala problematic as the function prototype must be fixed and known at conversion time.

```
val addFive = addTwoInts(5, _: Int)
```

Figure 5: Partially Applied addTwoInts Function in **Scala**

To simplify this, Hamskill will require that any partially defined functions are either declared in the same file/module. I will investigate using a predefined list of functions, but this may not be feasible or support will be very limited due to the requirement to define the parameter type.

2.7 Higher Order Support

Scala and Haskell are both functional programming languages; one important consequence of this is that both support higher order functions. Hamskill will support functions as input parameters to functions. If time allows, I will also investigate the ability to return functions from functions. The extent to which this is supported will be dependent on the extent to which partially applied functions are supported as defined in section 2.6.

2.8 Lambda to Anonymous Functions

There is significant similarity between a Lamda function in Haskell and an anonymous function in Scala. One primary difference is that Scala requires the developer to specify the types of the parameters in the anonymous function while Scala does not.

3 Tentative Schedule

If you are satisfied with this plan, I will draft a schedule. I did not want to put too much thought into this until I got your buyoff on the plan as a whole.

Bibliography

[1] A gentle introduction to haskell: Modules. https://www.haskell.org/tutorial/modules.html. (Accessed on 03/01/2016).

[2] Home jruby.org. http://jruby.org/. (Accessed on 02/24/2016).

[3] Platform specific notes. http://www.jython.org/archive/21/platform.html. (Accessed on 02/25/2016).

[4] Renjin.org — about. http://www.renjin.org/about.html. (Accessed on 02/25/2016).

[5] The scala programming language. http://www.scala-lang.org/. (Accessed on 02/25/2016).

[6] Thiago Arrais. haskell-lexer.g. http://eclipsefp.sourceforge.net/repo/net.sf.eclipsefp.haskell.core.jparser/antlr-src/haskell-lexer.g. (Accessed on 03/01/2016).

[7] "Ingo Wechsung". Frege/frege: Frege is a haskell for the jvm. it brings purely functional programing to the java platform. https://github.com/Frege/frege. (Accessed on 02/25/2016).