William Rabenstein

Professor Wilkinson

CMSC 331

15 May 2016

Haskell

The opening to the report establishing Haskell can be summarized as, “There are too many functional languages that are too similar. Lets make on language to unite them all.” As a functional language it inherits principle from lambda calculus, and follows in the footsteps of Lisp, which was invented in the late 1950’s. Developments in functional languages between then and the creation of Haskell are Scheme, a variant of Lisp made in the 1970’s, and ML, a language for theorem proving. Post ML, a type of functional language called lazy functional languages emerged. The difference is that it only makes calls what and when it needs to. By the 1980’s many papers were published advancing the idea, and many seperate researchers were developing their own languages using the principle of lazy functional programming. One of the most developed and influential being Miranda. Eventually Haskell was formed using the following design principles: Lazy, Purely functional, Type Classes, and Ability to Change. Laziness forces being purely functional, because it requires that output from functions are reliable. Being easily changed was a major goal going into the project, but rapidly changing makes supporting a language hard. To fix this Haskell 98 was declared the official eternal version. Although its creator denied being used as a basis for Haskell, Miranda and Haskell have many of the same notations and features. Overall, part of the basis for Haskell’s success as a language is, ironically, that it never became too successful. Having a smaller community of researchers and skilled programmers meant that they could change often and leave some things to the user.

The design principles or features of being; lazy, purely functional, having type classes, and its ease of modification are what separate it from most other languages in use. As stated above, being lazy necessitates that the language be functional. This is because in a lazy language the user has no real idea of when parts of their code will be executed. For example, a benefit brought from being lazy is partial execution lists defined in an infinite manner. You can easily create and infinite array of numbers with [1..], and even use it in a program that goes over the list. As a lazy language it will only create and work with the minimum number of elements in the list to achieve the task set for it. So, as long as you don’t tell it to recurse over the entire array, or ask for the last item, you won’t enter an endless loop.

Type Classes are perhaps the most unique and important difference, a large departure from even Miranda, even though they weren’t initially designed as an intentional feature, but as a solution to a problem. Having type classes, means that every type is treated like a class would be in other languages in that it can inherit. This inheritance is like a Java Interface, in that it defines properties about the type, and what functions it may be used in. Common ones that are easily explained are Elt (can be used as an element in an array), Num (A number), Real (A real number). Note that type classes are always capitalized while variables and functions are not. This means that, for example Integer is an Integral Real Num, while a Double is a Real Fractional Number. This is important because the two types of division integer and fractional, are called from the same operator, but overload so you can only call Integral/Integral or Fractional/Fractional.

The final unique part of Haskell, though I didn’t list is, is Monads. Monads are a way of organizing expressions and wrapping data. The IO () type you will see with many operations is the most common form of this. This allows you to perform seperately, non-purely functional operations, in this case input or output from the system. This is seperate so that any possible side-effects are limited. The idea of monadic IO first appeared in the 1996 report for Haskell 1.3. Monads are an almost direct result of type classes, where you can declare something is a monad that hold a secondary type. For instance the type the getChar, which takes a character from the command line is IO Char. This cannot be used as a parameter for pure functions, as it is something that can change at runtime.

Haskell is a unique language in that it was designed by committee to cover most current, at the time, uses for lazy languages and not one or two people for a specific use. Specifically it was focused on being used for language research, and testing the possibilities of functional languages.

Sources

“A History of Haskell: Being Lazy With Class” Third ACM SIGPLAN History of Programming Languages Conference (HOPL-III), 2007.

This is a paper written by members of the committee that created Haskell on its Creation

Marlow, Simon. *Parallel and Concurrent Programming in Haskell*. O'Reilly Media Inc, 2013.

OSullivan, Bryan, et al. *Real World Haskell*. Southeast University Press, 2010.

“Yet Another Haskell Tutorial.” *Wikibooks*, en.wikibooks.org/wiki/Yet\_Another\_Haskell\_Tutorial.