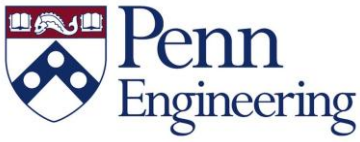
University of Pennsylvania
Department of Computer and Information Science



***About this Project*:** **Implementing Explicit Type Application in Haskell**

A new feature for the Glasgow Haskell Compiler (GHC)

***What it Does for the User*:** **What Does the Feature Do?**

- Explicit Type Application enables the user to specify the types of arguments to generic functions directly.
- This allows GHC to typecheck programs that would otherwise be ambiguous.
- It also improves Haskell's code expressiveness, giving the user the ability to specify clearly specify what types will be instantiated for generic (also known as polymorphic) functions.

***How it is Implemented*:** **Changes to GHC's Parser and Typechecker**

- Adds a new reserved character – '&' – to denote type application to the parser, allowing the type to be parsed into the abstract syntax tree, preparing it for typechecking.
- The typechecker was modified to detect this special node in the abstract syntax tree, and use it as a "flag" to instantiate the type directly, instead of attempting to infer the type.
- Haskell uses an intermediate language, called "Core", to perform optimizations. Core includes explicit type application everywhere by default; thus, once the typechecker is finished, the resulting code automatically fits into Core and is ready for further transformations along the path to machine code.

```
show :: forall a. Show a => a -> String
-- Given an argument that is "Showable," returns a String.
read :: forall b. Read b => String -> b
-- Reads a String and returns a result, which must have a "Readable" type.

does_not_compile = show (read "5")
-- Fails, since (read "5") has type "b", while show wants an "a," and it does not
have enough information to prove that "b" is an instance of "Show," only "Read"

compiles_happily = show (read &Int "5")
-- Succeeds, since it now can deduce "b ~ Int" [type b equals type Int] and Int is
both an instance of "Show" and "Read"
```

Hamidhasan G. Ahmed BSE in Computer Science, '2015
Advisor: Dr. Stephanie Weirich, Professor – Computer and Information Science

Summer 2013