1 Background

Programming Languages fall into different categories also known as ”paradigms”. They exhibit different characteristics according to the paradigm they fall into. It has been argued [?] that rather than classifying a language into a particular paradigm, it is more accurate that a language exhibits a set of characteristics from a number of paradigms. Either way, the broader the scope of a language the more the expressibility or use it has.

Programming Languages that fall into the same family, in our case declarative programming languages, can be of different paradigms and can have very contrasting, conflicting characteristics and behaviours. The two most important ones in the family of declarative languages are the Functional and Logical style of programming.

Functional Programming, [?] gets its name as the fundamental concept is to apply mathematical functions to arguments to get results. A program itself consists of functions and functions only which when applied to arguments produce results without changing the state that is values on variables and so on. Higher order functions allow functions to be passed as arguments to other functions. The roots lie in λ-calculus [?], a formal system in mathematical logic and computer science for expressing computation based on function abstraction and application using variable binding and substitution. It can be thought as the smallest programming language [?], a single rule and a single function definition scheme. In particular there are typed and untyped λ calculi. In the untyped λ calculus functions have no predetermined type whereas typed lambda calculus puts restriction on what sort(type) of data can a function work with. SCHEME is based on the untyped variant while ML and HASKELL are based on typed λ calculus. Most typed λ calculus langauges are based on Hindley-Milner or Damas-Milner or Damas- Hindley-Milner [?] type system. The ability of the type system to give the most general type of a program without any help (annotation). The algorithm [?] works by initially assigning undefined types to all inputs, next check the body of the function for operations that impose type constraints and go on mapping the types of each of the variables, lastly unifying all of the constraints giving the type of the result.

Logical Programming, [?] on the other hand is based on formal logic. A program is a set of rules and formulæ in symbolic logic that are used to derive new formulas from the old ones. This is done until the one which gives the solution is not derived.

The languages to be worked with being HASKELL and PROLOG respectively. Some differences include things like, HASKELL uses Pattern Matching while PROLOG uses Unification, HASKELL is all about functions while PROLOG is on Horn Clause Logic and so on.

PROLOG [?] being one of the most dominant Logic Programming Languages has spawned a number of distributions and is present from academia to industry.

HASKELL is one the most popular [?] functional languages around and is the first language to incorporate Monads [?] for safe *IO*. Monads can be described as composable computation descriptions [?] . Each monad consists of a description of what has action has to be executed, how the action has to be run and how to combine such computations. An action can describe an impure or side-effecting computation, for example, *IO* can be performed outside the language but can be brought together with pure functions inside in a program resulting in a separation and maintaining safety with practicality. HASKELL computes results lazily and is strongly typed.

The languages taken up are contrasting in nature and bringing them onto the same plate is tricky. The differences in typing, execution, working among others lead to an altogether mixed bag of properties.

The selection of languages is not uncommon and this not only the case with HASKELL, PROLOG seems to be the all time favourite for "let's implement PROLOG in the language X for proving it's power and expressibility". The PROLOG language has been partially implemented [?] in other languages like SCHEME [?], LISP [?, ?, ?], JAVA [?, ?], JAVASCRIPT [?] and the list [?] goes on and on.

The technique of embedding is a shallow one, it is as if the embedded language floats over the host. Over time there has been an approach that branches out, which is Paradigm Integration. A lot of work has been done on Unifying the Theories of Programming [?, ?, ?, ?, ?, ?]. All sorts of hybrid languages which have characteristics from more than one paradigm are coming into the mainstream.

Before moving on, let us take a look at some terms related to the content above. To begin with Foreign Function Interfaces (FFI) [?], a mechanism by which a program written in one programming language can make use of services written in another. For example, a function written in C can be called within a program written in HASKELL and vice versa through the FFI mechanism. Currently the HASKELL foreign function interface works only for one language. Another notable example is the Common Foreign Function Interface (CFFI) [?] for LISP which provides fairly complete support for C functions and data. JAVA provides the Java Native Interface(JNI) for the working with other languages. Moreover there are services that provide a common platform for multiple languages to work with each other and run their programs. They can be termed as multi lingual run times which lay down a common layer for languages to use each others functions. An example for this is the Microsoft Common Language Runtime (CLR) [?] which is an implementation of the Common Language Infrastructure (CLI) standard [?].

Another important concept is meta programming [?], which involves writing computer programs that write or manipulate other programs. The language used to write meta programs is known as the meta language while the the language in which the program to be modified is written is the object language. If both of them are the same then the language is said to be reflective. HASKELL programs can be modified using Template HASKELL [?] an extension to the language which provides services to jump between the two types of programs. The abstract syntax trees in the form of HASKELL data types can be modified at compile time which playing with the code and going back and forth.

A specific tool used in meta programming is quasi quotation [?, ?, ?], permits HASKELL expressions and patterns to be constructed using domain specific, programmer-defined concrete syntax. For example, consider a particular application that requires a complex data type. To accommodate the same it has to represented using HASKELL syntax and preforming pattern matching may turn into a tedious task. So having the option of using specific syntax reduces the programmer from this burden and this is where a quasi-quoter comes into the picture. Template HASKELL provides the facilities mentioned above. For example, consider the following code in PROLOG to append two lists,

```
append([], X, X).

append([X$\mid$Xs], Ys, [X$\mid$Zs]) :- append(Xs, Ys, Zs).
```

going through the code, the first rule says that and empty list appended with any list results in the list itself. The second predicate matches the head of the first and the resulting lists and then re-curses on the tails. The same in HASKELL,

```
append(Ps, Qs, Rs) = (Ps = []         $\&$ Qs = Rs) $\parallel$

($\exists$ X, Xs, Ys $\rightarrow$ Ps = [X$\mid$Xs] $\&$

Rs = [X$\mid$Ys] $\&$

append(Xs, Qs, Ys))
```

Consider the Object Functional Programming Language, SCALA [?], it is purely functional but with objects and classes. With the above in mind, coming back to the problem of implementing PROLOG in HASKELL. There have been quite a few attempts to "merge" the two programming languages from different programming paradigms. The attempts fall into two categories as follows,

1. Embedding, where PROLOG is merely translated to the host language HASKELL or a Foreign Function Interface.

2. Paradigm Integration, developing a hybrid programming language that is a Functional Logic Programming Language with a set of characteristics derived from both the participating languages.

The approaches listed above are next in line for discussions.