1 Background

Programming Languages fall into different categories also known as ”paradigms”. They exhibit different characteristics according to the paradigm they fall into. Many a times it has been argued [?] that rather than classifying a language into 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 the family of Declarative Programming Languages, can be of different paradigms and can have very contrasting, conflicting characteristics and behaviours. The two most important ones are the Functional and Logical Style of Programming.

Functional Programming, [?] gets its name as the fundamental concept is to apply 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 is typed and untyped lambda calculus. Untyped lambda calculus being restriction free that is function application has no restrictions whereas typed lambda calculus puts restriction on what sort(type) of data can a function work with. Another aspect is the type systems of the languages that fall into this category. The type systems 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/or formulas in symbolic logic which are used to derive new formulas from the old ones. This is done till the one which gives the solution is not derived.

The languages in question 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 [?] is chosen because it is the most dominant Logic Programming Language and has spawned a number of distributions from academia to industry with the number being near the best part of 20.

HASKELL is one the most popular [?] Functional Languages around and is the first language to incorporate Monads [?] for safe IO. 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”. PROLOG has been partially implemented [?] in other languages like SCHEME [?], LISP [?, ?, ?], JAVA [?, ?], JAVASCRIPT [?] and the list [?] goes on and on.

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.

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.