1 Prototype 1

This chapter discusses the aspects of opening a language while preserving the original structure of a closed recursive structure in HASKELL. Also discussed are the issues related to customizing certain aspects such as meta-syntactic variables.

1.1 What we do in this Prototype

This prototype throws light on the process of tackling the issues involved in creating a data type to replicate the target language type system while conforming to the host language restrictions and also utilizing the benefits.

We have a PROLOG like language in HASKELL defined via *data*.

The language defined is recursive in nature.

We convert it into a non recursive data type.

1.2 Creating a data type

A type system consists of a set of rules to define a "type" to different constructs in a programming language such as variables, functions and so on. A static type system requires types to be attached to the programming constructs before hand which results in finding errors at compile time and thus increase the reliability of the program. The other end is the dynamic type system which passes through code which would not have worked in former environment, it comes of as less rigid.

The advantages of static typing [**?**]

1. Earlier detection of errors

2. Better documentation in terms of type signatures

3. More opportunities for compiler optimizations

4. Increased run-time efficiency

5. Better developer tools

For dynamic typing

1. Less rigid

2. Ideal for prototyping / unknown / changing requirements or unpredictable behaviour

3. Re-usability

Transitional paragraph An ideal case would would be something that is .... dont know what to write

To start with, replicating the single type "term" in PROLOG one must consider the distinct constructs it can be associated to such as complex structures (for example predicates, clauses etc.), don't cares, cuts, variables and so on.

Consider the language below,

Even though *Term* has a number of constructors the resulting construct has a single type. Hence, a function would still be untyped / singly typed,

The above data type is recursive as seen in the constructor,

One of the issues with the above is that it is not possible to distinguish the structure of the data from the data type itself [?]. Consider the following, a reduced version of the above data type,

Also one cannot create Quantifiers plus logic

To split a data type into two levels, a single recursive data type is replaced by two related data types. Consider the following,

One result of the approach is that the non-recursive type *FlatTerm* is modular and generic as the structure "FlatTerm" is separate from it's type which is "a". Simply speaking we can have something like

and a generic fuinction like,

1.3 Working with the language

Creating instances,

After flattening do fixing,

Opening up the language somehow so as to accommodate your own variables.

1.4 Black box

hello