Bidirectional Type Checking and Inference (Draft)

Isitha Subasinghe

26th November 2021 (Draft)

Abstract

Type theory is infamously considered difficult to get started in. There is simply too much to learn and this information is scattered all over the place. This paper addresses this issue by providing the basics for building a state of the art type inference and checking algorithm based on bidirectional type inference as described by Dunfield and Krishnaswami.

1 Introduction

Type inference is at the core of modern langauges, yet the knowledge to implement these algorithms are not easily accessible. Understanding type inference requires the reader to have a grasp of Type Theory, Set Theory, Lambda Calculus and First Order Logic, this paper aims to reduce the burden of learning type inference and provides a central location for a reader to understand enough of these concepts to get started on programming their own implementation of the previously mentioned type inference algorithm.

2 Related Work

Hindley-Milner type inference, discovered by Hindley and Milner independently have two known algorithms for type inference, Algorithm W and Algorithm J, both of which require a certain cognitive effort by the reader.

3 Setup

Let's define a module called "Eval" used to contain our algorithm. This is a simple but necessary step for any Haskell program.

module *Eval* **where**

In addition to the above code we need another module called "Main" to function as our entry point. We mostly don't have to worry about this Module, it is a simple interface to interact with the "Eval" module.

```
module Main where
```

4 Lambda Calculus

Knowledge of lambda calculus is required to understand some of the type theory needed to understand the paper referenced previously. Here we provide a quick, mediocre introduction to lambda calculus.

4.1 α conversion

α conversion is.

4.2 β reductions

β reductions are

4.3 η reductions

η reductions

4.4 Simply Typed Lambda Calculus

Simply Typed Lambda Calculus can be given by the following grammar presented in BNF form.

A Appendix

A.1 Notation