An Intro to Dependent Types with Idris

Thomas Gebert

August 9, 2016

Problems with regular type systems

Haskell, F#, and OCaml all have great type systems, but still there are still holes that can be improved on

```
head []
-- *** Exception: Prelude.head: empty list

printf "blah %s %s" "hello"
-- *** Exception: printf: argument list ended
-- prematurely
```

What causes these problems?

- Most functional languages have two languages that can't interact: the Type language, and the programming language
  - Types exist only as an enforcement layer
- Since these languages can't interact, all the types and conditions for them must be known by the programmer ahead of time, and can't can't be deduced from the context of the code.

The solution to these problems (and many others) are dependent types

A note about compiler hacks

F#'s printfn works as you would expect due to a special case in the compiler doing static analysis on that particular case.

This is totes OK, but that only works for that particular case.

So what the hell is a dependent type?

A dependently typed language generally means two things

- Types, like functions, are first-class citizens that can be built dynamically (without it being dynamic typing)
- Types (return types, input types, etc) can change depending on values
- Functions can be called inside the type signature