Super advanced functional programming

Or: dependently-typed programming in Agda

Dr. Dominic Mulligan

Programming, Logic, and Semantics Group,
University of Cambridge

Part III, Advanced Functional Programming, March 2017

System F_ω

Very expressive type theory:

- Used as a compiler intermediate language (e.g. GHC)
- Can embed almost all useful (co)datatypes within it
- Can express common programming abstractions with higher-kinds

System F_ω

Very expressive type theory:

- Used as a compiler intermediate language (e.g. GHC)
- Can embed almost all useful (co)datatypes within it
- Can express common programming abstractions with higher-kinds

But:

- The monadic abstraction has associated laws
- Must be checked by hand, on pen-and-paper

Internalising reasoning about programs

How can we internalise this checking of laws?

Internalising reasoning about programs

How can we internalise this checking of laws?

Requires an embedded higher-order logic

Requires types that *depend* on terms:

$$\forall x : \mathbb{N}.\ x + 0 = x$$

is a type, and its inhabitants are proofs

Internalising reasoning about programs

How can we internalise this checking of laws?

Requires an embedded higher-order logic

Requires types that *depend* on terms:

$$\forall x : \mathbb{N}.\ x + 0 = x$$

is a type, and its inhabitants are proofs

Moved from left plane to right plane of λ-cube

What's the advantage?

Agda can be seen as both a programming language and a proof checker

What's the advantage?

Agda can be seen as both a programming language and a proof checker

Agda:

- Allows us to encode very powerful invariants in types that guarantee program correctness
- Acts as a foundation for mathematics, based not on sets, but on functions and types

Rest of this lecture: an interactive introduction to Agda...