Dependent Types in the Idris Programming Language

Part 2: Proofs, Predicates and Totality

Edwin Brady (ecb10@st-andrews.ac.uk)
University of St Andrews, Scotland, UK
@edwinbrady

OPLSS, July 1st 2017
What are the benefits of Idris over Haskell, other than getting help from the compiler?
What are the benefits of Idris over Haskell, other than getting help from the compiler?

Is there a "standard" library, and how can I contribute to libraries?
What are the benefits of Idris over Haskell, other than getting help from the compiler?

Is there a “standard” library, and how can I contribute to libraries?

Have dependent types ever helped you catch a serious bug?
Are there rules about names? (e.g. must types begin with a capital letter?)
- Are there rules about names? (e.g. must types begin with a capital letter?)
- What is the most important feature missing from Idris?
Questions from Yesterday (2)

- Are there rules about names? (e.g. must types begin with a capital letter?)
- What is the most important feature missing from Idris?
- Are there any plans to implement Idris in Idris?
Questions from Yesterday (3)

Is there anything fundamental about a dependently typed language that makes it slower at run time?
Questions from Yesterday (3)

- Is there anything fundamental about a dependently typed language that makes it slower at run time?
- Why do you have a PHP backend?
Is there anything fundamental about a dependently typed language that makes it slower at run time?

Why do you have a PHP backend?

Can I have a book?
Type-driven Development
A total function is a function which, for all well-typed inputs, either

- *Terminates* with a well-typed result
- *Produces* a *finite, non-empty* prefix of a well-typed *infinite* result in finite time
Why do we care?

If we care about *types*, we should care about *totality*

Given $f : T$

- If $f$ is *total*, we know that it will *always* give a result of type $T$
- If $f$ is *partial*, we know that *if* it gives a result, it will be of type $T$
Why do we care?

If we care about *types*, we should care about *totality*

Given \( f : \) Theorem

- If \( f \) is *total*, we know that it will *always* give a result of type Theorem
- If \( f \) is *partial*, we know that *if* it gives a result, it will be of type Theorem
Idris checks:

- Coverage: patterns for all *well-typed inputs*
- Termination: there is a *decreasing* argument
- Productivity: recursive call is *guarded* by a constructor
Demonstration: Proofs, Predicates and Totality