The Implementation of Idris 2

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

SPLV, 17th August 2020

Idris

Idris (`http://idris-lang.org/`) is a functional programming language with *first class types*, supporting *type-driven development*. In these lectures:

- A brief introduction to Idris
  - Type-driven development, first class types, interactive editing.
- Implementing *TinyIdris*, a scaled down version
  - Just the core features: minimal syntax, type checking, evaluation, unification

Course info: `https://github.com/edwinb/SPLV20`

Goals

1. Understand how a dependently typed language works, primarily *type checking* and *elaboration*
   - That is, the `Core` representation

Goals

1. Understand how a dependently typed language works, primarily *type checking* and *elaboration*
   - That is, the `Core` representation
2. Implement some components of a *complete* (if small) dependently typed language
   - . . . which you can use as the basis of your own projects

Goals

1. Understand how a dependently typed language works, primarily *type checking* and *elaboration*
   - That is, the `Core` representation
2. Implement some components of a *complete* (if small) dependently typed language
   - ...which you can use as the basis of your own projects
3. Be able to contribute to Idris 2!
   - Enough knowledge that `https://idris2.readthedocs.io/en/latest/implementation/index.html` can teach you the rest...

Non-goals

There's a lot of Idris 2, so we can't cover all of it! So we will miss:

- Quantities (linearity, erasure)
- High level features (e.g. `case`, `with`)
- Parsing (it's conventional. . .)
- Code generation (also *relatively* straightforward)
- *insert your favourite feature*
- . . .

However:

- Please ask me about these (on the SPLV Slack)
- Once you understand the core, other features are easier to learn

Questions

- Please ask questions!
- Suggested protocol:
  - Ask in the chat
  - I will keep an eye out (and Ohad might prod me)...
  - I won't read out names, so don't worry about appearing on the recording!
- Please also ask questions in the SPLV Slack
  - I'll summarise in the next lecture

Course Plan

- *Lecture 1* (Today) Idris Overview
    - Core features, structure of the system
        - What we need to implement, and how we're going to implement it!
    - Introducing TinyIdris
    - Warm up exercises

Course Plan

- *Lecture 1* (Today) Idris Overview
  - Core features, structure of the system
    - What we need to implement, and how we're going to implement it!
  - Introducing TinyIdris
  - Warm up exercises
- *Lecture 2* Core language, term representation
  - Dealing with variable names
  - Term manipulation: weakening, contraction, substitution...

Course Plan

- *Lecture 1* (Today) Idris Overview
  - Core features, structure of the system
    - What we need to implement, and how we're going to implement it!
  - Introducing TinyIdris
  - Warm up exercises
- *Lecture 2* Core language, term representation
  - Dealing with variable names
  - Term manipulation: weakening, contraction, substitution...
- *Lecture 3* Type checking and evaluation
  - Representing normal forms
  - Type checking dependent types
  - Conversion checking

Course Plan

- *Lecture 1* (Today) Idris Overview
  - Core features, structure of the system
    - What we need to implement, and how we're going to implement it!
  - Introducing TinyIdris
  - Warm up exercises
- *Lecture 2* Core language, term representation
  - Dealing with variable names
  - Term manipulation: weakening, contraction, substitution...
- *Lecture 3* Type checking and evaluation
  - Representing normal forms
  - Type checking dependent types
  - Conversion checking
- *Lecture 4* Unification
  - Introducing implicit syntax

Introduction: Type-driven Development in Idris 2

Implementing Idris 2

- Implemented in Idris 2
  - Initially implemented in Idris 1 then ported
  - Main benefit: certain classes of error *can't happen*!
  - `https://github.com/idris-lang/Idris2`
- Compiles via Chez Scheme (`https://scheme.com/`)
  - or, optionally, via Racket
- Performance: about an order of magnitude faster than Idris 1!

Structure of Idris 2

- *Idris 2* high level syntax
  - All the high level features
  - Annotated with source location
  - *desugars* to . . .

Structure of Idris 2

- *Idris 2* high level syntax
  - All the high level features
  - Annotated with source location
  - *desugars* to . . .
- *TTImp*, an intermediate type theory with *implicits*
  - Supports *local* definitions (`let`)
  - `case` blocks, `with`, `rewrite`, . . .
  - All *interactive editing* support is at this level
  - *elaborates* to . . .

Structure of Idris 2

- *Idris 2* high level syntax
  - All the high level features
  - Annotated with source location
  - *desugars* to ...
- *TTImp*, an intermediate type theory with *implicits*
  - Supports *local* definitions (`let`)
  - `case` blocks, `with`, `rewrite`, ...
  - All *interactive editing* support is at this level
  - *elaborates* to ...
- *QTT*, a core type theory with *quantities*
  - *Only* data declarations and pattern matching definitions
  - Everything *completely explicit*
  - *compiles* to ...

Structure of Idris 2

- *CExp*, an untyped lambda calculus
  - *Erased* variables are gone
  - Various transformations: inlining, `case` manipulation
  - Directly (more or less) maps to...

Structure of Idris 2

- *CExp*, an untyped lambda calculus
  - *Erased* variables are gone
  - Various transformations: inlining, `case` manipulation
  - Directly (more or less) maps to...
- Chez Scheme
  - `https://scheme.come`
  - or, optionally, a back end of your own design
    - See `https://idris2.readthedocs.io/en/latest/backends/custom.html`

TinyIdris

- A *very* cut-down implementation of Idris 2
- Supports:
    - `data` definitions
    - Top level pattern matching function definitions
    - Implicit syntax (to some extent)
    - That's all!
- Minimal, but captures most of the difficulties we need to overcome in a full scale implementation
- Similar in *structure* to the real Idris 2
    - TinyIdris source files map directly to Idris 2 equivalents

Idris

Demonstration: TinyIdris

sicsa*

A tour of TinyIdris

Two most important parts of the module hierarchy:

- `Core`: the core type theory (TT)
  - `Core.Core`: The “monad” carrying all the context
  - `Core.TT`: TT terms (more on this tomorrow)
  - `Core.CaseTree`: Compiled case trees, for evaluation
  - `Core.Context`: Storing definitions
  - `Core.Normalise`: Evaluation
  - `Core.Unify`: Unification
- `TTImp`: the surface languge (TT + implicits)
  - `TTImp.Elab.Term`: Elaboration to `TT`
  - `TTImp.ProcessDecl`: Elaborating top level declarations

Demonstration: `Core.Core`

Today's exercises

1. Browse the source code for `TinyIdris-v1`
   - ... but don't worry too much about the details just yet
2. Look in `Code/Lecture1/WarmupExercise` and complete the definitions