ITI0212 Functional programming

Lecture 1 - Introduction

Pawel Sobocinski

Introduction and admin matters

Rough schedule

- Pawel Sobocinski - weeks 1 through 5
- week 6 (2-6 March) no lectures, no labs
- Chad Nester - weeks 7 through 10
- Ed Morehouse - weeks 11 through 14
- Niccolò Veltri - weeks 15 & 16

What this course is about?

- functional programming
  - functions are first-class objects
  - pattern-matching, recursion
  - pure vs non-pure code
- the importance of types
- cross-fertilisation

What is Idris?

Overview

Idris is a general purpose pure functional programming language with dependent types. Dependent types allow *types* to be predicated on *values*, meaning that some aspects of a program's behaviour can be specified precisely in the type. It is compiled, with eager evaluation. Its features are influenced by Haskell and ML, and include:

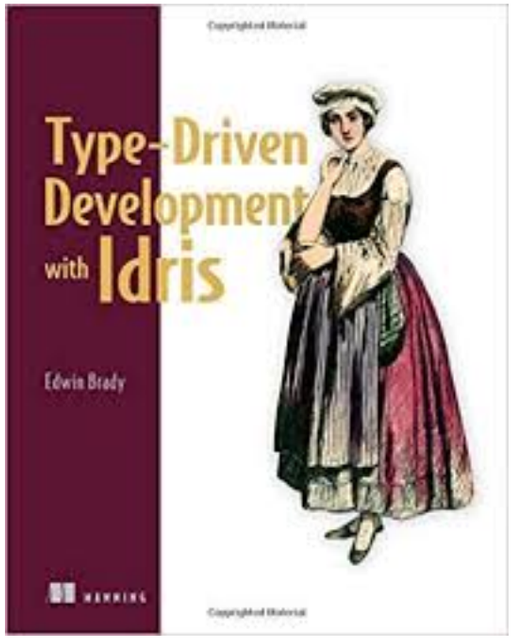
- Full dependent types with dependent pattern matching
- Simple foreign function interface (to C)
- Compiler-supported interactive editing: the compiler helps you write code using the types
- `where` clauses, `with` rule, simple `case` expressions, pattern matching `let` and lambda bindings
- Dependent records with projection and update
- Interfaces (similar to type classes in Haskell)
- Type-driven overloading resolution
- `do` notation and idiom brackets
- Indentation significant syntax
- Extensible syntax
- Cumulative universes
- Totality checking
- Hugs style interactive environment

Why Idris?

- Haskell-like syntax but strict evaluation 😍
- exciting features from recent research on programming languages that may be influential in future programming language designs
  - type-driven development
  - types as first-class objects
  - dependent types
  - total vs partial functions

Resources

- Primary textbook
  - Edwin Brady - Type-driven development with Idris
  - available online via library



- Other textbooks
  - Miran Lipovača - Learn you a Haskell for Great Good! - http://learnyouahaskell.com/chapters
  - Simon Thompson - The craft of functional programming

Assessment structure

- 4 small assignments worth 5% each and marked in labs = 20%
- 1 big assignment worth 30%.
  - spec released 24 February
  - due 27 April
- final written 2 hour exam worth 50%

Timetable

- Lectures: Tuesdays 10am, ICT-A2
- Labs: Wednesdays 10am, ICT-404
- You are expected to come to all lectures and labs
  - Tomorrow’s lab is **optional**

The REPL and basic types

Installing Idris

idris-lang.org

Home | News | Example | Documentation | Contributions | Download

Recent Posts

Idris 1.3.2 Released
Idris 1.3.1 released
Idris 1.3.0 released
Idris 1.2.0 released
Idris 1.1.1 released

Meta

Log in
Entries RSS
Comments RSS
WordPress.org

Download

Source

You can download the source from the following places:

- Hackage has the most recent released version. At the prompt, type:
  - `cabal update`
  - `cabal install idris`
- The development version is available from github:

```
git clone git://github.com/idris-lang/Idris-dev.git
```

There are various options available (pass `-f [option]` to `cabal install`) to customise the installation, which you can use depending on which libraries you have available. They are:

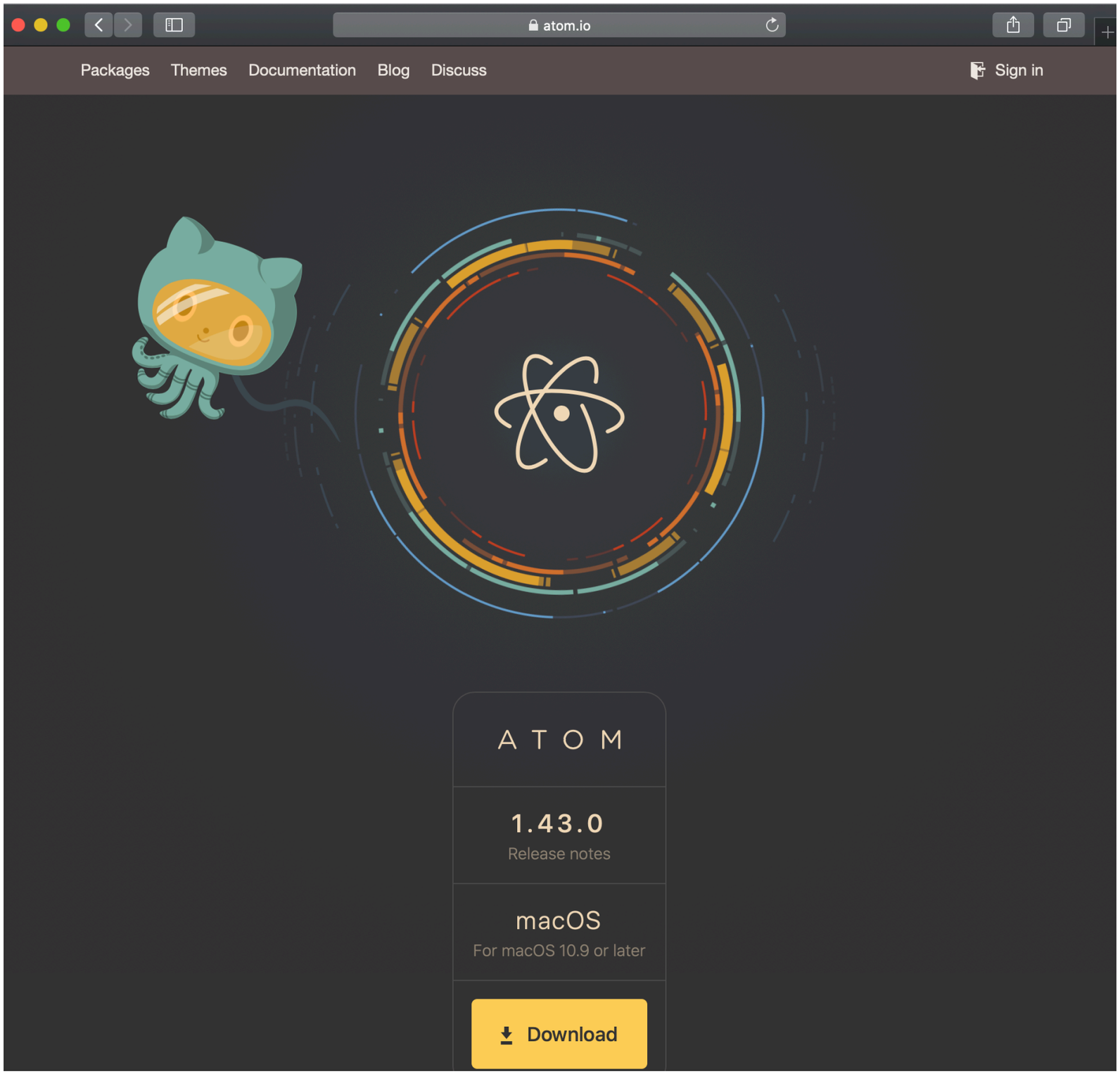
- `llvm` to build the LLVM code generation
- `ffi` to use `libffi`, which is required for the type providers feature.
- `gmp` to use `libgmp` for big integer support (instead of the less efficient `mini-gmp`).
- `curses` to use `libncurses` to get the screen width for pretty-printing.

More detailed **installation instructions** that are maintained by the Idris community are available on the wiki for the following platforms:

- Mac OS
- Windows
- Ubuntu
- Debian

Binary

Recommended editor



REPL

- Read - Eval - Print Loop
- REPL interactive environment
  - After installation, type `idris` at the command prompt
  - `:q` quits

```
⌥⌘3                                                    zsh
% idris
     ____    __     _
    /  _/___/ /____(_)____
    / // __  / ___/ / ___/     Version 1.3.2
  _/ // /_/ / /  / (__  )      http://www.idris-lang.org/
 /___/\__,_/_/  /_/____/       Type :? for help

Idris is free software with ABSOLUTELY NO WARRANTY.
For details type :warranty.
Idris> :q
Bye bye
%
```

Hello world

Hello.idr — ~/Dropbox/mystuff/teaching/functionalProgramming/code

Hello.idr

```
1  module Main
2
3  main: IO ()
4  main = putStrLn "Hello, world!"
5
```

~/Dropbox/mystuff/teaching/functionalProgramming/code/Hello.idr 5:1 LF UTF-8 Idris GitHub Git (0)

Running `Hello.idr`

```
⌥⌘3                                                    idris
% idris Hello.idr
     ____    __     _
    /  _/___/ /____(_)____
    / // __  / ___/ / ___/     Version 1.3.2
  _/ // /_/ / /  / (__  )      http://www.idris-lang.org/
 /___/\__,_/_/  /_/____/       Type :? for help

Idris is free software with ABSOLUTELY NO WARRANTY.
For details type :warranty.
*Hello> :exec
Hello, world!
*Hello>
```

Compiling `Hello.idr`

```
% idris Hello.idr -o Hello
% ./Hello
Hello, world!
%
```

Basic numeric types and arithmetic

- `Int` - fixed width signed integer type, at least 31 bits
- `Integer` - unbounded signed integer type
- `Nat` - unbounded unsigned integer type
- `Double` - double-precision floating point

`Integer` is default

The **context** means that `Double` is the only possibility

```
% idris
     ____    __     _
    /  _/___/ /____(_)____
    / // __  / ___/ / ___/     Version 1.3.2
  _/ // /_/ / /  / (__  )      http://www.idris-lang.org/
 /___/\__,_/_/  /_/____/       Type :? for help

Idris is free software with ABSOLUTELY NO WARRANTY.
For details type :warranty.
Idris> 3+4*12
51 : Integer
Idris> it - 6
45 : Integer
Idris> 2.0 * 5
10.0 : Double
Idris>
```

Type conversion with `cast` and `the`

```
Idris is free software with ABSOLUTELY NO WARRANTY.
For details type :warranty.
Idris> :let integerval = 2*2
Idris> :let doubleval = 0.5
Idris> integerval
4 : Integer
Idris> doubleval
0.5 : Double
Idris> integerval + doubleval
(input):1:12:When checking an application of function Prelude.Interfaces.+:
        Type mismatch between
                Double (Type of doubleval)
        and
                Integer (Expected type)
Idris> cast integerval + doubleval
4.5 : Double
Idris> the Integer (cast it)
4 : Integer
Idris> the Int 6
6 : Int
Idris>
```

- Prelude is a list of simple, useful functions written in Idris and loaded by default
- `the` is actually a simple but interesting function defined in the Prelude

```
module Prelude.Basics

import Builtins

%access public export

Not : Type -> Type
Not a = a -> Void

||| Identity function.
id : a -> a
id x = x

||| Manually assign a type to an expression.
||| @ a the type to assign
||| @ value the element to get the type
the : (a : Type) -> (value : a) -> a
the _ = id
```

:doc

```
% idris
     ____    __     _
    /  _/___/ /____(_)____
    / // __  / ___/ / ___/     Version 1.3.2
  _/ // /_/ / /  / (__  )      http://www.idris-lang.org/
 /___/\__,_/_/  /_/____/       Type :? for help

Idris is free software with ABSOLUTELY NO WARRANTY.
For details type :warranty.
Idris> :doc the
Prelude.Basics.the : (a : Type) -> (value : a) -> a
    Manually assign a type to an expression.
    Arguments:
        a : Type  -- the type to assign

        value : a  -- the element to get the type

    The function is: Total & public export
Idris>
```

- Prelude is a list of simple, useful functions written in Idris and loaded by default
- `the` is actually a simple but interesting function defined in the Prelude

```
module Prelude.Basics

import Builtins

%access public export

Not : Type -> Type
Not a = a -> Void

||| Identity function.
id : a -> a
id x = x

||| Manually assign a type to an expression.
||| @ a the type to assign
||| @ value the element to get the type
the : (a : Type) -> (value : a) -> a
the _ = id
```

Characters and strings

```
⌥⌘3                                    idris
 _/ // /_/ / /  / (__  )      http://www.idris-lang.org/
/___/\__,_/_/  /_/____/       Type :? for help

Idris is free software with ABSOLUTELY NO WARRANTY.
For details type :warranty.
Idris> :t 'a'
'a' : Char
Idris> :t "Hello world"
"Hello world" : String
Idris> :let greeting = "Hello!"
Idris> :t greeting
greeting : String
Idris> length greeting
6 : Nat
Idris> reverse greeting
"!olleH" : String
Idris> greeting ++ " " ++ greeting
"Hello! Hello!" : String
Idris> length (greeting ++ " " ++ greeting)
13 : Nat
Idris>
```

- `Char` - character type
- `String` - string type
  - `++` - string concatenation

Booleans

```
⌥⌘3                                    idris
    / // __  / ___/ / ___/     Version 1.3.2
  _/ // /_/ / /  / (__  )      http://www.idris-lang.org/
 /___/\__,_/_/  /_/____/       Type :? for help

Idris is free software with ABSOLUTELY NO WARRANTY.
For details type :warranty.
Idris> :doc Bool
Data type Prelude.Bool.Bool : Type
    Boolean Data Type

    The function is: public export
Constructors:
    False : Bool


        The function is: public export
    True : Bool


        The function is: public export
Idris>
```

- && - logical and
- || - logical or

```
Idris> 3>2
True : Bool
Idris> 3==2
False : Bool
Idris> 3>2 && 3==2
False : Bool
Idris> 3>2 || 3==2
True : Bool
Idris>
```

Functions - the building blocks of FP

Function types and definitions

type declaration

type of function

```
increment: Int -> Int
increment x = x + 1
```

function definition

```
% idris Increment.idr
     ____    __     _
    /  _/___/ /____(_)____
    / // __  / ___/ / ___/     Version 1.3.2
  _/ // /_/ / /  / (__  )      http://www.idris-lang.org/
 /___/\__,_/_/  /_/____/       Type :? for help

Idris is free software with ABSOLUTELY NO WARRANTY.
For details type :warranty.
Type checking ./Increment.idr
*Increment> increment 3
4 : Int
*Increment>
```

Partially applying functions

```
add : Int -> Int -> Int
add x y = x + y
```

```
% idris Add.idr
     ____    __     _
    /  _/___/ /____(_)____
    / // __  / ___/ / ___/     Version 1.3.2
  _/ // /_/ / /  / (__  )      http://www.idris-lang.org/
 /___/\__,_/_/  /_/____/       Type :? for help

Idris is free software with ABSOLUTELY NO WARRANTY.
For details type :warranty.
Type checking ./Add.idr
*Add> add 2 3
5 : Int
*Add> add 2
add 2 : Int -> Int
*Add> :let add2 = add 2
*Add> :t add2
add2 : Int -> Int
*Add> add2 75
77 : Int
```