Introduction to Programming with Dependent Types in Idris

Jeremy @ HKU

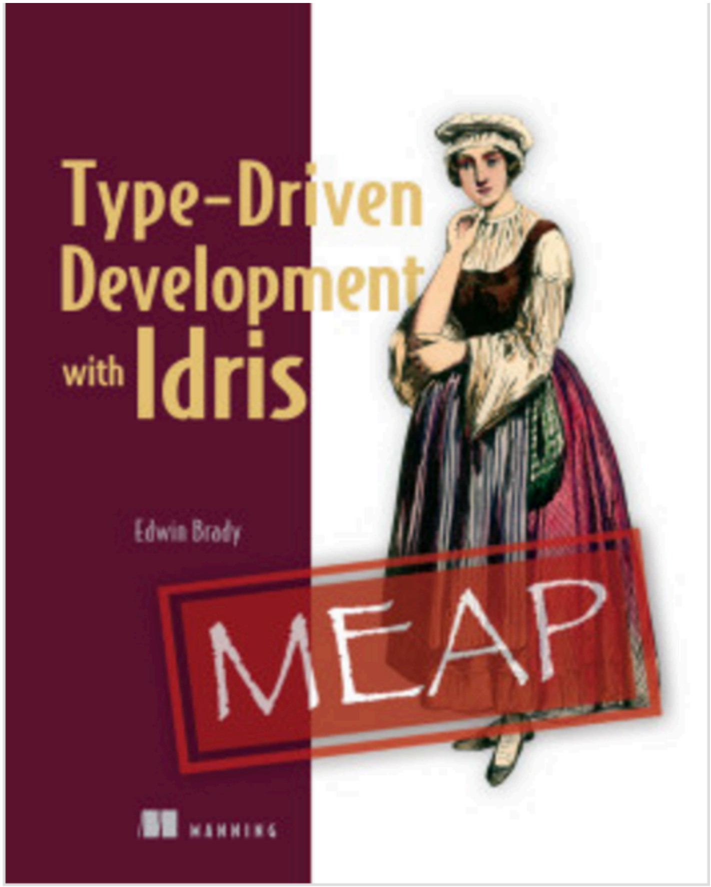
Agenda

- Dependent types in a nutshell
- Getting started with Idris
- Interactive development with Types
- Programming with First Class Type

- This book (still work-in-progress) is written by the author of Idris

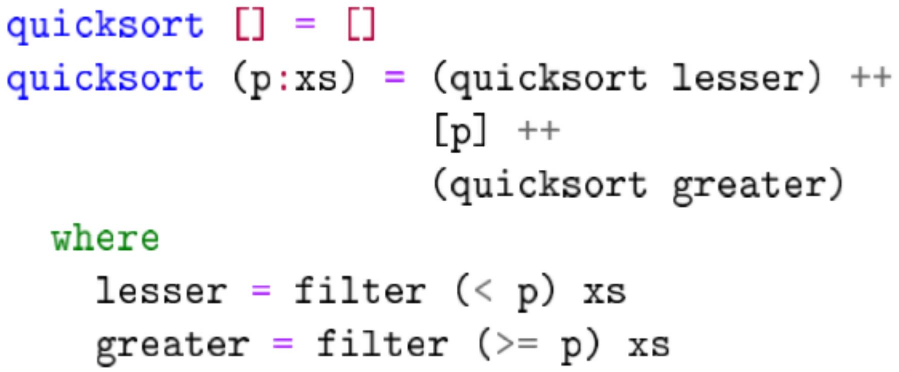
- Beginner-friendly with lots of examples and exercises (no particular background assumed)

- The talk will follow what has been written up to now

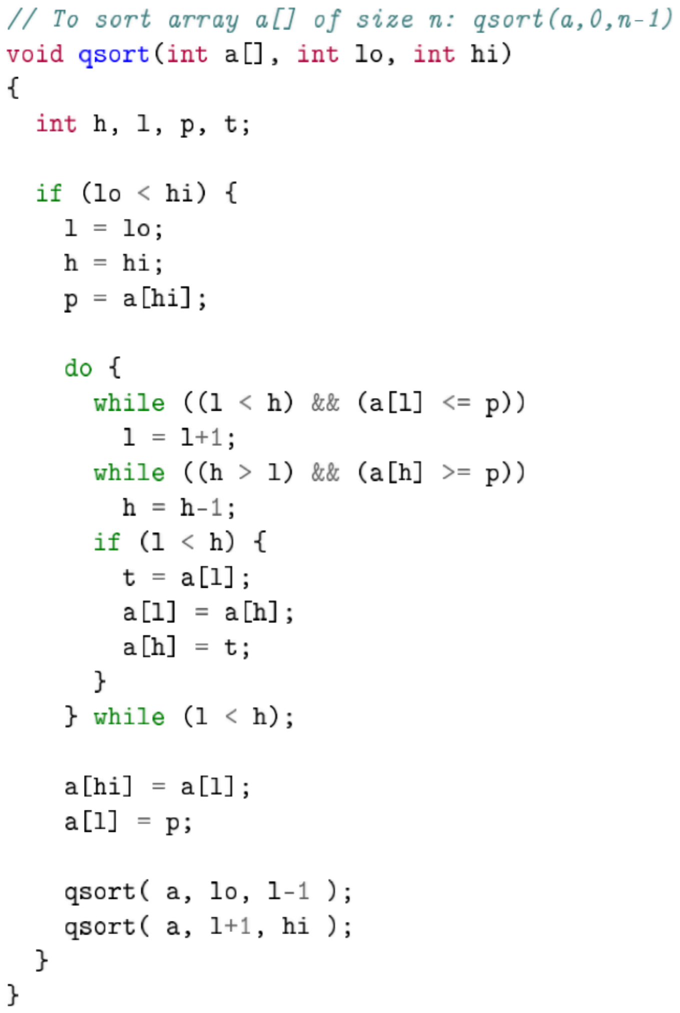


Functional Programming

- As opposed to imperative programming in C, C++, Java, etc.



```haskell
quicksort [] = []
quicksort (p:xs) = (quicksort lesser) ++
                   [p] ++
                   (quicksort greater)
  where
    lesser = filter (< p) xs
    greater = filter (>= p) xs
```



```c
// To sort array a[] of size n: qsort(a,0,n-1)
void qsort(int a[], int lo, int hi)
{
  int h, l, p, t;

  if (lo < hi) {
    l = lo;
    h = hi;
    p = a[hi];

    do {
      while ((l < h) && (a[l] <= p))
        l = l+1;
      while ((h > l) && (a[h] >= p))
        h = h-1;
      if (l < h) {
        t = a[l];
        a[l] = a[h];
        a[h] = t;
      }
    } while (l < h);

    a[hi] = a[l];
    a[l] = p;

    qsort( a, lo, l-1 );
    qsort( a, l+1, hi );
  }
}
```

Types Matter (I)

- Types are a formal language that proves the absence of certain behaviors

- It has deep connections with varieties of logic, via the *Curry-Howard correspondence*

Types Matter (II)

- The trend over the last 20 years goes toward more *expressive types* in modern programming languages

- They are capable of guaranteeing a large variety of program correctness properties:
  - Preventing resource allocation errors
  - Enforcing security in communication protocols
  - Preventing buffer overruns
  - Preventing data races and deadlocks in concurrent systems
  - …

Dependent Types (I)

- Dependent types can depend on the values of terms
- Example: *Vect 3 Int*
- *appendV :: Vect n A -> Vect m A -> Vect (m + n) A*

Dependent Types (II)

- More broadly, dependent types allow programmers to put arbitrary specifications in types
- Example: avoid *out-of-bounds* error at compile time

Natural numbers less than n

```
lookup :: Vec Int n -> Fin n -> Int

lookup :: List Int -> Int -> Int
```

Getting Started with Idris

- Download http://www.idris-lang.org/download/

- Editors: Vim, Emacs, Atom, …

- Fire up REPL ☺
    - :t
    - :doc

Example 1: Hello World

Types as First-class Constructs

- In most languages, there is syntactic separation between types and values (e.g., *x = int* is not allowed in Java)

- In Idris there are no such restrictions
  - Types are first-class
  - Any construct can appear as part of a type

Example 2: Calculating a Type

Interactive Programming with Types

- Dependent Types also give the compiler more information

- We follow the process of “Type, Define, Refine”

- Our editor provides interactive editing mode

Type, Define, Refine

An approach of type-driven development:

1. Write a *Type* to represent the system we are modelling
2. *Define* a function over that type
3. *Refine* the type and definition as necessary to capture any missing properties, or to help ensure totality

Interactive editing mode helps with the above process!

Example 3 : Type, Define, Refine

Type-directed Search

- Given enough information in the type, Idris can search for a valid expression satisfying that type

- This is extremely handy and shows how smart compiler can be

Example 4: Zipping two vectors

Parametricity

- Also known as “Free Theorem”

- You can know a lot by just looking at the type

- Example: *a -> a*, *a -> b -> a*

- A paper (“Theorem for free!”) written by Philip Wadler

Implicit Arguments

- Look again at the following type

  appendV : Vect n A -> Vect m A -> Vect (m + n) A

- *n* and *A* are called type level variables
- They are not declared anywhere, and are thus referred as *implicit* arguments
- How can they be useful? How to use them in definition?

Example 5: Implicit arguments

Example 6: Sorting a vector

Example 7: Transposing a matrix

Programming with First Class Type

- In Idris, types can be manipulated just like any other language constructs.

- For example, we can give informative names to complex types (representing polygon)

- Dependent pattern matching (recall *valToString*)

- Differences with ordinary functions (runtime-representation, totality)

Example 8: A Type safe lookup function

- It should be possible to know at compile time whether the index is out of bounds when the program is run
- The *index* function is a type safe lookup function (from *Data.Vect* module):

index : Fin n -> Vect n a -> a

- *Fin n* denotes numbers that are strictly less than *n*
- But if the index is read from user input, the number isn’t always going to be within the bound of the vector

Example 9: A Type safe *printf* Function

Want to Know More?

- Idris can also do theorem proving
- Idris Tutorial: http://docs.idris-lang.org/en/latest/tutorial/
- Excellent Talk by the author: https://youtu.be/X36ye-1x_HQ
- Questions?