Informatics 1

Functional Programming Lecture 2

Thursday 2 October 2008

The Rule of Leibniz

Philip Wadler

University of Edinburgh

Tutorial, Labs, Lab week

ITO will assign you to tutorials, starting next week

Tuesday/Wednesday	Computation and Logic
Thursday/Friday	Functional Programming

Drop-in laboratories

Mondays	35pm	West
Tuesdays	25pm	West
Wednesdays	25pm	West
Thursdays	25pm	South
Fridays	35pm	West

Computer Lab West and South, Appleton Tower, level 5 (not level 4)

Lab Week Exercise due 5pm Friday 3 October

Required text and reading

Haskell: The Craft of Functional Programming, Second Edition, Simon Thompson, Addison-Wesley, 1999.

Reading assignment:

Thompson, Chapters 1–3 (pp. 1–52): by Mon 29 Sep 2008.
Thompson, Chapters 4–5 (pp. 53–95): by Mon 7 Oct 2008.
Thompson, Chapters 6–7 (pp. 96–134): by Mon 15 Oct 2008.

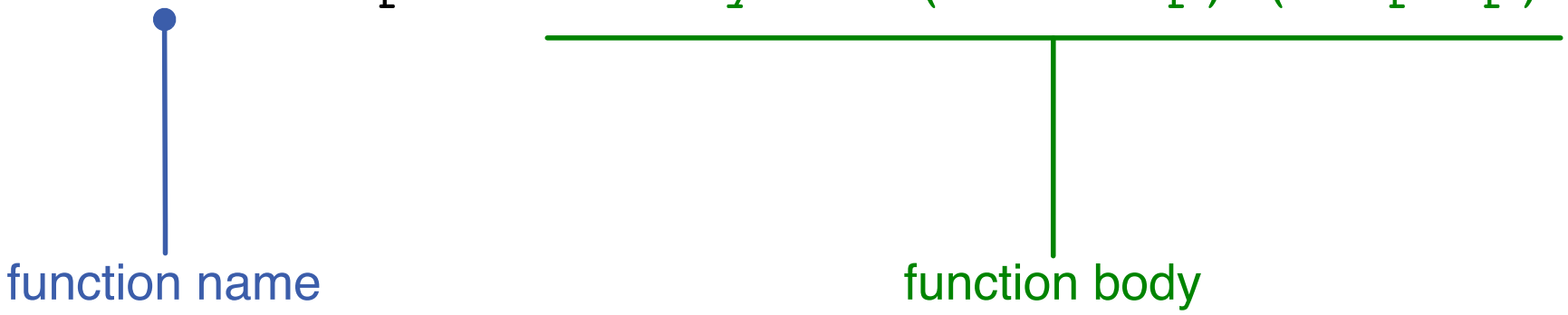
Review: Terminology

Type signature

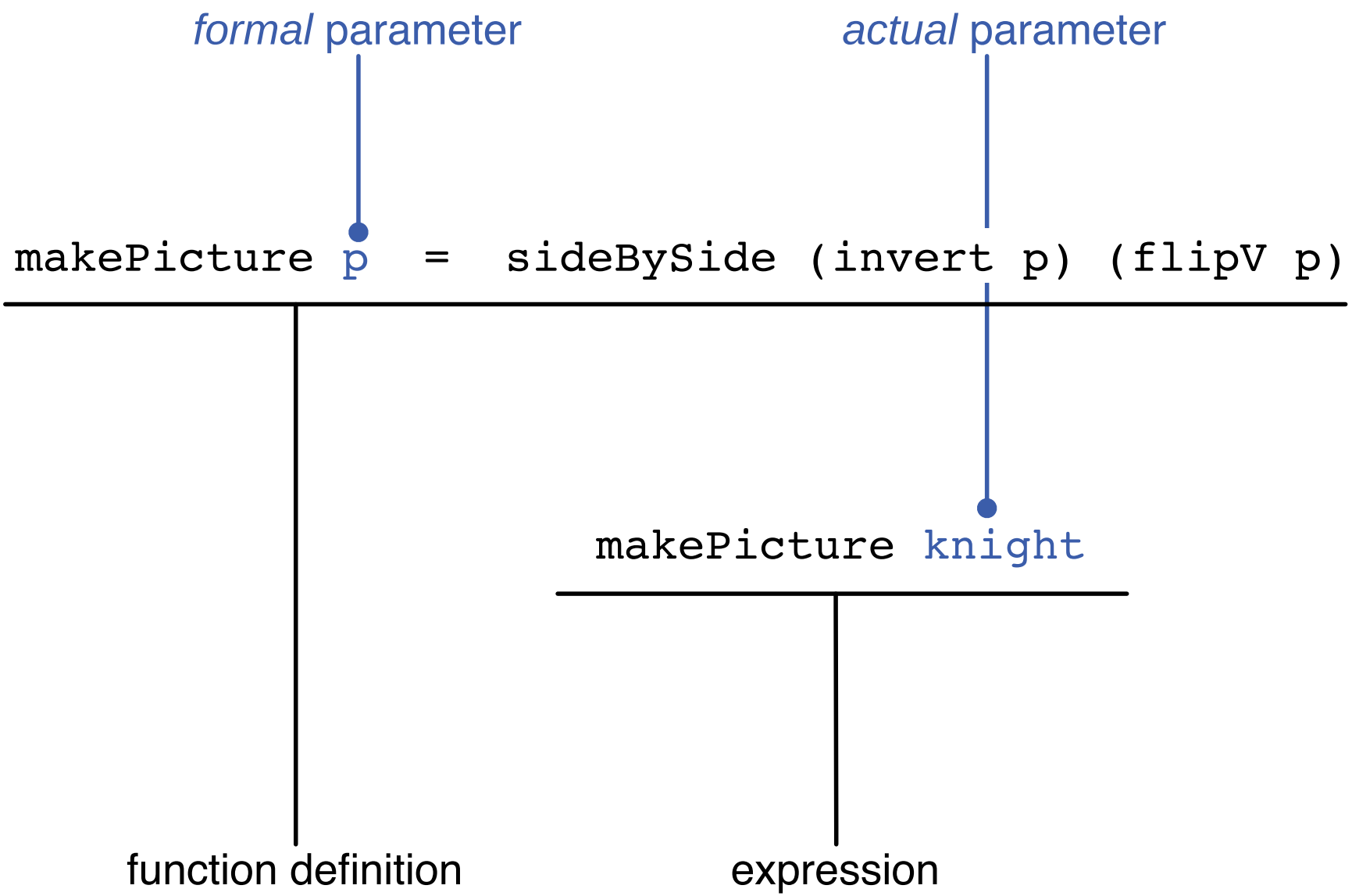
```
makePicture :: Picture -> Picture
```

Function declaration

```
makePicture p  =  sideBySide (invert p) (flipV p)
```



Review: Terminology



Part I

The Rule of Leibniz

Operations on numbers

```
[culross]wadler: ghci
   ___                ___ _
  / _ \ /\  /\/ __(_)
 / /_\// /_/ / /   | |      GHC Interactive, version 6.7
/ /_\\/ __  / /___| |      http://www.haskell.org/ghc/
\____/\/ /_/\____/|_|      Type :? for help.

Loading package base ... linking ... done.
Prelude> 2+2
4
Prelude> 3*3
9
Prelude>
```

Functions over numbers

lect02.hs

```
square :: Integer -> Integer
square x  =  x * x

pyth :: Integer -> Integer -> Integer
pyth a b =  square a + square b
```

Testing our functions

```
[culross]wadler: ghci lect02.hs
   ___         ___ _
  / _ \ /\  /\/ __(_)
 / /_\// /_/ / /   | |      GHC Interactive, version 6.7
/ /_\\/ __  / /___| |      http://www.haskell.org/ghc/
\____/\/ /_/\____/|_|      Type :? for help.

Loading package base ... linking ... done.
[1 of 1] Compiling Main             ( lect02.hs, interpreted )
Ok, modules loaded: Main.
*Main> square 3
9
*Main> pyth 3 4
25
*Main>
```

A few more tests

```
*Main> square 0
0
*Main> square 1
1
*Main> square 2
4
*Main> square 3
9
*Main> square 4
16
*Main> square (-3)
9
*Main> square 10000000000
100000000000000000000
```

Functions over numbers

```
square :: Integer -> Integer
square x  =  x * x

pyth :: Integer -> Integer -> Integer
pyth x y  =  square x + square y
```

Declaration and evaluation

Declaration (file lect02a.hs)

```
square :: Integer -> Integer
square x  =  x * x

pyth :: Integer -> Integer -> Integer
pyth x y  =  square x + square y
```

Evaluation

```
% ghci lect02a.hs
   ___         ___ _
  / _ \ /\  /\/ __(_)
 / /_\// /_/ / /   | |      GHC Interactive, version 6.7
/ /_\\/ __  / /___| |      http://www.haskell.org/ghc/
\____/\/ /_/\____/|_|      Type :? for help.

Loading package base-1.0 ... linking ... done.
Compiling Main             ( lect2.hs, interpreted )
Ok, modules loaded: Main.
*Main> pyth 3 4
25
*Main>
```

The Rule of Leibniz

```
square :: Integer -> Integer
square x  =  x * x

pyth :: Integer -> Integer -> Integer
pyth x y  =  square x + square y
```

"Equals may be substituted for equals."

— Gottfried Wilhelm Leibniz (1646–1716)

```
  pyth 3 4
=
  square 3 + square 4
=
  3*3 + 4*4
=
  9 + 16
=
  25
```

Numerical operations are functions

```
(+) :: Integer -> Integer -> Integer
(*) :: Integer -> Integer -> Integer
```

```
Main*> 3+4
7
Main*> 3*4
12
```

`3 + 4` *stands for* `(+) 3 4`

`3 * 4` *stands for* `(*) 3 4`

```
Main*> (+) 3 4
7
Main*> (*) 3 4
12
```

Precedence and parentheses

Function application takes *precedence* over infix operators

(Function applications *binds more tightly than* infix operators)

```
square 3 + square 4
```

stands for

```
(square 3) + (square 4)
```

Multiplication takes *precedence* over addition

(Multiplication *binds more tightly than* addition)

```
3*3 + 4*4
```

stands for

```
(3*3) + (4*4)
```

Associativity

Addition is associative

```
    3 + (4 + 5)
=
    3 + 9
=
    12
=
    7 + 5
=
    (3 + 4) + 5
```

Addition *associates to the left*

```
    3 + 4 + 5
```

stands for

```
    (3 + 4) + 5
```

Strings and concatenation

```
  (++) :: String -> String -> String

Main*> "hello" ++ "world"
"helloworld"

  bracket :: String -> String
  bracket x  =  "<" ++ x ++ ">"

Main*> bracket "Haskell"
"<Haskell>"
```

Concatenation is associative

```
  "<" ++ ("Haskell" ++ ">")
=
  "<" ++ "Haskell>"
=
  "<Haskell>"
=
  "<Haskell" ++ ">"
=
  ("<" ++ "Haskell") ++ ">"
```

Infix operators are just functions

```
(++) :: String -> String -> String

"hello" ++ "world"
```

stands for

```
(++) "hello" "world"
```

Concatenation associates to the right

```
"<" ++ "Haskell" ++ ">"
```

stands for

```
"<" ++ ("Haskell" ++ ">")
```

Why we use infix operators

```
"<" ++ "Haskell" ++ ">"
```

stands for

```
(++) "<" ((++) "Haskell" ">")
```

Gottfried Wilhelm Leibniz (1646–1716)

Anticipated symbolic logic, and discovered calculus (independently of Newton).

> "The only way to rectify our reasonings is to make them as tangible as those of the Mathematicians, so that we can find our error at a glance, and when there are disputes among persons, we can simply say: Let us calculate, without further ado, to see who is right."

> "In symbols one observes an advantage in discovery which is greatest when they express the exact nature of a thing briefly and, as it were, picture it; then indeed the labor of thought is wonderfully diminished."

Part II

QuickCheck

A program (file `lect02a.hs`)

```
square :: Integer -> Integer
square x  =  x * x

pyth :: Integer -> Integer -> Integer
pyth a b =  square a + square b
```

Running a program

```
[culross]wadler: ghci lect02a.hs
GHCi, version 6.8.3: http://www.haskell.org/ghc/ :? for help
Loading package base ... linking ... done.
[1 of 1] Compiling Main             ( lect02a.hs, interpreted )
Ok, modules loaded: Main.
*Main> square 3
9
*Main> pyth 3 4
25
*Main>
```

Another program (file `lect02b.hs`)

```
import Test.QuickCheck

square :: Integer -> Integer
square x  =  x * x

pyth :: Integer -> Integer -> Integer
pyth a b =  square a + square b

prop_square :: Integer -> Bool
prop_square x  =
  square x >= 0

prop_squares :: Integer -> Integer -> Bool
prop_squares x y  =
  square (x+y) == square x + 2*x*y + square y

prop_pyth :: Integer -> Integer -> Bool
prop_pyth x y  =
  square (x+y) == pyth x y + 2*x*y
```

Running another program

```
[culross]wadler: ghci lect02b.hs
GHCi, version 6.8.3: http://www.haskell.org/ghc/ :? for help
Loading package base ... linking ... done.
[1 of 1] Compiling Main             ( lect02b.hs, interpreted )
*Main> quickCheck prop_square
Loading package old-locale-1.0.0.0 ... linking ... done.
Loading package old-time-1.0.0.0 ... linking ... done.
Loading package random-1.0.0.0 ... linking ... done.
Loading package mtl-1.1.0.1 ... linking ... done.
Loading package QuickCheck-2.1 ... linking ... done.
+++ OK, passed 100 tests.
*Main> quickCheck prop_squares
+++ OK, passed 100 tests.
*Main> quickCheck prop_pyth
+++ OK, passed 100 tests.
```

Yet another program (file `lect02c.hs`)

```
import Test.QuickCheck

square :: Int -> Int
square x  =  x * x

prop_square :: Int -> Bool
prop_square x  =
  square x >= 0
```

Running yet another program

```
[culross]wadler: ghci lect02c.hs
GHCi, version 6.8.3: http://www.haskell.org/ghc/ :? for help
Loading package base ... linking ... done.
[1 of 1] Compiling Main             ( lect02c.hs, interpreted )
*Main> quickCheck prop_square
Loading package old-locale-1.0.0.0 ... linking ... done.
Loading package old-time-1.0.0.0 ... linking ... done.
Loading package random-1.0.0.0 ... linking ... done.
Loading package mtl-1.1.0.1 ... linking ... done.
Loading package QuickCheck-2.1 ... linking ... done.
*** Failed! Falsifiable (after 41 tests and 6 shrinks):
46341
*Main> 46341*46341
2147488281
*Main> 2^31
2147483648
```

Fixing the program (file `lect02d.hs`)

```
import Test.QuickCheck

square :: Int -> Int
square x  =  x * x

prop_square :: Int -> Property
prop_square x  =
  abs x < 2^15 ==> square x >= 0
```

Running the fixed program

```
[culross]wadler: ghci lect02d.hs
GHCi, version 6.8.3: http://www.haskell.org/ghc/ :? for help
Loading package base ... linking ... done.
[1 of 1] Compiling Main             ( lect02d.hs, interpreted )
*Main> quickCheck prop_square
Loading package old-locale-1.0.0.0 ... linking ... done.
Loading package old-time-1.0.0.0 ... linking ... done.
Loading package random-1.0.0.0 ... linking ... done.
Loading package mtl-1.1.0.1 ... linking ... done.
Loading package QuickCheck-2.1 ... linking ... done.
*** Gave up! Passed only 49 tests.
*Main>
```

More fun (file `lect02e.hs`)

```
import Test.QuickCheck

prop_assoc_Integer :: Integer -> Integer -> Integer -> Bool
prop_assoc_Integer x y z  =  (x+y)+z == x+(y+z)

prop_assoc_String :: String -> String -> String -> Bool
prop_assoc_String x y z  =  (x++y)++z == x++(y++z)
```