EECS 368
Programming Language Paradigms

Dr. Andy Gill

Department of Electrical Engineering & Computer Science
University of Kansas

November 5, 2015

KU

The Glasgow Haskell Compiler

- A powerful combination of an optimizing compiler and a Hugs-style interactive environment
- Also on the web

`www.haskell.org/ghc`

KU

GHCi can be started using % ghci

```
andy$ ghci
GHCi, version 6.10.1: http://www.haskell.org/ghc/  :? for help
Loading package ghc-prim ... linking ... done.
Loading package integer ... linking ... done.
Loading package base ... linking ... done.
Prelude>
```

GHCi can evaluate expressions

```
andy$ ghci
> 2 + 3 * 4
14
> (2+3)*4
20
> sqrt (3^2 + 4^2)
5.0
```

The Standard Prelude

The library file Prelude.hs provides a large number of standard functions. In addition to the familiar numeric functions such as + and *, the library also provides many useful functions on lists.

- Select the first element of a list:

```
> head [1,2,3,4,5]
1
```

- Remove the first element from a list:

```
> tail [1,2,3,4,5]
[2,3,4,5]
```

- Select the nth element of a list:

```
> [1,2,3,4,5] !! 2
3
```

- Select the first n elements of a list:

```
> take 3 [1,2,3,4,5]
[1,2,3]
```

- Remove the first n elements from a list:

```
> drop 3 [1,2,3,4,5]
[4,5]
```

- Calculate the length of a list:

```
> length [1,2,3,4,5]
5
```

- Calculate the sum of a list of numbers:

```
> sum [1,2,3,4,5]
15
```

- Calculate the product of a list of numbers:

```
> product [1,2,3,4,5]
120
```

- Append two lists:

```
> [1,2,3] ++ [4,5]
[1,2,3,4,5]
```

- Reverse a list:

```
> reverse [1,2,3,4,5]
[5,4,3,2,1]
```

Function Application

In mathematics, function application is denoted using parentheses, and multiplication is often denoted using juxtaposition or space.

```
f(a,b) + c d
```

Apply the function f to a and b, and add the result to the product of c and d.

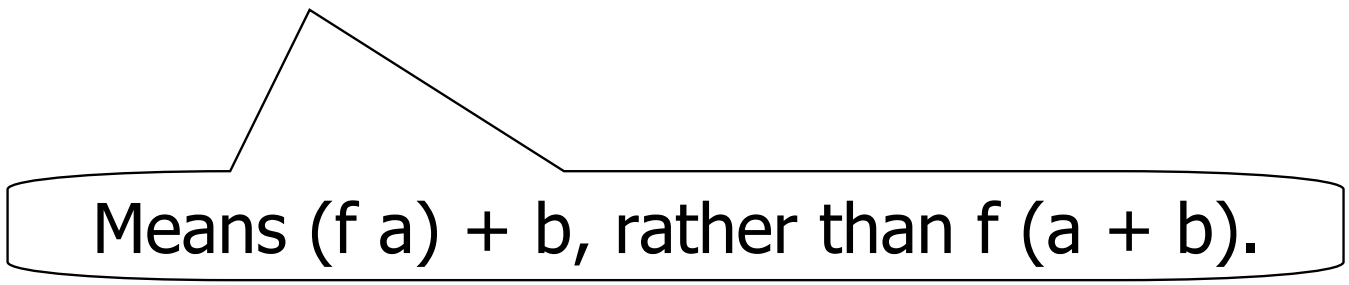
In Haskell, function application is denoted using space, and multiplication is denoted using *.

```
f a b + c*d
```

As previously, but in Haskell syntax.

Moreover, function application is assumed to have higher priority than all other operators.

```
f a + b
```



Examples

Mathematics	Haskell
`f(x)`	`f x`
`f(x,y)`	`f x y`
`f(g(x))`	`f (g x)`
`f(x,g(y))`	`f x (g y)`
`f(x)g(y)`	`f x * g y`

Haskell Scripts

- As well as the functions in the standard prelude, you can also define your own functions;
- New functions are defined within a script, a text file comprising a sequence of definitions;
- By convention, Haskell scripts usually have a `.hs` suffix on their filename.

KU

My First Script

When developing a Haskell script, it is useful to keep two windows open, one running an editor for the script, and the other running Hugs.
Start an editor, type in the following two function definitions, and save the script as Test.hs:

```
double x    = x + x
quadruple x = double (double x)
```

KU

Leaving the editor open, in another window start up Hugs with the new script

```
andy$ ghci Test.hs
```

Now both Prelude.hs and Test.hs are loaded, and functions from both scripts can be used:

```
andy$ ghci
> quadruple 10
40
> take (double 2) [1,2,3,4,5,6]
[1,2,3,4]
```

Leaving GHC open, type the following two definitions, and resave:

```
factorial n = product [1..n]
average ns  = sum ns `div` length ns
```

Note:

- div is enclosed in back quotes, not forward;
- x `f` y is just syntactic sugar for f x y.

GHC does not automatically detect that the script has been changed, so a reload command must be executed before the new definitions can be used:

```
> :r
[1 of 1] Compiling Main ( Test.hs, interpreted )
Ok, modules loaded: Main.
> factorial 10
3628800
> average [1,2,3,4,5]
3
```

Naming Requirements

- Function and argument names must begin with a lower-case letter. For example:

```
myFun    fun1    arg_2    x'
```

- By convention, list arguments usually have an s suffix on their name. For example:

```
xs    ns    nss
```

The Layout Rule

In a sequence of definitions, each definition must begin in precisely the same column:

```
a = 10
b = 20
c = 30
```

```
a = 10
 b = 20
c = 30
```

```
  a = 10
b = 20
  c = 30
```

The layout rule avoids the need for explicit syntax to indicate the grouping of definitions.