CS 360 - Introduction to Haskell

Mark Boady

July 25, 2013

Quiz/Homework Notes

- Answer all of multipart questions.
- Non-Tail Recursive Length

```
(define (length L)
    (if (null? L)
    0
    (+ 1 (length (cdr L))))
)
```

Quiz/Homework Notes

- Tail recursive Length

```
(define (len L n)
    (if (null? L)
    n
    (len (cdr L) (+ 1 n)))
)
(define (length L) (len L 0))
```

Haskell

- GHCi is installed on tux (www.haskell.org)
  - ghci
- To quit
  - Ctrl-D (end of file)
  - :quit (Note the colon before the work!)
- Load Files
  - You can load files from the current directory
  - :load fact.hs

Math Functions

- The basic math functions use infix notation

```
Prelude> 9*10
90
Prelude> 9 + 8
17
Prelude> 10-20
-10
Prelude> 4 < 5
True
Prelude> 10-1.1
8.9
```

Defining Functions

- Defining a variable
  - let x = 10
- Defining a function
  - let gcd1 u v = if v == 0 then u else gcd1 v (mod u v)
- Calling a function

```
Prelude> gcd1 100 15
```

- Defining multi-line functions in the interpreter requires special syntax

```
Prelude> :{
Prelude| let { fact 0 = 1
Prelude| ; fact n = n * fact (n-1)
Prelude| }
Prelude| :}
```

Loading Files

- You can easily define functions in files and load them
- The contents of a file fact.hs

```
fact 0 = 1
fact n = n * fact ( n-1)
```

- Running it in Haskell

```
Prelude> :load fact.hs
[1 of 1] Compiling Main ( fact.hs, interpreted )
Ok, modules loaded: Main.
*Main> fact 34
295232799039604140847618609643520000000
```

Currying

- We can make new functions my applying a function with fewer inputs then required.
- The addition symbol takes two inputs A+B
- We can make a plus1 function by only giving the plus symbol one input
  - let plus1 = (1+)
- This works with higher order functions to
  - let square_list = map (\x -> x * x)
- The lambda syntax of Haskell is used above
  - (\x -> x * x)
- We can compose functions using the dot operator

```
*Main> let secretfunction = ((*3) . (2+))
*Main> secretfunction 5
21
```

List Commands

- Concat
  - 1 : [2,3,4]
- Merge
  - [1,2,3] ++ [4.5.6]
- Get the first element (car)
  - head [1,2,3]
- Get all but the head (cdr)
  - tail [1,2,3]
- We can save code with pattern matching like in ML
  - let (h:t) = [1,2,3,4]
- Null Test
  - null []

Advanced Lists

- There is a special shorthand for maping over lists
  - let square_list list = [x * x | x <- list]
- All evaluation is lazy, so infinite lists aren't a problem
  - let ones = 1 : ones
- We can use take and drop to get a subset of an infinite list
  - take 30 ones
  - take 5 (drop 4 [2..])
- The command [2..] gives all integers starting at 2

Finding all Primes

- The list of all primes is infinite
- It is a filtered version of all integers
- The **Sieve of Eratosthenes** filters primes out of the list of integers

```
let sieve (p:lis) = p :
    sieve [n | n <- lis, mod n p /= 0]
let primes = sieve [2..]
```

Twin Primes

- A twin prime is a prime p where p+2 or p-2 is also prime.
- We can tell is something is prime by searching our prime list

```
let orderedSearch a (h:t) = if a < h then False
    else (if a == h then True
    else orderedSearch a t )
```

- Now, we can ask if a number is a twin prime.

```
let isTwinPrime a = (orderedSearch (a+2) primes)
    || (orderedSearch (a-2) primes)
```

- We have a boolean function telling us which numbers to keep.

```
let tprimes = filter isTwinPrime primes
```

Programming Assignment 1

- Due 7/30/13 by midnight
- Scheme Functions
  - range
  - sequence
  - iterator
  - lookup
  - lookup-env
- Review Submission Guidelines