CSCI 312 Principles of Programming Languages

Haskell Tutorial

Qingsen Wang qwang06@email.wm.edu

April 7, 2017

Outline

1. Haskell Quick Start

2. Knight Placement Problem

The figure is from https://orlandoespinosa.files.wordpress.com/2014/05/relax-and-take-it-easy-orlando-espinosa.jpg

Haskell – How to play with your code

GHCI

```
> ghci
Prelude> :l test
Prelude> :r
Prelude> :t main
```

GHC

```
> ghc -o test test.hs
```

RUNHASKELL

```
> runhaskell test.hs
```

Haskell – Comments

```
-- some comments

{-
    Comments with multiple lines
-}
```

Haskell – Basic Types

- Int
- Integer can be unlimited
- Float
- Double
- Bool True or False
- Char `let c = 'a'` A string is a list of chars

The first letter is capitalized!

Haskell – Basic Operations

```
addEx = 7 + 3
subEx = 7 - 3
multEx = 7 * 3
divEx = 7 / 3

modEx = mod 7 3
modEx = 7 `mod` 3

powEx = 7 ^ 3
```

Make it an infix operator

Logic operators:

```
||   &&   not xor

and   or
```

apply to a list

Equality test:

```
==   /=
```

Haskell – List

Generate a list

```
emptyList = []

week = ["Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday"]

fromOneToTen = [1..10]

evenFromOneToTen = [2,4..10]

positiveInteger = [1..]

points = [[30,40], [20,50], [10,0]]
```

List comprehension

```
times3 = [x * 3 | x <-[1..10], x * 3 <= 50 ]

combinations = [(x,y)| x <-[1..10], y<-[1..10], x /= y]
```

Tuple, just like Python

For tuple, you may check `zip`, `unzip`, `fst` and `snd` for more information.

Haskell – List

List Operation

```
list1=[1,2,3,4,5]
list2=[6,7]
```

Op.	Example	Value	Op.	Example	Value
!!	`list1!!1`	`2`	elem	`elem 3 list1`	`True`
null	`null list1`	`False`	length	`length list1`	`5`
:	`0:list1`	`[0,1,2,3,4,5]`	++	`list1 ++ list2`	`[1,2,3,4,5,6,7]`
maximum	`maximum list1`	`5`	minimum	`minimum list1`	`1`
splitAt	`splitAt 2 list1`	`([1,2],[3,4,5])`	reverse	`reverse list1`	`[5,4,3,2,1]`

Haskell – List

List Operation (cont')

```
list1=[1,2,3,4,5]
list2=[6,7]
```

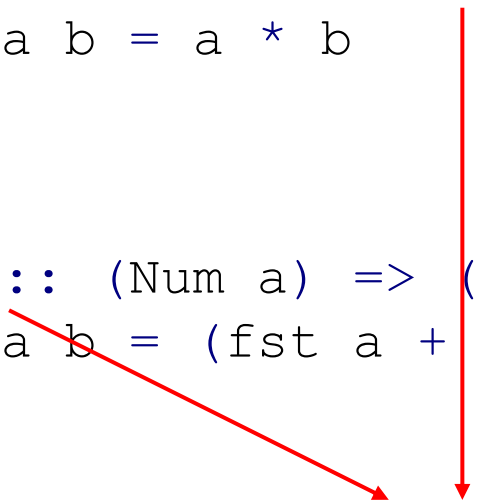
Op.	Example	Value	Op.	Example	Value
drop	`drop 2 list1`	`[3,4,5]`	take	`take 2 list1`	`[1,2]`
init	`init list1`	`[1,2,3,4]`	last	`last list1`	`5`
head	`head list1`	`1`	tail	`tail list1`	`[2,3,4,5]`
sum	`sum list1`	`15`	product	`product list1`	`12`

Haskell – Function

Declaration

```
areaOfRect :: Int -> Int -> Int
areaOfRect a b = a * b
```

```
addVectors :: (Num a) => (a, a) -> (a, a) -> (a, a)
addVectors a b = (fst a + fst b, snd a + snd b)
```



Optional ! Why not bother yourself?

Haskell – Function

Write a function

1

```
areaOfRect a b = a * b
```

2

```
num2Text 1 = "one"
num2Text 2 = "two"
num2Text 3 = "three"
num2Text x = "I don't care"
```

3

```
num2Text num
         | num == 1 = "one"
         | num == 2 = "two"
         | num == 3 = "three"
         | otherwise = "I don't care"
```

Haskell – Function

Write a function

4

```
bmiTell weight height
    | bmi <= 18.5 = "You're underweight!"
    | bmi <= 25.0 = "You're supposedly normal"
    | bmi <= 30.0 = "Lose some weight!"
    | otherwise   = "You're a whale, congratulations!"
    where bmi = weight / height ^ 2
```

5

```
tell [] = "The list is empty"
tell (x:[]) = "The list has one element: " ++ show x
tell (x:y:[]) = "The list has two elements: " ++
show x ++ " and " ++ show y
tell (x:y:_) = "This list is long. The first two
elements are: " ++ show x ++ " and " ++ show y
```

Pattern match is used a lot in Haskell

Haskell – Function

More recursive

```
factorial n = product [1..n]
```

1

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

2

```
myreverse [] = []
myreverse (x:xs) = myreverse xs ++ [x]
```

3

```
quicksort [] = []
quicksort (x:xs) = quicksort [a | a <- xs, a <= x]
++ [x] ++ quicksort [a | a <- xs, a > x]
```

Same line

Haskell – Function

More math functions


```
pi
exp
log
**
^
truncate
round
ceiling
floor
sin
cos
..
```

Haskell – Other

Google Google Google!

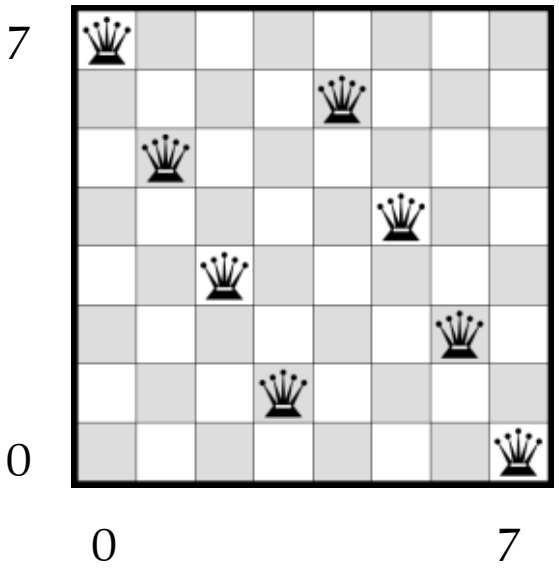
Learn You a Haskell for Great Good! http://learnyouahaskell.com/chapters

More Examples

1
```
fib = 1 : 1 : [ a+b | (a,b) <- zip fib (tail fib) ]
```

2
```
queens n = solve n
        where
        solve 0 = [ [ ] ]
        solve k = [ q:b | b <- solve (k-1), q <- [0..(n-
1)], safe q b]
        safe q b = and [ not (checks q b i) | i <-
[0..(length b - 1)]]
        checks q b i = q == b!!i || abs(q - b!!i) == i+1
```

Haskell – Other



Input: the size 8

Output: A list of solutions

One solution: [7,5,3,1,6,4,2,0]

```
queens n = solve n
        where
        solve 0 = [ [ ] ]
        solve k = [ q:b | b <- solve (k-1), q <- [0..(n-
1)], safe q b]
        safe q b = and [ not (checks q b i) | i <-
[0..(length b - 1)]]
        checks q b i = q == b!!i || abs(q - b!!i) == i+1
```

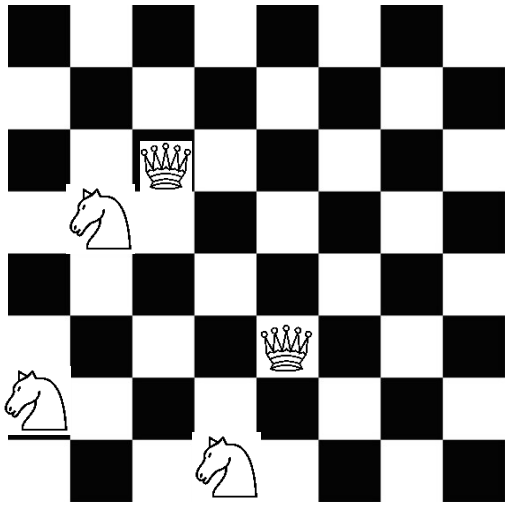
The figure is from http://www.aiai.ed.ac.uk/~gwickler/images/8-queens-config.png

Knight Placement Problem

Problem Description

The knight can't be caught by any queen

The knight can't catch any queen



Input:

A placement plan of queens

e.g. [0, 0, 6, 0, 3, 0, 0, 0]

Output:

All the possible places to place the knight

e.g. [[1,2], [1,2], [0], [0], [0], [7,8], [7,8], [2,4,5,7,8]]

Knight Placement Problem

One simple way: (simple for thinking, may not for implementation)
Check whether each cell is safe -> if yes, include this cell; otherwise skip

When is not safe?

$x_K == x_Q$

$y_K == y_Q$

$|x_K - x_Q| == |y_K - y_Q|$

$|x_K - x_Q| + |y_K - y_Q| == 3$

Feel free to implement your own ideas.

- Don’t get surprised if you can finish it within 20 lines.
- The index starts from 1 instead of 0
- Use the comments a lot so that we can understand you better
- You can assume all the inputs are valid

Reference

- Learn You a Haskell for Great Good! http://learnyouahaskell.com/chapters
- Hackage https://hackage.haskell.org/packages/
- Starting with Haskell https://www.fpcomplete.com/school/starting-with-haskell
- Wikibook – Haskell https://en.wikibooks.org/wiki/Haskell
- A Gentile Introduction to Haskell https://www.haskell.org/tutorial/
- Programming Languages Principles and Paradigms, 2nd Edition, by Allen B. Tucker and Robert E. Noonan