List Comprehension in Haskell

Principles of Programming Languages

List Comprehension

Set comprehension of MATHS

- They're normally used for building more specific sets out of general sets.
- A basic comprehension for a set that contains the first ten even natural numbers is

$$S = \{2 \cdot x \mid x \in \mathbb{N},\ x \leq 10\}$$

- The part before the pipe is called the output function,
- x is the variable,
- N is the input set and
- x <= 10 is the predicate.
- That means that the set contains the doubles of all natural numbers that satisfy the predicate.

In general

- In general, the list comprehension $[\ f\ x\ |\ x \leftarrow l\ ]$
  - stands for a list obtained by iterating over all the elements **x** of the list ***l***, and applying the function ***f*** to **x**.
  - The fragment **x <-** ***l*** of a list comprehension is called ***generator***.

```
> [ x + 1 | x ← fromTo 0 9 ]
[1,2,3,4,5,6,7,8,9,10]
```

- The list comprehension $[\ x\ |\ x \leftarrow l,\ p\ x\ ]$
  - stands for the sublist of *l* obtained by iterating over all the elements x of *l* such that *p* x == **True**. The part *p* x is called a ***filter***.

```
> [ x | x ← fromTo 0 9, x `mod` 2 == 0 ]
[0,2,4,6,8]
```

```
> [ x | x ← fromTo 0 100, prime x ]
[2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83,89,97]
```

List comprehension

$$S = \{2 \cdot x \mid x \in \mathbb{N},\ x \leq 10\}$$

- If we wanted to write that in Haskell, we could do something like take 10 [2,4..].

```
ghci> [x*2 | x <- [1..10]]
[2,4,6,8,10,12,14,16,18,20]
```

List Comprehension

- List comprehensions are very similar to set comprehensions.
- The list comprehension we could use is [x*2 | x <- [1..10]].
- **x** is drawn from [1..10] and for every element in [1..10] (which we have bound to x), we get that element, only doubled. Here's that comprehension in action.

List comprehension

Add a condition (predicate)

- Predicates go after the binding parts and are separated from them by a comma.

- Let's say we want only the elements which, doubled, are greater than or equal to 12

```
ghci> [x*2 | x <- [1..10], x*2 >= 12]
[12,14,16,18,20]
```

List comprehension

Another example

If we wanted all numbers from 50 to 100 whose remainder when divided with the number 7 is 3.

```
ghci> [ x | x <- [50..100], x `mod` 7 == 3]
[52,59,66,73,80,87,94]
```

filtering

- Weeding out lists by predicates is also called filtering. We took a list of numbers and we filtered them by the predicate.

List comprehension

- Example

We want a comprehension that replaces each odd number greater than 10 with "BANG!" and each odd number that's less than 10 with "BOOM!“.

If a number isn't odd, we throw it out of our list.

```
boomBangs xs = [ if x < 10 then "BOOM!" else "BANG!" | x <- xs, odd x]
```

The last part of the comprehension is the **predicate**. The function **odd** returns **True** on an odd number and **False** on an even one. The element is included in the list only if all the predicates evaluate to True.

List comprehension

- Not only can we have multiple predicates in list comprehensions (an element must satisfy all the predicates to be included in the resulting list),
- We can also draw from multiple lists.
- Example - If we have two lists, [2,5,10] and [8,10,11] and we want to get the products of all the possible combinations between numbers in those lists.

```
ghci> [ x*y | x <- [2,5,10], y <- [8,10,11]]
[16,20,22,40,50,55,80,100,110]
```

List comprehension

Let length’ be a new version of length.

```
length' xs = sum [1 | _ <- xs]
```

- _ means that we don't care what we'll draw from the list anyway so instead of writing a variable name that we'll never use, we just write _.
- This function replaces every element of a list with 1 and then sums that up.
- This means that the resulting sum will be the length of our list.

Strings – List comprehension

- Since strings are lists, we can use list comprehensions to process and produce strings.
- Example – a function that takes a string and removes everything except uppercase letters from it.

```
removeNonUppercase st = [ c | c <- st, c `elem` ['A'..'Z']]
```

```
ghci> removeNonUppercase "Hahaha! Ahahaha!"
"HA"
ghci> removeNonUppercase "IdontLIKEFROGS"
"ILIKEFROGS"
```

- *The predicate here does all the work. It says that the character will be included in the new list only if it's an element of the list ['A'..'Z'].*

Nested list comprehension

- Nested list comprehensions are also possible if you're operating on lists that contain lists. A list contains several lists of numbers.
- Consider the example - to remove all odd numbers without flattening the list.

```
ghci> let xxs = [[1,3,5,2,3,1,2,4,5],[1,2,3,4,5,6,7,8,9],[1,2,4,2,1,6,3,1,3,2,3,6]]
ghci> [ [ x | x <- xs, even x ] | xs <- xxs]
[[2,2,4],[2,4,6,8],[2,4,2,6,2,6]]
```

Next - Data