Programming Languages
Worksheet for 2. Introduction to Haskell

Shin-Cheng Mu

Sep. 20, 2018

If you have your notebook computer with you (and have Haskell Platform installed), start `ghci` and try the following tasks.

List Deconstruction

1. (a) What is the type of the function *head*? Use the command `:t` to find out the type of a value.

   (b) Since the input type of *head* is a list ($[a]$), let us try it on some input.

      i. *head* $[1, 2, 3] =$
      ii. *head* "abcde" $=$
      iii. *head* $[\,] =$

   (c) In words, what does the function *head* do?

2. (a) What is the type of the function *tail*?

   (b) Try *tail* on some input.

      i. *tail* $[1, 2, 3] =$
      ii. *tail* "abcde" $=$
      iii. *tail* $[\,] =$

(c) In words, what does the function *tail* do?

(d) For what xs is it always true that $head\ xs : tail\ xs = xs$?

3. (a) What is the type of the function *last*?

   (b) Try *last* on some input. Think about some input yourself.

   i. *last* =

   ii. *last* =

   iii. *last* =

   (c) In words, what does the function *last* do?

4. (a) What is the type of the function *init*?

   (b) Try *init* on some input. Think about some input yourself.

   i. *init* =

   ii. *init* =

   iii. *init* =

   (c) In words, what does the function *init* do?

   (d) What property does *init* and *last* jointly satisfy?

5. (a) What is the type of the function *null*?

   (b) Try *init* on some input. Think about some input yourself.

   i. *null* =

   ii. *null* =

   iii. *null* =

(c) Can you write down a definition of *null*, by pattern matching?

List Generation

1. What are the results of the following expressions?

   (a) $[0..25] \;=$

   (b) $[0, 2..25] \;=$

   (c) $[25..0] \;=$

   (d) $['a'..'z'] \;=$

   (e) $[1..] \;=$

2. What are the results of the following expressions?

   (a) $[x \mid x \leftarrow [1..10]] \;=$

   (b) $[x \times x \mid x \leftarrow [1..10]] \;=$

   (c) $[(x, y) \mid x \leftarrow [0..2], y \leftarrow \texttt{"abc"}] \;=$

   (d) What is the type of the expression above?

(e) $[x \times x \mid x \leftarrow [1..10], \mathit{odd}\ x]\ =$

3. What are the results of the following expressions?

   (a) $[(a, b) \mid a \leftarrow [1..3], b \leftarrow [1..2]] =$

   (b) $[(a, b) \mid b \leftarrow [1..2], a \leftarrow [1..3]] =$

   (c) $[(i, j) \mid i \leftarrow [1..4], j \leftarrow [(i + 1)..4]] =$

   (d) $[(i, j) \mid i \leftarrow [1..4], \mathit{even}\ i,\ j \leftarrow [(i + 1)..4], \mathit{odd}\ j] =$

   (e) $['a'|i \leftarrow [0..10]]\ =$

Combinators on Lists

1. (a) What is the type of the function !! (two exclamation marks)?

   (b) Try !! on some input. Think about some input yourself. Note that !! is an infix operator.

      i. $[1, 2, 3]\,!!\,1\ =$

      ii. $\qquad !! \quad =$

      iii. $\qquad !! \quad =$

   (c) In words, what does the function !! do?

2. (a) What is the type of the function *length*?

(b) Try *length* on some input.

   i. *length* =

   ii. *length* =

(c) In words, what does the function *length* do?

3. (a) What is the type of the function $+\!\!+$? (In ASCII one types `++`.)

   (b) Try $+\!\!+$ on some input. Think about some input yourself. Note that $+\!\!+$ is an infix operator.

   i.

   ii.

   (c) In words, what does the function $+\!\!+$ do?

   (d) Wait a minute... Both : and $+\!\!+$ appear to add elements to a list. How are they different?

4. (a) What is the type of the function *concat*?

   (b) Try *concat* on some input.

   i. *concat* =

   ii. *concat* =

   (c) In words, what does the function *concat* do?

5. (a) What is the type of the function *take*?

(b) Try *take* on some input. Since *take* expects an integer and list, try it on some extreme cases. For example, when the integer is zero, negative, or larger than the length of the list.

   i. *take* $=$

   ii. *take* $=$

   iii. *take* $=$

(c) In words, what does the function *take* do?

6. (a) What is the type of the function *drop*?

   (b) Try *drop* on some input. Like *take*, try it on some extreme cases.

      i. *drop* $=$

      ii. *drop* $=$

      iii. *drop* $=$

   (c) In words, what does the function *drop* do?

   (d) Does *take*, *drop*, and $(+\!\!+)$ together satisfy some properties?

7. (a) What is the type of the function *map*?

   (b) Try *map* on some input. It is a little bit harder, since *map* expects a functional argument.

      i. $map\ square\ [1, 2, 3, 4] =$

      ii. $map\ (1+)\ [1, 2, 3, 4] =$

      iii. $map\ (const\ 'a')\ [1..10] =$

   (c) In words, what does the function *map* do?

   (d) Is (1+) a function? Try it.

i. $(1+)\ 2 \ =$

ii. $((1+) \cdot (1+) \cdot (1+))\ 0 \ =$
where $(\cdot)$ is function composition.

Sectioning

- Infix operators are *curried* too. The operator $(+)$ may have type $Int \rightarrow Int \rightarrow Int$.
- Infix operator can be partially applied too.

$$\begin{aligned}(x\ \oplus)\ y &= x \oplus y \\ (\oplus\ y)\ x &= x \oplus y\end{aligned}$$

  - $(1\ +) :: Int \rightarrow Int$ increments its argument by one.
  - $(1.0\ /) :: Float \rightarrow Float$ is the "reciprocal" function.
  - $(/\ 2.0) :: Float \rightarrow Float$ is the "halving" function.

1. Define a function $doubleAll :: List\ Int \rightarrow List\ Int$ that doubles each number of the input list. E.g.

   - $doubleAll\ [1, 2, 3] = [2, 4, 6]$.

   - How do you define a new function? I'd suggest you to
     (a) create a new text file (using your favourite editor) in your current working directory (the directory you executed `ghci`). The file should have extension `.hs`.
     (b) Type your definitions in the file.
     (c) Load the file into `ghci` by the command `:l <filename>`.

2. Define a function $quadAll :: List\ Int \rightarrow List\ Int$ that multiplies each number of the input list by 4. Of course, it's cool only if you define $quadAll$ using $doubleAll$.

λ Abstraction

- Every once in a while you may need a small function which you do not want to give a name to. At such moments you can use the λ notation:
  - $map\ (\lambda x \rightarrow x \times x)\ [1, 2, 3, 4] = [1, 4, 9, 16]$
  - In ASCII λ is written \.

1. What is the type of $(\lambda x \rightarrow x + 1)$?
2. $(\lambda x \rightarrow x + 1)\ 2\ \ =$
3. What is the type of $(\lambda x \rightarrow \lambda y \rightarrow x + 2 \times y)$?
4. What is the type of $(\lambda x \rightarrow \lambda y \rightarrow x + 2 \times y)\ 1$?
5. $(\lambda x \rightarrow \lambda y \rightarrow x + 2 \times y)\ 1\ 2\ \ =$
6. What is the type of $(\lambda x\ y \rightarrow x + 2 \times y)$?
7. What is the type of $(\lambda x\ y \rightarrow x + 2 \times y)\ 1$?
8. $(\lambda x\ y \rightarrow x + 2 \times y)\ 1\ 2\ \ =$
9. Define *doubleAll* :: *List Int* $\rightarrow$ *List Int* again. This time using a λ expression.

10. **Pattern matching in λ.** To extract, for example, the two components of a pair
    (a) What is the type of $(\lambda(x, y) \rightarrow (y, x))$?

    (b) $(\lambda(x, y) \rightarrow (y, x))\ (1, \text{'}a\text{'})\ \ =$
    (c) Alternatively, try
    $(\lambda p \rightarrow (snd\ p, fst\ p))\ (1, \text{'}a\text{'})\ \ =$

Back to Lists

1. (a) What is the type of the function *filter*?

   (b) Try *filter* on some input.
      i. $\mathit{filter}\ \mathit{even}\ [1..10]\ =$
      ii. $\mathit{filter}\ (> 10)\ [1..20]\ =$
      iii. $\mathit{filter}\ (\lambda x \to x\ \text{`}\mathit{mod}\text{`}\ 3 == 1)\ [1..20]\ =$

   (c) In words, what does the function *filter* do?

2. (a) What is the type of the function *takeWhile*?

   (b) Try *takeWhile* on some input.
      i. $\mathit{takeWhile}\ \mathit{even}\ [1..10]\ =$
      ii. $\mathit{takeWhile}\ (< 10)\ [1..20]\ =$
      iii. $\mathit{takeWhile}\ (\lambda x \to x\ \text{`}\mathit{mod}\text{`}\ 3 == 1)\ [1..20]\ =$

   (c) In words, what does the function *takeWhile* do? How does it differ from *filter*?

   (d) Define a function $\mathit{squaresUpto} :: \mathit{Int} \to \mathit{List}\ \mathit{Int}$ such that $\mathit{squaresUpto}\ n$ is the list of all positive square numbers that are at most n. For some examples,
      - $\mathit{squaresUpto}\ 10 = [1, 4, 9]$.
      - $\mathit{squaresUpto}\ (-1) = [\,]$

3. (a) What is the type of the function *dropWhile*?

(b) Try $dropWhile$ on some input.

i. $dropWhile\ even\ [1..10]\ =$

ii. $dropWhile\ (<10)\ [1..20]\ =$

iii. $dropWhile\ (\lambda x \rightarrow x\ `mod`\ 3 == 1)\ [1..20]\ =$

(c) In words, what does the function $dropWhile$ do?

4. (a) What is the type of the function zip?

(b) Try zip on some input.

i. zip [1..10] "abcde" $=$

ii. zip "abcde" [0..] $=$

iii. zip $=$

(c) In words, what does the function zip do?

(d) Define $positions :: Char \rightarrow String \rightarrow List\ Int$, such that $positions\ x\ xs$ returns the positions of occurrences of x in xs. E.g.

- $positions$ 'o' "roodo" $= [1,2,4]$.

Check the handouts if you just cannot figure out how.

(e) What if you want only the position of the *first* occurrence of x? Define $pos :: Char \rightarrow String \rightarrow Int$, by reusing $positions$.

Morals of the Story

- Lazy evaluation helps to improve modularity.

- List combinators can be conveniently re-used. Only the relevant parts are computed.

- The combinator style encourages “wholemeal programming”.
  - Think of aggregate data as a whole, and process them as a whole!

Fold on Lists

Now we’ve finally come to the most important function on list we will introduce: the fold.

1. What is the type of the function *foldr*?

2. Try the following:
   (a) $\mathit{foldr}\ (+)\ 0\ [1..10]\ =$
   (b) $\mathit{foldr}\ (\times)\ 1\ [1..10]\ =$

One way to look at $\mathit{foldr}\ (\oplus)\ e$ is that it replaces $[\,]$ with e and $(:)$ with $(\oplus)$:

$$\begin{array}{ll} & \mathit{foldr}\ (\oplus)\ e\ [1,2,3,4] \\ = & \mathit{foldr}\ (\oplus)\ e\ (1:(2:(3:(4:[\,])))) \\ = & 1 \oplus (2 \oplus (3 \oplus (4 \oplus e))). \end{array}$$

1. Define $\mathit{prod} :: \mathit{List}\ \mathit{Int} \to \mathit{Int}$, which computes the product of a list of numbers, using *foldr*. E.g.
   - $\mathit{prod}\ [2,3,4] = 24$.

2. (a) Try the following
      i. $3\ `\mathit{max}`\ 5\ =$
      ii. $5\ `\mathit{max}`\ 3\ =$

(b) Define $myMaximum :: List\ Int \to Int$ that returns the maximum element in a list, using max and $foldr$. (Hint: the largest Int is denoted by $maxBound$ in Haskell.) (I want you to define $myMaximum$, because there is already a $maximum$ doing the same job.)

3. What does $foldr\ (:)\ [\,]$ do?

4. Define $myLength :: List\ a \to Int$ that computes the same function as $length$, using $foldr$. Check the handouts if you just cannot figure out how.

5. Define $myMap :: (a \to b) \to List\ a \to List\ b$ computes the same function as map, using $foldr$. Check the handouts if you just cannot figure out how.

6. Define $append :: List\ a \to List\ a \to List\ a$ such that $append\ xs\ ys$ is the same as $xs +\!\!+\ ys$. Of course, do not use $+\!\!+$ but use $foldr$. Check the handouts if you just cannot figure out how.

In fact, *any* function that takes a list as its input can be written in terms of $foldr$ — although it might not be always practical.