Fun With Haskell: Sample Problems and Testing

Nathaniel Wesley Filardo

January 18, 2012

Metadata
Questions?

- Any questions from last time?

Metadata
Overview of today

- Mostly intended for people to ask questions.
- Review using exercises from CalTech [1].
- Also: introduction to automated model checking:
  - QuickCheck.
  - SmallCheck / LazySmallCheck.

Mathematical Examples
Appending Lists Using `foldr`

- We're going to write our own ++.
- Ultimately, using foldr. First, directly.
- What do we need?

Mathematical Examples
Appending Lists Using `foldr`

- We're going to write our own ++.
- Ultimately, using foldr. First, directly.
- What do we need?
    - Specification!

Mathematical Examples
Appending Lists Using `foldr`

- We're going to write our own ++.
- Ultimately, using foldr. First, directly.
- What do we need?
  - Specification!
  - "Append takes two lists, `a` and `b`, and returns a list composed of the elements of `a` in `a`-order followed by the elements of `b` in `b`-order."

Mathematical Examples
Appending Lists Using `foldr`

- "Append takes two lists, `a` and `b`, and returns a list composed of the elements of `a` in `a`-order followed by the elements of `b` in `b`-order."

Mathematical Examples
Appending Lists Using `foldr`

- "Append takes two lists, `a` and `b`, and returns a list composed of the elements of `a` in `a`-order followed by the elements of `b` in `b`-order."
- OK, yes, but...

Mathematical Examples
Appending Lists Using `foldr`

- "Append takes two lists, `a` and `b`, and returns a list composed of the elements of `a` in `a`-order followed by the elements of `b` in `b`-order."
- OK, yes, but...
- How about we try some induction?
  - What to induct over?

Mathematical Examples
Appending Lists Using `foldr`

- "Append takes two lists, `a` and `b`, and returns a list composed of the elements of `a` in `a`-order followed by the elements of `b` in `b`-order."
- OK, yes, but...
- How about we try some induction?
  - What to induct over?
  - What's the base case?
  - What's the induction step?

Mathematical Examples
Appending Lists Using `foldr`

- "Append takes two lists, `a` and `b`, and returns a list composed of the elements of `a` in `a`-order followed by the elements of `b` in `b`-order."
- Induct on the length of `a`.
  - Base case?
  - Induction case?

Mathematical Examples
Appending Lists Using `foldr`

- "Append takes two lists, `a` and `b`, and returns a list composed of the elements of `a` in `a`-order followed by the elements of `b` in `b`-order."
- Induct on the length of `a`.
  - Base case?
    - `a` is nil. What should append do?
  - Induction case?

Mathematical Examples
Appending Lists Using `foldr`

- "Append takes two lists, `a` and `b`, and returns a list composed of the elements of `a` in `a`-order followed by the elements of `b` in `b`-order."
- Induct on the length of `a`.
  - Base case?
    - `a` is nil. What should append do?
  - Induction case?
    - `a` is cons of `ah` and `at`. What should append do?

Mathematical Examples
Appending Lists Using `foldr`

- a is nil. What should append do?

```
appendList [] b = b
```

Mathematical Examples
Appending Lists Using `foldr`

- `a` is nil. What should append do?

```
appendList [] b = b
```

- `a` is cons of `ah` and `at`. What should append do?

```
appendList (ah:at) b = ah : (appendList at b)
```

Mathematical Examples
Appending Lists Using `foldr`

- So, all together:

```
appendList []       b = b
appendList (ah:at) b = ah : (appendList at b)
```

- Let's do a quick sanity check using QuickCheck:

```
> import Test.QuickCheck
> quickCheck (\a b -> appendList a b == a ++ b)
+++ OK, passed 100 tests.
```

Mathematical Examples
Appending Lists Using `foldr`

- Or a more verbose sanity check using QuickCheck:

```
> verboseCheck (\a b -> appendList a b
                                == a ++ b)
Passed:
[()]
[(),(),(),(),(),(),(),(),(),(),()]
```

- Hey! That's only sort of helpful!

Mathematical Examples
Appending Lists Using `foldr`

- Or a more verbose sanity check using QuickCheck:

```
> verboseCheck (\a b -> appendList a b
                                == a ++ b)
Passed:
[()]
[(),(),(),(),(),(),(),(),(),(),()]
```

- Hey! That's only sort of helpful!
- `appendList` and `++` are polymorphic; QuickCheck chose to use `()`.

Mathematical Examples
Appending Lists Using `foldr`

- Or a more verbose sanity check using QuickCheck:

```
> verboseCheck (\a b -> appendList a b
                                == a ++ b)
Passed:
[()]
[(),(),(),(),(),(),(),(),(),(),()]
```

- Hey! That's only sort of helpful!
- `appendList` and `++` are polymorphic; QuickCheck chose to use `()`.
- We'd rather it test on something with more than one constructor.

Mathematical Examples
Appending Lists Using `foldr`

- Let's tell it to use Ints:

```
> :set -XScopedTypeVariables
> verboseCheck (\a (b :: [Int]) ->
        appendList a b == a ++ b)
... lots of numbers ...
+++ OK, passed 100 tests.
```

- Much better!

Mathematical Examples
Appending Lists Using `foldr`

- Let's tell it to use Ints:

```
> :set -XScopedTypeVariables
> verboseCheck (\a (b :: [Int]) ->
        appendList a b == a ++ b)
... lots of numbers ...
+++ OK, passed 100 tests.
```

- Much better!
- (Could use `{-# LANGUAGE ScopedTypeVariables #-}` at the top of a file, too.)

Mathematical Examples
Appending Lists Using `foldr`

- Now, remember foldr?

```
foldr f z [] = z
foldr f z (x:xs) = x `f` (foldr f z xs)
```

Mathematical Examples
Appending Lists Using `foldr`

- Now, remember foldr?

```
foldr f z [] = z
foldr f z (x:xs) = x `f` (foldr f z xs)
```

- Alternatively:
  - Given any `g`:

```
g [] = z
g (x:xs) = f x (g xs)
```

  - Then `g = foldr f z`.

Mathematical Examples
Appending Lists Using `foldr`

- So we have:

```
appendList []       b = b
appendList (ah:at) b = ah : (appendList at b)
```

- Does that look like?

```
g [] = z
g (x:xs) = f x (g xs)
```

- Sort of.

Mathematical Examples
Appending Lists Using `foldr`

- So we have:

```
appendList []       b = b
appendList (ah:at) b = ah : (appendList at b)
```

- Does that look like?

```
g [] = z
g (x:xs) = f x (g xs)
```

- Sort of.
- Flip arguments:

```
alf b []       = b
alf b (ah:at) = ah : (alf b at)
```

Mathematical Examples
Appending Lists Using `foldr`

- So now we have

```
alf b []       = b
alf b (ah:at) = ah : (alf b at)
```

- So the universal property of foldr tell us:

```
alf b = foldr (:) b
```

- So, making the other argument explicit:

```
alf b a = foldr (:) b a
```

- And, finally, recalling the definition of `alf`:

```
appendList a b = foldr (:) b a
```

Mathematical Examples
Appending Lists Using `foldr`

Just checking:

```
> quickCheck (\a (b :: [Int]) ->
                foldr (:) b a == a ++ b)
+++ OK, passed 100 tests.
```

Mathematical Examples
Inserting Into an Ordered List

- The core of insertion sort.
- Specification?
  - Given an ascending-ordered list `ys` of orderable things, and another thing of the same type `x`, return the ordered list containing `x` and all elements of `ys`.
  - Note: if `x` is equal to something in `ys`, the above specification says we return a list with two equal elements.

Mathematical Examples
Inserting Into an Ordered List

- Base case?

```
ascInsert x [] = [x]
```

- Inductive step?

```
ascInsert x (y:ys) = -- ...
```

  - What do we need to do?

Mathematical Examples
Inserting Into an Ordered List

```
ascInsert x (y:ys) = -- ...
```

- What do we need to do?

Mathematical Examples
Inserting Into an Ordered List

```
ascInsert x (y:ys) = -- ...
```

- What do we need to do?
- `compare x y`:

```
ascInsert x (y:ys) = case compare x y of
  {- ... -}
```

Mathematical Examples
Inserting Into an Ordered List

```
ascInsert x (y:ys) = -- ...
```

- What do we need to do?
- `compare x y`:

```
ascInsert x (y:ys) = case compare x y of
  {- ... -}
```

- Easy case arm first:

```
ascInsert x (y:ys) = case compare x y of
  GT -> y : (ascInsert x ys)
  {- ... -}
```

Mathematical Examples
Inserting Into an Ordered List

```
ascInsert x [] = [x]
ascInsert x (y:ys) = case compare x y of
  GT -> y : (ascInsert x ys)
  {- ... -}
```

Mathematical Examples
Inserting Into an Ordered List

```
ascInsert x [] = [x]
ascInsert x (y:ys) = case compare x y of
  GT -> y : (ascInsert x ys)
  {- ... -}
```

- Only one other arm! So, in full:

```
ascInsert x [] = [x]
ascInsert x (y:ys) = case compare x y of
  GT -> y : (ascInsert x ys)
  _  -> x : y : ys
```

Mathematical Examples
Inserting Into an Ordered List

```
ascInsert x [] = [x]
ascInsert x (y:ys) = case compare x y of
  GT -> y : (ascInsert x ys)
  {- ... -}
```

- Only one other arm! So, in full:

```
ascInsert x [] = [x]
ascInsert x (y:ys) = case compare x y of
  GT -> y : (ascInsert x ys)
  _  -> x : y : ys
```

- Note: not the right structure for `foldr`!

Mathematical Examples
Inserting Into an Ordered List

- Now, to test it!

Mathematical Examples
Inserting Into an Ordered List

- Now, to test it!
- Can't use arbitrary inputs! List needs to be ordered!
- One answer: ==> combinator from QuickCheck.

```
ascTest (x :: Int) ys =
  sorted ys ==> sorted (ascInsert x ys)
```

Mathematical Examples
Inserting Into an Ordered List

- Now, to test it!
- Can't use arbitrary inputs! List needs to be ordered!
- One answer: `==>` combinator from QuickCheck.

```
ascTest (x :: Int) ys =
  sorted ys ==> sorted (ascInsert x ys)
```

- Problem: define `sorted`.

```
sorted [] = True
sorted [x] = True
sorted (x1:x2:xs) = x1 <= x2 && sorted (x2:xs)
```

Mathematical Examples
Inserting Into an Ordered List

```
ascTest (x :: Int) ys =
  sorted ys ==> sorted (ascInsert x ys)
```

- So, run QuickCheck!

```
*Main> quickCheck ascTest
*** Gave up Passed only 46 tests.
```

Mathematical Examples
Inserting Into an Ordered List

```
ascTest (x :: Int) ys =
  sorted ys ==> sorted (ascInsert x ys)
```

- So, run QuickCheck!

```
*Main> quickCheck ascTest
*** Gave up Passed only 46 tests.
```

- That doesn't sound good.
- What happened?

Mathematical Examples
Inserting Into an Ordered List

```
ascTest (x :: Int) ys =
  sorted ys ==> sorted (ascInsert x ys)
```

- So, run QuickCheck!

```
*Main> quickCheck ascTest
*** Gave up Passed only 46 tests.
```

- That doesn't sound good.
- What happened?
  - Most lists aren't sorted!
  - Our pre-condition said to throw almost all of them out!

Mathematical Examples
Inserting Into an Ordered List

- One of the motivations for LazySmallCheck.
- The ==> combinator could use laziness to determine when it's generated an unacceptable input and stop early.
- So, if we use the ==> operator from LazySmallCheck:
  - Note: idential syntax, different imports.
  - Or see the whole file for **qualified names**.

```
ascTestL (x :: Int) ys =
  sorted ys ==> sorted (ascInsert x ys)
```

- And run its `smallCheck` function:

```
> smallCheck 2 ascTestL
OK, required 2 tests at depth 0
OK, required 10 tests at depth 1
OK, required 43 tests at depth 2
```

Mathematical Examples
Inserting Into an Ordered List

- QuickCheck has some special cases for this:
  - Notably, the `OrderedList a` type, which is guaranteed to only generate ordered lists.

```
ascTest2 (x :: Int) (Ordered ys) =
  sorted ys ==> sorted (ascInsert x ys)
```

  - And so if we run that...

```
*Main> quickCheck ascTest2
+++ OK, passed 100 tests.
```

- See the documentation for more.

Mathematical Examples
Inserting Into an Ordered List

- So, `foldr` naturally captures "in-order" traversal of a list.
- What about in-order traversals of other structures?

Next time

- You tell me?

Bib

Available from: `http://courses.cms.caltech.edu/cs11/material/haskell/index.html`.