Programming Paradigms 2024

Session 10: Functors and applicative functors

Hans Hüttel

19 November 2024

Our plan for today

1. The learning goals
2. Presentations of the preparation problems
3. Problem no. 1
4. Problem no. 2
5. Break
6. Problem no. 3
7. Problem no. 4
8. If time allows: More problems at your own pace.
9. We evaluate today's session – please stay until the end!

Learning goals

- To understand the notion of a functor
- To understand how familiar type constructs such as Maybe types and lists can be instances of the type class Functor
- To understand the notion of applicative functions
- To understand the pure and <∗> operations and how they can be used in the applicative style of programming in Haskell To be able to apply the notions of functors and applicative functors for writing well-structured Haskell programs

Preparation problem – Onions and functors

An onion consists of a finite number of layers surrounding a core. In this problem, we let the core be a value. Here is an onion with six layers and core "bingo".

Define Onion as an instance of Functor. *Hint:* Be inspired by how the book shows how one can let the Tree type become an instance of Functor.

Preparation problem – The applicative laws

Check that the first two applicative laws at the top of page 163 hold for the Maybe type. *Hint:* Use the definitions of pure and <*> on page 160.

Problem 1 – Unbounded trees (20 minutes)

The type of unbounded trees UTree is given by

```
data UTree a = Node a [UTree a]
```

Define an instance of Functor for UTree.

Problem 2 – Arrow types (15 minutes)

The function type constructor `((->)r)` is defined such that `f a` will be `(r -> a)`.

Define an instance of Functor for this type constructor. *Hint:* Look at the type of `fmap`.

In order to test your solution, add the following at the start of the file containing your code:

```
import qualified Prelude
import Prelude hiding (Functor, fmap)
```

Problem 3 – Funny star (15 minutes)

For the applicative functor for lists we have a definion of the "funny star" composition <*> on page 160.

Give an alternative *recursive* definition of it that uses fmap.

Problem 4 – Products of threes (15 minutes)

Use the fact that the list type can be seen as an applicative functor to define a function prodthree that takes three lists of numbers and computes the list of all products of triples of numbers in the list. As an example,
prodthree [1,2,3] [4,5,6] [7,8,9] should give us the list

```
[28,32,36,35,40,45,42,48,54,56,64,72,70,80,
90,84,96,108,84,96,108, 105,120,135,126,
144,162]
```

Hint: Somewhere a funny star keeps shining.

Evaluation

- What did you find difficult?
- What surprised you?
- What went well?
- What could be improved? How does the setup work this time?
- Is there a particular problem that we should follow up on with a short video?