Chalmers | GÖTEBORGS UNIVERSITET
Alejandro Russo, Computer Science and Engineering

Advanced Functional Programming TDA342/DIT260

Tuesday, August 27th, 2018, Maskinhuset, 14:00hs (4hs)

Alejandro Russo, tel. 0729744968

- The maximum amount of points you can score on the exam: 60 points. The grade for the exam is as follows:

  Chalmers: **3**: 24 - 35 points, **4**: 36 - 47 points, **5**: 48 - 60 points.
  GU: Godkänd 24-47 points, Väl godkänd 48-60 points
  PhD student: 36 points to pass.

- Results: within 21 days.

- **Permitted materials (Hjälpmedel):** Dictionary (Ordlista/ordbok).

  You may bring up to two pages (on one A4 sheet of paper) of pre-written notes — a "summary sheet". These notes may be typed or handwritten. They may be from any source. If this summary sheet is brought to the exam it must also be handed in with the exam (so make a copy if you want to keep it).

- **Notes:**

  - Read through the paper first and plan your time.
  - Answers preferably in English, some assistants might not read Swedish.
  - If a question does not give you all the details you need, you may make reasonable assumptions. Your assumptions must be clearly stated. If your solution only works under certain conditions, state them.
  - Start each of the questions on a new page.
  - The exact syntax of Haskell is not so important as long as the graders can understand the intended meaning. If you are unsure just put in an explanation of your notation.
  - Hand in the summary sheet (if you brought one) with the exam solutions.
  - As a recommendation, consider spending around 1h 20 minutes per exercise. However, this is only a recommendation.
  - To see your exam: *by appointment (send email to Alejandro Russo)*

```
example2 :: Fac Integer
example2 = View "Leonor" (View "Martin"
                                  (Raw 42)
                                  (Raw 100)
                         )
                         (Raw 7)
> projection example2 ["Leonor","Martin"] []
Just 42
> projection example2 ["Leonor"]          ["Martin"]
Just 100
> projection example2 ["Martin"]          ["Leonor"]
Just 7
> projection example2 ["Leonor"]          []
Nothing
> projection example2 []                  []
Nothing
```

a) Write the function *projection*. *(5p)*

b) In this part, we will build computations based on faceted values by giving it a monadic structure! For that, we start by extending our data type definition as follows to encode the monadic operations in a deep-embedded fashion.

```
data Fac a where
  Raw  :: a → Fac a
  View :: User → Fac a → Fac a → Fac a
  Bind :: Fac a → (a → Fac b) → Fac b
```

If *Fac* is a monad, then programs can safely manipulate faceted values for different users' views without compromising their confidentiality. For instance, the following code

```
p1 = example1 >>= λx → return (x + 1)
p2 = example2 >>= λx → return (x + 1)
```

denotes the faceted value *View* `"Leonor"` (*Raw* 43) (*Raw* 8)—observe that the bind applies $x + 1$ to every leaf in the tree-structured denoted by faceted values. More concretely,

```
> projection p1 ["Leonor"] []
Just 43
> projection p1 []          ["Leonor"]
Just 8
> projection p2 ["Leonor"] ["Martin"]
Just 101
```

Give an instance of the type-class *Monad* and extend your definition of *projection* to consider *Bind*. *(7p)*

c) Transform your code to implement *Bind* as shallow-embedded. *(8p)*

Problem 3: (Miscellaneous)

a) Write a function of type $a \to (b, b)$, where a and b are polymorphic types. *(2p)*

b) Write a function of type $a \to (b, c)$, where a, b, and c are polymorphic types. *(2p)*

c) Give an example of a data type definition which is not a functor and explain why this is the case. You should be clear why *fmap* cannot be implemented.

(4p)

d) Define a data type that is a functor and not applicative. Define *fmap* and justify why the instance *Applicative* cannot be defined.

(4p)

e) Let us consider the alternative implementation you did for monads from the previous exercise.

$$
\begin{aligned}
&\mathit{return}' :: a \to m\ a \\
&\mathit{join} \quad\ :: m\ (m\ a) \to m\ a \\
&\mathit{fmap}' \ :: (a \to b) \to m\ a \to m\ b
\end{aligned}
$$

Now, let us consider the function $(+)$ with the following type signature:

$$(+) :: \mathit{Num}\ a \Rightarrow a \to a \to a$$

What is the output of *join* $(+)$? Hint: remember the reader monad.

(8p)