EXAM FUNCTIONAL PROGRAMMING
Thursday the 5th of November 2015, 17.00 h. - 20.00 h.

Name:
Student number:

Before you begin: Do not forget to write down your name and student number above. If necessary, explain your answers (in English or Dutch). For multiple choice questions, clearly circle what you think is the (one and only) best answer. Use the empty boxes under the other questions to write your answer and explanations in. Use the blank paper provided with this exam only as scratch paper (kladpapier). At the end of the exam, only hand in the filled-in exam paper. Answers will not only be judged for correctness, but also for clarity and conciseness. A total of one hundred points can be obtained; divide by 10 to obtain your grade. Good luck!

In any of your answers below you may (but do not have to) use the following well-known Haskell functions/operators: *zipWith*, *zip*, *id*, *concat*, *foldr* (and variants), *map*, *filter*, *const*, *all*, *any*, *flip*, *fst*, *snd*, *not*, (.), *elem*, *take*, *drop*, *takeWhile*, *dropWhile*, *head*, *tail*, *repeat*, *replicate*, (++), *lookup*, *max*, *min* and all members of the type classes *Eq*, *Num*, *Ord*, *Show* and *Read*.

1. (i) To commemorate the 200th birthday of George Boole, define a type class, call it *BoolA* with a single argument *a*, to represent the concept of a boolean algebra. It should support functions that represent binary conjunction (*andb*), binary disjunction (*orb*) and unary negation (*notb*), as well as two constants *bot* and *top*. Here *top* represents the neutral element of *andb*, and *bot* that of *orb*.

   . . . /**6**

   (ii) Give an instance definition for *BoolA Bool*.

   . . . /**6**

(iii) This exercise was disqualified from the exam due to its incorrect phrasing.

(iv) Give an instance definition for values of type $[\,a\,]$ (for any a that is an instance of $BoolA$) that extends the operations to lists elementwise. For example, $andb\ [\,True, False\,]\ [\,True, True, False\,] = [\,andb\ True\ True, andb\ False\ True\,] = [\,True, False\,]$. Note that the length of the result is the same as the minimum of the length of the two arguments. Again make sure that top and bot are in fact neutral for the right operator.

$\boxed{\ldots/\mathbf{6}}$

(v) Why is it not a good idea to have $andb$ and orb return a list that is as long as the longest (this could then be done by correctly padding the shorter list)? In which situation would this not be a problem?

$\boxed{\ldots/\mathbf{4}}$

2. (i) Implement the function $minimum :: (Ord\ a) \Rightarrow [a] \rightarrow a$ for lists that computes the smallest element of a list. It should display a nice run-time error message when you pass the empty list. Also, you must implement it with a fold (choose one of *foldr*, *foldr1*, *foldl*, and *foldl1*).

   .../**6**

   (ii) Which of the other three could you have used as well? Which do you think are most suitable to use, and why?

   .../**3**

   (iii) Reflect on why it would or would not be a good idea to use a strict fold function, such as *foldl'*.

   .../**3**

   (iv) A fellow student has defined a function $sort :: (Ord\ a) \Rightarrow [a] \rightarrow [a]$ for sorting a list. Define a QuickCheck property that verifies that the first element of a sorted list is the smallest in the list. Make sure the QuickCheck property never crashes: only non-empty lists may contribute to the test set.

   .../**6**

3. $\boxed{\ldots/\mathbf{20}}$ The following multiple choice questions are each worth 5 points.

(i) What is the type of *flip foldr*, where $flip :: (a \to b \to c) \to b \to a \to c$ switches the arguments a and b of a function.

a. $(a \to c) \to [a \to (a \to c) \to c] \to a \to c$
b. $b \to (a \to b \to b) \to [a] \to b$
c. It is type incorrect, since *foldr* takes three arguments.
d. It is type incorrect, but for another reason then the one listed under (c).

(ii) Which expressions are equivalent, i.e., can replace each other in any context?

a. *takeWhile p . dropWhile p* and *dropWhile p . takeWhile p*
b. *takeWhile p . dropWhile q* and *takeWhile* (\ x -> q x && p x)
c. *dropWhile p . takeWhile q* and *takeWhile q . dropWhile p*
d. *dropWhile p . dropWhile q* and *takeWhile* (\ x -> p x && q x)

(iii) I A general purpose language cannot be embedded in another general purpose language.
II A major advantage of shallow over deep EDSLs is that you can optimize EDSL programs before running them.

a. Both I and II are true
b. Only I is true
c. Only II is true
d. Both I and II are false

(iv) Assume $\perp$ is an expression that always crashes, and $f :: Int \to Int \to Int$ is a function that always crashes. Which of the following statements is false:

a. *seq* (f 1) 3 crashes.
b. (\ x -> ()) $\perp$ equals ().
c. *head* (*map* $\perp$ $[1, 2]$) crashes.
d. For basic types, *seq* and *deepseq* have the same behaviour.

4. Given the following Haskell definition where $g :: Int \to Maybe\ Int$ and $h :: a \to Maybe\ a$

```
f w = do
  x <- g w
  let xs = do
    z <- [1,2]
    v <- ['a','b']
    return (z,v)
  y <- h (snd (head xs))
  return y
```

Complete the following explanation by filling in the gaps:

In the *Maybe* monad, __________ signals failure and __________ a successful computation. In the above program, the type of x is __________, the type of y is __________, the type of xs is ______________, and the type of f is ____________________. If we call f and would print the value of xs to the screen then we'd see ______________________________. $\boxed{\ldots/\mathbf{10}}$

5. The following definitions make *Maybe* into a monad:

$$
\begin{aligned}
&(1)\ f \gg\!= g &&= \textbf{case}\ f\ \textbf{of}\\
& && \quad \mathit{Nothing} \rightarrow \mathit{Nothing}\\
& && \quad \mathit{Just}\ x \quad \rightarrow g\ x\\
&(2)\ \mathit{return}\ x = \mathit{Just}\ x
\end{aligned}
$$

The following is often called the third monad law:

$$(m \gg\!= f) \gg\!= g = m \gg\!= (\backslash\, x \rightarrow f\ x \gg\!= g)$$

(i) Prove that the law holds if m equals *Nothing*. Hint: to simplify the proof you may first want to prove Lemma A that says that $\mathit{Nothing} \gg\!= f = \mathit{Nothing}$.

$\boxed{\ldots/\mathbf{5}}$

(ii) Prove that the law holds for $m = \mathit{Just}\ x$. Again, it may be wise to first prove a Lemma B that $\mathit{Just}\ x \gg\!= f = f\ x$.

$\boxed{\ldots/\mathbf{6}}$

6. Induction

$$(1)\ [\,] \qquad ++\ ys = ys$$
$$(2)\ (x : xs)\ ++\ ys = x : (xs\ ++\ ys),$$

$$(3)\ foldr\ op\ e\ [\,] = e$$
$$(4)\ foldr\ op\ e\ (x : xs) = op\ x\ (foldr\ op\ e\ xs)$$

(i) Given that *op* is an associative binary operator, and *e* a neutral element of *op*, prove by induction that

$$foldr\ op\ e\ (xs\ ++\ ys) = op\ (foldr\ op\ e\ xs)\ (foldr\ op\ e\ ys)\ .$$

. . . /**15**