• The problems marked “I” are individual assignments and are due by Wednesday Jan. 31st before class. They do not require implementing Haskell functions.

• The problems marked “GP” are group projects (a group has exactly three students) and require writing Haskell code. The reports for these are due by Wednesday Feb. 7th before class, i.e., you have to submit a group report. Details and guidelines for writing reports will be given next week; also check the class Web page regularly for new announcements and material.

INDIVIDUAL ASSIGNMENT 3

Problem 1 (I, Lists) Haskell uses the data constructor “:” to construct a new list from a given element \(x\) and a list \(xs\), and the function “++” to append two lists. Hence their type signatures are (recall that “\(a\)” is a type variable):

\[
\begin{align*}
(:) & : a \to [a] \to [a] \\
(++) & : [a] \to [a] \to [a]
\end{align*}
\]

Which of the following equations are correct? (Give a short explanation)

\[
\begin{align*}
a) \: \emptyset : xs &= xs \\
b) \: \emptyset : [x] &= [[], xs] \\
c) \: xs : \emptyset &= xs \\
d) \: xs : [y] &= [xs] \\
e) \: x : y &= [x, y] \\
f) \: (x : xs) ++ ys &= x : (xs ++ ys)
\end{align*}
\]

Problem 2 (I, Reductions) Consider the Haskell function definitions

\[
\begin{align*}
sqr \; x &= x^2 \\
\text{first} \; x \; y &= x
\end{align*}
\]

and reduce the expression \(sqr \; (\text{first} \; (sqr \; 2) \; (sqr \; 3))\) according to the strategies

a) leftmost innermost, and

b) leftmost outermost.

GROUP PROJECT 1

Problem 3 (GP, Powerset) Define a Haskell function powerset that returns for a set \(A\) (represented as a list of type \([a]\)) the powerset \(2^A\) of \(A\) (with type \([\text {[a]}]\), i.e., represented as a list of lists).

Example: \(\text{powerset} \; [1, 2] = [[1, 2], [1], [2], []]\) (the order of the elements in the results may be different).

Problem 4 (GP, List Comprehensions) For the following problems, use list comprehensions:

a) A number \(n\) is called perfect, if it is equal to the sum of its factors \(i \in \{1, \ldots, n-1\}\). For example, 28 is perfect, since \(28 = 1 + 2 + 4 + 7 + 14\). Define in Haskell the list of all perfect numbers.

b) A number \(n\) is called square free if no square number (apart from 1) is a factor of \(n\). Define in Haskell the list of all square free numbers.
Problem 5 (GP, XML (to be continued)) XML documents are similar to HTML documents (but with user-defined tags instead of the fixed HTML tags) and can be regarded as trees. Consider, for example, the XML document

```
<books>
  <book year="1995" edition="1">
    <title>Foundations of Databases</title>
    <author>Abiteboul</author>
    <author>Rull</author>
    <author>Vianu</author>
  </book>
  <book edition="1" year="2000">
    <title>Constraint Databases</title>
    <editor>Kuper</editor>
    <editor>Libkin</editor>
    <editor>Paredaens</editor>
  </book>
</books>
```

It has a single root element which is delimited by the start tag `<books>` and the end tag `</books>`. The `books` element has two children elements of type `book`. Each book element has a `year` and an `edition` attribute (in the start tag) and a `title` child element. A book element has `author` or `editor` children elements. Every `author` and `editor` element has a single child of type `String` (Text); hence they are leaf elements.

We can declare a user-defined generic data type `XMLTree` (short: XML) in Haskell that can represent any XML element (not just books) as follows: An XML element (= XML tree) is either an XML leaf (i.e., a `String`, e.g., “Foundations of Databases”), or an XML non-leaf element.

A non-leaf element has several things:

- a tag (which is a just a `String`) defining the “XML type” of the non-leaf element (e.g., “book” or “editor”)

- a set of (attribute="value") pairs (e.g., the first book element has a `year` attribute whose value is the string “1995”).

Note that the order of attributes is not important and that each attribute can occur only once within a start tag

- a list of children which are of type XML element (hence XML is a recursive type). For example, the list of children elements for the first `book` element above has one `title` and three `author` children elements.

The order of children elements is relevant (e.g., we can speak of the first, second, and third author of a book).

a) Define in Haskell a data type `XML` that corresponds to the XML trees explained above. Define the above sample XML document as a value `booksTree` of type `XML`.

b) Define a function `tags :: XML -> [Tag]` that returns the tags of a given XML tree in preorder. For example, given the XML tree `booksTree` above, we get

```
tags booksTree =>
  ["books","book","title","author","author","author","book","title","editor","editor","editor"]
```