

LiU-FP2010 Part II, Linkping, 17–19 May 2010
Exercises, Lecture 5: Type Classes
Henrik Nilsson

For the examination, do all of the following:

1. Define a type-class `Size` for an overloaded operation

```
size :: Size a => a -> Integer
```

that estimates the number of nodes of a data structure. Define instances for the following types:

<code>()</code>	The unit type
<code>Bool</code>	Booleans
<code>Char</code>	Characters
<code>Int</code>	Fixed-precision integers
<code>Double</code>	Double-precision floating point numbers
<code>(t₁, ..., t_n)</code>	Tuples for $n \in [2, 4]$
<code>[t]</code>	Lists of elements of type t
<code>Maybe t</code>	The option type

For example, we take the size of values of primitive types like `()`, `Bool`, and `Int` to be 1, the size of a tuple to be 1 + the sum of the sizes of the fields, the size of a list to be the sum of the sizes of the contained elements plus the number of `(:)`-cells and an additional one for the empty list, and so on.

2. Interval arithmetic, as the name suggests, is arithmetic defined on numerical intervals. For example, it can be used to compute error bounds. Say we know $x \in [l_x, u_x]$, and $y \in [l_y, u_y]$, then

$$x + y \in [l_x + l_y, u_x + u_y] \quad \text{and} \quad x - y \in [l_x - u_y, u_x - l_y]$$

Let us represent an interval as follows:

```
data Iv1 = Iv1 Double Double deriving (Show, Eq)
```

Note: the `Eq` instance is perhaps not very useful, but necessary in order to make `Iv1` a `Num` instance.

Make `Iv1` an instance of the type classes `Num` and `Fractional` (Methods `(+)`, `(-)`, `(*)`, `abs`, `signum`, `fromInteger` for `Eq`; methods `(/)`, `recip`, `(fromRational)` for `Fractional`). You should enforce the invariant that for any value `Iv1 l u`, $l \leq u$. Use `error` to give suitable error messages when partial operations are undefined.

The above instances will make it possible to use overloaded numerical literals to construct intervals containing only that specific number. E.g. `1` denotes `Iv1 1.0 1.0` when used at type `Iv1`. Define an operator

```
(+/-) :: Double -> Double -> Iv1
```

for constructing symmetric intervals around a specific number. E.g. `1 +/- 0.5` denotes `Iv1 0.5 1.5`.

As an alternative to making certain operations partial, would it be feasible and useful to consider and compute with an extended set of intervals? Say adding specific value constructors to the type `Iv1` for representing the intervals $[0, \infty)$, $(-\infty, 0]$, $(-\infty, \infty)$?