FUNCTIONAL PROGRAMMING MT2020

Sheet 5

9.1 Suppose a type of natural numbers is defined by

```
> data Nat = Zero | Succ Nat
```

Use recursion to define functions $\mathit{int} :: \mathit{Nat} \to \mathit{Int}$ and $\mathit{nat} :: \mathit{Int} \to \mathit{Nat}$ which embed the natural numbers in Int in the obvious way.

Use recursion (on the second argument) to define functions

$$\mathit{add}, \mathit{mul}, \mathit{pow}, \mathit{tet} :: \mathit{Nat} \to \mathit{Nat} \to \mathit{Nat}$$

which implement addition, multiplication, exponentiation, and what Goodstein calls tetration.

($x\ \text{`}\mathit{tet}\text{`}\ n = x \mathbin{\hat{}} x \mathbin{\hat{}} \cdots \mathbin{\hat{}} x$ where there are n copies of x.)

9.2 What property characterises $\mathit{foldNat}$, the fold for Nat? Define $\mathit{foldNat}$.

What are the deconstructors for Nat, and what characterises the unfold $\mathit{unfoldNat}$? Define $\mathit{unfoldNat}$.

Express each of int and nat as either $\mathit{foldNat}$ or $\mathit{unfoldNat}$.

Finally express add, mul, pow and tet as folds.

10.1 Prove directly by induction that

$$\mathit{fold}\ c\ n\ (xs +\!\!+ ys) \quad = \quad \mathit{fold}\ c\ (\mathit{fold}\ c\ n\ ys)\ xs$$

for all lists xs and ys (whether partial, finite or infinite).

10.2 Use fold fusion to show that the section $(+\!\!+ bs)$ is a fold.

Deduce without resort to induction that

$$\mathit{fold}\ c\ n\ (xs +\!\!+ ys) \quad = \quad \mathit{fold}\ c\ (\mathit{fold}\ c\ n\ ys)\ xs$$

10.3 Use fold fusion to show that $\mathit{filter}\ p$ is a fold.

Deduce that

$$\mathit{filter}\ p\ (xs +\!\!+ ys) \quad = \quad \mathit{filter}\ p\ xs +\!\!+ \mathit{filter}\ p\ ys$$

10.4 A data type very like that of lists might be defined by

```
> data Liste a = Snoc (Liste a) a | Lin
```

There will be elements of $Liste\ \alpha$ and of $[\alpha]$ corresponding to finite lists, for example $Snoc\ (Snoc\ (Snoc\ Lin\ 1)\ 2)\ 3$ corresponds to $1 : (2 : (3 : [\,]))$, that is $[1, 2, 3]$.

Write a recursive definition of a function $cat :: Liste\ \alpha \to Liste\ \alpha \to Liste\ \alpha$ which concatenates two elements of $Liste$.

Define a function *folde* which is the natural fold for $Liste\ \alpha$.

Express *cat* in terms of *folde*.

Define (as folds) functions $list :: Liste\ \alpha \to [\alpha]$ and $liste :: [\alpha] \to Liste\ \alpha$ which express the identification of finite lists represented as elements of $Liste\ \alpha$ and of $[\alpha]$. (That is, they should be mutually inverse on finite lists.)

What does *liste* return when applied to an infinite list? What are the infinite objects of type $Liste\ \alpha$?

Find equivalent definitions of *list* and *liste* as instances of *loop* and the corresponding function for $Liste\ \alpha$.

10.5 Recall that the unfold function for $[\alpha]$

```
> unfold n h t x | n x       = []
>                | otherwise = h x : unfold n h t (t x)
```

yields the identity function *unfold null head tail* when applied to the deconstructors for $[\alpha]$.

Using the same property for the identity of $Liste\ \alpha$ define the unfold function *unfolde* for $Liste\ \alpha$.

Write *list* and *liste* as the appropriate unfolds.

You may want to know that there are predefined functions $init :: [\alpha] \to [\alpha]$ and $last :: [\alpha] \to \alpha$ for which $xs = init\ xs +\!\!+ [last\ xs]$ for all non-null xs. It might help to work out first how these might be defined by recursion.

Geraint Jones, 2020