

Prof. Dr. Jürgen Giesl
Peter Schneider-Kamp
Stephan Swiderski

Exercises *Functional Programming* – Sheet 8

Solutions will be collected until Wednesday, June 13, 2007 in the exercise course.

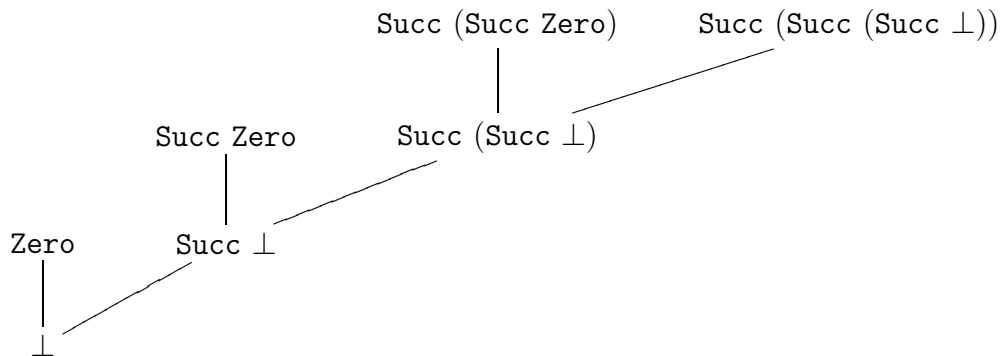
Exercises can be solved both in English and in German.

Exercise 1 (2 + 2 + 2 + 1 points)

Consider the following data type declaration for natural numbers:

```
data Nats = Zero | Succ Nats
```

A graphical representation of the first four levels of the domain for `Nats` could look like this:



Now consider the following data type declarations

- (i) `data BoolPair = P (Bool, Bool)`
- (ii) `data NatsTree = L Nats | N NatsTree NatsTree`

- (a) Give a graphical representation of the whole domain for the type `BoolPair` from (i).
- (b) Give a graphical representation of the first three levels of the domain for the type `NatsTree` from (ii). The third level contains the element `L Zero`, for example.
- (c) How many elements does the fourth level of the domain for the type `NatsTree` from (ii) contain?

- (d) Give Haskell expressions that correspond to the following elements of the domain for the type `NatsTree` from (ii), i.e., for each of these elements, give a Haskell expression that has this element as its semantics:

- `L ⊥`
- `N ⊥ (L Zero)`
- `N ⊥ (N ⊥ (L ⊥))`

Exercise 2 (3 + 3 points)

In this exercise, the aim is to implement a Haskell program which generates the domain for `Nats` up to a given depth. To represent all necessary components of this domain, the following data structure should be used:

```
data DomainForNats = Zero | Succ DomainForNats | Bottom
```

The type `DomainForNats` represents the domain for `Nats` where the constructor `Bottom` represents \perp . For example, the element `Succ ⊥` of the domain for `Nats` is represented as `Succ Bottom` in our Haskell program.

- (a) Write the Haskell function

```
moreDefinedSuccessors :: DomainForNats -> [DomainForNats]
```

which returns a list of all more defined direct successors of a given value. For example:

- `moreDefinedSuccessors Bottom` evaluates to `[Zero, Succ Bottom]`
- `moreDefinedSuccessors Zero` evaluates to `[]`
(there are no more defined successors, since `Zero` is fully defined)
- `moreDefinedSuccessors (Succ Bottom)` evaluates to `[Succ Zero, Succ (Succ Bottom)]`

- (b) Write the Haskell function `generate :: Int -> [[DomainForNats]]` where `generate n` returns a list of the `n` first levels of the domain for `Nats`. For example:

- `generate 0` evaluates to `[]`
- `generate 1` evaluates to `[[Bottom]]`
- `generate 2` evaluates to `[[Bottom], [Zero, Succ Bottom]]`
- `generate 3` evaluates to `[[Bottom], [Zero, Succ Bottom], [Succ Zero, Succ (Succ Bottom)]]`