Lab 7

Functional Programming (ITI0212)

2021.03.09

This week we learned about totality for functions on inductive- and coinductive data types.

In order to complete the tasks below you should download this week's lecture script from the course web page and either `import Lecture7` in your lab file or else copy/paste the relevant definitions.

Task 1
Write the length function for `CoList`s:

```
length  :  CoList a -> CoNat
```

Make sure that Idris recognizes it as total.

Task 2
Write the drop function for `CoList`s, which discards a given number of head elements. If the `CoList` contains fewer than the given number of elements, it should return the empty `CoList`.

```
drop  :  Nat -> CoList a -> CoList a
```

Make sure that Idris recognizes it as total.

Task 3
Write the filter function for `CoList`s, which keeps only those elements that satisfy the predicate.

```
filter  :  (a -> Bool) -> CoList a -> CoList a
```

No function satisfying this specification can be total, why not? Write a term,

```
  filter ?predicate ?sequence
```

that will not yield a result in any finite amount of time.

Task 4
Write the zip function for `Stream`s:

```
zipStream  :  (a -> b -> c) -> Stream a -> Stream b -> Stream c
```

Make sure that Idris recognizes it as total.

Task 5
Write the function that zips an initial segment of a `Stream` with a `List`:

```
zipStreamList  :  (a -> b -> c) -> Stream a -> List b -> List c
```

Make sure that Idris recognizes it as total.

Task 6
Use your `zipStreamList` function and the stream of natural numbers to write the function that enumerates a list as a one-liner:

```
enumerate  :  List a -> List (Pair Nat a)
```

Make sure that Idris recognizes it as total.

Task 7
Write the bounded subtraction function for `CoNat`s:

```
minus  :  CoNat -> CoNat -> CoNat
```

No function satisfying this specification can be total, why not? Write a term,

```
  ?some_conat `minus` ?another_conat
```

that will not yield a result in any finite amount of time.

Task 8
Write the multiplication function for `CoNat`s in such a way that it is total (even if Idris doesn't recognize it as such):

```
times  :  CoNat -> CoNat -> CoNat
```

Hint: start by writing the familiar recursive definition that we use for `Nat`s, then try to restructure your definition in such a way that all recursive calls are constructor-guarded and any intervening functions are themselves total. You should assume that $0 \times n = 0 = n \times 0$ for any `CoNat` n. If your function is total then the expression `infinity `times` infinity` should yield a result in finite time.