Lab 12

Functional Programming

2024-05-03

This week we are learning about first-order logic under the propositions-as-types interpretation. The constructive logic that this interpretation realizes is somewhat different from the Boolean logic with which you're surely familiar. For example, because it is not required that every proposition be decidable, we cannot prove the principle of the *excluded middle*:

```
excluded_middle  :  {a : Type} -> a `Or` Not a
```

Nevertheless, many familiar theorems from Boolean logic remain constructively valid.

Task 1
Convince Idris that the principle of *contraposition* is valid:

```
contrapositive  :  (a -> b) -> Not b -> Not a
```

Task 2
Convince Idris that the principle of *double-negation introduction* is valid:

```
dni  :  a -> Not (Not a)
```

Task 3
The converse of double-negation introduction, known as *double-negation elimination* is *not* constructively provable. However, an important special case, where `a` is itself a negation, *is* provable.

Convince Idris that the principle of *triple-negation reduction* is valid:

```
tnr  :  Not (Not (Not a)) -> Not a
```

Hint: the principle of double-negation introduction will be helpful here.

Task 4
Convince Idris that two nonempty lists with different heads are different:

```
heads_differ  :  Not (x = y) -> Not (x :: xs = y :: ys)
```

Hint: your `contrapositive` function from task 1 may be helpful here.
Note: the overloading of `(::)` seems to confuse Idris here so you should `%hide Prelude.Stream.(::)`.

Task 5
Convince Idris that there exist nonzero natural numbers (recalling the interpretation of `Some`):

```
some_nonzero  :  Some Nat (\ n => Not (n = 0))
```

Task 6
Convince Idris that each natural number is either odd or even (recalling the interpretation of `Or`):

```
odd_or_even  :  (n : Nat) -> Odd n `Or` Even n
```