Lab 12

Functional Programming (ITI0212)

2020.05.06

This week we learned about empty types and how to express negation in 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. We also learned about decidable propositions and predicates, and in particular, about decidable equality relations. You should import this week's lecture file to complete the tasks below.

Constructive Logic

The constructive logic realized by the propositions-as-types interpretation has fewer theorems than Boolean logic. 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 valid constructively.

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
In general, the converse of double-negation introduction 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 elimination* is valid:

```
tne  :  Not $ Not $ Not a -> Not a
```

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

Task 4
Convince Idris that the following *de-Morgan* principles are valid:

```
dm1  :  Not a `Or` Not b -> Not (a `And` b)
```

```
dm2  :  Not a `And` Not b -> Not (a `Or` b)
```

Deciding Vector Equality

Task 5
Convince Idris that two nonempty `Vect`s are equal just in case their heads are equal and their tails are equal:

```
cons_equal  :  {xs , ys  : Vect n a} ->
  x = y -> xs = ys -> x::xs = y::ys

decons_equal  :  {xs , ys  : Vect n a} ->
  x::xs = y::ys -> (x = y) `And` (xs = ys)
```

Task 6
Convince Idris that two nonempty `Vect`s with different heads or with different tails are not equal:

```
heads_differ  :  {xs , ys  : Vect n a} ->
  Not (x = y) -> Not (x::xs = y::ys)

tails_differ  :  {xs , ys  : Vect n a} ->
  Not (xs = ys) -> Not (x::xs = y::ys)
```

Task 7
Convince Idris that assuming the element type has decidable equality, if two nonempty `Vect`s differ then either their head elements differ or else their tail `Vect`s differ:

```
decons_differ  :  DecEq a => {xs , ys  : Vect n a} ->
  Not (x::xs = y::ys) -> Not (x = y) `Or` Not (xs = ys)
```

Task 8
Use the lemmas that you just proved in order to complete a named implementation of decidable equality for vectors whose element type itself has decidable equality:

```
implementation [custom] DecEq a => DecEq (Vect n a)  where
  decEq xs ys  =  ?Goal_decEq
```