Lab 12: Inductive equality

Functional Programming (ITI0212)

This week we saw how to represent equality as a dependent type. For any two elements of the same type, `x y : α`, we have a type `x = y : Prop`.

There is an element `h : x = y` just when `x` and `y` are equal, either by definition or by manipulating `x` and `y` until they are definitionally equal. Definitional equality is the equality that can be automatically checked by reducing expressions to their normal forms. Functions for manipulating equations are based on the eliminator for equality, which captures the principle of substitution.

Task 1
Equality is sometimes axiomatized as the "least equivalence relation" (on a type).

A relation `R` on a type α is a type constructor `R : α → α → Prop`. `Eq` is certainly one of these.

An *equivalence relation* is a relation which satisfies:

- $\forall$ `a, R a a` (reflexivity)
- $\forall$ `a b, R a b ↔ R b a` (symmetry)
- $\forall$ `a b c, R a b ∧ R b c → R a c` (transitivity)

The definition of equality as an inductive type does not ask for these properties: they are theorems that we can deduce from the definition.

In the lecture we proved the symmetry property.

1. Prove that `=` is reflexive.
2. Prove that `=` is transitive.

Hint: try pattern matching on equality, or using `Eq.subst` (which itself pattern matches on an equality).

Task 2
To say that `=` is the *least* equivalence relation means that for any other equivalence relation `R`, we have `a = b → R a b`.

Prove the statement:

```
least_equiv (R : α → α → Prop) (h : Equivalence R) : a = b → R a b
```

Task 3
Let's prove some equalities of *types*. Prove that:

- `Vector α (1 + 2) = Vector α (2 + 1)`,
- `Vector α (n + m) = Vector α (m + n)`.

Hint: you can make use of `Nat.add_comm`.

If you have an equality of types, you can use `cast` to convert values of one type to values of the other.