# Lab 12

Functional Programming (ITI0212)

2023-04-21

This week we explored the *propositions as types* perspective more in detail, and we saw how we can express logical connectives between propositions and prove simple logical statements. In this week's lab we will construct some simple proofs of logical facts, and give a hint to how the logical system that we can play around in Idris is not exactly the same to the one based on booleans that you're already familiar with.

*Note:* just like last week's lab, do not forget to add `%default total` to the beginning of your script. This will tell Idris that all the functions we are defining should be total, so that we do not accidentally fall into the case in which we can prove any statement by simply giving a function that does not terminate.

## Logical connectives

Throughout the rest of the lab, you can directly use their equivalent versions available in the Idris standard library that you already know:

```
And : Type -> Type -> Type
And = Pair

Or : Type -> Type -> Type
Or = Either

Not : Type -> Type
Not a = a -> Void

Implies : Type -> Type -> Type
Implies a b = a -> b

Exists : (t : Type) -> (p : t -> Type) -> Type
Exists = DPair

Forall : (t : Type) -> (p : t -> Type) -> Type
Forall t p = (x : t) -> p x
```

Alternatively, you can use the definitions of connectives that we saw in this week's lecture by defining a new inductive datatype; pick whichever you feel more comfortable with!

*Note:* there are a lot of exercises in this week's lab! Only 8 of them are supposed to be strongly suggested, and the rest are optional, but every single one of them can be completed with not too much effort. (we still encourage you to try and tackle them!) As a general guideline if you feel stuck, try to case split on the arguments that you were given as much as possible. Very often you will have only possible way of closing the goal, using the arguments you were given: if the goal is an `And`, then it must be a pair, if the goal is an `Implies`, then it must be a function; if the goal is an `Exists`, then it must be a dependent pair, and if the goal is a `Forall`, then it must be a dependent function, etc.

## The "and" and "or" connectives

**Task 1** (Commutativity of `And`)
Prove that the logical "and" is commutative: if `p` and `q` hold, then `q` and `p` holds. Prove this by giving a definition for:

```
and_comm : And p q -> And q p
```

(Does this function look familiar? And if you have already encountered it before, what was its name?)

**Task 2** (Associativity of `And`)
Prove that the logical "and" is associative:

```
and_assoc : (p `And` q) `And` r  ->  p `And` (q `And` r)
```

Can you try to prove the same statement, but switching input and output types?

**Task 3** (**Optional:** Properties of `Or`)
Prove that the logical "or" is also commutative and associative.

**Task 4** (Distributivity of `And` and `Or`)
Prove that the logical "and" and logical "or" distribute over each other:

```
and_distr : p `And` (q `Or` r)  ->  (p `And` q) `Or` (p `And` r)
```

To see how this resembles a distributivity law, think about the "and" operator as being multiplication between numbers and the "or" operator as being addition of numbers. Then, you can think of this as being the same as collecting a common factor out of the sum of two expressions: $a(b + c) = ab + ac$.

(Indeed, this is not by chance! The correspondence between types and different parts of mathematics that you're already familiary with goes even deeper. If you're curious, try to convince yourself that `Fin 3 `Or` Fin 4` is equivalent to `Fin 7` by checking the number of possible terms with those types.)

## Playing with implication

A proposition `a` is said to "imply" another proposition `b` when knowing that `a` holds allows you to also know that `b` holds.

**Task 5** (Modus ponens)
Prove the following fact about implication, commonly known as "modus ponens". For example, take `a` to be the proposition "it is raining", and `b` to be the proposition "I am taking my umbrella". If I take my umbrella whenever it rains (`a `Implies` b`), and I know it's now raining (`a`), then I am now taking my umbrella (`b`).

```
modus_ponens : (a `Implies` b) `And` a  ->  b
```

**Task 6** (Implication from a conjunction)
Prove the following fact about implication:

```
implies_implies  :  (a `And` b) `Implies` c
                 ->  a `Implies` (b `Implies` c)
```

(Interpret the type of the function with `And` as `Pair` and with `Implies` as functions; does this function look familiar? And if so, what was it called?)

**Task 7** (**Optional:** Implication into an `And`)
Prove the following fact about implications:

```
implies_and  :  a `Implies` (b `And` c)
             ->  (a `Implies` b) `And` (a `Implies` c)
```

**Task 8** (**Optional:** Implication from an `Or`)
Prove the following fact about implications:

```
or_implies  :  (a `Implies` c) `And` (b `Implies` c)
            ->  (a `Or` b) `Implies` c
```

(There is an equivalent function in the Idris standard library, but it is considerably harder to find than the rest of the functions in this lab!)

## Playing with negation

**Task 9** (Contraposition)
Contraposition is a logical principle regarding implication. Let's make a concrete example: let's say that whenever it rains, I take the umbrella. By contraposition, if you see me walking around without an umbrella, then it must mean that it is not raining. This is precisely what it means to take the contrapositive of an implication (the implication is the fact that "raining implies I take the umbrella").

```
contraposition : a `Implies` b  ->  Not b `Implies` Not a
```

**Task 10** (**Optional:** Law of excluded middle...?)
Convince yourself that it is not possible to write a function with this type in Idris:

```
law_of_excluded_middle : p `Or` (Not p)
```

The fact that this is not provable is due to the way we interpret things as being "true" in Idris and its underlying logic: to say that `a` holds means that we have a concrete term of that type. So, if we managed to prove the `law_of_excluded_middle` it would mean that, for any mathematical claim, we have a program that tells us whether it's true or false, and even gives a proof of it! Asking that such a program exists and always terminates is in general impossible due to the halting problem.

*Note*: it is not possible to prove it for *every* proposition; there might be some propositions for which this is provable. (Can you spot an example that we've seen in this week's lecture?)

**Task 11** (Double negation introduction)
Prove the following statement: if you know that something is true, then it is impossible that it is false.

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

Interestingly, the converse of double negation introduction is not provable in Idris, just like the law of the excluded middle of the previous exercise (and in fact, they are equivalent! From one, you can get the other.)

**Task 12** (**Optional:** Triple negation elimination)
Double negation elimination is not provable in Idris. But surprisingly, *triple* negation elimination is! Prove it:

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

(Hint: you might want to use `double_negation_intro` to prove this, but it is also possible to prove this without any auxiliary function!)

## Playing with quantifiers

**Task 13** (Existence with a conjunction)
Prove the following statement about the existential quantifier: if there exists something with type `t` such that two properties `p` and `q` hold, then you can say that there exists one such that `p` holds and there also exists one such that `q` holds.

```
exist_and : Exists t (\x => p x `And` q x)
  `Implies` (Exists t p `And` Exists t q)
```

(Does the opposite direction of this statement hold? Can you come up with a counterexample from real life?)

**Task 14** (**Optional:** `Exists` and negation of `Forall`)
Prove the following statement: if there exists something with type `t` such that some property `p` holds on it, then it is impossible that the negation of `p` holds for every elements of type `t`.

```
not_exists : Exists t p -> Not (Forall t (\x => Not (p x)))
```

**Task 15** (**Optional:** Switching `Exists` and `Forall`)
Now, for something more complex. Prove the following statement:

```
exists_forall_then_forall_exists : {p : a -> b -> Type}
         -> Exists a (\x => Forall b (\y => p x y))
  `Implies` Forall b (\y => Exists a (\x => p x y))
```

A concrete example: if there exists a brain for every person, then it is true that for each person there exists a brain. (Does the opposite direction of this statement hold? That is, if every person has a brain, then that brain is the same for each person?)