Law of Sub-Type Extension

by Sven Nilsen, 2017

Assume that there are 3 functions of type `T → bool`:

a : T → bool
b : T → bool
c : T → bool

When `a` and `b` are not equivalent:

a ¬= b

With other words, there exists a value such that `a` and `b` returns different booleans:

x : [a] y ∧ [b] ¬y

x : T
y : bool

Then, the following two laws are equivalent (either are both true or both false):

a <=> b ∧ ¬c
a ∨ c <=> b

If both are true, then the sub-type `x` that separates `a` and `b` from each other is defined by `c`:

x : [c] true

The proof was checked with the `pocket_prover` library by evaluating the tautology for all values:

(a ¬= b) → ((a = (b ∧ ¬c)) = ((a ∧ c) = b))

Source code:

```
println!("Result: {}", prove3(|a, b, c| {
    imply(
        // Premise.
        not(eq(a, b)),

        // Conclusion.
        eq(
            eq(a, and(b, not(c))),
            eq(or(a, c), b)
        )
    )
}));
```