Equational reasoning

Changing a function call

Question

Substitution rule `((x) -> M)(a) -> [a/x]M`

Use this, together with the basic rules of algebra, to show the following equivalence:

```
((x, y) -> x + y + 1)(a + 1, b)
                ===
((x, y) -> x + y + 1)(a, b + 1)
```

Answer

```
((x, y) -> x + y + 1)(a + 1, b)
 ===
[a+1/x,b/y](x + y + 1)
 ===
((a+1)+b+1)
 ===
(a+(b+1)+1)
 ===
[a/x,b+1/y](x+y+1)
 ===
((x, y) -> x + y + 1)(a, b+1)
```

Swapping an if-statement

Question

Use these rules for booleans:

```
if (true)  x else y -> x
if (false) x else y -> y
!true -> false
!false -> true
```

Prove the following equivalence. You'll need to quantify over x, showing that this equivalence is valid if you substitute in all possible values for x (i.e.: true and false).

`if (x) a else b === if (!x) b else a`

Answer

```
forall x. if (x) a else b === if (!x) b else a
```

Must prove it for `x=true` and `x=false` :

```
x = true =>

if (x) a else b
===
if (true) a else b
===
a
===
if (false) b else a
===
if (!true) b else a
===
if (!x) b else a
```

```
x = false =>

if (x) a else b
===
if (false) a else b
===
b
===
if (true) b else a
===
if (!false) b else a
===
if (!x) b else a
```

Un-nesting an if statement

Question

We now add the basic evaluation rules for && and ||

```
true  && true  -> true
true  && false -> false
false && true  -> false
false && false -> false

true  || true  -> true
true  || false -> true
false || true  -> true
false || false -> false
```

Note that these define && and || as pure mathematical operators, without short circuiting. Use them along with previous rules to show the following equivalences:

```
if (x) { if (y) a else b  } else b
              ===
if (x && y) a else b
```

```
if (x) a else { if (y) a else b }
              ===
if (x || y) a else b
```

Answer

```
Lemma: forall y, false && y === false
Proof:
   If y=true: false && y === false && true === false
   If y=false: false && y === false && false === false

Lemma: forall y, true && y === y
Proof:
  If y=true: true && y === true && true === true === y
  If y=false: true && y === true && false === false === y

If x = false:
  if (x) { if (y) a else b  } else b
     ===
  if (false) { if (y) a else b} else b
     ===
  b
     ===
  if (false) a else b
     ===
  if (false && y) a else b
     ===
  if (x && y) a else b


If x = true:
  if (x) { if (y) a else b  } else b
    ===
  if (true) { if (y) a else b } else b
    ===
  if (y) a else b
    ===
  if (true && y) a else b
    ===
  if (x && y) a else b
```

```
Lemma: forall y, true || y === true
Proof:
  If y=true: true || y === true || true === true
  If y=false: true || y === true || false === true

Lemma: Forall y, false || y === y
Proof:
  If y=true: false || y === false || true === true === y
  If y=false: false ||y === false || false === false === y

If x=true:

  if (x) a else { if (y) a else b }
     ===
  if (true) a else { if (y) a else b }
     ===
   a
     ===
   if (true) a else b
     ===
   if (true || y) a else b
     ===
   if (x || y) a else b

If x=false:

  if (x) a else { if (y) a else b }
    ===
  if (false) a else { if (y) a else b }
    ===
  if (y) a else b
    ===
  if (false || y) a else b
    ===
  if (x || y) a else b

An alternative proof of the second one, using De Morgan's law and the
 ↪  previous identities:

if (x) a else { if (y) a else b }
  -> if (!x) { if (y) a else b } else a
  -> if (!x) { if (!y) b else a } else a
  -> if (!x && !y) b else a
  === if (x || y) a else b // De Morgan's
```

Conditional-to-function

Question

Rule:

```
x = A; M -> [A/x]M
```

Show that

```
if (A) o.foo() else o.bar()
              ===
f = if (A) (() -> o.foo()) else (() -> o.bar());
f()
```

Answer

```

Lemma: forall x, (if (x) f else g)() === if (x) f() else g()
Proof:
  If x = true:
    (if (x) f else g)()
    ===
    (if (true) f else g)()
    ===
    f()
    ===
    if (true) f() else g()
    ===
    if (x) f() else g()
  If x = false:
    (if (x) f else g)()
    ===
    (if (false) f else g)()
    ===
    g()
    ===
    if (false) f() else g()
    ===
    if (x) f() else g()

if (A) o.foo() else o.bar()
  ===
(if (A) o.foo() else (() -> o.bar())())
  ===
(if (A) (() -> o.foo())() else (() -> o.bar())())
  ===
(if (A) (() -> o.foo()) else (() -> o.bar()))()
  ===
```

```
f = if (A) (() -> o.foo()) else (() -> o.bar());
f()
```

Functoriality of map:

Question

Here is the list datatype in Functional Java

```
public datatype IntList =  Cons Int IntList |  Nil;
```

Here is the induction principle for lists. In plain English, it states: If a property P is true for the empty list, and, from the assumption that P is true for a list l, then it is true for the list "Cons n l" for any n, then P is true for all lists.

```
  ∀P, P(Nil) ⇒
   (∀ (n : Int), ∀ (l : IntList), P(l) ⇒ P(Cons n l))
       ⇒ ∀ (l : IntList), P(l)
```

Prove the following property:

```
map(f, map(g, l)) = map((x) -> f(g(x)), l)
```

Answer

Start with induction on l.

Base case: `l` is the empty list `Nil`

```
map(f, map(g, Nil)) = map((x) -> f(g(x)), Nil)
```

Reducing the expression using the map function definition we have:

```
Nil = Nil
```

Inductive step:

Our induction hyphotesis is:

```
map(f, map(g, l)) = map((x) -> f(g(x)), l)
```

We must prove:

```
map(f, map(g, Cons i l)) = map((x) -> f(g(x)), Cons i l)
```

Where `i` is an `Int` .

Reducing the expression using the map function definition:

```
map(f, map(g, Cons i l)) = map((x) -> f(g(x)), Cons i l)
map(f, (Cons (g(i)) map(g, l))) = (Cons ((x) -> f(g(x)))(i) (map((x) -> f(g(x)), l)))
(Cons (f(g(i))) (map(f, map(g, l)))) = (Cons ((x) -> f(g(x)))(i) (map((x) -> f(g(x)), l)))
```

Applying the induction hyphotesis we have

```
(Cons (f(g(i))) (map(f, map(g, l)))) = (Cons (f(g(i))) (map(f, map(g, l))))
```

Which follows from reflexivity.