Generalizing your Induction Hypothesis

Speaker: Andrew Appel

COS 326

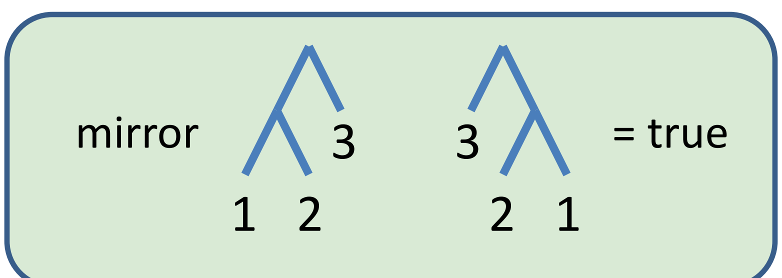
Princeton University

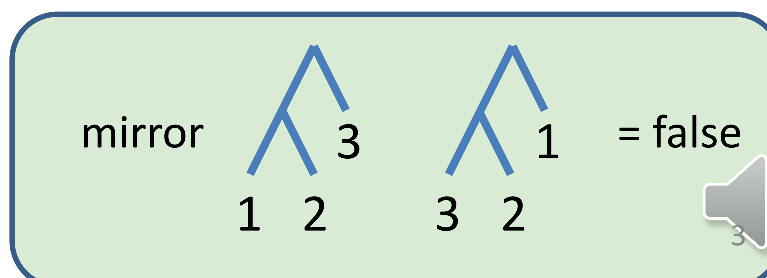
A PROOF ABOUT TWO TREES

Image credit: pxfuel.com, licensed for free use

Reflection tester

type tree = Leaf of int | Node of tree * tree

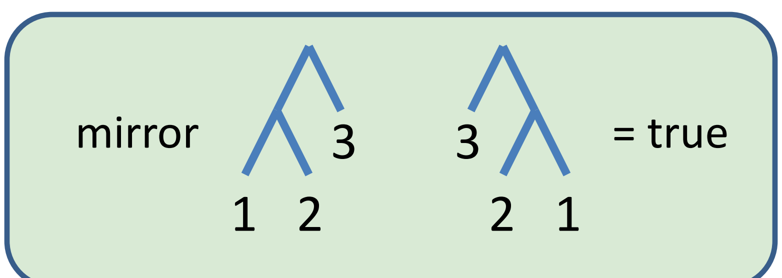


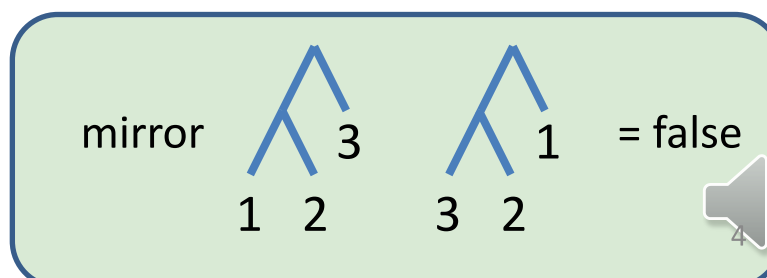


Reflection tester

```
type tree = Leaf of int | Node of tree * tree


let rec mirror (t1: tree) (t2: tree) : bool =
  match t1 with
  | Leaf i -> (match t2 with
              | Leaf j -> i=j
              | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                   | Leaf _ -> false
                   | Node (b',a') -> mirror b b' && mirror a a')
```



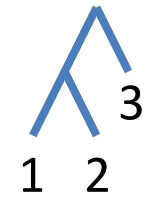


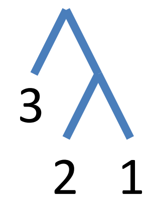
Examples

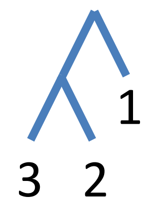
```
let foo = Node(Node(Leaf 1, Leaf 2), Leaf 3)

let bar = Node(Leaf 3, Node(Leaf 2, Leaf 1))

let baz = Node(Node(Leaf 3, Leaf 2), Leaf 1)
```



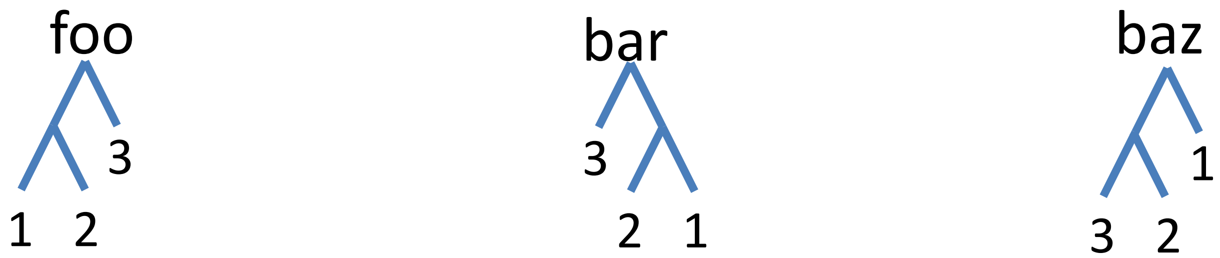




mirror foo bar = true

mirror foo baz = false

Claim!



Theorem: ∀ t:tree. mirror t bar = mirror bar t

Examples:

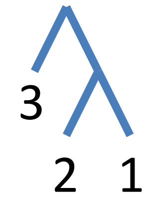
mirror foo bar = true = mirror bar foo

mirror foo baz = false = mirror baz foo

Proof attempt 1

type tree = Leaf of int | Node of tree * tree

let bar = Node(Leaf 3, Node(Leaf 2, Leaf 1))



Theorem: ∀ t:tree. mirror t bar = mirror bar t

Proof:

By induction on t.

Case: t = Leaf i

mirror t bar

==

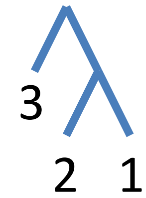
⋮ *(we hope)*

== mirror bar t

Proof attempt 1

type tree = Leaf of int | Node of tree * tree

let bar = Node(Leaf 3, Node(Leaf 2, Leaf 1))



Theorem: ∀ t:tree. mirror t bar = mirror bar t

Proof:
By induction on t.
Case: t = Leaf i

mirror t bar
== mirror (Leaf i) bar
== match bar with Leaf j -> i=j | Node(_,_) -> false
== match Node(Leaf 3, Node(Leaf 2, Leaf 1)) with Leaf j -> i=j | Node(_,_) -> false
== false

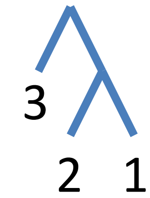
⋮ *(we hope)*

== mirror bar t

Proof attempt 1

type tree = Leaf of int | Node of tree * tree

let bar = Node(Leaf 3, Node(Leaf 2, Leaf 1))



Theorem: ∀ t:tree. mirror t bar = mirror bar t

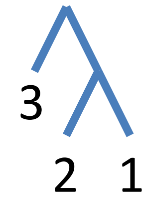
Proof:
By induction on t.
Case: t = Leaf i

```
mirror t bar
== mirror (Leaf i) bar
== match bar with Leaf j -> i=j | Node(_,_) -> false
== match Node(Leaf 3, Node(Leaf 2, Leaf 1)) with Leaf j -> i=j | Node(_,_) -> false
== false
   •
   •  (we hope)
   •
== mirror (Node(Leaf 3,Node(Leaf 2, Leaf 1))) (Leaf i)
== mirror bar t
```

Proof attempt 1

type tree = Leaf of int | Node of tree * tree

let bar = Node(Leaf 3, Node(Leaf 2, Leaf 1))



Theorem: ∀ t:tree. mirror t bar = mirror bar t

Proof:

By induction on t.

Case: t = Leaf i

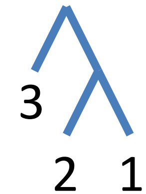
mirror t bar
== mirror (Leaf i) bar
== match bar with Leaf j -> i=j | Node(_,_) -> false
== match Node(Leaf 3, Node(Leaf 2, Leaf 1)) with Leaf j -> i=j | Node(_,_) -> false
== false
== false
== mirror (Node(Leaf 3,Node(Leaf 2, Leaf 1))) (Leaf i)
== mirror bar t

Done with this case!

Proof attempt 1

type tree = Leaf of int | Node of tree * tree

let bar = Node(Leaf 3, Node(Leaf 2, Leaf 1))



Theorem: ∀ t:tree. mirror t bar = mirror bar t

Case: t = Node(a,b)
mirror t bar
== mirror (Node (a,b)) bar

Where a and b satisfy I.H.,
mirror a bar = mirror bar a
mirror b bar = mirror bar b

⋮ *(we hope)*

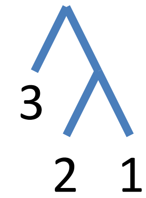
== mirror bar t

```
let rec mirror (t1: tree) (t2: tree) : bool =
  match t1 with
  | Leaf i -> (match t2 with
                 | Leaf j -> i=j
                 | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                    | Leaf _ -> false
                    | Node (b',a') ->
                        mirror b b' && mirror a a')
```

Proof attempt 1

type tree = Leaf of int | Node of tree * tree

let bar = Node(Leaf 3, Node(Leaf 2, Leaf 1))



Theorem: ∀ t:tree. mirror t bar = mirror bar t

Case: t = Node(a,b)

mirror t bar

== mirror (Node (a,b)) bar

== match bar with Leaf _ -> false | Node(b’,a’) -> mirror b b’ && mirror a a’

⋮ *(we hope)*

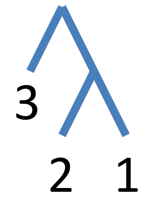
== mirror bar t

```
let rec mirror (t1: tree) (t2: tree) : bool =
  match t1 with
  | Leaf i -> (match t2 with
              | Leaf j -> i=j
              | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                  | Leaf _ -> false
                  | Node (b',a') ->
                    mirror b b' && mirror a a')
```

Proof attempt 1

type tree = Leaf of int | Node of tree * tree

let bar = Node(Leaf 3, Node(Leaf 2, Leaf 1))



Theorem: ∀ t:tree. mirror t bar = mirror bar t

Case: t = Node(a,b)

mirror t bar

== mirror (Node (a,b)) bar

== match bar with Leaf _ -> false | Node(b’,a’) -> mirror b b’ && mirror a a’

== mirror b (Leaf 3) && mirror a (Node(Leaf 2, Leaf 1))

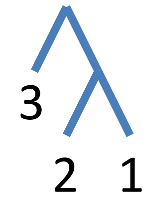
⋮ *(we hope)*

== mirror bar t

```
let rec mirror (t1: tree) (t2: tree) : bool =
  match t1 with
  | Leaf i -> (match t2 with
               | Leaf j -> i=j
               | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                  | Leaf _ -> false
                  | Node (b',a') ->
                    mirror b b' && mirror a a')
```

Proof attempt 1

type tree = Leaf of int | Node of tree * tree
let bar = Node(Leaf 3, Node(Leaf 2, Leaf 1))



Theorem: ∀ t:tree. mirror t bar = mirror bar t

Case: t = Node(a,b)
mirror t bar
== mirror (Node (a,b)) bar
== match bar with Leaf _ -> false | Node(b’,a’) -> mirror b b’ && mirror a a’
== mirror b (Leaf 3) && mirror a (Node(Leaf 2, Leaf 1))

⋮ *(we hope)*

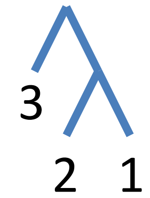
== mirror (Node(Leaf 2, Leaf 1)) a && mirror (Leaf 3) b
== mirror (Node(Leaf 3, Node(_,_))) (Node(a,b))
== mirror bar t

```
let rec mirror t1 t2 =
  match t1 with
  | Leaf i -> (match t2 with
                | Leaf j -> i=j
                | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                   | Leaf _ -> false
                   | Node (b',a') ->
                      mirror b b' &&
                      mirror a a')
```

Proof attempt 1

type tree = Leaf of int | Node of tree * tree

let bar = Node(Leaf 3, Node(Leaf 2, Leaf 1))



Theorem: ∀ t:tree. mirror t bar = mirror bar t

Case: t = Node(a,b)

mirror t bar

== mirror (Node (a,b)) bar

== match bar with Leaf _ -> false | Node(b’,a’) -> mirror b b’ && mirror a a’

== mirror b (Leaf 3) && mirror a (Node(Leaf 2, Leaf 1))

== mirror a (Node(Leaf 2, Leaf 1)) && mirror b (Leaf 3)

⋮ *(we hope)*

== mirror (Node(Leaf 2, Leaf 1)) a && mirror (Leaf 3) b

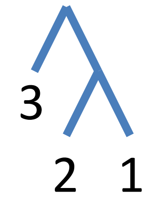
== mirror (Node(Leaf 3, Node(_,_))) (Node(a,b))

== mirror bar t

Proof attempt 1

type tree = Leaf of int | Node of tree * tree

let bar = Node(Leaf 3, Node(Leaf 2, Leaf 1))



Theorem: ∀ t:tree. mirror t bar = mirror bar t

Case: t = Node(a,b)

mirror t bar

== mirror (Node (a,b)) bar

== match bar with Leaf _ -> false | Node(b’,a’) -> mirror b b’ && mirror a a’

== mirror b (Leaf 3) && mirror a (Node(Leaf 2, Leaf 1))

== mirror a (Node(Leaf 2, Leaf 1)) && mirror b (Leaf 3)

== mirror (Node(Leaf 2, Leaf 1)) a && mirror (Leaf 3) b

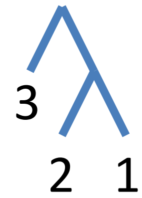
== mirror (Node(Leaf 3, Node(_,_))) (Node(a,b))

== mirror bar t

FAIL!

type tree = Leaf of int | Node of tree * tree

let bar = Node(Leaf 3, Node(Leaf 2, Leaf 1))



Theorem: ∀ t:tree, mirror t bar = mirror bar t

Case: t = Node(a,b)

mirror t bar

== mirror (Node (a,b)) bar

== match bar with Leaf _ -> false | Node(b',a') -> mirror b b' && mirror a a'

== mirror b (Leaf 3) && mirror a (Node(Leaf 2, Leaf 1))

== mirror a (Node(Leaf 2, Leaf 1)) && mirror b (Leaf 3)

Induction hyp tells us:
mirror a bar = mirror bar a
mirror b bar = mirror bar b

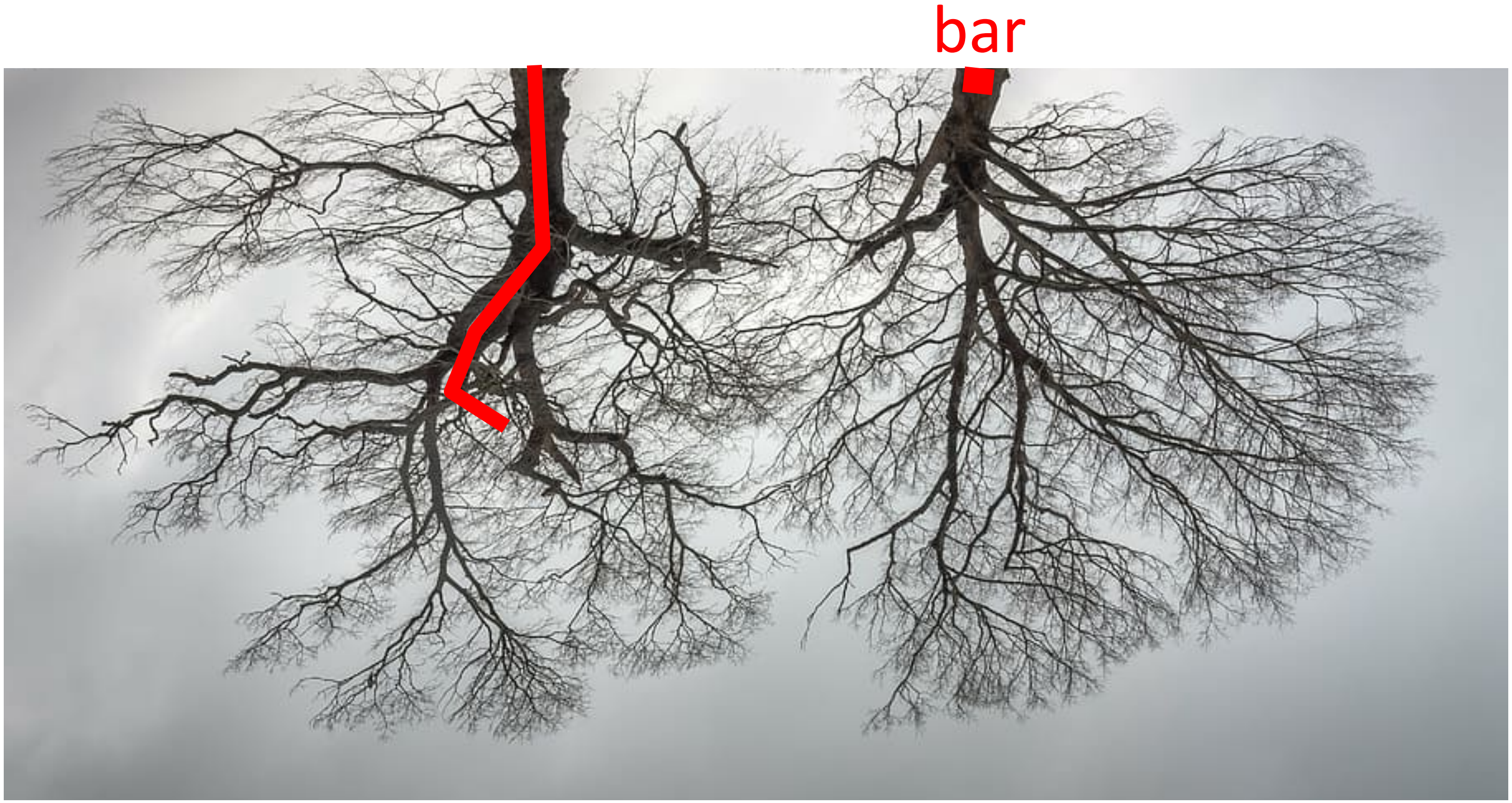
== mirror (Node(Leaf 2, Leaf 1)) a && mirror (Leaf 3) b

== mirror (Node(Leaf 3, Node(_,_))) (Node(a,b))

== mirror bar t

What’s the problem?

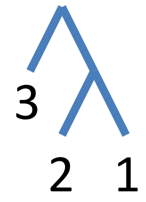
What’s the problem?



Solution: prove a more general theorem!

type tree = Leaf of int | Node of tree * tree

let bar = Node(Leaf 3, Node(Leaf 2, Leaf 1))



Theorem: ∀ t:tree, mirror t bar = mirror bar t

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof!

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Leaf i
Need to prove: ∀ u:tree. mirror t u = mirror u t

Proof!

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Leaf i
  Need to prove: ∀ u:tree. mirror t u = mirror u t
  Assume an arbitrary u about which we know nothing (except its type, “tree”)

Proof!

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Leaf i
  Need to prove: ∀ u:tree. mirror t u = mirror u t
  Assume u: tree.
  Need to prove: mirror t u = mirror u t

Proof!

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Leaf i
  Need to prove: ∀ u:tree. mirror t u = mirror u t
  Assume u: tree.
    mirror t u
 == mirror (Leaf i) u

 ⋮

 == mirror u t

```
let rec mirror t1 t2 =
  match t1 with
  | Leaf i -> (match t2 with
                 | Leaf j -> i=j
                 | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                    | Leaf _ -> false
                    | Node (b',a') ->
                       mirror b b' &&
                       mirror a a')
```

Proof!

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Leaf i
  Need to prove: ∀ u:tree. mirror t u = mirror u t
  Assume u: tree.
      mirror t u
  == mirror (Leaf i) u
  == match u with Leaf j -> i=j | Node(_,_) -> false

  ⋮

  == mirror u t

```
let rec mirror t1 t2 =
  match t1 with
  | Leaf i -> (match t2 with
                | Leaf j -> i=j
                | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                | Leaf _ -> false
                | Node (b',a') ->
                  mirror b b' &&
                  mirror a a')
```

Now, need case analysis on u

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Leaf i
  Need to prove: ∀ u:tree. mirror t u = mirror u t
  Assume u: tree.
    mirror t u
 == mirror (Leaf i) u
 == match u with Leaf j -> i=j | Node(_,_) -> false

 ⋮

 == mirror u t

```
let rec mirror t1 t2 =
  match t1 with
  | Leaf i -> (match t2 with
               | Leaf j -> i=j
               | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                  | Leaf _ -> false
                  | Node (b',a') ->
                    mirror b b' &&
                    mirror a a')
```

Case analysis on u: first subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Leaf i
  Need to prove: ∀ u:tree. mirror t u = mirror u t
  Assume u: tree.
  Subcase: u = Leaf j
      mirror t u
  == mirror (Leaf i) u
  == match u with Leaf j -> i=j | Node(_,_) -> false

  ⋮

  == mirror u t

```
let rec mirror t1 t2 =
  match t1 with
  | Leaf i -> (match t2 with
               | Leaf j -> i=j
               | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                  | Leaf _ -> false
                  | Node (b',a') ->
                     mirror b b' &&
                     mirror a a')
```

Case analysis on u: first subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Leaf i
  Need to prove: ∀ u:tree. mirror t u = mirror u t
  Assume u: tree.
  Subcase: u = Leaf j

```
    mirror t u
 == mirror (Leaf i) u
 == match u with Leaf j -> i=j | Node(_,_) -> false
 == (i=j)
    ⋮
 == mirror u t
```

```
let rec mirror t1 t2 =
  match t1 with
  | Leaf i -> (match t2 with
             | Leaf j -> i=j
             | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                 | Leaf _ -> false
                 | Node (b',a') ->
                   mirror b b' &&
                   mirror a a')
```

Case analysis on u: first subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Leaf i
  Need to prove: ∀ u:tree. mirror t u = mirror u t
  Assume u: tree.
  Subcase: u = Leaf j

```
    mirror t u
 == mirror (Leaf i) u
 == match u with Leaf j -> i=j | Node(_,_) -> false
 == (i=j)
    ⋮
 == mirror (Leaf j) (Leaf i)
 == mirror u t
```

```
let rec mirror t1 t2  =
  match t1 with
  | Leaf i -> (match t2 with
               | Leaf j -> i=j
               | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                  | Leaf _ -> false
                  | Node (b',a') ->
                    mirror b b' &&
                    mirror a a')
```

First subcase done

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Leaf i

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Leaf j

```
    mirror t u
== mirror (Leaf i) u
== match u with Leaf j -> i=j | Node(_,_) -> false
== (i=j)
== (j=i)
== mirror (Leaf j) (Leaf i)
== mirror u t
```

Done with Subcase (u=Leaf j).

```
let rec mirror t1 t2  =
  match t1 with
  | Leaf i -> (match t2 with
               | Leaf j -> i=j
               | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                   | Leaf _ -> false
                   | Node (b',a') ->
                     mirror b b' &&
                     mirror a a')
```

Case analysis on u: **second** subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Leaf i

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Node(g,h)

mirror t u

==

```
let rec mirror t1 t2  =
 match t1 with
 | Leaf i -> (match t2 with
              | Leaf j -> i=j
              | Node(_,_) -> false)
 | Node(a,b) -> (match t2 with
                 | Leaf _ -> false
                 | Node (b',a') ->
                   mirror b b' &&
                   mirror a a')
```

Case analysis on u: second subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Leaf i

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Node(g,h)

mirror t u

== mirror (Leaf i) (Node(g,h))

```
let rec mirror t1 t2  =
  match t1 with
  | Leaf i -> (match t2 with
                 | Leaf j -> i=j
                 | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                    | Leaf _ -> false
                    | Node (b',a') ->
                       mirror b b' &&
                       mirror a a')
```

Case analysis on u: second subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Leaf i

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Node(g,h)

mirror t u

== mirror (Leaf i) (Node(g,h))

== false

•
•
•

== mirror u t

```
let rec mirror t1 t2  =
  match t1 with
  | Leaf i -> (match t2 with
               | Leaf j -> i=j
               | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                  | Leaf _ -> false
                  | Node (b',a') ->
                     mirror b b' &&
                     mirror a a')
```

Case analysis on u: second subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Leaf i

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Node(g,h)

mirror t u

== mirror (Leaf i) (Node(g,h))

== false

-
-
-

== mirror (Node(g,h) (Leaf i)

== mirror u t

```
let rec mirror t1 t2  =
  match t1 with
  | Leaf i -> (match t2 with
               | Leaf j -> i=j
               | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                  | Leaf _ -> false
                  | Node (b',a') ->
                    mirror b b' &&
                    mirror a a')
```

Case analysis on u: second subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Leaf i

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Node(g,h)

mirror t u

== mirror (Leaf i) (Node(g,h))

== false

== false

== mirror (Node(g,h) (Leaf i)

== mirror u t

```
let rec mirror t1 t2 =
  match t1 with
  | Leaf i -> (match t2 with
              | Leaf j -> i=j
              | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                  | Leaf _ -> false
                  | Node (b',a') ->
                     mirror b b' &&
                     mirror a a')
```

Case analysis on u: second subcase done.

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Leaf i
  Need to prove: ∀ u:tree. mirror t u = mirror u t
  Assume u: tree.
  Subcase: u = Node(g,h)
    mirror t u
 == mirror (Leaf i) (Node(g,h))
 == false
 == mirror (Node(g,h) (Leaf i)
 == mirror u t
Done with Subcase (u=Node(g,h)).
Done with Case (t=Leaf i).

```
let rec mirror t1 t2 =
  match t1 with
  | Leaf i -> (match t2 with
                | Leaf j -> i=j
                | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                   | Leaf _ -> false
                   | Node (b',a') ->
                     mirror b b' &&
                     mirror a a')
```

Case analysis on t: second case

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Node(a,b)

Need to prove: ∀ u:tree. mirror t u = mirror u t

```
let rec mirror t1 t2  =
  match t1 with
  | Leaf i -> (match t2 with
                | Leaf j -> i=j
                | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                    | Leaf _ -> false
                    | Node (b',a') ->
                      mirror b b' &&
                      mirror a a')
```

Case analysis on t: second case

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Node(a,b)

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Need to prove: mirror t u = mirror u t

```
let rec mirror t1 t2  =
  match t1 with
  | Leaf i -> (match t2 with
                  | Leaf j -> i=j
                  | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                      | Leaf _ -> false
                      | Node (b',a') ->
                        mirror b b' &&
                        mirror a a')
```

Case analysis on u: first subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Node(a,b)

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Leaf i.

mirror t u

==

```
let rec mirror t1 t2  =
  match t1 with
  | Leaf i -> (match t2 with
                | Leaf j -> i=j
                | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                   | Leaf _ -> false
                   | Node (b',a') ->
                      mirror b b' &&
                      mirror a a')
```

Case analysis on u: first subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Node(a,b)

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Leaf i.

mirror t u

== mirror (Node(a,b)) (Leaf i)

```
let rec mirror t1 t2  =
  match t1 with
  | Leaf i -> (match t2 with
                 | Leaf j -> i=j
                 | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                     | Leaf _ -> false
                     | Node (b',a') ->
                        mirror b b' &&
                        mirror a a')
```

Case analysis on u: first subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Node(a,b)

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Leaf i.

mirror t u

== mirror (Node(a,b)) (Leaf i)

== false

```
let rec mirror t1 t2 =
  match t1 with
  | Leaf i -> (match t2 with
                 | Leaf j -> i=j
                 | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                      | Leaf _ -> false
                      | Node (b',a') ->
                         mirror b b' &&
                         mirror a a')
```

Case analysis on u: first subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Node(a,b)

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Leaf i.

```
    mirror t u
== mirror (Node(a,b)) (Leaf i)
== false
== mirror (Leaf i) (Node(a,b))
```

```
let rec mirror t1 t2 =
  match t1 with
  | Leaf i -> (match t2 with
                 | Leaf j -> i=j
                 | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                    | Leaf _ -> false
                    | Node (b',a') ->
                      mirror b b' &&
                      mirror a a')
```

Case analysis on u: first subcase done.

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Node(a,b)

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Leaf i.

mirror t u

== mirror (Node(a,b)) (Leaf i)

== false

== mirror (Leaf i) (Node(a,b))

== mirror u t

Done with Subcase (u=Leaf i).

```
let rec mirror t1 t2  =
  match t1 with
  | Leaf i -> (match t2 with
                  | Leaf j -> i=j
                  | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                     | Leaf _ -> false
                     | Node (b',a') ->
                        mirror b b' &&
                        mirror a a')
```

Case analysis on u: second subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Node(a,b)

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Node(g,h).

mirror t u

==

```
let rec mirror t1 t2 =
  match t1 with
  | Leaf i -> (match t2 with
               | Leaf j -> i=j
               | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                  | Leaf _ -> false
                  | Node (b',a') ->
                    mirror b b' &&
                    mirror a a')
```

Case analysis on u: second subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Node(a,b)

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Node(g,h).

mirror t u

== mirror (Node(a,b)) (Node(g,h))

```
let rec mirror t1 t2  =
 match t1 with
 | Leaf i -> (match t2 with
                | Leaf j -> i=j
                | Node(_,_) -> false)
 | Node(a,b) -> (match t2 with
                  | Leaf _ -> false
                  | Node (b',a') ->
                     mirror b b' &&
                     mirror a a')
```

Case analysis on u: second subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Node(a,b)
  Need to prove: ∀ u:tree. mirror t u = mirror u t
  Assume u: tree.
  Subcase: u = Node(g,h).
    mirror t u
  == mirror (Node(a,b)) (Node(g,h))
  == mirror b h && mirror a g

```
let rec mirror t1 t2 =
  match t1 with
  | Leaf i -> (match t2 with
                | Leaf j -> i=j
                | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                   | Leaf _ -> false
                   | Node (b',a') ->
                      mirror b b' &&
                      mirror a a')
```

Case analysis on u: second subcase

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Node(a,b)

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Node(g,h).

mirror t u

== mirror (Node(a,b)) (Node(g,h))

== mirror b h && mirror a g

== mirror a b && mirror b h

```
let rec mirror t1 t2  =
  match t1 with
  | Leaf i -> (match t2 with
              | Leaf j -> i=j
              | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                 | Leaf _ -> false
                 | Node (b',a') ->
                    mirror b b' &&
                    mirror a a')
```

What does the induction hypothesis tell us?

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Node(a,b)
  Need to prove: ∀ u:tree. mirror t u = mirror u t
  Assume u: tree.
  Subcase: u = Node(g,h).
    mirror t u
  == mirror (Node(a,b)) (Node(g,h))
  == mirror b h && mirror a g
  == mirror a b && mirror b h

Induction hyp tells us:
∀ u:tree. mirror a u = mirror u a
  and
∀ u:tree. mirror b u = mirror u b

Why? Because a and b are the immediate subtrees of t

What does the induction hypothesis tell us?

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Node(a,b)
Need to prove: ∀ u:tree. mirror t u = mirror u t
Assume u: tree.
Subcase: u = Node(g,h).
mirror t u
== mirror (Node(a,b)) (Node(g,h))
== mirror b h && mirror a g
== mirror a b && mirror b h
== mirror b a && mirror b h

Induction hyp tells us:
∀ u:tree. mirror a u = mirror u a
and
∀ u:tree. mirror b u = mirror u b

What does the induction hypothesis tell us?

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Node(a,b)
  Need to prove: ∀ u:tree. mirror t u = mirror u t
  Assume u: tree.
  Subcase: u = Node(g,h).
    mirror t u
  == mirror (Node(a,b)) (Node(g,h))
  == mirror b h && mirror a g
  == mirror a b && mirror b h
  == mirror b a && mirror b h
  == mirror b a && mirror h b

Induction hyp tells us:
∀ u:tree. mirror a u = mirror u a
  and
∀ u:tree. mirror b u = mirror u b

Finishing the proof

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Node(a,b)

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Node(g,h).

```
   mirror t u
== mirror (Node(a,b)) (Node(g,h))
== mirror b h && mirror a g
== mirror a g && mirror b h
== mirror g a && mirror b h
== mirror g a && mirror h b
== mirror (Node(g,h)) (Node(a,b))
```

```
let rec mirror t1 t2  =
  match t1 with
  | Leaf i -> (match t2 with
               | Leaf j -> i=j
               | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                  | Leaf _ -> false
                  | Node (b',a') ->
                    mirror b b' &&
                    mirror a a')
```

Finishing the proof.

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Node(a,b)

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Node(g,h).

mirror t u

== mirror (Node(a,b)) (Node(g,h))

== mirror b h && mirror a g

== mirror a g && mirror b h

== mirror g a && mirror b h

== mirror g a && mirror h b

== mirror (Node(g,h)) (Node(a,b))

== mirror u t

Done with Subcase (u=Node(g,h)),

Done with Case (t=Node(a,b)

```
let rec mirror t1 t2  =
  match t1 with
  | Leaf i -> (match t2 with
               | Leaf j -> i=j
               | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                    | Leaf _ -> false
                    | Node (b',a') ->
                      mirror b b' &&
                      mirror a a')
```

Finishing the proof.

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:
By induction on t.
Case: t = Node(a,b)
  Need to prove: ∀ u:tree. mirror t u = mirror u t
  Assume u: tree.
  Subcase: u = Node(g,h).
      mirror t u
  == mirror (Node(a,b)) (Node(g,h))
  == mirror b h && mirror a g
  == mirror a g && mirror b h
  == mirror g a && mirror b h
  == mirror g a && mirror h b
  == mirror (Node(g,h)) (Node(a,b))
  == mirror u t
Done with Subcase (u=Node(g,h)),
Done with Case (t=Node(a,b))
QED

```
let rec mirror t1 t2 =
  match t1 with
  | Leaf i -> (match t2 with
               | Leaf j -> i=j
               | Node(_,_) -> false)
  | Node(a,b) -> (match t2 with
                  | Leaf _ -> false
                  | Node (b',a') ->
                     mirror b b' &&
                     mirror a a')
```

Summary of the proof

Theorem: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof:

By induction on t.

Case: t = Leaf i

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Leaf j

mirror t u == . . . == mirror u t

Subcase: u = Node(g,h)

mirror t u == . . . == mirror u t

Case: t = Node(a,b)

Need to prove: ∀ u:tree. mirror t u = mirror u t

Assume u: tree.

Subcase: u = Leaf j

mirror t u == . . . == mirror u t

Subcase: u = Node(g,h)

mirror t u == . . . == mirror u t

QED

Our original proof goal

Theorem 1: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof . . . QED

Theorem 2: ∀ t:tree. mirror t bar = mirror bar t

Proof.

Assume t:tree.

Must prove: mirror t bar = mirror bar t.

Our original proof goal

Theorem 1: ∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Proof . . . QED

Theorem 2: ∀ t:tree. mirror t bar = mirror bar t

Proof.
Assume t:tree.
  Must prove: mirror t bar = mirror bar t.
  Apply Theorem 1, instantiating variable t with t, instantiating u with bar.
QED.

Moral of the story:

WHEN PROVING BY INDUCTION, SOMETIMES YOU MUST GENERALIZE THE THEOREM

(OR ELSE THE INDUCTION HYPOTHESIS WON'T FIT)

Another example

```
let rec same (i: int) (j: int) : bool =
  if i=0 then j=0
  else j>0 && same (i-1) (j-1)
```

Claim: ∀ x:nat. same x 3 = same 3 x

Remark: x:nat means that x≥0

Examples:

same 3 3 = true = same 3 3

same 4 3 = false = same 3 4

Now prove this!

```
let rec same (i: int) (j: int) : bool = if i=0 then j=0 else j>0 && same (i-1) (j-1)
```

Theorem: ∀ x:nat.   same x 3 = same 3 x

Now prove this!

```
let rec same (i: int) (j: int) : bool = if i=0 then j=0 else j>0 && same (i-1) (j-1)
```

Theorem: ∀ x:nat.   same x 3 = same 3 x

By induction on x.

Case: x=0

same x 3

==

⋮

== same 3 x

Now prove this!

```
let rec same (i: int) (j: int) : bool = if i=0 then j=0 else j>0 && same (i-1) (j-1)
```

Theorem: ∀ x:nat.   same x 3 = same 3 x

By induction on x.

Case: x=0

same x 3

== same 0 3

== if 0=0 then 3=0 else …

⋮

== same 3 x

Now prove this!

let rec same (i: int) (j: int) : bool = if i=0 then j=0 else j>0 && same (i-1) (j-1)

Theorem: ∀ x:nat.   same x 3 = same 3 x

By induction on x.

Case: x=0

same x 3

== same 0 3

== if 0=0 then 3=0 else …

== 3=0

== false

⋮

== same 3 x

Now prove this!

let rec same (i: int) (j: int) : bool = if i=0 then j=0 else j>0 && same (i-1) (j-1)

Theorem: ∀ x:nat.   same x 3 = same 3 x

By induction on x.

Case: x=0

```
     same x 3
  == same 0 3
  == if 0=0 then 3=0 else …
  == 3=0
  == false
        ⋮
  == if 3=0 then 0=0 else 0>0 && same (3-1) (0-1)
  == same 3 x
```

Now prove this!

let rec same (i: int) (j: int) : bool = if i=0 then j=0 else j>0 && same (i-1) (j-1)

Theorem: ∀ x:nat.   same x 3 = same 3 x

By induction on x.

Case: x=0

      same x 3
   == same 0 3
   == if 0=0 then 3=0 else …
   == 3=0
   == false
   == false && same (3-1) (0-1)
   == 0>0 && same (3-1) (0-1)
   == if 3=0 then 0=0 else 0>0 && same (3-1) (0-1)
   == same 3 x

Done with Case: x=0.

Now prove this!

let rec same (i: int) (j: int) : bool = if i=0 then j=0 else j>0 && same (i-1) (j-1)

Theorem: ∀ x:nat.   same x 3 = same 3 x
By induction on x.

Where a satisfies I.H.,
same a 3 = same 3 a

Case: x=a+1, where a:nat

    same x 3
  == same (a+1) 3

⋮

  == same 3 x

Now prove this!

```
let rec same (i: int) (j: int) : bool = if i=0 then j=0 else j>0 && same (i-1) (j-1)
```

Theorem: ∀ x:nat.   same x 3 = same 3 x

By induction on x.

Case: x=a+1, where a:nat

        same x 3
  == same (a+1) 3
  == if (a+1)=0 then 3=0 else 3>0 && same (a+1-1) (3-1)

  ⋮

  == same 3 x

Now prove this!

let rec same (i: int) (j: int) : bool = if i=0 then j=0 else j>0 && same (i-1) (j-1)

Theorem: ∀ x:nat.   same x 3 = same 3 x

By induction on x.

Case: x=a+1, where a:nat

      same x 3
  == same (a+1) 3
  == if (a+1)=0 then 3=0 else 3>0 && same (a+1-1) (3-1)
  == 3>0 && same a 2
  == same a 2

      ⋮

  == same 3 x

Now prove this!

```
let rec same (i: int) (j: int) : bool = if i=0 then j=0 else j>0 && same (i-1) (j-1)
```

Theorem: ∀ x:nat.   same x 3 = same 3 x

By induction on x.

Case: x=a+1, where a:nat

```
      same x 3
  == same (a+1) 3
  == if (a+1)=0 then 3=0 else 3>0 && same (a+1-1) (3-1)
  == 3>0 && same a 2
  == same a 2
        ⋮
  == same 2 a
  == a+1>0 && same 2 a
  == if 3=0 then (a+1)=0 else a+1>0 && same (3-1) (a+1-1)
  == same 3 x
```

Now prove this!

```
let rec same (i: int) (j: int) : bool = if i=0 then j=0 else j>0 && same (i-1) (j-1)
```

Theorem: ∀ x:nat.   same x 3 = same 3 x

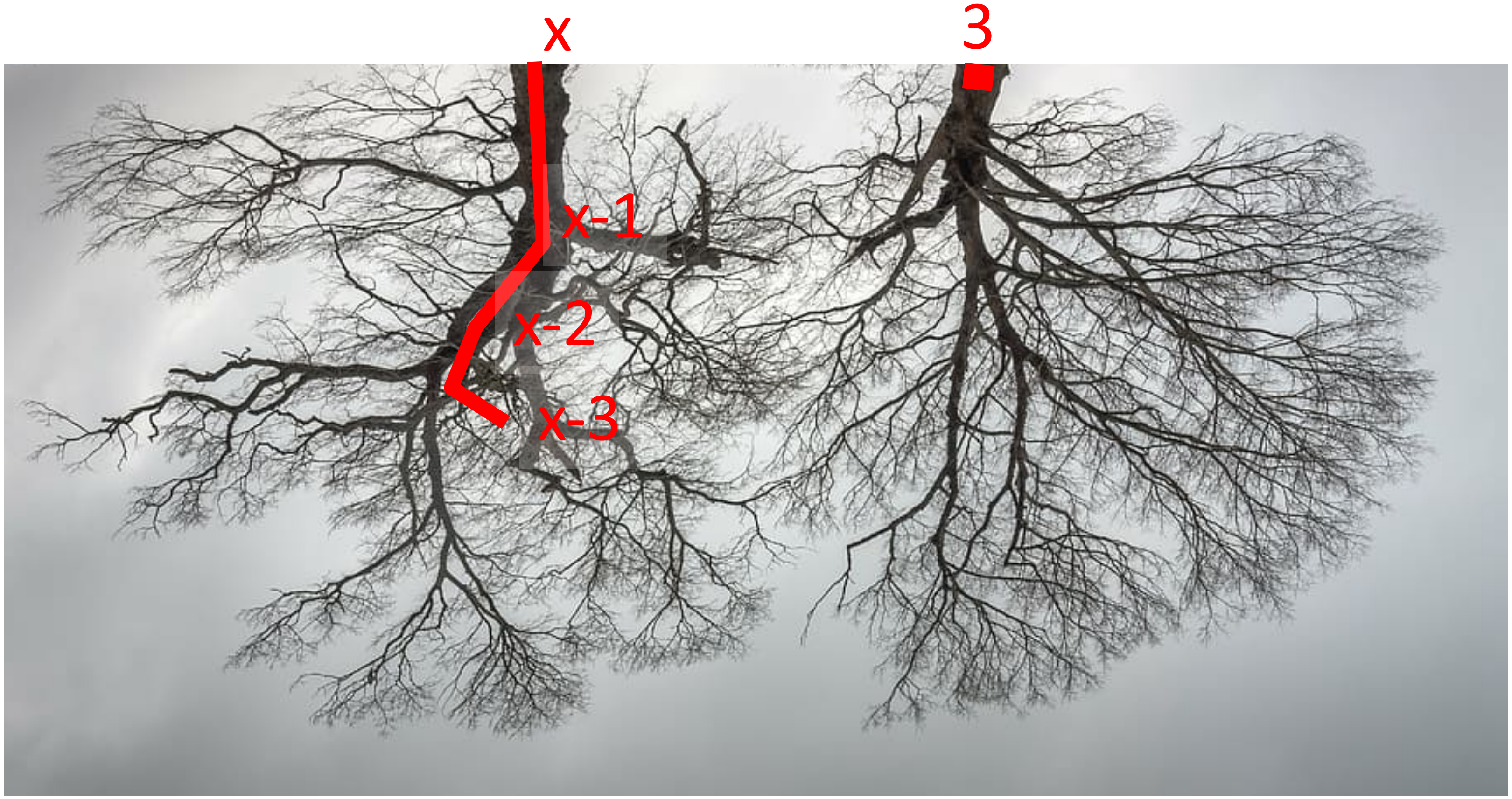
By induction on x.

Case: x=a+1, where a:nat

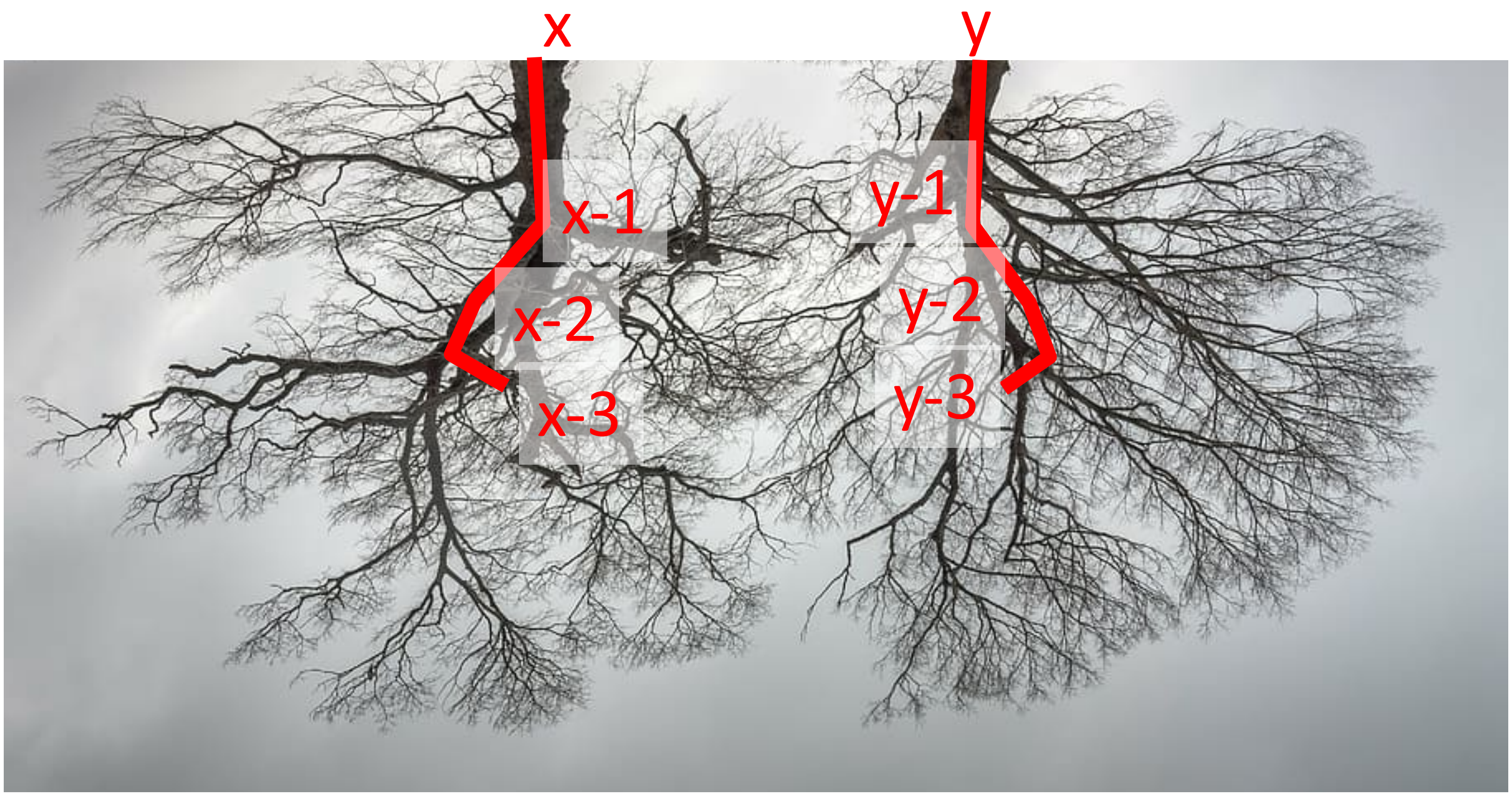
```
      same x 3
  == same (a+1) 3
  == if (a+1)=0 then 3=0 else 3>0 && same (a+1-1) (3-1)
  == 3>0 && same a 2
  == same a 2
         ⋮
  == same 2 a
  == a+1>0 && same 2 a
  == if 3=0 then (a+1)=0 else a+1>0 && same (3-1) (a+1-1)
  == same 3 x
```

Induction hyp tells us:
same a 3 = same 3 a

What’s the problem?



What’s the problem?



Now prove this!

```
let rec same (i: int) (j: int) : bool = if i=0 then j=0 else j>0 && same (i-1) (j-1)
```

Theorem 3: ∀ x:nat.   same x three = same three x

First, prove a more general theorem:

Theorem 4: ∀ x:nat. ∀ y:nat. same x y = same y x

Exercise

- Finish the proof yourself!

It looks just like the proof about

∀ t:tree. ∀ u:tree. mirror t u = mirror u t

Conclusion:

WALK DOWN BOTH TREES TOGETHER, IN YOUR PROOF;

DON’T STAY AT THE ROOT OF ONE OF THE TREES.