Lecture 5: Lists

int_list
space

A "list of numbers" is either,

- [], or "nil" / "empty"
- x :: xs where x : int
  "Cons" xs : int list
  "list of numbers"

→ And that's it!

A boolean is either

- true, or
- false

→ and that's it!

A nat num. is either

- O, or
- 1+n, where n nat

→ and that's it!

E.g.

nice notation

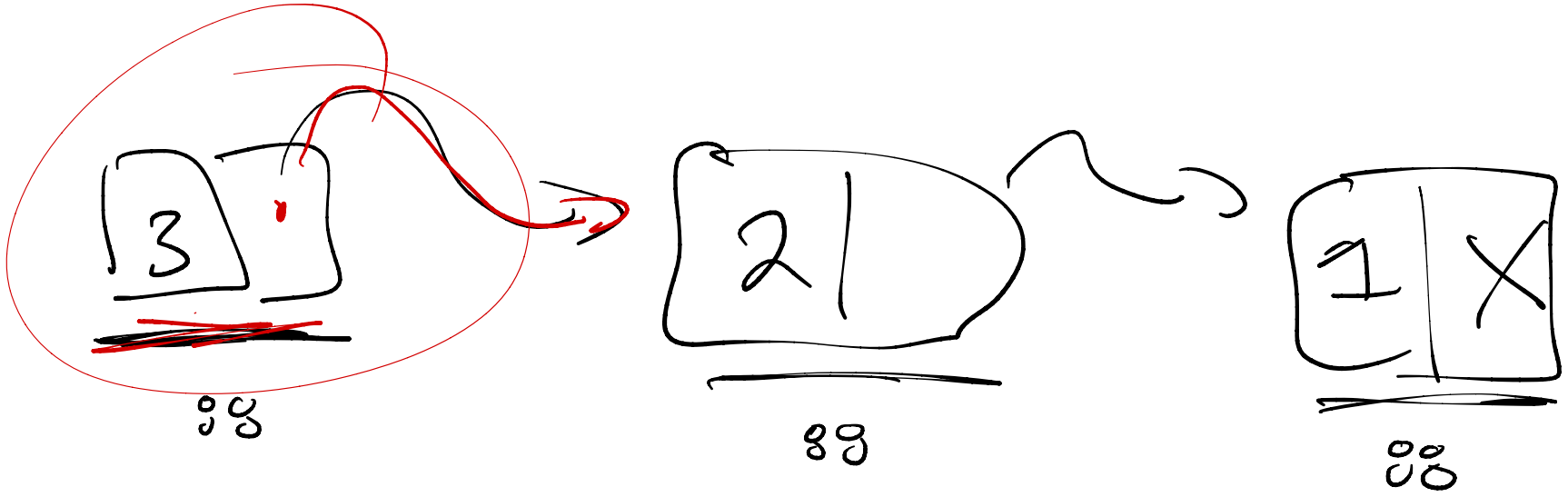
```
[] : int list                          []

1 :: [] : int list                     [1]

2 :: [] : int list                     [2]

2 :: (1 :: []) : int list              [2, 1]

3 :: (2 :: (1 :: [])) : int list       [3, 2, 1]
```



X = []

3 :: (2 :: (1 :: []))

"cons"
"node"

[3, 2, 1]

"nil"
"empty"

(* Purpose: compute the length of a list *)

(* E.g. length([3,2,1]) = 3

length(3::(2::(1::[]))) = 3

length(4::2::(1::[])) = 3

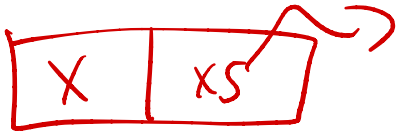
*)

fun length (xs : int list) : int =

case xs of

[] => 0

| x :: xs => 1 + length(xs)



new variables bound by the case

If l is null then ⟶

else

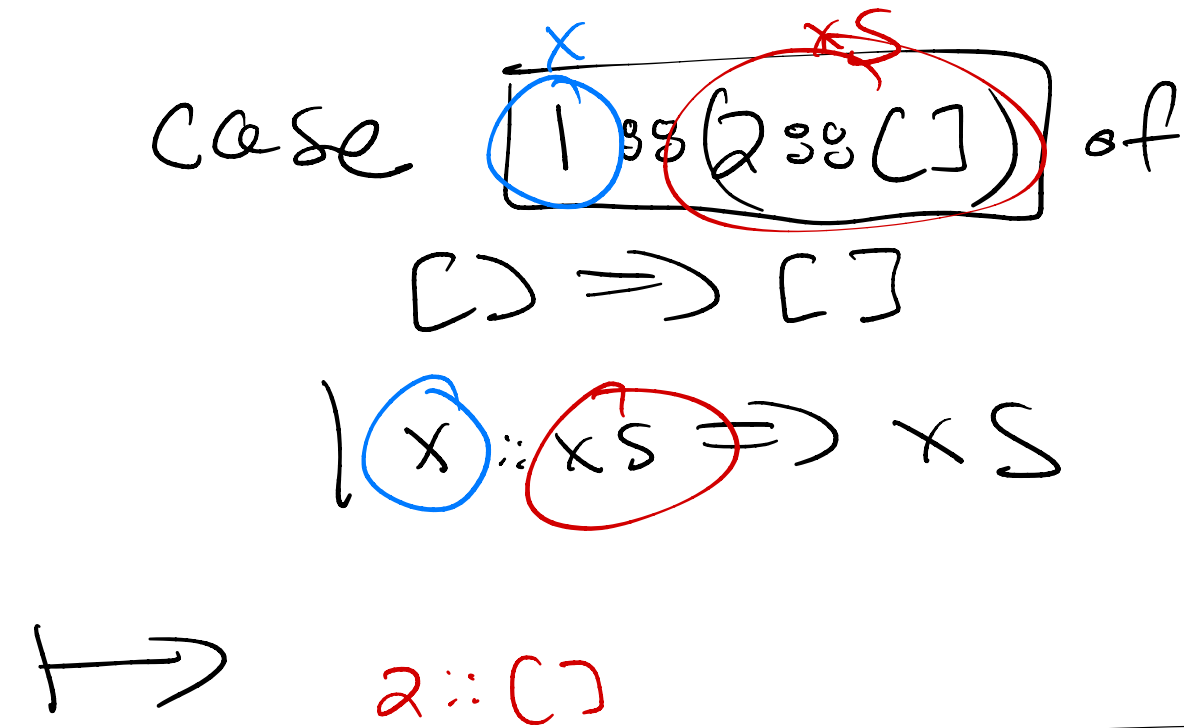
x = l->head

xs = l->next

...

Don't do this:

```
case l of
    [] => 0
  | _ => let val x::xs = l
         in
             1 + length(xs)
         end
```


case 1 :: (2 :: []) of
[] => []
| x :: xs => xs
|-> 2 :: []
case 1 :: (2 :: []) of
[] => 17
| x :: xs => x
|-> 1

length(3::(2::1::[]))

↦ case 3::2::1::[] of [] ⇒ 0
| x::xs ⇒ 1+length(xs)

```
fun length(l) =
  case l of
    [] ⇒ 0
  | x::xs ⇒ 1+length(xs)
```

↦ 1 + length(2::1::[])

↦ 1 + case 2::(1::[]) of [] ⇒ 0
| x::xs ⇒ 1+length(xs)

↦ 1 + (1+ length(1::[]))

↦ 1 + 1 + case 1::[] of [] ⇒ 0
| x::xs ⇒ 1+length(xs)

↦ 1 + 1 + 1 + length([])

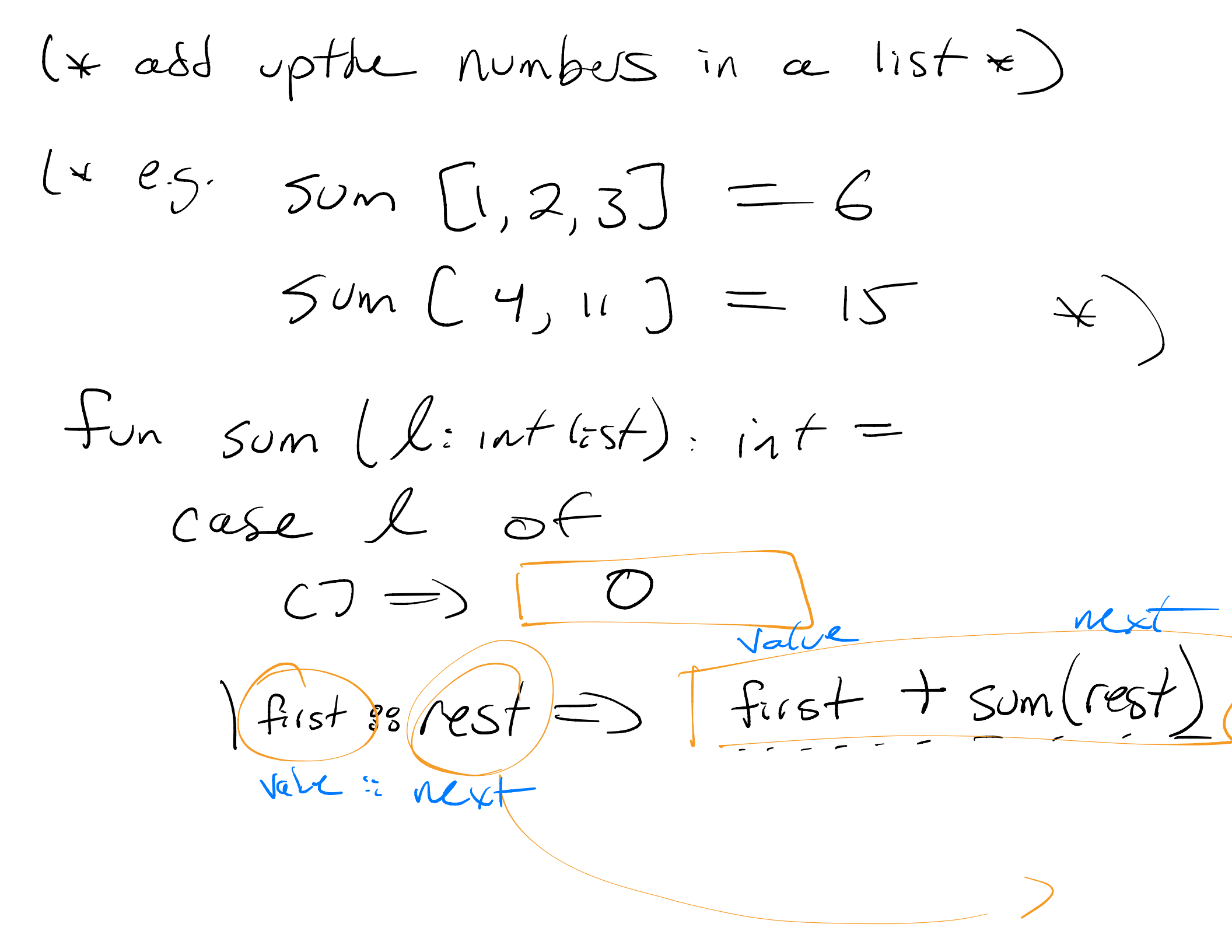
↦ 1 + 1 + 1 + case [] of [] ⇒ 0

↦ 1 + 1 + 1 + 0 ↦* 3

```
(* add up the numbers in a list *)

(* e.g. sum [1,2,3] = 6
        sum [4,11] = 15      *)

fun sum (l: int list) : int =
    case l of
        [] => 0
      | first :: rest => first + sum(rest)
```



```
fun sum (l: int list): int =
  case l of
    [] => 0
  | first :: rest => first + sum(rest)
```

sum(4 :: (11 :: []))

↦ case 4 :: (11 :: []) of
  [] => 0
  | first :: rest => first + sum(rest)

↦ 4 + sum(11 :: [])

↦ 4 + case (11 :: []) of [] => 0
  | f :: r => f + sum(r)

↦ 4 + (11 + sum [])

↦ 4 + 11 + (case [] of [] => 0 | ————)

↦ 4 + 11 + 0

↦ 15

(* Purpose: Given a list of employee's salaries.

1

Give everyone a raise by

a given amount a *)

(* 3

raiseSalaries([15, 16, 14], 5)

= [20, 21, 19] *)

2 fun raiseSalaries(l: int list, a: int): int list =

2) fun raiseSalaries(l: int list, a: int): int list =

case l of

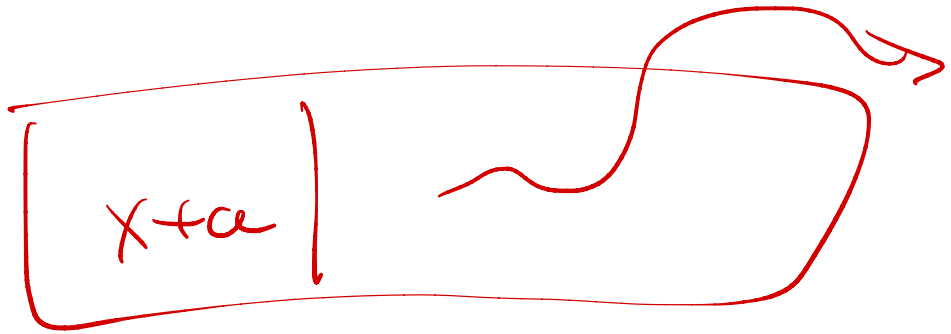
[] => []

| x :: xs =>

x first value
xs rest next
raiseSalary(xs)

(x+a) :: raiseSalaries(xs, a)

case a of
0 =>
1 =>
???



rS ([15], [16,14], 5)

↦ case ↗ of ([] -> [] | x:xs -> (x+a) : rS(xs a)

↦ (15+5) : rS([16,14], 5)

↦ 20 : rS([16,14], 5)

↦ 20 : (16+5) : rS([14], 5)

↦ 20 : 21 : rS([14], 5))

↦ 20 : 21 : (14+5) : rS[]

↦ 20 : 21 : 19 : []

is [20, 21, 19]

Template: raiseSalaries

sum

int list

int

```
fun f(l: int list) : T =
    case l of
        [] => e0 : T
      | x::xs => e1 : T
```

int list

int

int list

int

e1 gets to use x : int, xs : int list, f(xs) : T

int

int list

① case [] of [] ⇒ e_0 | ....

↦ e_0

case l of

[] ⇒ [e_0] : T (int list → int)

| x::xs ⇒ [e_1] : T (int list → int)

② case 4 (11::[]) of [] ⇒ e_0 | x::xs ⇒ e_1

↦ e_1 with v for x

vs for xs

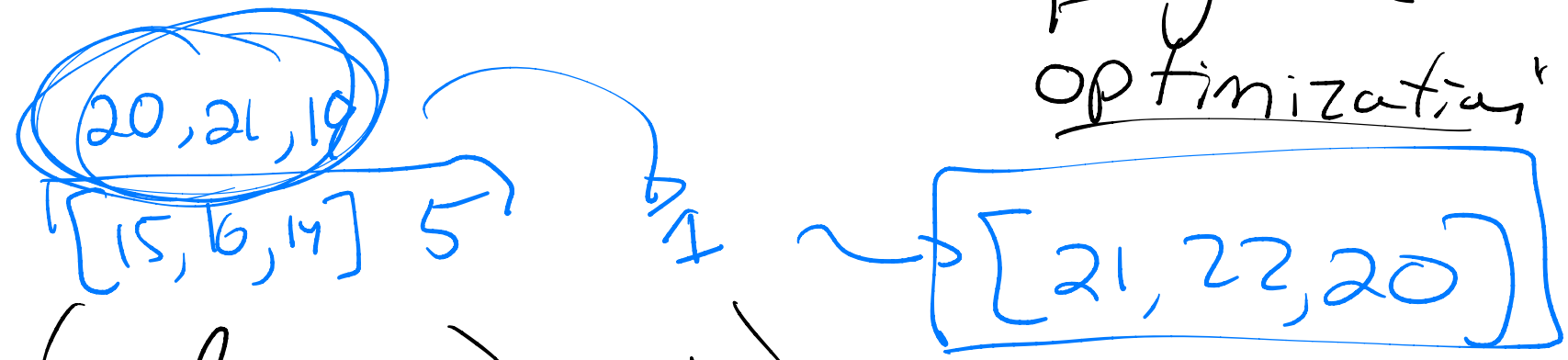
List induction

To prove something about all lists, you should

1. prove it for []
2. prove it for x::xs

   assuming ind. hyp. for xs

Theorem: Prove correct a "program optimization"



$rS(rS(l, a), b)$ "slow"

$=$

$rS(l, a+b)$ "fast"

6

Induction on list l

Case for []:

To show: $\beta(rS([], a), b) = rS([], a+b)$

$rS(rS([], a), b)$

$\mapsto rs([], b)$

$\mapsto []$

$\leftarrow rS([], a+b)$

$rS([], a+b)$

$\mapsto []$

✓

Case for $x :: xs$:

Inductive hypothesis (IH): $rS(rS(xs, a), b) = rS(xs, a+b)$

To Show: $rS(rS(x :: xs, a), b) = rS(x :: xs, a+b)$

$rS(rS(x :: xs, a), b)$

$\mapsto rS((x+a) :: rS(xs, a), b)$

$\mapsto (x+a)+b :: rS(rS(xs, a), b)$

$= x+(a+b) :: rS(rS(xs, a), b)$

$= x+(a+b) :: rS(xs, a+b)$ by IH

$\leftarrow rS(x :: xs, a+b)$

General rules: for $e_1 = e_2$

(1) If $e_1 \mapsto e_2$ then $e_1 = e_2$

(2) R $e_1 = e_1$

S $e_1 = e_2$ then $e_2 = e_1$

T If $e_1 = e_2$ and $e_2 = e_3$ then $e_1 = e_3$

(3) replace equals with equals inside a bigger expression