```
let rec fl f a l = match l with [] -> a
| x::xs -> fl f (f a x) xs;;

let rec fr f l a = match l with [] -> a
| x::xs -> f x (fr f xs a);;
let rec rev_map f l a = match l with [] -> a
| x::xs -> rev_map f xs (f x :: a);;
let (+) a b = a + b;;
fl (+) 0 (rev_map (fun x -> x * 2) l []) = fr (fun x a -> a + 2 * x) l 0

base case: l = []
fl (+) acc (rev_map (fun x -> x * 2) [] [])
rev_map (fun x -> x * 2) [] [] = match [] with [] -> a
| x::xs -> rev_map f xs (f x :: a)
= []

fl (+) acc [] =  match [] with [] -> acc
| x::xs -> fl f (f a x) xs =
acc =
= fr (fun x a -> a + 2 * x) [] 0 + acc = match [] with [] -> a
```

| x::xs -> f x (fr f xs a)

0 + acc = acc

fr (fun x a -> a + 2 * x) l 0 + acc = fl (+) acc (rev_map (fun x -> x * 2) l [])

fr (fun x a -> a + 2 * x) xs 0 = fl (+) acc (rev_map (fun x -> x * 2) xs []) - acc

for (x::xs)

fl (+) acc (rev_map (fun x -> x * 2) (x::xs) []) =

match x::xs with [] -> a

| x::xs -> fl (+) ((+) a x) xs

= fl (+) (acc + 2x) (rev_map (fun x -> x * 2) xs []) = fr (fun x a -> a + 2 * x) xs 0 + acc + 2x

acc + fr (fun x a -> a + 2 * x) (x::xs) 0 = match (x::xs) with [] -> a

| x::xs -> f x (fr f xs a)

=

acc + (fun x a -> a + 2 * x) x (fr f xs a)

= acc + (fun x a -> a + 2 * x) x (fr (fun x a -> a + 2 * x) xs 0) =

acc + (fun x a -> a + 2 * x) x (fl (+) acc (rev_map (fun x -> x * 2) xs []) - acc) =

fl (+) acc (rev_map (fun x -> x * 2) xs []) + 2x = 2x + acc + fr (fun x a -> a + 2 * x) xs 0 = fr (fun x a -> a + 2 * x) xs 0 + (2x+acc)

fl (+) (acc + 2x) (rev_map (fun x -> x * 2) xs [])

= 2x + (fl (+) acc (rev_map (fun x -> x * 2) xs []))