```
(fun a b c -> c (a + b)) 3
```

int → int → (int → 'a) → 'a

```
(fun a b -> (+) b)
```

'a -> int -> (int -> int)

'a -> int -> int -> int

```
(fun a b c -> b (c a) :: [a]) "x"
```

('a -> string) -> (string -> 'a) -> string list

b c output

```
(fun a b -> List.fold_left b 1 (List.map ( * ) a))
```

(int -> int) list

(int list) -> (int -> (int -> int) -> int) -> int

a

b

output

```
(let x = List.map in x (<))
```

List.map (<)

('a -> 'b) -> 'a list -> 'b list

'a list -> ('a -> bool) list