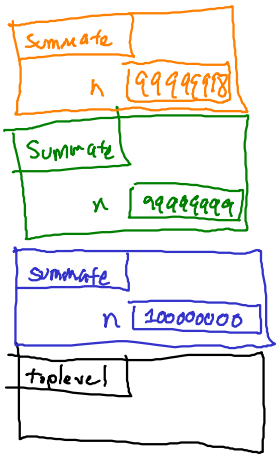
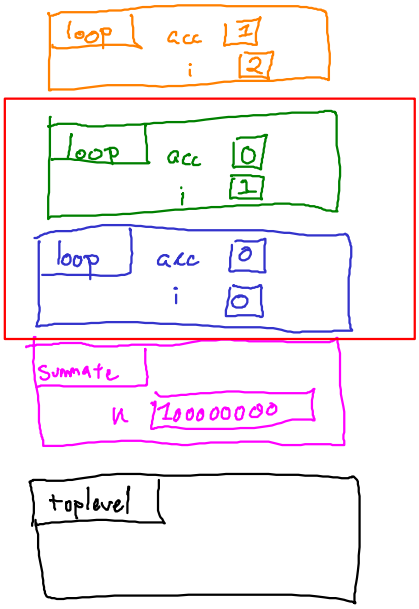
```
let rec summate (n:int):int =
  if n=0 then 0 else
     n + summate (n-1)
in
 summate 100000000
```

```
let rec summate (n:int):int =
  if n=0 then 0 else
     n + summate (n-1)
in
 100000000 + ( 99999999 + ( if 99999998 =0 then 0 else
                              99999998 + summate (99999998-1) ) )
```



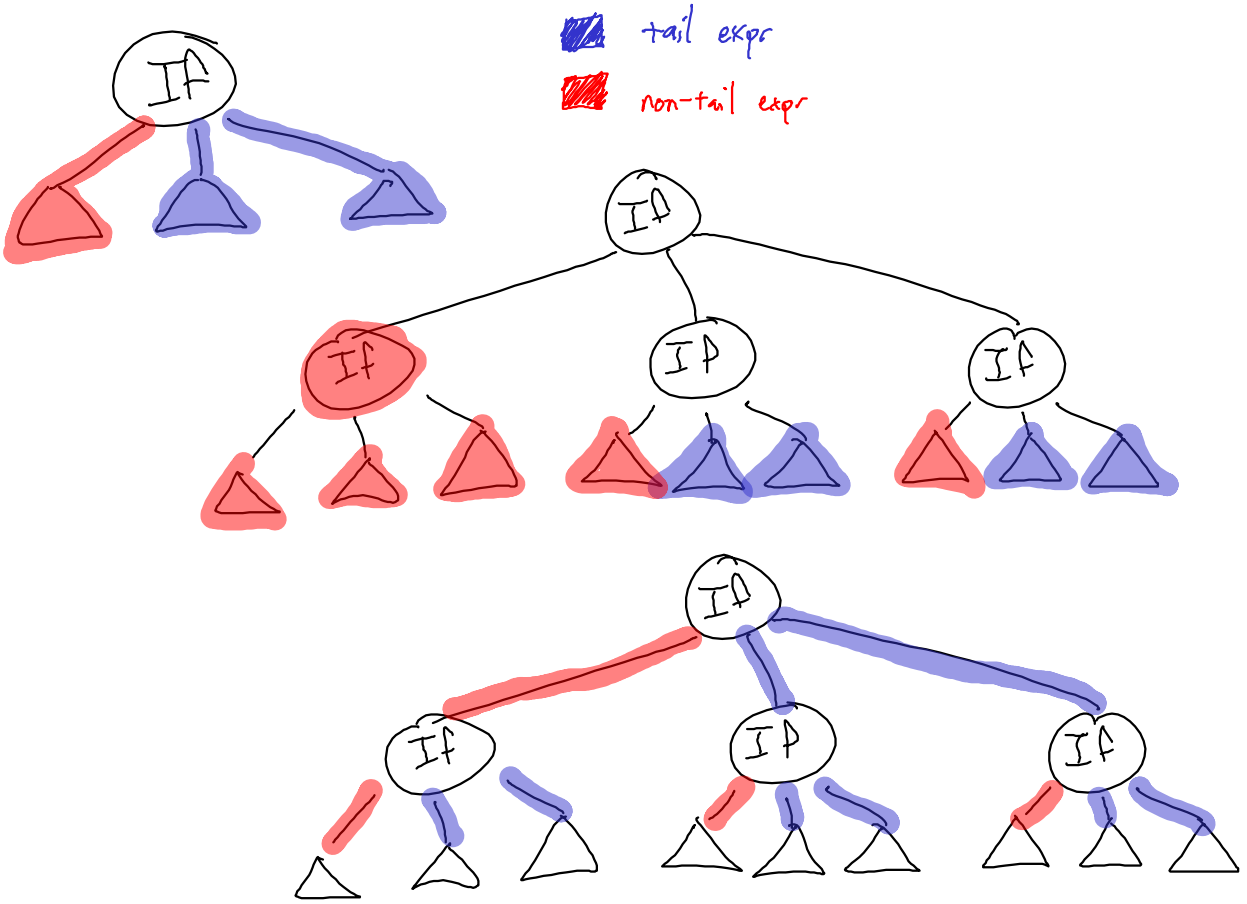
```
let summate (n:int) : int =
    let rec loop (acc:int) (i:int) : int =
        if i=n+1 then acc else
            loop (acc+i) (i+1)
    in
    loop 0 0
in
summate 100 000 000
```



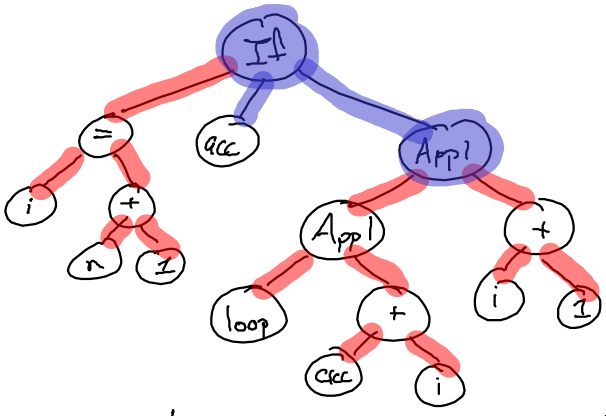

```
let rec loop (acc:int) (i:int) : int =
    if i= 100 000 000 +1 then acc else
        loop (acc+i) (i+1)
in
loop 3 3
```

Tail call optimization removes stack frames on "tail calls" in order to allow recursive functions to use less stack memory

A tail expression of a larger expression is a subexpression which ① produces the result of larger expression and ② is the last thing the larger expression does.




```
type expr =
  | EInt of int
  | EBool of bool
  | EUnaryOp of unary_operator * expr
  | EBinaryOp of binary_operator * expr * expr
  | ETuple of expr list
  | ELet of string * expr * expr
  | EVar of string
  | EIf of expr * expr * expr
  | EAppl of expr * expr
  | ESet of expr * expr * expr
```



A tail expression of a larger expression is a subexpression which ① produces the result of larger expression and ② is the last thing the larger expression does.

A tail call is an application expression which is a tail expression of the body of a function.