# Relazione

Sommario

[Relazione 1](#_Toc146903334)

[Esercizio 1 1](#_Toc146903335)

[Riscrittura delle funzioni 1](#_Toc146903336)

[Esecuzione 1](#_Toc146903337)

## Esercizio 1

Si assume che il contesto che forza l’esecuzione completa del risultato (che in questo esercizio è una lista) avvenga come avverrebbe se sei eseguisse un foldl o mapM sulla lista risultato, quindi nel caso il risultato fosse del tipo (exp1 : exp2), l’espressione 1 sarebbe valutata completamente prima di iniziare la valutazione dell’espressione 2. Il ché influenza l’ordine di esecuzione di sottoespressioni condivise tra le due.

Le regole per error “ouch !!” non sono state rimosse intenzionalmente nella riscrittura per tenere la procedura “più basilare” anche se sono chiaramente ridondanti.

### Riscrittura delle funzioni

enumFrom@(Float,Char) = \ z -> case z of

(x, c) -> : a[(\*@Float x 1.1@Float , succ@Char c)] (enumFrom@(Float,Char) a)

f :: Float -> ( any , ( Float , Char )) -> Bool

f = \ x -> \ y' -> case y' of

(\_ , y'') -> case y'' of

( y , \_ ) -> <@Float x y

myMap = \ f -> \ l -> case l of

: x xs -> : (f x) (myMap f xs)

[] -> []

myFilt = \ p -> \ l -> case l of

[] -> []

: x xs -> if (p x) then (: x ys[myFilt p xs]) else ys

myZip = \ l1 -> \ l2 -> case l1 of

[] -> []

: x xs -> case l2 of

[] -> []

: y ys -> : (x,y) (myZip xs ys)

\_ -> error "ouch !!"

\_ -> error "ouch !!"

### Esecuzione

myMap snd (myFilt (f 0@Float) (myZip (error "ERROR":"do") [(2,'a')..]))

myMap snd (myFilt (f 0@Float) (myZip (: (error "ERROR") "do") (enumFrom@(2@Float,'a'@Char))))

{-

prima iterazione

-}

case (myFilt (f 0@Float) (myZip (: (error "ERROR") "do") (enumFrom@(Float,Char) (2@Float,'a'@Char)))) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

case (case (myZip (: (error "ERROR") "do") (enumFrom@(Float,Char) (2@Float,'a'@Char)))of

[] -> []

: x xs -> if (f'[f 0@Float] x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

case (case (case (: (error "ERROR") "do") of

[] -> []

: x xs -> case (enumFrom@(Float,Char) (2@Float,'a'@Char)) of

[] -> []

: y ys -> : (x,y) (myZip xs ys)

\_ -> error "ouch !!"

\_ -> error "ouch !!"

) of

[] -> []

: x xs -> if (f'[f 0@Float] x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

case (case (case (enumFrom@(Float,Char) (2@Float,'a'@Char)) of

[] -> []

: y ys -> : ((error "ERROR"), y) (myZip "do" ys)

\_ -> error "ouch !!"

) of

[] -> []

: x xs -> if (f'[f 0@Float] x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

case (case (case (case (2@Float,'a'@Char)of

(x, c) -> : a[(\*@Float x 1.1@Float , succ@Char c)] (enumFrom@(Float,Char) a)

) of

[] -> []

: y ys -> : ((error "ERROR"), y) (myZip "do" ys)

\_ -> error "ouch !!"

) of

[] -> []

: x xs -> if (f'[f 0@Float] x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

case (case (case (: a[(\*@Float 2@Float 1.1@Float , succ@Char 'a'@Char)] (enumFrom@(Float,Char) a)) of

[] -> []

: y ys -> : ((error "ERROR"), y) (myZip "do" ys)

\_ -> error "ouch !!"

) of

[] -> []

: x xs -> if (f'[f 0@Float] x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

case (case (: ((error "ERROR"), a[(\*@Float 2@Float 1.1@Float , succ@Char 'a'@Char)]) (myZip "do" (enumFrom@(Float,Char) a))) of

[] -> []

: x xs -> if (f'[f 0@Float] x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

case (if (f'[f 0@Float] x[((error "ERROR"), a[(\*@Float 2@Float 1.1@Float , succ@Char 'a'@Char)])]) then

(: x ys[myFilt f' (myZip "do" (enumFrom@(Float,Char) a))])

else

ys) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

case (if (f'[(\ y' -> case y' of

(\_ ,y'') -> case y'' of

( y , \_ ) -> <@Float 0@Float y

)] x[((error "ERROR"), a[(\*@Float 2@Float 1.1@Float , succ@Char 'a'@Char)])])

then

(: x ys[myFilt f' (myZip "do" (enumFrom@(Float,Char) a))])

else

ys) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

{-

ATTENZIONE!

per leggibilità continuerò ad usare f' come placeholder per:

\ y' -> case y' of

(\_ ,y'') -> case y'' of

( y , \_ ) -> <@Float 0@Float y

sarebbe la valutazione di (f 0)

-}

case (if (case x[((error "ERROR"), a[(\*@Float 2@Float 1.1@Float , succ@Char 'a'@Char)])] of

(\_ ,y'') -> case y'' of

( y , \_ ) -> <@Float 0@Float y

)

then

(: x ys[myFilt f' (myZip "do" (enumFrom@(Float,Char) a))])

else

ys) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

case (if (case a[(\*@Float 2@Float 1.1@Float , succ@Char 'a'@Char)] of

( y , \_ ) -> <@Float 0@Float y

)

then

(: ((error "ERROR"), a) ys[myFilt f' (myZip "do" (enumFrom@(Float,Char) a))])

else

ys) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

case (if (<@Float 0@Float t[\*@Float 2@Float 1.1@Float])

then

(: ((error "ERROR"), a[(t, succ@Char 'a'@Char)])

ys[myFilt f' (myZip "do" (enumFrom@(Float,Char) a))])

else

ys) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

case (if (<@Float 0@Float 2.2@Float)

then

(: ((error "ERROR"), a[(2.2@Float, succ@Char 'a'@Char)])

ys[myFilt f' (myZip "do" (enumFrom@(Float,Char) a))])

else

ys) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

case (if True

then

(: ((error "ERROR"), a[(2.2@Float, succ@Char 'a'@Char)])

ys[myFilt f' (myZip "do" (enumFrom@(Float,Char) a))])

else

ys) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

case (: ((error "ERROR"), a[(2.2@Float, succ@Char 'a'@Char)])

ys[myFilt f' (myZip "do" (enumFrom@(Float,Char) a))]

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []

: (snd ((error "ERROR"), a[(2.2@Float, succ@Char 'a'@Char)]))

(myMap snd (myFilt f' (myZip "do" (enumFrom@(Float,Char) a))))

{-

viene forzata la valutazione completa

-}

: a[(2.2@Float, succ@Char 'a'@Char)]

(myMap snd (myFilt f' (myZip "do" (enumFrom@(Float,Char) a))))

: (2.2@Float, 'b'@Char)

(myMap snd (myFilt f' (myZip "do" (enumFrom@(Float,Char) (2.2@Float, 'b'@Char)))))

{-

seconda iterazione

-}

: (2.2@Float, 'b'@Char)

(case (myFilt f' (myZip "do" (enumFrom@(Float,Char) (2.2@Float, 'b'@Char)))) of

: x xs -> : (snd x) (myMap snd xs))

[] -> []

: (2.2@Float, 'b'@Char)

(case (case (myZip "do" (enumFrom@(Float,Char) (2.2@Float, 'b'@Char))) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])

: (2.2@Float, 'b'@Char)

(case (case (case "do" of

[] -> []

: x xs -> case (enumFrom@(Float,Char) (2.2@Float, 'b'@Char)) of

[] -> []

: y ys -> : (x,y) (myZip xs ys)

\_ -> error "ouch !!"

\_ -> error "ouch !!"

) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])

: (2.2@Float, 'b'@Char)

(case (case (case (enumFrom@(Float,Char) (2.2@Float, 'b'@Char)) of

[] -> []

: y ys -> : ('d'@Char, y) (myZip "o" ys)

\_ -> error "ouch !!"

) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])

: (2.2@Float, 'b'@Char)

(case (case (case (case (2.2@Float, 'b'@Char) of

(x, c) -> : a[(\*@Float x 1.1@Float , succ@Char c)] (enumFrom@(Float,Char) a)

) of

[] -> []

: y ys -> : ('d'@Char, y) (myZip "o" ys)

\_ -> error "ouch !!"

) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])

: (2.2@Float, 'b'@Char)

(case (case (case (: a[(\*@Float 2.2@Float 1.1@Float , succ@Char 'b'@Char)] (enumFrom@(Float,Char) a)) of

[] -> []

: y ys -> : ('d'@Char, y) (myZip "o" ys)

\_ -> error "ouch !!"

) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])

: (2.2@Float, 'b'@Char)

(case (case (: ('d'@Char, a[(\*@Float 2.2@Float 1.1@Float , succ@Char 'b'@Char)]) (myZip "o" (enumFrom@(Float,Char) a))) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])

: (2.2@Float, 'b'@Char)

(case (if (f' x[('d'@Char, a[(\*@Float 2.2@Float 1.1@Float , succ@Char 'b'@Char)])]) then

(: x ys[myFilt f' (myZip "o" (enumFrom@(Float,Char) a))])

else

ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])

: (2.2@Float, 'b'@Char)

(case (if (case (x[('d'@Char, a[(\*@Float 2.2@Float 1.1@Float , succ@Char 'b'@Char)])]) of

(\_, y'') -> case y'' of

(y, \_) -> <@Float 0@Float y)

then

(: x ys[myFilt f' (myZip "o" (enumFrom@(Float,Char) a))])

else

ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])

: (2.2@Float, 'b'@Char)

(case (if (case a[(\*@Float 2.2@Float 1.1@Float , succ@Char 'b'@Char)] of

(y, \_) -> <@Float 0@Float y)

then

(: ('d'@Char, a) ys[myFilt f' (myZip "o" (enumFrom@(Float,Char) a))])

else

ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])

: (2.2@Float, 'b'@Char)

(case (if (<@Float 0@Float y[\*@Float 2.2@Float 1.1@Float])

then

(: ('d'@Char, a[(y, succ@char 'b'@char)]) ys[myFilt f' (myZip "o" (enumFrom@(Float,Char) a))])

else

ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])

: (2.2@Float, 'b'@Char)

(case (if (<@Float 0@Float 2.4200000000000004@Float)

then

(: ('d'@Char, a[(2.4200000000000004@Float, succ@char 'b'@char)])

ys[myFilt f' (myZip "o" (enumFrom@(Float,Char) a))])

else

ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])

{-

da qui in poi abbrevio `2.4200000000000004` con `2.42`

-}

: (2.2@Float, 'b'@Char)

(case (if True

then

(: ('d'@Char, a[(2.42@Float, succ@char 'b'@char)])

ys[myFilt f' (myZip "o" (enumFrom@(Float,Char) a))])

else

ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])

: (2.2@Float, 'b'@Char)

(case (: ('d'@Char, a[(2.42@Float, succ@char 'b'@char)])

ys[myFilt f' (myZip "o" (enumFrom@(Float,Char) a))]) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])

: (2.2@Float, 'b'@Char)

(: (snd (: ('d'@Char, a[(2.42@Float, succ@char 'b'@char)])))

(myMap snd (myFilt f' (myZip "o" (enumFrom@(Float,Char) a)))))

{-

viene forzata la valutazione completa

-}

: (2.2@Float, 'b'@Char)

(: a[(2.42@Float, succ@char 'b'@char)]

(myMap snd (myFilt f' (myZip "o" (enumFrom@(Float,Char) a)))))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(myMap snd (myFilt f' (myZip "o" (enumFrom@(Float,Char) (2.42@Float, 'c'@char))))))

{-

terza iterazione

-}

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (myFilt f' (myZip "o" (enumFrom@(Float,Char) (2.42@Float, 'c'@char)))) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (case (myZip "o" (enumFrom@(Float,Char) (2.42@Float, 'c'@char))) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (case (case "o" of

[] -> []

: x xs -> case (enumFrom@(Float,Char) (2.42@Float, 'c'@char)) of

[] -> []

: y ys -> : (x,y) (myZip xs ys)

\_ -> error "ouch !!"

\_ -> error "ouch !!"

) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (case (case (enumFrom@(Float,Char) (2.42@Float, 'c'@char)) of

[] -> []

: y ys -> : ('o'@Char, y) (myZip [] ys)

\_ -> error "ouch !!"

) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (case (case (case (2.42@Float, 'c'@char) of

(x, c) -> : a[(\*@Float x 1.1@Float , succ@Char c)] (enumFrom@(Float,Char) a)

) of

[] -> []

: y ys -> : ('o'@Char, y) (myZip [] ys)

\_ -> error "ouch !!"

) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (case (case (: a[(\*@Float 2.42@Float 1.1@Float , succ@Char 'c'@char)] (enumFrom@(Float,Char) a)) of

[] -> []

: y ys -> : ('o'@Char, y) (myZip [] ys)

\_ -> error "ouch !!"

) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (case (: ('o'@Char, a[(\*@Float 2.42@Float 1.1@Float , succ@Char 'c'@char)])

(myZip [] (enumFrom@(Float,Char) a))) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt f' xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (if (f' x[('o'@Char, a[(\*@Float 2.42@Float 1.1@Float , succ@Char 'c'@char)])]) then

(: x ys[myFilt f' (myZip [] ys[(enumFrom@(Float,Char) a)])])

else

ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (if (case x[('o'@Char, a[(\*@Float 2.42@Float 1.1@Float , succ@Char 'c'@char)])] of

(\_ , y'') -> case y'' of

( y , \_ ) -> <@Float 0@Float y)

then

(: x ys[myFilt f' (myZip [] (enumFrom@(Float,Char) a))])

else

ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (if (case a[(\*@Float 2.42@Float 1.1@Float , succ@Char 'c'@char)] of

( y , \_ ) -> <@Float 0@Float y)

then

(: ('o'@char, a) ys[myFilt f' (myZip [] (enumFrom@(Float,Char) a))])

else

ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (if (<@Float 0@Float t[\*@Float 2.42@Float 1.1@Float])

then

(: ('o'@char, a[(t , succ@Char 'c'@char)])

ys[myFilt f' (myZip [] (enumFrom@(Float,Char) a))])

else

ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (if (<@Float 0@Float 2.6620002@Float)

then

(: ('o'@char, a[(2.6620002@Float , succ@Char 'c'@char)])

ys[myFilt f' (myZip [] (enumFrom@(Float,Char) a))])

else

ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

{-

da qui in poi abbrevio `2.6620002` con `2.662`

-}

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (if True

then

(: ('o'@char, a[(2.662@Float , succ@Char 'c'@char)])

ys[myFilt f' (myZip [] (enumFrom@(Float,Char) a))])

else

ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(case (: ('o'@char, a[(2.662@Float , succ@Char 'c'@char)])

ys[myFilt f' (myZip [] (enumFrom@(Float,Char) a))]) of

: x xs -> : (snd x) (myMap snd xs)

[] -> []))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(: (snd ('o'@char, a[(2.662@Float , succ@Char 'c'@char)]))

(myMap snd (myFilt f' (myZip [] (enumFrom@(Float,Char) a))))))

{-

viene forzata la valutazione completa

-}

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(: a[(2.662@Float , succ@Char 'c'@char)]

(myMap snd (myFilt f' (myZip [] (enumFrom@(Float,Char) a))))))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(: (2.662@Float, 'd'@char)

(myMap snd (myFilt f' (myZip [] (enumFrom@(Float,Char) (2.662@Float, 'd'@char)))))))

{-

quarta iterazione

-}

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(: (2.662@Float, 'd'@char)

(case (myFilt f' (myZip [] (enumFrom@(Float,Char) (2.662@Float, 'd'@char)))) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(: (2.662@Float, 'd'@char)

(case (case (myZip [] (enumFrom@(Float,Char) (2.662@Float, 'd'@char))) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt p xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(: (2.662@Float, 'd'@char)

(case (case (case [] of

[] -> []

: x xs -> case (enumFrom@(Float,Char) (2.662@Float, 'd'@char)) of

[] -> []

: y ys -> : (x,y) (myZip xs ys)

\_ -> error "ouch !!"

\_ -> error "ouch !!"

) of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt p xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(: (2.662@Float, 'd'@char)

(case (case [] of

[] -> []

: x xs -> if (f' x) then (: x ys[myFilt p xs]) else ys

) of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(: (2.662@Float, 'd'@char)

(case [] of

: x xs -> : (snd x) (myMap snd xs)

[] -> [])))

: (2.2@Float, 'b'@Char)

(: (2.42@Float, 'c'@char)

(: (2.662@Float, 'd'@char)

[]))

[(2.2,'b'),(2.42,'c'),(2.662,'d')]

## Esercizio 2

### Esercizio 1 parte C

* enumFrom ha una sola regola
* f ha una sola regola
* myMap
  + f (x:xs) e \_ [] non hanno overlap perché il secondo costruttore è diverso
* myFilt
  + p (x:xs) e \_ [] non hanno overlap perché il secondo costruttore è diverso
* myZip
  + [] \_ e \_ [] hanno overlap [] []
  + [] \_ e ( x : xs ) ( y : ys ) non hanno overlap perché il primo costruttore è diverso
  + [] \_ e \_ \_ hanno overlap [] []
  + \_ [] e ( x : xs ) ( y : ys ) non hanno overlap perché il secondo costruttore è diverso
  + \_ [] e \_ \_ hanno overlap [] []
  + ( x : xs ) ( y : ys ) e \_ \_ hanno overlap ( x : xs ) ( y : ys )

### Esercizio 1 parte A

* f ha una sola regola
* naiveF ha una sola regola
* validateF ha una sola regola
* transposeMat ha una sola regola
* transposeQT
  + (C x) e (Q ul ur ll lr) non hanno overlap (primo costruttore)
* sumMat ha una sola regola
* sumQt
  + (C x) (C y) e (C x) (Q ul ur ll lr) non hanno overlap (secondo costruttore)
  + (C x) (C y) e (Q ul ur ll lr) (C y) non hanno overlap (primo costruttore)
  + (C x) (C y) e (Q ula ura lla lra) (Q ulb urb llb lrb) non hanno overlap (primo costruttore)
  + (C x) (Q ul ur ll lr) e (Q ul ur ll lr) (C y) non hanno overlap (primo costruttore)
  + (C x) (Q ul ur ll lr) e (Q ula ura lla lra) (Q ulb urb llb lrb)non hanno overlap (primo costruttore)
  + (Q ul ur ll lr) (C y)e (Q ula ura lla lra) (Q ulb urb llb lrb)non hanno overlap (secondo costruttore)
* sumTransposeMat ha una sola regola
* sumTransposeQT
  + (C x) e (Q ul ur ll lr) non hanno overlap (primo costruttore)
* apply ha una sola regola
* deepApply
  + exp (C a) (F v) e exp (C a) (N u l) non hanno overlap (terzo costruttore)
  + exp (C a) (F v) e exp (Q ul ur ll lr) (F v) non hanno overlap (secondo costruttore)
  + exp (C a) (F v) e exp (Q ul ur ll lr) (N u l) non hanno overlap (secondo costruttore)
  + exp (C a) (N u l) e exp (Q ul ur ll lr) (F v) non hanno overlap (secondo costruttore)
  + exp (C a) (N u l) e exp (Q ul ur ll lr) (N u l) non hanno overlap (secondo costruttore)
  + exp (Q ul ur ll lr) (F v) e exp (Q ul ur ll lr) (N u l) non hanno overlap (terzo costruttore)
* sumBT
  + (F a) (F b) e (F a) (N u l) non hanno overlap (secondo costruttore)
  + (F a) (F b) e (N u l) (F b) non hanno overlap (primo costruttore)
  + (F a) (F b) e (N la ra) (N lb rb) non hanno overlap (primo costruttore)
  + (F a) (N u l) e (N u l) (F b) non hanno overlap (primo costruttore)
  + (F a) (N u l) e (N la ra) (N lb rb) non hanno overlap (primo costruttore)
  + (N u l) (F b) e (N la ra) (N lb rb) non hanno overlap (secondo costruttore)
* productVec ha una sola regola
* deepProductVec
  + exp (F a) (F b) e exp (F a) (N u l) non hanno overlap (terzo costruttore)
  + exp (F a) (F b) e exp (N u l) (F b) non hanno overlap (secondo costruttore)
  + exp (F a) (F b) e exp (N la ra) (N lb rb) non hanno overlap (secondo costruttore)
  + exp (F a) (N u l) e exp (N u l) (F b) non hanno overlap (secondo costruttore)
  + exp (F a) (N u l) e exp (N la ra) (N lb rb) non hanno overlap (secondo costruttore)
  + exp (N u l) (F b) e exp (N la ra) (N lb rb) non hanno overlap (terzo costruttore)
* maybeCompressMat ha una sola regola
* maybeCompressMatRec
  + exp (Q ul ur ll lr) e exp (C x) non hanno overlap (secondo costruttore)
* mergeQT
  + (C ul) (C ur) (C ll) (C lr) e ul ur ll lr hanno overlap (C ul) (C ur) (C ll) (C lr)
* maybeCompressVec ha una sola regola
* maybeCompressVecRec
  + exp (N u l) e exp (F x) non hanno overlap (secondo costruttore)
* mergeBT ha una sola regola
  + (F u) (F l) e u l hanno overlap (F u) (F l)
* checkSize ha una sola regola