Lecture 11

- Theorems and lemmas } Correctness of mergesort
- Data types

```
fun ms(t:tree):tree =
    case t of
      Empty => Empty
    | Node(l,x,r) =>
merge(merge(ms l,
            ms r),
      Node(Empty,x,Empty))
```

A tree is sorted iff

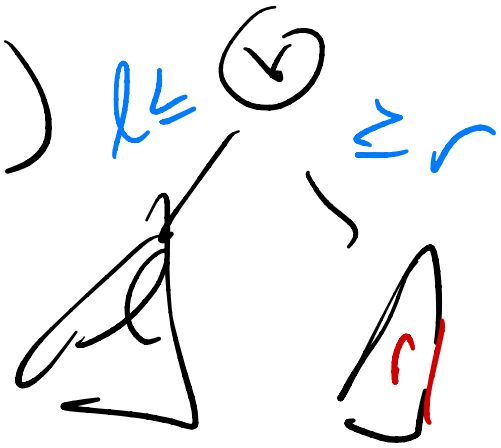
it's Empty

or it's Node(l, x, r) where l sorted

r sorted

$l \leq_t x$

$r \geq_t x$



(or e = t where t sorted.)

A tree $t \leq_t$ a number x

$\text{Empty} \leq_t x$ always

$\text{Node}(l, y, r) \leq_t x$ $\quad l \leq_t x$

$r \leq_t x$

$y \leq x$

every number in the tree is $\leq x$

Theorem for any tree t, $ms(t)$ is sorted

main result

Induction on t.

Case for Empty:

To show: $ms(\text{Empty})$ is sorted

$\mapsto$ Empty

by definition

Case for Node(l, x, r):

IH for l: ms(l) is sorted

IH for r: ms(r) is sorted

To show: ms(Node(l, x, r)) is sorted

merge(mergesl, msr)) (x) !!

By IH on l, msl
IH on r, msr are sorted, so
by Lemma 1, merge(msl, msr) is sorted

By Lemma 1 again, is sorted
and (x) !! is sorted, so merge(merge(msl, msr), (x)) is sorted

"Lemma 1" If t_1, t_2 are sorted

then merge (t_1, t_2) is sorted

thing you put along the way

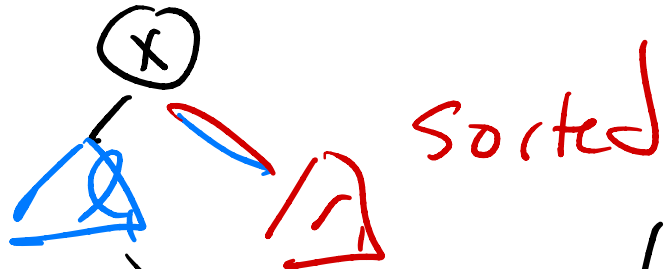
Proof by ind. on t_1.

Case for Empty: merge(Empty, t_2) is sorted

$\mapsto t_2$

and t_2 is sorted by assumption ✓

Case for Node(l_1, x_1, r_1):



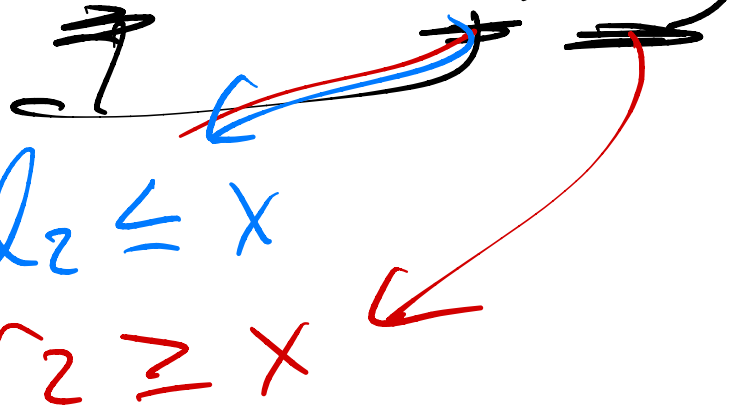
IH: for any sorted t_2, merge(l_1, t_2) is sorted

IH for any sorted t_2, merge(r_1, t_2) is sorted

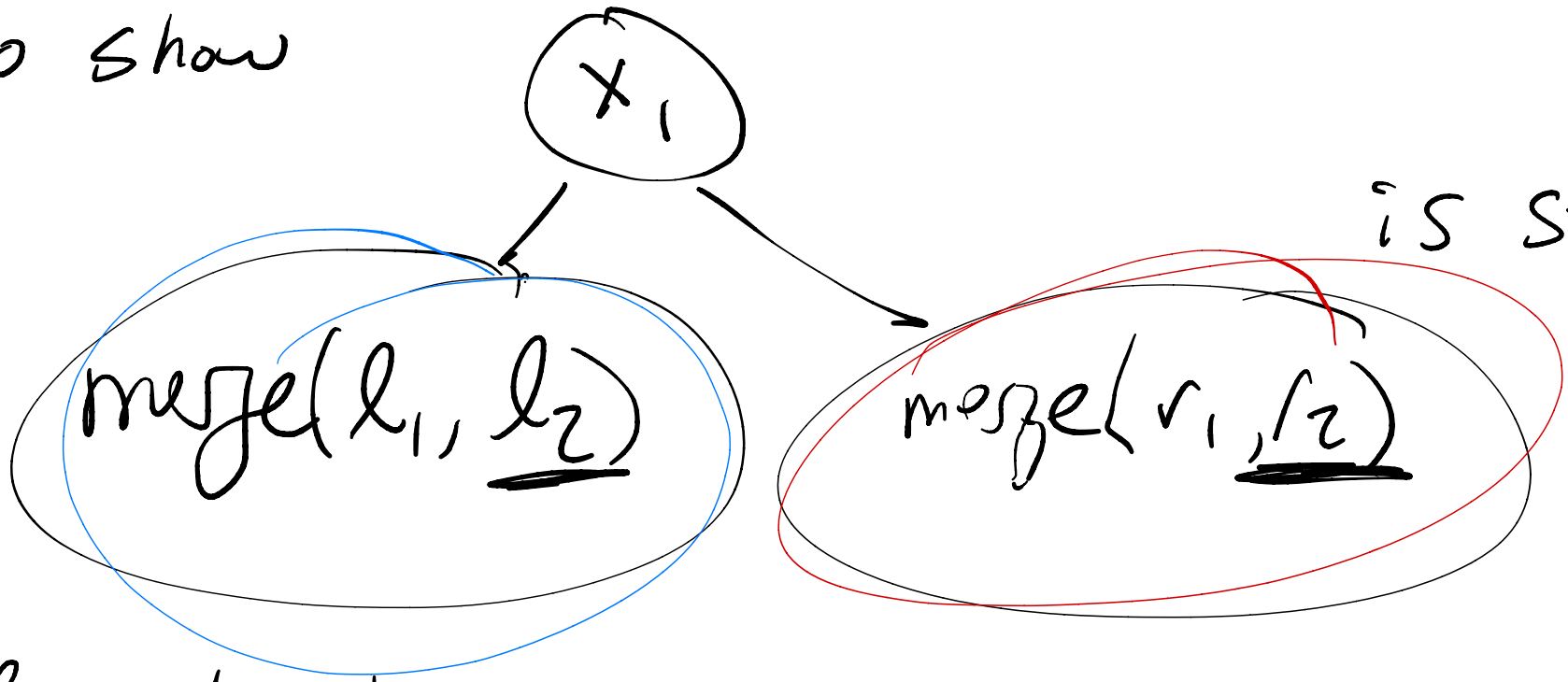
To Show: merge(Node(l_1, x_1, r_1), t_2) is sorted

$\mapsto$ Node(merge(l_1, l_2), x_1, merge(r_1, r_2))

where splitAt(t_2, x_1) = (l_2, r_2)



To show



is sorted

Suffices to show:

merge(l_1, l_2) is sorted, by IH if l_2 sorted

merge(r_1, r_2) is sorted by IH if r_2 sorted

merge(l_1, l_2) $\leq x_1$ if $l_1 \leq x_1$ because t_1 sorted and $l_2 \leq x_1$ by Lemma 5 (t_2 sorted)

merge(r_1, r_2) $\geq x_1$ if $r_1 \geq x_1$ b/c t_1 sorted and $r_2 \geq x_1$ "

```
fun merge (t1, t2) =
  case t1 of
      Empty => t2
    | Node(l1, x1, r1) => let val (l2, r2) = split#1(t2, x1)
    in
    Node( merge(l1, l2),
          x1,
          merge(r1, r2))
```

Lemma 2: If t is sorted and $splitAt(t, x) = (l, r)$ then l is sorted and r is sorted

Lemma 3: If $t_1 \leq x$ and $t_2 \leq x$ then $merge(t_1, t_2) \leq x$

Lemma 4 If $t_1 \geq x$ and $t_2 \geq x$ then $merge(t_1, t_2) \geq x$

Lemma 5 for all sorted t

if $splitAt(t, b) = (l, r)$

then $l \leq b$

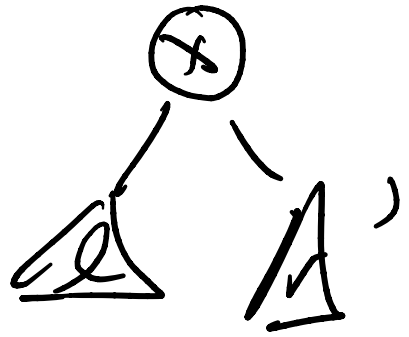
$r \geq b$

Proof by ind on t:

Case for Empty: $splitAt(Empty, b) = (Empty, Empty)$

To show: $Empty \leq b$ ✓

$Empty \geq b$

Case for 

, where $b < x$:

$splitAt(\nearrow, b) = (ll, Node(lr, x, r))$

where $(ll, lr) = splitAt(l, b)$

(sorted)

IH for l: $ll \le b$

$lr \ge b$

To show: $ll \le b$

$Node(lr, x, r) \ge b$ if $lr \ge b$, $x \ge b$, $r \ge b$ but $r \ge x \ge b$ b/c t sorted

Case $b \ge x$: - - - -

Datatypes and pattern matching

```
datatype bool = true | false

datatype list = [] | :: of (int * list)

datatype tree = Empty | Node of (tree * int * tree)
```

Constructors

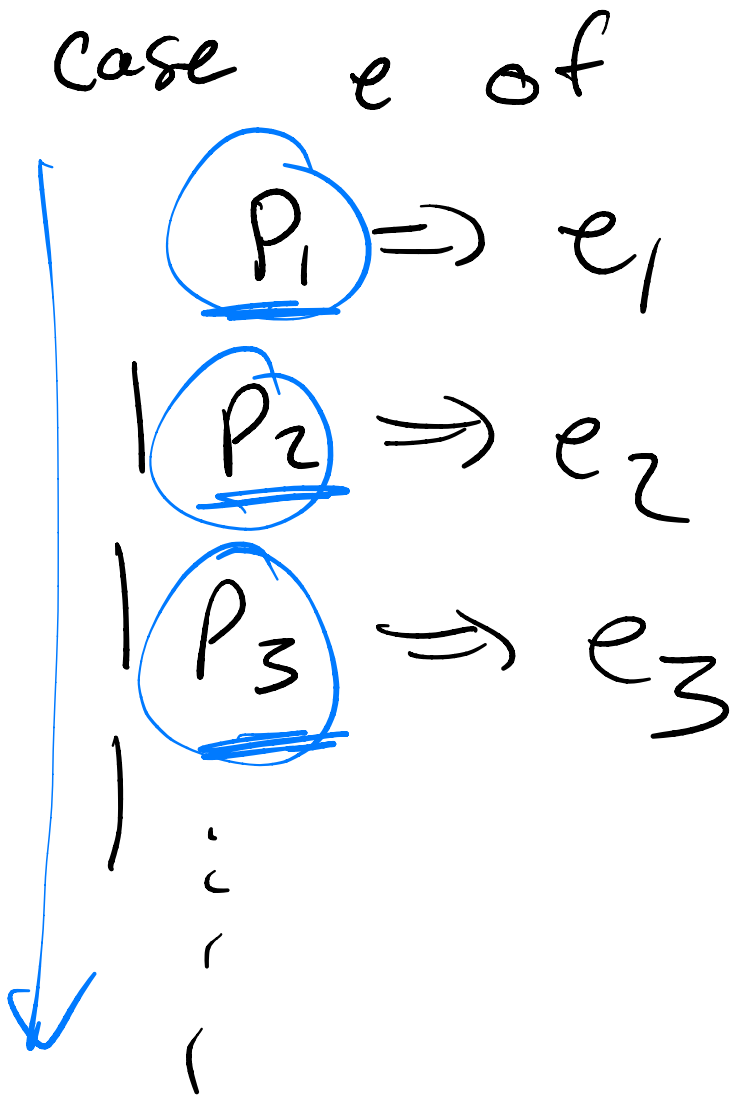
Node(Empty, 4, Empty)

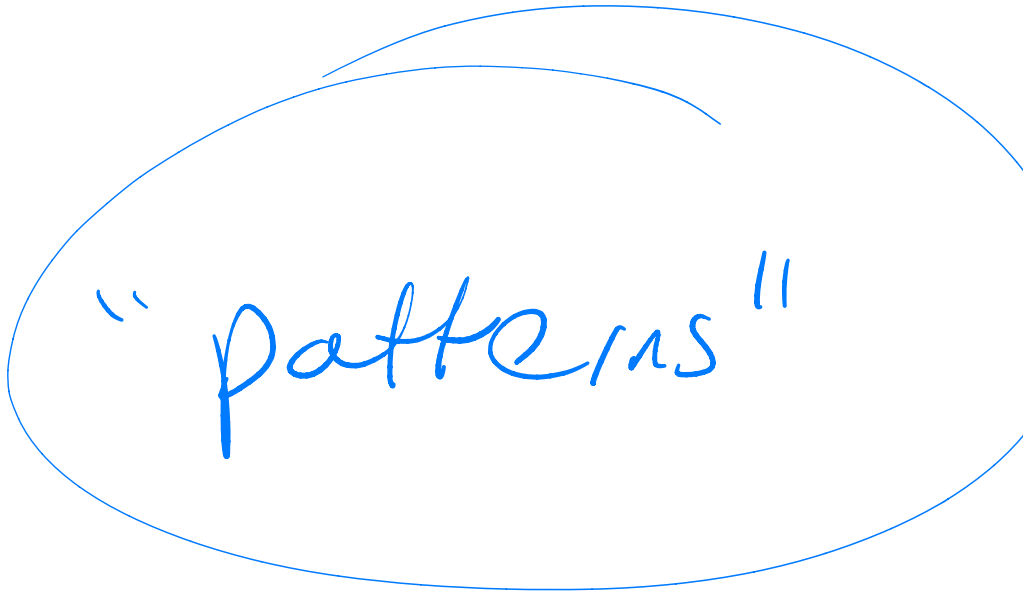
tree int tree

Constructors make values

datatype T = C_1 of T_1
| C_2 of T_2
| ⋮

Pattern matching





Patterns	Has type	Matches values	bind variables
x	any T	any	$x : T$
$\_$	any T	any	nothing
(P_1, P_2)	$T_1 * T_2$	(V_1, V_2) when P_1 matches V_1, P_2 matches V_2	what P_1 and P_2 do
$C(P)$	datatype T = $\ldots$ of T_1	$C\ V$ when P matches V	what P does

Case v of

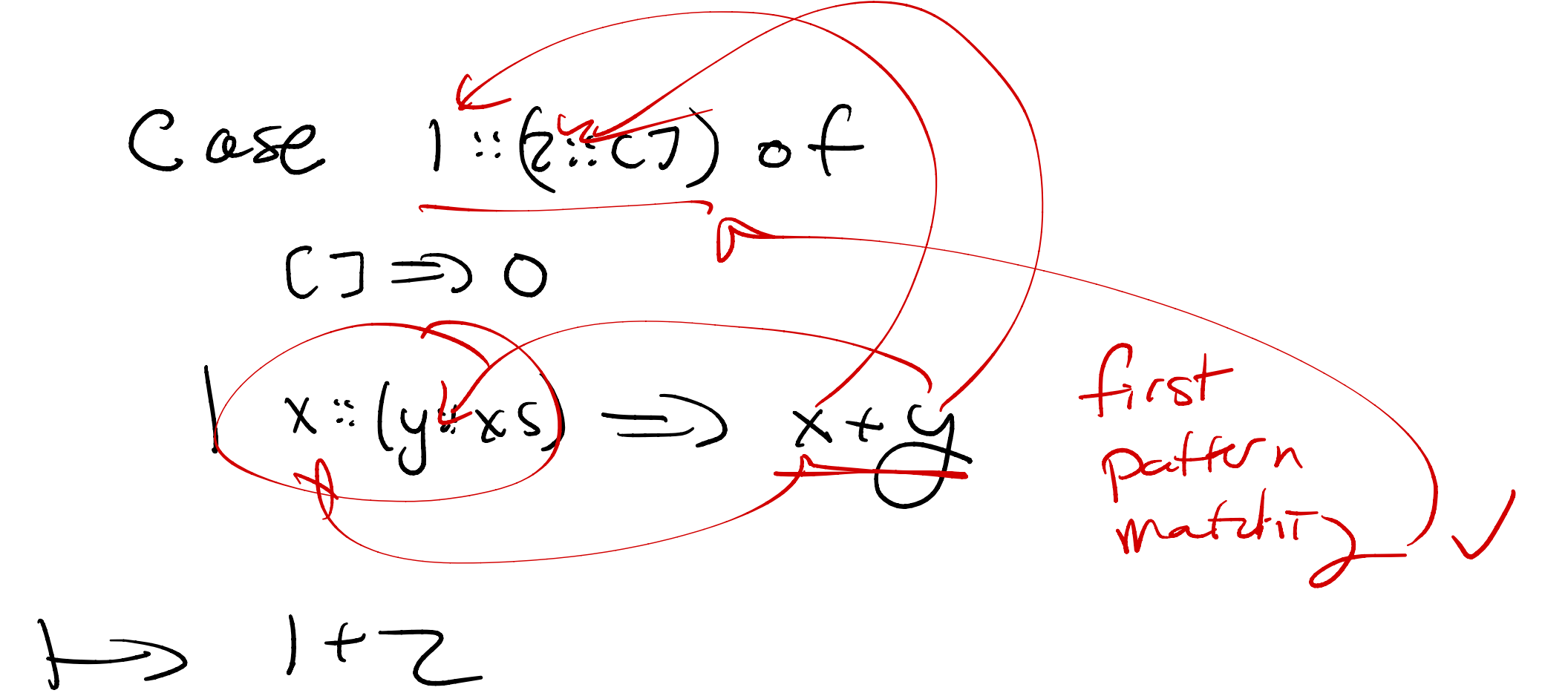
$p_1 \Rightarrow e_1$

$\mid p_2 \Rightarrow e_2$

$\mid \vdots$

find first p_i that matches v,

and then step to e_i (with substitution)


case 1 :: (2 :: []) of
[] => 0
| x :: (y :: xs) => x + y
first pattern matches
↦ 1 + 2

```
fun merge (l1, l2) =
    case l1 of
        [] => l2
      | x::xs, => case l2 of
                      [] => l1
                    | y::ys => ...
                               x::merge(xs, y::ys)
                               ...
```

```
fun merge (l1, l2) =
  case l1 of
    [] => l2
  | x::xs, => case l2 of
                [] => l1
              | y::ys => ...
                         x::merge(xs, y::ys)
                         ...
```

```
case (l1, l2) of
    ([], _) => l2
  | (_, []) => l1
  | (x::xs, y::ys) => ... x::merge(xs, y::ys) ...
```

Case e of

$P_1 \Rightarrow e_1$

$| P_2 \Rightarrow e_2$

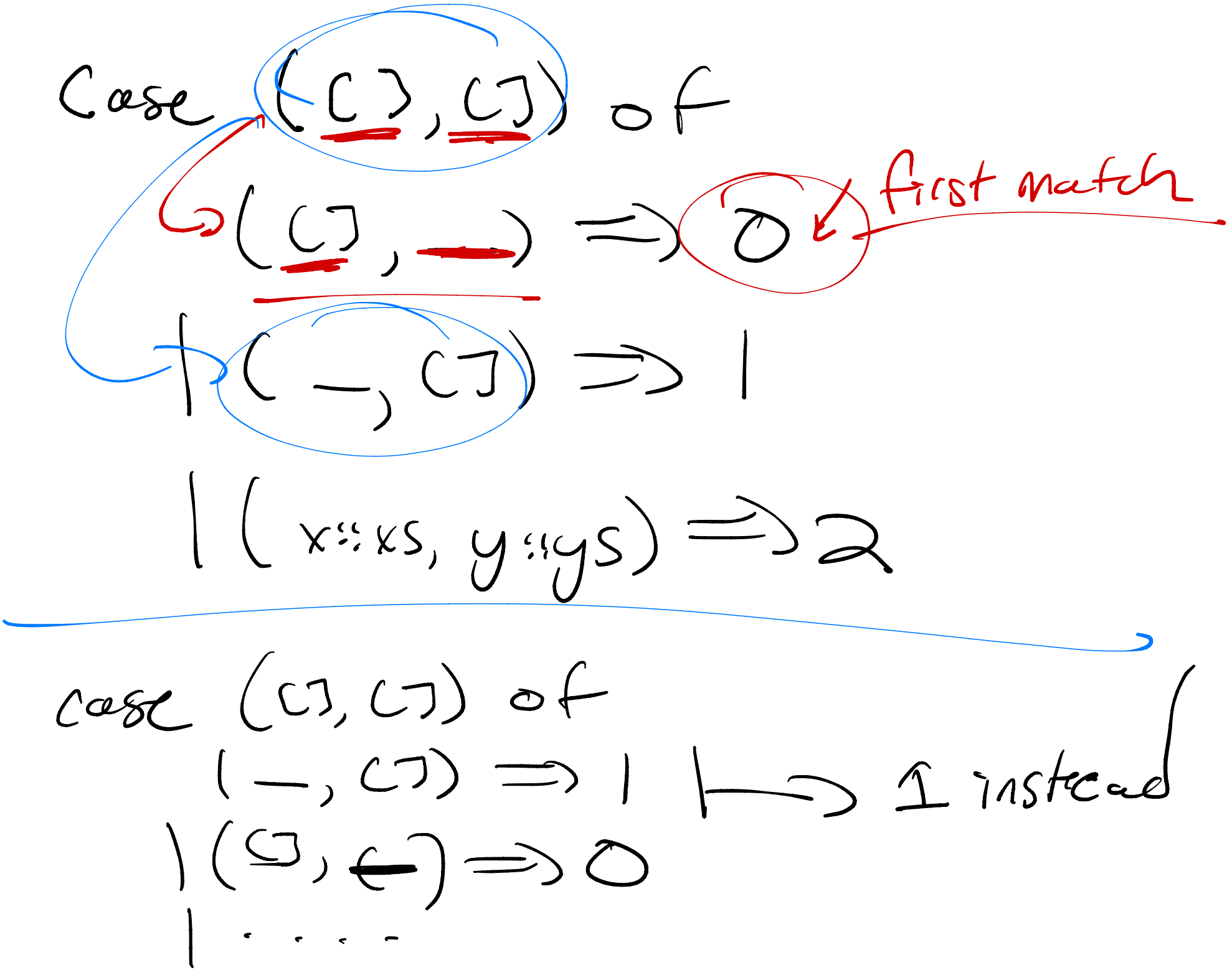
$| P_3 \Rightarrow e_3$

$| P_4 \Rightarrow e_4$

① overlap: first-match

② redundancy

③ exhaustiveness


case (x, y) of
(x, _) => 0
first match
| (_, y) => 1
| (x::xs, y::ys) => 2
case (x, y) of
(_, y) => 1
1 instead
| (x, _) => 0
| ...

```
case e of
  ([], _)     => 0
| (_, [])     => 1
| ([], [])    => 2
| (x::xs, y::ys) => 3
```

redundant

```
case e of
  ([], []) => 2
  ([], _)  => 0
  (_, [])  => 1
  (x::xs, y::ys) =>
```