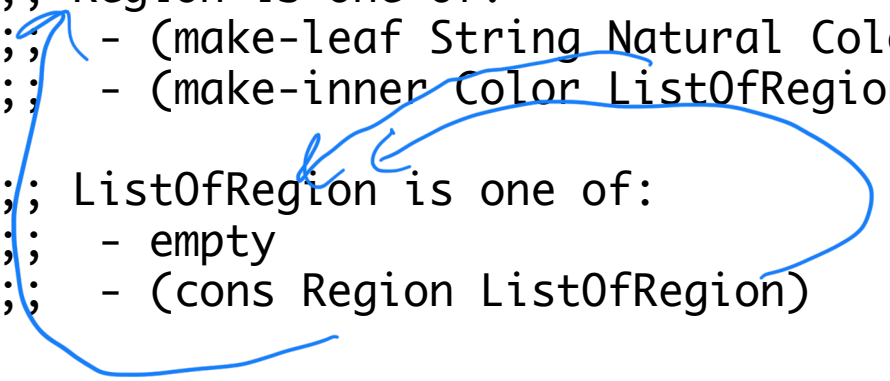
Today

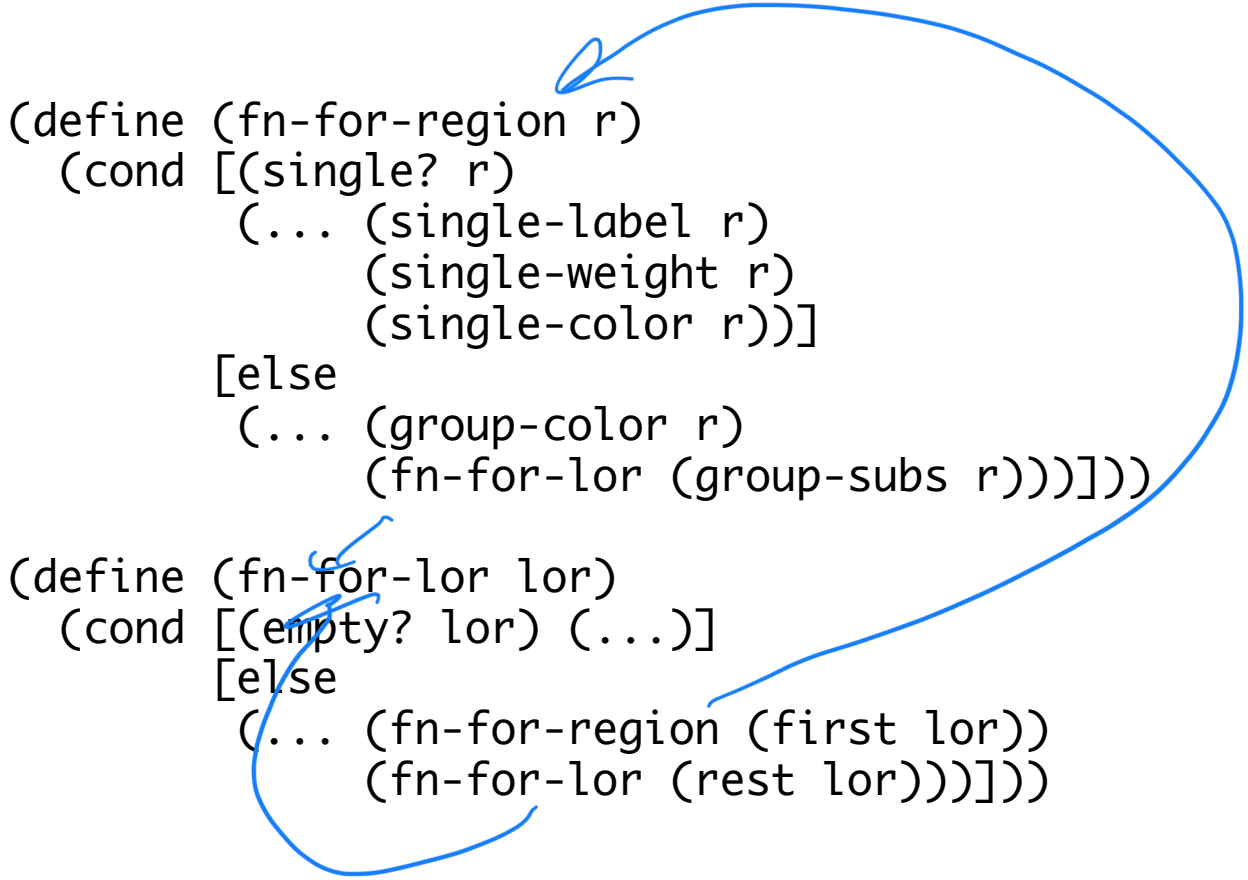
- 2 one-of templating
  - orthogonal axes of variation
  - reasoning about and simplifying code at the model level

```
(define-struct leaf (label weight color))
(define-struct inner (color subs))
;; Region is one of:
;;   - (make-leaf String Natural Color)
;;   - (make-inner Color ListOfRegion)

;; ListOfRegion is one of:
;;   - empty
;;   - (cons Region ListOfRegion)
```



- And we know the type comments completely determine the template
  - as such they are a model of the template
  - less detailed, but they say enough to understand much about the template



```
(define (fn-for-region r)
  (cond [(single? r)
         (... (single-label r)
              (single-weight r)
              (single-color r))]
        [else
         (... (group-color r)
              (fn-for-lor (group-subs r)))]))

(define (fn-for-lor lor)
  (cond [(empty? lor) (...)]
        [else
         (... (fn-for-region (first lor))
              (fn-for-lor (rest lor)))]))
```

all-labels

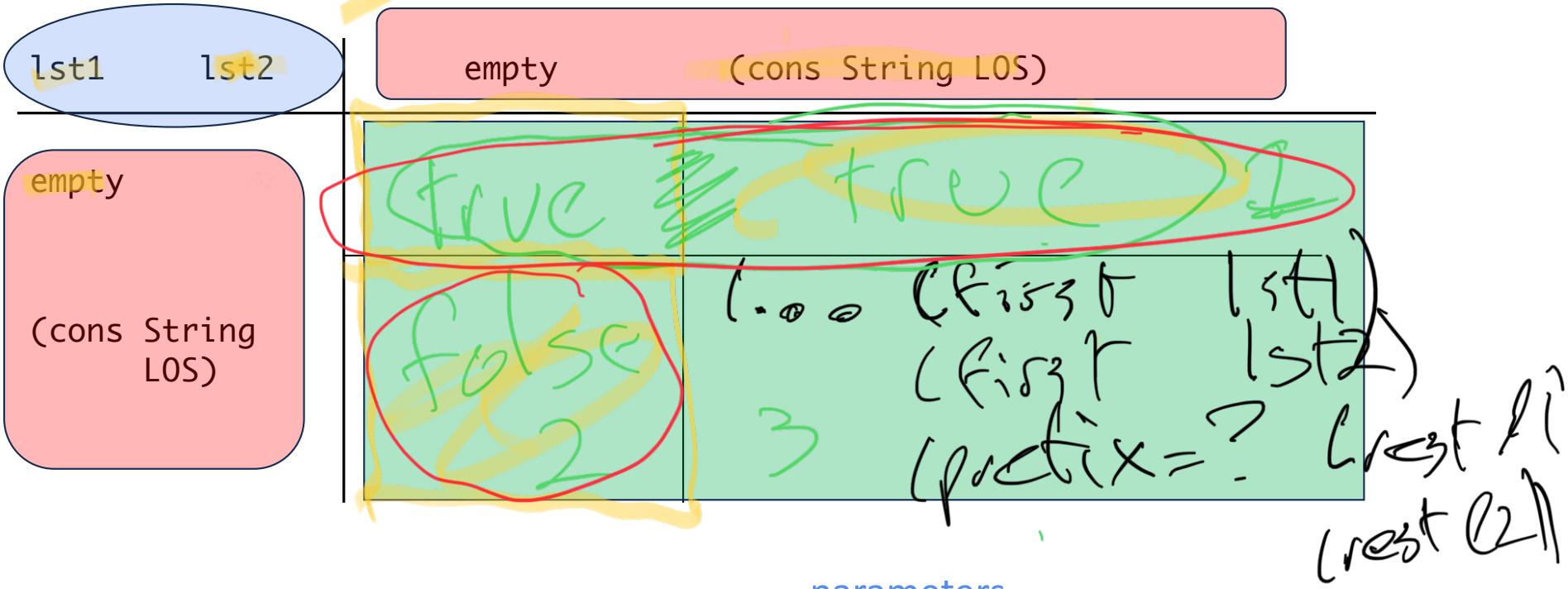
```
(define-struct terminal (label weight color))
(define-struct group (color subs))
;; Region is one of:
;;  - (make-terminal   (list String) Natural Color)
;;  - (make-group Color  (NMR ListOfRegion)  )

;; ListOfRegion is one of:
;;  - empty
;;  - (cons (append (NMR Region)
;;                  (NR ListOfRegion)))
```

- Working with models
  - is one of the most important ideas in science and engineering
  - helps control complexity
    - by making it possible to reason in terms of simpler description
    - How many NR are there? Is there MR? How many?...
  - today:
    can manipulate design at model level
    to manipulate the actual resulting code

Cross product of type comments

prefix=?



parameters

type expressions (bits of type comments)

expressions (code)

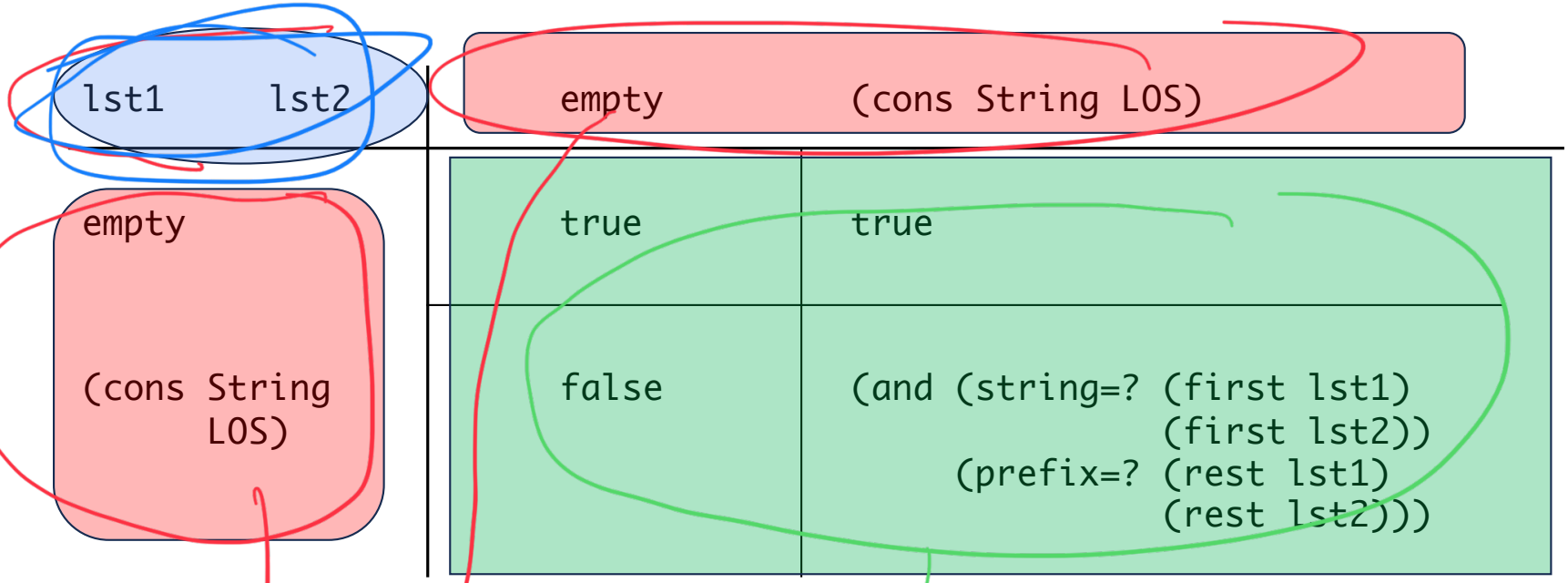
prefix=?

lst1 lst2	empty	(cons String LOS)
empty		
(cons String LOS)		

prefix=?

lst1 lst2	empty	(cons String LOS)
empty	true [1]	true [1]
(cons String LOS)	false [2]	(and (string=? (first lst1) (first lst2)) (prefix=? (rest lst1) (rest lst2))) [3]

prefix=?



parameters

type expressions (bits of type comments)

expressions (code)

questions

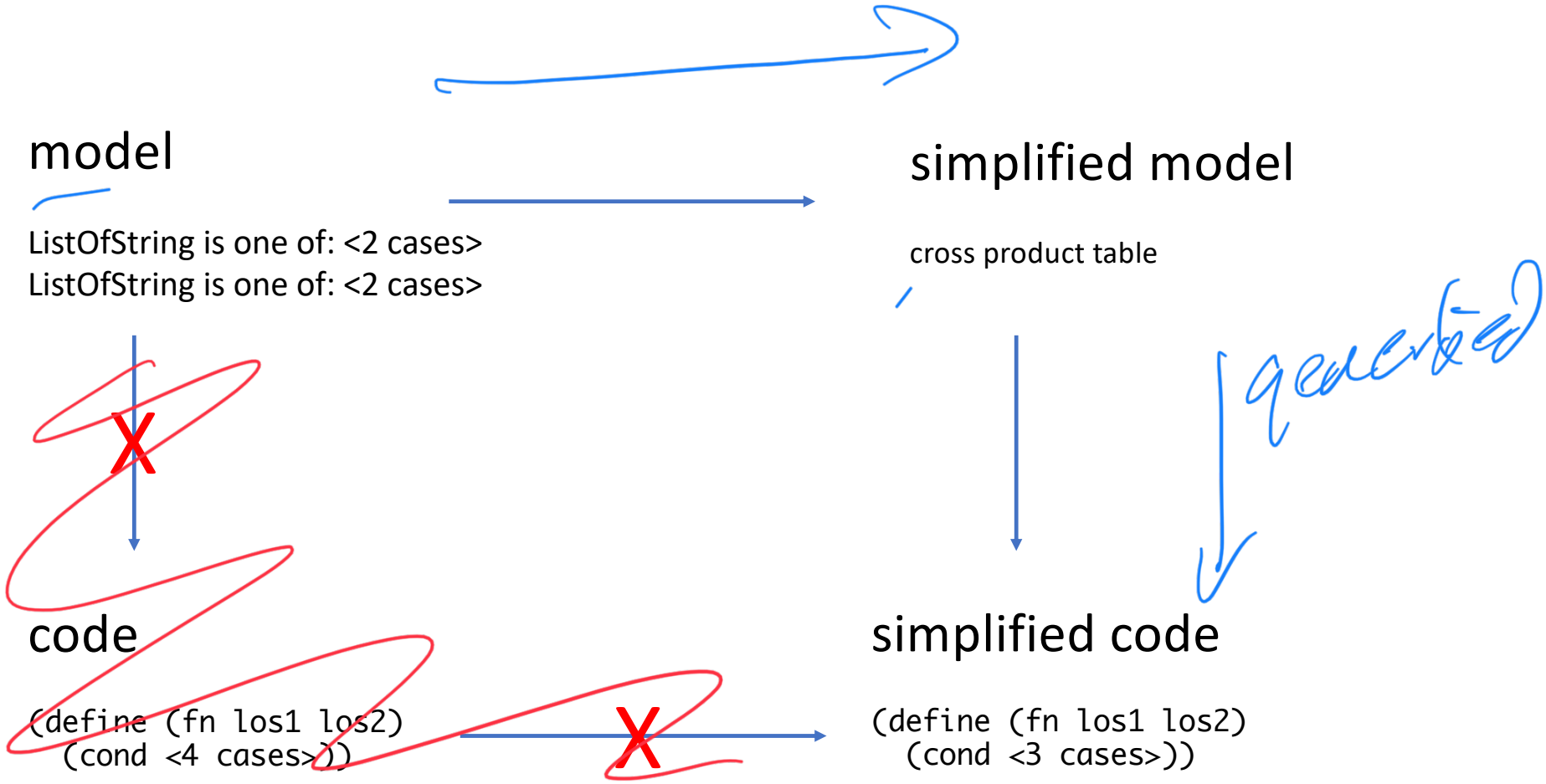
answers

model

ListOfString is one of: <2 cases>
ListOfString is one of: <2 cases>

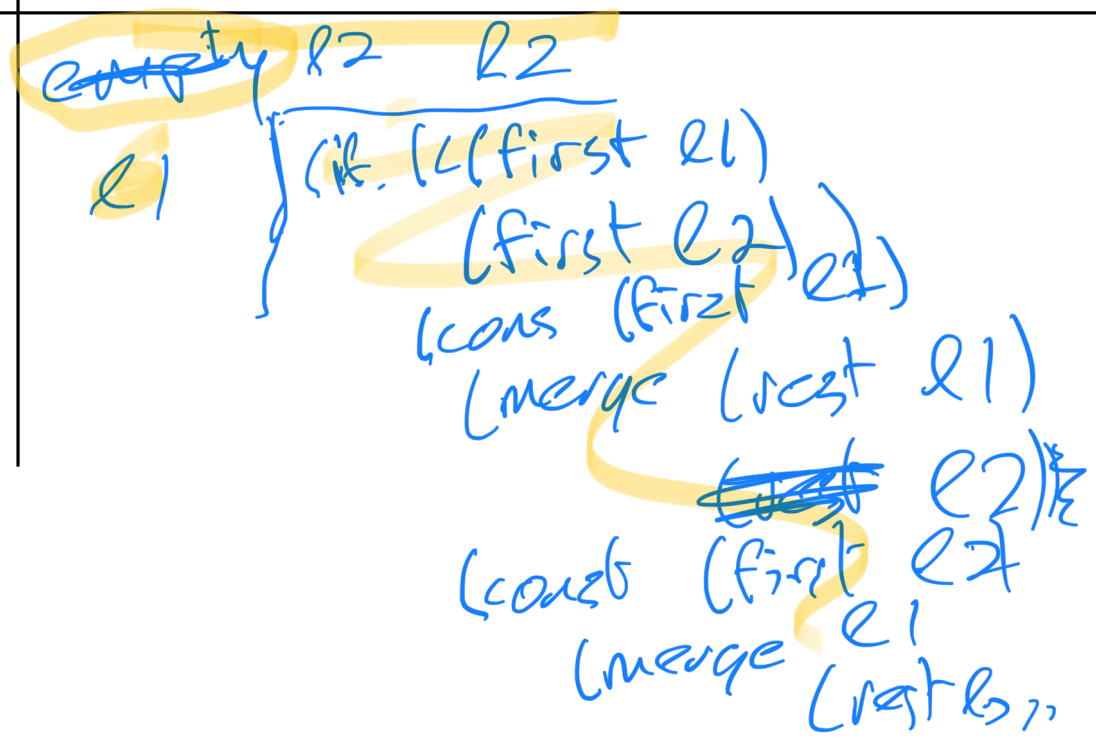
code

```
(define (fn los1 los2)
  (cond <4 cases>))
```


model
ListOfString is one of: <2 cases>
ListOfString is one of: <2 cases>
simplified model
cross product table
generated
code
(define (fn los1 los2)
(cond <4 cases>))
simplified code
(define (fn los1 los2)
(cond <3 cases>))

merge

l2 l1	empty	(cons N LON)
empty		
(cons N LON)		



merge

l2 l1	empty	(cons N LON)
empty	empty	l2

(cons N LON)	l1	(if (< (first l1) (first l2))
		    (cons (first l1)
		          (merge (rest l1) l2))
		    (cons (first l2)
		          (merge l1 (rest l2))))

merge

l2 l1	empty		(cons N LON)	
empty	l2	[1]	l2	[1]
(cons N LON)	l1	[2]	(if (< (first l1) (first l2)) (cons (first l1) (merge (rest l1) l2)) (cons (first l2) (merge l1 (rest l2))))	[3]

has-path?

bt p	false	(make-node Nat Str BT BT)
empty	false	true
(cons "L" Path)		(first p) (has? (rest p) (node-l t))
(cons "R" Path)		(has? (rest p) (node-r t))

has-path?

bt p	false	(make-node Nat Str BT BT)
empty	false [1]	true [2]
(cons "L" Path)	false [1]	(has-path? (node-l bt) (rest p)) [3]
(cons "R" Path)	false [1]	(has-path? (node-r bt) (rest p)) [4]

11 false