Optimization Coaching

Vincent St-Amour
Sam Tobin-Hochstadt
Matthias Felleisen

PLT

Harvard - October 31st, 2012

```
#lang typed/racket/base
(require racket/match racket/math racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         "flonum.rkt"
         "flomap-struct.rkt")
(provide flomap-flip-horizontal flomap-flip-vertical flomap-transpose
         flomap-cw-rotate flomap-ccw-rotate
         (struct-out invertible-2d-function) Flomap-Transform
         transform-compose rotate-transform whirl-and-pinch-transform
         flomap-transform)
(: flomap-flip-horizontal (flomap -> flomap))
(define (flomap-flip-horizontal fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- w-1 x) y)))))
(define (flomap-flip-vertical fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k x (fx- h-1 y))))))
(define (flomap-transpose fm)
  (match-define (flomap vs c w h) fm)
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y x)))))
(define (flomap-cw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- h-1 y) x)))))
(define (flomap-ccw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y (fx- w-1 x))))))
(struct: invertible-2d-function ([f : (Flonum Flonum -> (values Flonum Flonum))]
                                 [g : (Flonum Flonum -> (values Flonum Flonum))]))
(define-type Flomap-Transform (Integer Integer -> invertible-2d-function))
(: transform-compose (Flomap-Transform Flomap-Transform -> Flomap-Transform))
(define ((transform-compose t1 t2) w h)
  (match-define (invertible-2d-function f1 g1) (t1 w h))
  (match-define (invertible-2d-function f2 g2) (t2 w h))
  (invertible-2d-function (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (f2 x y)])
                              (f1 x y)))
                          (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (g1 x y)])
                              (g2 x y)))))
(: flomap-transform (case-> (flomap Flomap-Transform -> flomap)
                            (flomap Flomap-Transform Real Real Real Real -> flomap)))
(define flomap-transform
  (case-lambda
    [(fm t)
     (match-define (flomap vs c w h) fm)
     (match-define (invertible-2d-function f g) (t w h))
     (define x-min +inf.0)
     (define x-max -inf.0)
     (define y-min +inf.0)
     (define y-max -inf.0)
     (let: y-loop : Void ([y : Integer  0])
       (when (y . fx< . h)
         (let: x-loop : Void ([x : Integer  0])
           (cond [(x . fx< . w)
                  (define-values (new-x new-y) (f (+ 0.5 (fx->fl x)) (+ 0.5 (fx->fl y))))
                  (when (new-x . < . x-min) (set! x-min new-x))
                  (when (new-x . > . x-max) (set! x-max new-x))
                  (when (new-y . < . y-min) (set! y-min new-y))
                  (when (new-y . > . y-max) (set! y-max new-y))
                  (x-loop (fx+ x 1))]
                 [else
                  (y-loop (fx+ y 1))])))))
     (flomap-transform fm t x-min x-max y-min y-max)]
    [(fm t x-min x-max y-min y-max)
     (let ([x-min  (real->double-flonum x-min)]
           [x-max  (real->double-flonum x-max)]
           [y-min  (real->double-flonum y-min)]
           [y-max  (real->double-flonum y-max)])
       (match-define (flomap vs c w h) fm)
       (match-define (invertible-2d-function f g) (t w h))
       (define int-x-min (fl->fx (floor x-min)))
       (define int-x-max (fl->fx (ceiling x-max)))
       (define int-y-min (fl->fx (floor y-min)))
       (define int-y-max (fl->fx (ceiling y-max)))
       (define new-w (- int-x-max int-x-min))
       (define new-h (- int-y-max int-y-min))
       (define x-offset (+ 0.5 (fx->fl int-x-min)))
       (define y-offset (+ 0.5 (fx->fl int-y-min)))
       (inline-build-flomap
        c new-w new-h
        (λ (k x y i)
          (define-values (old-x old-y) (g (+ (fx->fl x) x-offset)
                                          (+ (fx->fl y) y-offset)))
          (flomap-bilinear-ref fm k old-x old-y)))))]))
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         racket/match racket/math
         "flonum.rkt"
         "flomap-struct.rkt"
         "flomap-stats.rkt")
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
         fmneg fmabs fmsqr fmsin fmcos fmtan fmlog fmexp fmsqrt fmasin fmacos fmatan
         fmround fmfloor fmceiling fmtruncate fmzero
         fm+ fm- fm* fm/ fmmin fmmax
         flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
; ============================================================================================
; Unary
(: flomap-lift-helper : (Float -> Float) -> (flomap -> flomap))
(define (flomap-lift-helper f)
  (λ: ([fm : flomap])
    (match-define (flomap vs c w h) fm)
    (flomap (inline-build-flvector (* c w h) (λ (i) (f (unsafe-flvector-ref vs i))))
            c w h)))
(: flomap-lift ((Flonum -> Real) -> (flomap -> flomap)))
(define (flomap-lift op)
  (flomap-lift-helper (λ (x) (real->double-flonum (op x)))))
(define fmneg (flomap-lift-helper -))
(define fmabs (flomap-lift-helper abs))
(define fmsqr (flomap-lift-helper sqr))
(define fmsin (flomap-lift-helper sin))
(define fmcos (flomap-lift-helper cos))
(define fmtan (flomap-lift-helper tan))
(define fmlog (flomap-lift-helper fllog))
(define fmexp (flomap-lift-helper exp))
(define fmsqrt (flomap-lift-helper flsqrt))
(define fmasin (flomap-lift-helper asin))
(define fmacos (flomap-lift-helper acos))
(define fmatan (flomap-lift-helper atan))
(define fmround (flomap-lift-helper round))
(define fmfloor (flomap-lift-helper floor))
(define fmceiling (flomap-lift-helper ceiling))
(define fmtruncate (flomap-lift-helper truncate))
(define fmzero (flomap-lift-helper (λ (x) (if (x . = . 0.0) 1.0 0.0))))
; ;; ============================================================================================
; Binary
(: flomap-lift-helper2 : Symbol (Float Float -> Float) -> ((U Real flomap) (U Real flomap) -> flomap))
(define (flomap-lift-helper2 name f)
  (let: ()
    (λ: ([fm1 : (U Real flomap)] [fm2 : (U Real flomap)])
      (cond
        [(and (real? fm1) (real? fm2))
         (error name "expected at least one flomap argument; given ~e and ~e" fm1 fm2)]
        [(real? fm1)  (let ([fm1  (real->double-flonum fm1)])
                        ((flomap-lift-helper (λ (v) (f fm1 v))) fm2))]
        [(real? fm2)  (let ([fm2  (real->double-flonum fm2)])
                        ((flomap-lift-helper (λ (v) (f v fm2))) fm1))]
        [else
         (match-define (flomap vs1 c1 w h) fm1)
         (match-define (flomap vs2 c2 w2 h2) fm2)
         (cond
           [(not (and (= w w2) (= h h2)))
            (error name "expected same-size flomaps; given sizes ~ex~e and ~ex~e" w h w2 h2)]
           [(= c1 c2)  (define n (* c1 w h))
                       (define res-vs (make-flvector n))
                       (flomap (inline-build-flvector n (λ (i) (f (unsafe-flvector-ref vs1 i)
                                                                  (unsafe-flvector-ref vs2 i))))
                               c1 w h)]
           [(= c1 1)  (inline-build-flomap
                       c2 w h
                       (λ (k x y i) (f (unsafe-flvector-ref vs1 (coords->index 1 w 0 x y))
                                       (unsafe-flvector-ref vs2 i))))]
           [(= c2 1)  (inline-build-flomap
                       c1 w h
                       (λ (k x y i) (f (unsafe-flvector-ref vs1 i)
                                       (unsafe-flvector-ref vs2 (coords->index 1 w 0 x y)))))]
           [else
            (error name (string-append "expected flomaps with the same number of components, "
                                       "or a flomap with 1 component and any same-size flomap; "
                                       "given flomaps with ~e and ~e components")
                   c1 c2)])])))))
(: flomap-lift2 (Symbol (Flonum Flonum -> Real) -> ((U Real flomap) (U Real flomap) -> flomap)))
(define (flomap-lift2 name f)
  (flomap-lift-helper2 name (λ (x y) (real->double-flonum (f x y)))))
(define fm+ (flomap-lift-helper2 'fm+ +))
(define fm- (flomap-lift-helper2 'fm- -))
(define fm* (flomap-lift-helper2 'fm* *))
(define fm/ (flomap-lift-helper2 'fm/ /))
(define fmmin (flomap-lift-helper2 'fmmin min))
(define fmmax (flomap-lift-helper2 'fmmax max))
(: flomap-normalize (flomap -> flomap))
(define (flomap-normalize fm)
  (define-values (v-min v-max) (flomap-extreme-values fm))
  (define v-size (- v-max v-min))
  (let* ([fm  (fm- fm v-min)]
         [fm  (if (v-size . = . 0.0) fm (fm/ fm v-size))])
    fm))
(define fmdiv/zero
  (flomap-lift-helper2 'fmdiv/zero (λ (x y) (if (y . = . 0.0) 0.0 (/ x y)))))
(: flomap-divide-alpha (flomap -> flomap))
(define (flomap-divide-alpha fm)
  (match-define (flomap _ c w h) fm)
  (cond [(c . <= . 1)  fm]
        [else
         (define alpha-fm (flomap-ref-component fm 0))
         (flomap-append-components alpha-fm (fmdiv/zero (flomap-drop-components fm 1) alpha-fm))]))
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fx->fl fl->fx)
         racket/match racket/math
         "flonum.rkt"
         "flomap.rkt")
(provide deep-flomap deep-flomap? deep-flomap-argb deep-flomap-z
         deep-flomap-width deep-flomap-height deep-flomap-z-min deep-flomap-z-max
         deep-flomap-size deep-flomap-alpha deep-flomap-rgb
         flomap->deep-flomap
         ; Sizing
         deep-flomap-inset deep-flomap-trim deep-flomap-scale deep-flomap-resize
         ; Z-adjusting
         deep-flomap-scale-z deep-flomap-smooth-z deep-flomap-raise deep-flomap-tilt
         deep-flomap-emboss
         deep-flomap-bulge deep-flomap-bulge-round deep-flomap-bulge-round-rect
         deep-flomap-bulge-spheroid deep-flomap-bulge-horizontal deep-flomap-bulge-vertical
         deep-flomap-bulge-ripple
         ; Compositing
         deep-flomap-pin deep-flomap-pin*
         deep-flomap-lt-superimpose deep-flomap-lc-superimpose deep-flomap-lb-superimpose
         deep-flomap-ct-superimpose deep-flomap-cc-superimpose deep-flomap-cb-superimpose
         deep-flomap-rt-superimpose deep-flomap-rc-superimpose deep-flomap-rb-superimpose
         deep-flomap-vl-append deep-flomap-vc-append deep-flomap-vr-append
         deep-flomap-ht-append deep-flomap-hc-append deep-flomap-hb-append)
(struct: deep-flomap ([argb : flomap] [z : flomap])
  #:transparent
  #:guard
  (λ (argb-fm z-fm name)
    (match-define (flomap _ 4 w h) argb-fm)
    (match-define (flomap _ 1 zw zh) z-fm)
    (unless (and (= w zw) (= h zh))
      (error 'deep-flomap
             "expected flomaps of equal dimension; given dimensions ~ex~e and ~ex~e" w h zw zh))
    (values argb-fm z-fm)))
(: flomap->deep-flomap (flomap -> deep-flomap))
(define (flomap->deep-flomap argb-fm)
  (match-define (flomap _ 4 w h) argb-fm)
  (deep-flomap argb-fm (make-flomap 1 w h)))
(: deep-flomap-width (deep-flomap -> Nonnegative-Fixnum))
(define (deep-flomap-width dfm)
  (define w (flomap-width (deep-flomap-argb dfm)))
  (with-asserts ([w  nonnegative-fixnum?])
    w))
(: deep-flomap-height (deep-flomap -> Nonnegative-Fixnum))
(define (deep-flomap-height dfm)
  (define h (flomap-height (deep-flomap-argb dfm)))
  (with-asserts ([h  nonnegative-fixnum?])
    h))
(: deep-flomap-z-min (deep-flomap -> Flonum))
(define (deep-flomap-z-min dfm)
  (flomap-min-value (deep-flomap-z dfm)))
(: deep-flomap-z-max (deep-flomap -> Flonum))
(define (deep-flomap-z-max dfm)
  (flomap-max-value (deep-flomap-z dfm)))
(: deep-flomap-size (deep-flomap -> (values Nonnegative-Fixnum Nonnegative-Fixnum)))
(define (deep-flomap-size dfm)
  (values (deep-flomap-width dfm) (deep-flomap-height dfm)))
(: deep-flomap-alpha (deep-flomap -> flomap))
(define (deep-flomap-alpha dfm)
  (flomap-ref-component (deep-flomap-argb dfm) 0))
(: deep-flomap-rgb (deep-flomap -> flomap))
(define (deep-flomap-rgb dfm)
  (flomap-drop-components (deep-flomap-argb dfm) 1))
; ============================================================================================
; Z adjusters
(: deep-flomap-scale-z (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-scale-z dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (deep-flomap argb-fm (fm* z-fm z)))
(: deep-flomap-smooth-z (deep-flomap Real -> deep-flomap))
(define (deep-flomap-smooth-z dfm σ)
  (let ([σ  (exact->inexact σ)])
    (match-define (deep-flomap argb-fm z-fm) dfm)
    (define new-z-fm (flomap-blur z-fm σ))
    (deep-flomap argb-fm new-z-fm)))
; deep-flomap-raise and everything derived from it observe an invariant:
; when z is added, added z must be 0.0 everywhere alpha is 0.0
(: deep-flomap-raise (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-raise dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (define alpha-fm (deep-flomap-alpha dfm))
  (deep-flomap argb-fm (fm+ z-fm (fm* alpha-fm z))))
(: deep-flomap-emboss (deep-flomap Real (U Real flomap) -> deep-flomap))
(define (deep-flomap-emboss dfm xy-amt z-amt)
  (let ([σ       (/ xy-amt 3.0)])
    (define z-fm (flomap-normalize (deep-flomap-alpha dfm)))
    (define new-z-fm (fm* (flomap-blur z-fm σ) z-amt))
    (deep-flomap-raise dfm new-z-fm)))
(: deep-flomap-bulge-helper (deep-flomap (Flonum Flonum -> Flonum) -> deep-flomap))
(define (deep-flomap-bulge-helper dfm f)
  (let ()
    (define-values (w h) (deep-flomap-size dfm))
    (define half-x-size (- (* 0.5 (fx->fl w)) 0.5))
    (define half-y-size (- (* 0.5 (fx->fl h)) 0.5))
    (define z-fm
      (inline-build-flomap
       1 w h
       (λ (_ x y i)
         (f (- (/ (fx->fl x) half-x-size) 1.0)
            (- (/ (fx->fl y) half-y-size) 1.0)))))
    (deep-flomap-raise dfm z-fm)))
(: deep-flomap-bulge (deep-flomap (Flonum Flonum -> Real) -> deep-flomap))
(define (deep-flomap-bulge dfm f)
  (deep-flomap-bulge-helper dfm (λ (cx cy) (real->double-flonum (f cx cy)))))
```

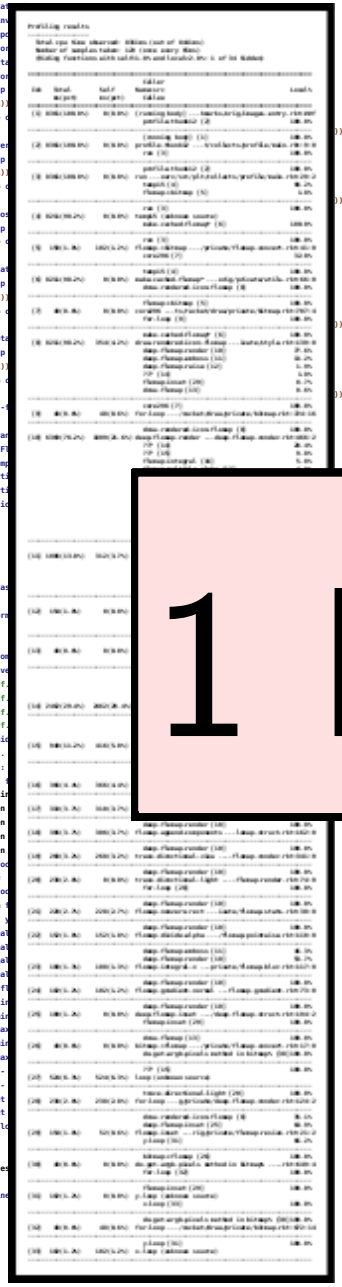

It renders text, too!

```
#lang typed/racket/base
(require racket/match racket/math racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         "flonum.rkt"
         "flomap-struct.rkt")
(provide flomap-flip-horizontal flomap-flip-vertical flomap-transpose
         flomap-cw-rotate flomap-ccw-rotate
         (struct-out invertible-2d-function) Flomap-Transform
         transform-compose rotate-transform whirl-and-pinch-transform
         flomap-transform)
(: flomap-flip-horizontal (flomap -> flomap))
(define (flomap-flip-horizontal fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- w-1 x) y)))))
(define (flomap-flip-vertical fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k x (fx- h-1 y))))))
(define (flomap-transpose fm)
  (match-define (flomap vs c w h) fm)
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y x)))))
(define (flomap-cw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- h-1 y) x)))))
(define (flomap-ccw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y (fx- w-1 x))))))
(struct: invertible-2d-function ([f : (Flonum Flonum -> (values Flonum Flonum))]
                                 [g : (Flonum Flonum -> (values Flonum Flonum))]))
(define-type Flomap-Transform (Integer Integer -> invertible-2d-function))
(: transform-compose (Flomap-Transform Flomap-Transform -> Flomap-Transform))
(define ((transform-compose t1 t2) w h)
  (match-define (invertible-2d-function f1 g1) (t1 w h))
  (match-define (invertible-2d-function f2 g2) (t2 w h))
  (invertible-2d-function (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (f2 x y)])
                              (f1 x y)))
                          (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (g1 x y)])
                              (g2 x y)))))
(: flomap-transform (case-> (flomap Flomap-Transform -> flomap)
                            (flomap Flomap-Transform Real Real Real Real -> flomap
(define flomap-transform
  (case-lambda
    [(fm t)
     (match-define (flomap vs c w h) fm)
     (match-define (invertible-2d-function f g) (t w h))
     (define x-min +inf.0)
     (define x-max -inf.0)
     (define y-min +inf.0)
     (define y-max -inf.0)
     (let: y-loop : Void ([y : Integer  0])
       (when (y . fx< . h)
         (let: x-loop : Void ([x : Integer  0])
           (cond [(x . fx< . w)
                  (define-values (new-x new-y) (f (+ 0.5 (fx->fl x)) (+ 0.5 (fx->f
                  (when (new-x . < . x-min) (set! x-min new-x))
                  (when (new-x . > . x-max) (set! x-max new-x))
                  (when (new-y . < . y-min) (set! y-min new-y))
                  (when (new-y . > . y-max) (set! y-max new-y))
                  (x-loop (fx+ x 1))]
                 [else
                  (y-loop (fx+ y 1))])))))
     (flomap-transform fm t x-min x-max y-min y-max)]
    [(fm t x-min x-max y-min y-max)
     (let ([x-min  (real->double-flonum x-min)]
           [x-max  (real->double-flonum x-max)]
           [y-min  (real->double-flonum y-min)]
           [y-max  (real->double-flonum y-max)])
       (match-define (flomap vs c w h) fm)
       (match-define (invertible-2d-function f g) (t w h))
       (define int-x-min (fl->fx (floor x-min)))
       (define int-x-max (fl->fx (ceiling x-max)))
       (define int-y-min (fl->fx (floor y-min)))
       (define int-y-max (fl->fx (ceiling y-max)))
       (define new-w (- int-x-max int-x-min))
       (define new-h (- int-y-max int-y-min))
       (define x-offset (+ 0.5 (fx->fl int-x-min)))
       (define y-offset (+ 0.5 (fx->fl int-y-min)))
       (inline-build-flomap
        c new-w new-h
        (λ (k x y i)
          (define-values (old-x old-y) (g (+ (fx->fl x) x-offset)
                                          (+ (fx->fl y) y-offset)))
          (flomap-bilinear-ref fm k old-x old-y)))))]))
```

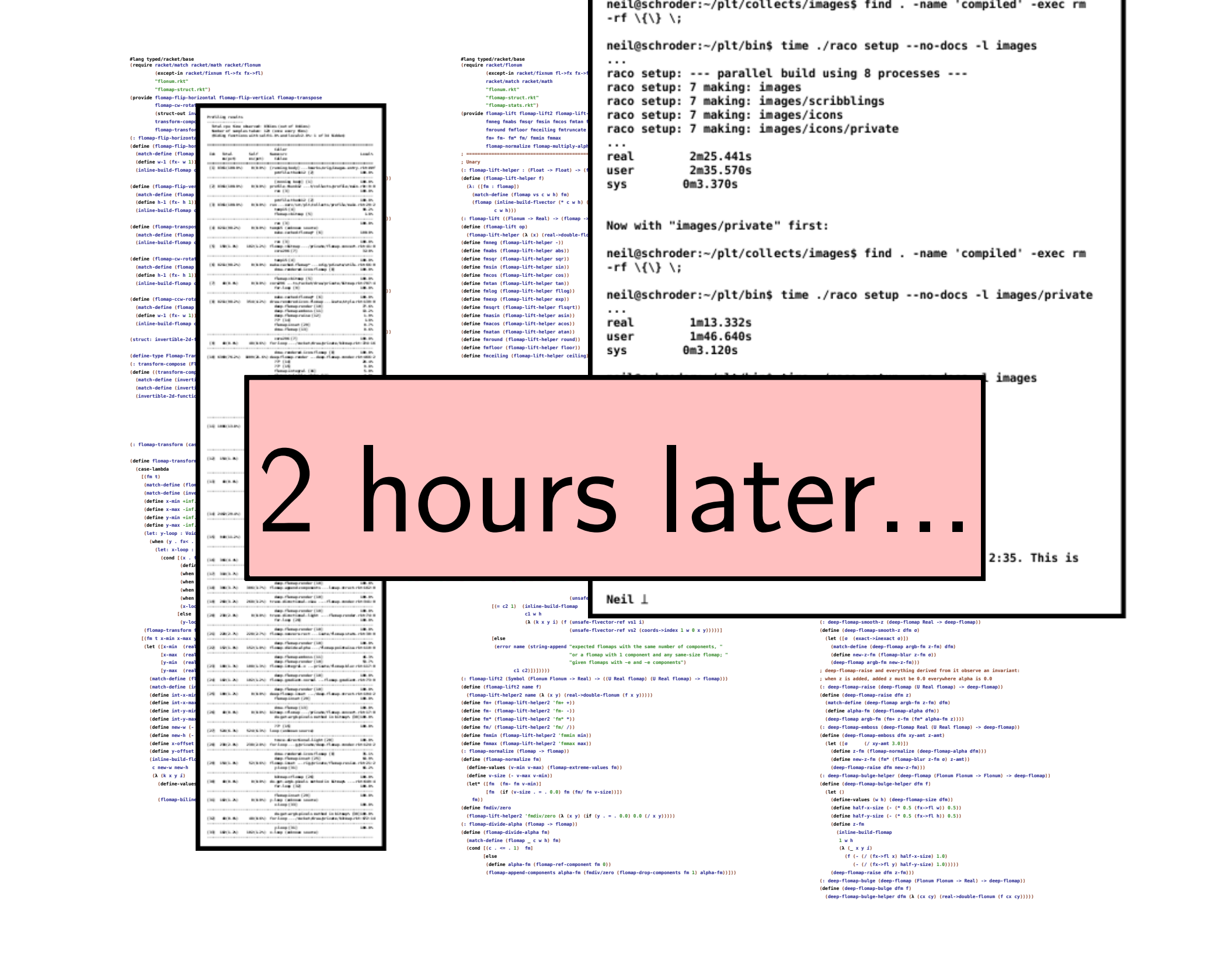
```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         racket/match racket/math
         "flonum.rkt"
         "flomap-struct.rkt"
         "flomap-stats.rkt")
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
         fmneg fmabs fmsqr fmsin fmcos fmtan fmlog fmexp fmsqrt fmasin fmacos fmatan
         fmround fmfloor fmceiling fmtruncate fmzero
         fm+ fm- fm* fm/ fmmin fmmax
         flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
; ============================================================================================
; Unary
(: flomap-lift-helper : (Float -> Float) -> (flomap -> flomap))
(define (flomap-lift-helper f)
  (λ: ([fm : flomap])
    (match-define (flomap vs c w h) fm)
    (flomap (inline-build-flvector (* c w h) (λ (i) (f (unsafe-flvector-ref vs i))))
            c w h)))
(: flomap-lift ((Flonum -> Real) -> (flomap -> flomap)))
(define (flomap-lift op)
  (flomap-lift-helper (λ (x) (real->double-flonum (op x)))))
(define fmneg (flomap-lift-helper -))
(define fmabs (flomap-lift-helper abs))
(define fmsqr (flomap-lift-helper sqr))
(define fmsin (flomap-lift-helper sin))
(define fmcos (flomap-lift-helper cos))
(define fmtan (flomap-lift-helper tan))
(define fmlog (flomap-lift-helper fllog))
(define fmexp (flomap-lift-helper exp))
(define fmsqrt (flomap-lift-helper flsqrt))
(define fmasin (flomap-lift-helper asin))
(define fmacos (flomap-lift-helper acos))
(define fmatan (flomap-lift-helper atan))
(define fmround (flomap-lift-helper round))
(define fmfloor (flomap-lift-helper floor))
(define fmceiling (flomap-lift-helper ceiling))
```

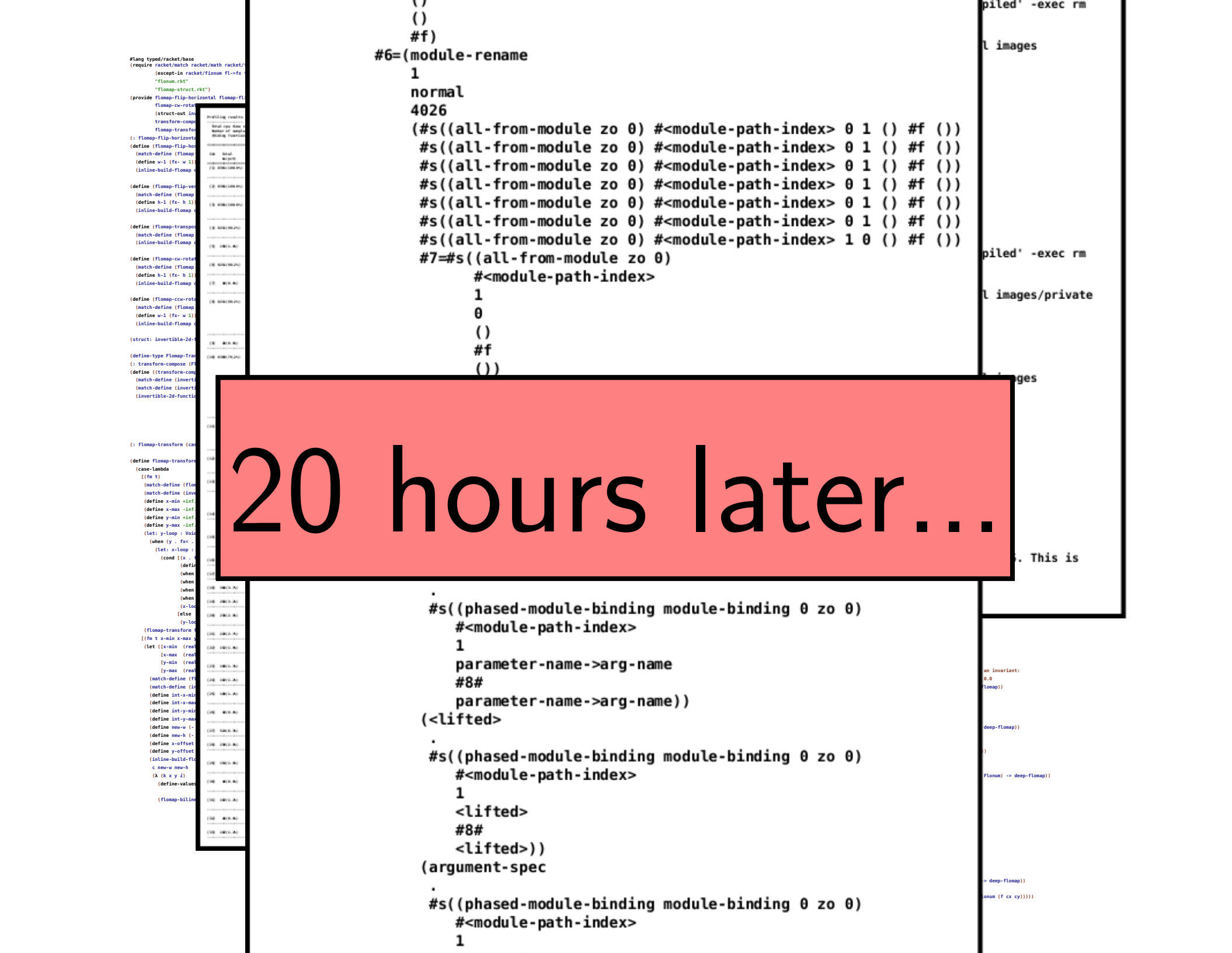
```
                                (unsafe-flvector-ref vs2 i))))]
      [(= c2 1)  (inline-build-flomap
                  c1 w h
                  (λ (k x y i) (f (unsafe-flvector-ref vs1 i)
                                  (unsafe-flvector-ref vs2 (coords->index 1 w 0 x y)))))]
      [else
       (error name (string-append "expected flomaps with the same number of components, "
                                  "or a flomap with 1 component and any same-size flomap; "
                                  "given flomaps with ~e and ~e components")
              c1 c2)])))))
(: flomap-lift2 (Symbol (Flonum Flonum -> Real) -> ((U Real flomap) (U Real flomap) -> flomap)))
(define (flomap-lift2 name f)
  (flomap-lift-helper2 name (λ (x y) (real->double-flonum (f x y)))))
(define fm+ (flomap-lift-helper2 'fm+ +))
(define fm- (flomap-lift-helper2 'fm- -))
(define fm* (flomap-lift-helper2 'fm* *))
(define fm/ (flomap-lift-helper2 'fm/ /))
(define fmmin (flomap-lift-helper2 'fmmin min))
(define fmmax (flomap-lift-helper2 'fmmax max))
(: flomap-normalize (flomap -> flomap))
(define (flomap-normalize fm)
  (define-values (v-min v-max) (flomap-extreme-values fm))
  (define v-size (- v-max v-min))
  (let* ([fm  (fm- fm v-min)]
         [fm  (if (v-size . = . 0.0) fm (fm/ fm v-size))])
    fm))
(define fmdiv/zero
  (flomap-lift-helper2 'fmdiv/zero (λ (x y) (if (y . = . 0.0) 0.0 (/ x y)))))
(: flomap-divide-alpha (flomap -> flomap))
(define (flomap-divide-alpha fm)
  (match-define (flomap _ c w h) fm)
  (cond [(c . <= . 1)  fm]
        [else
         (define alpha-fm (flomap-ref-component fm 0))
         (flomap-append-components alpha-fm (fmdiv/zero (flomap-drop-components fm 1) alpha-fm))]))
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fx->fl fl->fx)
         racket/match racket/math
         "flonum.rkt"
         "flomap.rkt")
(provide deep-flomap deep-flomap? deep-flomap-argb deep-flomap-z
         deep-flomap-width deep-flomap-height deep-flomap-z-min deep-flomap-z-max
         deep-flomap-size deep-flomap-alpha deep-flomap-rgb
         flomap->deep-flomap
         ; Sizing
         deep-flomap-inset deep-flomap-trim deep-flomap-scale deep-flomap-resize
         ; Z-adjusting
         deep-flomap-scale-z deep-flomap-smooth-z deep-flomap-raise deep-flomap-tilt
         deep-flomap-emboss
         deep-flomap-bulge deep-flomap-bulge-round deep-flomap-bulge-round-rect
         deep-flomap-bulge-spheroid deep-flomap-bulge-horizontal deep-flomap-bulge-vertical
         deep-flomap-bulge-ripple
         ; Compositing
         deep-flomap-pin deep-flomap-pin*
         deep-flomap-lt-superimpose deep-flomap-lc-superimpose deep-flomap-lb-superimpose
         deep-flomap-ct-superimpose deep-flomap-cc-superimpose deep-flomap-cb-superimpose
         deep-flomap-rt-superimpose deep-flomap-rc-superimpose deep-flomap-rb-superimpose
         deep-flomap-vl-append deep-flomap-vc-append deep-flomap-vr-append
         deep-flomap-ht-append deep-flomap-hc-append deep-flomap-hb-append)
(struct: deep-flomap ([argb : flomap] [z : flomap])
  #:transparent
  #:guard
  (λ (argb-fm z-fm name)
    (match-define (flomap _ 4 w h) argb-fm)
    (match-define (flomap _ 1 zw zh) z-fm)
    (unless (and (= w zw) (= h zh))
      (error 'deep-flomap
             "expected flomaps of equal dimension; given dimensions ~ex~e and ~ex~e" w h zw zh))
    (values argb-fm z-fm)))
(: flomap->deep-flomap (flomap -> deep-flomap))
(define (flomap->deep-flomap argb-fm)
  (match-define (flomap _ 4 w h) argb-fm)
                -fm (make-flomap 1 w h)))
                h (deep-flomap -> Nonnegative-Fixnum))
                p-width dfm)
                -width (deep-flomap-argb dfm)))
                  nonnegative-fixnum?])

                ht (deep-flomap -> Nonnegative-Fixnum))
                p-height dfm)
                -height (deep-flomap-argb dfm)))
                  nonnegative-fixnum?])

                n (deep-flomap -> Flonum))
                p-z-min dfm)
                 (deep-flomap-z dfm)))
                x (deep-flomap -> Flonum))
                p-z-max dfm)
                 (deep-flomap-z dfm)))
                 (deep-flomap -> (values Nonnegative-Fixnum Nonnegative-Fixnum)))
                p-size dfm)
                map-width dfm) (deep-flomap-height dfm)))
                a (deep-flomap -> flomap))
                p-alpha dfm)
                nent (deep-flomap-argb dfm) 0))
                (deep-flomap -> flomap))
                p-rgb dfm)
                onents (deep-flomap-argb dfm) 1))
                ================================================================================


                e-z (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-scale-z dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (deep-flomap argb-fm (fm* z-fm z)))
(: deep-flomap-smooth-z (deep-flomap Real -> deep-flomap))
(define (deep-flomap-smooth-z dfm σ)
  (let ([σ  (exact->inexact σ)])
    (match-define (deep-flomap argb-fm z-fm) dfm)
    (define new-z-fm (flomap-blur z-fm σ))
    (deep-flomap argb-fm new-z-fm)))
; deep-flomap-raise and everything derived from it observe an invariant:
; when z is added, added z must be 0.0 everywhere alpha is 0.0
(: deep-flomap-raise (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-raise dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (define alpha-fm (deep-flomap-alpha dfm))
  (deep-flomap argb-fm (fm+ z-fm (fm* alpha-fm z))))
(: deep-flomap-emboss (deep-flomap Real (U Real flomap) -> deep-flomap))
(define (deep-flomap-emboss dfm xy-amt z-amt)
  (let ([σ        (/ xy-amt 3.0)])
    (define z-fm (flomap-normalize (deep-flomap-alpha dfm)))
    (define new-z-fm (fm* (flomap-blur z-fm σ) z-amt))
    (deep-flomap-raise dfm new-z-fm)))
(: deep-flomap-bulge-helper (deep-flomap (Flonum Flonum -> Flonum) -> deep-flomap))
(define (deep-flomap-bulge-helper dfm f)
  (let ()
    (define-values (w h) (deep-flomap-size dfm))
    (define half-x-size (- (* 0.5 (fx->fl w)) 0.5))
    (define half-y-size (- (* 0.5 (fx->fl h)) 0.5))
    (define z-fm
      (inline-build-flomap
       1 w h
       (λ (_ x y i)
         (f (- (/ (fx->fl x) half-x-size) 1.0)
            (- (/ (fx->fl y) half-y-size) 1.0)))))
    (deep-flomap-raise dfm z-fm)))
(: deep-flomap-bulge (deep-flomap (Flonum Flonum -> Real) -> deep-flomap))
(define (deep-flomap-bulge dfm f)
  (deep-flomap-bulge-helper dfm (λ (cx cy) (real->double-flonum (f cx cy)))))
```



1 hour later...


neil@schroder:~/plt/collects/images$ find . -name 'compiled' -exec rm
-rf \{\} \;
neil@schroder:~/plt/bin$ time ./raco setup --no-docs -l images
...
raco setup: --- parallel build using 8 processes ---
raco setup: 7 making: images
raco setup: 7 making: images/scribblings
raco setup: 7 making: images/icons
raco setup: 7 making: images/icons/private
...
real 2m25.441s
user 2m35.570s
sys 0m3.370s
Now with "images/private" first:
neil@schroder:~/plt/collects/images$ find . -name 'compiled' -exec rm
-rf \{\} \;
neil@schroder:~/plt/bin$ time ./raco setup --no-docs -l images/private
...
real 1m13.332s
user 1m46.640s
sys 0m3.120s
2 hours later...
2:35. This is
Neil ⊥

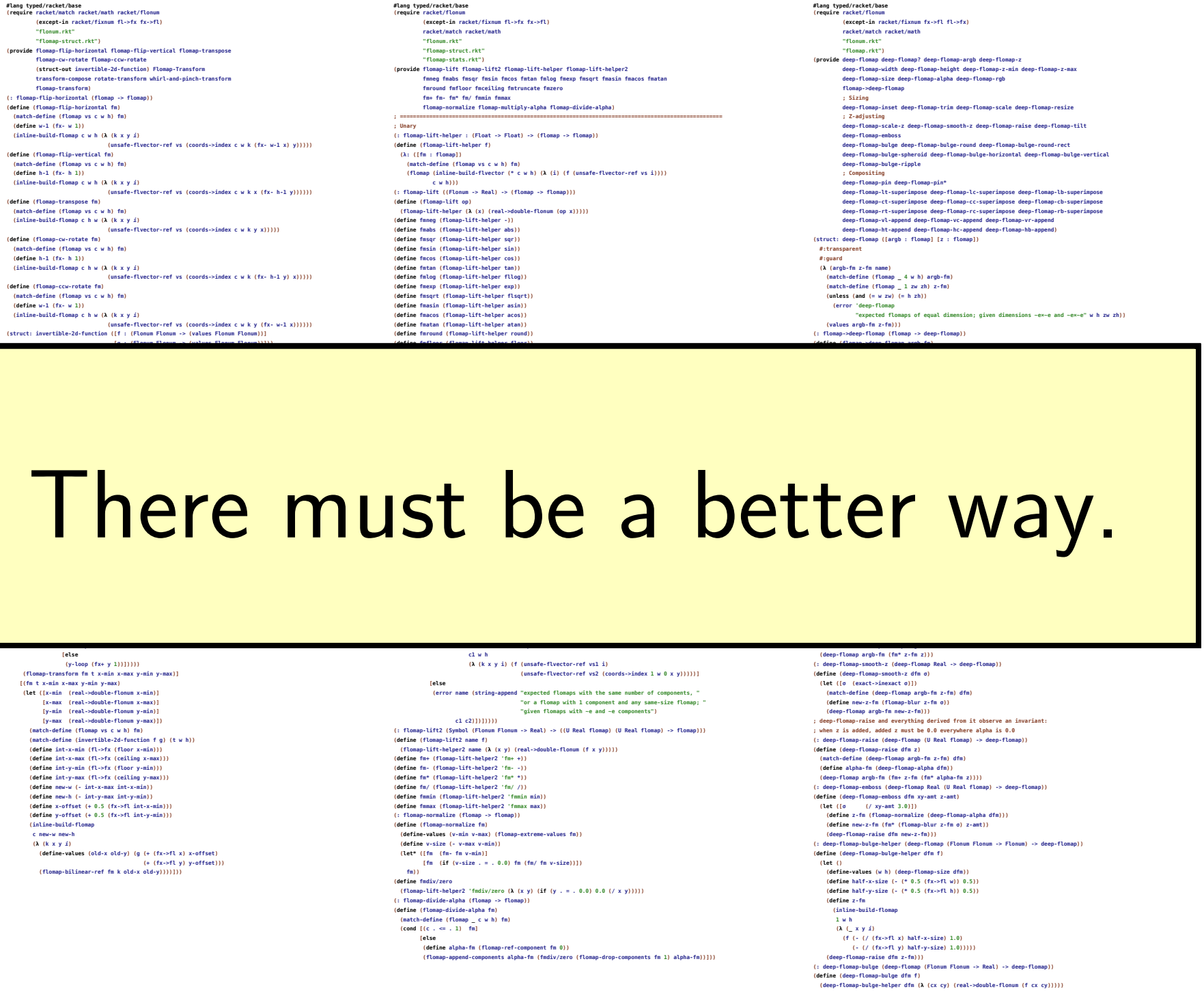

20 hours later...

```
#lang typed/racket/base
(require racket/match racket/math racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         "flonum.rkt"
         "flomap-struct.rkt")
(provide flomap-flip-horizontal flomap-flip-vertical flomap-transpose
         flomap-cw-rotate flomap-ccw-rotate
         (struct-out invertible-2d-function) Flomap-Transform
         transform-compose rotate-transform whirl-and-pinch-transform
         flomap-transform)
(: flomap-flip-horizontal (flomap -> flomap))
(define (flomap-flip-horizontal fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- w-1 x) y)))))
(define (flomap-flip-vertical fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k x (fx- h-1 y))))))
(define (flomap-transpose fm)
  (match-define (flomap vs c w h) fm)
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y x)))))
(define (flomap-cw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- h-1 y) x)))))
(define (flomap-ccw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y (fx- w-1 x))))))
(struct: invertible-2d-function ([f : (Flonum Flonum -> (values Flonum Flonum))]
                                 [g : (Flonum Flonum -> (values Flonum Flonum))]))
(define-type Flomap-Transform (Integer Integer -> invertible-2d-function))
(: transform-compose (Flomap-Transform Flomap-Transform -> Flomap-Transform))
(define ((transform-compose t1 t2) w h)
  (match-define (invertible-2d-function f1 g1) (t1 w h))
  (match-define (invertible-2d-function f2 g2) (t2 w h))
  (invertible-2d-function (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (f2 x y)])
                              (f1 x y)))
                          (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (g1 x y)])
                              (g2 x y)))))
(: flomap-transform (case-> (flomap Flomap-Transform -> flomap)
                            (flomap Flomap-Transform Real Real Real Real -> flomap)))
(define flomap-transform
  (case-lambda
    [(fm t)
     (match-define (flomap vs c w h) fm)
     (match-define (invertible-2d-function f g) (t w h))
     (define x-min +inf.0)
     (define x-max -inf.0)
     (define y-min +inf.0)
     (define y-max -inf.0)
     (let: y-loop : Void ([y : Integer  0])
       (when (y . fx< . h)
         (let: x-loop : Void ([x : Integer  0])
           (cond [(x . fx< . w)
                  (define-values (new-x new-y) (f (+ 0.5 (fx->fl x)) (+ 0.5 (fx->fl y))))
                  (when (new-x . < . x-min) (set! x-min new-x))
                  (when (new-x . > . x-max) (set! x-max new-x))
                  (when (new-y . < . y-min) (set! y-min new-y))
                  (when (new-y . > . y-max) (set! y-max new-y))
                  (x-loop (fx+ x 1))]
                 [else
                  (y-loop (fx+ y 1))]))))
     (flomap-transform fm t x-min x-max y-min y-max)]
    [(fm t x-min x-max y-min y-max)
     (let ([x-min  (real->double-flonum x-min)]
           [x-max  (real->double-flonum x-max)]
           [y-min  (real->double-flonum y-min)]
           [y-max  (real->double-flonum y-max)])
       (match-define (flomap vs c w h) fm)
       (match-define (invertible-2d-function f g) (t w h))
       (define int-x-min (fl->fx (floor x-min)))
       (define int-x-max (fl->fx (ceiling x-max)))
       (define int-y-min (fl->fx (floor y-min)))
       (define int-y-max (fl->fx (ceiling y-max)))
       (define new-w (- int-x-max int-x-min))
       (define new-h (- int-y-max int-y-min))
       (define x-offset (+ 0.5 (fx->fl int-x-min)))
       (define y-offset (+ 0.5 (fx->fl int-y-min)))
       (inline-build-flomap
        c new-w new-h
        (λ (k x y i)
          (define-values (old-x old-y) (g (+ (fx->fl x) x-offset)
                                          (+ (fx->fl y) y-offset)))
          (flomap-bilinear-ref fm k old-x old-y)))))]))
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         racket/match racket/math
         "flonum.rkt"
         "flomap-struct.rkt"
         "flomap-stats.rkt")
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
         fmneg fmabs fmsqr fmsin fmcos fmtan fmlog fmexp fmsqrt fmasin fmacos fmatan
         fmround fmfloor fmceiling fmtruncate fmzero
         fm+ fm- fm* fm/ fmmin fmmax
         flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
; ============================================================================================
; Unary
(: flomap-lift-helper : (Flonum -> Flonum) -> (flomap -> flomap))
(define (flomap-lift-helper f)
  (λ: ([fm : flomap])
    (match-define (flomap vs c w h) fm)
    (flomap (inline-build-flvector (* c w h) (λ (i) (f (unsafe-flvector-ref vs i))))
            c w h)))
(: flomap-lift ((Flonum -> Real) -> (flomap -> flomap)))
(define (flomap-lift op)
  (flomap-lift-helper (λ (x) (real->double-flonum (op x)))))
(define fmneg (flomap-lift-helper -))
(define fmabs (flomap-lift-helper abs))
(define fmsqr (flomap-lift-helper sqr))
(define fmsin (flomap-lift-helper sin))
(define fmcos (flomap-lift-helper cos))
(define fmtan (flomap-lift-helper tan))
(define fmlog (flomap-lift-helper fllog))
(define fmexp (flomap-lift-helper exp))
(define fmsqrt (flomap-lift-helper flsqrt))
(define fmasin (flomap-lift-helper asin))
(define fmacos (flomap-lift-helper acos))
(define fmatan (flomap-lift-helper atan))
(define fmround (flomap-lift-helper round))
(define fmfloor (flomap-lift-helper floor))
(define fmceiling (flomap-lift-helper ceiling))
(define fmtruncate (flomap-lift-helper truncate))
(define fmzero (flomap-lift-helper (λ (x) (if (x . = . 0.0) 1.0 0.0))))
; ;; ============================================================================================
; Binary
(: flomap-lift-helper2 : Symbol (Float Float -> Float) -> ((U Real flomap) (U Real flomap) -> flomap))
(define (flomap-lift-helper2 name f)
  (let: ()
    (λ: ([fm1 : (U Real flomap)] [fm2 : (U Real flomap)])
      (cond
        [(and (real? fm1) (real? fm2))
         (error name "expected at least one flomap argument; given ~e and ~e" fm1 fm2)]
        [(real? fm1)  (let ([fm1  (real->double-flonum fm1)])
                        ((flomap-lift-helper (λ (v) (f fm1 v))) fm2))]
        [(real? fm2)  (let ([fm2  (real->double-flonum fm2)])
                        ((flomap-lift-helper (λ (v) (f v fm2))) fm1))]
        [else
         (match-define (flomap vs1 c1 w h) fm1)
         (match-define (flomap vs2 c2 w2 h2) fm2)
         (cond
           [(not (and (= w w2) (= h h2)))
            (error name "expected same-size flomaps; given sizes ~ex~e and ~ex~e" w h w2 h2)]
           [(= c1 c2)  (define n (* c1 w h))
                       (define res-vs (make-flvector n))
                       (flomap (inline-build-flvector n (λ (i) (f (unsafe-flvector-ref vs1 i)
                                                                  (unsafe-flvector-ref vs2 i))))
                               c1 w h)]
           [(= c1 1)  (inline-build-flomap
                       c2 w h
                       (λ (k x y i) (f (unsafe-flvector-ref vs1 (coords->index 1 w 0 x y))
                                       (unsafe-flvector-ref vs2 i))))]
           [(= c2 1)  (inline-build-flomap
                       c1 w h
                       (λ (k x y i) (f (unsafe-flvector-ref vs1 i)
                                       (unsafe-flvector-ref vs2 (coords->index 1 w 0 x y)))))]
           [else
            (error name (string-append "expected flomaps with the same number of components, "
                                       "or a flomap with 1 component and any same-size flomap; "
                                       "given flomaps with ~e and ~e components")
                   c1 c2)])])))))
(: flomap-lift2 (Symbol (Flonum Flonum -> Real) -> ((U Real flomap) (U Real flomap) -> flomap)))
(define (flomap-lift2 name f)
  (flomap-lift-helper2 name (λ (x y) (real->double-flonum (f x y)))))
(define fm+ (flomap-lift-helper2 'fm+ +))
(define fm- (flomap-lift-helper2 'fm- -))
(define fm* (flomap-lift-helper2 'fm* *))
(define fm/ (flomap-lift-helper2 'fm/ /))
(define fmmin (flomap-lift-helper2 'fmmin min))
(define fmmax (flomap-lift-helper2 'fmmax max))
(: flomap-normalize (flomap -> flomap))
(define (flomap-normalize fm)
  (define-values (v-min v-max) (flomap-extreme-values fm))
  (define v-size (- v-max v-min))
  (let* ([fm  (fm- fm v-min)]
         [fm  (if (v-size . = . 0.0) fm (fm/ fm v-size))])
    fm))
(define fmdiv/zero
  (flomap-lift-helper2 'fmdiv/zero (λ (x y) (if (y . = . 0.0) 0.0 (/ x y)))))
(: flomap-divide-alpha (flomap -> flomap))
(define (flomap-divide-alpha fm)
  (match-define (flomap _ c w h) fm)
  (cond [(c . <= . 1)  fm]
        [else
         (define alpha-fm (flomap-ref-component fm 0))
         (flomap-append-components alpha-fm (fmdiv/zero (flomap-drop-components fm 1) alpha-fm))]))
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fx->fl fl->fx)
         racket/match racket/math
         "flonum.rkt"
         "flomap.rkt")
(provide deep-flomap deep-flomap? deep-flomap-argb deep-flomap-z
         deep-flomap-width deep-flomap-height deep-flomap-z-min deep-flomap-z-max
         deep-flomap-size deep-flomap-alpha deep-flomap-rgb
         flomap->deep-flomap
         ; Sizing
         deep-flomap-inset deep-flomap-trim deep-flomap-scale deep-flomap-resize
         ; Z-adjusting
         deep-flomap-scale-z deep-flomap-smooth-z deep-flomap-raise deep-flomap-tilt
         deep-flomap-emboss
         deep-flomap-bulge deep-flomap-bulge-round deep-flomap-bulge-round-rect
         deep-flomap-bulge-spheroid deep-flomap-bulge-horizontal deep-flomap-bulge-vertical
         deep-flomap-bulge-ripple
         ; Compositing
         deep-flomap-pin deep-flomap-pin*
         deep-flomap-lt-superimpose deep-flomap-lc-superimpose deep-flomap-lb-superimpose
         deep-flomap-ct-superimpose deep-flomap-cc-superimpose deep-flomap-cb-superimpose
         deep-flomap-rt-superimpose deep-flomap-rc-superimpose deep-flomap-rb-superimpose
         deep-flomap-vl-append deep-flomap-vc-append deep-flomap-vr-append
         deep-flomap-ht-append deep-flomap-hc-append deep-flomap-hb-append)
(struct: deep-flomap ([argb : flomap] [z : flomap])
  #:transparent
  #:guard
  (λ (argb-fm z-fm name)
    (match-define (flomap _ 4 w h) argb-fm)
    (match-define (flomap _ 1 zw zh) z-fm)
    (unless (and (= w zw) (= h zh))
      (error 'deep-flomap
             "expected flomaps of equal dimension; given dimensions ~ex~e and ~ex~e" w h zw zh))
    (values argb-fm z-fm)))
(: flomap->deep-flomap (flomap -> deep-flomap))
(define (flomap->deep-flomap argb-fm)
  (match-define (flomap _ 4 w h) argb-fm)
  (deep-flomap argb-fm (make-flomap 1 w h)))
(: deep-flomap-width (deep-flomap -> Nonnegative-Fixnum))
(define (deep-flomap-width dfm)
  (define w (flomap-width (deep-flomap-argb dfm)))
  (with-asserts ([w  nonnegative-fixnum?])
    w))
(: deep-flomap-height (deep-flomap -> Nonnegative-Fixnum))
(define (deep-flomap-height dfm)
  (define h (flomap-height (deep-flomap-argb dfm)))
  (with-asserts ([h  nonnegative-fixnum?])
    h))
(: deep-flomap-z-min (deep-flomap -> Flonum))
(define (deep-flomap-z-min dfm)
  (flomap-min-value (deep-flomap-z dfm)))
(: deep-flomap-z-max (deep-flomap -> Flonum))
(define (deep-flomap-z-max dfm)
  (flomap-max-value (deep-flomap-z dfm)))
(: deep-flomap-size (deep-flomap -> (values Nonnegative-Fixnum Nonnegative-Fixnum)))
(define (deep-flomap-size dfm)
  (values (deep-flomap-width dfm) (deep-flomap-height dfm)))
(: deep-flomap-alpha (deep-flomap -> flomap))
(define (deep-flomap-alpha dfm)
  (flomap-ref-component (deep-flomap-argb dfm) 0))
(: deep-flomap-rgb (deep-flomap -> flomap))
(define (deep-flomap-rgb dfm)
  (flomap-drop-components (deep-flomap-argb dfm) 1))
; ============================================================================================
; Z adjusters
(: deep-flomap-scale-z (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-scale-z dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (deep-flomap argb-fm (fm* z-fm z)))
(: deep-flomap-smooth-z (deep-flomap Real -> deep-flomap))
(define (deep-flomap-smooth-z dfm σ)
  (let ([σ  (exact->inexact σ)])
    (match-define (deep-flomap argb-fm z-fm) dfm)
    (define new-z-fm (flomap-blur z-fm σ))
    (deep-flomap argb-fm new-z-fm)))
; deep-flomap-raise and everything derived from it observe an invariant:
; when z is added, added z must be 0.0 everywhere alpha is 0.0
(: deep-flomap-raise (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-raise dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (define alpha-fm (deep-flomap-alpha dfm))
  (deep-flomap argb-fm (fm+ z-fm (fm* alpha-fm z))))
(: deep-flomap-emboss (deep-flomap Real (U Real flomap) -> deep-flomap))
(define (deep-flomap-emboss dfm xy-amt z-amt)
  (let ([σ       (/ xy-amt 3.0)])
    (define z-fm (flomap-normalize (deep-flomap-alpha dfm)))
    (define new-z-fm (fm* (flomap-blur z-fm σ) z-amt))
    (deep-flomap-raise dfm new-z-fm)))
(: deep-flomap-bulge-helper : deep-flomap (Flonum Flonum -> Flonum) -> deep-flomap)
(define (deep-flomap-bulge-helper dfm f)
  (let ()
    (define-values (w h) (deep-flomap-size dfm))
    (define half-x-size (- (* 0.5 (fx->fl w)) 0.5))
    (define half-y-size (- (* 0.5 (fx->fl h)) 0.5))
    (define z-fm
      (inline-build-flomap
       1 w h
       (λ (_ x y i)
         (f (- (/ (fx->fl x) half-x-size) 1.0)
            (- (/ (fx->fl y) half-y-size) 1.0)))))
    (deep-flomap-raise dfm z-fm)))
(: deep-flomap-bulge (deep-flomap (Flonum Flonum -> Real) -> deep-flomap))
(define (deep-flomap-bulge dfm f)
  (deep-flomap-bulge-helper dfm (λ (cx cy) (real->double-flonum (f cx cy)))))
```


define-syntax-rule
(define (flomap-lift-helper f) ...)



There must be a better way.

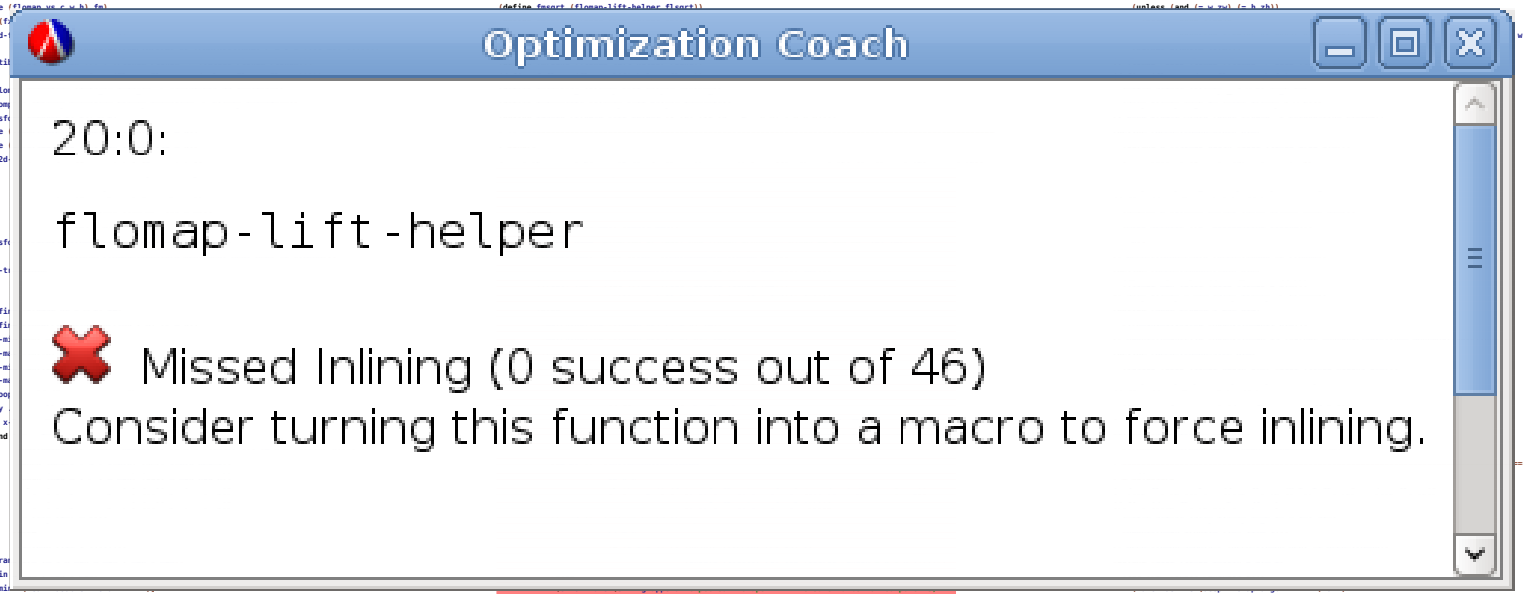
Optimization Coach

Seconds later

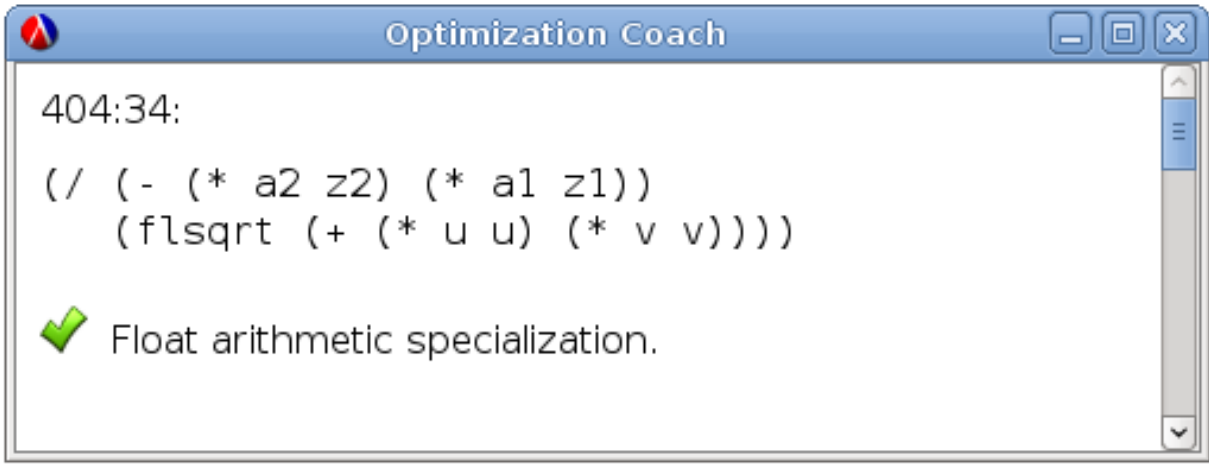
```
#lang typed/racket/base
(require racket/match racket/math racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         "flonum.rkt"
         "flomap-struct.rkt")
(provide flomap-flip-horizontal flomap-flip-vertical flomap-transpose
         flomap-cw-rotate flomap-ccw-rotate
         (struct-out invertible-2d-function) Flomap-Transform
         transform-compose rotate-transform whirl-and-pinch-transform
         flomap-transform)
(: flomap-flip-horizontal (flomap -> flomap))
(define (flomap-flip-horizontal fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- w-1 x) y)))))
(define (flomap-flip-vertical fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c w h (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k x (fx- h-1 y))))))
(define (flomap-transpose fm)
  (match-define (flomap vs c w h) fm)
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y x)))))
(define (flomap-cw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define h-1 (fx- h 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k (fx- h-1 y) x)))))
(define (flomap-ccw-rotate fm)
  (match-define (flomap vs c w h) fm)
  (define w-1 (fx- w 1))
  (inline-build-flomap c h w (λ (k x y i)
                               (unsafe-flvector-ref vs (coords->index c w k y (fx- w-1 x))))))
(struct: invertible-2d-function ([f : (Flonum Flonum -> (values Flonum Flonum))]
                                 [g : (Flonum Flonum -> (values Flonum Flonum))]))
(define-type Flomap-Transform (Integer Integer -> invertible-2d-function))
(: transform-compose (Flomap-Transform Flomap-Transform -> Flomap-Transform))
(define ((transform-compose t1 t2) w h)
  (match-define (invertible-2d-function f1 g1) (t1 w h))
  (match-define (invertible-2d-function f2 g2) (t2 w h))
  (invertible-2d-function (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (f2 x y)])
                              (f1 x y)))
                          (λ: ([x : Flonum] [y : Flonum])
                            (let-values ([(x y)  (g1 x y)])
                              (g2 x y)))))
(: flomap-transform (case-> (flomap Flomap-Transform -> flomap)
                            (flomap Flomap-Transform Real Real Real Real -> flomap)))
(define flomap-transform
  (case-lambda
    [(fm t)
     (match-define (flomap vs c w h) fm)
     (match-define (invertible-2d-function f g) (t w h))
     (define x-min +inf.0)
     (define x-max -inf.0)
     (define y-min +inf.0)
     (define y-max -inf.0)
     (let: y-loop : Void ([y : Integer  0])
       (when (y . fx< . h)
         (let: x-loop : Void ([x : Integer  0])
           (cond [(x . fx< . w)
                  (define-values (new-x new-y) (f (+ 0.5 (fx->fl x)) (+ 0.5 (fx->fl y))))
                  (when (new-x . < . x-min) (set! x-min new-x))
                  (when (new-x . > . x-max) (set! x-max new-x))
                  (when (new-y . < . y-min) (set! y-min new-y))
                  (when (new-y . > . y-max) (set! y-max new-y))
                  (x-loop (fx+ x 1))]
                 [else
                  (y-loop (fx+ y 1))])))))
     (flomap-transform fm t x-min x-max y-min y-max)]
    [(fm t x-min x-max y-min y-max)
     (let ([x-min  (real->double-flonum x-min)]
           [x-max  (real->double-flonum x-max)]
           [y-min  (real->double-flonum y-min)]
           [y-max  (real->double-flonum y-max)])
       (match-define (flomap vs c w h) fm)
       (match-define (invertible-2d-function f g) (t w h))
       (define int-x-min (fl->fx (floor x-min)))
       (define int-x-max (fl->fx (ceiling x-max)))
       (define int-y-min (fl->fx (floor y-min)))
       (define int-y-max (fl->fx (ceiling y-max)))
       (define new-w (- int-x-max int-x-min))
       (define new-h (- int-y-max int-y-min))
       (define x-offset (+ 0.5 (fx->fl int-x-min)))
       (define y-offset (+ 0.5 (fx->fl int-y-min)))
       (inline-build-flomap
        c new-w new-h
        (λ (k x y i)
          (define-values (old-x old-y) (g (+ (fx->fl x) x-offset)
                                          (+ (fx->fl y) y-offset)))
          (flomap-bilinear-ref fm k old-x old-y)))))]))
```

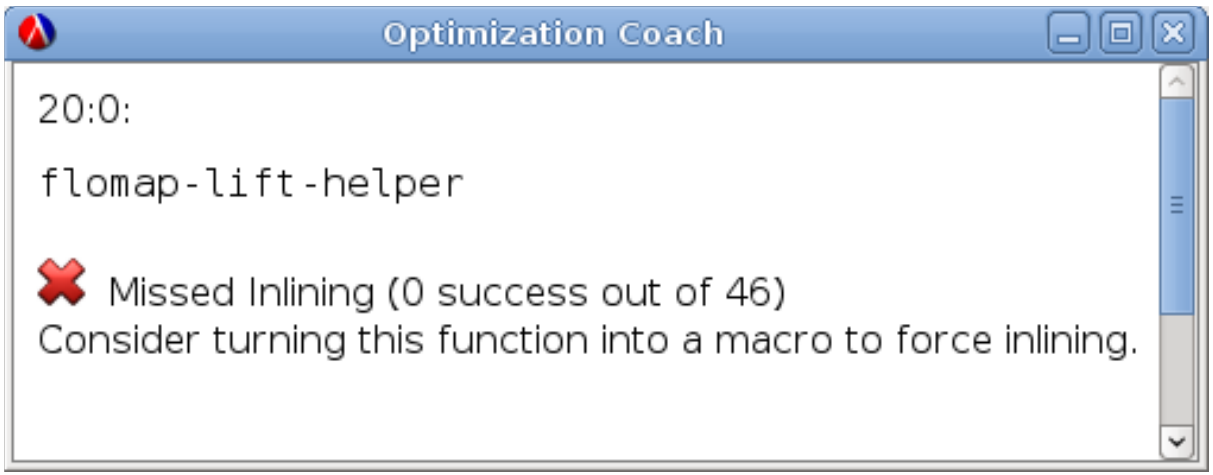
```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fl->fx fx->fl)
         racket/match racket/math
         "flonum.rkt"
         "flomap-struct.rkt"
         "flomap-stats.rkt")
(provide flomap-lift flomap-lift2 flomap-lift-helper flomap-lift-helper2
         fmneg fmabs fmsqr fmsin fmcos fmtan fmlog fmexp fmsqrt fmasin fmacos fmatan
         fmround fmfloor fmceiling fmtruncate fmzero
         fm+ fm- fm* fm/ fmmin fmmax
         flomap-normalize flomap-multiply-alpha flomap-divide-alpha)
; ============================================================================================
; Unary
(: flomap-lift-helper : (Float -> Float) -> (flomap -> flomap))
(define (flomap-lift-helper f)
  (λ: ([fm : flomap])
    (match-define (flomap vs c w h) fm)
    (flomap (inline-build-flvector (* c w h) (λ (i) (f (unsafe-flvector-ref vs i))))
            c w h)))
(: flomap-lift ((Flonum -> Real) -> (flomap -> flomap)))
(define (flomap-lift op)
  (flomap-lift-helper (λ (x) (real->double-flonum (op x)))))
(define fmneg (flomap-lift-helper -))
(define fmabs (flomap-lift-helper abs))
(define fmsqr (flomap-lift-helper sqr))
(define fmsin (flomap-lift-helper sin))
(define fmcos (flomap-lift-helper cos))
(define fmtan (flomap-lift-helper tan))
(define fmlog (flomap-lift-helper fllog))
(define fmexp (flomap-lift-helper exp))
(define fmsqrt (flomap-lift-helper flsqrt))
(define fmasin (flomap-lift-helper asin))
(define fmacos (flomap-lift-helper acos))
(define fmatan (flomap-lift-helper atan))
(define fmround (flomap-lift-helper round))
(define fmfloor (flomap-lift-helper floor))
(define fmceiling (flomap-lift-helper ceiling))
(define fmtruncate (flomap-lift-helper truncate))
(define fmzero (flomap-lift-helper (λ (x) (if (x . = . 0.0) 1.0 0.0))))
; ;; ============================================================================================
; Binary
(: flomap-lift-helper2 : Symbol (Float Float -> Float) -> ((U Real flomap) (U Real flomap) -> flomap))
(define (flomap-lift-helper2 name f)
  (let: ()
    (λ: ([fm1 : (U Real flomap)] [fm2 : (U Real flomap)])
      (cond
        [(and (real? fm1) (real? fm2))
         (error name "expected at least one flomap argument; given ~e and ~e" fm1 fm2)]
        [(real? fm1)  (let ([fm1  (real->double-flonum fm1)])
                        ((flomap-lift-helper (λ (v) (f fm1 v))) fm2))]
        [(real? fm2)  (let ([fm2  (real->double-flonum fm2)])
                        ((flomap-lift-helper (λ (v) (f v fm2))) fm1))]
        [else
         (match-define (flomap vs1 c1 w h) fm1)
         (match-define (flomap vs2 c2 w2 h2) fm2)
         (cond
           [(not (and (= w w2) (= h h2)))
            (error name "expected same-size flomaps; given sizes ~ex~e and ~ex~e" w h w2 h2)]
           [(= c1 c2)  (define n (* c1 w h))
                       (define res-vs (make-flvector n))
                       (flomap (inline-build-flvector n (λ (i) (f (unsafe-flvector-ref vs1 i)
                                                                  (unsafe-flvector-ref vs2 i))))
                               c1 w h)]
           [(= c1 1)  (inline-build-flomap
                       c2 w h
                       (λ (k x y i) (f (unsafe-flvector-ref vs1 (coords->index 1 w 0 x y))
                                       (unsafe-flvector-ref vs2 i))))]
           [(= c2 1)  (inline-build-flomap
                       c1 w h
                       (λ (k x y i) (f (unsafe-flvector-ref vs1 i)
                                       (unsafe-flvector-ref vs2 (coords->index 1 w 0 x y)))))]
           [else
            (error name (string-append "expected flomaps with the same number of components, "
                                       "or a flomap with 1 component and any same-size flomap; "
                                       "given flomaps with ~e and ~e components")
                   c1 c2)])])))))
(: flomap-lift2 (Symbol (Flonum Flonum -> Real) -> ((U Real flomap) (U Real flomap) -> flomap)))
(define (flomap-lift2 name f)
  (flomap-lift-helper2 name (λ (x y) (real->double-flonum (f x y)))))
(define fm+ (flomap-lift-helper2 'fm+ +))
(define fm- (flomap-lift-helper2 'fm- -))
(define fm* (flomap-lift-helper2 'fm* *))
(define fm/ (flomap-lift-helper2 'fm/ /))
(define fmmin (flomap-lift-helper2 'fmmin min))
(define fmmax (flomap-lift-helper2 'fmmax max))
(: flomap-normalize (flomap -> flomap))
(define (flomap-normalize fm)
  (define-values (v-min v-max) (flomap-extreme-values fm))
  (define v-size (- v-max v-min))
  (let* ([fm  (fm- fm v-min)]
         [fm  (if (v-size . = . 0.0) fm (fm/ fm v-size))])
    fm))
(define fmdiv/zero
  (flomap-lift-helper2 'fmdiv/zero (λ (x y) (if (y . = . 0.0) 0.0 (/ x y)))))
(: flomap-divide-alpha (flomap -> flomap))
(define (flomap-divide-alpha fm)
  (match-define (flomap _ c w h) fm)
  (cond [(c . <= . 1)  fm]
        [else
         (define alpha-fm (flomap-ref-component fm 0))
         (flomap-append-components alpha-fm (fmdiv/zero (flomap-drop-components fm 1) alpha-fm))]))
```

```
#lang typed/racket/base
(require racket/flonum
         (except-in racket/fixnum fx->fl fl->fx)
         racket/match racket/math
         "flonum.rkt"
         "flomap.rkt")
(provide deep-flomap deep-flomap? deep-flomap-argb deep-flomap-z
         deep-flomap-width deep-flomap-height deep-flomap-z-min deep-flomap-z-max
         deep-flomap-size deep-flomap-alpha deep-flomap-rgb
         flomap->deep-flomap
         ; Sizing
         deep-flomap-inset deep-flomap-trim deep-flomap-scale deep-flomap-resize
         ; Z-adjusting
         deep-flomap-scale-z deep-flomap-smooth-z deep-flomap-raise deep-flomap-tilt
         deep-flomap-emboss
         deep-flomap-bulge deep-flomap-bulge-round deep-flomap-bulge-round-rect
         deep-flomap-bulge-spheroid deep-flomap-bulge-horizontal deep-flomap-bulge-vertical
         deep-flomap-bulge-ripple
         ; Compositing
         deep-flomap-pin deep-flomap-pin*
         deep-flomap-lt-superimpose deep-flomap-lc-superimpose deep-flomap-lb-superimpose
         deep-flomap-ct-superimpose deep-flomap-cc-superimpose deep-flomap-cb-superimpose
         deep-flomap-rt-superimpose deep-flomap-rc-superimpose deep-flomap-rb-superimpose
         deep-flomap-vl-append deep-flomap-vc-append deep-flomap-vr-append
         deep-flomap-ht-append deep-flomap-hc-append deep-flomap-hb-append)
(struct: deep-flomap ([argb : flomap] [z : flomap])
  #:transparent
  #:guard
  (λ (argb-fm z-fm name)
    (match-define (flomap _ 4 w h) argb-fm)
    (match-define (flomap _ 1 zw zh) z-fm)
    (unless (and (= w zw) (= h zh))
      (error 'deep-flomap
             "expected flomaps of equal dimension; given dimensions ~ex~e and ~ex~e" w h zw zh))
    (values argb-fm z-fm)))
(: flomap->deep-flomap (flomap -> deep-flomap))
(define (flomap->deep-flomap argb-fm)
  (match-define (flomap _ 4 w h) argb-fm)
  (deep-flomap argb-fm (make-flomap 1 w h)))
(: deep-flomap-width (deep-flomap -> Nonnegative-Fixnum))
(define (deep-flomap-width dfm)
  (define w (flomap-width (deep-flomap-argb dfm)))
  (with-asserts ([w  nonnegative-fixnum?])
    w))
(: deep-flomap-height (deep-flomap -> Nonnegative-Fixnum))
(define (deep-flomap-height dfm)
  (define h (flomap-height (deep-flomap-argb dfm)))
  (with-asserts ([h  nonnegative-fixnum?])
    h))
(: deep-flomap-z-min (deep-flomap -> Flonum))
(define (deep-flomap-z-min dfm)
  (flomap-min-value (deep-flomap-z dfm)))
(: deep-flomap-z-max (deep-flomap -> Flonum))
(define (deep-flomap-z-max dfm)
  (flomap-max-value (deep-flomap-z dfm)))
(: deep-flomap-size (deep-flomap -> (values Nonnegative-Fixnum Nonnegative-Fixnum)))
(define (deep-flomap-size dfm)
  (values (deep-flomap-width dfm) (deep-flomap-height dfm)))
(: deep-flomap-alpha (deep-flomap -> flomap))
(define (deep-flomap-alpha dfm)
  (flomap-ref-component (deep-flomap-argb dfm) 0))
(: deep-flomap-rgb (deep-flomap -> flomap))
(define (deep-flomap-rgb dfm)
  (flomap-drop-components (deep-flomap-argb dfm) 1))
; ============================================================================================
; Z adjusters
(: deep-flomap-scale-z (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-scale-z dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (deep-flomap argb-fm (fm* z-fm z)))
(: deep-flomap-smooth-z (deep-flomap Real -> deep-flomap))
(define (deep-flomap-smooth-z dfm σ)
  (let ([σ  (exact->inexact σ)])
    (match-define (deep-flomap argb-fm z-fm) dfm)
    (define new-z-fm (flomap-blur z-fm σ))
    (deep-flomap argb-fm new-z-fm)))
; deep-flomap-raise and everything derived from it observe an invariant:
; when z is added, added z must be 0.0 everywhere alpha is 0.0
(: deep-flomap-raise (deep-flomap (U Real flomap) -> deep-flomap))
(define (deep-flomap-raise dfm z)
  (match-define (deep-flomap argb-fm z-fm) dfm)
  (define alpha-fm (deep-flomap-alpha dfm))
  (deep-flomap argb-fm (fm+ z-fm (fm* alpha-fm z))))
(: deep-flomap-emboss (deep-flomap Real (U Real flomap) -> deep-flomap))
(define (deep-flomap-emboss dfm xy-amt z-amt)
  (let ([σ        (/ xy-amt 3.0)])
    (define z-fm (flomap-normalize (deep-flomap-alpha dfm)))
    (define new-z-fm (fm* (flomap-blur z-fm σ) z-amt))
    (deep-flomap-raise dfm new-z-fm)))
(: deep-flomap-bulge-helper (deep-flomap (Flonum Flonum -> Flonum) -> deep-flomap))
(define (deep-flomap-bulge-helper dfm f)
  (let ()
    (define-values (w h) (deep-flomap-size dfm))
    (define half-x-size (- (* 0.5 (fx->fl w)) 0.5))
    (define half-y-size (- (* 0.5 (fx->fl h)) 0.5))
    (define z-fm
      (inline-build-flomap
       1 w h
       (λ (_ x y i)
         (f (- (/ (fx->fl x) half-x-size) 1.0)
            (- (/ (fx->fl y) half-y-size) 1.0)))))
    (deep-flomap-raise dfm z-fm)))
(: deep-flomap-bulge (deep-flomap (Flonum Flonum -> Real) -> deep-flomap))
(define (deep-flomap-bulge dfm f)
  (deep-flomap-bulge-helper dfm (λ (cx cy) (real->double-flonum (f cx cy)))))
```

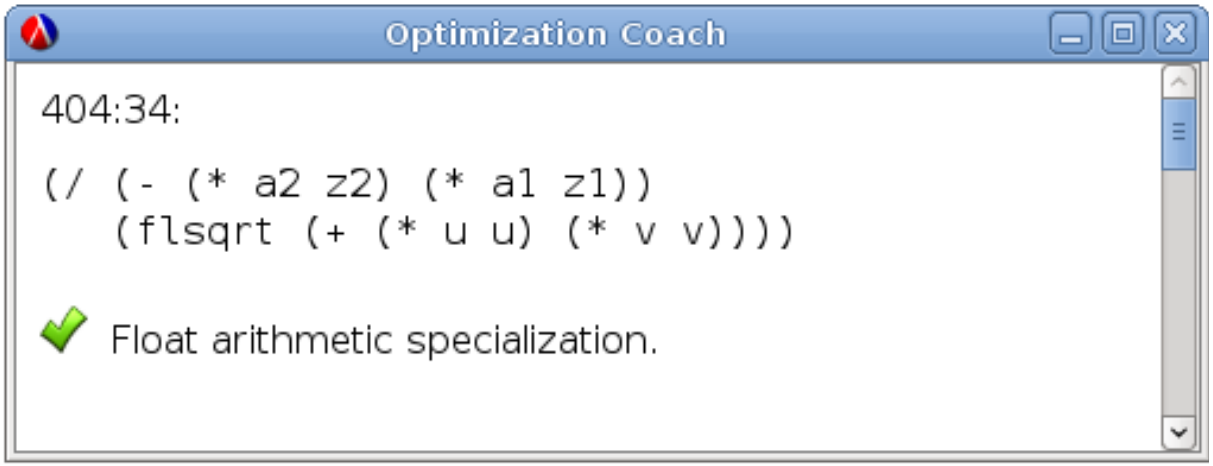

Optimization Coach
20:0:
flomap-lift-helper
Missed Inlining (0 success out of 46)
Consider turning this function into a macro to force inlining.

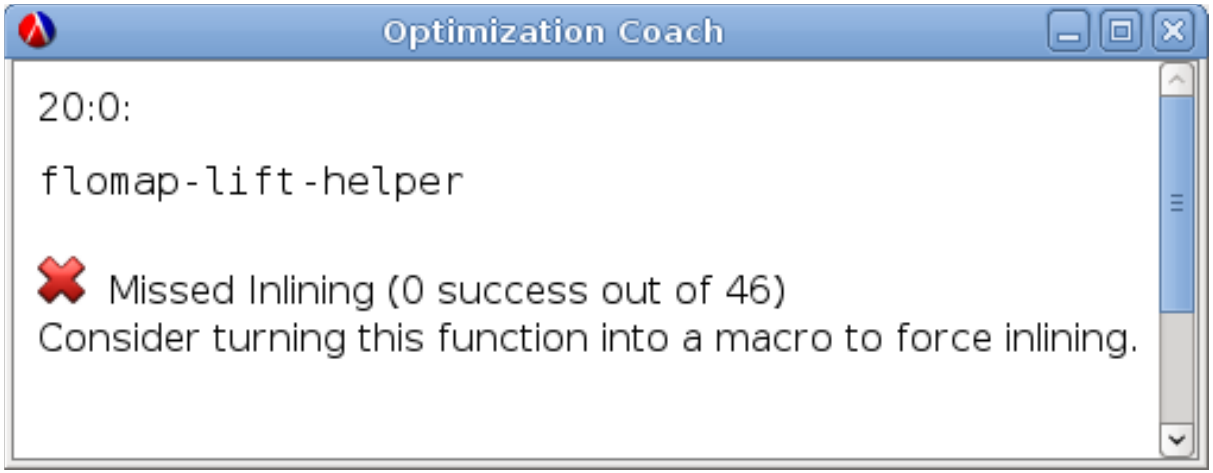
Dialog between compilers and programmers





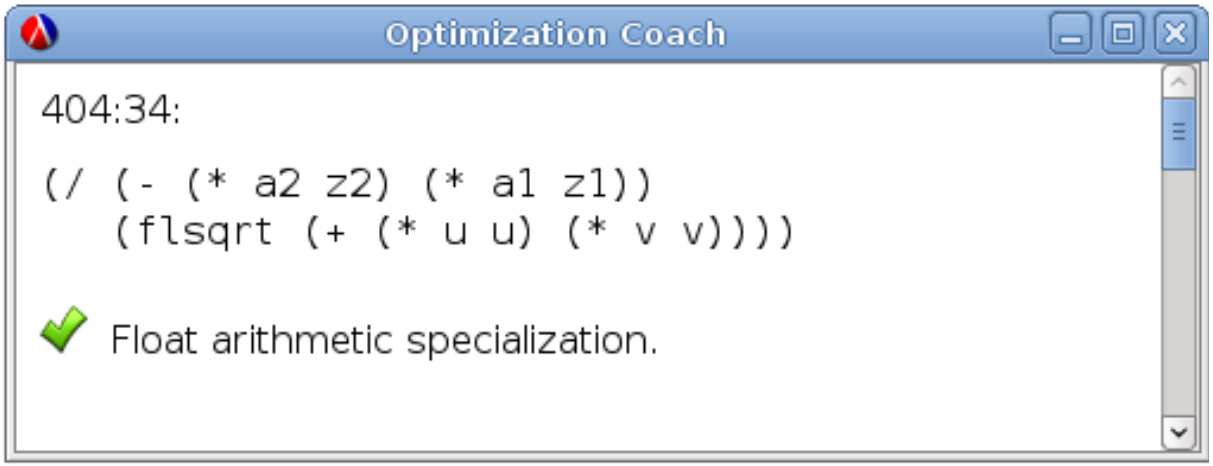
Dialog between compilers and programmers

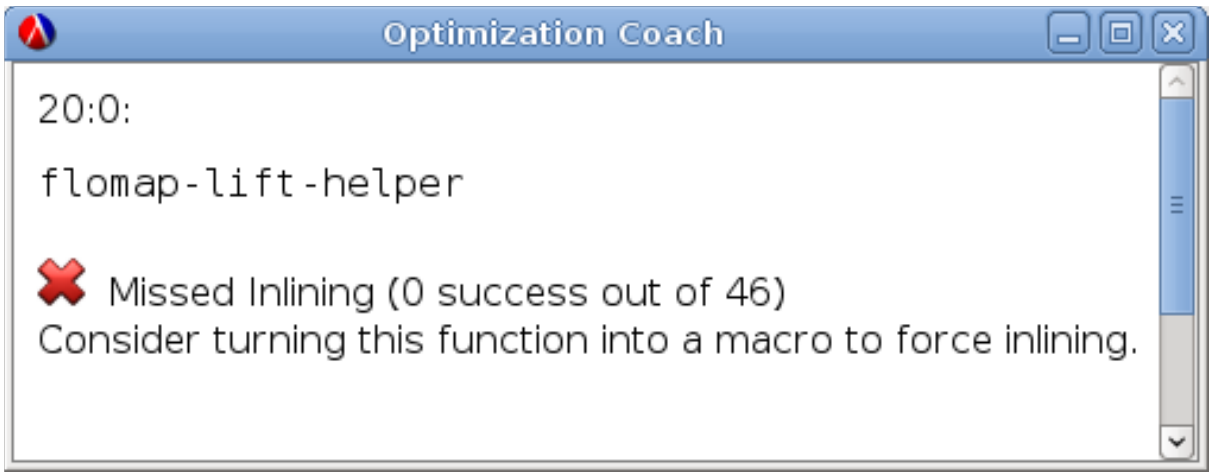




Successes

Dialog between compilers and programmers





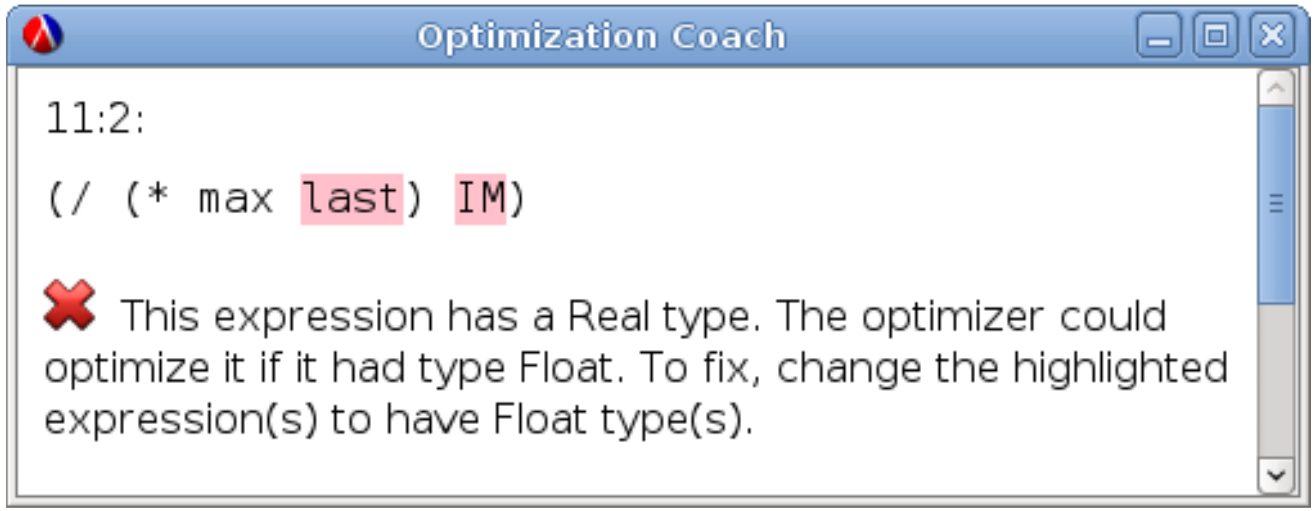
Successes

Near misses

+

Recommendations

Programmers can do more than compilers



Recommendations can change semantics!

```
(/ 1 3)       ➡ 1/3
(/ 1.0 3.0)   ➡ 0.3333333333333333
```

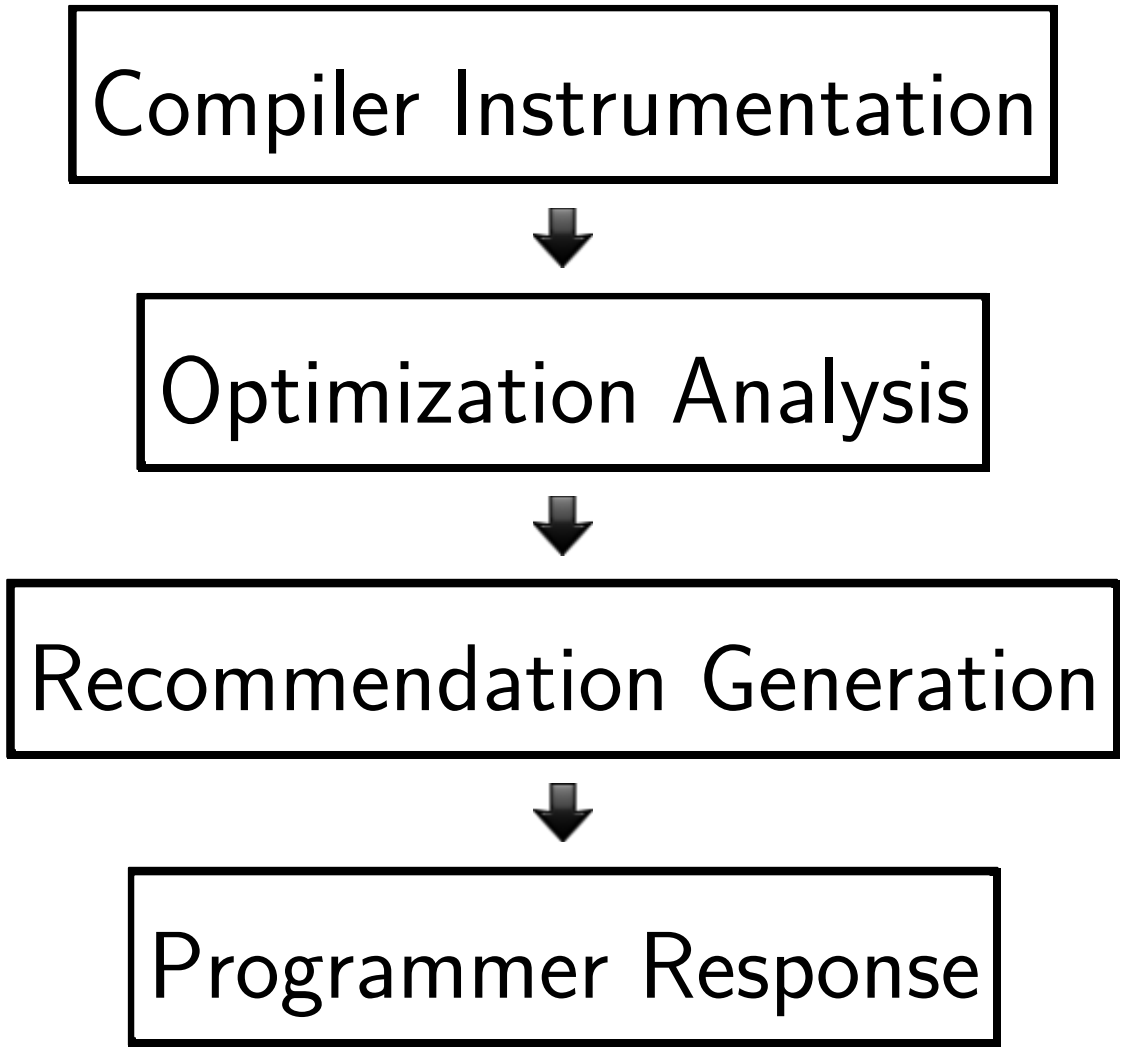
How does it work?

How well does it work?

How to extend it?

How does it work?

Overview


Compiler Instrumentation
Optimization Analysis
Recommendation Generation
Programmer Response

Type-Driven Specialization

```
#lang typed/racket/base

(define IM   139968)
(define IA     3877)
(define IC    29573)

(define last 42)
(define min  35.3)
(define max  156.8)
(define (gen-random)
  (set! last (modulo (+ (* last IA) IC) IM))
  (+ (/ (* (- max min) last) IM) min))
```

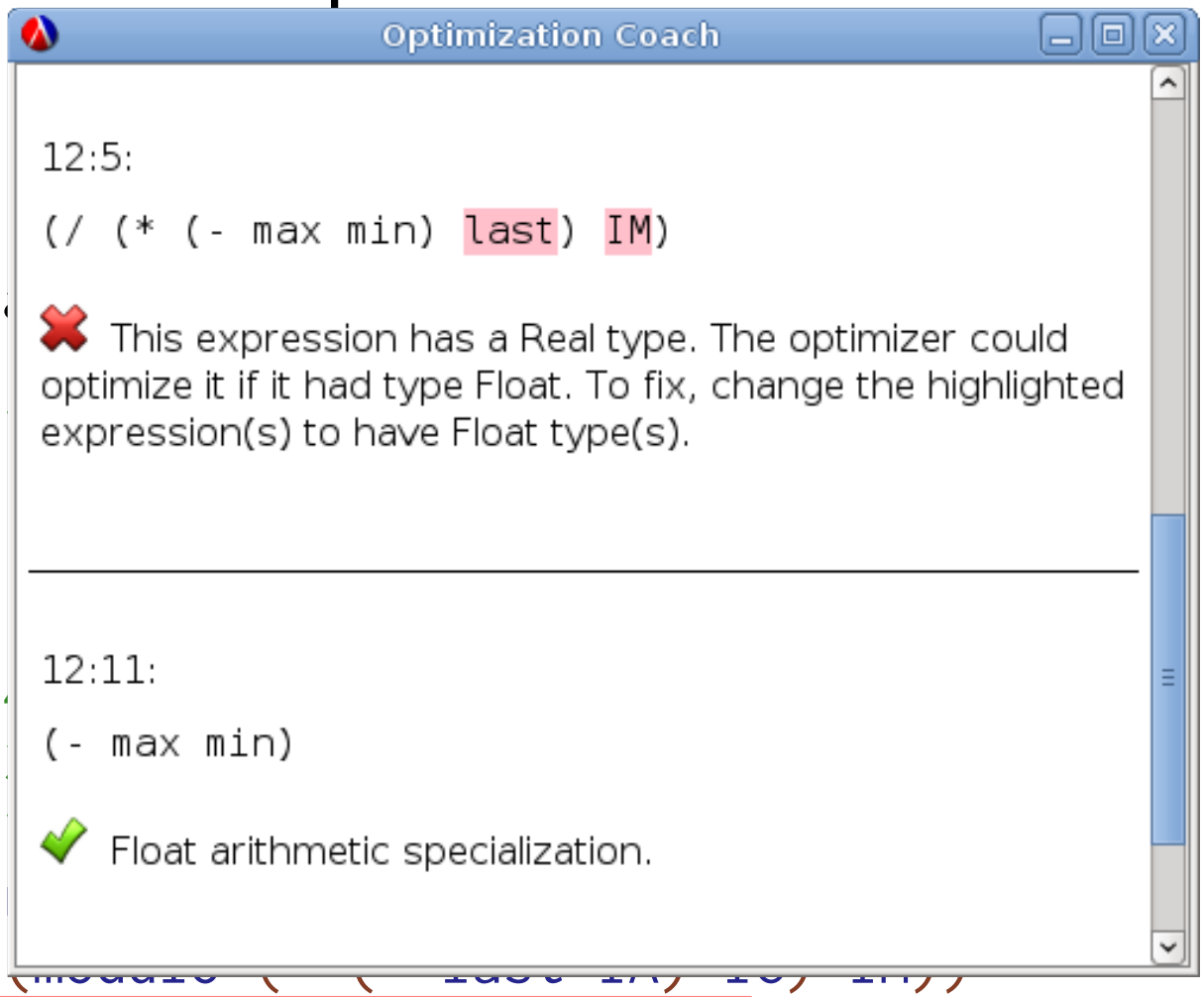
Type-Driven Specialization

```
#lang typed/racket/base

(define IM
(define IA
(define IC

(define last 42)
(define min  35.3)
(define max  156.8)
(define (gen-random)
  (set! last (modulo (+ (* last IA) IC) IM))
  (+ (/ (* (- max min) last) IM) min))
```

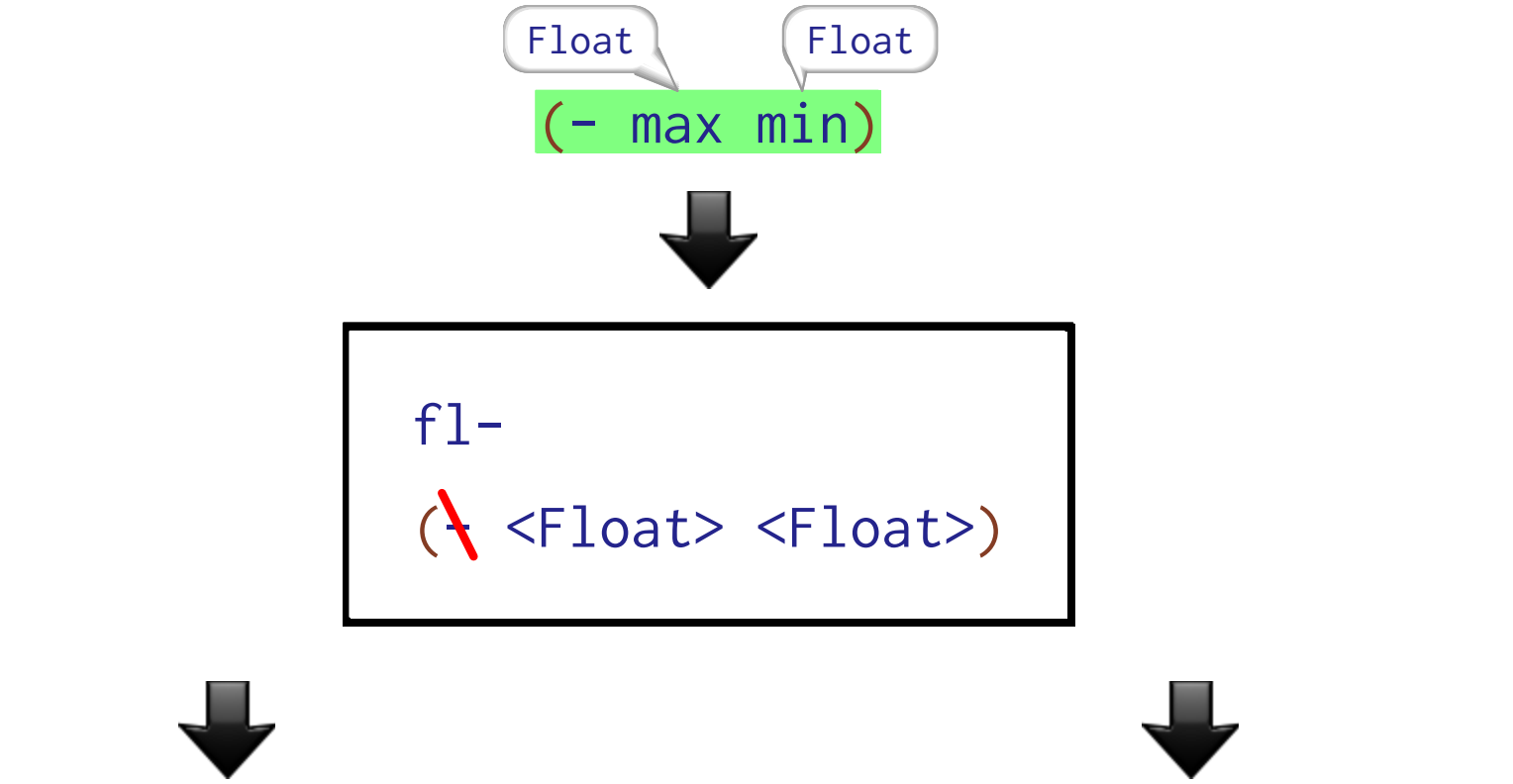
```
fl-
(- <Float> <Float>)
```

Type-Driven Specialization

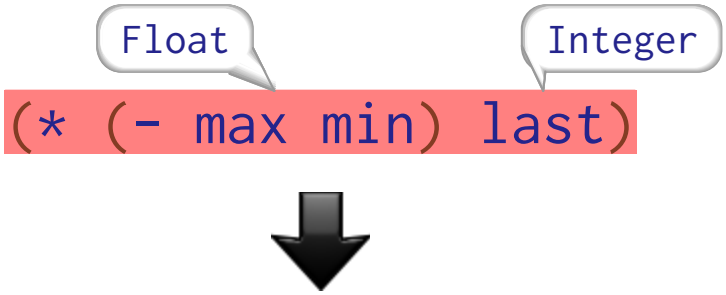
```
#lang typed/r

(define IM
(define IA
(define IC

(define last
(define min
(define max
(define (gen-
  (set! last
  (+ (/ (* (- max min) last) IM) min))
```



Compiler Instrumentation



Compiler Instrumentation



(* <Number> <Number>) ; no change

(* (- max min) last)

```
TR opt failure: prng-example.rkt 12:8
  (* (- max min) last)
  Float Integer
  generic multiplication
```

Optimization Analysis

Incomprehensible failure pruning

Irrelevant failure pruning

Harmless failure pruning

Optimization proximity

Causality merging

Locality merging

Optimization Analysis

Incomprehensible failure pruning

Irrelevant failure pruning

Harmless failure pruning

Optimization proximity

Causality merging

Locality merging

Optimization Analysis

Incomprehensible failure pruning

Irrelevant failure pruning

Harmless failure pruning

Optimization proximity

Causality merging

Locality merging

Optimization Analysis

Incomprehensible failure pruning

Irrelevant failure pruning

Can't trace back to user program.

Optimization proximity

Causality merging

Locality merging

Optimization Analysis

Incomprehensible failure pruning

Irrelevant failure pruning

Non-optimized semantics is desirable.

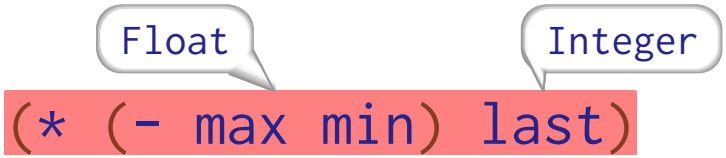
Optimization proximity

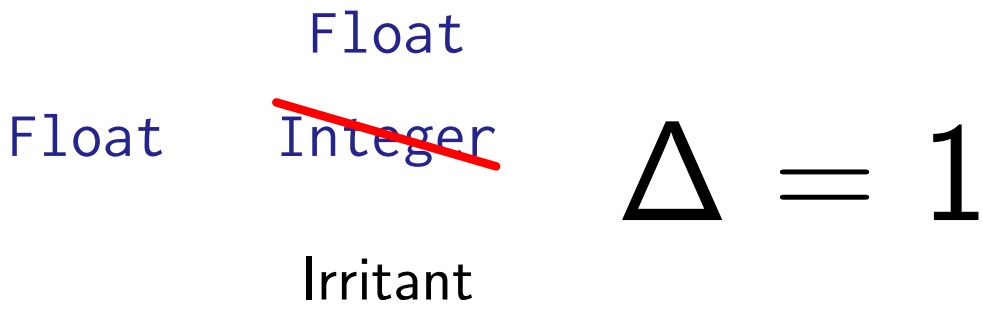
Causality merging

Locality merging

Optimization Analysis

Optimization proximity



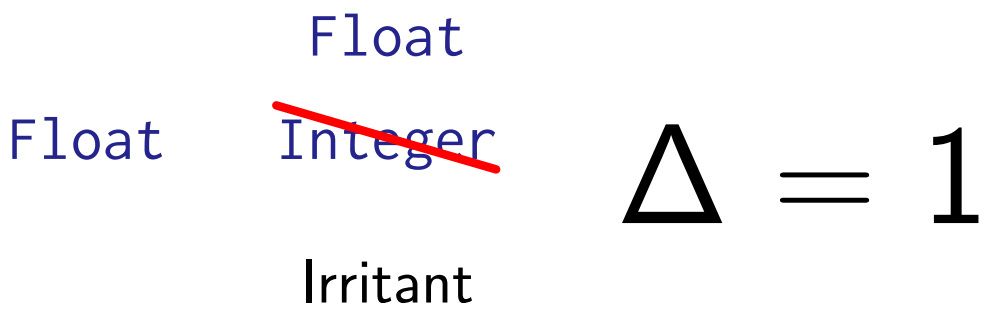


$$\Delta = 1$$

Optimization Analysis

Optimization proximity

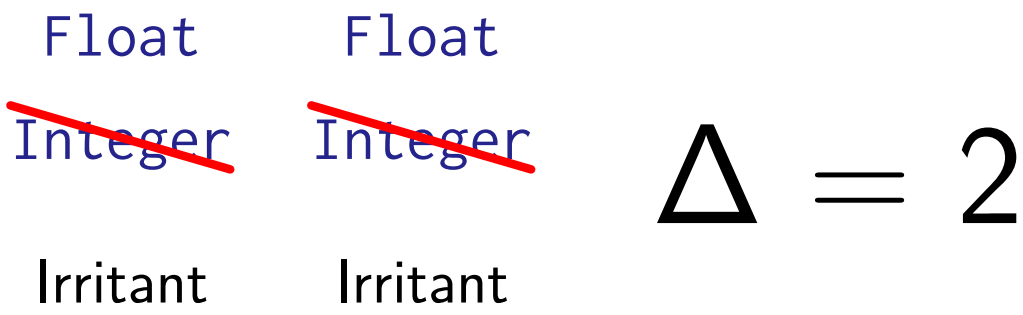
Near miss, report



$$\Delta = 1$$

Optimization Analysis

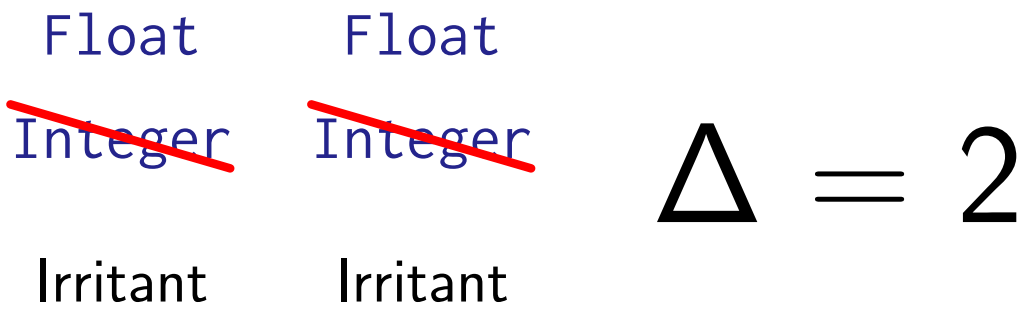
Optimization proximity



Optimization Analysis

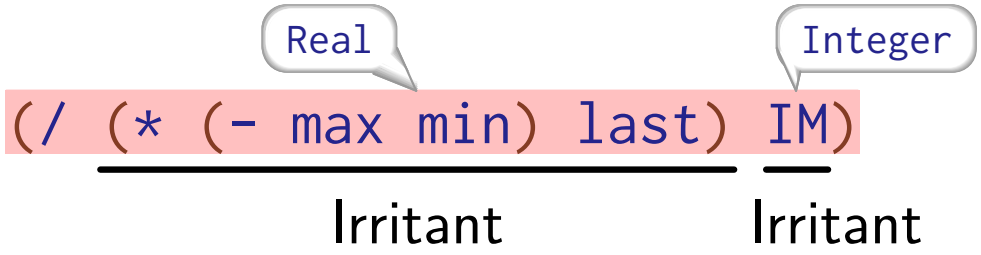
Optimization proximity

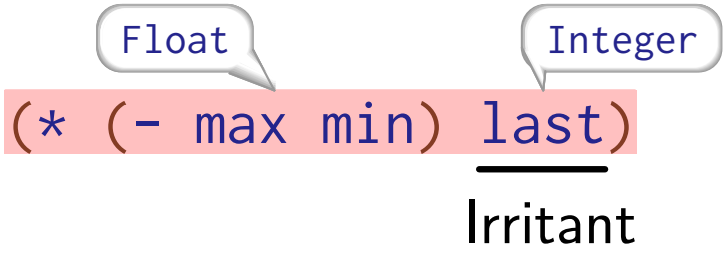
Too far, don't report



Optimization Analysis

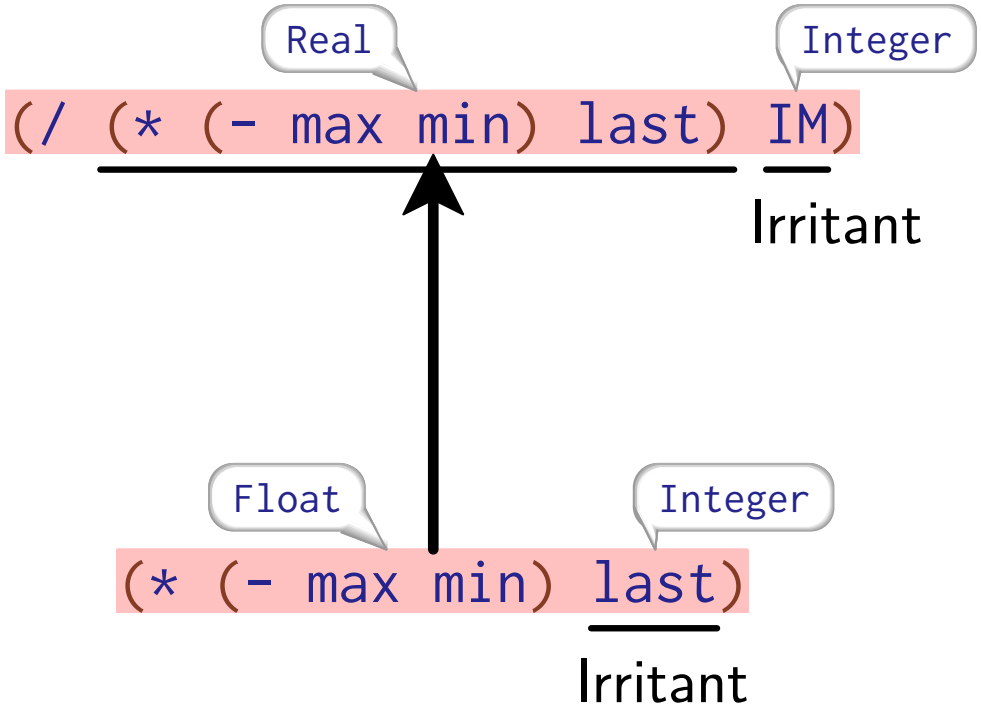
Causality merging





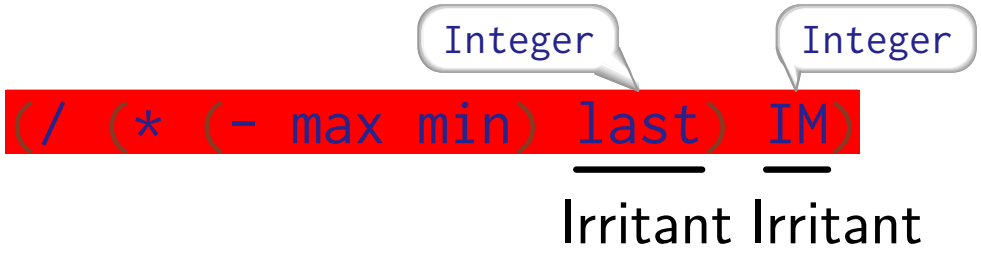
Optimization Analysis

Causality merging



Optimization Analysis

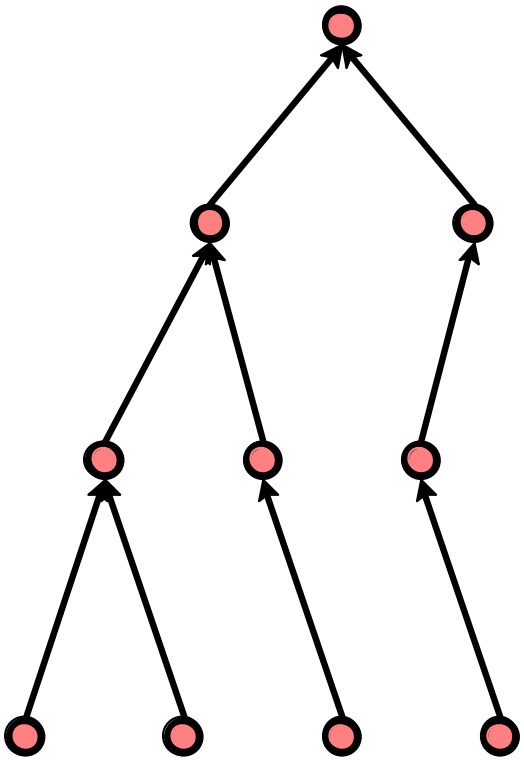
Causality merging



Badness $= 2$

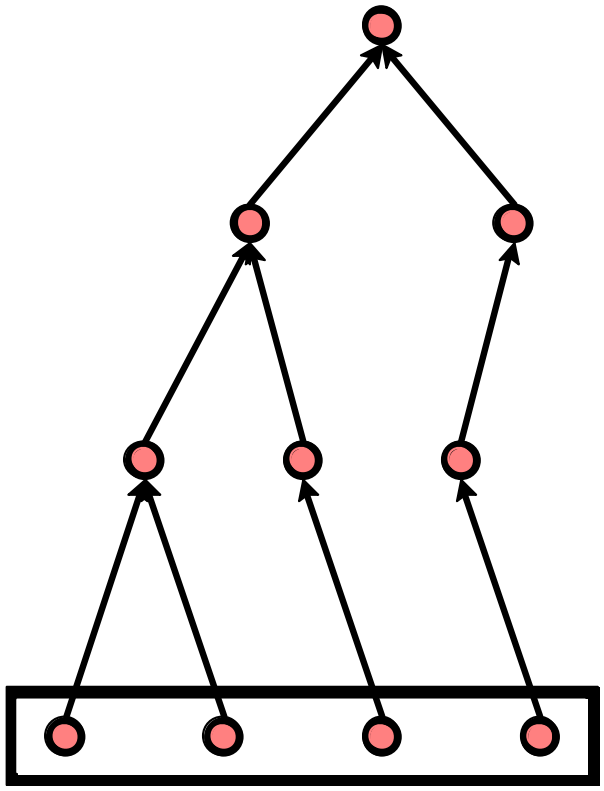
Optimization Analysis

Causality merging



Optimization Analysis

Causality merging

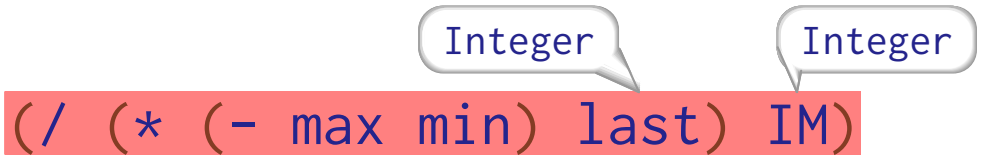


Near Miss = Root

Badness = # Nodes

Irritants = Leaves

Recommendation Generation



Float

Float ~~Integer~~

Irritant

12:5:

(/ (* (- max min) last) IM)

This expression has a Real type. The optimizer could optimize it if it had type Float. To fix, change the highlighted expression(s) to have Float type(s).

Programmer Response

```
                    (->fl last)  (->fl IM)
(+ (/ (* (- max min) last) IM) min)
```

12:5:

```
(/ (* (- max min) last) IM)
```

This expression has a Real type. The optimizer could optimize it if it had type Float. To fix, change the highlighted expression(s) to have Float type(s).

The Racket inliner in one slide

Based on a design by Serrano

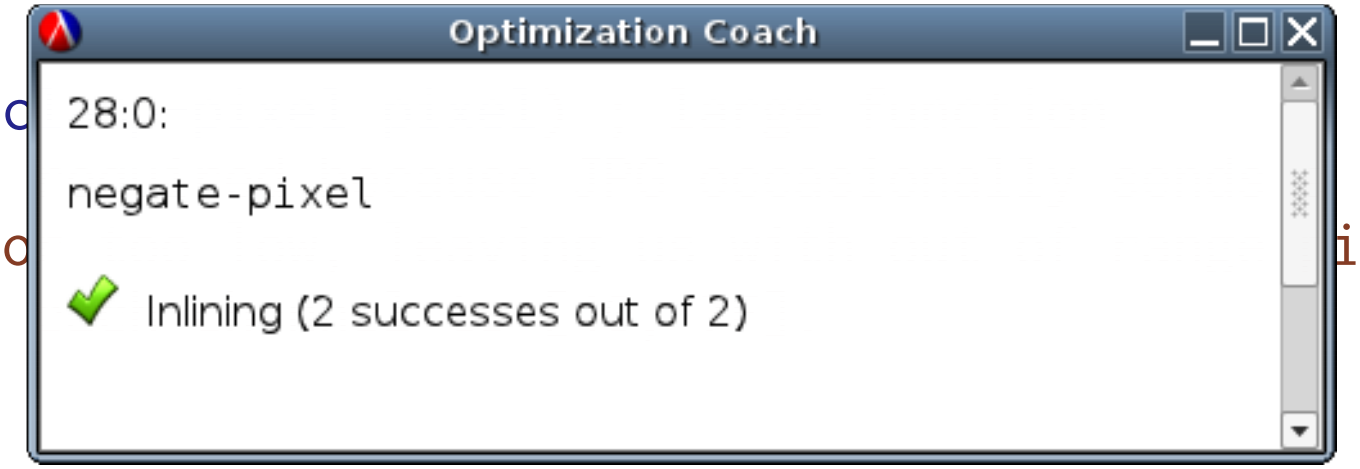
Fuel

- Inlining cost $\propto$ Function size
- Inlining stops when out of fuel

Loop unrolling for free!

```
(define (negate-pixel pixel) ; small function
  (+ #x010101 (bitwise-xor pixel #x3f3f3f)))


(define (clamp-pixel pixel) ; large function
  ; Clamp required because JPG occasionally sends a delta too
  ; high or too low, leaving us with out-of-range pixels.
  ; Clamp each channel to [40, 7F].
  ...)


(define (kernel-decode base-pixel delta-pixel)
  ... (negate-pixel ...) ...
  ... (clamp-pixel clampable) ...)


... (kernel-decode ...) ...
```

```
(define (negate-pixel pixel) ; small function
  (+ #x010101 (bitwise-xor pixel #x3f3f3f)))
```

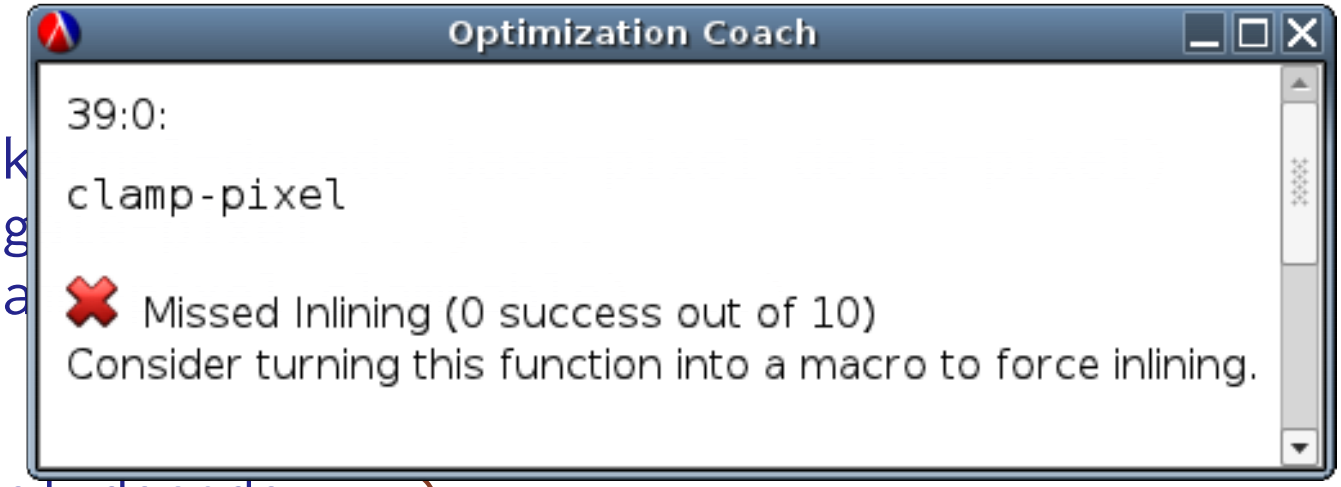



```
(define (c                                          delta too
  ; Clamp                                          ixels.
  ; high c
  ; Clamp
  ...)

(define (kernel-decode base-pixel delta-pixel)
  ... (negate-pixel ...) ...
  ... (clamp-pixel clampable) ...)


... (kernel-decode ...) ...
```

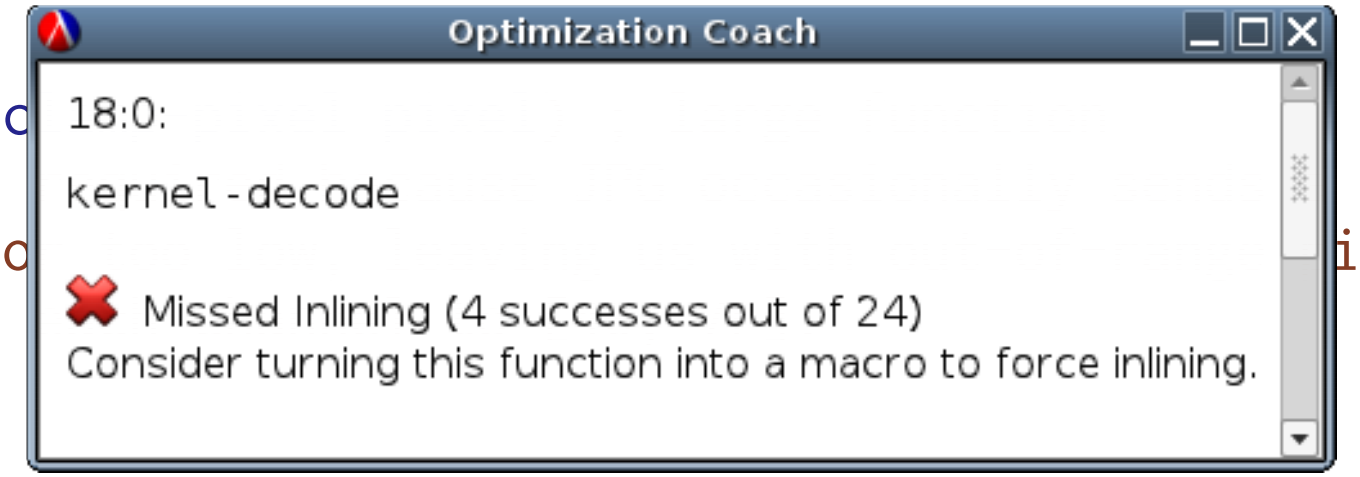
```
(define (negate-pixel pixel) ; small function
  (+ #x010101 (bitwise-xor pixel #x3f3f3f)))
```

```
(define (clamp-pixel pixel) ; large function
  ; Clamp required because JPG occasionally sends a delta too
  ; high or too low, leaving us with out-of-range pixels.
  ; Clamp each channel to [40, 7F].
  ...)
```

```
(define (k
  ... (neg
  ... (cla

... (kernel-decode ...) ...
```



```
(define (negate-pixel pixel) ; small function
  (+ #x010101 (bitwise-xor pixel #x3f3f3f)))

(define (c
  ; Clamp                                    delta too
  ; high o                                  ixels.
  ; Clamp
  ...)
```




```
(define (kernel-decode base-pixel delta-pixel)
  ... (negate-pixel ...) ...
  ... (clamp-pixel clampable) ...)

... (kernel-decode ...) ...
```

Compiler Instrumentation

```
inlining:
#(negate-pixel #<path:video.rkt> 28 0 793 73 #f)
in: video.rkt:18:0: kernel-decode
size: 1 fuel: 96


no inlining, out of fuel:
#(clamp-pixel #<path:video.rkt> 39 0 1228 534 #f)
in: video.rkt:18:0: kernel-decode
size: 99 fuel: 96
```

Compiler Instrumentation

```
inlining:
#(negate-pixel #<path:video.rkt> 28 0 793 73 #f)
in: video.rkt:18:0: kernel-decode
size: 1 fuel: 96
```

```
no inlining, out of fuel:
#(clamp-pixel #<path:video.rkt> 39 0 1228 534 #f)
in: video.rkt:18:0: kernel-decode
size: 99 fuel: 96
```

Compiler Instrumentation

```
inlining:
#(negate-pixel #<path:video.rkt> 28 0 793 73 #f)
in: video.rkt:18:0: kernel-decode
size: 1 fuel: 96


no inlining, out of fuel:
#(clamp-pixel #<path:video.rkt> 39 0 1228 534 #f)
in: video.rkt:18:0: kernel-decode
size: 99 fuel: 96
```

Compiler Instrumentation

```
inlining:
#(negate-pixel #<path:video.rkt> 28 0 793 73 #f)
in: video.rkt:18:0: kernel-decode
size: 1 fuel: 96
```

```
no inlining, out of fuel:
#(clamp-pixel #<path:video.rkt> 39 0 1228 534 #f)
in: video.rkt:18:0: kernel-decode
size: 99 fuel: 96
```

Optimization Analysis

Harmless Failure Pruning

```
no inlining, out of fuel:
#(for-loop #<path:video.rkt> 63 2 2353 281 #f)
in: video.rkt:63:2: for-loop
size: 52 fuel: 8
```

Unrolling has to stop at some point

Optimization Analysis

Locality Merging

4× `inlining: #(kernel-decode #<path:video.rkt> 18 0 427 276 #f)`

20× `no inlining, out of fuel: #(kernel-decode #<path:video.rkt> 18 0 427 276 #f)`

$4 < 20 \rightarrow$ Optimization failure

$\Delta = \overline{\text{(size - fuel)}}$

Recommendation Generation

No unrollings:	function $\rightarrow$ macro
Otherwise:	make smaller / break into pieces
Just over limit:	**`begin-encourage-inline`**
Fast/slow path:	break off slow path
Opt./kw. args:	multiple specialized functions

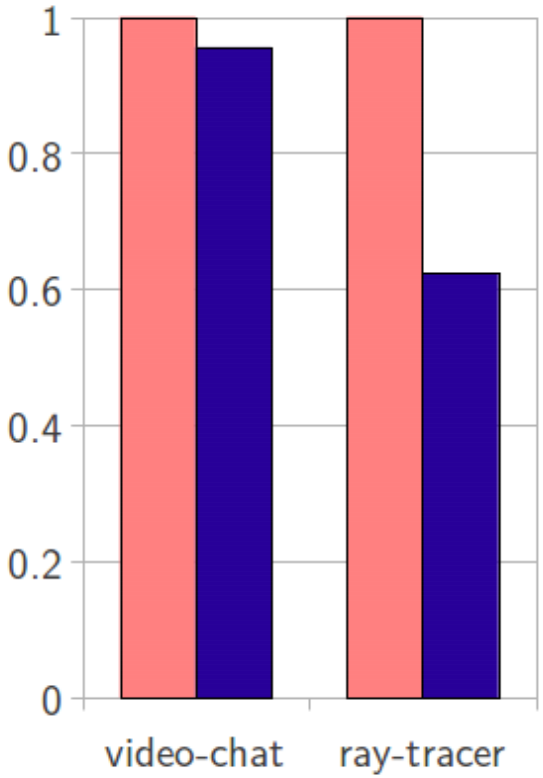
Programmer Response

```
define-syntax-rule
(define (clamp-pixel pixel) ; large function
  ; Clamp required because JPG occasionally sends a delta too
  ; high or too low, leaving us with out-of-range pixels.
  ; Clamp each channel to [40, 7F].
  ...)

define-syntax-rule
(define (kernel-decode base-pixel delta-pixel)
  ... (negate-pixel ...) ...
  ... (clamp-pixel clampable) ...)
```

How well does it work?

Baseline: Non-optimized

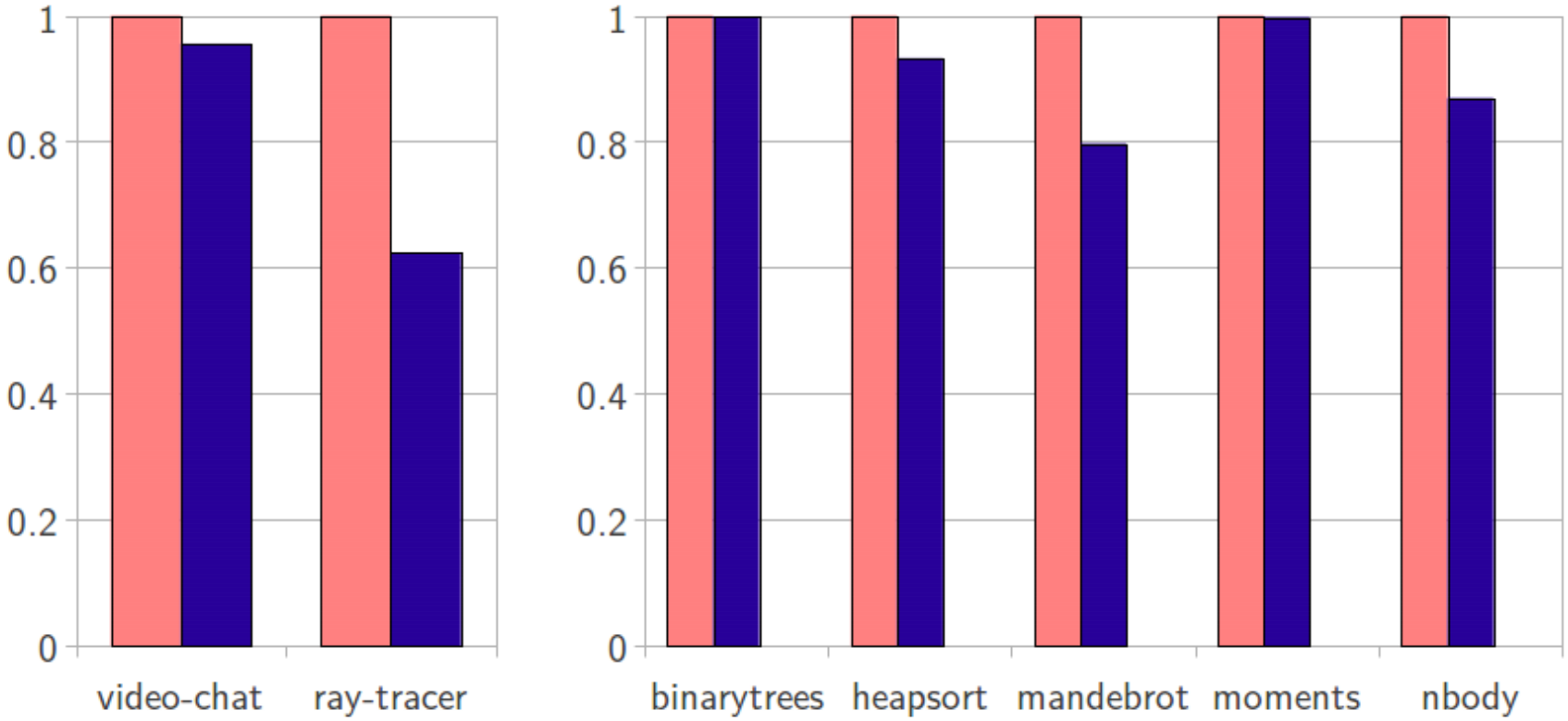
Coached: Followed recommendations (Minutes of work)



Lower is better

Baseline: Non-optimized

Coached: Followed recommendations (Minutes of work)

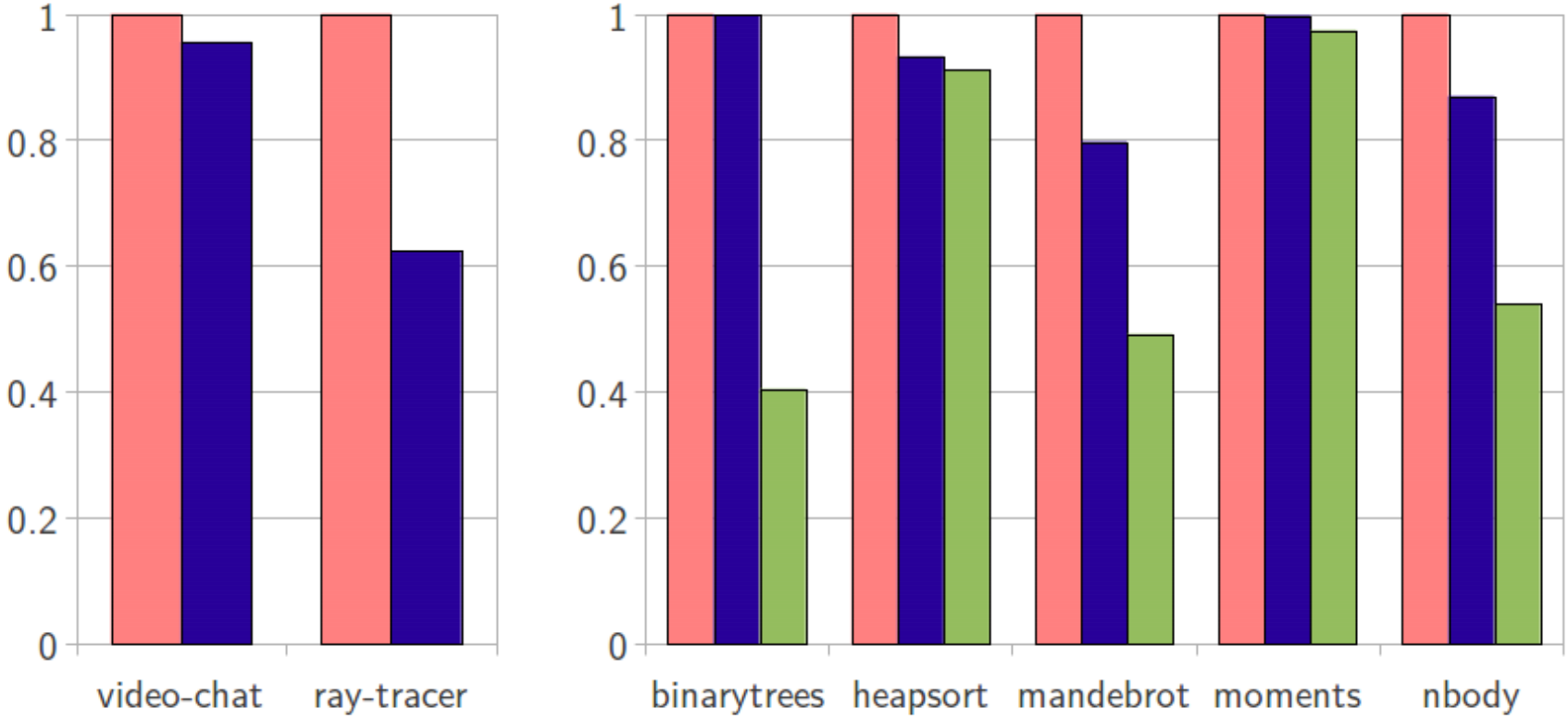


Lower is better

Baseline: Non-optimized

Coached: Followed recommendations (Minutes of work)

Gold standard: Hand-optimized by experts (Days of work)



Lower is better

How to extend it?

Our prototype {

- Type-driven rewriting
- Inlining

Common subexpression elimination

Test reordering

Scalar replacement

Loop-invariant code motion

Devirtualization

Specialization of polymorphic containers

Case-of-case transformation

Our prototype {

Type-driven rewriting

Inlining

Common subexpression elimination

Test reordering

Scalar replacement

Loop-invariant code motion

Devirtualization

Specialization of polymorphic containers

Case-of-case transformation

Coaching Recipe

- Log successes and failures
- Define optimization analysis metrics
- Add recommendation generation logic

Scalar replacement

```
(define twiddle-c
  (* c (exp (/ (* pi 0.0+1.0i
                  (->fl k))
               n/2))))
```

```
(define real-c (real-part c))
(define imag-c (imag-part c))
...
(define real-twiddle-c ...)
(define imag-twiddle-c ...)
```

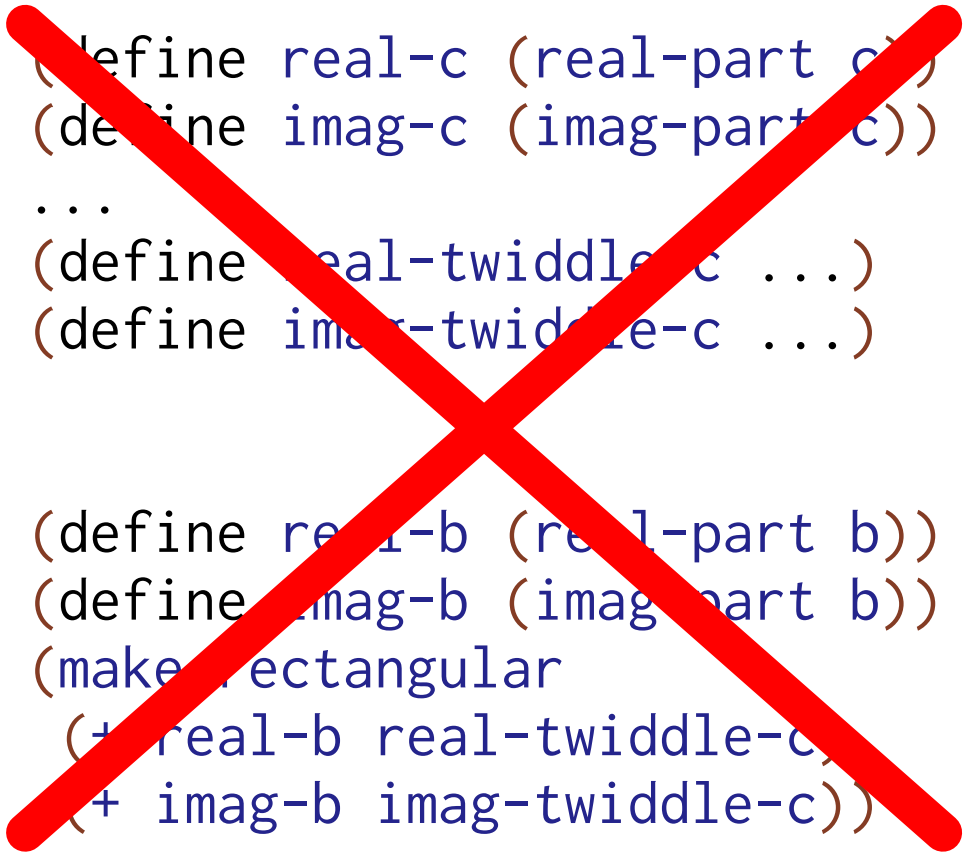
```
(+ b twiddle-c)
```

```
(define real-b (real-part b))
(define imag-b (imag-part b))
(make-rectangular
 (+ real-b real-twiddle-c)
 (+ imag-b imag-twiddle-c))
```

Scalar replacement

```
(define twiddle-c
  (* c (exp (/ (* pi 0.0+1.0i
                  (->fl k))
               n/2))))
```

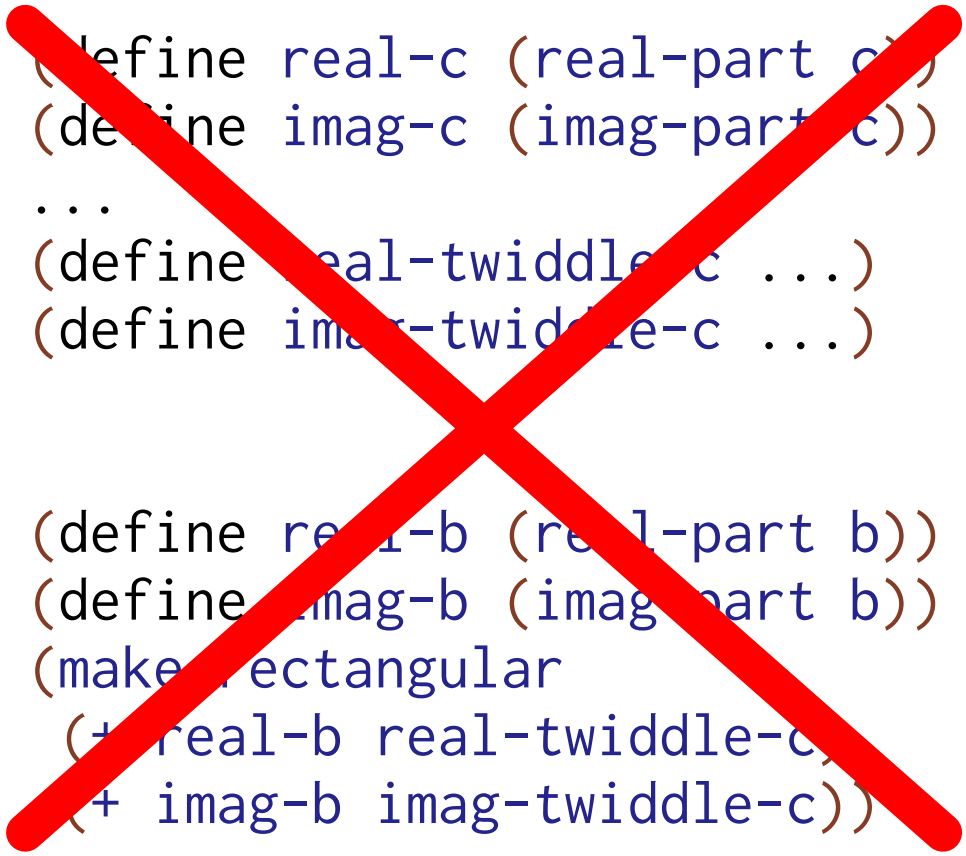
```
(printf "~a\n" twiddle-c)
(+ b twiddle-c)
```



Scalar replacement

```
(define twiddle-c
  (* c (exp (/ (* pi 0.0+1.0i
                  (->fl k))
                n/2))))
```

```
(printf "~a\n" twiddle-c)
(+ b twiddle-c)
```



$$\Delta = (\#\ \text{boxed uses}) \,/\, (\#\ \text{unboxed uses})$$

$$\text{Irritants} = \{\text{boxed uses}\}$$

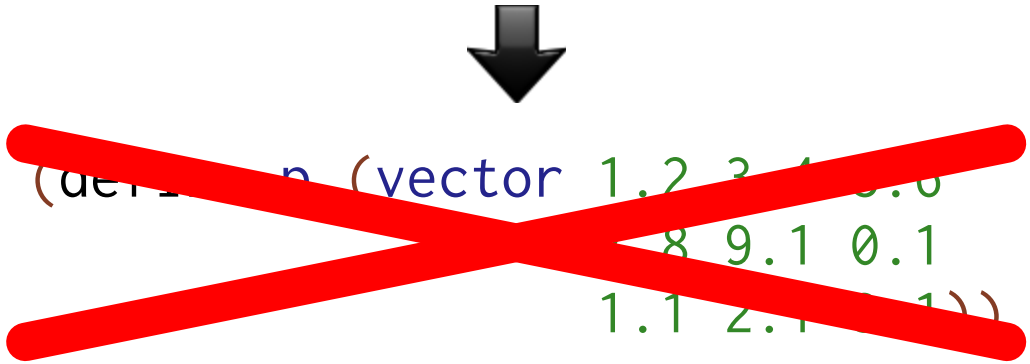
Specialization of polymorphic containers

```
(define-type 3D-path
  (Vectorof (List Float Float Float)))

(: p : 3D-path)
(define p (vector '(1.2 3.4 5.6)
                  '(7.8 9.1 0.1)
                  '(1.1 2.1 3.1)))
```

```
(define p (vector 1.2 3.4 5.6
                  7.8 9.1 0.1
                  1.1 2.1 3.1))
```

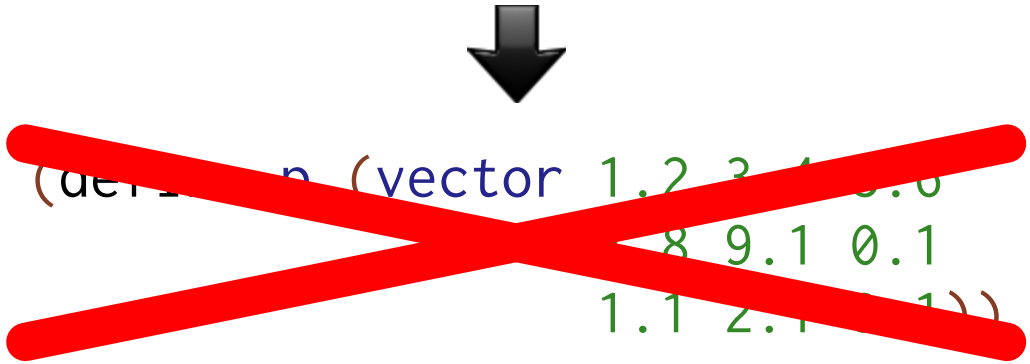
Specialization of polymorphic containers

```
(define-type 4D-path
  (Vectorof (List Float Float Float Float)))

(: p : 4D-path)
(define p (vector '(1.2 3.4 5.6 12.3)
                  '(7.8 9.1 0.1 45.6)
                  '(1.1 2.1 3.1 78.9)))
```



Specialization of polymorphic containers

```
(define-type 4D-path
  (Vectorof (List Float Float Float Float)))

(: p : 4D-path)
(define p (vector '(1.2 3.4 5.6 12.3)
                  '(7.8 9.1 0.1 45.6)
                  '(1.1 2.1 3.1 78.9)))
```



Δ = (element size) - (max optimized element size)

Conclusion

The take-away

Key idea: The compiler *talks back*

General optimization analysis techniques

\+ *Optimization-specific* heuristics

Targeted recommendations

The take-away

Key idea: The compiler *talks back*

General optimization analysis techniques

\+ *Optimization-specific* heuristics

Targeted recommendations

`racket-lang.org`

Demo