```
(struct state (point σ κ) #:transparent)
(struct ar (e ρ) #:transparent)
(struct fn (v) #:transparent)
(struct mt () #:transparent)
(struct rt (ctx) #:transparent)
(struct ref (x) #:transparent)
(struct app (e0 e1) #:transparent)
(struct lam (x e) #:transparent)
(define-values (F R Seen M Ξ)
     (values (mutable-set) (mutable-set) (mutable-set)
             (make-hash) (make-hash)))
(define (hash-add h k v)
  (hash-set h k (set-add (hash-ref h k (set)) v)))
(define (hash-add! h k v)
  (hash-set! h k (set-add (hash-ref h k (set)) v)))
(define (add-state! s)
  (unless (set-member? Seen s)
    (set-add! F s)
    (set-add! Seen s)))
(define (add-reduction! s0 s1)
  (set-add! R (cons s0 s1))
  (add-state! s1))
(define (step s)
  (match s
   [(state (cons (ref x) ρ) σ κ)
    (for ([v (in-set (hash-ref σ (hash-ref ρ x)))])
      (add-reduction! s (state v σ κ)))]
   [(state (cons (app e0 e1) ρ) σ κ)
    (add-reduction! s
     (state (cons e0 ρ) σ (cons (ar e1 ρ) κ)))]
   [(state v σ (cons (ar e ρ) κ))
    (add-reduction! s
     (state (cons e ρ) σ (cons (fn v) κ)))]
   [(state v σ (cons (fn (cons (lam x e) ρ)) κ))
    (define a (alloc s))
    (define ρ* (hash-set ρ x a))
    (define σ* (hash-add σ a v))
    (define ctx (list e ρ* σ*))
    (hash-add! Ξ ctx κ)
    (match (hash-ref M ctx #f)
      [#f
       (add-reduction! s
        (state (cons e ρ*) σ* (rt ctx)))]
      [results
       (for ([r (in-set results)])
         (match-define (cons v* σ**) r)
         (add-state! (state v* σ** κ)))])]
   [(state v σ (rt ctx))
    (hash-add! M ctx (cons v σ))
    (for ([κ (in-set (hash-ref Ξ ctx))])
      (add-reduction! s (state v σ κ)))]))
(define (analyze e)
  (set-clear! F) (set-clear! R) (set-clear! Seen)
  (hash-clear! Ξ) (hash-clear! M)
  (set-add! F (state (cons e (hash)) (hash) (mt)))
  (let loop ()
    (unless (set-empty? F)
      (define ς (set-first F))
      (set-remove! F ς)
      (step ς)
      (loop)))
  'the final system
  (list R M Ξ))
```