Abstracting Definitional Interpreters

David Van Horn

Abstracting Definitional Interpreters

David Van Horn — *Northeastern University*

Definitional interpreters written in monadic style can express a wide variety of interpreters for languages with effects. In this talk, we show that such interpreters, under a slight reworking, can also express a diverse set of abstract interpretations from flow analysis to symbolic execution.

We give a rational reconstruction of a definitional abstract interpreter for a higher-order language by constructing a series of units implementing monadic operations. We implement units realizing reachable state semantics, trace semantics, dead-code elimination, symbolic execution, and a finite store abstraction. The denouement of our story is a sound and computable abstract interpreter that arises from the composition of simple, independent components. Remarkably, this interpreter implements a form of pushdown control flow analysis (PDCFA) in which calls and returns are always properly matched in the abstract semantics. True to the definitional style of Reynolds, the evaluator involves no explicit mechanics to achieve this property; it is simply inherited from the defining language.

Warm-up:

```
(define (eval e ρ)
  (match e
    [(vbl x) (lookup ρ x)]
    [(app e0 e1)
     ((eval e0 ρ)
      (eval e1 ρ))]
    [(lam x e)
     (λ (v) (eval e (extend ρ x v)))]))
```

call-by-?

Warm-up:

```
(define (eval e ρ)
  (match e
    [(vbl x) (lookup ρ x)]
    [(app e0 e1)
     ((eval e0 ρ)
      (eval e1 ρ))]
    [(lam x e)
     (λ (v) (eval e (extend ρ x v)))]))
```

call-by-?

Yes.

Warm-up:

```
#lang lazy
(define (eval e ρ)
  (match e
    [(vbl x) (lookup ρ x)]
    [(app e0 e1)
     ((eval e0 ρ)
      (eval e1 ρ))]
    [(lam x e)
     (λ (v) (eval e (extend ρ x v)))]))
```

call-by-?

Yes.

Warm-up:

```
#lang racket
(define (eval e ρ)
  (match e
    [(vbl x) (lookup ρ x)]
    [(app e0 e1)
     ((eval e0 ρ)
      (eval e1 ρ))]
    [(lam x e)
     (λ (v) (eval e (extend ρ x v)))]))
```

call-by-?

Yes.

Warm-up:

```
#lang racket
(define (eval e ρ)
  (match e
    [(vbl x) (lookup ρ x)]
    [(app e0 e1)
     ((eval e0 ρ)
      (eval e1 ρ))]
    [(lam x e)
     (λ (v) (eval e (extend ρ x v)))]))
```

Left to right?

Call-by-?

Yes.

Warm-up:

#

(

> Definitional interpreters can inherit properties of the defining language.
>
> --Reynolds

call-by-?

Yes.

Warm-up:

#

(

> Definitional interpreters can inherit properties of the defining language.
>
> --Reynolds

abstract

^

call-by-?

Yes.

AN ABSTRACT INTERPRETER IS A SOUND (+ COMPUTABLE) APPROXIMATION OF EVALUATION

Contributions:

1) Abstract interpreters
   in Definitional Style

2) An observation on
   inherited properties

Outline:

* Constraint-based analyses
* Definitional abstract interpreters
* Inheriting the Control stack

Abstract Interpreters in Constraint Style

Introduction to Set Constraint-Based Program Analysis

Alexander Aiken*

SUB-EXP	CONSTRAINT
$\lambda x.e_0$	$\lambda_x \subseteq [\![\lambda x.e_0]\!]$
$e_1\ e_2$	for every $\lambda x.e_3$ in e $\lambda_x \subseteq [\![e_1]\!] \Rightarrow ([\![e_2]\!] \subseteq [\![x]\!] \ \wedge \ [\![e_3]\!] \subseteq [\![e_1\ e_2]\!])$

Introduction to Set Constraint-Based Program Analysis

Alexander Aiken*

SUB-EXP	CONSTRAINT
$\lambda x.e_0$	$\lambda_x \subseteq [\![\lambda x.e_0]\!]$
$e_1\ e_2$	for every $\lambda x.e_3$ in e $\lambda_x \subseteq [\![e_1]\!] \Rightarrow ([\![e_2]\!] \subseteq [\![x]\!] \ \wedge \ [\![e_3]\!] \subseteq [\![e_1\ e_2]\!])$

BUT WHERE IS THE INTERPRETER?

Abstract Interpreters in Definitional Style

```
(import unit^ bind^ rec^ δ^ env^)
(export ev^)

(define (ev e ρ)
  (match e
    [(vbl x) (get ρ x)]
    [(num n) (unit n)]
    [(lam x e) (unit (cons (lam x e) ρ))]
    [(ifz e0 e1 e2)
     (do v ← (rec e0 ρ)
       (match v
         [0 (rec e1 ρ)]
         [n (rec e2 ρ)]))]
    [(op1 o e0)
     (do v ← (rec e0 ρ)
       (δ o v))]
    [(op2 o e0 e1)
     (do v0 ← (rec e0 ρ)
         v1 ← (rec e1 ρ)
       (δ o v0 v1))]
    [(lrc f (lam x e0) e)
     (do a ← (ralloc f (cons (lam x e0) ρ))
       (rec e (extend-env ρ f a)))]
    [(app e0 e1)
     (do v0 ← (rec e0 ρ)
         v1 ← (rec e1 ρ)
       (match v0
         [(cons (lam x e) ρ0)
          (do a ← (alloc v0 v1)
            (rec e (extend-env ρ0 x a)))]))]))
```

ev@

DEFINITIONAL INTERPRETER IN MONADIC STYLE W/ EXPLICIT FIXPOINT

"DEMO"

```
#lang monadic-eval
(link eval@ ev@ δ@ env-sto@)
```

RUN OF THE MILL EVALUATOR

eval@

```
(import ev^)
(export eval^ unit^ rec^ bind^ err^)

(define (eval e) ((ev e (hash)) (hash)))
(define (rec e r) (ev e r))
(define ((unit v) s) (cons v s))
(define ((err) s) (cons 'err s))
(define ((bind a f) s)
  (match (a s)
    [(cons 'err s) (cons 'err s)]
    [(cons v s)    ((f v) s)]))
```

δ@

```
(import unit^ err^)
(export δ^)
(define (δ o . vs)
  (match* (o vs)
    [('add1 (list n))    (unit (add1 n))]
    [('sub1 (list n))    (unit (sub1 n))]
    [('- (list n))       (unit (- n))]
    [('+ (list n1 n2))   (unit (+ n1 n2))]
    [('- (list n1 n2))   (unit (- n1 n2))]
    [('* (list n1 n2))   (unit (* n1 n2))]
    [('quotient (list n1 n2))
     (if (zero? n2) (err) (quotient n1 n2))]))
```

INTERPRETATION OF PRIMITIVES

env-sto@

```
(import unit^)
(export env^)

(define ((get ρ x) σ)
  ((unit (hash-ref σ (hash-ref ρ x))) σ))

(define ((alloc f v) σ)
  (match f
    [(cons (lam x e) ρ)
     (define a (next σ))
     ((unit a) (update-sto σ a v))]))

(define ((ralloc x v) σ)
  (match v
    [(cons e ρ)
     (define a (next σ))
     ((unit a)
      (update-sto σ a
        (cons e (hash-set ρ x a))))]))
```

ENVIRONMENT + HEAP

eg-eval.rkt – DrRacket

eg-eval.rkt (define ...) Macro Stepper Run Stop

```
#lang monadic-eval
(link eval@ ev@ δ@ env-sto@)
(rec fact
  (λ (n)
    (if0 n
         1
         (* n (fact (- n 1))))))
  (fact 5))
```

Welcome to DrRacket, version 5.3.3.5--2013-02-25(08800641/d) [3m].
Language: monadic-eval; memory limit: 128 MB.

```
'(120
  .
  #hash((0
         .
         ((lam
           n
           (ifz
            (vbl n)
            (num 1)
            (op2
             *
             (vbl n)
             (app
              (vbl fact)
              (op2 - (vbl n) (num 1))))))
          .
          #hash((fact . 0))))
        (1 . 5)
        (2 . 4)
        (3 . 3)
        (4 . 2)
        (5 . 1)
        (6 . 0)))
>
```

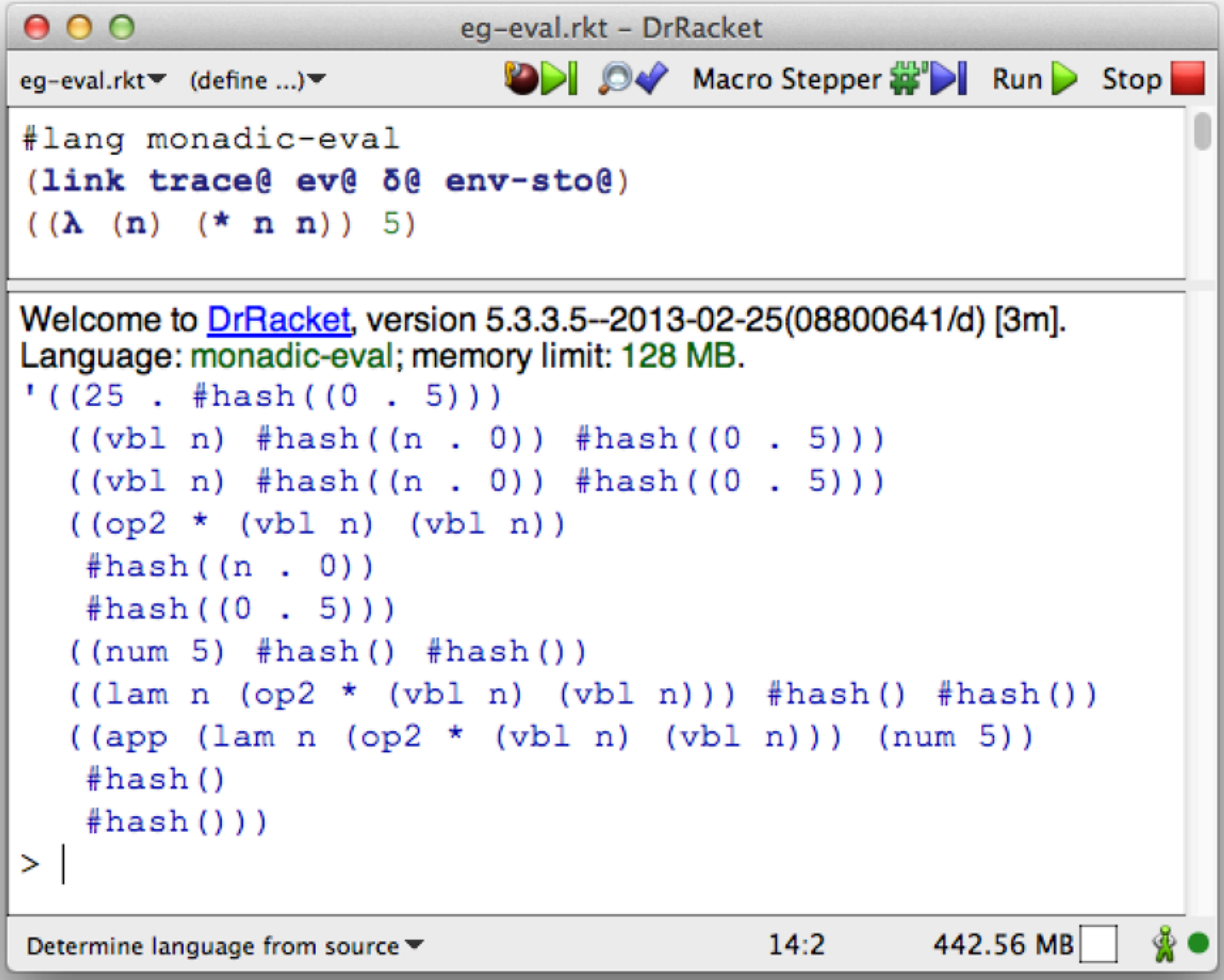
Determine language from source 9:0 489.74 MB

```
#lang monadic-eval
(link trace@ ev@ δ@ env-sto@)
```

TRACE SEMANTICS

trace@

```
(import ev^)
(export eval^ unit^ bind^ rec^ err^)

(define (eval e) (((rec e (hash)) (hash)) empty))

(define (((rec e r) s) t)
  (((ev e r) s) (cons (list e r s) t)))

(define (((unit v) s) t) (cons (cons v s) t))
(define (((err) s) t) (cons (cons 'err s) t))
(define (((bind a f) s) t)
  (match ((a s) t)
    [(cons (cons 'err s) t) (cons (cons 'err s) t)]
    [(cons (cons v s) t)
     (((f v) s) t)]))
```

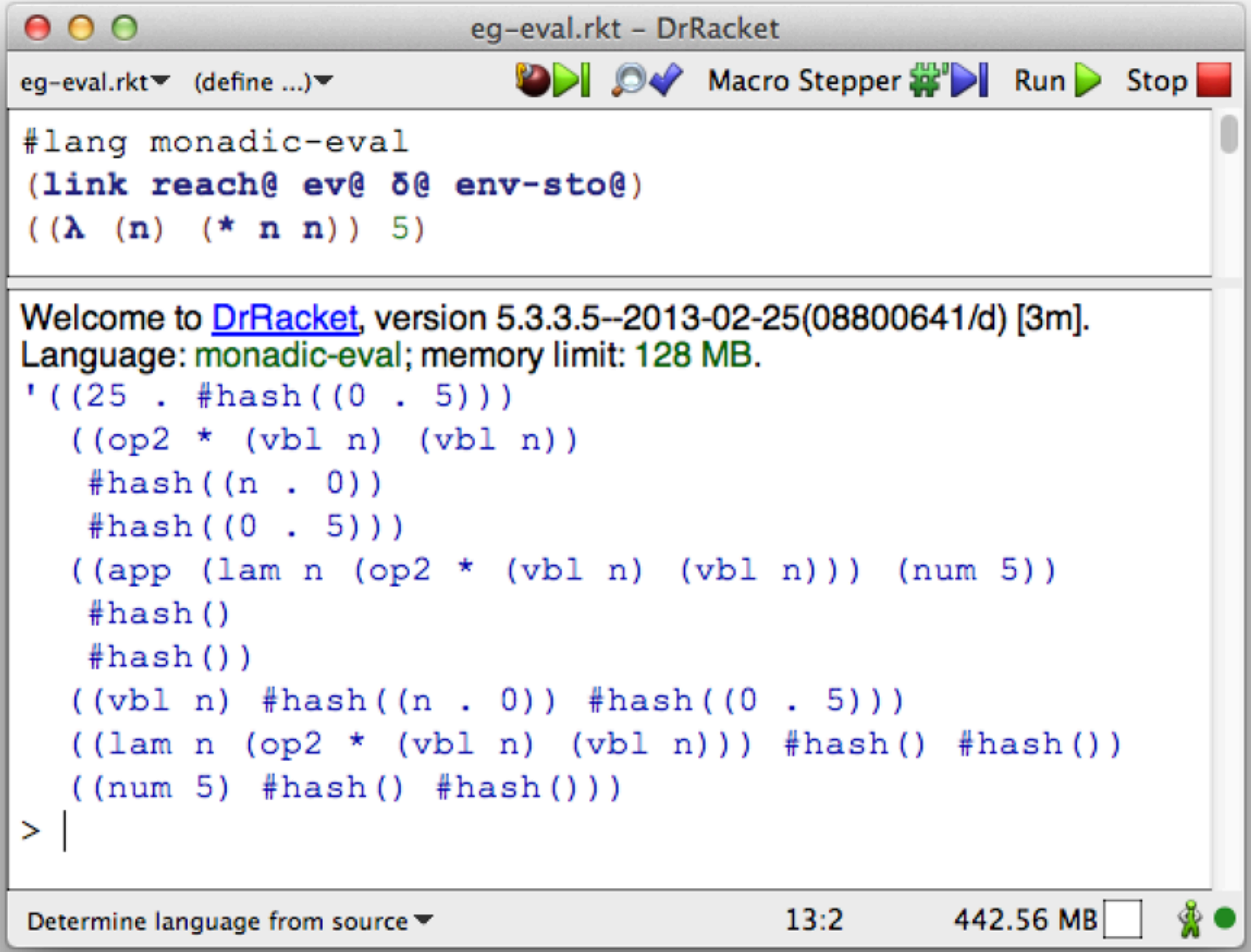

eg-eval.rkt – DrRacket
eg-eval.rkt
(define ...)
Macro Stepper
Run
Stop
#lang monadic-eval
(link trace@ ev@ δ@ env-sto@)
((λ (n) (* n n)) 5)
Welcome to DrRacket, version 5.3.3.5--2013-02-25(08800641/d) [3m].
Language: monadic-eval; memory limit: 128 MB.
'((25 . #hash((0 . 5)))
((vbl n) #hash((n . 0)) #hash((0 . 5)))
((vbl n) #hash((n . 0)) #hash((0 . 5)))
((op2 * (vbl n) (vbl n))
#hash((n . 0))
#hash((0 . 5)))
((num 5) #hash() #hash())
((lam n (op2 * (vbl n) (vbl n))) #hash() #hash())
((app (lam n (op2 * (vbl n) (vbl n))) (num 5))
#hash()
#hash()))
>
Determine language from source
14:2
442.56 MB

```
#lang monadic-eval
(link reach@ ev@ δ@ env-sto@)
```

REACHABLE STATE SEMANTICS

reach@

```
(import ev^)
(export eval^ unit^ bind^ rec^ err^)

(define (eval e)
  (match (((rec e (hash)) (hash)) (set))
    [(cons v reach)
     (cons v (set->list reach))]))

(define (((rec e r) s) t)
  (((ev e r) s) (set-add t (list e r s))))

(define (((unit v) s) t) (cons (cons v s) t))
(define (((err) s) t) (cons (cons 'err s) t))
(define (((bind a f) s) t)
  (match ((a s) t)
    [(cons (cons 'err s) t) (cons (cons 'err s) t)]
    [(cons (cons v s) t)
     (((f v) s) t)]))
```

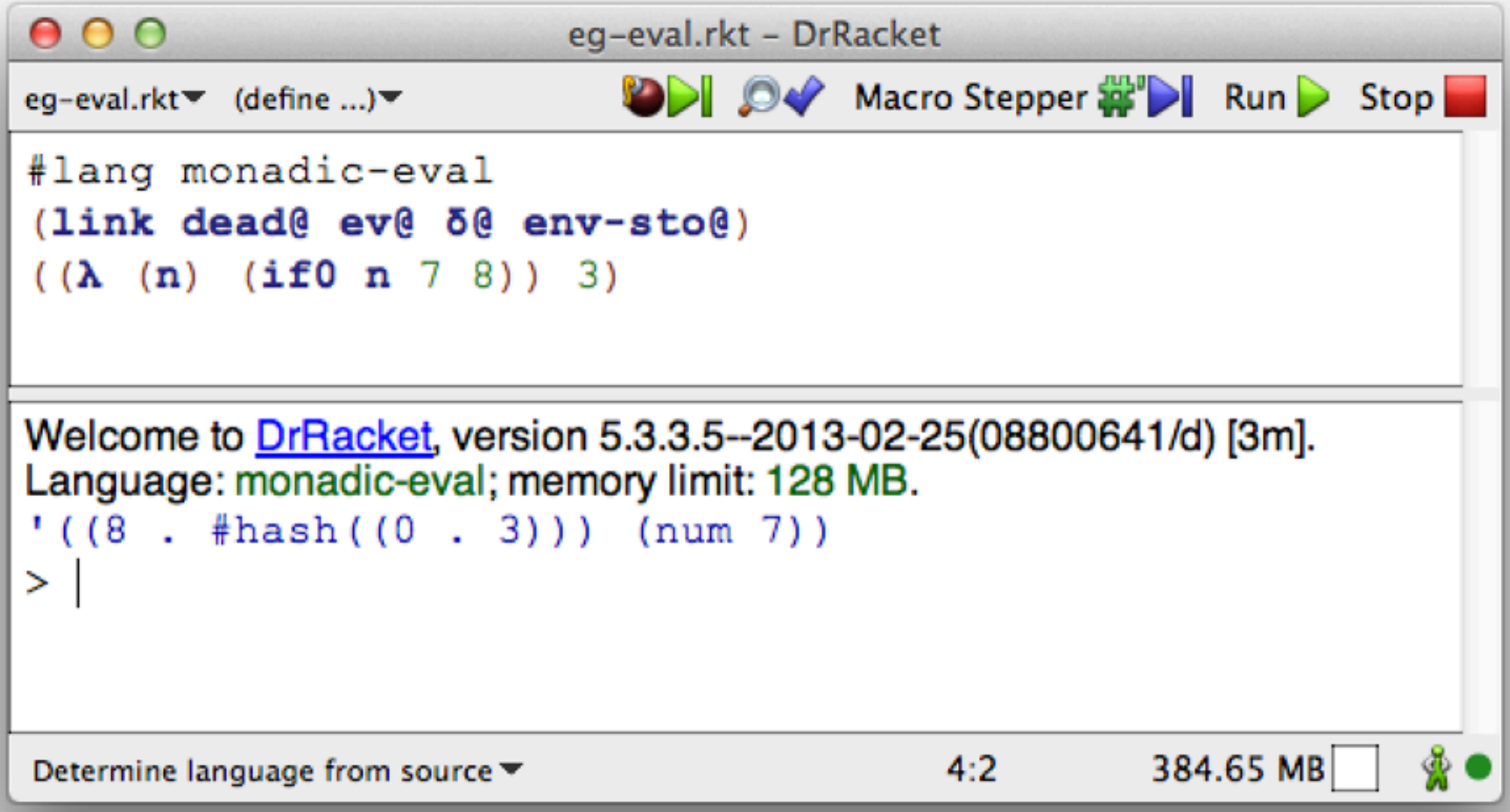

eg-eval.rkt – DrRacket
eg-eval.rkt
(define ...)
Macro Stepper
Run
Stop
#lang monadic-eval
(link reach@ ev@ δ@ env-sto@)
((λ (n) (* n n)) 5)
Welcome to DrRacket, version 5.3.3.5--2013-02-25(08800641/d) [3m].
Language: monadic-eval; memory limit: 128 MB.
'((25 . #hash((0 . 5)))
((op2 * (vbl n) (vbl n))
#hash((n . 0))
#hash((0 . 5)))
((app (lam n (op2 * (vbl n) (vbl n))) (num 5))
#hash()
#hash())
((vbl n) #hash((n . 0)) #hash((0 . 5)))
((lam n (op2 * (vbl n) (vbl n))) #hash() #hash())
((num 5) #hash() #hash()))
>
Determine language from source
13:2
442.56 MB

```
#lang monadic-eval
(link dead@ ev@ δ@ env-sto@)
```

DEAD CODE SEMANTICS

dead@

```
(import ev^)
(export eval^ unit^ bind^ rec^ err^)

(define (eval e)
  (match (((rec e (hash)) (hash)) (subexps e))
    [(cons v dead)
     (cons v (set->list dead))]))

(define (((rec e r) s) t)
  (((ev e r) s) (set-remove t e)))

(define (((unit v) s) t) (cons (cons v s) t))
(define (((err) s) t) (cons (cons 'err s) t))
(define (((bind a f) s) t)
  (match ((a s) t)
    [(cons (cons 'err s) t) (cons (cons 'err s) t)]
    [(cons (cons v s) t)
     (((f v) s) t)]))
```



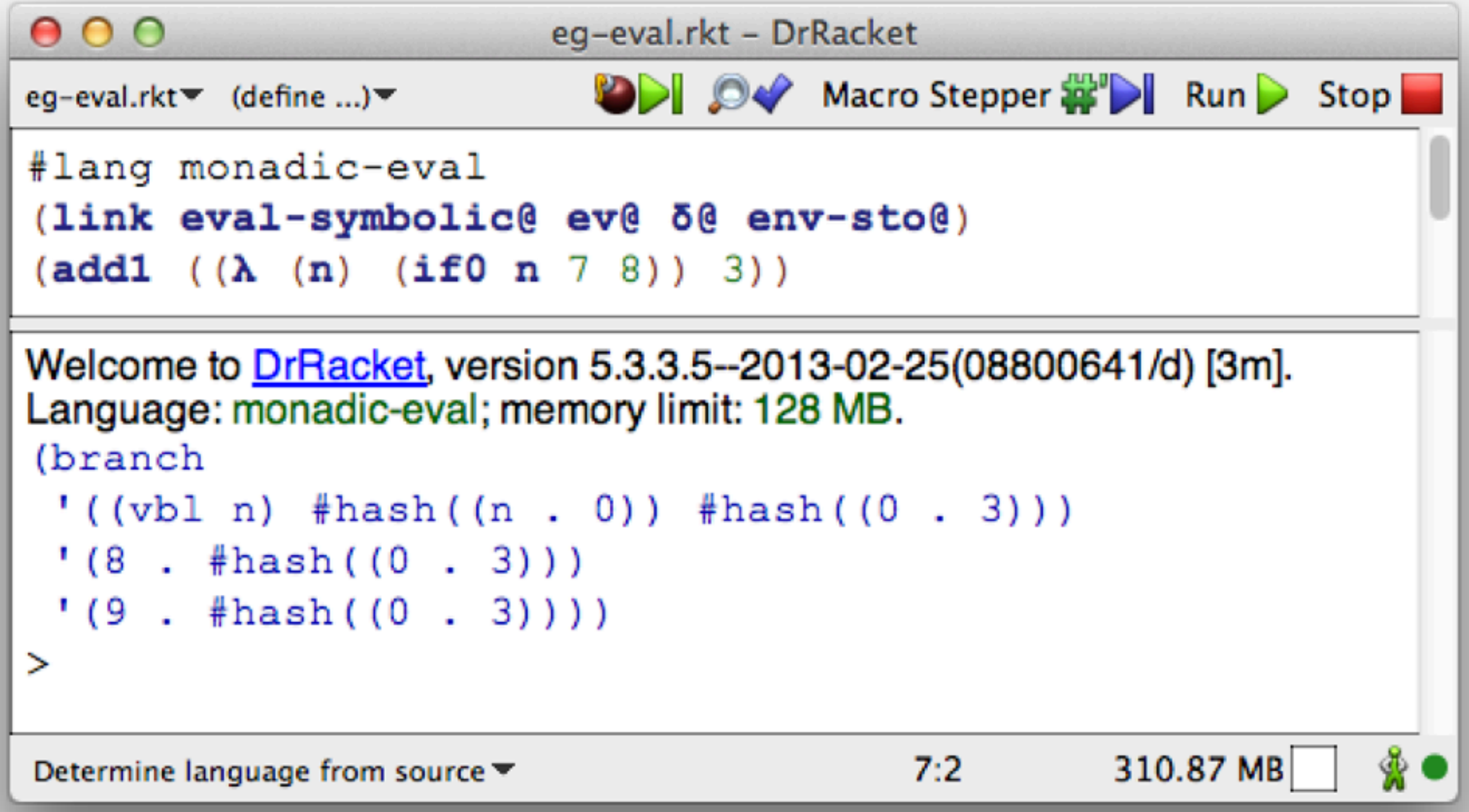
ANSWER IS 8; 7 IS DEAD

```
#lang monadic-eval
(link eval-symbolic@ ev@ δ@ env-sto@)
```

KING-STYLE SYMBOLIC EXECUTION

eval-symbolic@

```
(import ev^)
(export eval^ unit^ bind^ rec^ err^)

(struct branch (vc t f) #:transparent)

(define (eval e) ((rec e (hash)) (hash)))

(define ((rec e r) s)
  (match e
    [(ifz e0 e1 e2)
     (branch (list e0 r s)
             ((rec e1 r) s)
             ((rec e2 r) s))]
    [_  ((ev e r) s)]))

(define ((unit v) s) (cons v s))
(define ((err) s) (cons 'err s))
(define ((bind a f) s)
  (let loop ([res (a s)])
    (match res
      [(branch vc a1 a2)
       (branch vc (loop a1) (loop a2))]
      [(cons 'err s) (cons 'err s)]
      [(cons v s)
       ((f v) s)])))
```


eg-eval.rkt – DrRacket
eg-eval.rkt
(define ...)
Macro Stepper
Run
Stop
#lang monadic-eval
(link eval-symbolic@ ev@ δ@ env-sto@)
(add1 ((λ (n) (if0 n 7 8)) 3))
Welcome to DrRacket, version 5.3.3.5--2013-02-25(08800641/d) [3m].
Language: monadic-eval; memory limit: 128 MB.
(branch
'((vbl n) #hash((n . 0)) #hash((0 . 3)))
'(8 . #hash((0 . 3)))
'(9 . #hash((0 . 3))))
>
Determine language from source
7:2
310.87 MB

```
#lang monadic-eval
(link eval-symbol@ ev@ abs-δ@ env-sto@)
```

SYMBOLIC VALUES +

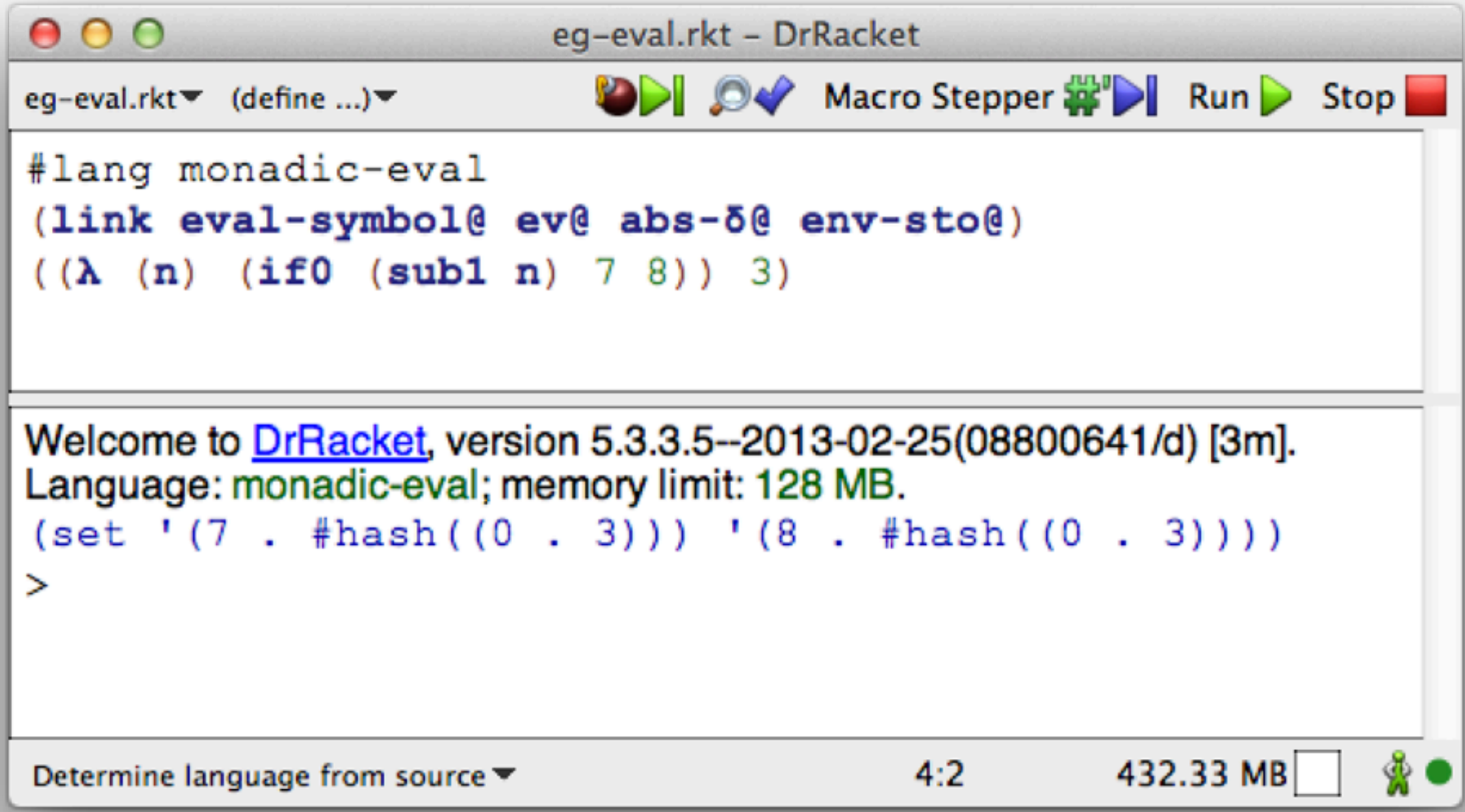
ABSTRACT OPERATIONS

eval-symbol@

```
(import ev^)
(export eval^ unit^ bind^ rec^ err^)

(define (eval e) ((rec e (hash)) (hash)))

(define ((unit v) s) (set (cons v s)))
(define ((err) s) (set (cons (set 'err s))))

(define ((both c0 c1) s)
  (set-union (c0 s) (c1 s)))

(define (rec e r)
  (match e
    [(ifz e0 e1 e2)
     (do v ← (rec e0 r)
       (match v
         [0            (ev e1 r)]
         [(? number?)  (ev e2 r)]
         [(? symbolic?)
          (both (ev e1 r)
                (ev e2 r))]))]
    [_  (ev e r)]))


(define ((bind a f) s)
  (for*/fold ([rs (set)])
    ([ans (a s)])
    (set-union rs
      (match ans
        [(cons 'err s)
         (set (cons 'err s))]
        [(cons v s)
         ((f v) s)]))))
```

abs-δ@

```
(import unit^ symbolic^ err^)
(export δ^)
(define (δ o . vs)
  (match* (o vs)
    [('add1 (list n))   (unit 'N)]
    [('sub1 (list n))   (unit 'N)]
    [('+ (list n1 n2)) (unit 'N)]
    [('- (list n1 n2)) (unit 'N)]
    [('* (list n1 n2)) (unit 'N)]
    [('quotient (list n1 (? number? n2)))
     (if (zero? n2)
         (err)
         (unit 'N))]
    [('quotient (list n1 n2))
     (both (unit 'N) (err))]))
```


eg-eval.rkt – DrRacket
eg-eval.rkt
(define ...)
Macro Stepper
Run
Stop
#lang monadic-eval
(link eval-symbol@ ev@ abs-δ@ env-sto@)
((λ (n) (if0 (sub1 n) 7 8)) 3)
Welcome to DrRacket, version 5.3.3.5--2013-02-25(08800641/d) [3m].
Language: monadic-eval; memory limit: 128 MB.
(set '(7 . #hash((0 . 3))) '(8 . #hash((0 . 3))))
>
Determine language from source
4:2
432.33 MB

EITHER 7 OR 8

```
#lang monadic-eval
(link eval-symbol@ ev@ abs-δ@ sto-0cfa@)
```

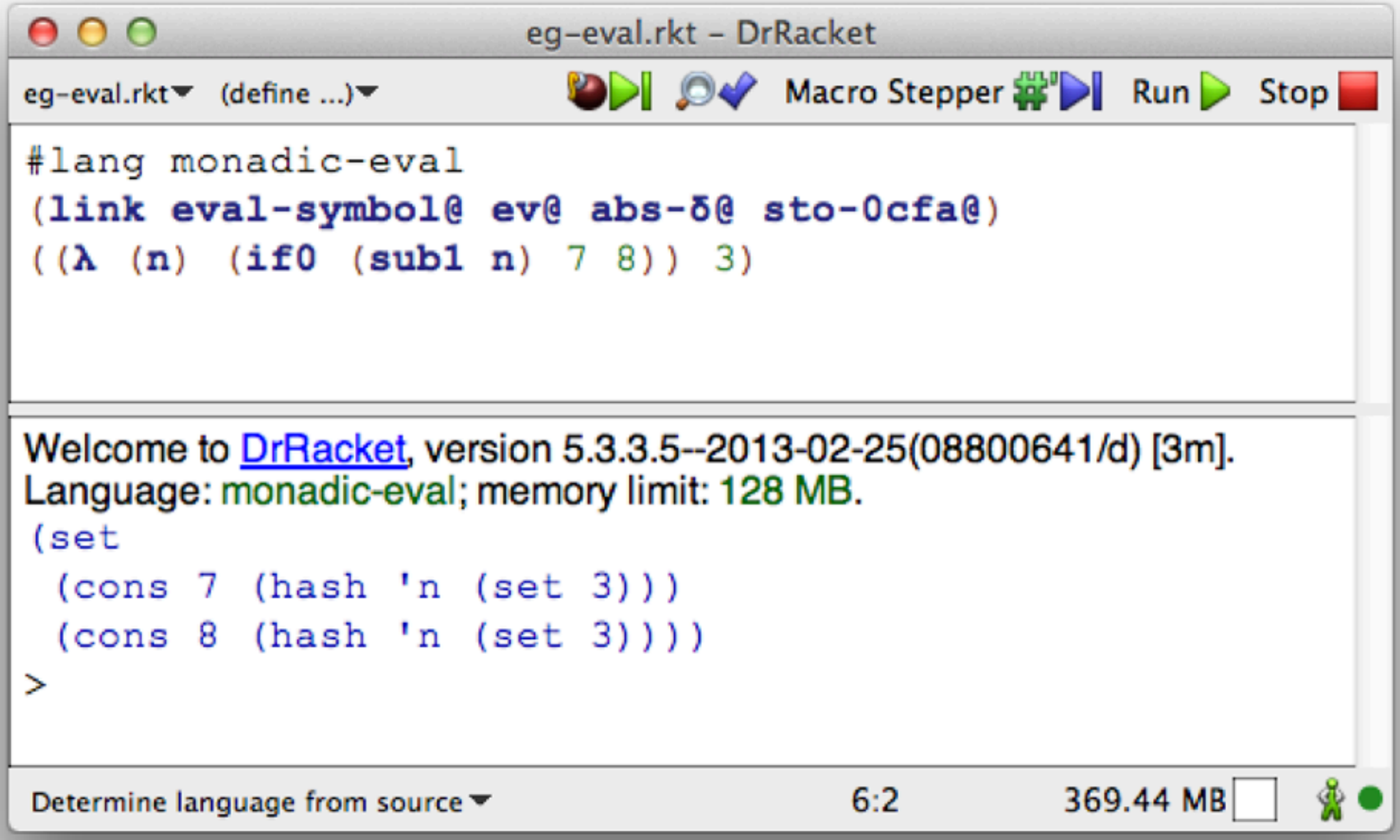
SYMBOLIC VALUES +

ABSTRACT OPERATIONS +

FINITE HEAP

sto-0cfa@

```
(import unit^ unit-vals^ unit-ans^)
(export env^ sto^)

(define ((get r x) s)
  ((unit-vals (lookup s r x)) s))

(define ((alloc f v) s)
  (match f
    [(cons (lam x e) r)
     (define a x) ; 0CFA-like abstraction
     (unit-ans a (join-sto s a v))]))

(define ((ralloc x v) s)
  (match v
    [(cons e r)
     (define a x)
     ((unit a)
      (join-sto s a
        (cons e (hash-set r x a))))]))
```


eg-eval.rkt – DrRacket
eg-eval.rkt
(define ...)
Macro Stepper
Run
Stop
#lang monadic-eval
(link eval-symbol@ ev@ abs-δ@ sto-0cfa@)
((λ (n) (if0 (sub1 n) 7 8)) 3)
Welcome to DrRacket, version 5.3.3.5--2013-02-25(08800641/d) [3m].
Language: monadic-eval; memory limit: 128 MB.
(set
(cons 7 (hash 'n (set 3)))
(cons 8 (hash 'n (set 3))))
>
Determine language from source
6:2
369.44 MB

eg-eval.rkt – DrRacket

eg-eval.rkt (define ...) Macro Stepper Run Stop

```
#lang monadic-eval
(link eval-symbol@ ev@ abs-δ@ sto-0cfa@)
((λ (f) ((λ (_) (f 2)) (f 1)))
 (λ (n) n))
```

```
(set
 (cons
  1
  (hash
   '
   _
   (set 1)
   'n
   (set 1 2)
   'f
   (set '((lam n (vbl n)) . #hash()))))
(cons
 2
 (hash
  '
```

1 OR 2 DUE TO FINITE HEAP...

Determine language from source 22:2 406.35 MB

eg-eval.rkt – DrRacket

eg-eval.rkt (define ...) Macro Stepper Run Stop

```
#lang monadic-eval
(link eval-symbol@ ev@ abs-δ@ sto-0cfa@)
((λ (f) ((λ (_) (f 2)) (f 1)))
 (λ (n) n))
```

```
> (rec Δ (λ (x) (x x))
     (Δ Δ))

Interactions disabled
```

...BUT CAN STILL DIVERGE

Determine language from source 26:0 406.35 MB

```
#lang monadic-eval
(link pdcfa@ ev@ abs-δ@ sto-0cfa@)
```

SYMBOLIC VALUES +

ABSTRACT OPERATIONS +

FINITE HEAP +

CO-INDUCTIVE MEMOIZATION

#lang monadic-eval

(link pdcfa@ ev@ abs-0@ stc-0cfa@)

TOTAL

SYMBOLIC VALUES +

ABSTRACT OPERATIONS +

FINITE HEAP +

CO-INDUCTIVE MEMOIZATION

pdcfa@

```
(import ev^)
(export eval^ symbolic^ rec^ unit^ bind^ err^)

;; iterates ev until reaching a fixed point in the memo-table
(define (eval e)
  (let loop ([m* (hash)] [anss (set)])
    (match ((((rec e (hash)) (hash)) (hash)) m*)
      [(and r (cons anss1 m1))
       (if (equal? r (cons anss m*))
           anss1
           (loop m1 anss1))])))
```

ITERATE EV

UNTIL MEMO FIXED

```
;; like ev but takes both branches on abstract values
(define (ev* e r)
  (match e
    [(ifz e0 e1 e2)
     (do v ← (rec e0 r)
       (match v
         [0            (rec e1 r)]
         [(? number?) (rec e2 r)]
         [(? symbolic?)
          (both (rec e1 r)
                (rec e2 r))]))]
    [_  (ev e r)]))
```

pdcfa@

```
(define ((((both c0 c1) s) m) m*)
  (match (((c0 s) m) m*)
    [(cons anss0 m)
     (match (((c1 s) m) m*)
       [(cons anss1 m)
        (cons (set-union anss0 anss1) m)])]))

(define ((((rec e r) s) m) m*)
  (define ers (list e r s))
  (define anss (hash-ref m ers #false))
  (if anss
      (cons anss m)
      (match ((((ev* e r) s)
                (hash-set m ers (hash-ref m* ers (set)))) m*)
        [(cons anss m)
         (cons anss (hash-set m ers anss))])))
```

CO-INDUCTIVE

MEMOIZATION

```
(define ((((bind a f) s) m) m*)
  (match (((a s) m) m*)
    [(cons anss m)
     (let-values
         ([(anss m)
           (for*/fold ([rs (set)]
                       [m m])
             ([ans anss])
             (match ans
               [(cons 'err s)
                (values (set-union rs (set (cons 'err s))) m
               [(cons v s)
                (match ((((f v) s) m) m*)
                  [(cons anss m)
                   (values (set-union rs anss) m)])]))])
       (cons anss m))]))
```

eg-eval.rkt – DrRacket

eg-eval.rkt (define ...) Macro Stepper Run Stop

```
#lang monadic-eval
(link pdcfa@ ev@ abs-δ@ sto-0cfa@)
((λ (f) ((λ (_) (f 2)) (f 1)))
 (λ (n) n))
```

Language: monadic-eval; memory limit: 128 MB.

```
(set
 (cons
  1
  (hash
   '
   _
   (set 1)
   'n
   (set 1 2)
   'f
   (set '((lam n (vbl n)) . #hash()))))
 (cons
  2
  (hash
```

1 OR 2 JUST

LIKE BEFORE...

Determine language from source 5:0 301.58 MB

eg-eval.rkt – DrRacket

eg-eval.rkt (define ...) Macro Stepper Run Stop

```
#lang monadic-eval
(link pdcfa@ ev@ abs-δ@ sto-0cfa@)
(rec Δ (λ (x) (x x))
  (Δ Δ))
```

...BUT ALWAYS TERMINATES

Welcome to DrRacket, version 5.3.3.5--2013-02-25(08800641/d) [3m].
Language: monadic-eval; memory limit: 128 MB.

```
(set)
> |
```

PROVES THIS PROGRAM DIVERGES

Determine language from source 4:2 301.58 MB

INHERITING THE CONTROL STACK

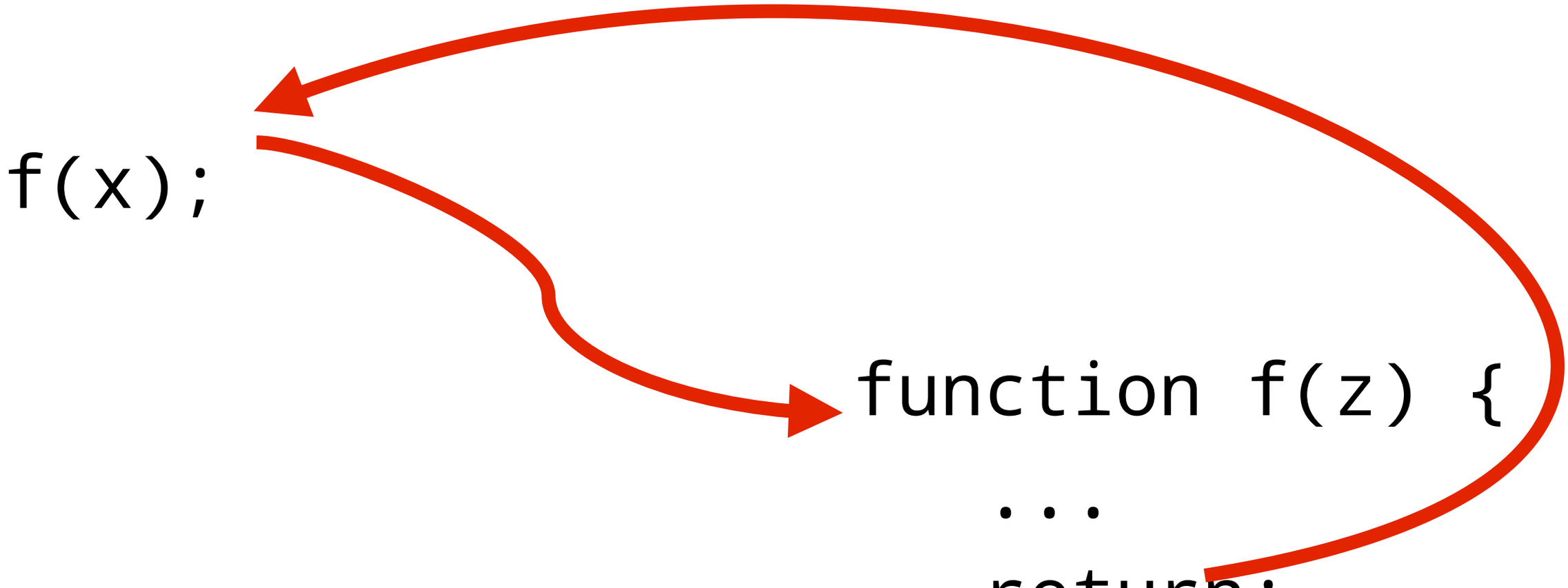
```
f(x);
```

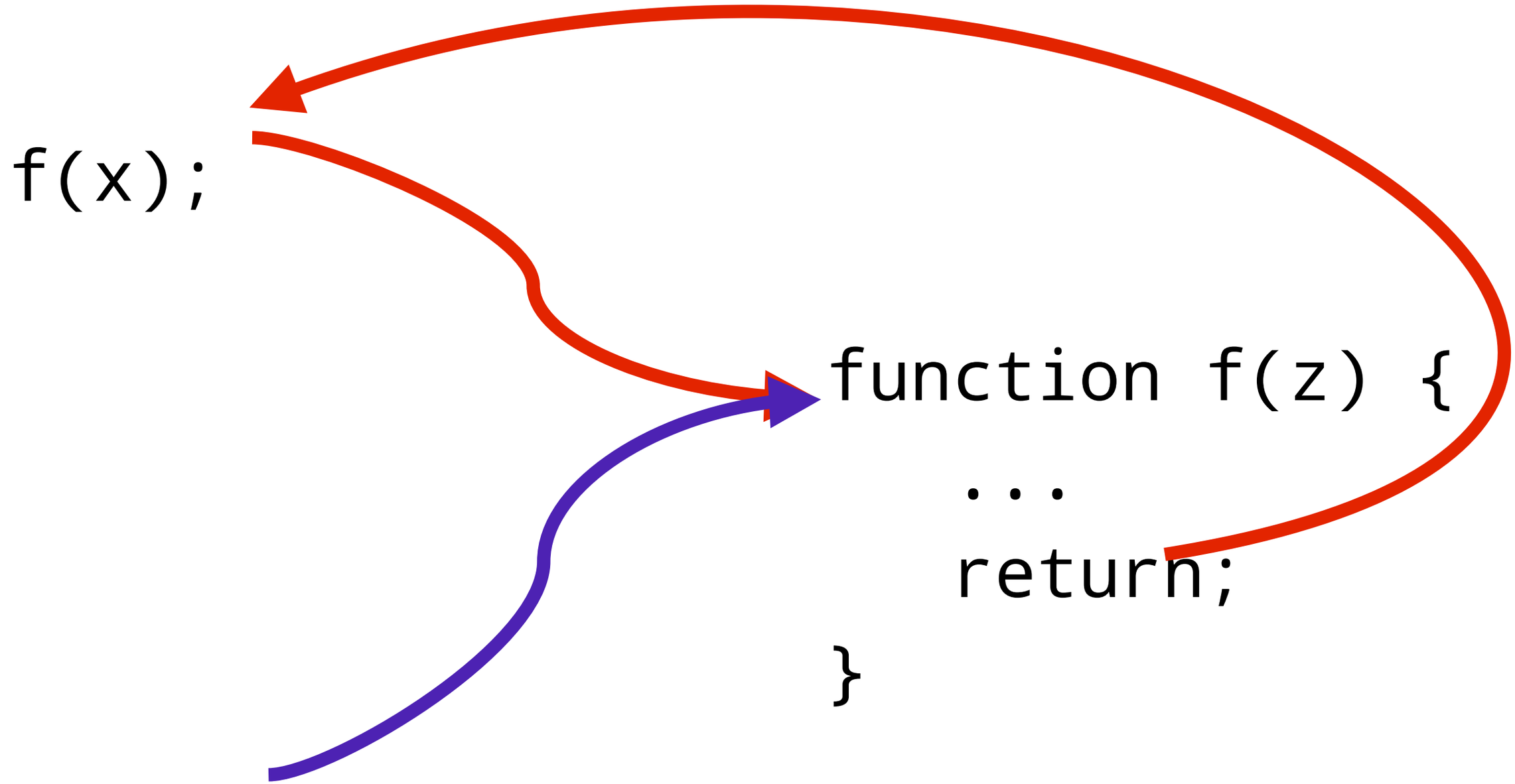
```
                    function f(z) {
                        ...
                        return;
                    }
```

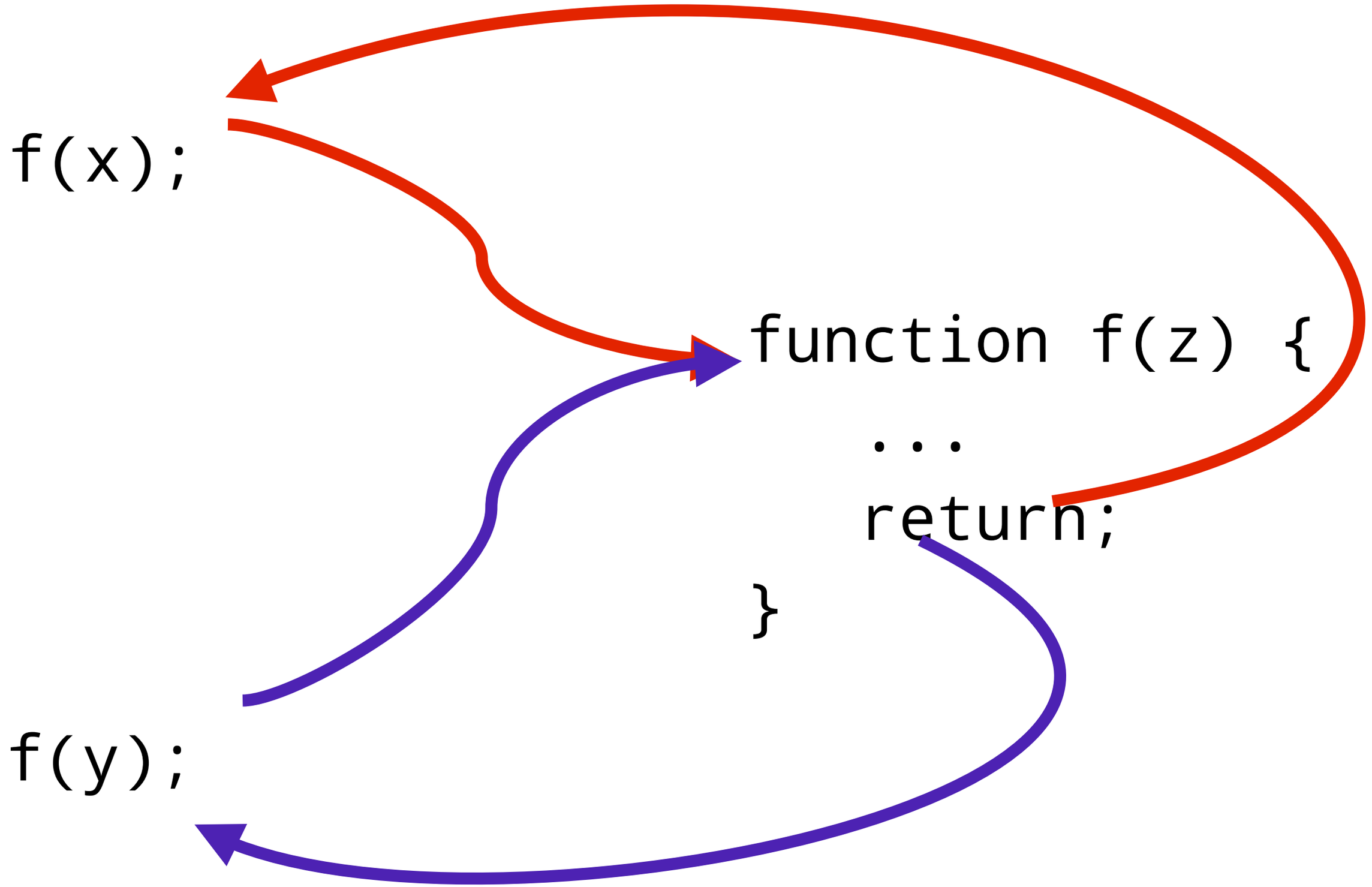
```
f(y);
```

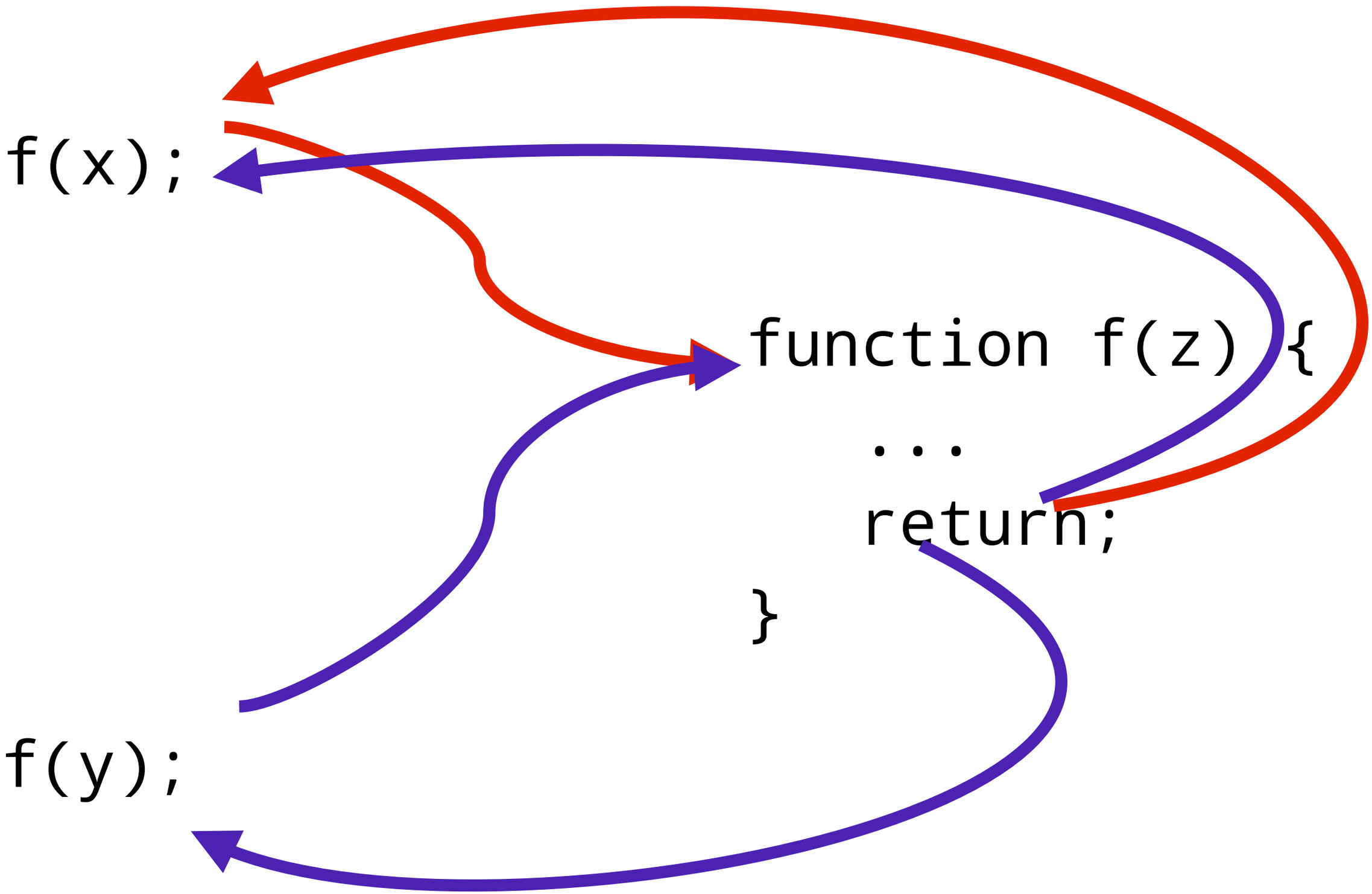
```
f(x);
```

```
function f(z) {
    ...
    return;
}

f(y);
```


f(x);
function f(z) {
...
return;
}
f(y);


f(x);
function f(z) {
...
return;
}
f(y);


f(x);
function f(z) {
...
return;
}
f(y);


f(x);
function f(z) {
...
return;
}
f(y);

```
(let* ((id (λ (x) ((λ (q) q) x)))
       (y (id 0))
       (z (id 1)))
  y)
```

0CFA: 0 OR 1

PDCFA: 0

WHY DOES OUR ABSTRACT INTERPRETER ONLY SAY 0?

WE INHERITED THE (PRECISE) CONTROL STACK FROM THE META-LANGUAGE

PLAY WITH NEW ABSTRACTIONS...

```
#lang monadic-eval
(link pdcfa@ ev@ pres-δ@ sto-widen@)
```

SYMBOLIC VALUES +
PRECISE OPERATIONS +
FINITE, WIDENING HEAP

pres-δ@

```
(import unit^ symbolic^ err^)
(export δ^)
(define (δ o . vs)
  (match* (o vs)
    [('add1 (list (? number? n)))   (unit (add1 n))]
    [('sub1 (list (? number? n)))   (unit (sub1 n))]
    [('- (list (? number? n)))      (unit (- n))]
    [('+ (list (? number? n1) (? number? n2))) (unit (+ n1 n2))]
    [('- (list (? number? n1) (? number? n2))) (unit (- n1 n2))]
    [('* (list (? number? n1) (? number? n2))) (unit (* n1 n2))]
    [('quotient (list (? number? n1) (? number? n2)))
     (if (zero? n2)
         (err)
         (unit (quotient n1 n2)))]
[('add1 (list n))   (unit 'N)]
[('sub1 (list n))   (unit 'N)]
[('+ (list n1 n2)) (unit 'N)]
[('* (list n1 n2)) (unit 'N)]
[('- (list n1 n2)) (unit 'N)]
[('quotient (list n1 (? number? n2)))
 (if (zero? n1)
     (err)
     (unit 'N))]
[('quotient (list n1 n2))
 (both (err) (unit 'N))]))
```

PRECISION PRESERVING

sto-widen@

```
(import unit^ unit-vals^ unit-ans^)
(export env^ sto^)

(define ((get r x) s)
  ((unit-vals (lookup s r x)) s))

(define ((alloc f v) s)
  (match f
    [(cons (lam x e) r)
     (define a x) ; 0CFA-like abstraction
     ;; widen on sto update
     (unit-ans a
       (update-sto s a
         (widen (hash-ref s a (set)) v)))]))

(define ((ralloc x v) s)
  (match v
    [(cons e r)
     (define a x)
     ((unit a)
      (join-sto s a (cons e (hash-set r x a))))]))

(define (widen s v)
  (match v
    [(? number?)
     (match s
       [(set) (set v)]
       [(set 'N _ ...) s]
       [(set (? (λ (m) (equal? m v))) _ ...) s]
       [(set (? number? m) _ ...)
        (join-numeric s)]
       [_
        (set-add s v)])]
    ['N
     (join-numeric s)]
    [x
     (set-add s x)]))
```

WIDEN ON HEAP COLLISION

PLAY WITH NEW COMPOSITIONS...

```
#lang monadic-eval
(link pdcfa-dead@ ev@ pres-δ@ sto-widen@)

#lang monadic-eval
(link pdcfa-dead@ ev@ abs-δ@ sto-0cfa@)
```

Open questions...

Open questions...

* is this stuff sound? (I believe it)
* How would you prove it? (no idea)
* is there a correspondence with CFA2?
* how does it work in practice?
* Can symbolic execution and CFA be combined systematically?

```
newtype ZPDCFAT var val m a = ZPDCFAT
  { unZPDCFAT :: AbstractT (CFAAddr var) ZCFATime var val m a }
  deriving
  ( Monad
  , MonadTrans
  , MonadFunctor
  , MonadPlus
  , MonadState s
  , MonadReader r
  , MonadEnvReader (Env var (CFAAddr var))
  , MonadEnvState env
  , MonadStoreState (Store ListSet (CFAAddr var) val)
  , MonadTimeState ZCFATime
  , MonadMorph ListSet
  )
```

CAN CALCULATE THIS WITH MONAD TRANSFORMERS

```
mkZPDCFAT ::
  (Monad m)
  => (Env var (CFAAddr var)
      -> Store ListSet (CFAAddr var) val
      -> ZCFATime
      -> m (ListSet (a, Store ListSet (CFAAddr var) val, ZCFATime))
     )
  -> ZPDCFAT var val m a
mkZPDCFAT = ZPDCFAT . mkAbstractT
```

```
runZPDCFAT ::
  (Monad m)
  => ZPDCFAT var val m a
  -> Env var (CFAAddr var)
  -> Store ListSet (CFAAddr var) val
  -> ZCFATime
  -> m (ListSet (a, Store ListSet (CFAAddr var) val, ZCFATime))
runZPDCFAT = runAbstractT . unZPDCFAT
```

```
driveZPDCFA ::
  (Ord val, Ord expr, Ord var)
  => ((expr -> ZPDCFA_Driver var val expr (ListSet val))
      -> (expr -> ZPDCFA_Driver var val expr (ListSet val))
     )
  -> expr
  -> ListSet (ListSet val, Store ListSet (CFAAddr var) val, ZCFATime)
driveZPDCFA eval expr =
  let loop mx =
        let (_VxSxT_list,(m1,mx')) =
              runIdentity
              $ flip runStateT (lbot,mx)
              $ runZPDCFAT (memoEval eval expr) Map.empty lbot ()
        in assert (mx == mx') $
        if m1 == mx'
          then _VxSxT_list
          else loop m1
  in loop lbot
```

http://david.darais.com/

https://github.com/dvanhorn/monadic-eval/

THANKS!