Software Contracts:

Semantics
Verification
Optimization

Sam Tobin-Hochstadt
Indiana University

Not in this talk

Gradual Types

Metaprogramming

Refinement Types

Concurrent Systems

Performance Tools

Contracts

R. Morris
Editor

On the Criteria To Be Used in Decomposing Systems into Modules

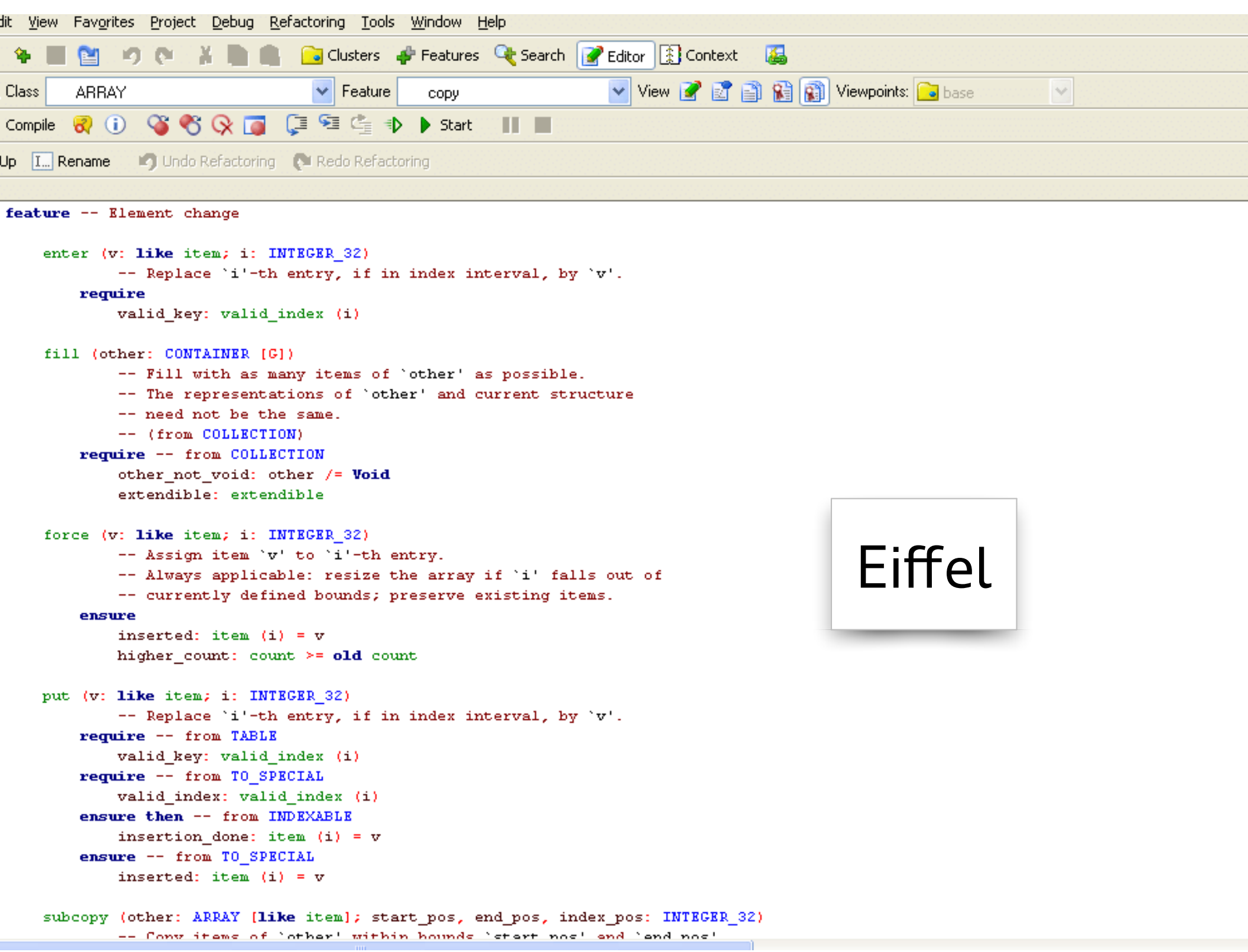
D.L. Parnas
Carnegie-Mellon University

**s modularization as a mechanism
bility and comprehensibility of a
the shortening of its development**

Introduction

A lucid statement of the philosophy of mod


Eiffel


dit View Favorites Project Debug Refactoring Tools Window Help
Clusters Features Search Editor Context
Class ARRAY Feature copy View Viewpoints: base
Compile Start
Up Rename Undo Refactoring Redo Refactoring
feature -- Element change
enter (v: like item; i: INTEGER_32)
-- Replace `i'-th entry, if in index interval, by `v'.
require
valid_key: valid_index (i)
fill (other: CONTAINER [G])
-- Fill with as many items of `other' as possible.
-- The representations of `other' and current structure
-- need not be the same.
-- (from COLLECTION)
require -- from COLLECTION
other_not_void: other /= Void
extendible: extendible
force (v: like item; i: INTEGER_32)
-- Assign item `v' to `i'-th entry.
-- Always applicable: resize the array if `i' falls out of
-- currently defined bounds; preserve existing items.
ensure
inserted: item (i) = v
higher_count: count >= old count
put (v: like item; i: INTEGER_32)
-- Replace `i'-th entry, if in index interval, by `v'.
require -- from TABLE
valid_key: valid_index (i)
require -- from TO_SPECIAL
valid_index: valid_index (i)
ensure then -- from INDEXABLE
insertion_done: item (i) = v
ensure -- from TO_SPECIAL
inserted: item (i) = v
subcopy (other: ARRAY [like item]; start_pos, end_pos, index_pos: INTEGER_32)
Eiffel

Now available in

- Python
- JavaScript
- Ruby
- C#
- Java
- ...

point-in-module

```
(provide point-in?)

(define (point-in? p x y)
  (define p-dot
    (pin-under p x y (disk 1)))
  (equal?
   (pict->argb-pixels p-dot)
   (pict->argb-pixels p)))
```

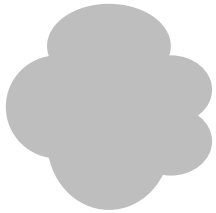
point-in-module

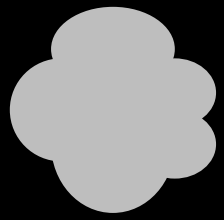
```
(provide point-in?)

(define (point-in? p x y)
  (define p-dot
    (pin-under p x y (disk 1)))
  (equal?
   (pict->argb-pixels p-dot)
   (pict->argb-pixels p)))
```

```
(point-in? (cloud 100 100) 0 0)
```

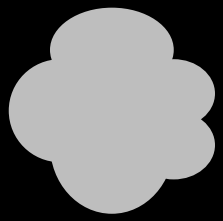
point-in-module

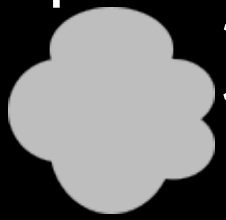
```
(provide point-in?)

(define (point-in? p x y)
  (define p-dot
    (pin-under p x y (disk 1)))
  (equal?
   (pict->argb-pixels p-dot)
   (pict->argb-pixels p)))
```

(point-in?  0 0)

```
(define (point-in? p x y)
  (define p-dot
    (pin-under p x y (disk 1)))
  (equal?
   (pict->argb-pixels p-dot)
   (pict->argb-pixels p)))

(point-in?  0 0)
```

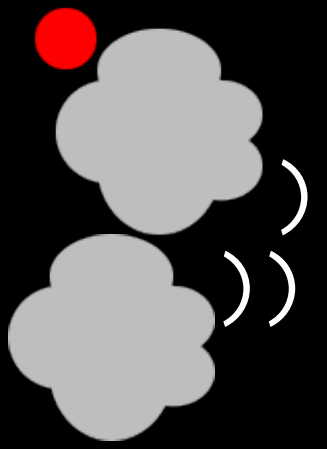
```
(define p-dot
  (pin-under  0 0 (disk 1)))
(equal?
 (pict->argb-pixels p-dot)
 (pict->argb-pixels  ))
```






(define p-dot
)
(equal?
(pict->argb-pixels p-dot)
(pict->argb-pixels))

```
(equal?
 (pict->argb-pixels      )
 (pict->argb-pixels      ))
```



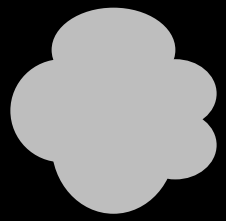
```
(equal?
 #"\0\377\377\377\0\3...")
 #"\377\377\0\0\377\3..."))
```

point-in-module

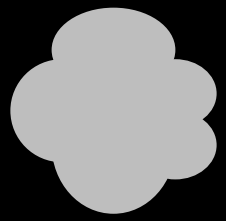
```
(provide point-in?)

(define (point-in? p x y)
  (define p-dot
    (pin-under p x y (disk 1)))
  (equal?
   (pict->argb-pixels p-dot)
   (pict->argb-pixels p)))
```

```
(point-in? (cloud 100 100) 50 50)
```

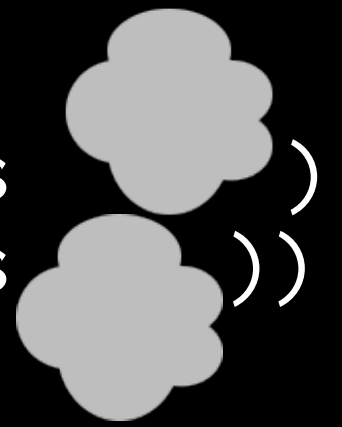
```
(define p-dot
  (pin-under                50 50 (disk 1)))
(equal?
 (pict->argb-pixels p-dot)
 (pict->argb-pixels      ))
```



```
(define p-dot
                                        )

(equal?
 (pict->argb-pixels p-dot)
 (pict->argb-pixels    ))
```



```
(equal?
 (pict->argb-pixels      )
 (pict->argb-pixels      ))
```



```
(equal?
 #"\0\377\377\377\0\3...")
 #"\0\377\377\377\0\3..."))
```

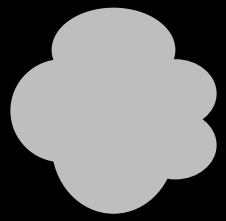
```
> (point-in? (cloud 100 100) #f #f)
```

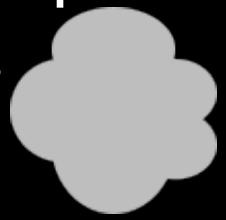
```
> (point-in? (cloud 100 100) #f #f)
pin-under: contract violation
  expected: (or/c real? pict-path?)
  given: #f
  in: the dx/fp argument of
      (->i
       ((base pict-convertible?)
        (dx/fp (or/c real? pict-path?))
        (dy/f
         (dx/fp)
         (if (real? dx/fp)
           real?
           (->
            pict-convertible?
            pict-path?
            (values real? real?))))
        (pict pict-convertible?))
       (result pict?))
  contract from: <pkgs>/pict-lib/pict/main.rkt
  blaming: point-in-module
   (assuming the contract is correct)
```

```
> (point-in? (cloud 100 100) #f #f)
pin-under: contract violation
  expected: (or/c real? pict-path?)
  given: #f
  in: the dx/fp argument of
      (->i
       ((base pict-convertible?)
        (dx/fp (or/c real? pict-path?))
        (dy/f
         (dx/fp)
         (if (real? dx/fp)
           real?
           (->
            pict-convertible?
            pict-path?
            (values real? real?))))
        (pict pict-convertible?))
```

blaming: point-in-module

```
(define p-dot
  (pin-under                  #f #f (disk 1)))
(equal?
 (pict->argb-pixels p-dot)
 (pict->argb-pixels          ))
```





```
(-> pict? real? real?
    boolean?)
```

point-in-module

```
(provide/contract
 [point-in? (-> pict? real? real?
                boolean?)])
(define (point-in? p x y)
  (define p-dot
    (pin-under p x y (disk 1)))
  (equal?
   (pict->argb-pixels p-dot)
   (pict->argb-pixels p)))
```

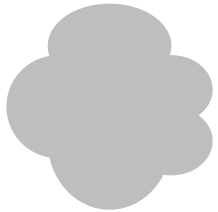
point-in-module

```
(provide/contract
 [point-in? (-> pict? real? real?
                boolean?)])
(define (point-in? p x y)
  (define p-dot
    (pin-under p x y (disk 1)))
  (equal?
   (pict->argb-pixels p-dot)
   (pict->argb-pixels p)))
```

```
(point-in? (cloud 100 100) #f #f)
```

point-in-module

```
(provide/contract
 [point-in? (-> pict? real? real?
                boolean?)])
(define (point-in? p x y)
  (define p-dot
    (pin-under p x y (disk 1)))
  (equal?
   (pict->argb-pixels p-dot)
   (pict->argb-pixels p)))
```

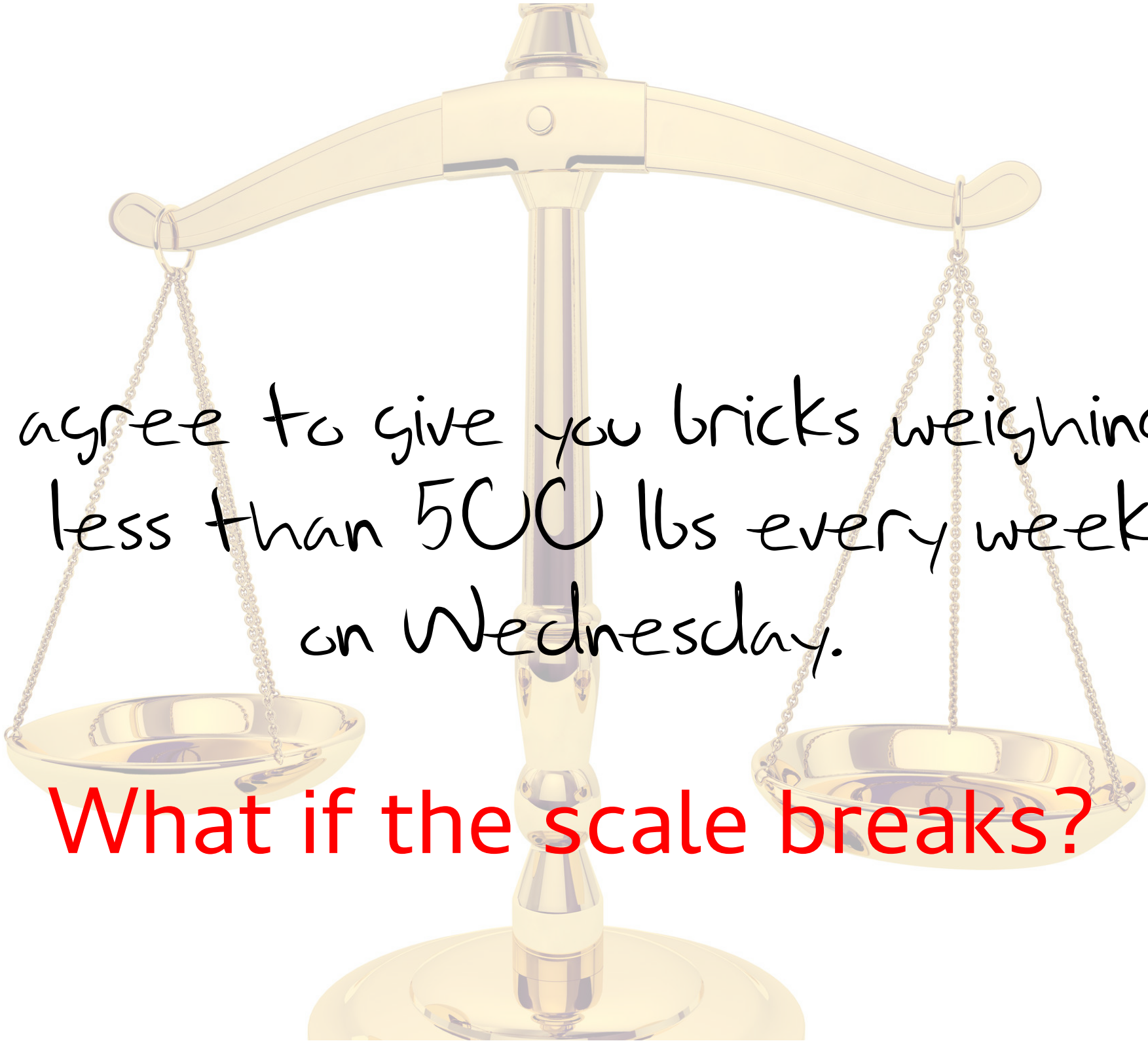
(point-in?  #f #f)

```
point-in?: contract violation
  expected: real?
  given: #f
  in: the 2nd argument of
      (-> pict? real? real? boolean?)
  contract from: point-in-module
  blaming: top-level
   (assuming the contract is correct)
```

Semantics

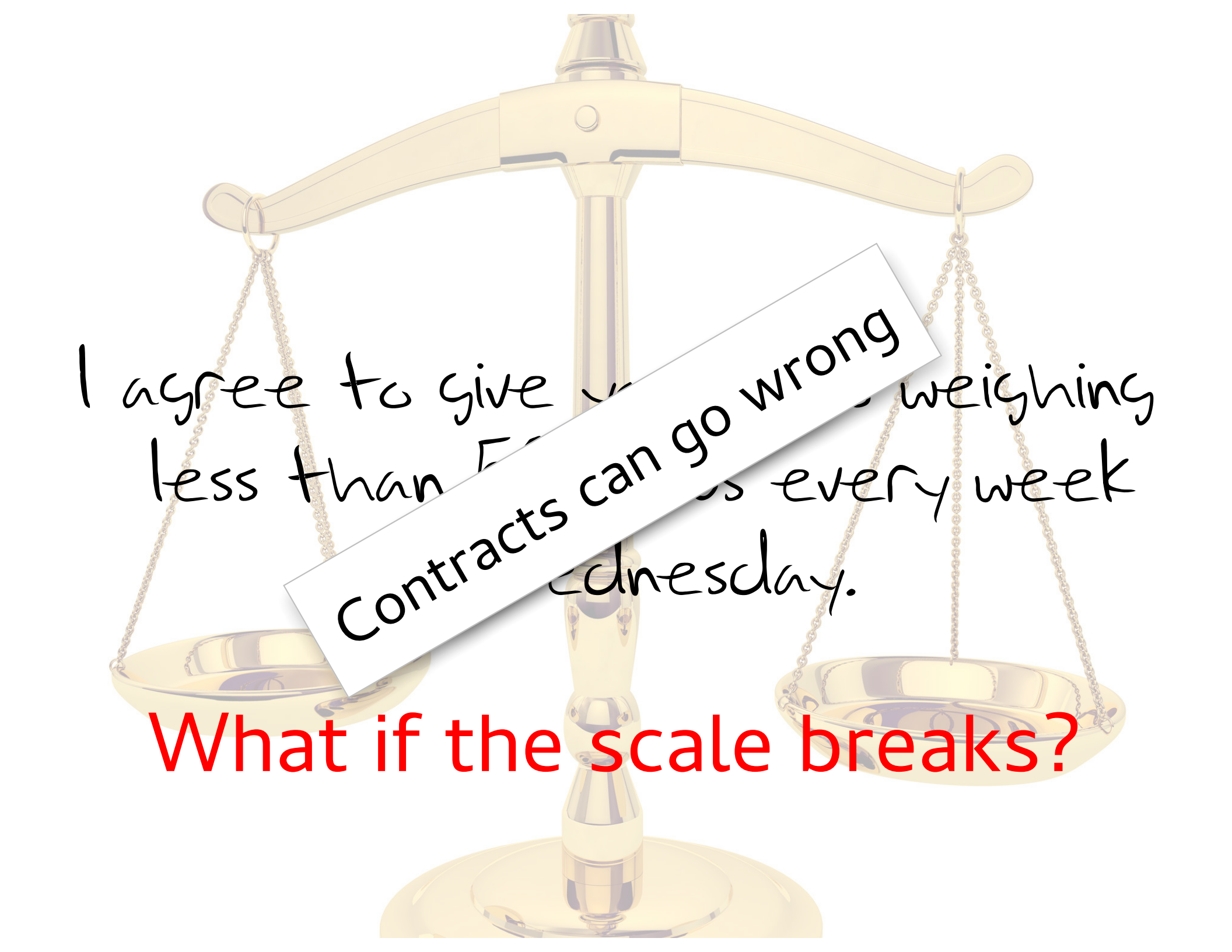

Signature

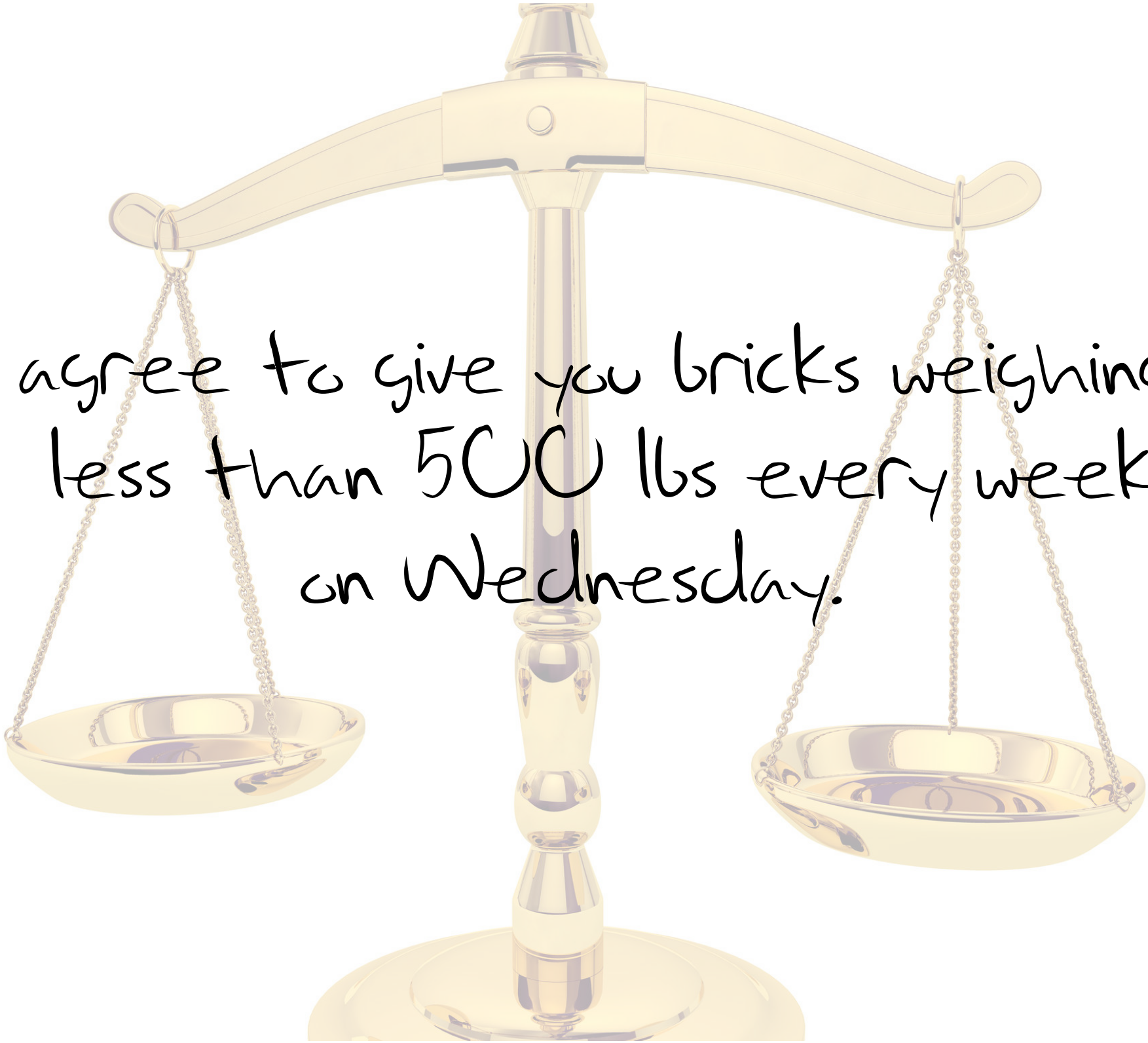
I agree to give you bricks weighing less than 500 lbs every week on Wednesday.

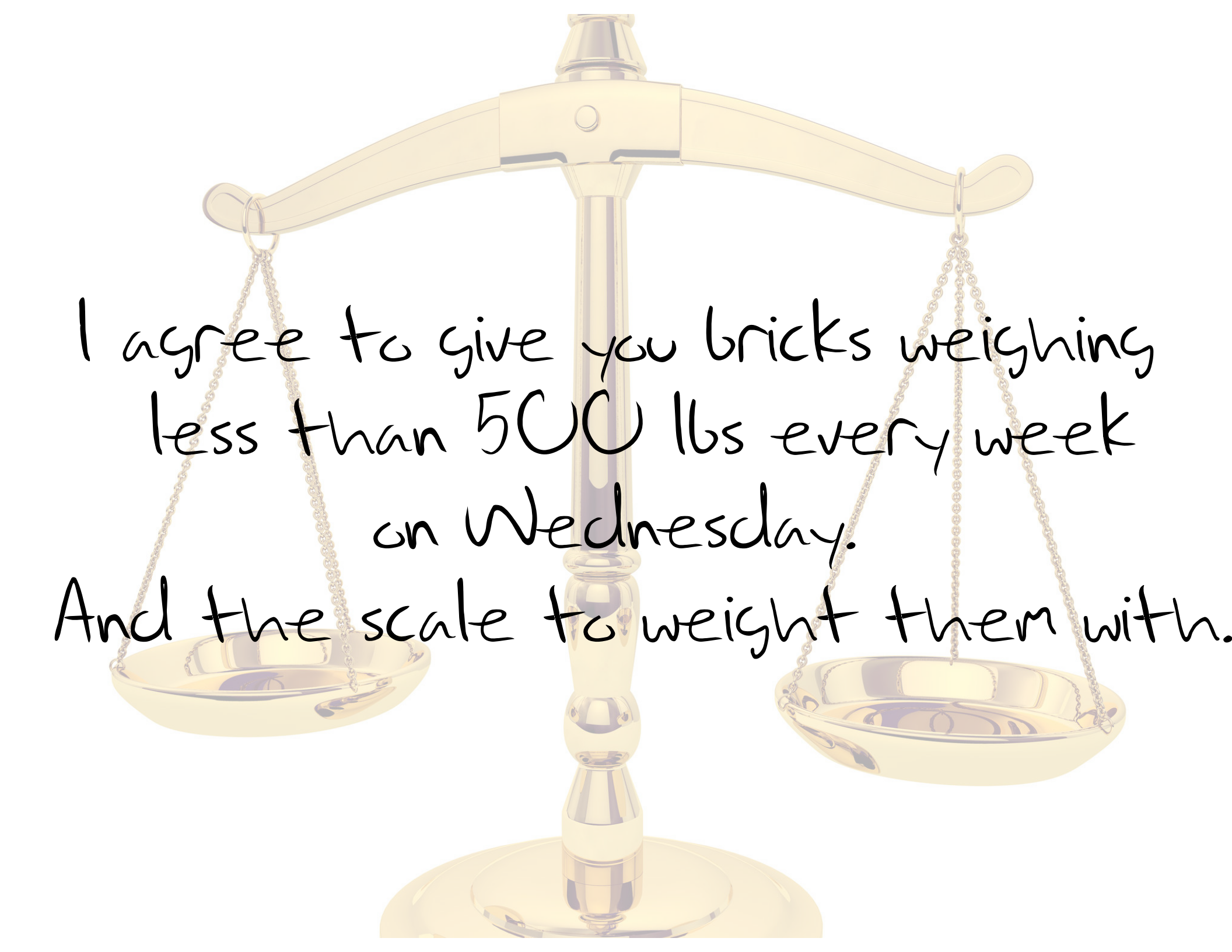


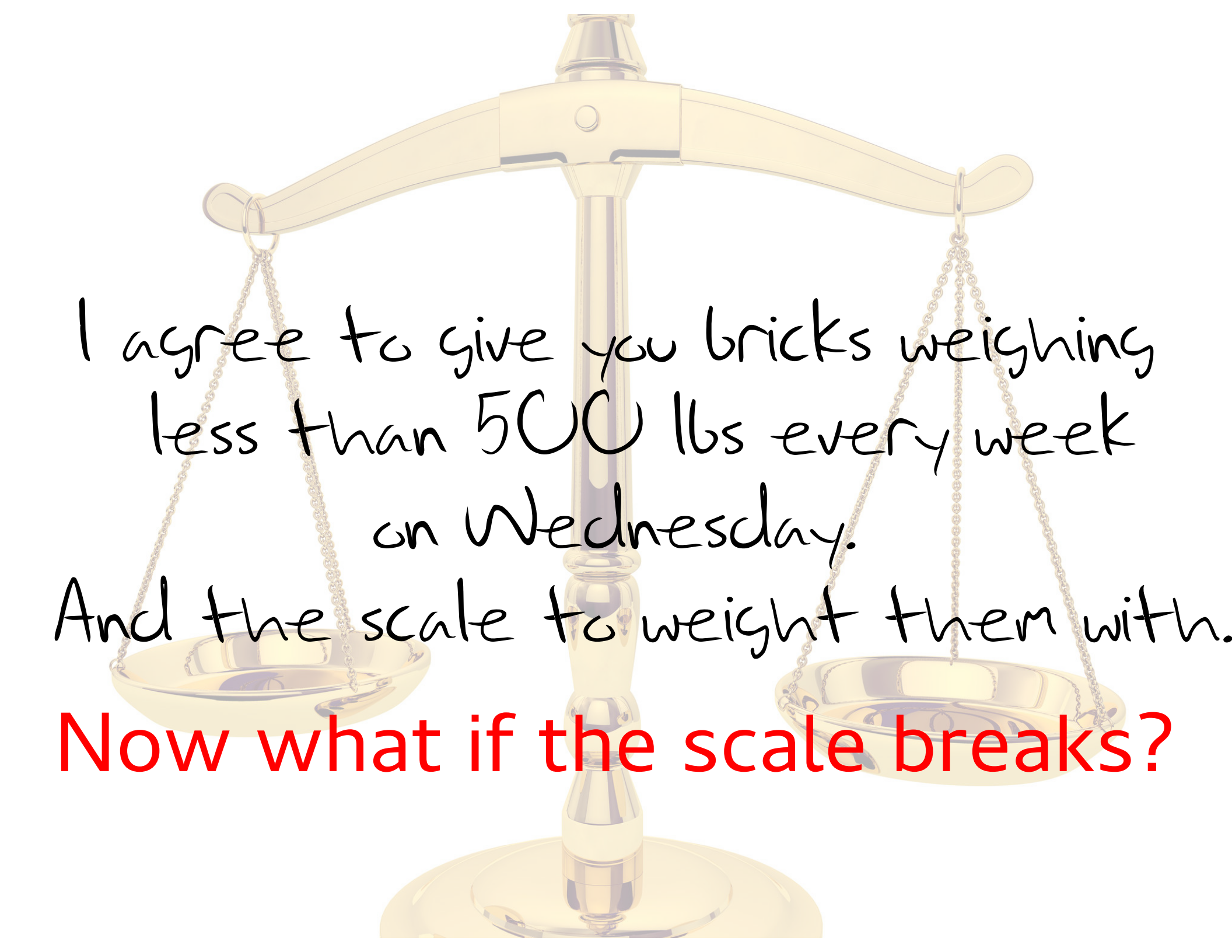
I agree to give you bricks weighing less than 500 lbs every week on Wednesday.

What if the scale breaks?


I agree to give
weighing
less than
every week
dnesday.
Contracts can go wrong
What if the scale breaks?


I agree to give you bricks weighing
less than 500 lbs every week
on Wednesday.


I agree to give you bricks weighing
less than 500 lbs every week
on Wednesday.
And the scale to weight them with.



I agree to give you bricks weighing less than 500 lbs every week on Wednesday.
And the scale to weight them with.

Now what if the scale breaks?

Dependent contracts

```
(provide
 [construction
  (->d [b bricks?]
       [scale (-> any/c weight)]
       #:pre (< (scale b) 500)
       any)])

(construction heavy-bricks scale)
```

Dependent contracts

```
(provide
 [construction
  (->d [b bricks?]
       [scale (-> <1000/c weight)]
       #:pre (< (scale b) 500)
       any)])
```

Dependent contracts

```
(provide
 [construction
  (->d [b bricks?]
       [scale (-> <1000/c weight)]
       #:pre (< (scale b) 500)
       any)])

(construction heavy-bricks fragile-scale)
```

Dependent contracts

```
(provide
 [construction
  (->d [b bricks?]
       [scale (-                  weight)]
       #:pre                    b) 500)
       any)])

(construction heavy-bricks fragile-scale)
```

Who should be blamed?

Solution: Complete Blame

- Says who should be blamed in tricky cases
- Ensures that contracts are complete
- Helps with gradual typing!

Semantics II

Many approaches to contracts

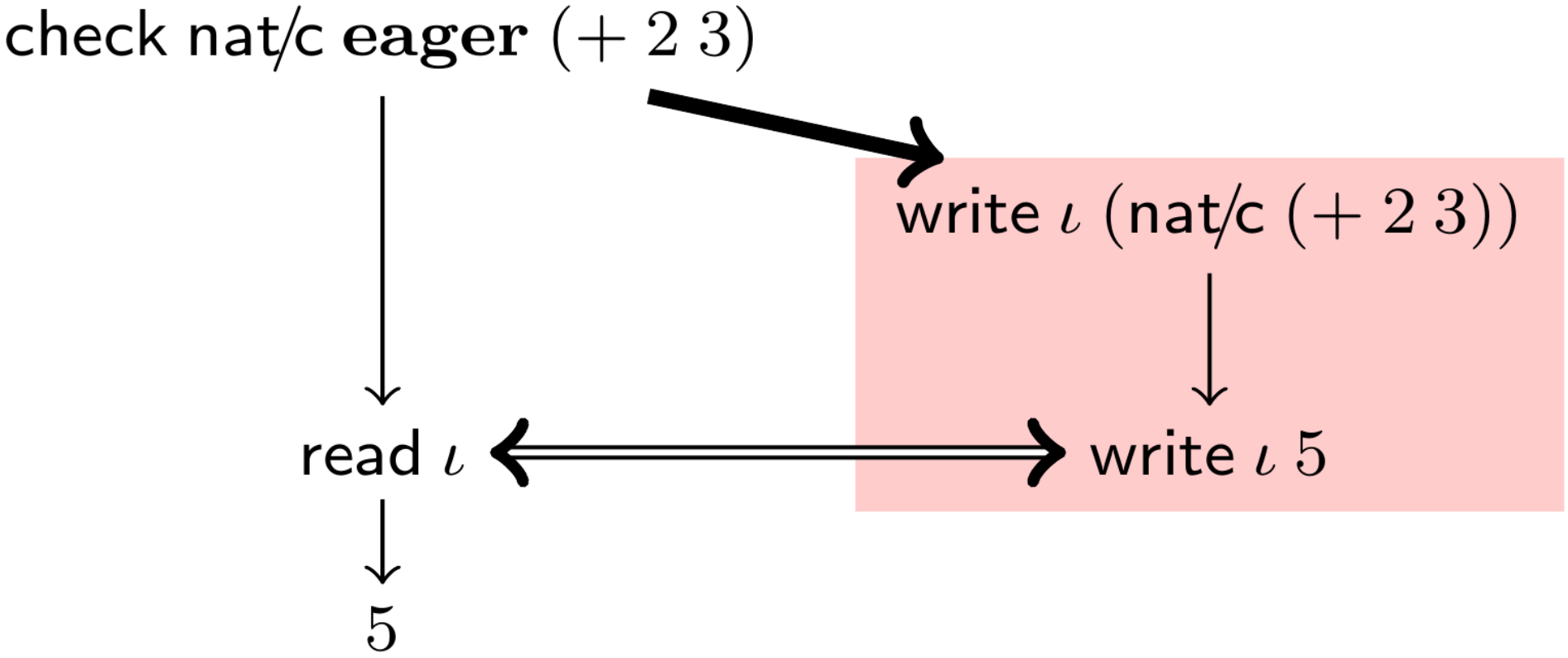
- Check everything now
- Check everything now, in parallel
- Check when needed
- ...

Many approaches to contracts

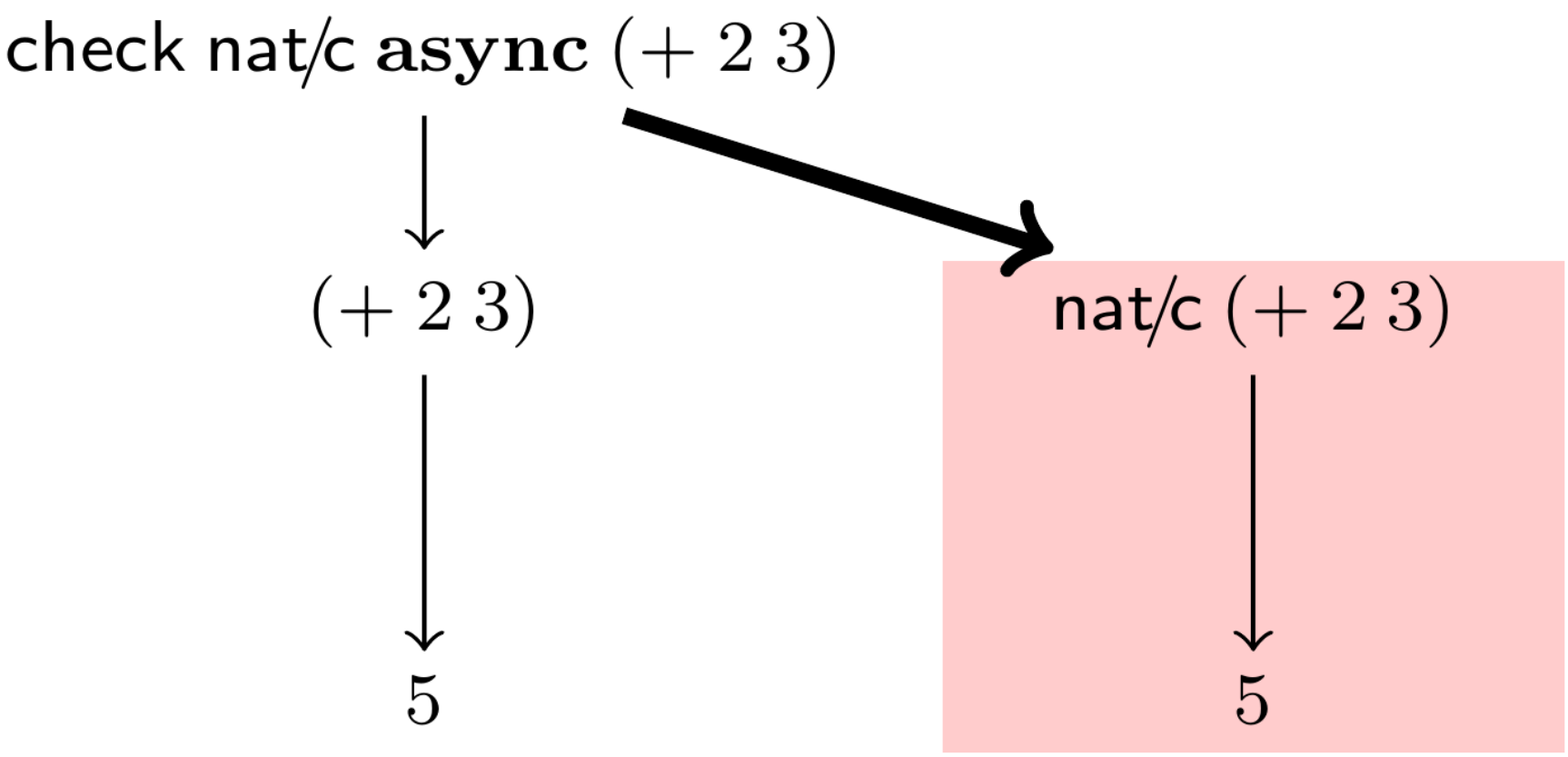
- Check everything now
- Check everything no
- Check when ne
- ...

Insight: communicating processes

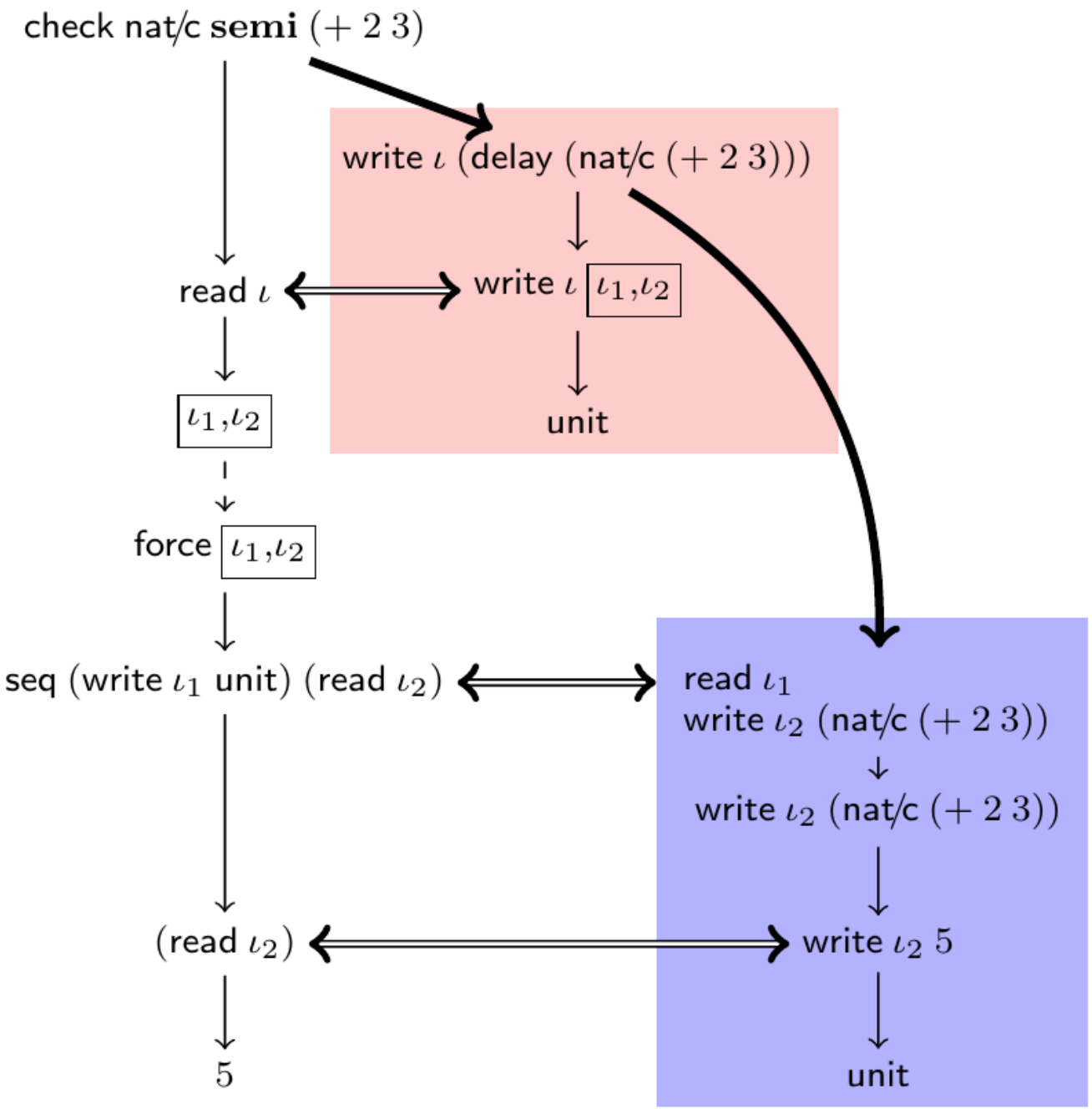
Eager Monitoring



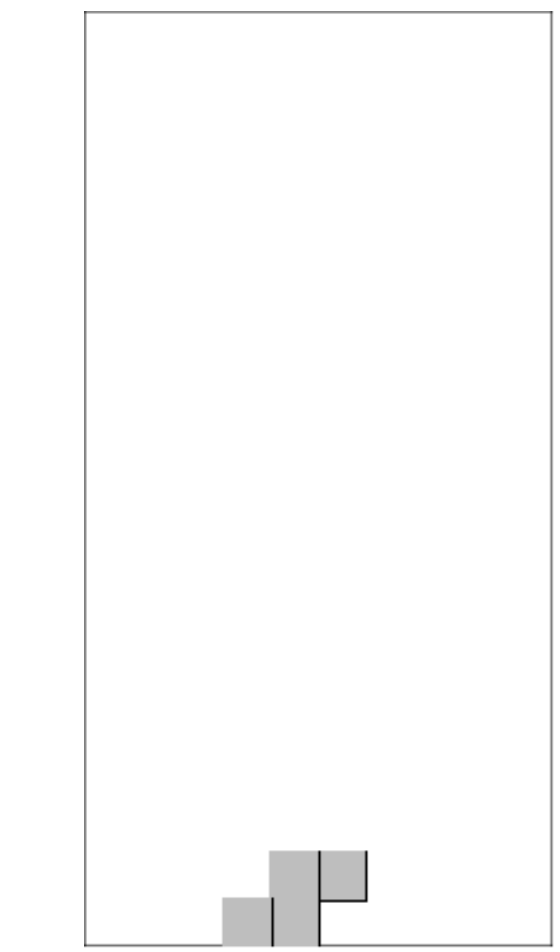
Asynchronous Monitoring



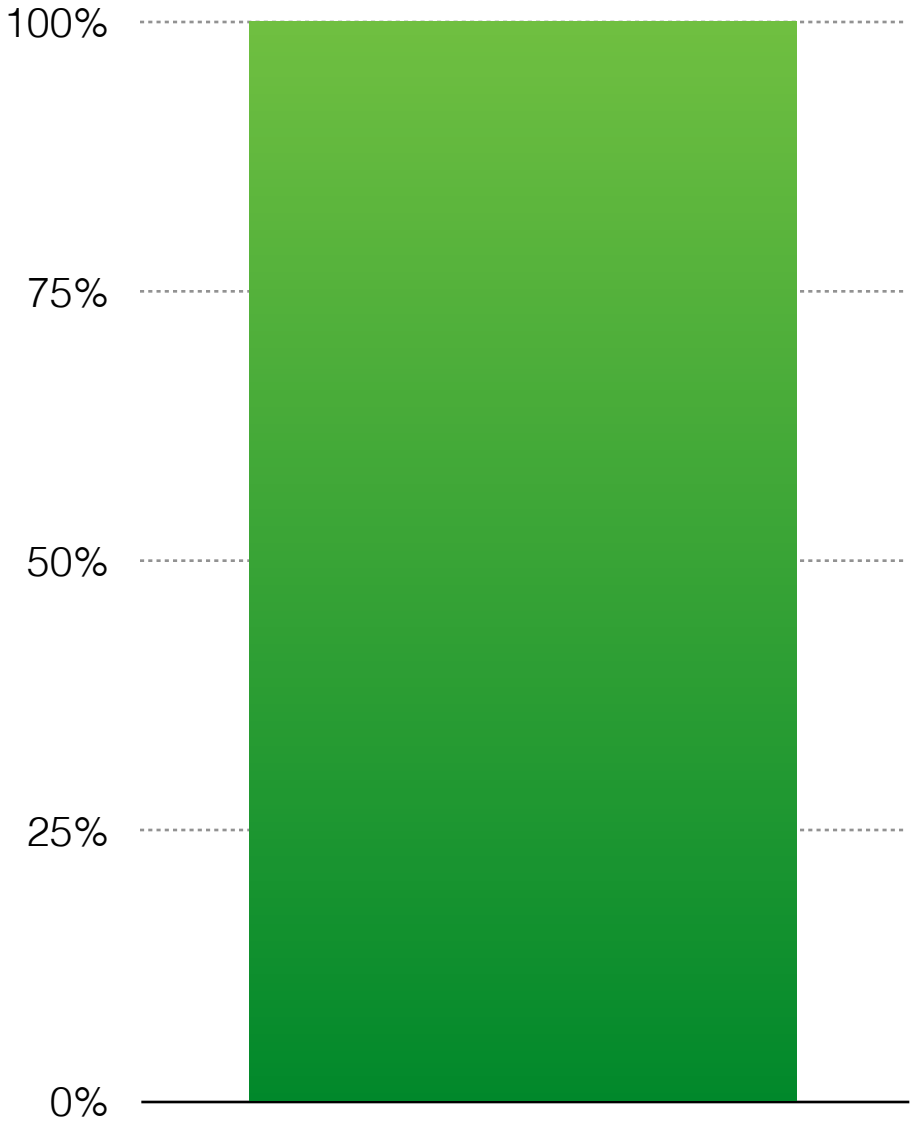
Semi-Eager Monitoring

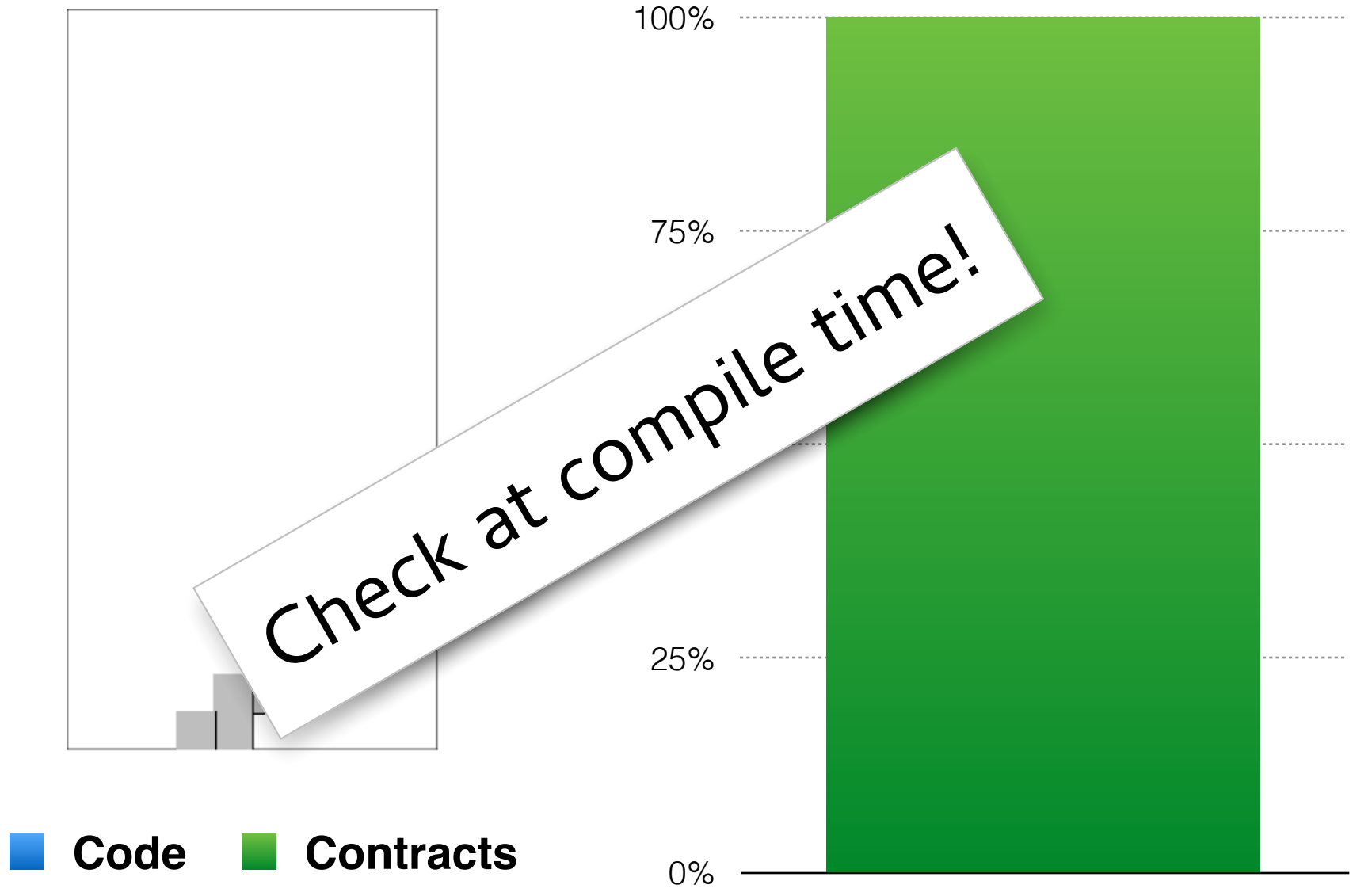


Verification



Code Contracts




100%
75%
25%
0%
Check at compile time!
Code
Contracts

rRacket internal error

posts by 2 authors G+1

Alex Knauth 4/22/15

Other recipients: users@racket-lang.org

I'm not sure how it got into this state, but now every time I start typing something or anything like that I get a DrRacket internal error window saying:

```
hash-ref: contract violation
  expected: hash?
  given: #f
  argument position: 1st
  other arguments...:
   'configure-runtime
   '()
  context...:
   /Applications/Racket/April-16/Racket v6.2.0.2/share/pkgs/drracket/drracket/private/syncheck/blueboxes-gui.rkt:450:4: compute-tag+rng method in ...ck/blueboxes-gui.rkt:175:2
   /Applications/Racket/April-16/Racket v6.2.0.2/share/pkgs/gui-lib/framework/private/coroutine.rkt:47:20
   /Applications/Racket/April-16/Racket v6.2.0.2/share/pkgs/gui-lib/framework/private/coroutine.rkt:56:0: coroutine-run
   /Applications/Racket/April-16/Racket v6.2.0.2/collects/racket/contract/private/arrow-val-first.rkt:265:18
   /Applications/Racket/April-16/Racket v6.2.0.2/share/pkgs/drracket/drracket/private/syncheck/blueboxes-gui.rkt:414:4: update-the-strs method in ...ck/blueboxes-gui.rkt:175:2
   /Applications/Racket/April-16/Racket v6.2.0.2/share/pkgs/gui-lib/framework/private/logging-timer.rkt:41:0: log-timeline/proc
```

acket Users ›

rRacket internal error

posts by 2 authors G+1

Alex Knauth 4/22/15

Other recipients: users@racket-lang.org

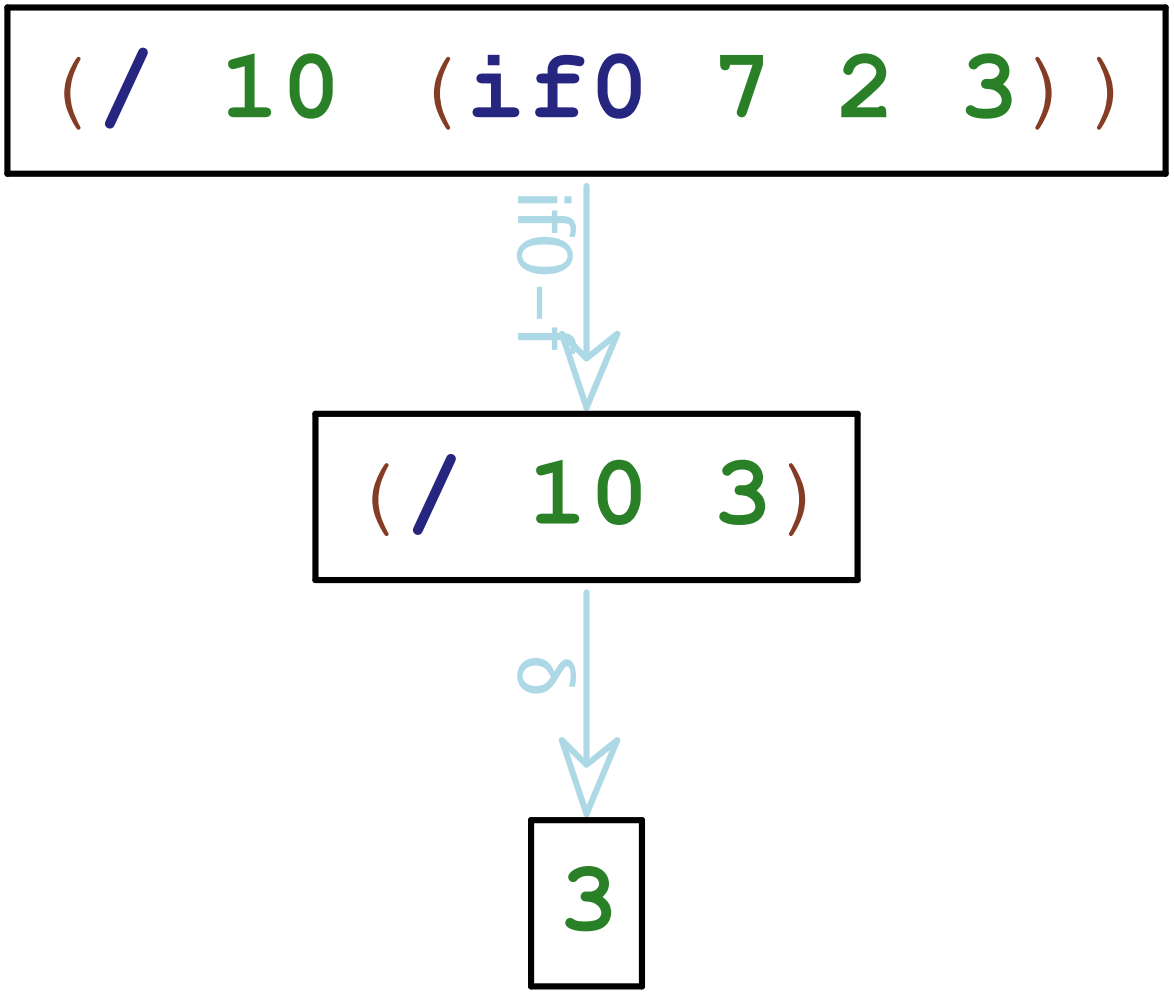
I'm not sure how it got into this state, but now every time I st ...ything like that I get a DrRacket internal error window saying:

hash-ref: contract violation
  expected: hash?
  given: #f
  argument position: 1st
  other arguments...:
   'configure-runtime
   '()
  context...:
   /Applications/Racket/April ...v6.2.0.2/share/pkgs/drracket/drracket/private/syncheck/blueboxes-gui.rkt:450:4: compute-tag+rng method in lueboxes-gui.rkt:175:2
   /Applications/Racket/April-16/Racket v6.2.0.2/share/pkgs/gui-lib/framework/private/coroutine.rkt:47:20
   /Applications/Racket/April-16/Racket v6.2.0.2/share/pkgs/gui-lib/framework/private/coroutine.rkt:56:0: coroutine-run
   /Applications/Racket/April-16/Racket v6.2.0.2/collects/racket/contract/private/arrow-val-first.rkt:265:18
   /Applications/Racket/April-16/Racket v6.2.0.2/share/pkgs/drracket/drracket/private/syncheck/blueboxes-gui.rkt:414:4: update-the-strs method in ...ck/blueboxes-gui.rkt:175:2
   /Applications/Racket/April-16/Racket v6.2.0.2/share/pkgs/gui-lib/framework/private/logging-timer.rkt:41:0: log-timeline/proc

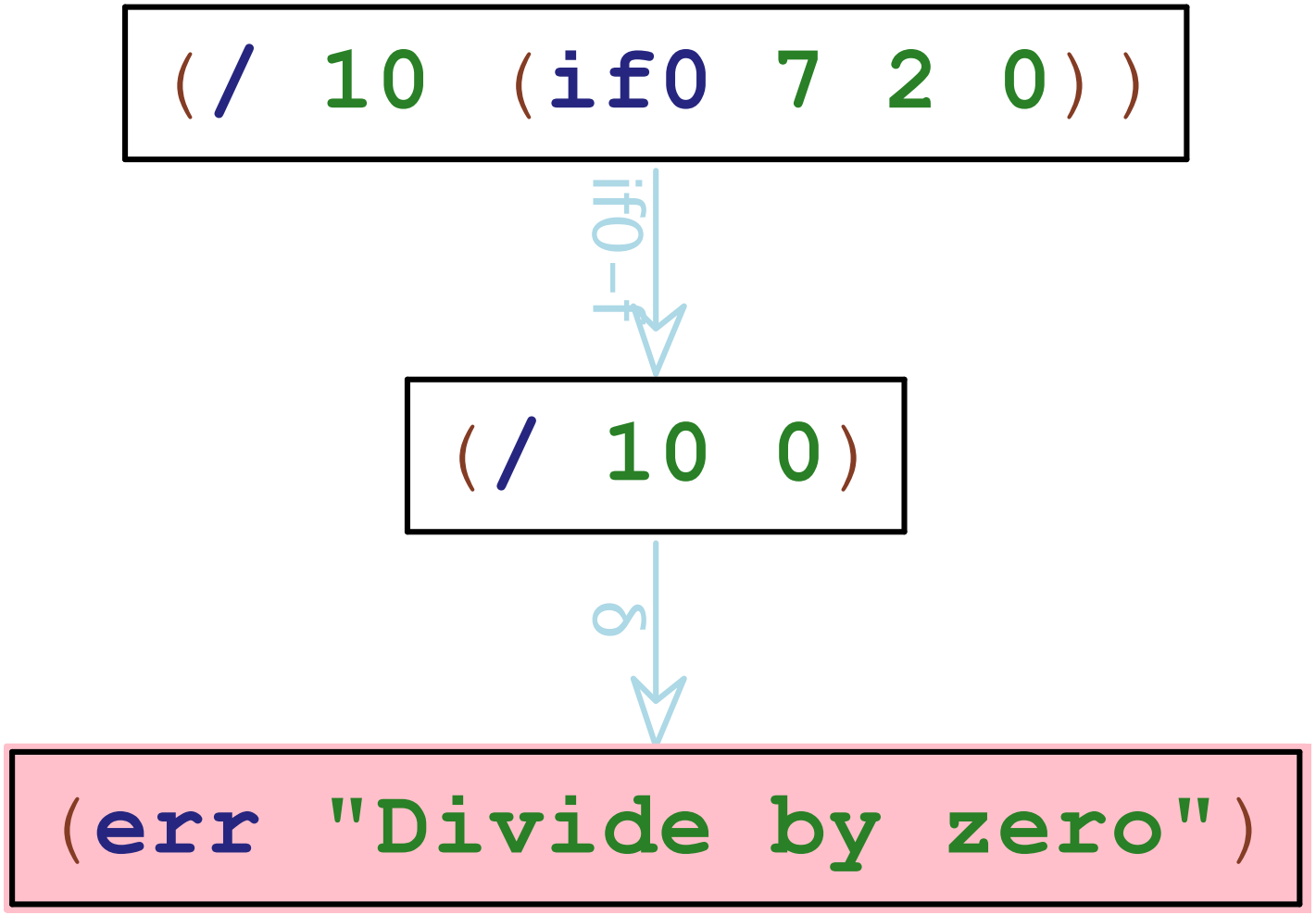
Check at compile time!

A simple language

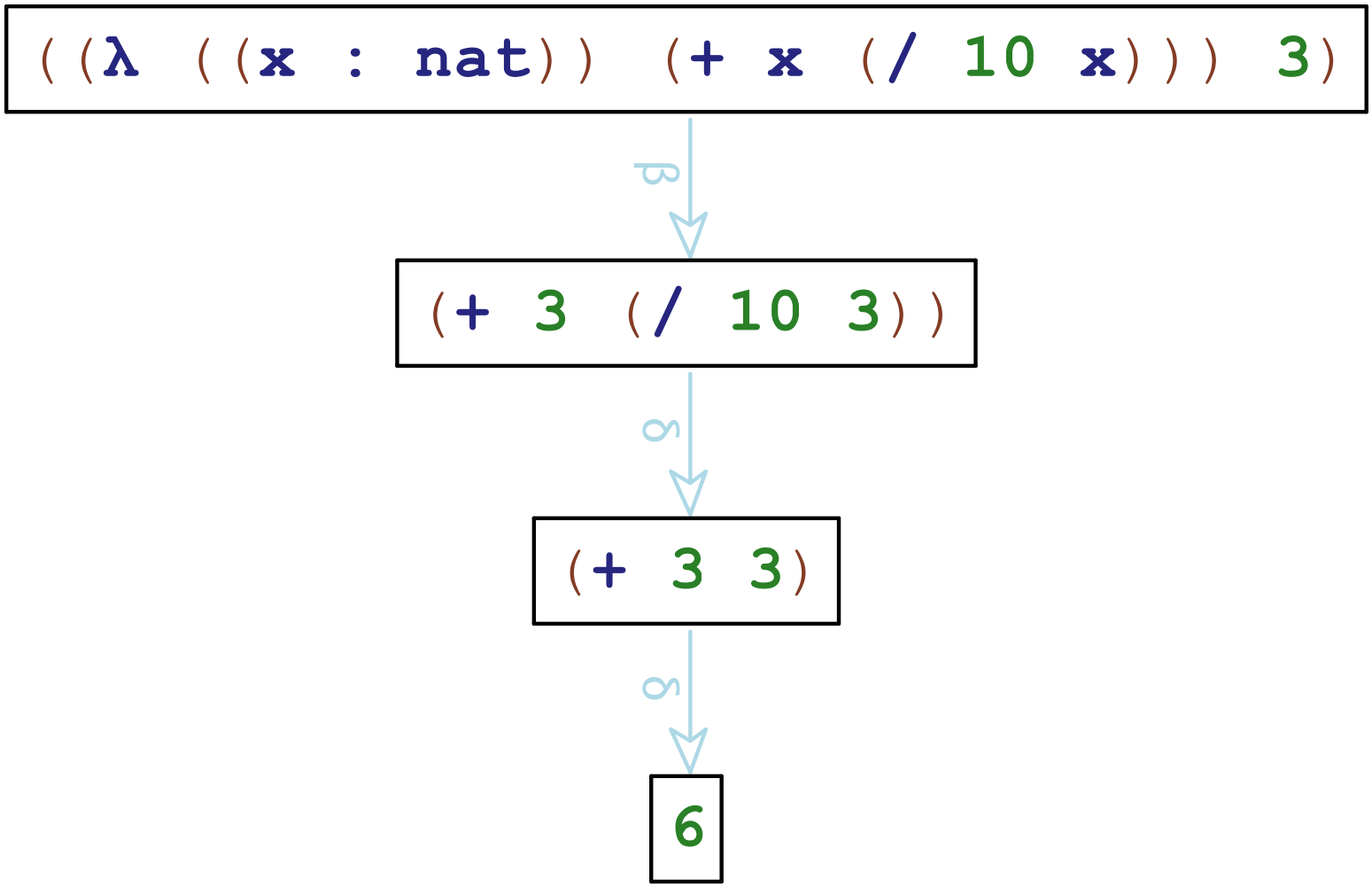
A simple language



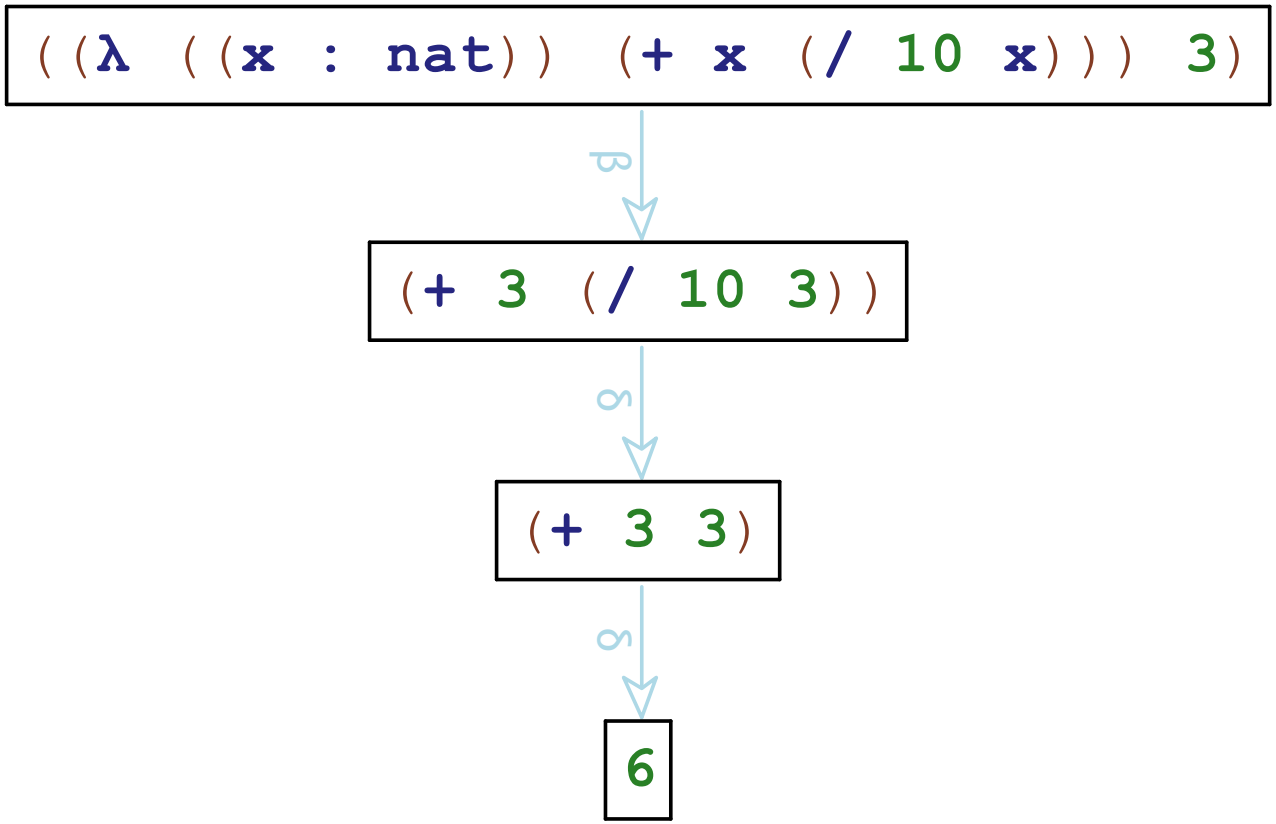
A simple language



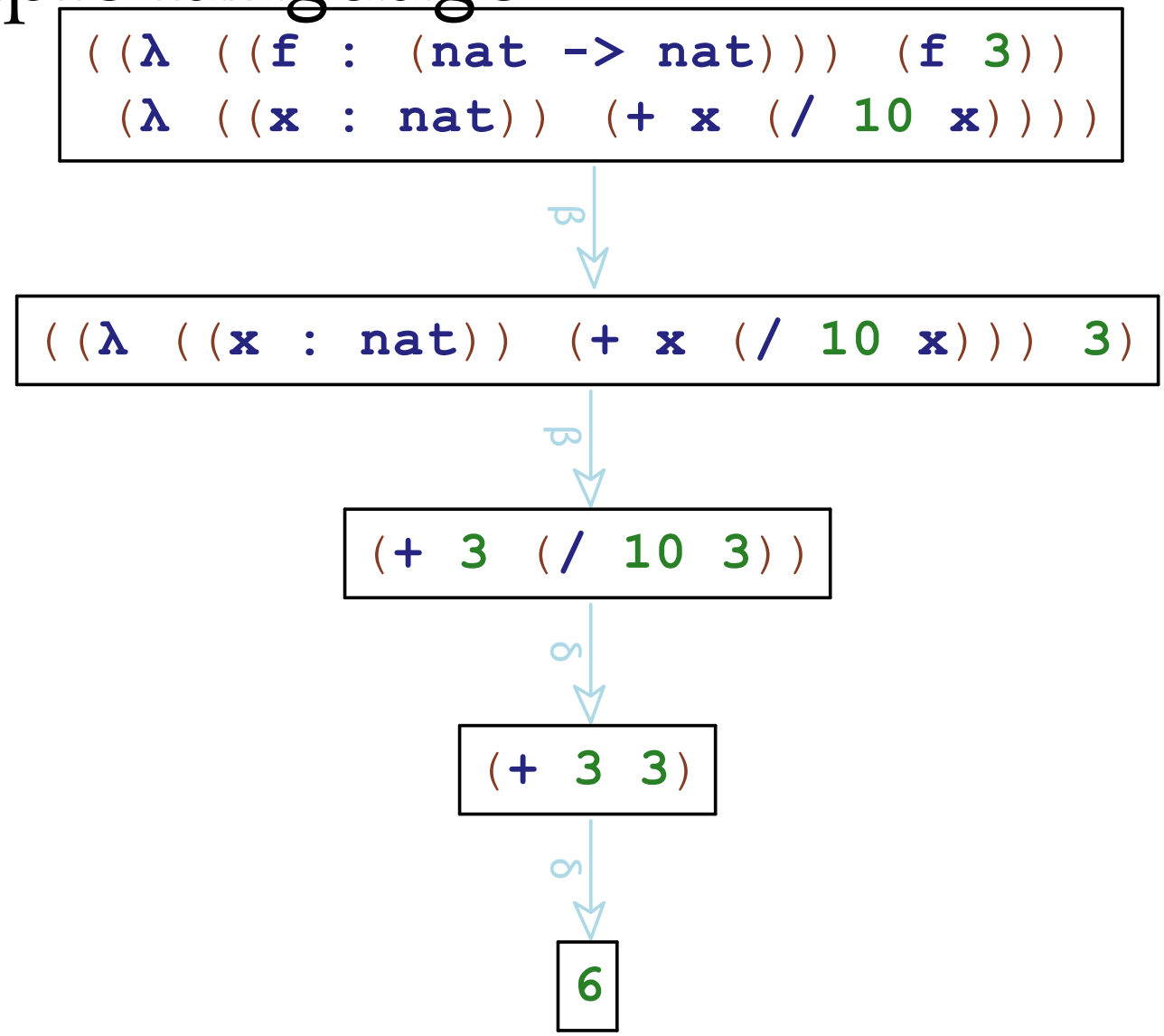
A simple language



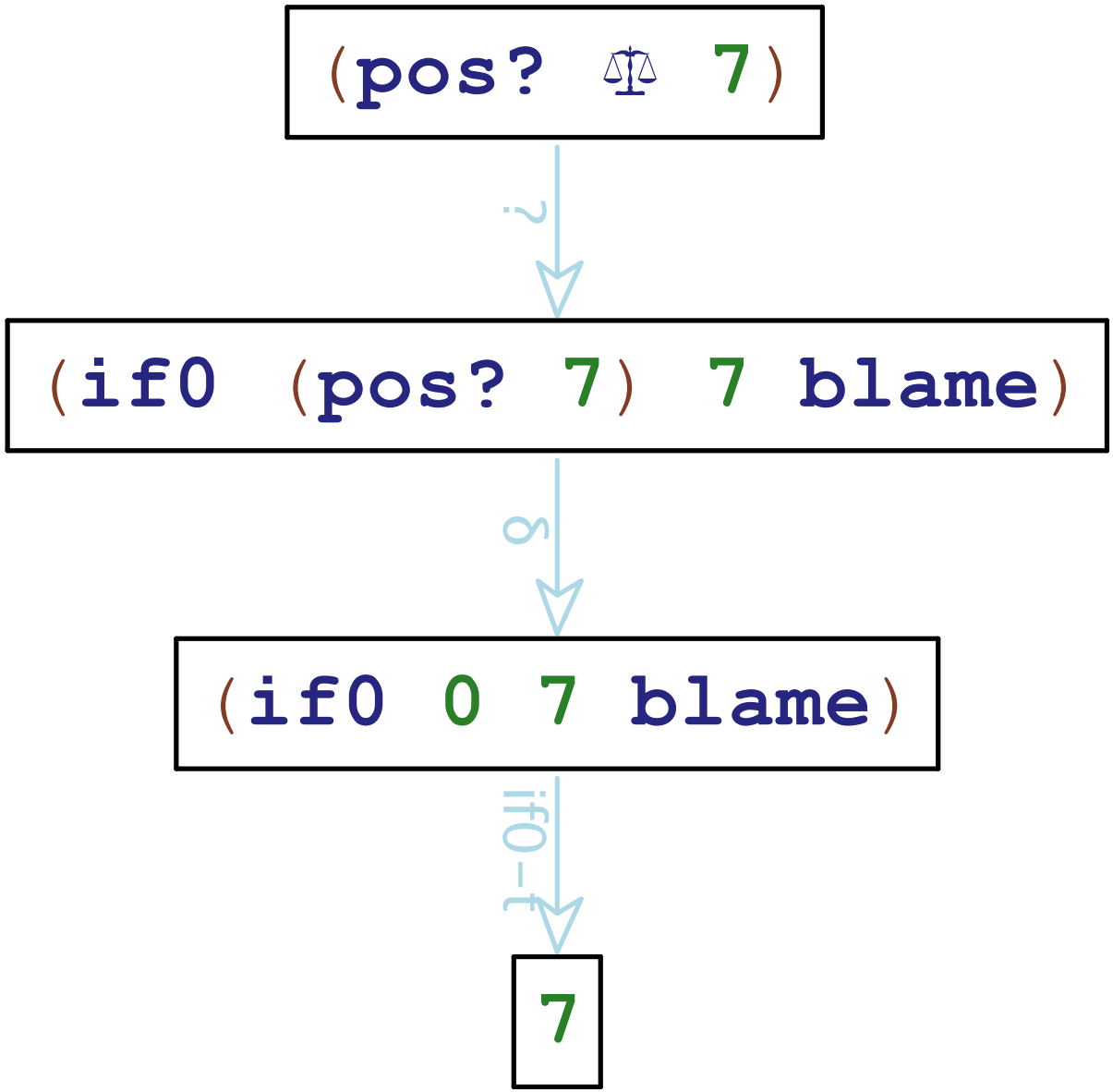
A simple language



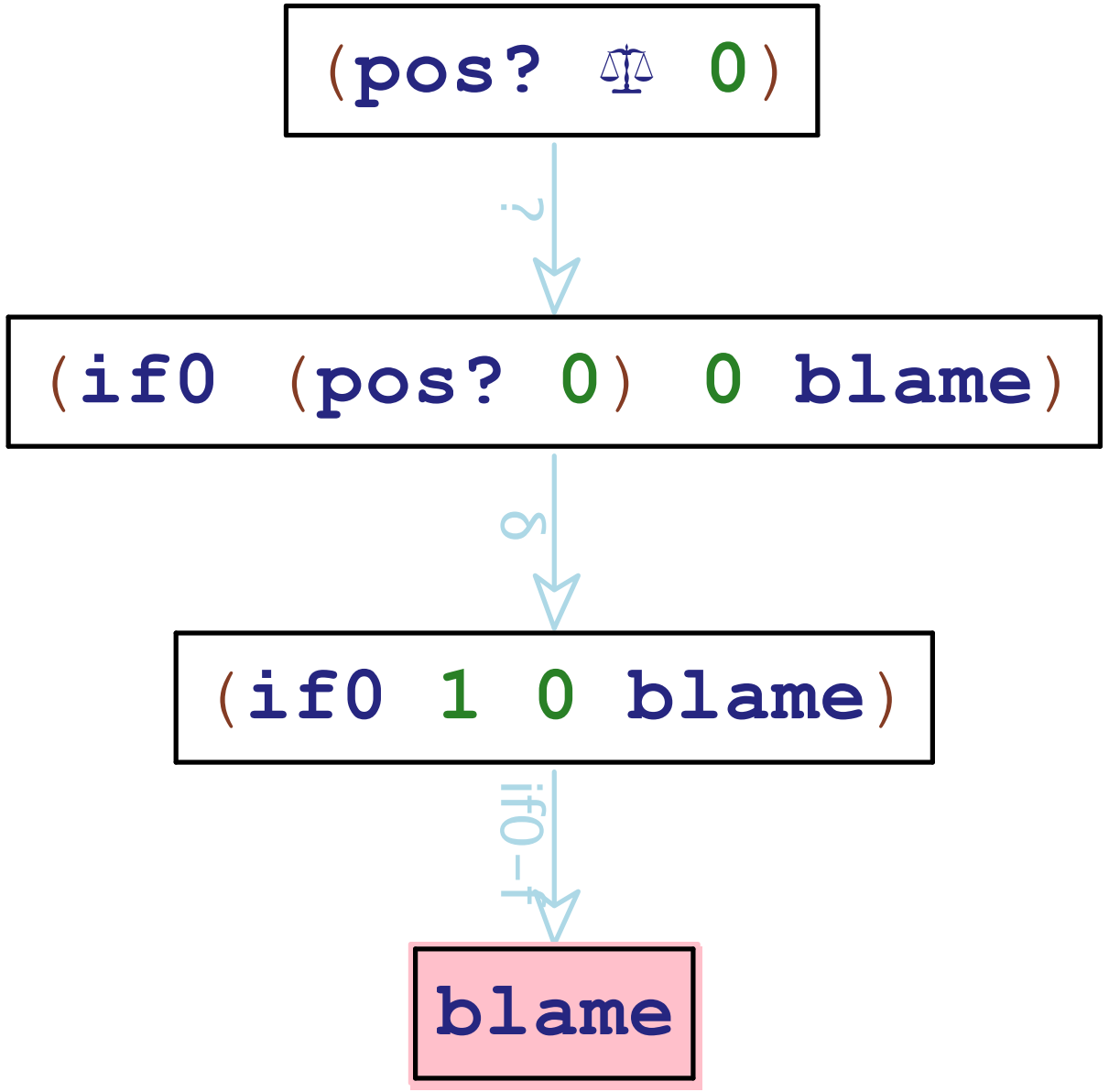
A simple language



A simple language



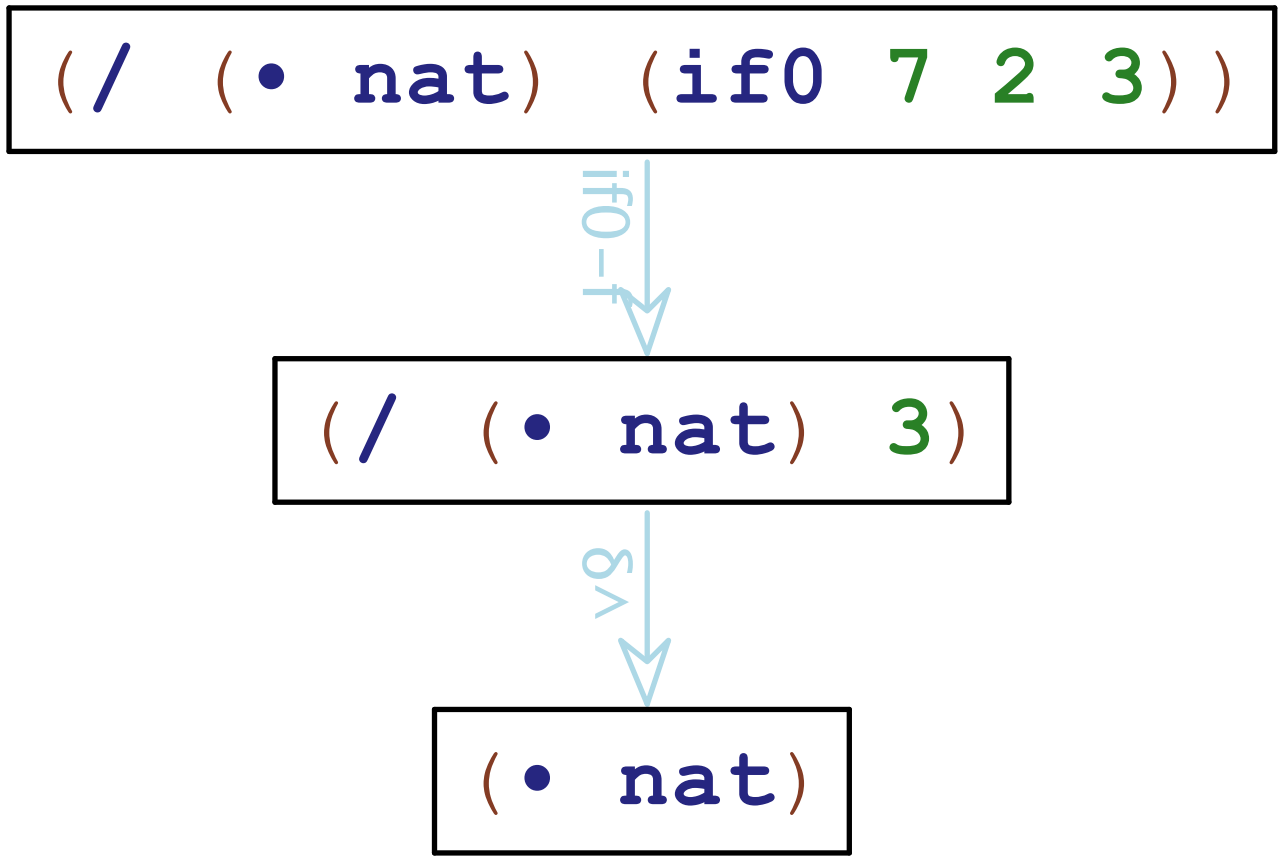
A simple language



With abstract values

```
(/ (• nat) (if0 7 2 3))
```

With abstract values



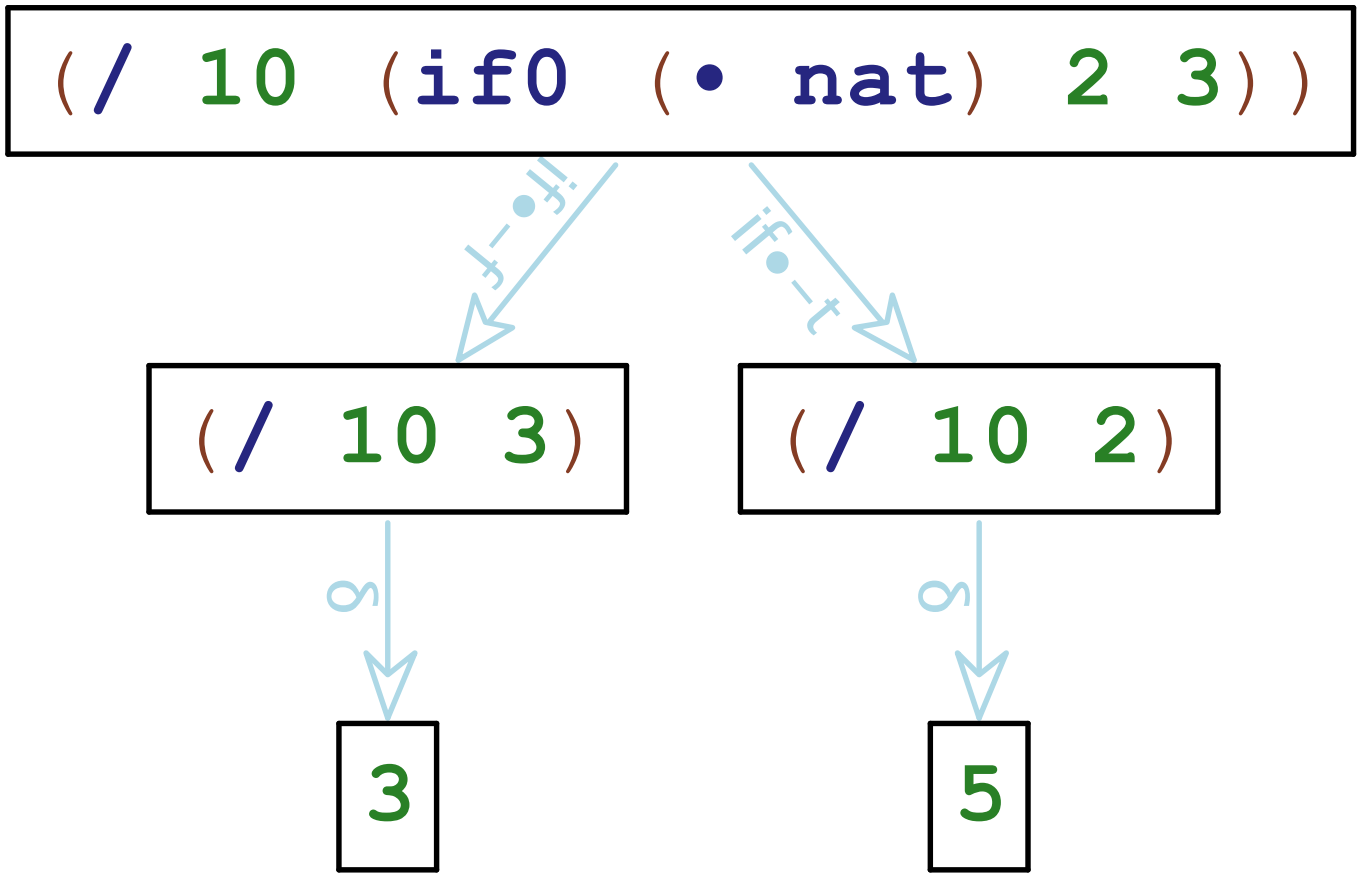
With abstract values

```
(/ 10 (if0 7 2 3))
```

With abstract values

```
(/ 10 (if0 (• nat) 2 3))
```

With abstract values



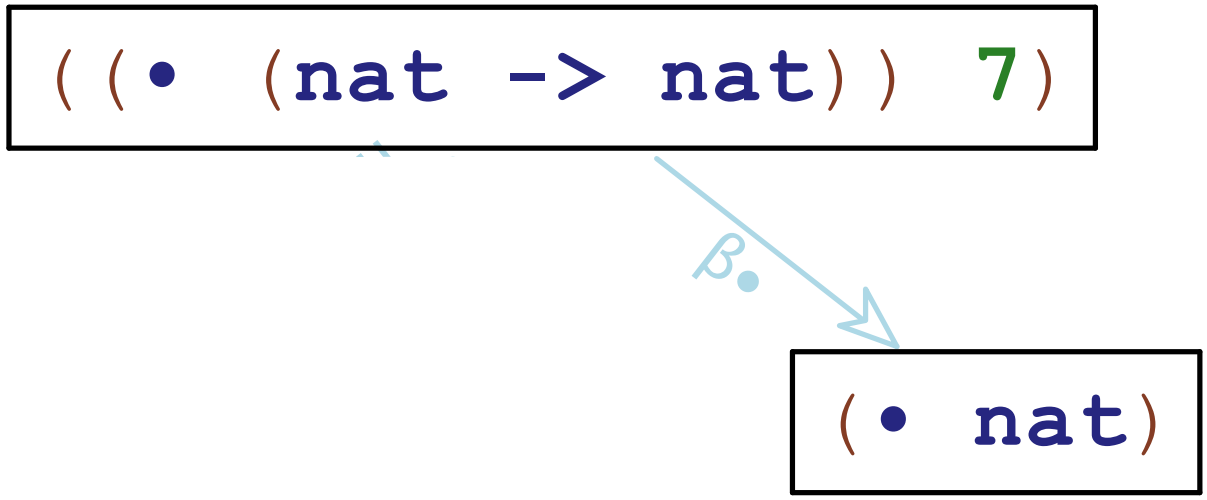
With abstract values

```
(add1 7)
```

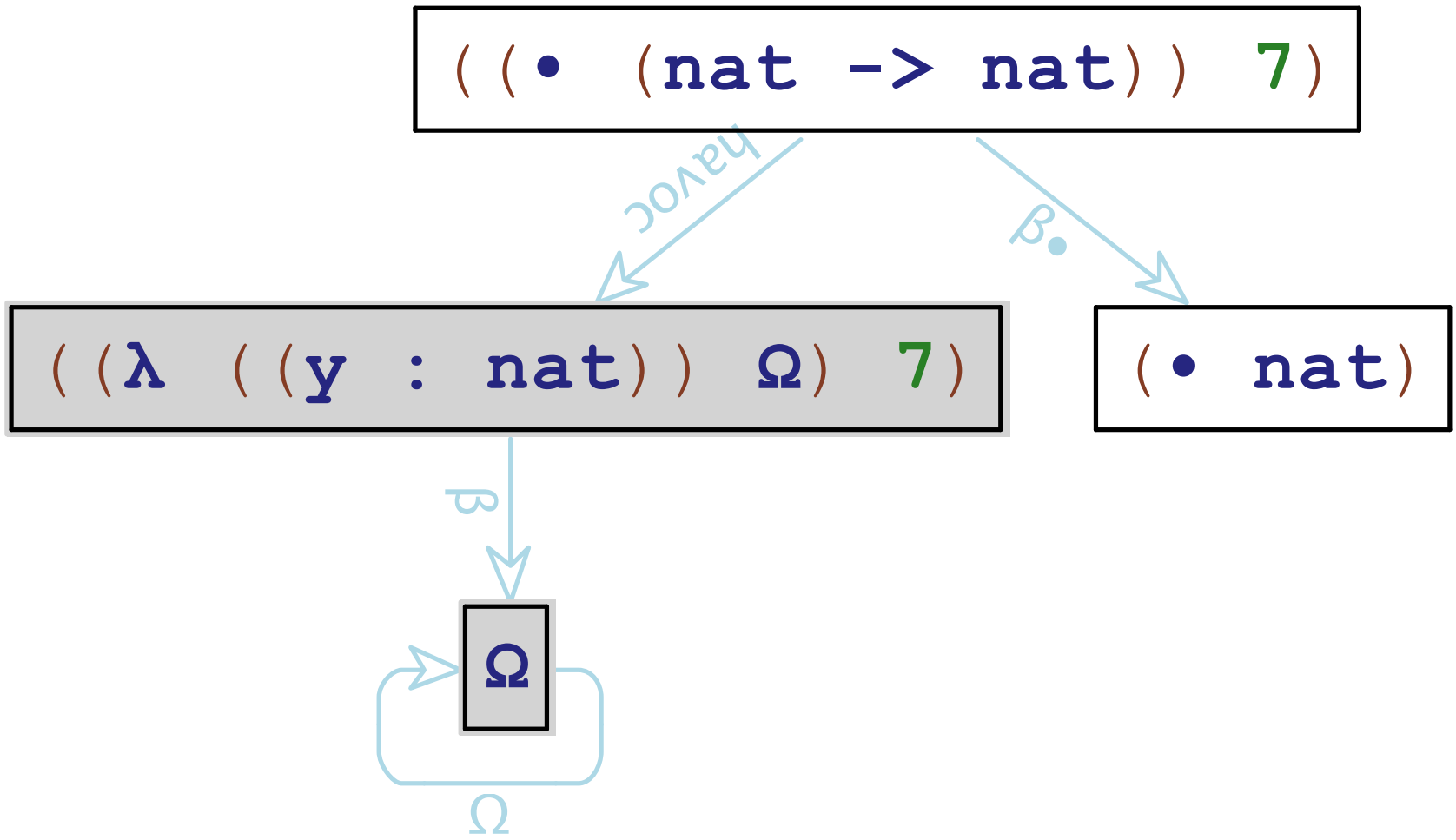
With abstract values

```
((• (nat -> nat)) 7)
```

With abstract values



With abstract values



With abstract values

```
((λ ((f : (nat -> nat))) (f 3))
 (λ ((x : nat)) (+ x (/ 10 x))))
```

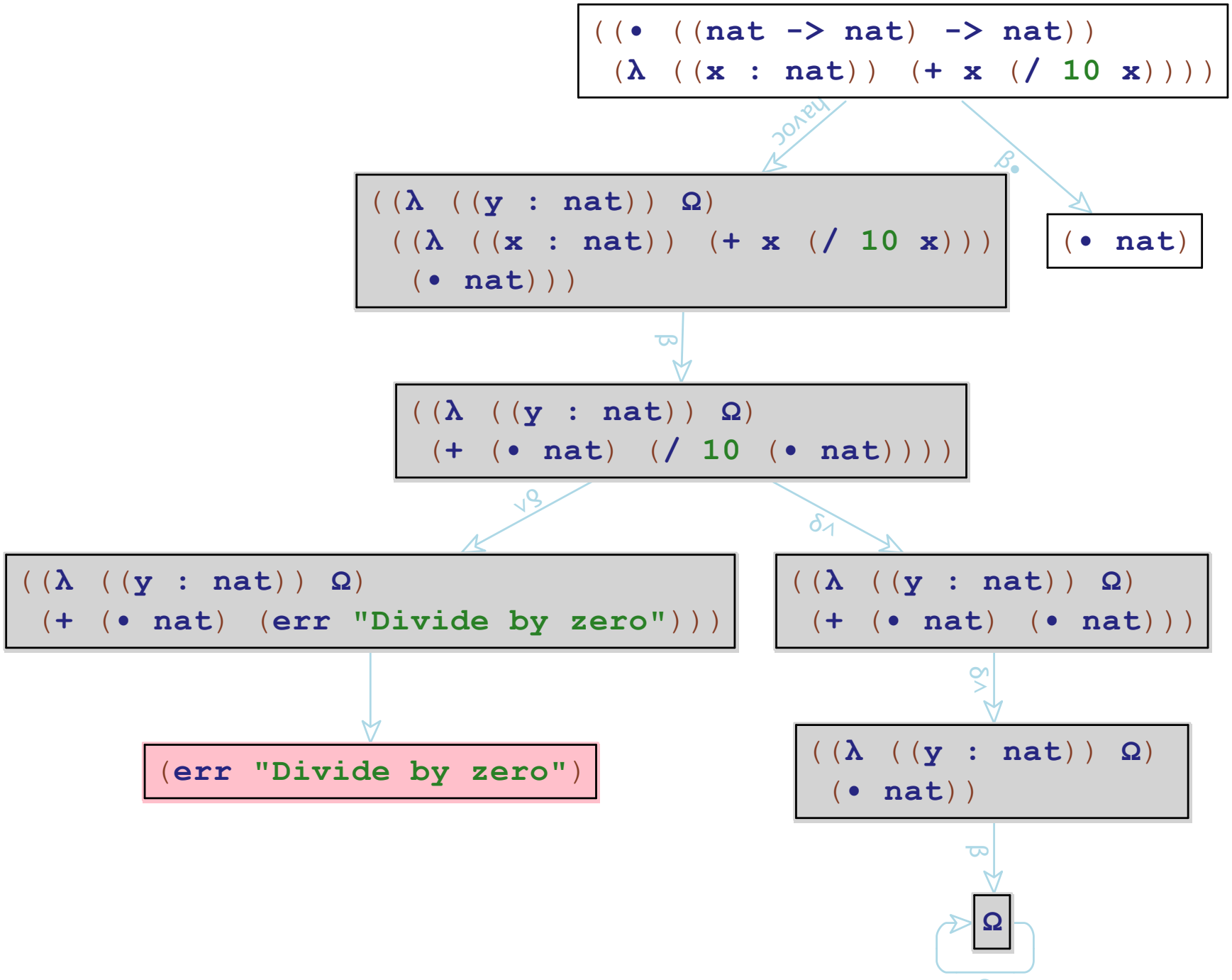
With abstract values

```
((• ((nat -> nat) -> nat))
  (λ ((x : nat)) (+ x (/ 10 x))))
```

With abstract values

```
((• ((nat -> nat) -> nat))
 (λ ((x : nat)) (+ x (/ 10 x))))
```

With abstract values

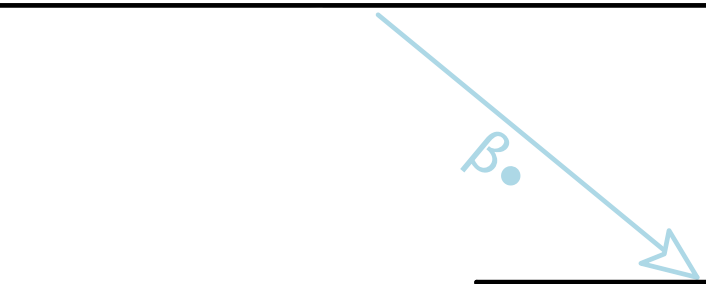


The higher-order case

```
((•  ((nat -> nat) -> nat)
     ((pos? -> pos?) -> pos?))
 (λ ((x : nat)) (+ x (/ 10 x))))
```

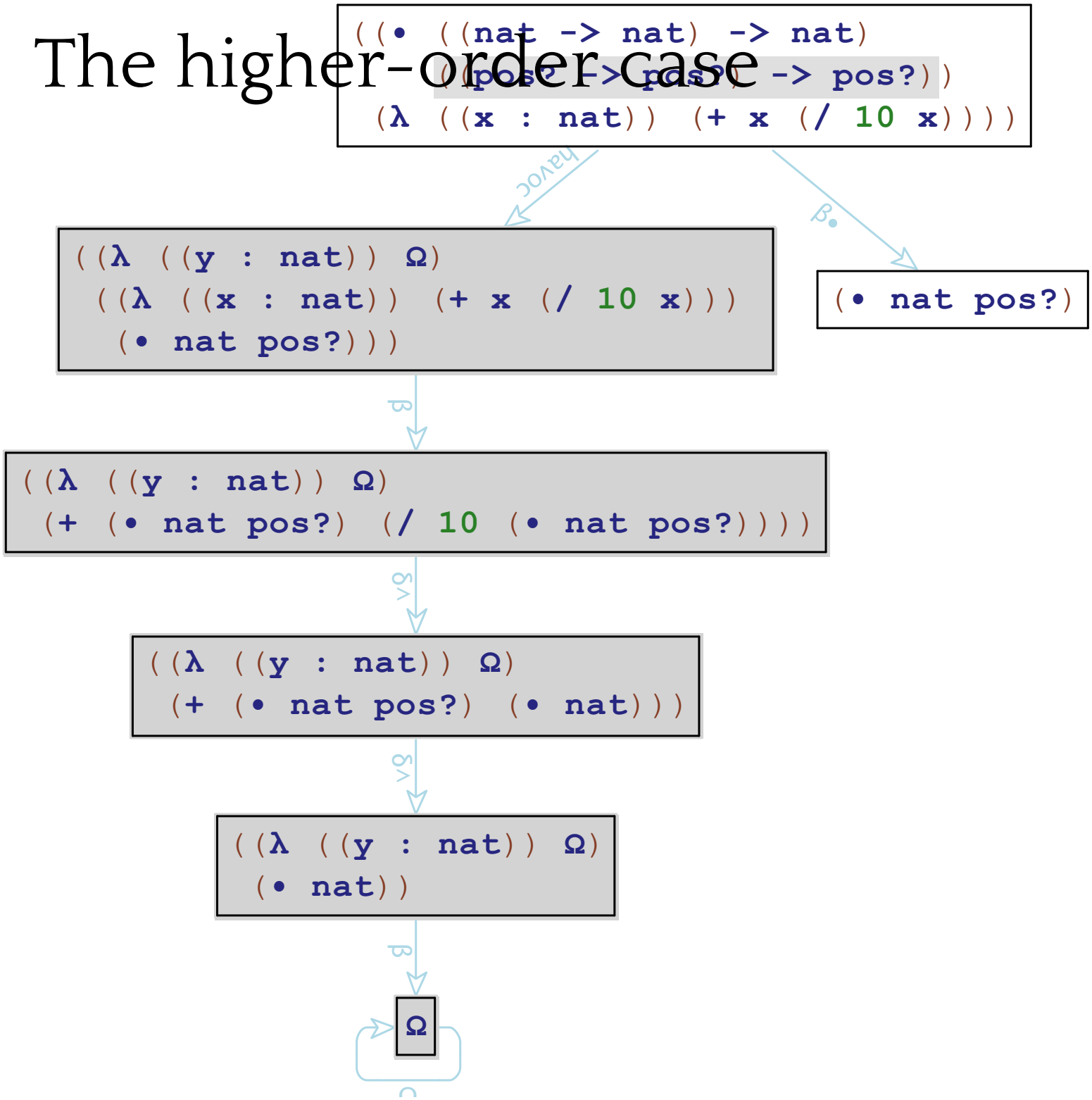
The higher-order case

```
((• ((nat -> nat) -> nat)
    ((pos? -> pos?) -> pos?))
 (λ ((x : nat)) (+ x (/ 10 x))))
```




```
(• nat pos?)
```

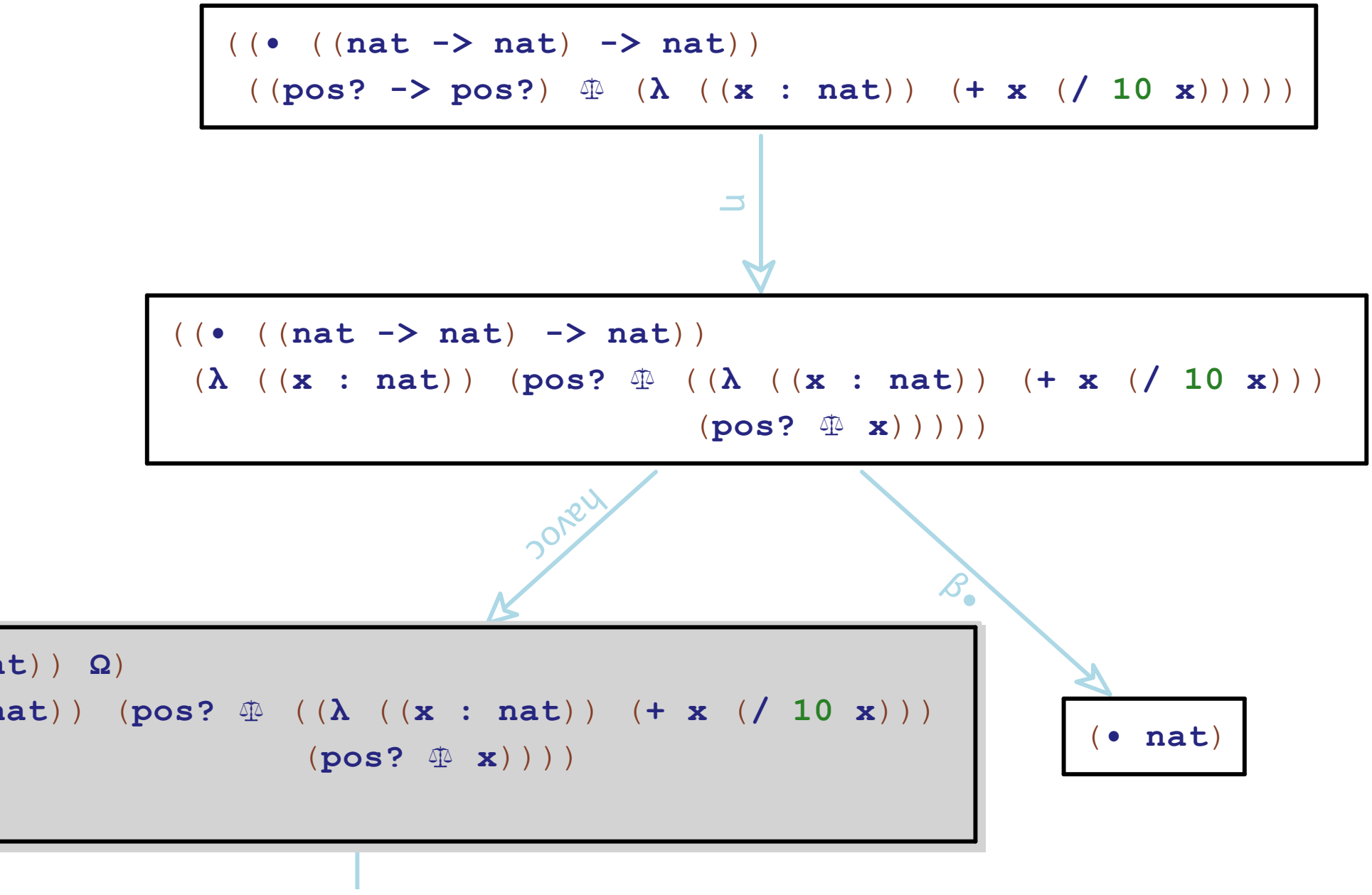
The higher-order case



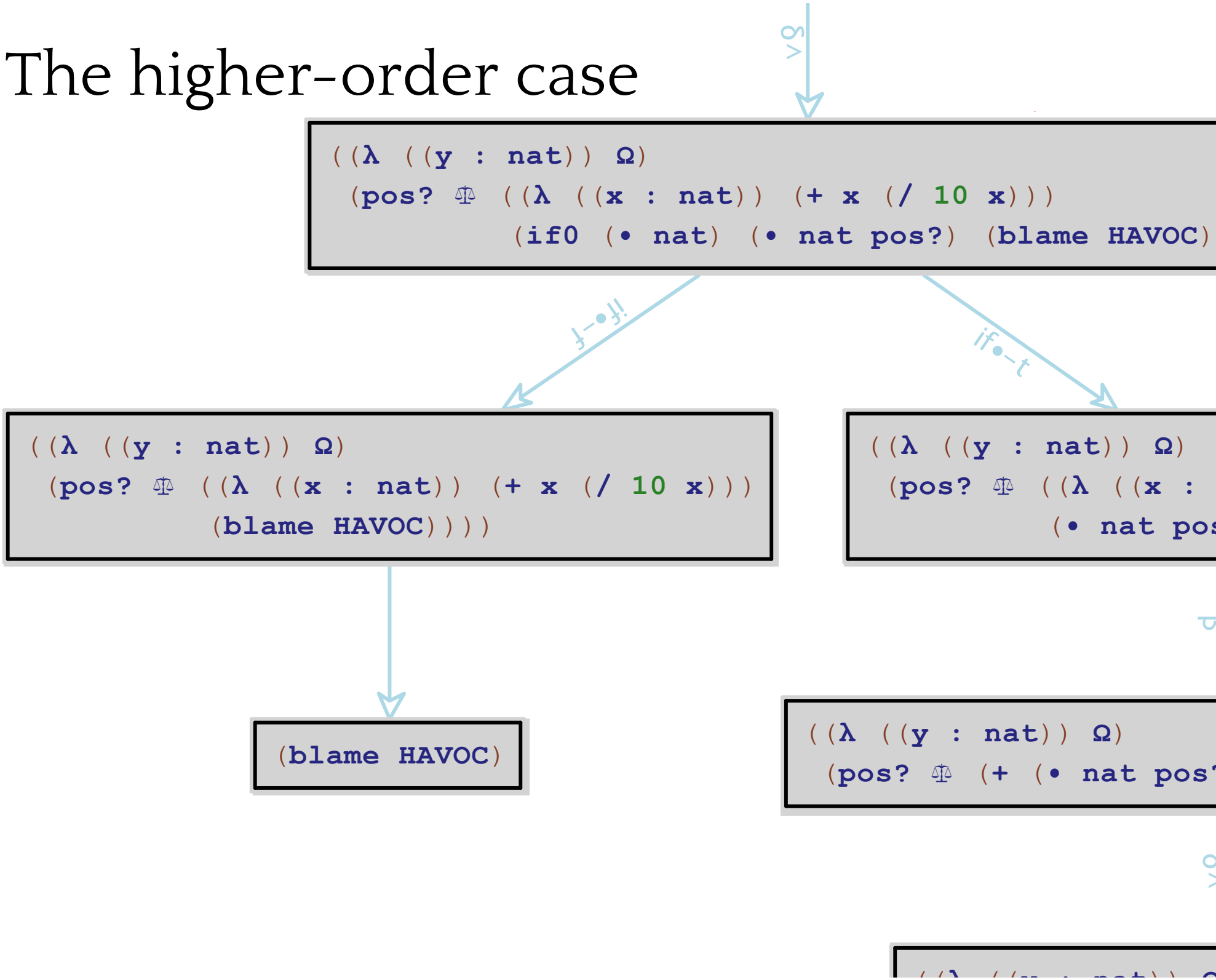
The higher-order case

```
((• ((nat -> nat) -> nat))
 ((pos? -> pos?) ⚖ (λ ((x : nat)) (+ x (/ 10 x)))))
```

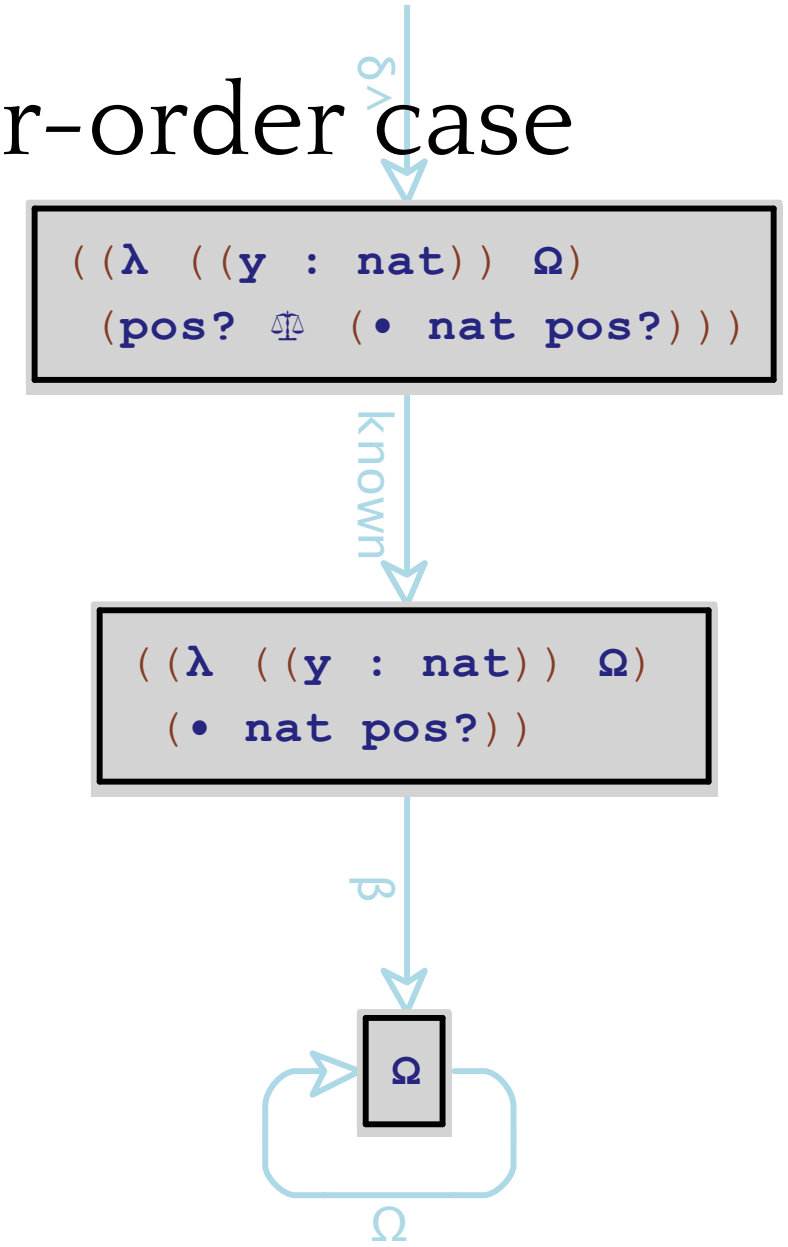
The higher-order case



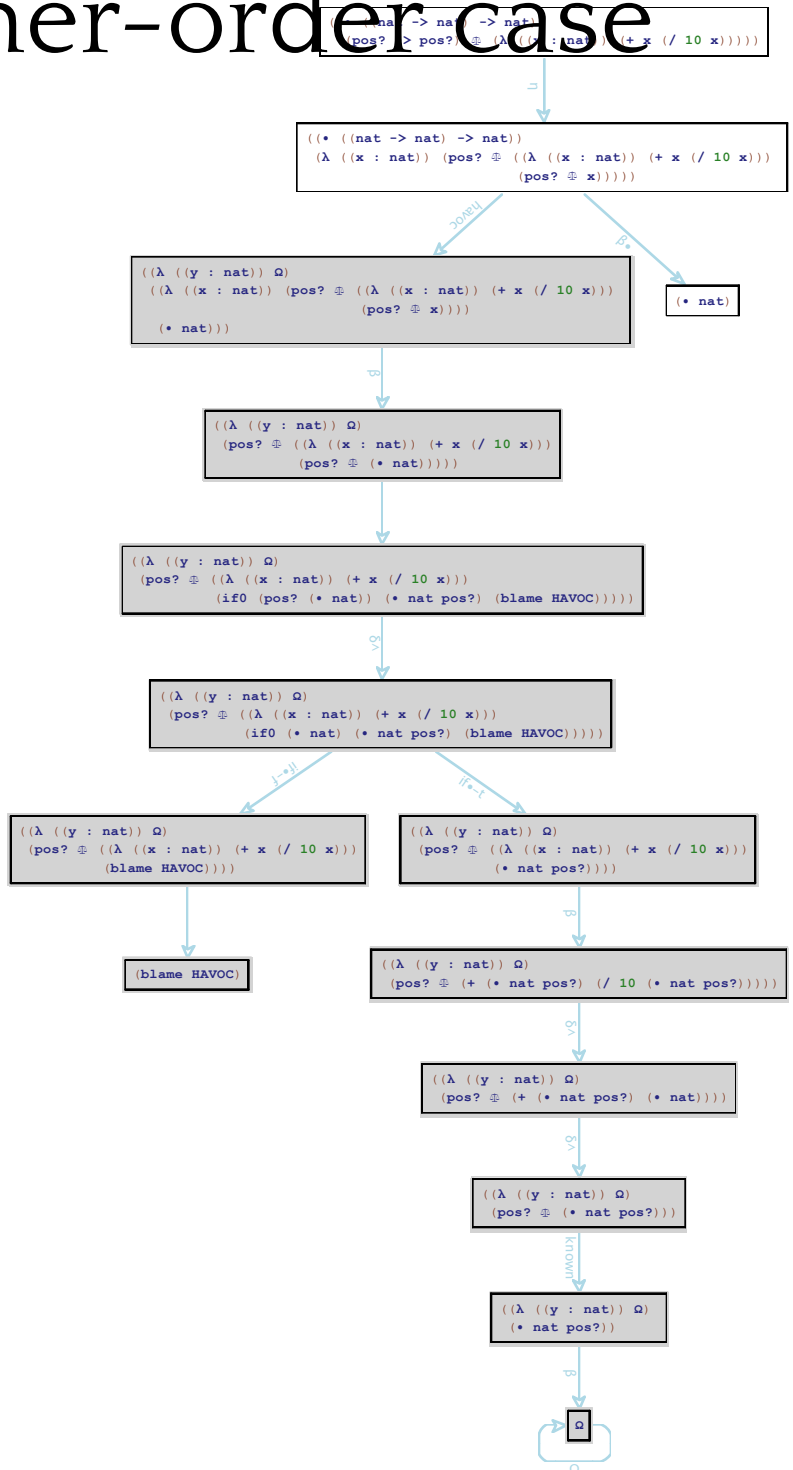
The higher-order case

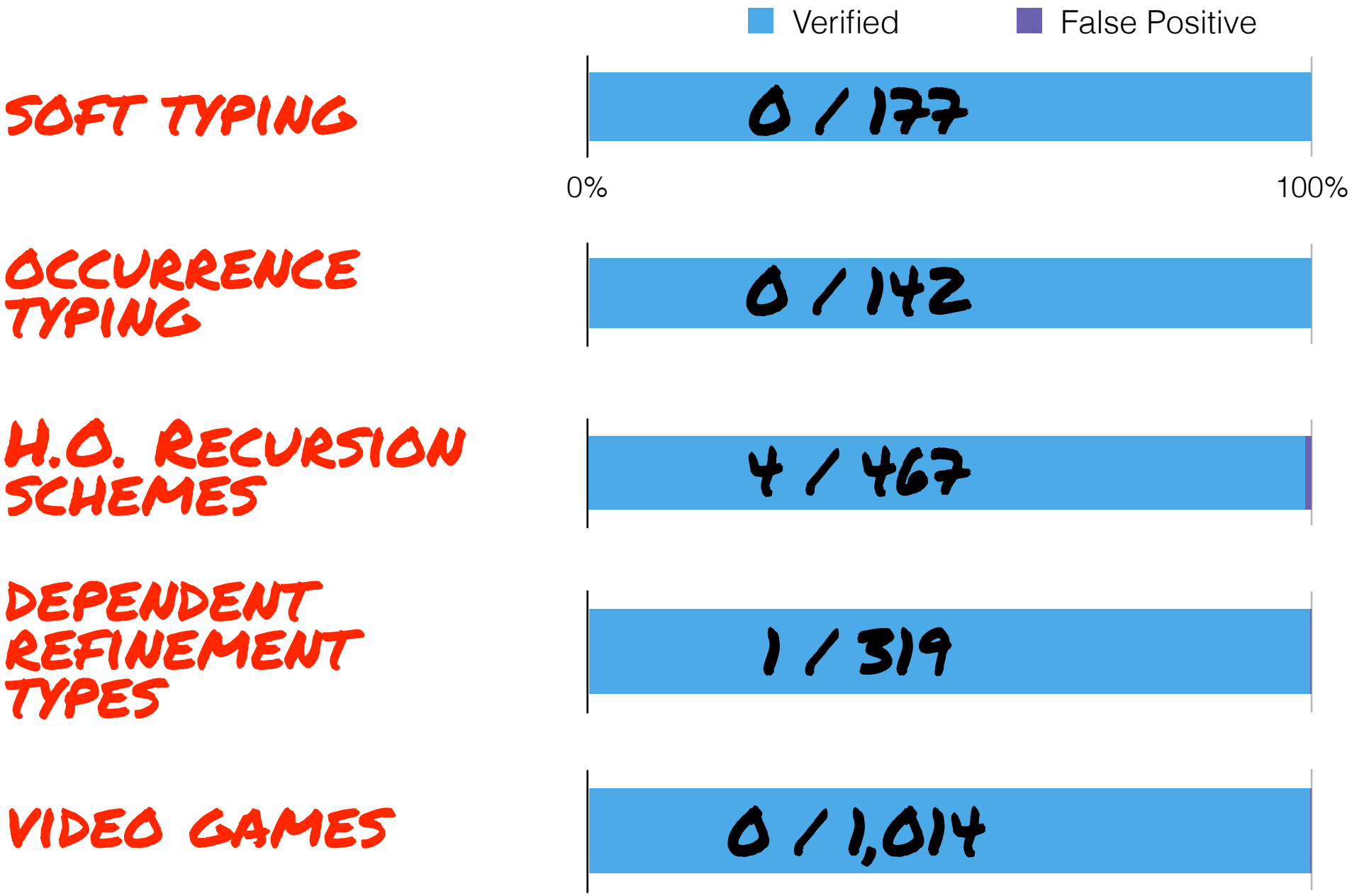


The higher-order case

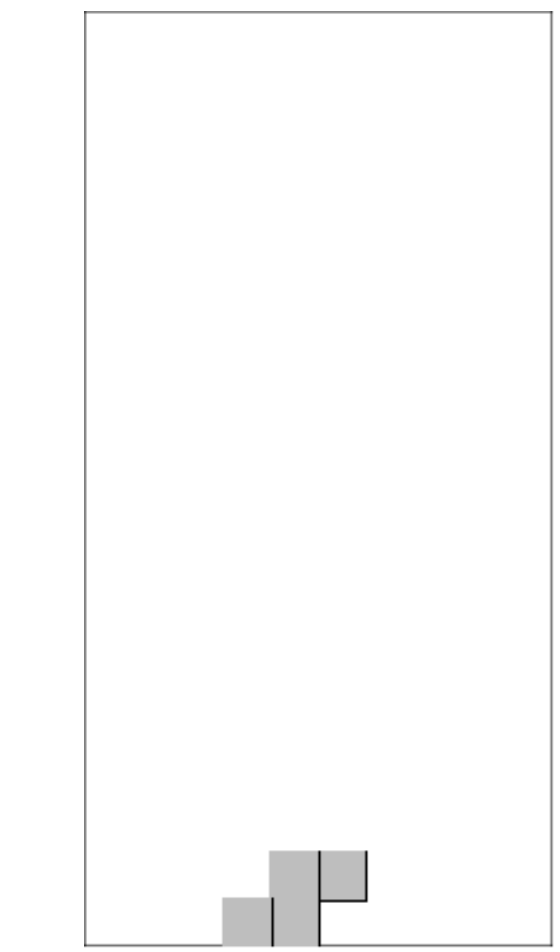


The higher-order case

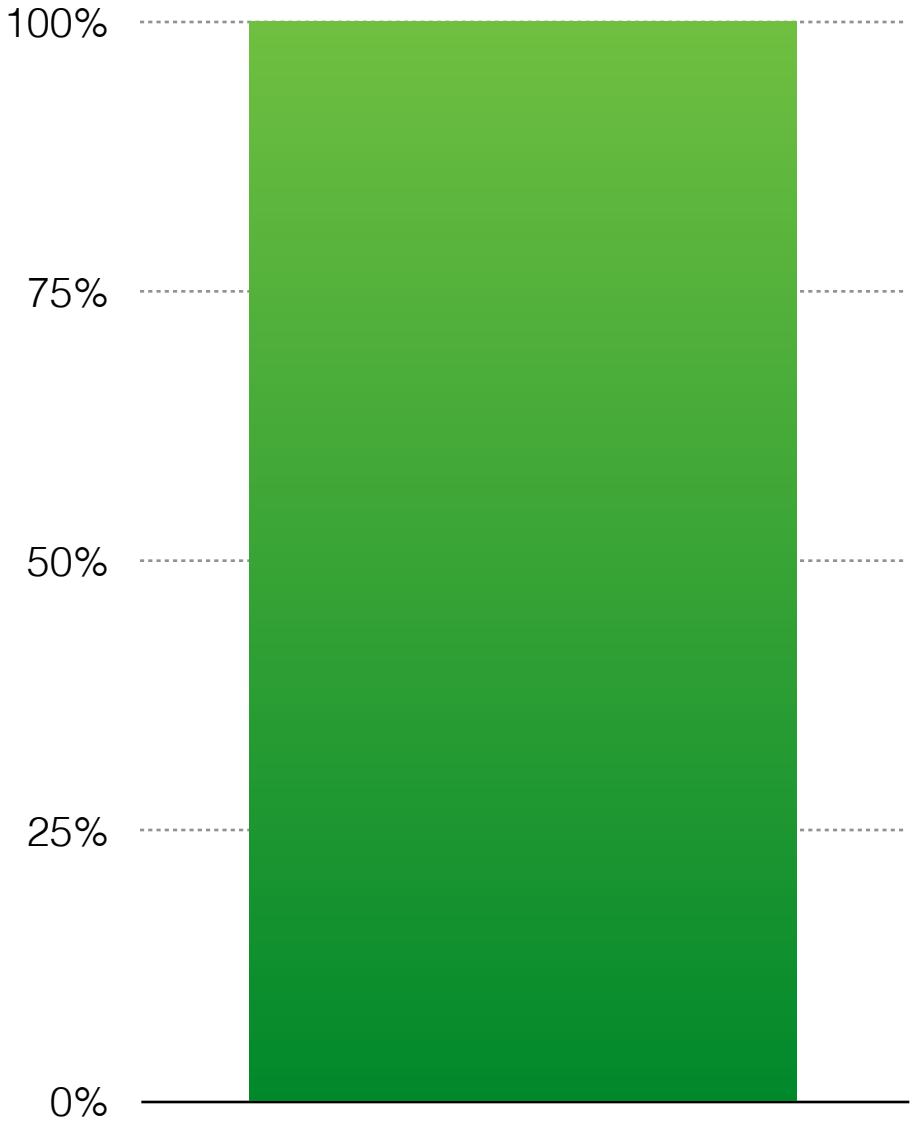


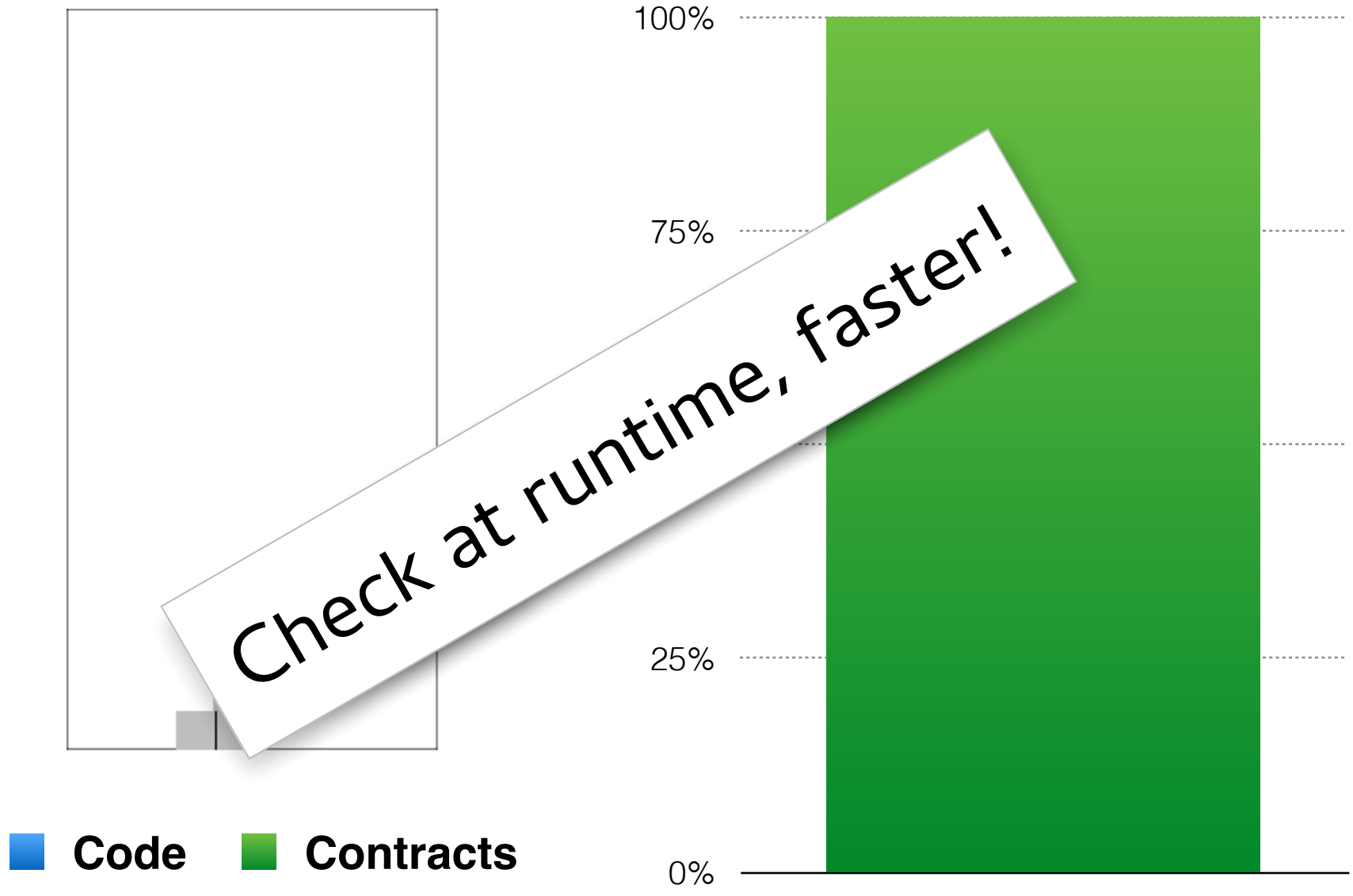

Verified
False Positive
SOFT TYPING
0 / 177
0%
100%
OCCURRENCE TYPING
0 / 142
H.O. RECURSION SCHEMES
4 / 467
DEPENDENT REFINEMENT TYPES
1 / 319
VIDEO GAMES
0 / 1,014

Optimization



Code Contracts




100%
75%
25%
0%
Check at runtime, faster!
Code
Contracts

A simpler program

```
(define (dot u v)
  (for/sum ([x u]
            [y v])
    (* x y)))
```

A simpler program

```
(define (dot u v)
  (for/sum ([x u]
            [y v])
    (* x y)))
```

506 ms (size 10000000)

A simpler program

```
(define/contract (dot u v)
  ((vectorof flonum?)
   (vectorof flonum?)
   . -> . flonum?)
  (for/sum ([x u] [y v])
    (* x y)))
```

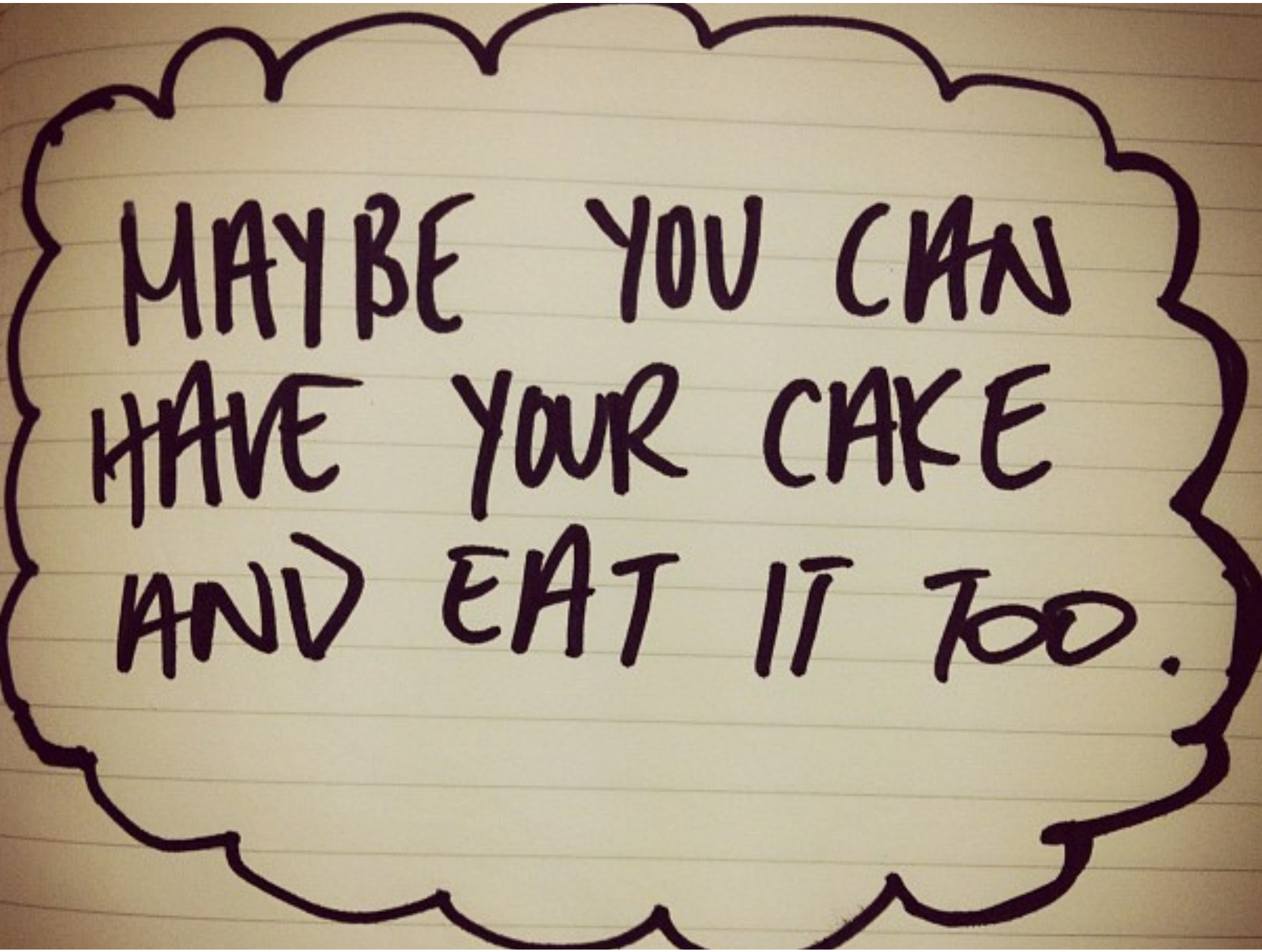
1159 ms (size 10000000)

Why are contracts hard to optimize?

```
(contract (-> integer? integer?)
          (lambda (x) x))
```

Why are contracts hard to optimize?

```
(chaperone-procedure
 (lambda (x) x)
 (lambda (v)
   (unless (integer? v) (error 'blame))
   v)
 (lambda (v)
   (unless (integer? v) (error 'blame))
   v))
```


MAYBE YOU CAN
HAVE YOUR CAKE
AND EAT IT TOO.

With Pycket ...

```
(define (dot u v)
  (for/sum ([x u]
            [y v])
    (* x y)))
```

12 ms (size 10000000)

With Pycket ...

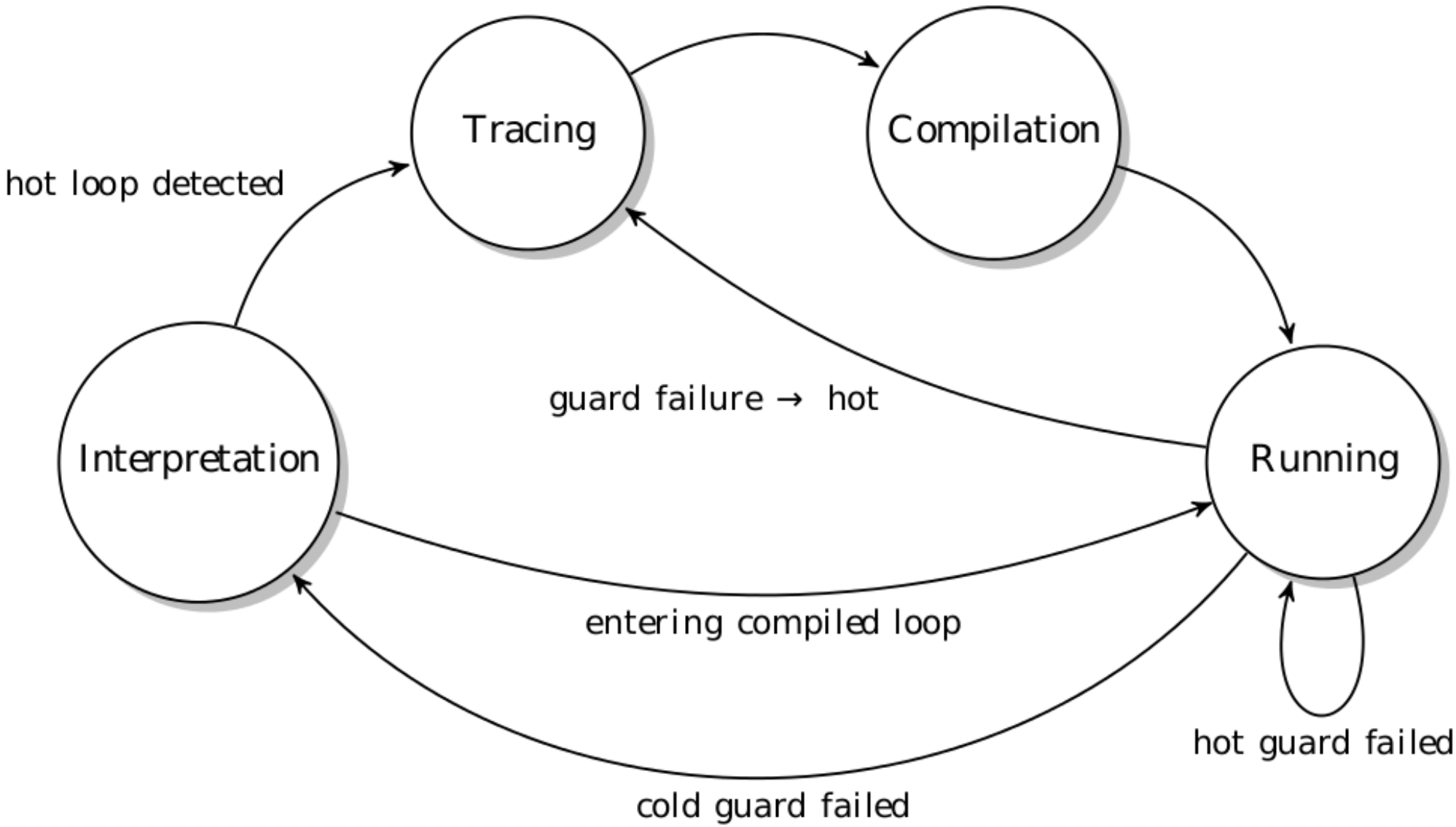
```
(define/contract (dot u v)
  ((vectorof flonum?)
   (vectorof flonum?)
   . -> . flonum?)
  (for/sum ([x u] [y v])
    (* x y)))
```

17 ms (size 10000000)

Tracing JIT

1. Interpret Program
2. Find hot loop
3. Record operations for one iteration
4. Optimize
5. Switch to new code

Tracing JIT



(Diagram from Antonio Cuni)

Dot product Inner Loop

loop header

```
label(p3, f58, i66, i70, p1, i17, i28, p38, p48)
guard_not_invalidated()
```

loop termination tests

```
i71 = i66 < i17
guard(i71 is true)
i72 = i70 < i28
guard(i72 is true)
```

vector access

```
f73 = getarrayitem_gc(p38, i66)
f74 = getarrayitem_gc(p48, i70)
```

core operations

```
f75 = f73 * f74
f76 = f58 + f75
```

increment loop counters

```
i77 = i66 + 1
i78 = i70 + 1
```

jump back to loop header

```
jump(p3, f76, i77, i78, p1, i17, i28, p38, p48)
```

Key Optimizations

Inlining (happens for free)

Constant propagation

Allocation Removal

We didn't write a JIT or an optimizer!

We didn't write a JIT or an optimizer!

RPython creates a JIT from an interpreter

CEK Machine

$$e ::= x \mid \lambda x.\, e \mid e\ e$$

$$\kappa ::= [] \mid \mathsf{arg}(e, \rho)::\kappa \mid \mathsf{fun}(v, \rho)::\kappa$$

$$\langle x, \rho, \kappa \rangle \longmapsto \langle \rho(x), \rho, \kappa \rangle$$

$$\langle (e_1\ e_2), \rho, \kappa \rangle \longmapsto \langle e_1, \rho, \mathsf{arg}(e_2, \rho)::\kappa \rangle$$

$$\langle v, \rho, \mathsf{arg}(e, \rho')::\kappa \rangle \longmapsto \langle e, \rho', \mathsf{fun}(v, \rho)::\kappa \rangle$$

$$\langle v, \rho, \mathsf{fun}(\lambda x.\, e, \rho')::\kappa \rangle \longmapsto \langle e, \rho'[x \mapsto v], \kappa \rangle$$

CEK Advantages

- Fast continuations
- Tail recursion
- Arbitrary size stack

CEK Advantages

Fast continuations

Tail recursion

Arbitrary size stack

Allocation everywhere

From CEK to JIT

1. Whole-program type inference
2. Translation to C
3. Adding JIT based on hints

Main Interpreter Loop

```
try:
    while True:
        driver.jit_merge_point()
        if isinstance(ast, App):
            prev = ast
        ast, env, cont = ast.interpret(env, cont)
        if ast.should_enter:
            driver.can_enter_jit()
except Done, e:
    return e.values
```

	Pycket	±	Pycket*	±	Racket	±	V8	±	PyPy	±
Bubble										
direct	640	1	656	1	1384	4	336	0	593	1
chaperone	768	0	778	1	6668	5				
proxy							105891	2579	1153	8
unsafe	496	1	550	1	955	1				
unsafe*	495	0	508	1	726	1				
Church										
direct	714	2	705	1	1243	6	2145	18	3263	14
chaperone	6079	54	8400	34	38497	66				
contract	1143	6	2310	8	10126	142	295452	1905		
proxy							53953	277	95391	848
wrap	3471	7	3213	5	4214	26	8731	45	59016	405
Struct										
direct	133	0	133	0	527	0	377	0	127	0
chaperone	134	0	134	1	5664	68				
proxy							26268	130	1168	38
unsafe	133	0	133	0	337	0				
unsafe*	133	0	133	0	337	0				
ODE										
direct	2158	6	2645	6	5476	91				
contract	2467	8	5099	8	12235	128				
Binomial										
direct	1439	8	6879	83	2931	24				
contract	17749	83	19288	61	52827	507				

Contracts: a useful techology and a source of research ideas

samth@indiana.edu

samth.github.io