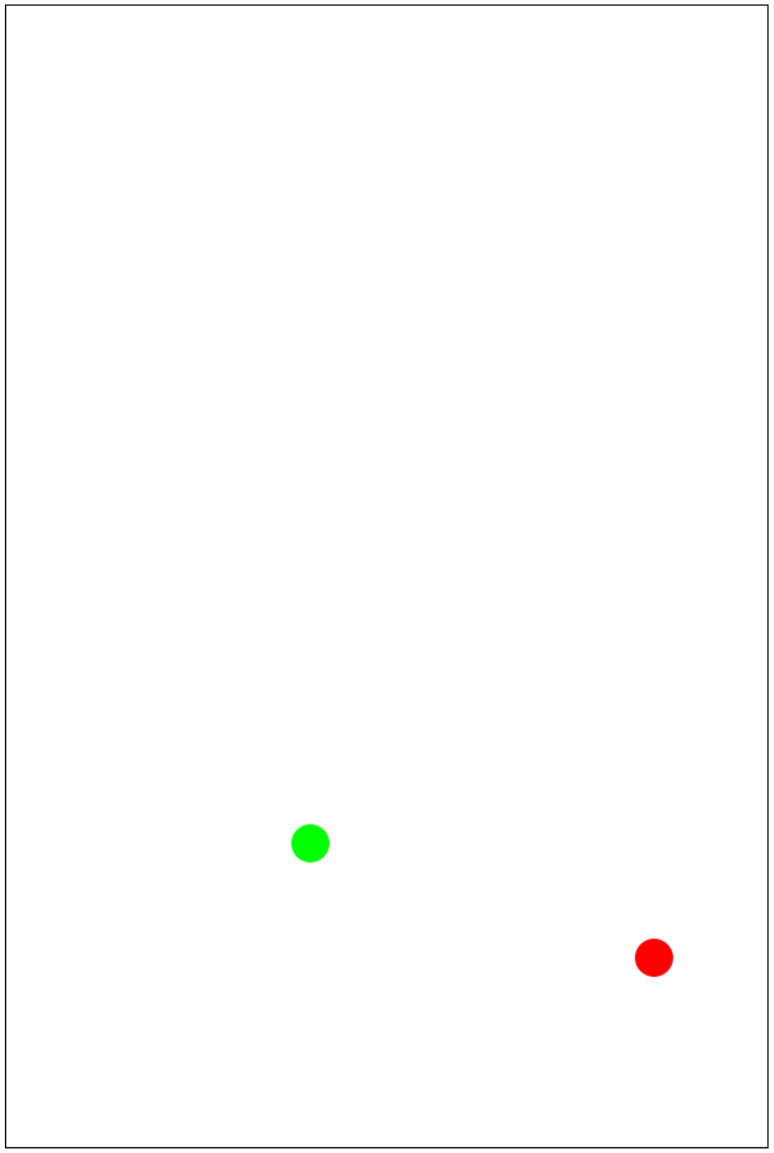
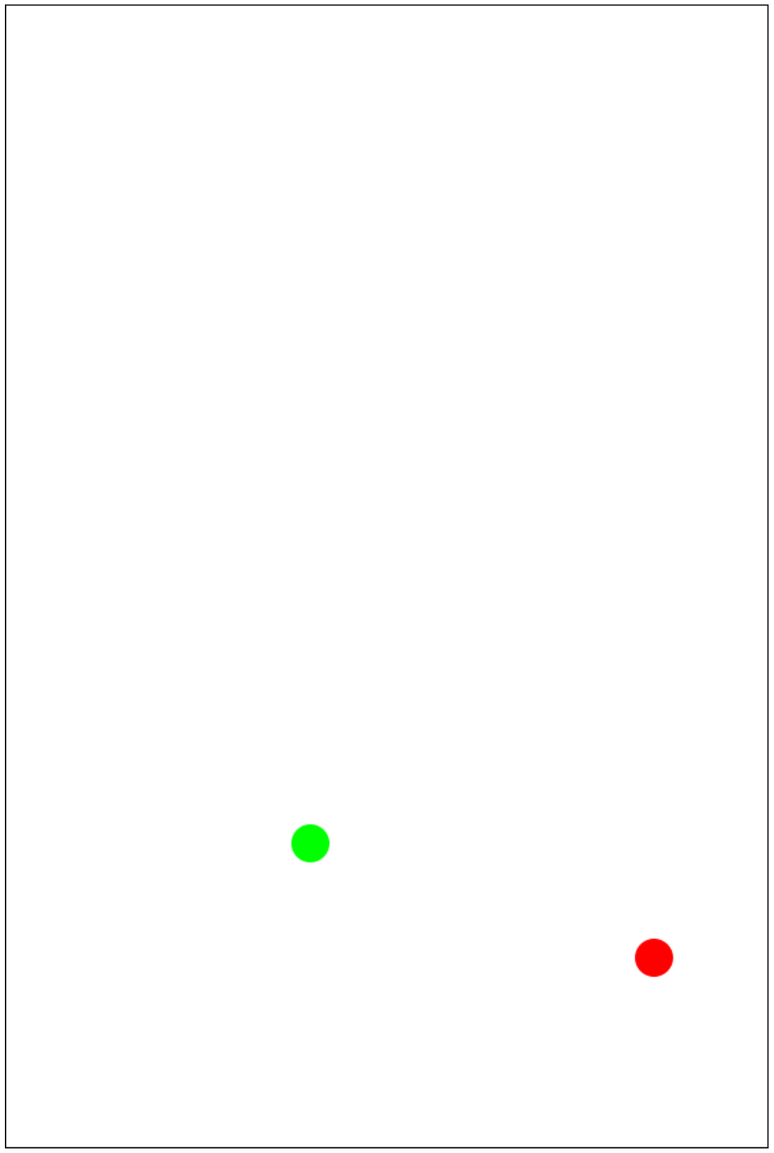
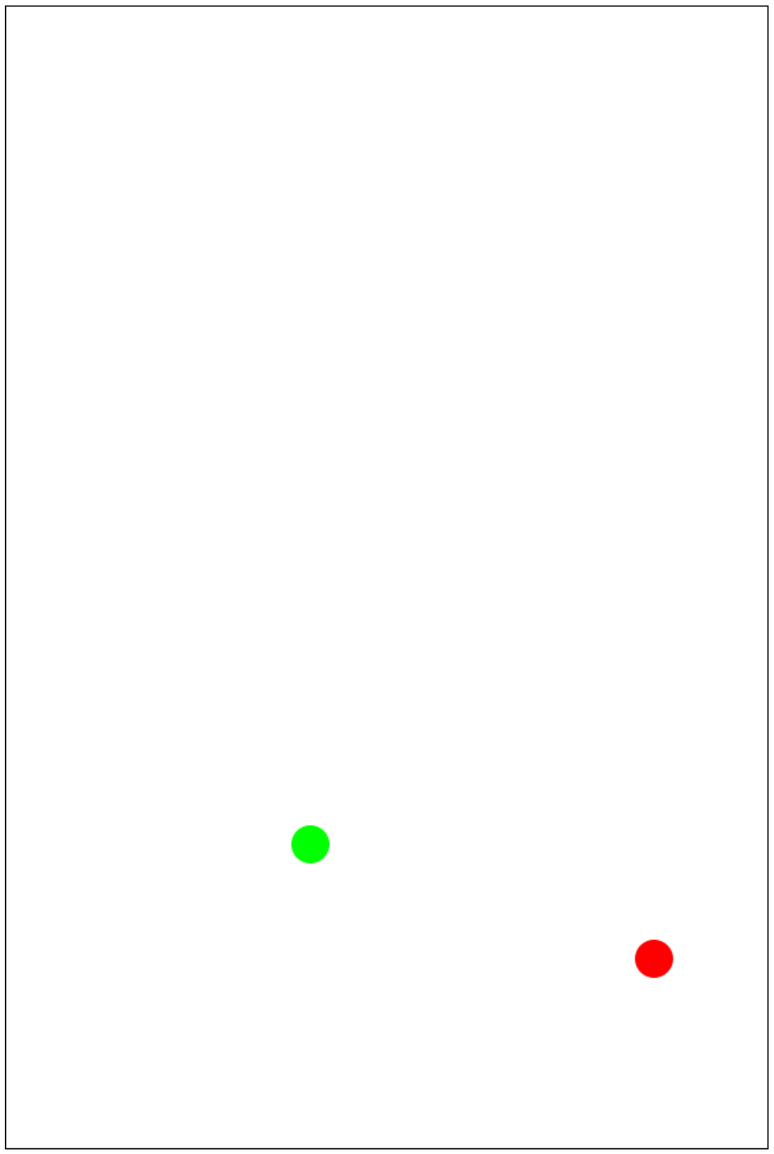
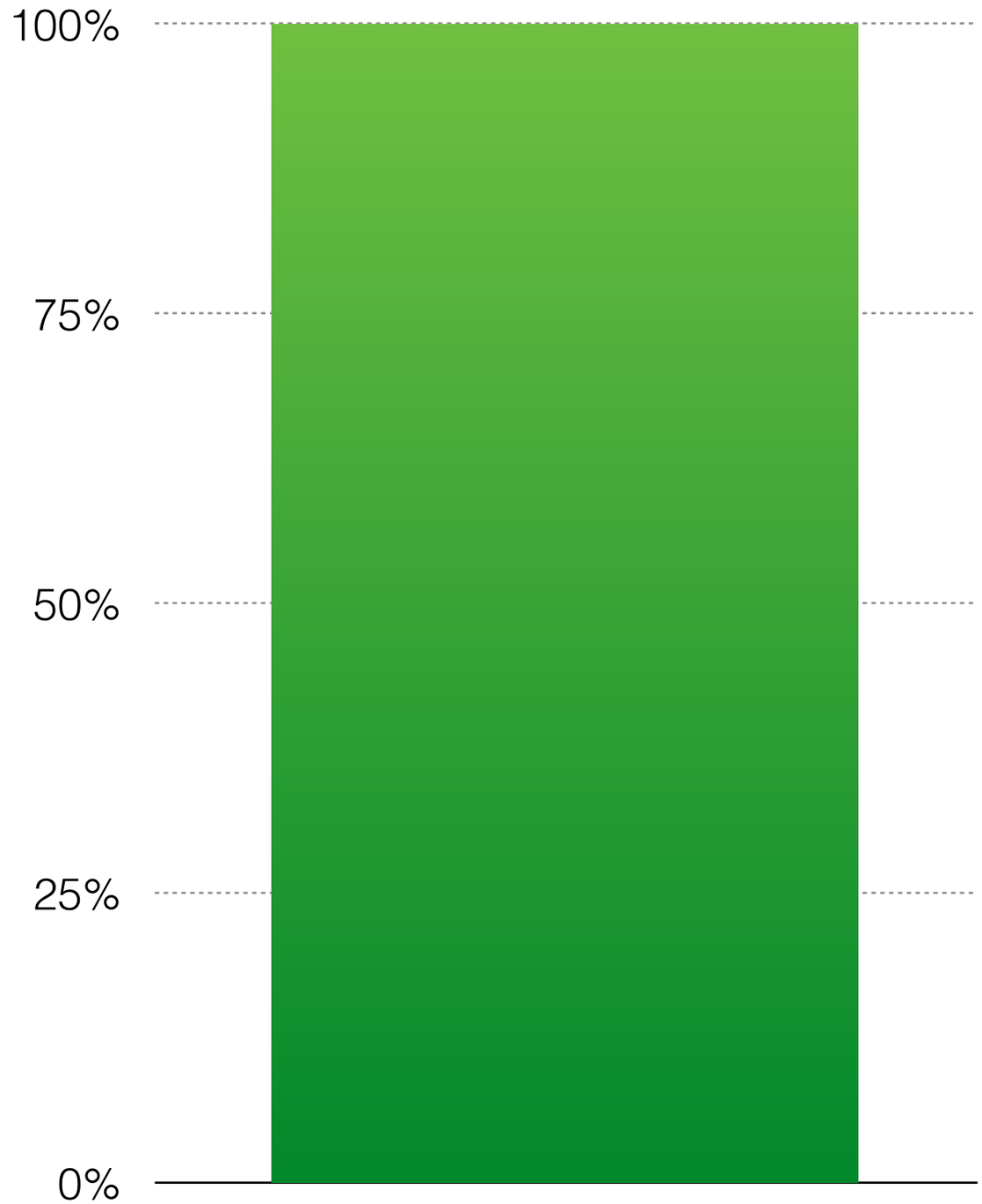
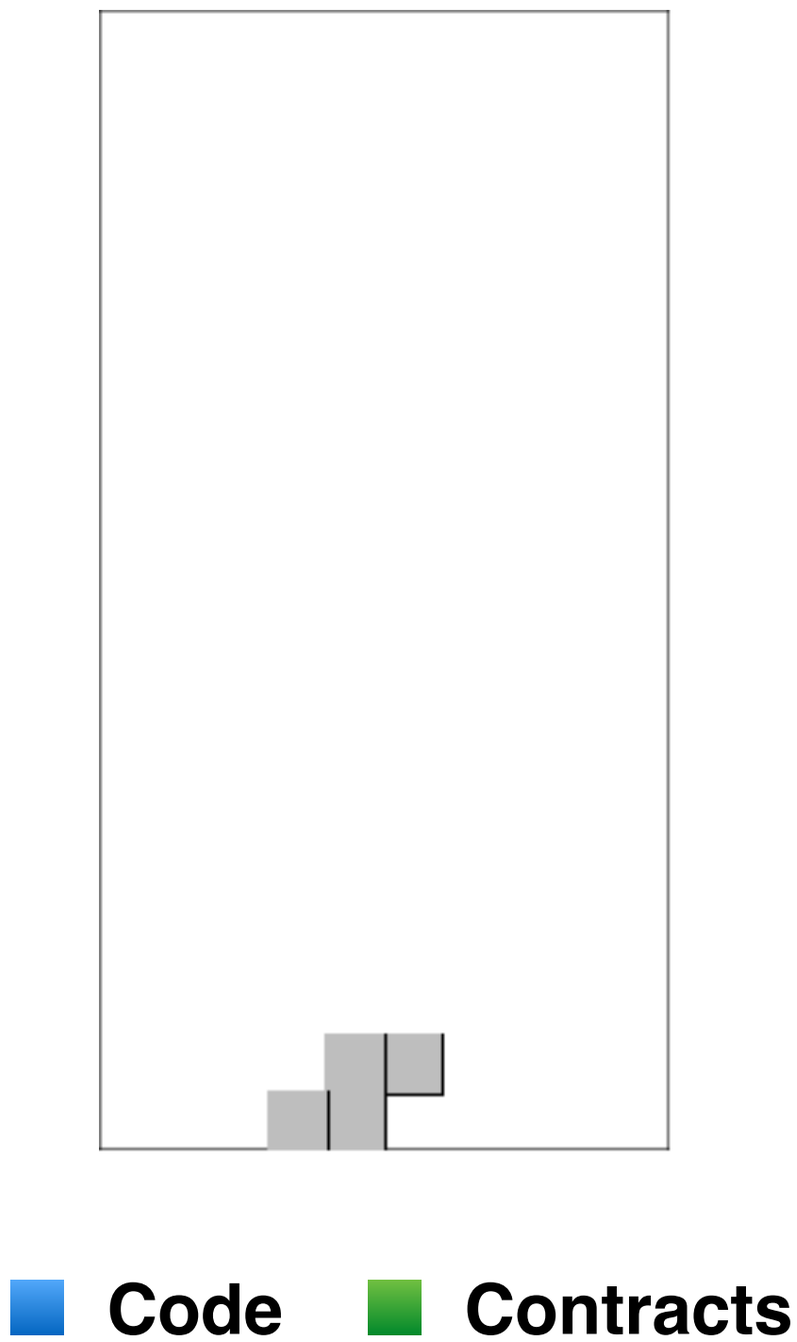


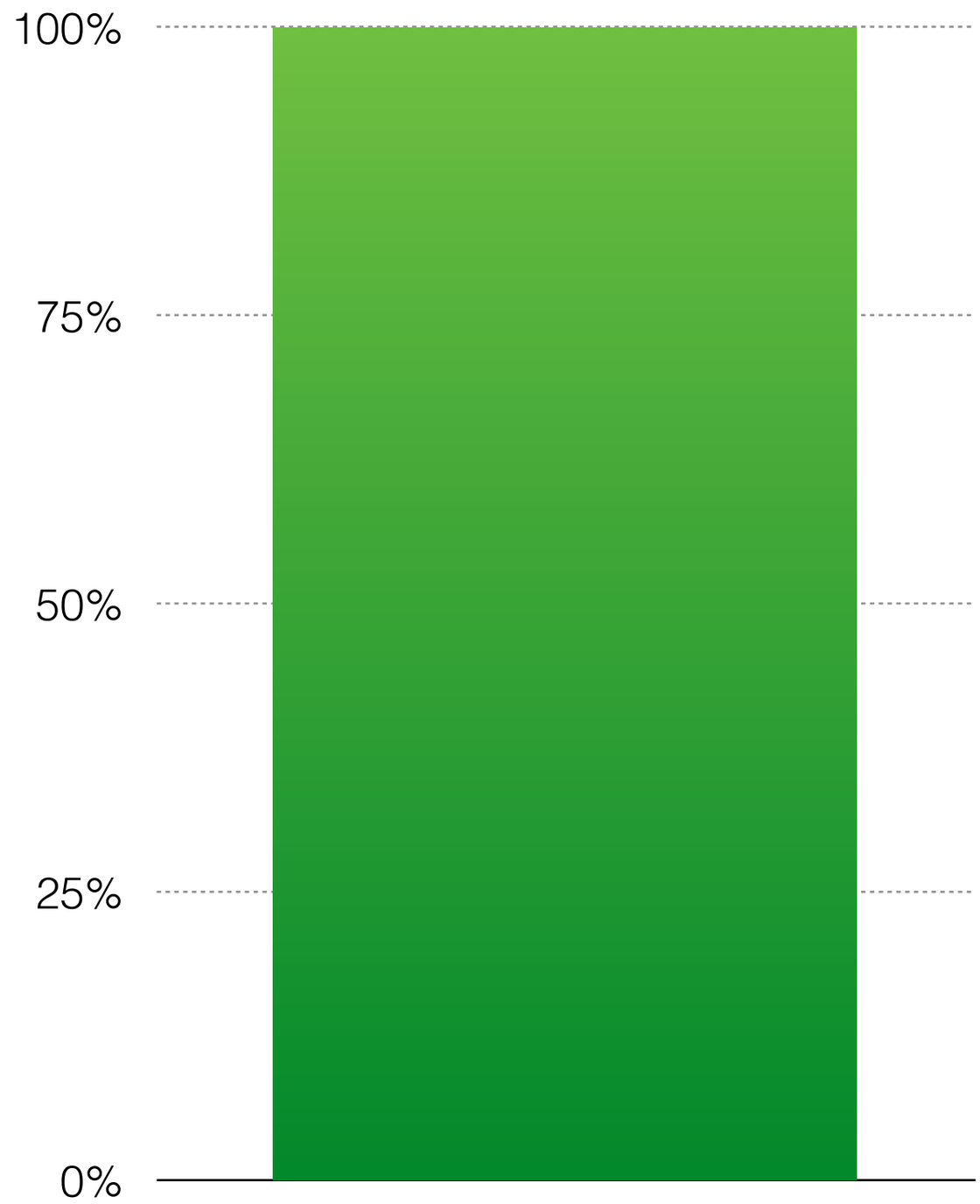


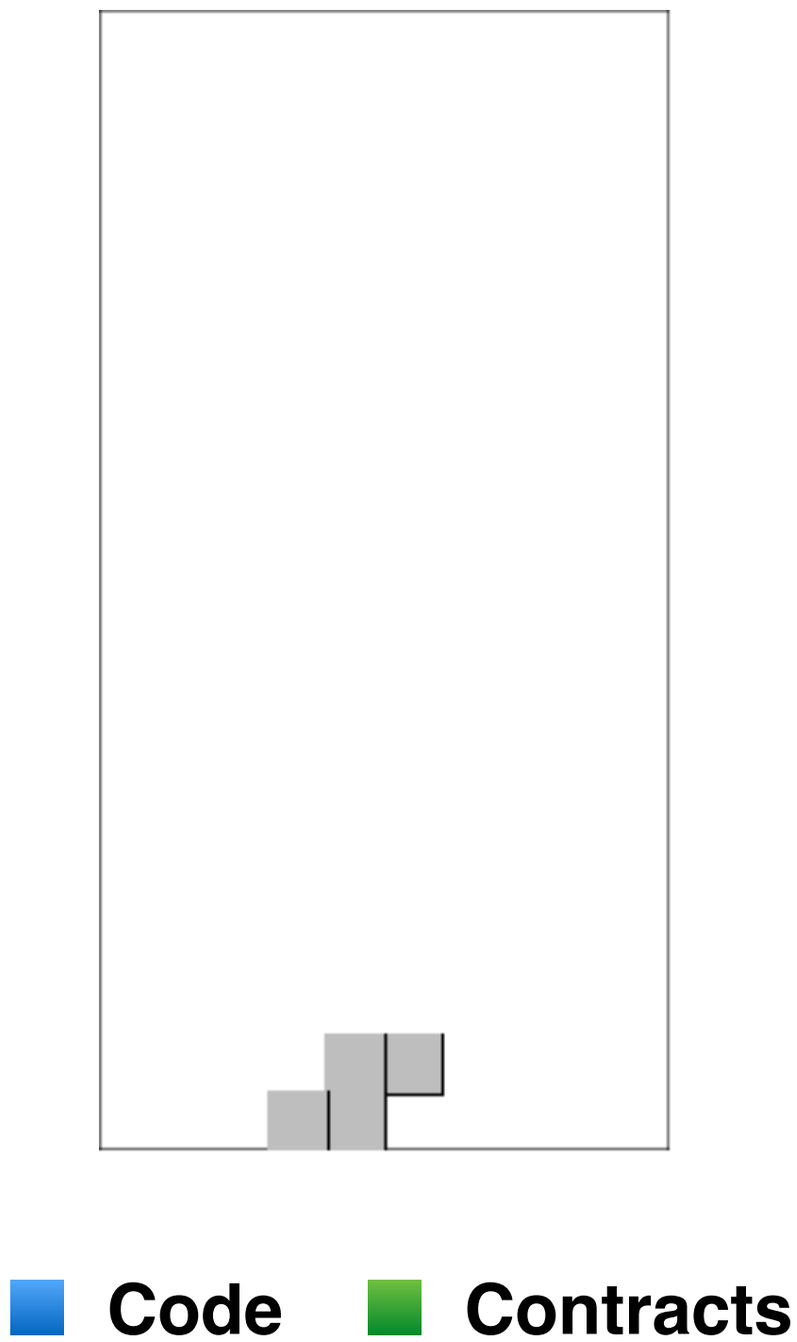


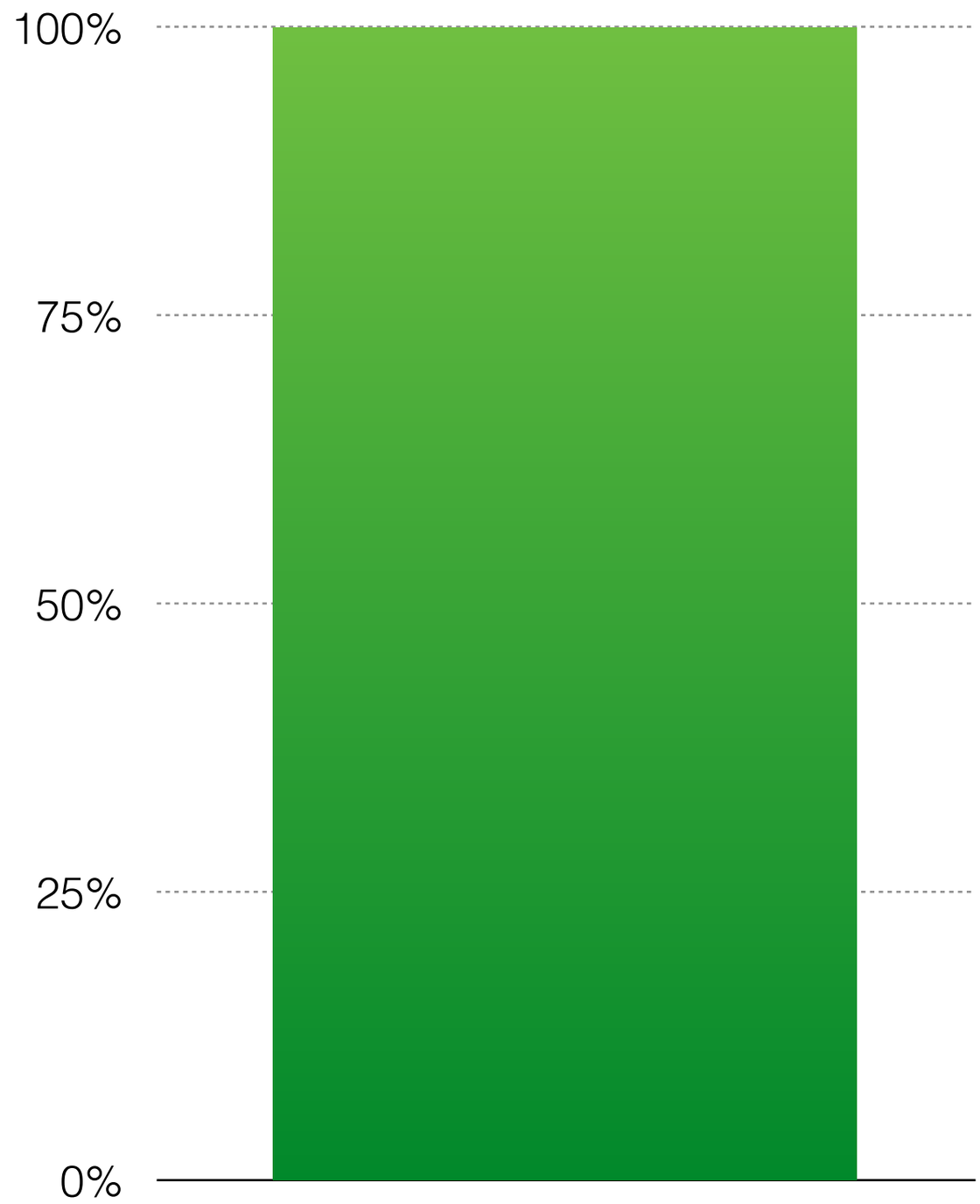


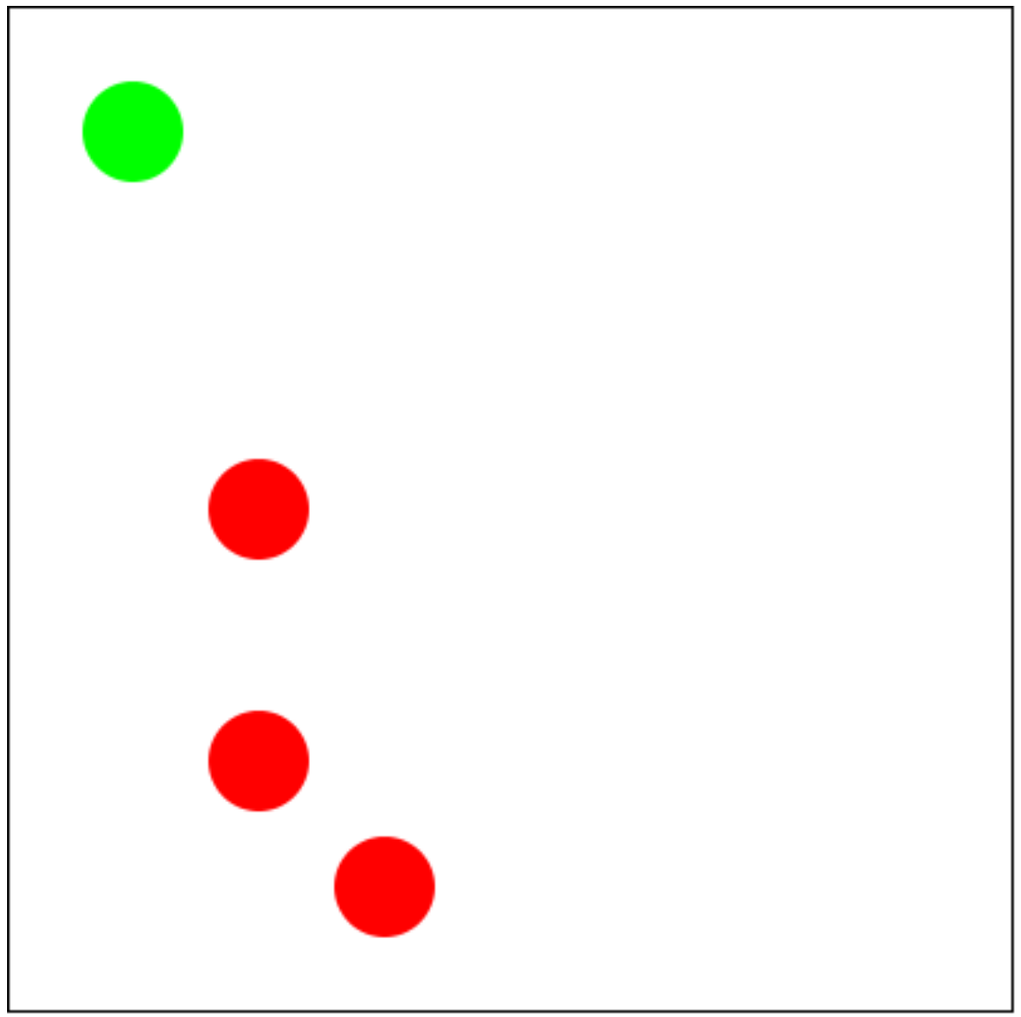
Code **Contracts**

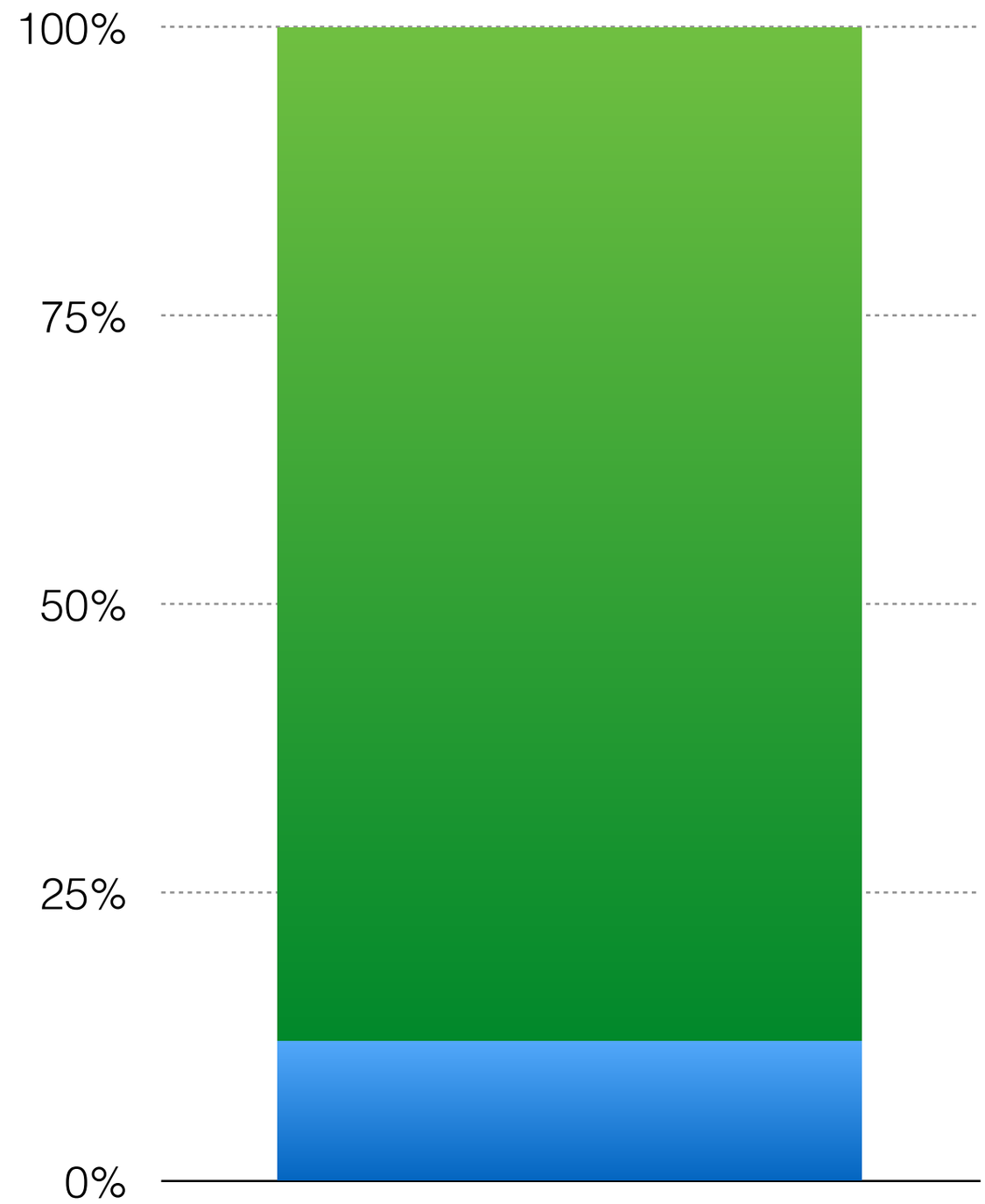

Code
Contracts


100%
75%
50%
25%
0%

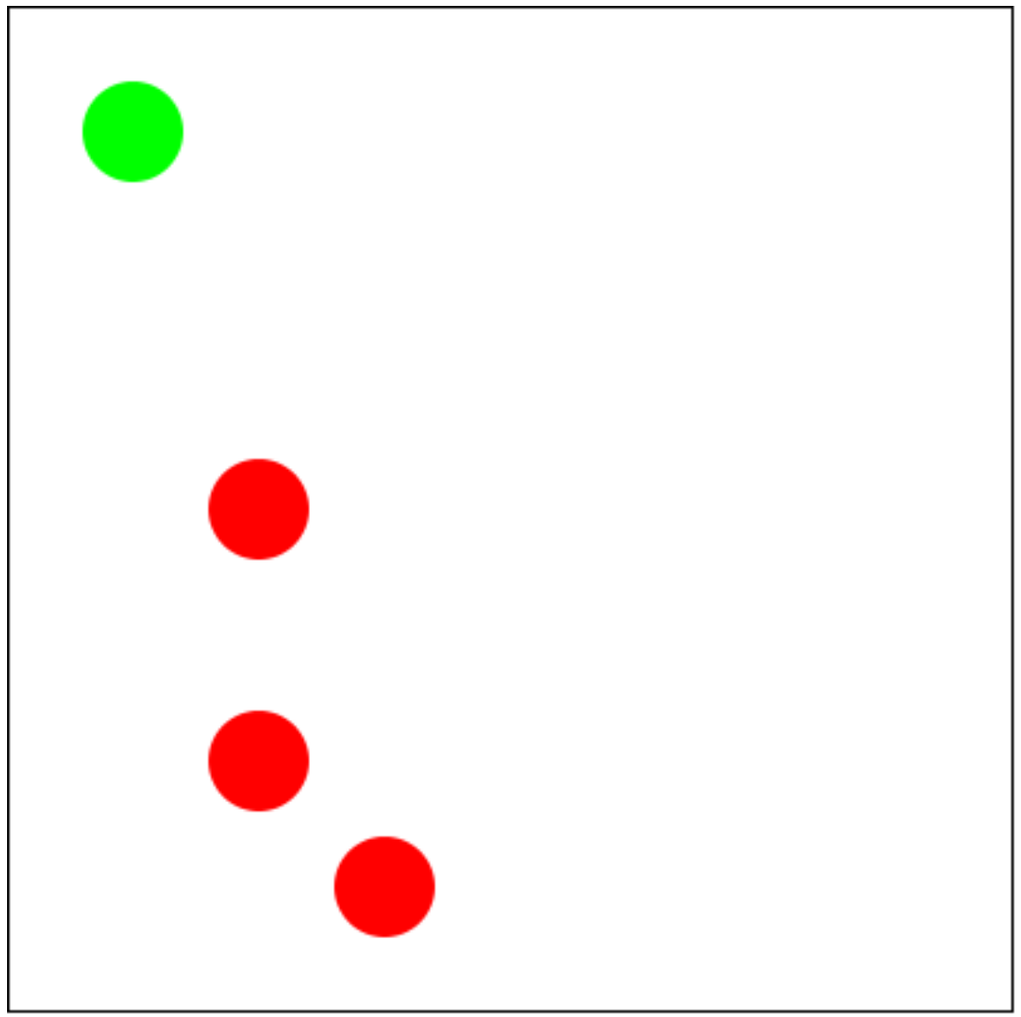

Code
Contracts

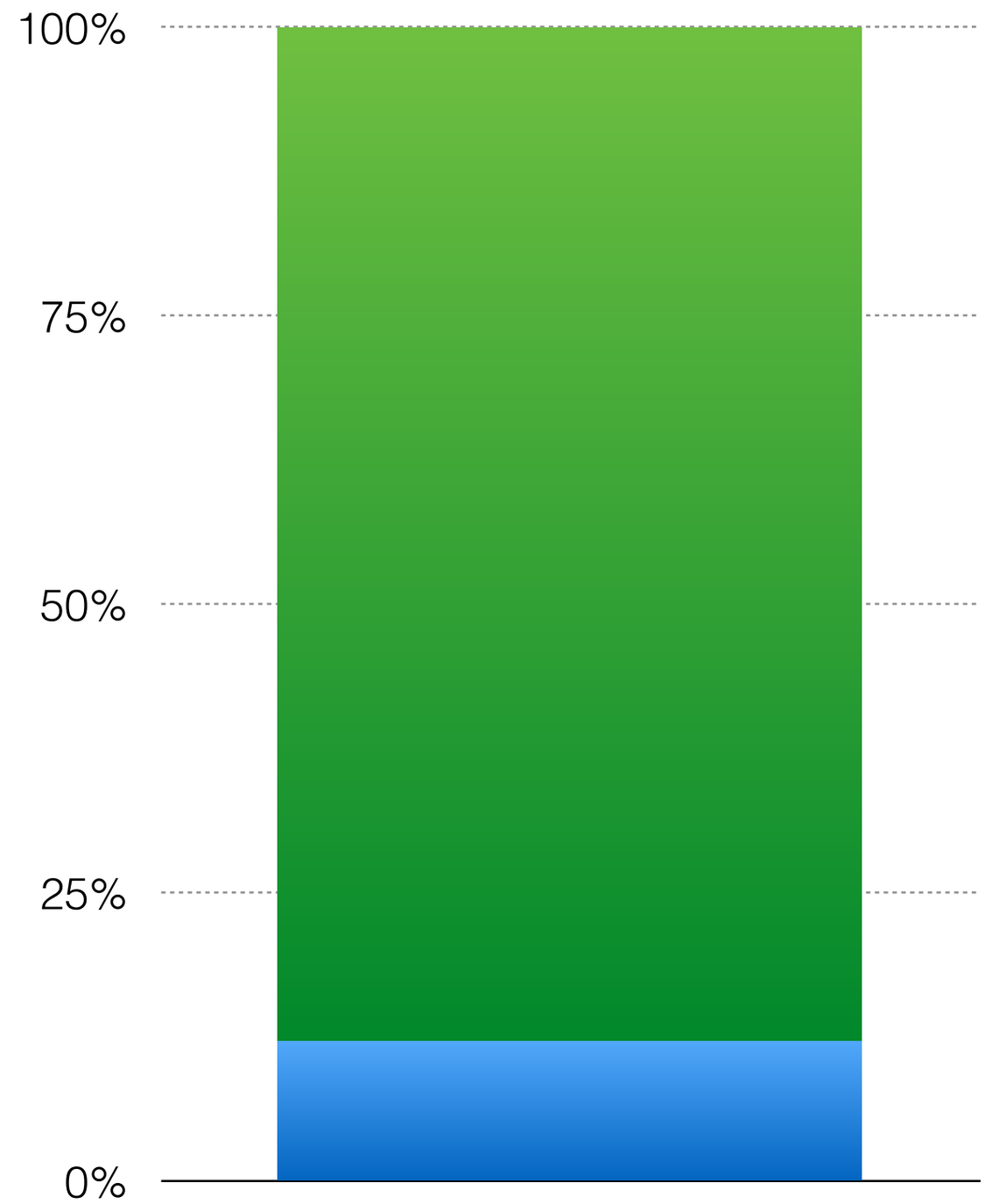

100%
75%
50%
25%
0%



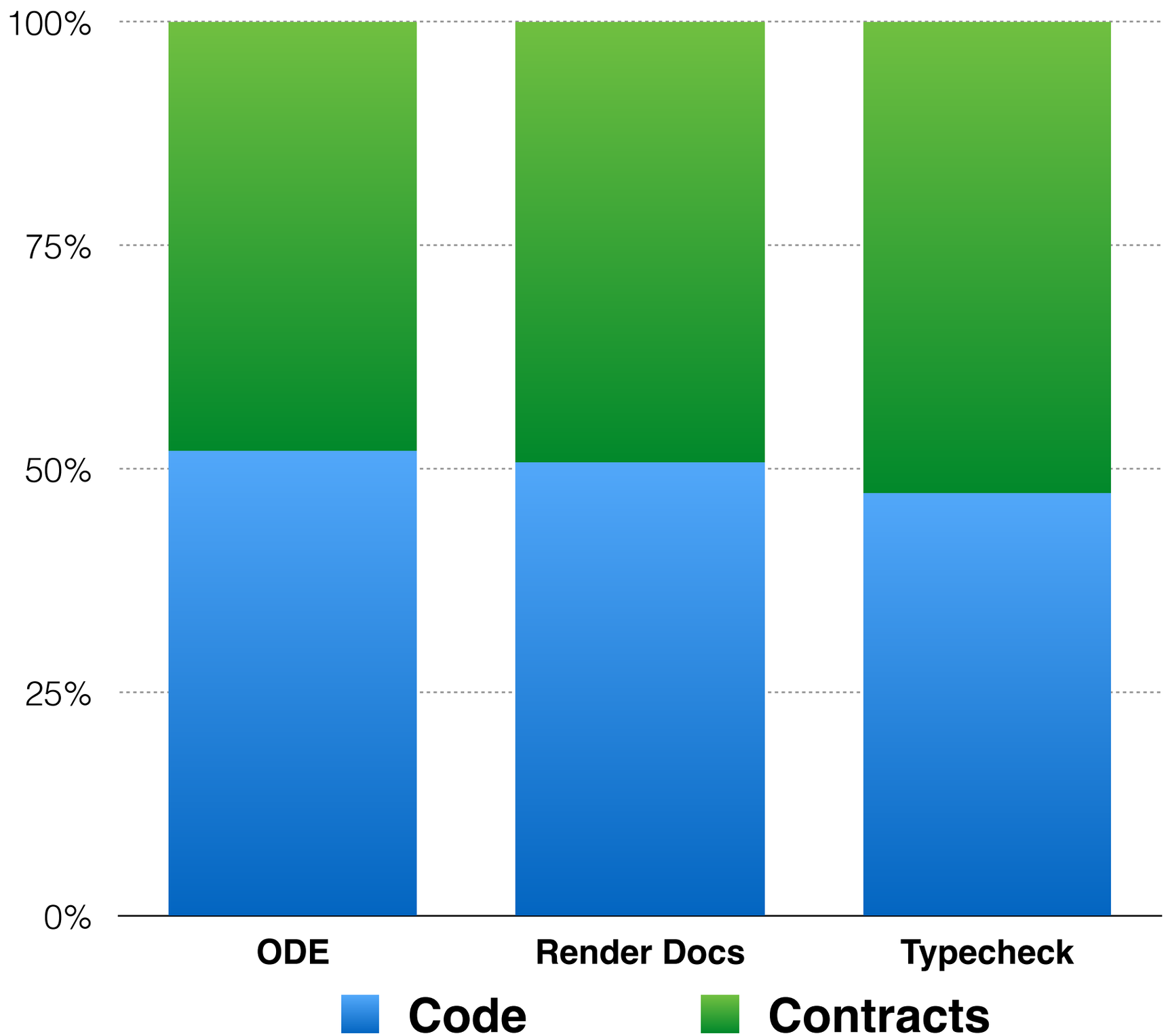


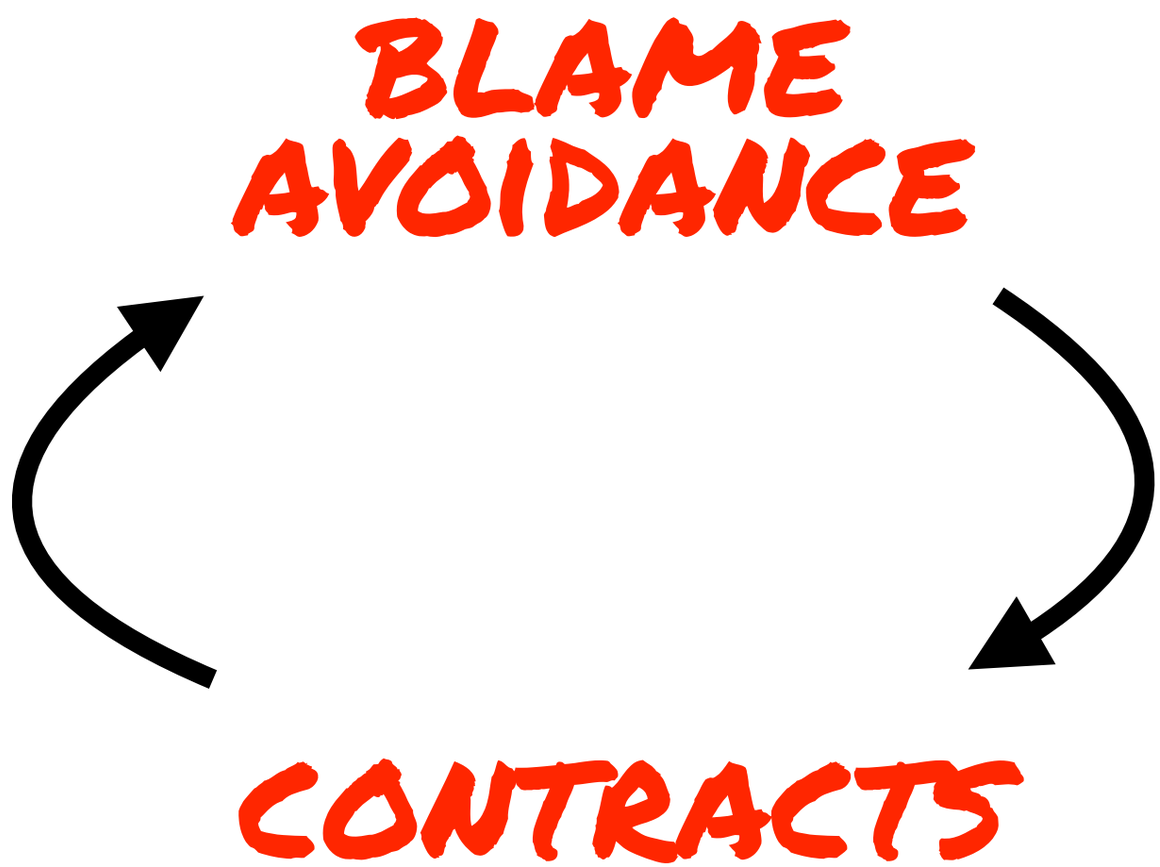
Code **Contracts**

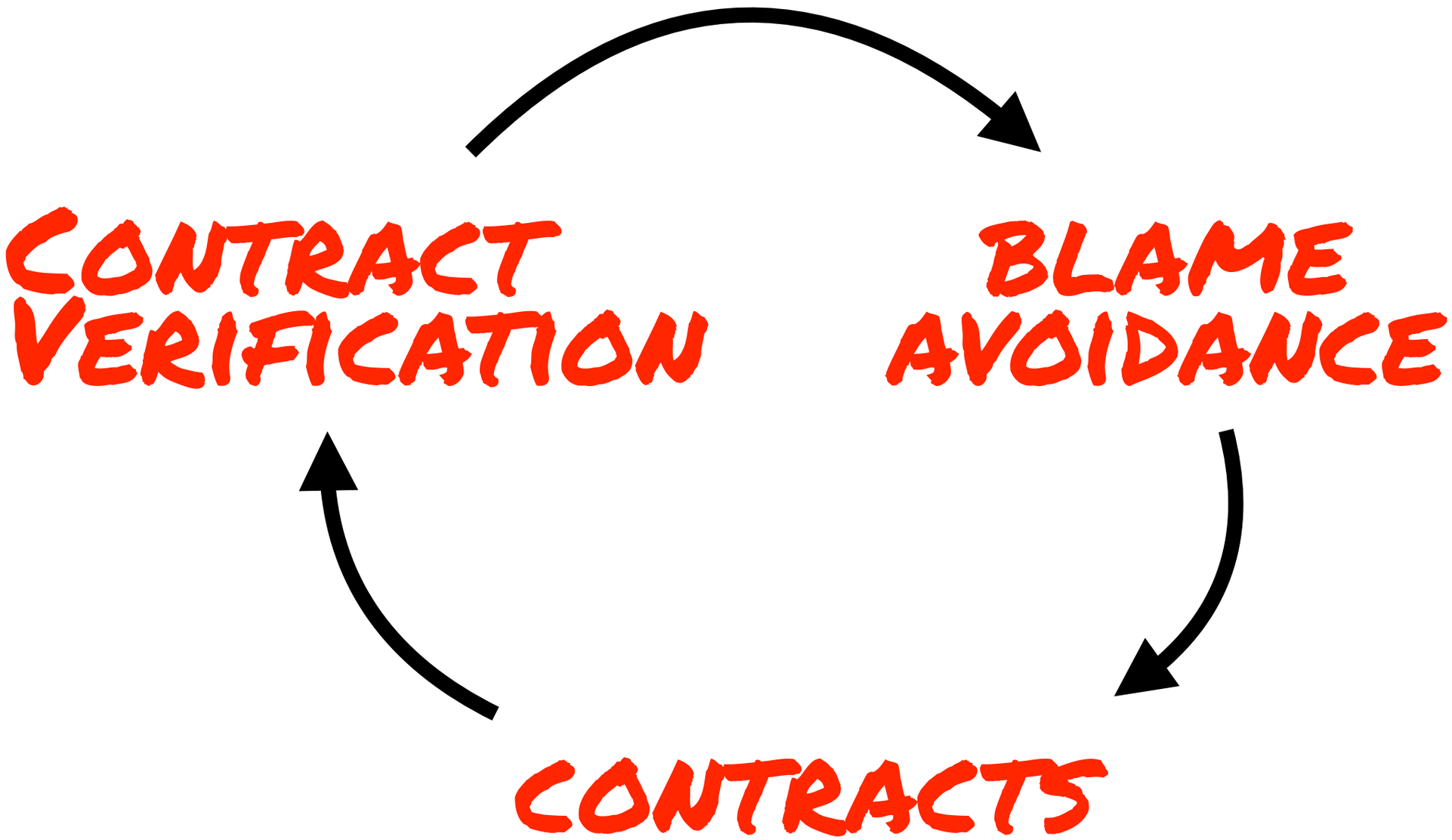


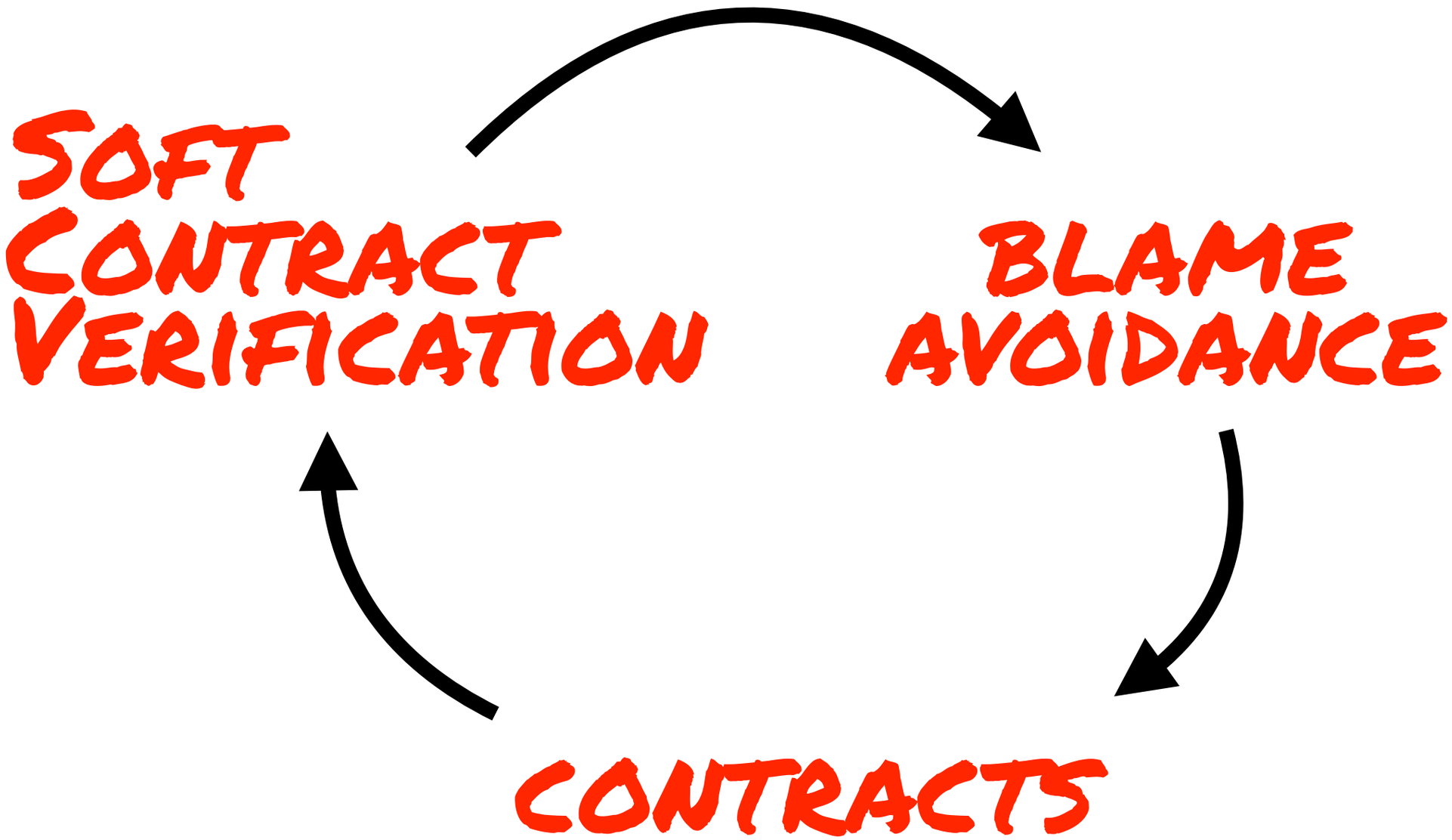


Code **Contracts**


100%
75%
50%
25%
0%
ODE
Render Docs
Typecheck
Code
Contracts


BLAME
AVOIDANCE
CONTRACTS


Contract Verification
Blame Avoidance
Contracts


Soft Contract Verification
Blame Avoidance
Contracts

Soft Contract Verification

Phuc C. Nguyen
Sam Tobin-Hochstadt
David Van Horn

SOFT CONTRACT VERIFICATION:

- FIRST-CLASS, DEPENDENT, RECURSIVE, DISJ., + CONJ. CONTRACTS
- LIFTS FIRST-ORDER SMT SOLVER WITHOUT ENCODINGS
- COMPETITIVE W/ BOTH LIGHT + HEAVY WEIGHT VERIFICATION TECHNIQUES
- WORKS EVEN WITHOUT APPARENT CONTRACTS

CPCF

PCF

```
;; Terms
M ::= X N / + - * =
      (λ ([X : T] ...) M)
      (M M ...)
      (if0 M M M)
```

CPCF

```
;; Terms
M ::= X N / + - * =
      (λ ([X : T] ...) M)
      (M M ...)
      (if0 M M M)
      (C ⚖ M)

;; Contracts
C ::= M
      (C ... -> C)
```

CPCF

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

if0-f

```
(/ 10 3)
```

δ

```
3
```

CPCF

```
(/ 10 (if0 7 2 0))
```

if0-t

```
(/ 10 0)
```

δ

```
(err "Divide by zero")
```

CPCF

```
((λ ((x : nat)) (+ x (/ 10 x))) 3)
```

β

```
(+ 3 (/ 10 3))
```

δ

```
(+ 3 3)
```

δ

```
6
```

CPCF

```
((λ ((x : nat)) (+ x (/ 10 x))) 3)
```

$\downarrow \beta$

```
(+ 3 (/ 10 3))
```

$\downarrow \delta$

```
(+ 3 3)
```

$\downarrow \delta$

```
6
```

CPCF

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

$\downarrow \beta$

```
((λ ((x : nat)) (+ x (/ 10 x))) 3)
```

$\downarrow \beta$

```
(+ 3 (/ 10 3))
```

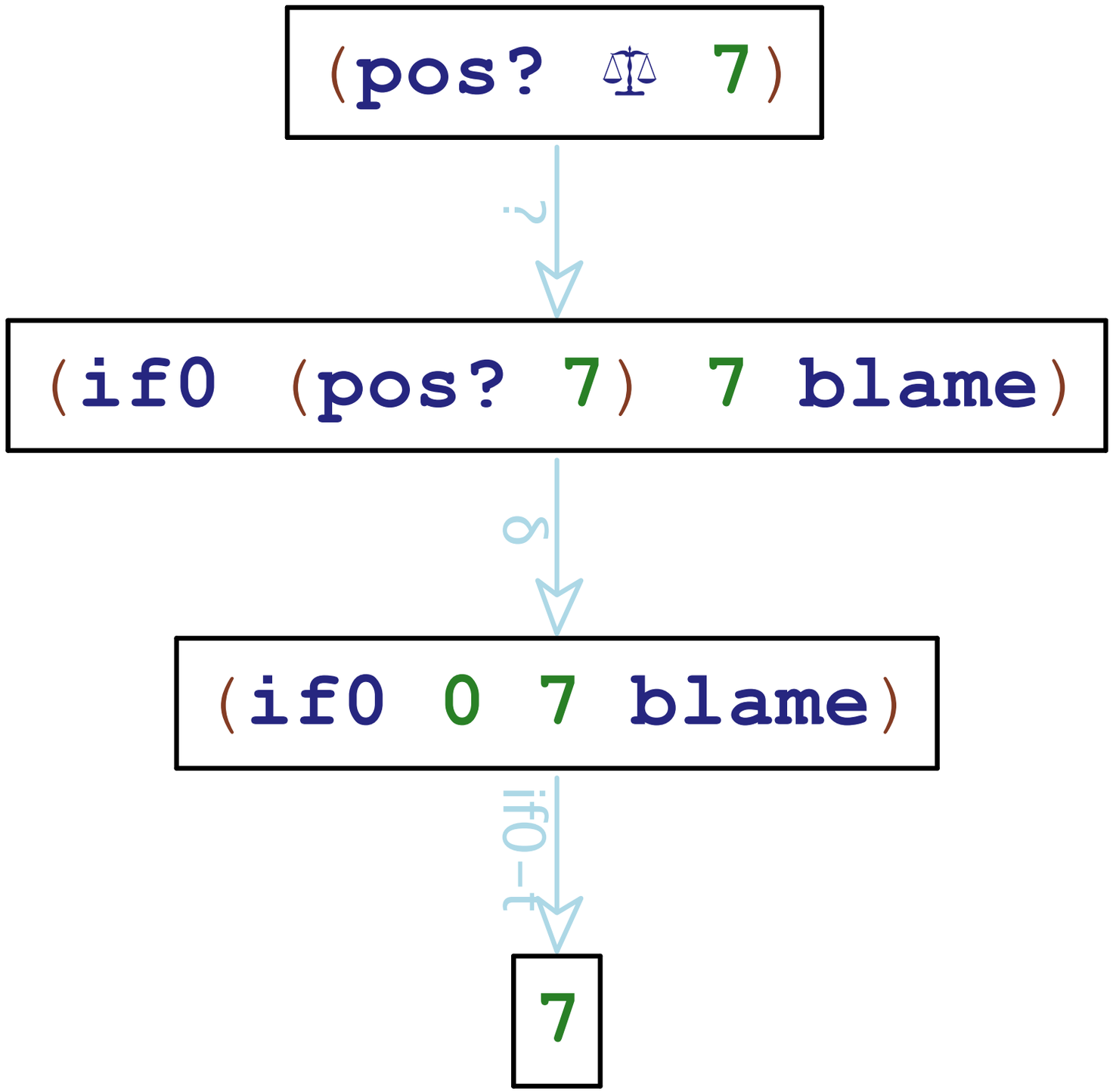
$\downarrow \delta$

```
(+ 3 3)
```

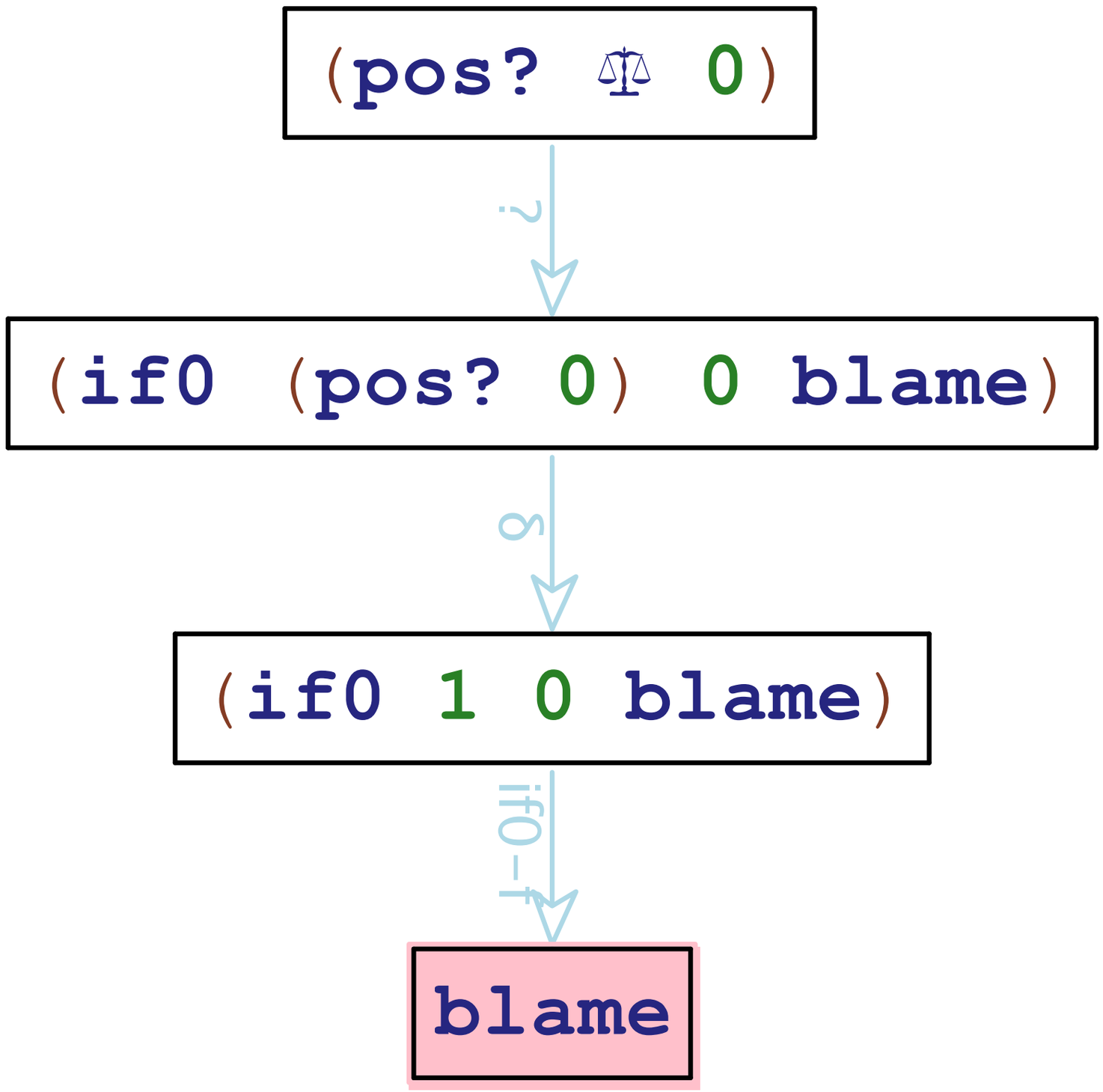
$\downarrow \delta$

```
6
```

CPCF



CPCF



CPCF

```
((zero? -> pos?) ⚖ (λ ((x : nat)) (+ 1 x)))
```

$\sqsupset$

```
(λ ((x : nat))
  (pos? ⚖ ((λ ((x : nat)) (+ 1 x))
           (zero? ⚖ x))))
```

CPCF

```
(((zero? -> pos?) ⚖ (λ ((x : nat)) (+ 1 x)))
 0)
```

$\sqsupseteq$

```
((λ ((x : nat))
   (pos? ⚖ ((λ ((x : nat)) (+ 1 x))
            (zero? ⚖ x))))
 0)
```

β

```
(pos? ⚖ ((λ ((x : nat)) (+ 1 x))
         (zero? ⚖ 0)))
```

ι

```
(pos? ⚖ ((λ ((x : nat)) (+ 1 x))
         (if0 (zero? 0) 0 blame)))
```

δ

CPCF

```
(pos? ⚖ ((λ ((x : nat)) (+ 1 x)) 0))
```

if0-t

```
(pos? ⚖ (+ 1 0))
```

β

```
(pos? ⚖ 1)
```

δ

```
(if0 (pos? 1) 1 blame)
```

δ

```
(if0 0 1 blame)
```

if0-f

```
1
```

CPCF

```
(((zero? -> pos?) ⚖ (λ ((x : nat)) (+ 1 x)))
 7)
```

$\downarrow$ ⊃

```
((λ ((x : nat))
   (pos? ⚖ ((λ ((x : nat)) (+ 1 x))
            (zero? ⚖ x))))
 7)
```

$\downarrow$ β

```
(pos? ⚖ ((λ ((x : nat)) (+ 1 x))
         (zero? ⚖ 7)))
```

$\downarrow$?

```
(pos? ⚖ ((λ ((x : nat)) (+ 1 x))
         (if0 (zero? 7) 7 blame)))
```

$\downarrow$ δ

CPCF

```
(pos? ⚖ ((λ ((x : nat)) (+ 1 x))
         (zero? ⚖ 7)))
```

β

```
(pos? ⚖ ((λ ((x : nat)) (+ 1 x))
         (if0 (zero? 7) 7 blame)))
```

ι

```
(pos? ⚖ ((λ ((x : nat)) (+ 1 x))
         (if0 1 7 blame)))
```

δ

```
(pos? ⚖ ((λ ((x : nat)) (+ 1 x))
         (if0 1 7 blame)))
```

if0-i

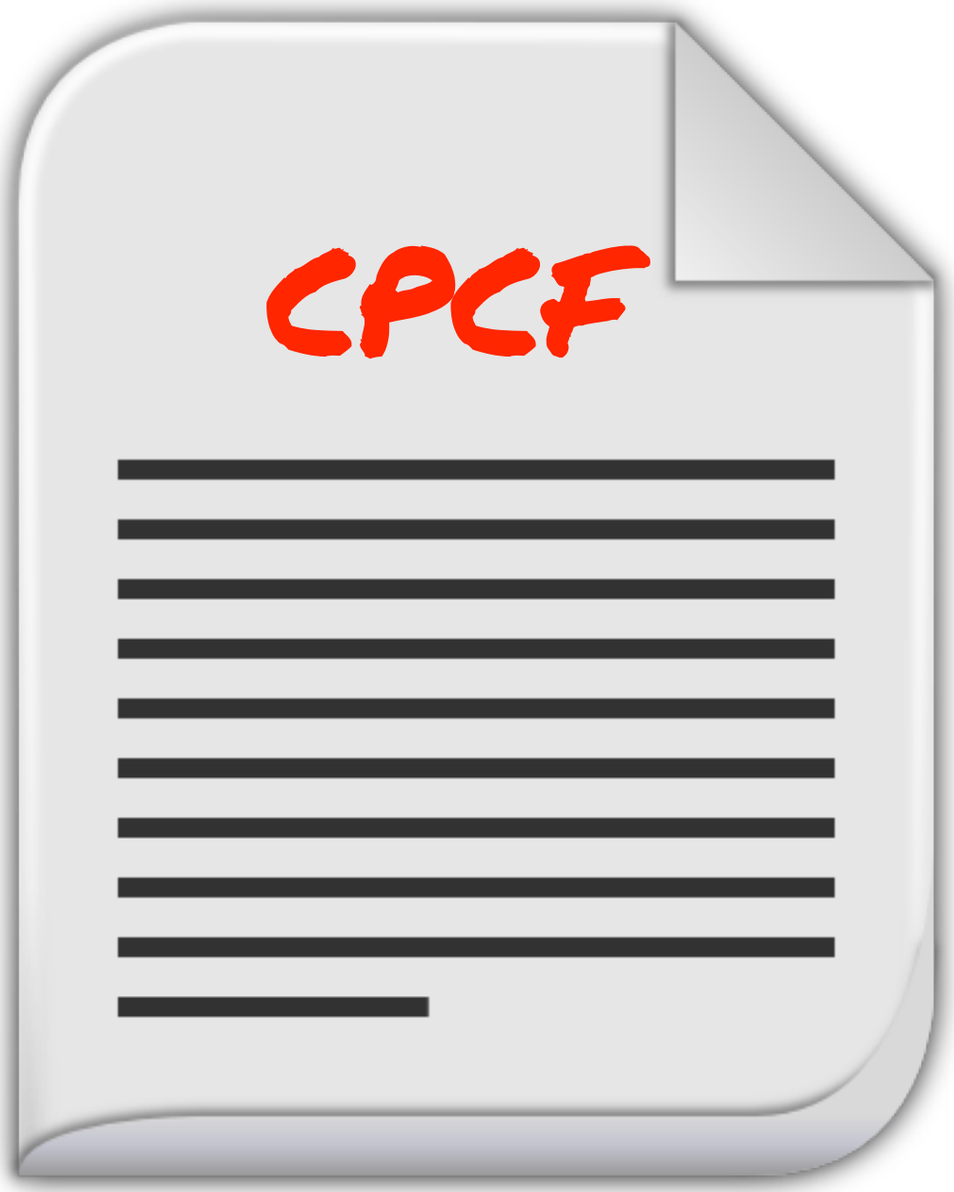
```
(pos? ⚖ ((λ ((x : nat)) (+ 1 x))
         blame))
```

```
blame
```

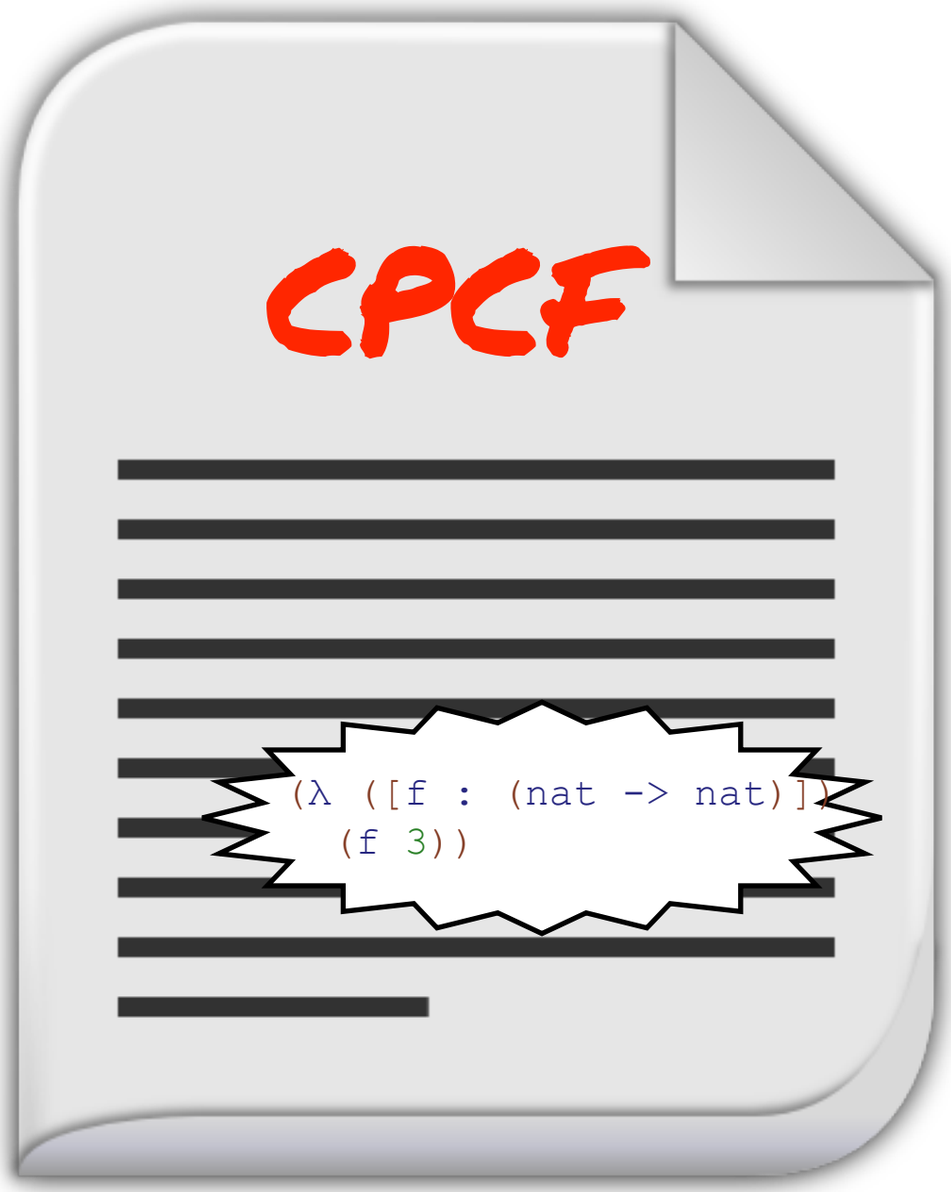
SYMBOLIC CPCF

ごめん

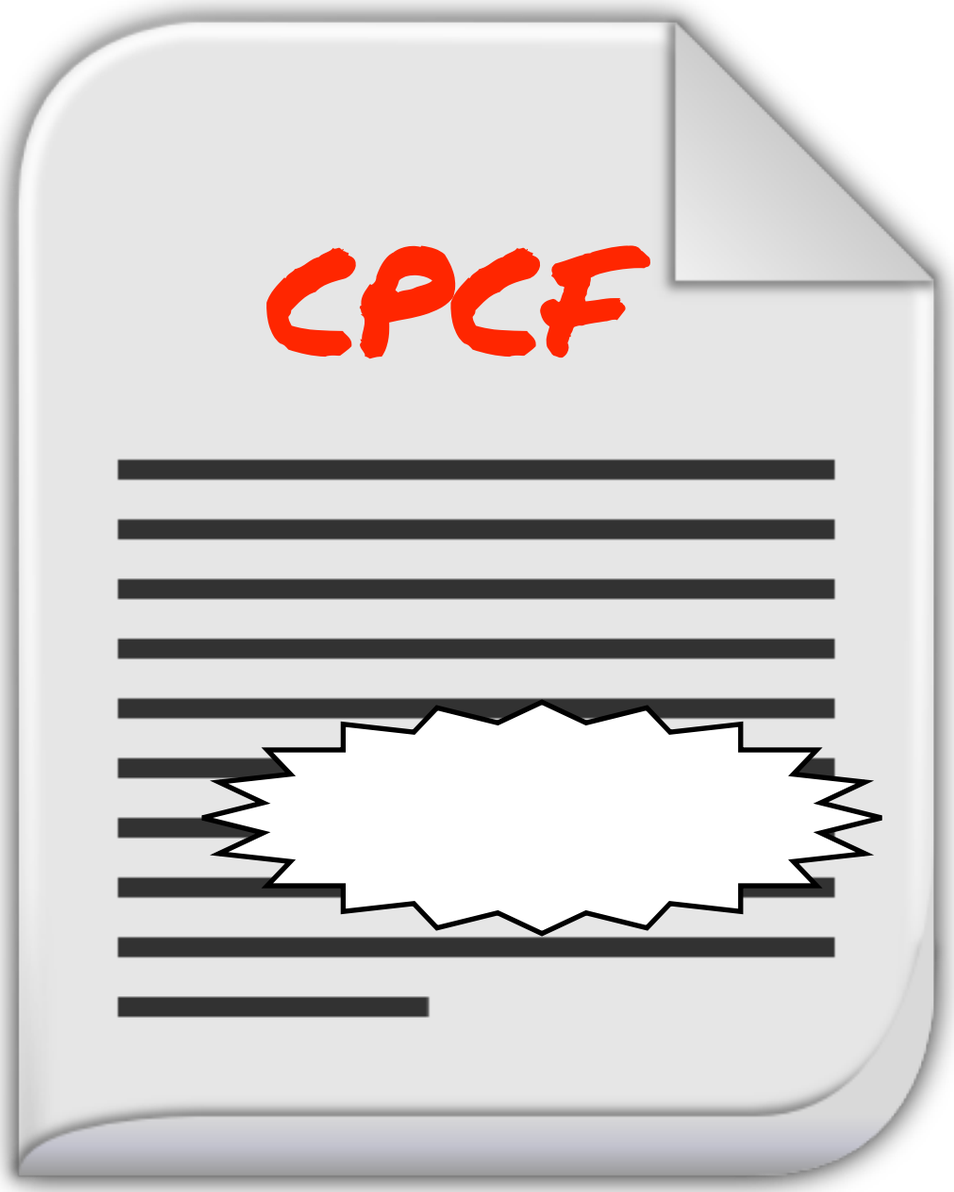
Abstraction



Abstraction



Abstraction

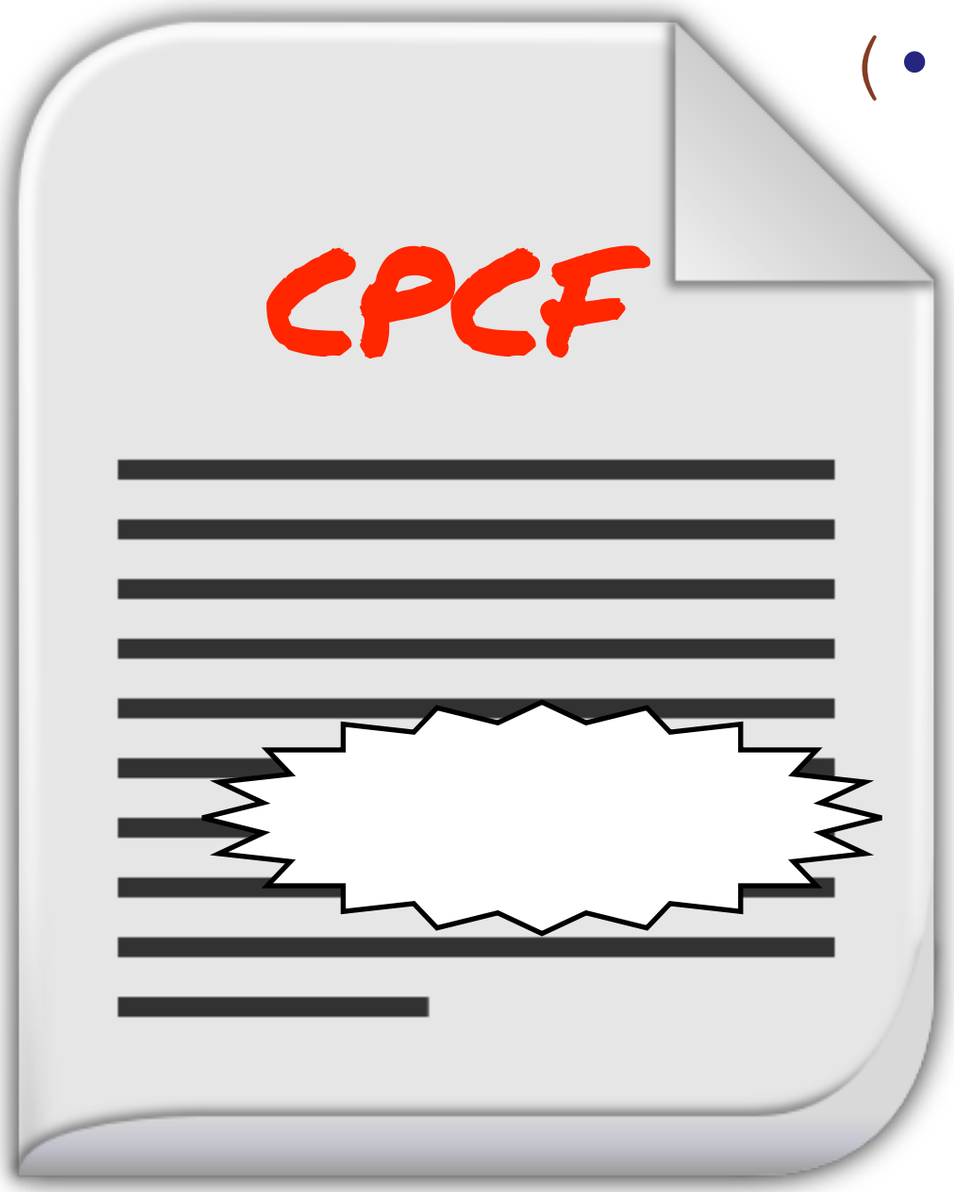



```
(λ ([f : (nat -> nat)])
  (f 3))
```

Abstraction


```
(• ((nat -> nat) -> nat))
```

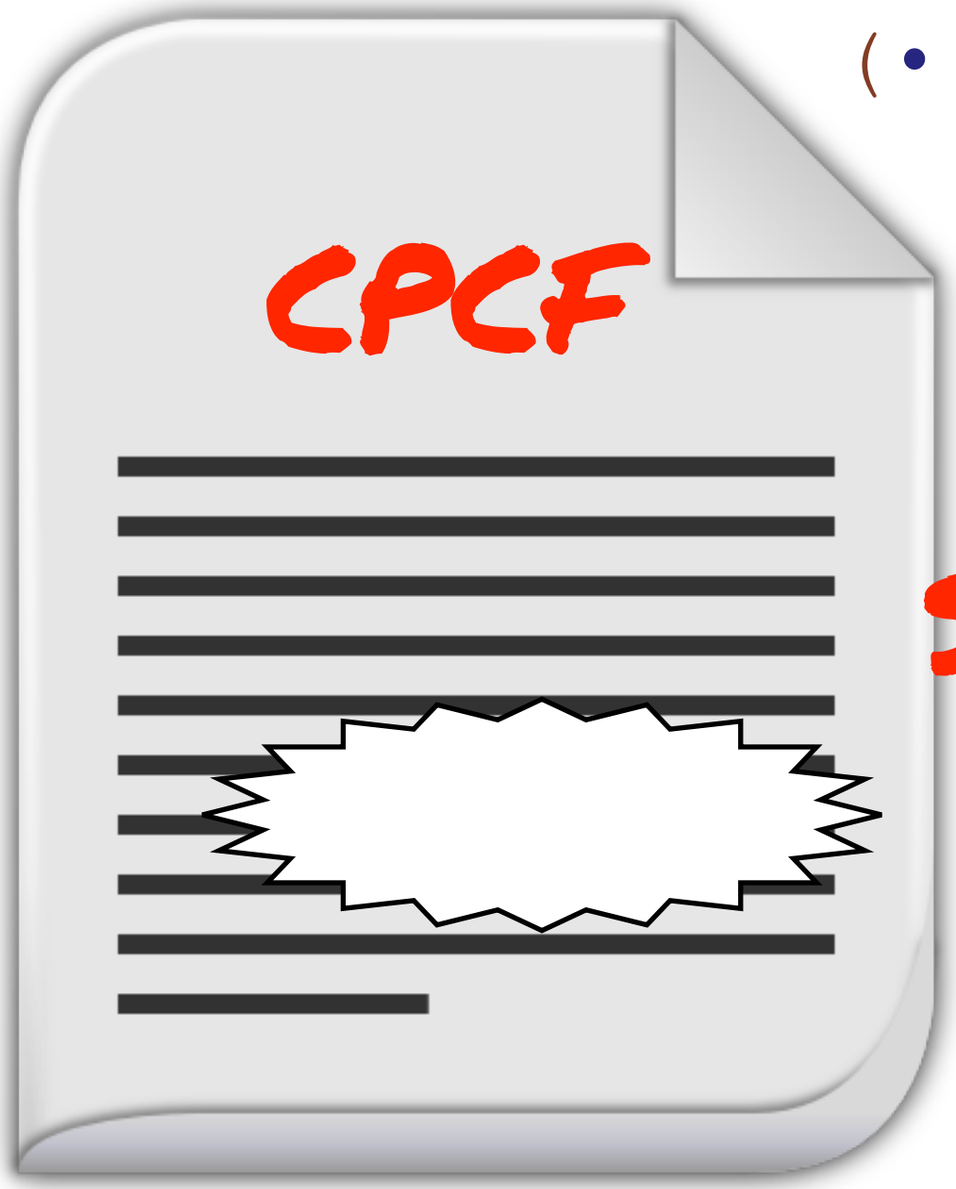
Abstraction




```
(• ((nat -> nat) -> nat)
   ((pos? -> prime?) -> prime?))
```

Abstraction

```
(• ((nat -> nat) -> nat)
   ((pos? -> prime?) -> prime?))
```



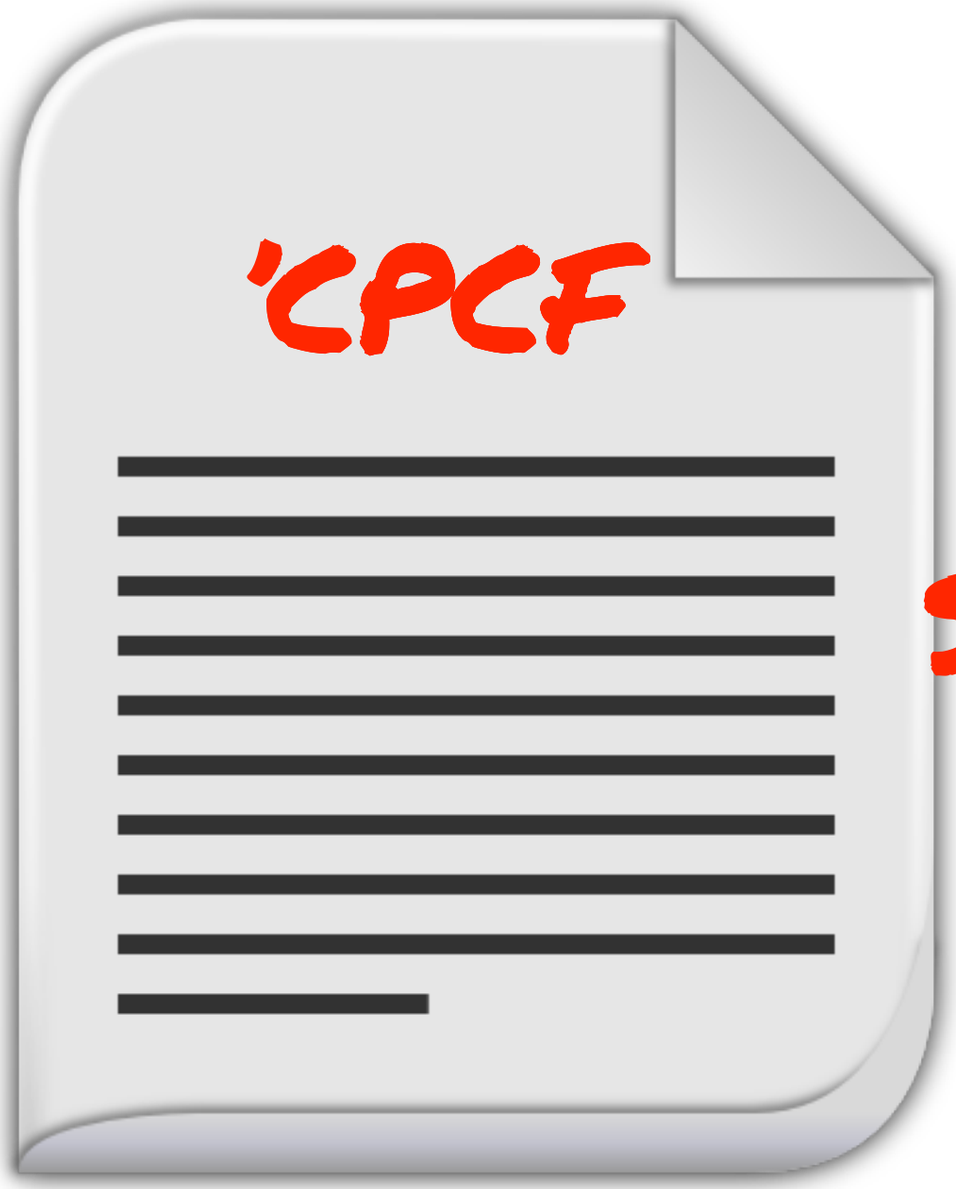
Symbolic values are sets of Contracts

Abstraction

```
(• ((nat -> nat) -> nat)
   ((pos? -> prime?) -> prime?))
```

Symbolic values are sets of Contracts

Abstraction



Symbolic values are sets of Contracts

Soundness:

All concretizations are approximated by 'CPCF

ESSENTIAL IDEAS:

1) CONTRACTS AS SYMBOLIC VALUES
2) CHECKS REFINE VALUES
3) REFINEMENTS INFLUENCE COMPUTATION
4) PARTITION VERIFICATION W/ BLAME
5) HEAP OF INVARIANTS -> SMT

'CPCF

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

'CPCF

```
(/ (• nat) (if0 7 2 3))
```

'CPCF

```
(/ (• nat) (if0 7 2 3))
```

$\xrightarrow{\text{if0-f}}$

```
(/ (• nat) 3)
```

$\xrightarrow{\text{>0}}$

```
(• nat)
```

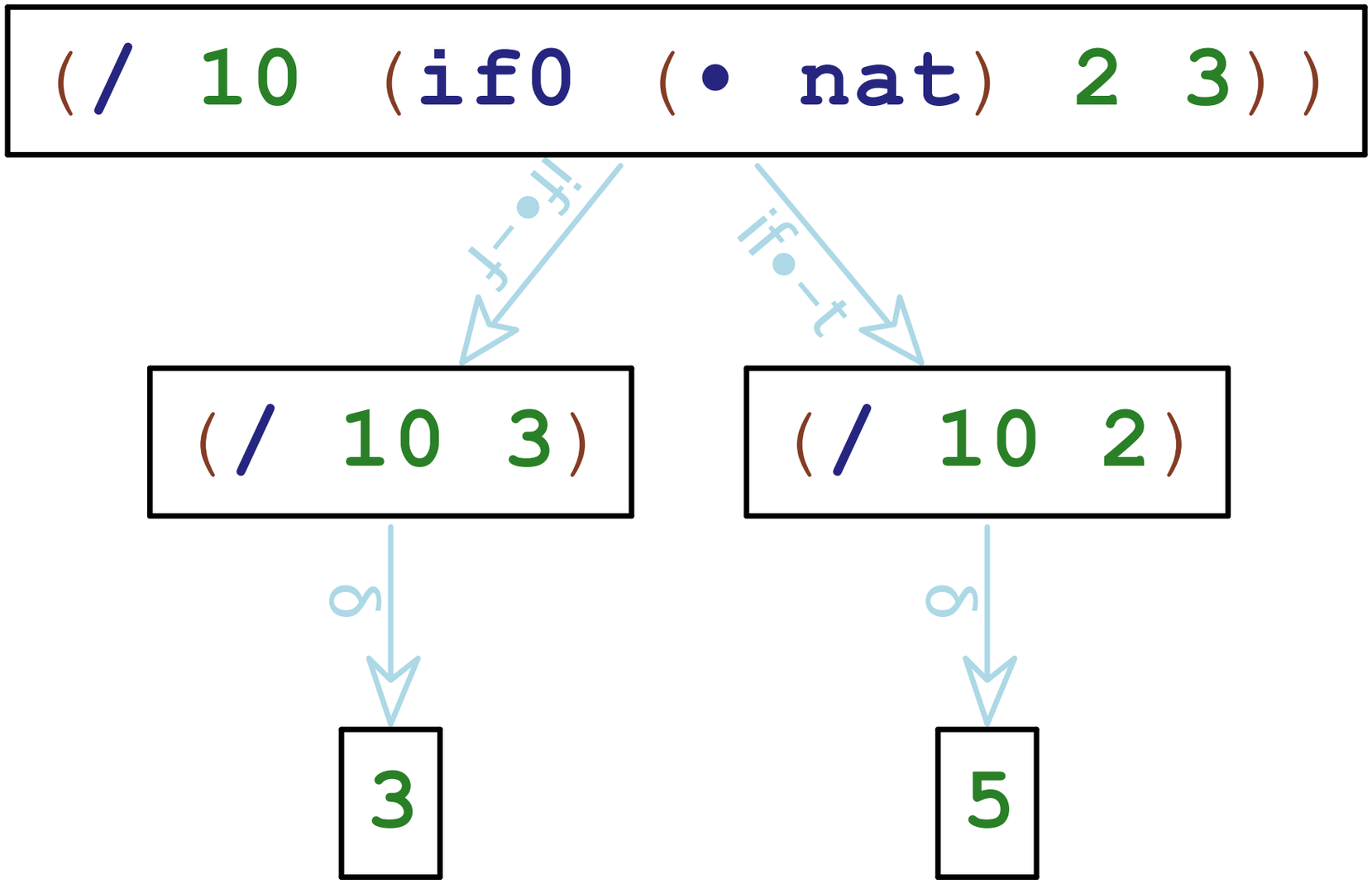
'CPCF

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

'CPCF

```
(/ 10 (if0 (• nat) 2 3))
```

'CPCF



'CPCF

```
(add1 7)
```

'CPCF

```
((• (nat -> nat)) 7)
```

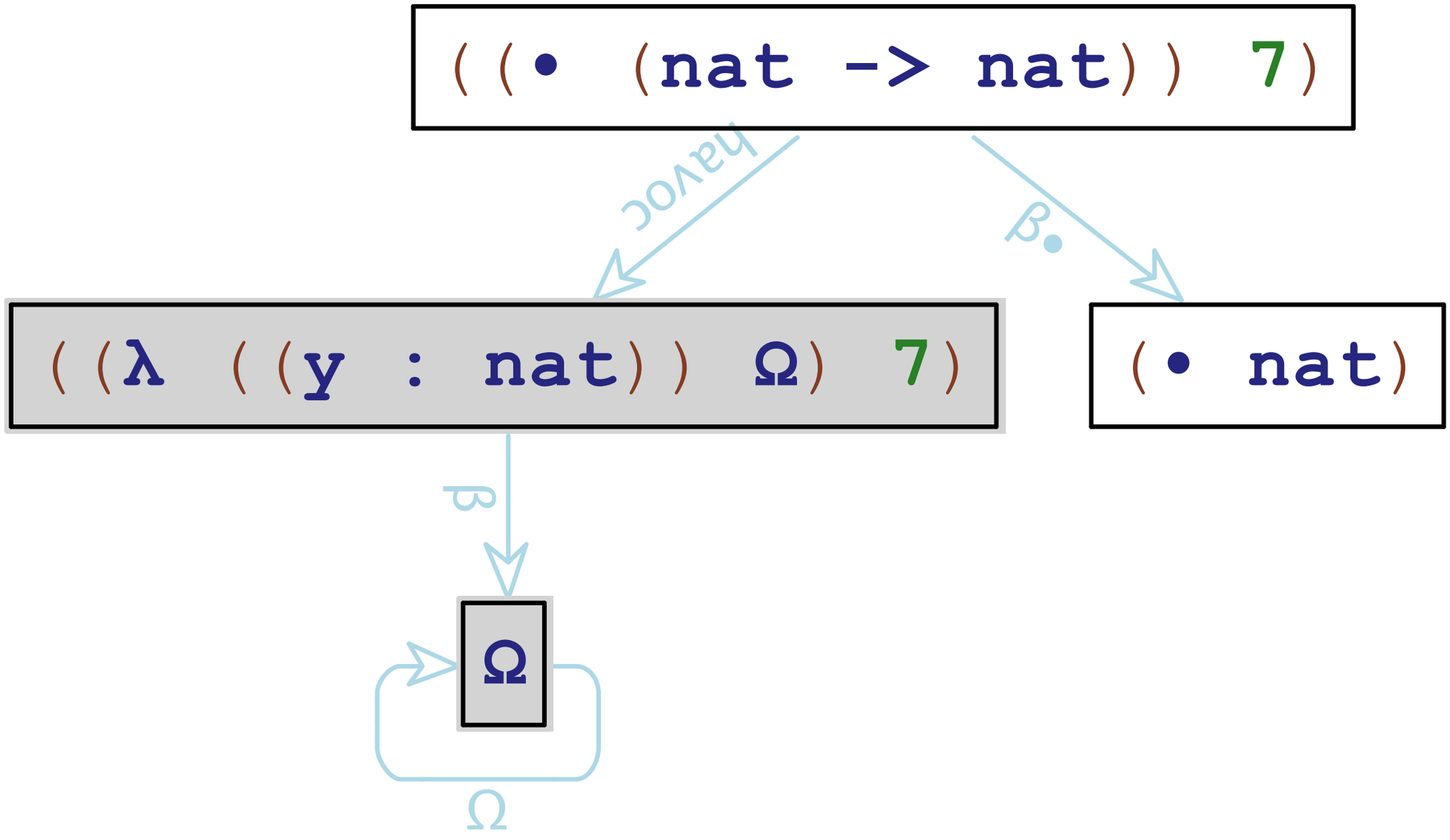
'CPCF

```
((• (nat -> nat)) 7)
```

$\beta_\bullet$

```
(• nat)
```

'CPCF



'CPCF

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

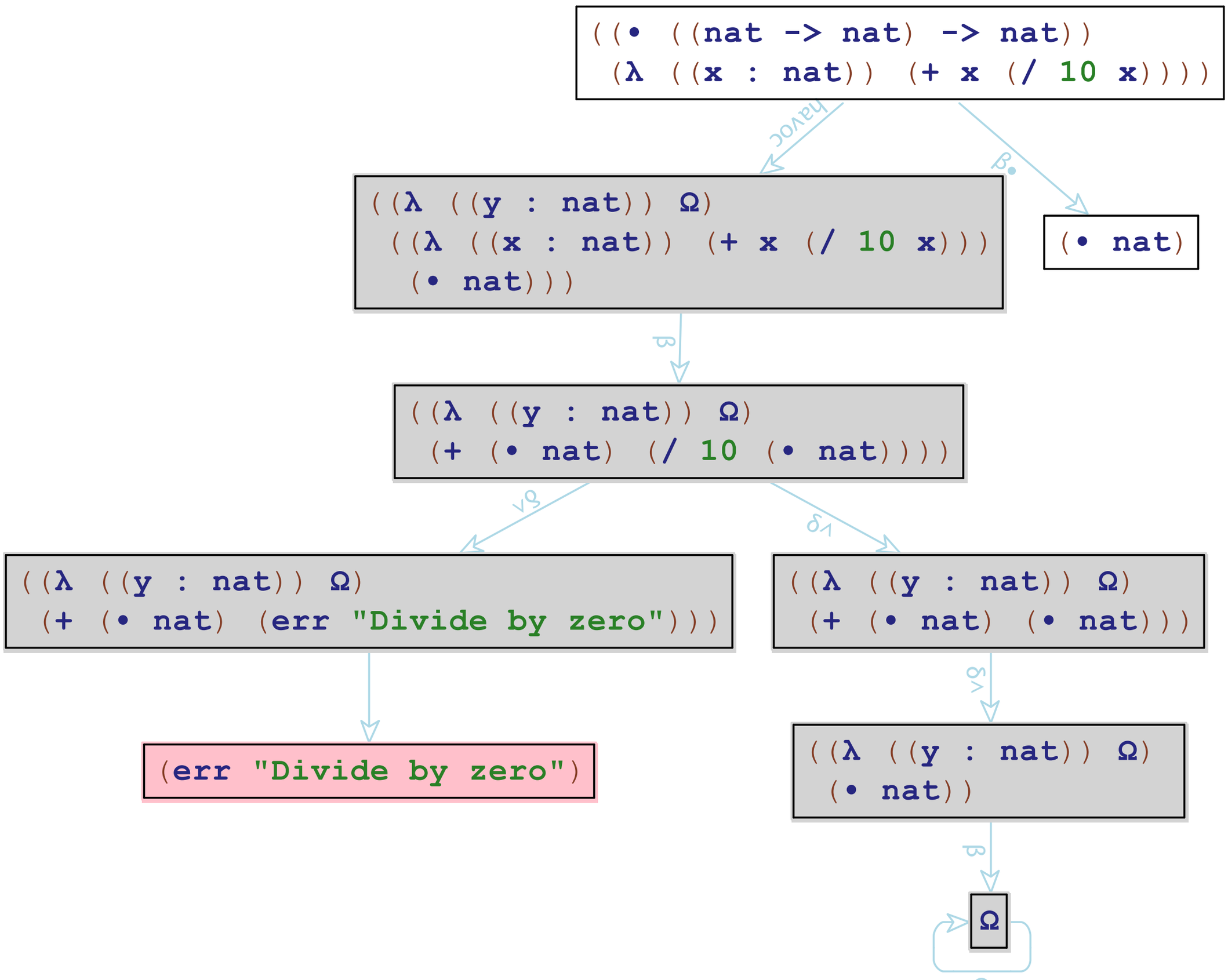
'CPCF

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

'CPCF

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

'CPCF


((• ((nat -> nat) -> nat))
(λ ((x : nat)) (+ x (/ 10 x))))
havoc
β•
((λ ((y : nat)) Ω)
((λ ((x : nat)) (+ x (/ 10 x)))
(• nat)))
(• nat)
β
((λ ((y : nat)) Ω)
(+ (• nat) (/ 10 (• nat))))
δ0
δ1
((λ ((y : nat)) Ω)
(+ (• nat) (err "Divide by zero")))
((λ ((y : nat)) Ω)
(+ (• nat) (• nat)))
(err "Divide by zero")
δ0
((λ ((y : nat)) Ω)
(• nat))
β
Ω

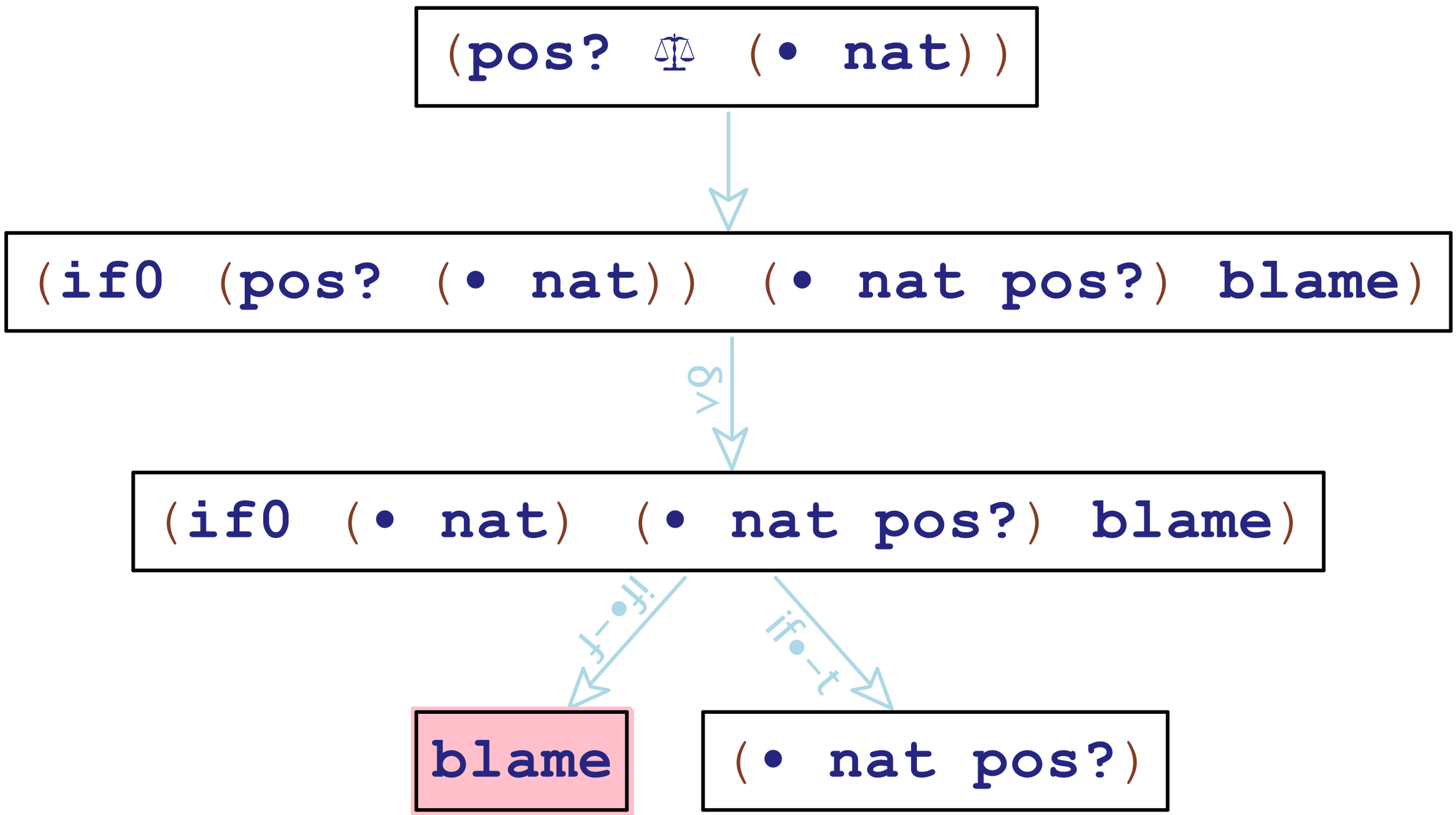
'CPCF

```
(pos? ⚖ 7)
```

'CPCF

```
(pos? ⚖ (• nat))
```

'CPCF


(pos? ⚖ (• nat))
(if0 (pos? (• nat)) (• nat pos?) blame)
β
(if0 (• nat) (• nat pos?) blame)
if•-f
if•-t
blame
(• nat pos?)

ESSENTIAL IDEAS:

1) CONTRACTS AS SYMBOLIC VALUES
2) CHECKS REFINE VALUES
3) REFINEMENTS INFLUENCE COMPUTATION
4) PARTITION VERIFICATION W/ BLAME
5) HEAP OF INVARIANTS -> SMT

'CPCF

```
(pos? ⚖ 7)
```

'CPCF

```
(pos? ⚖ (• nat pos?))
```

'CPCF

```
(pos? ⚖ (• nat pos?))
```

↓

```
(• nat pos?)
```

'CPCF

```
(pos? ⚖ (• nat pos?))
```

↓

```
(• nat pos?)
```

```
(/ 10 (• nat pos?))
```

↓ >0

```
(• nat)
```

ESSENTIAL IDEAS:

1) CONTRACTS AS SYMBOLIC VALUES
2) CHECKS REFINE VALUES
3) REFINEMENTS INFLUENCE COMPUTATION
4) PARTITION VERIFICATION W/ BLAME
5) HEAP OF INVARIANTS -> SMT

'CPCF

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

'CPCF

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

β•

```
(• nat pos?)
```

'CPCF

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

havoc

```
((λ ((y : nat)) Ω)
 ((λ ((x : nat)) (+ x (/ 10 x)))
  (• nat pos?)))
```

β•

```
(• nat pos?)
```

β

```
((λ ((y : nat)) Ω)
 (+ (• nat pos?) (/ 10 (• nat pos?))))
```

vδ

```
((λ ((y : nat)) Ω)
 (+ (• nat pos?) (• nat)))
```

vδ

```
((λ ((y : nat)) Ω)
 (• nat))
```

β

```
Ω
```

'CPCF

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

'CPCF

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

⊐

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

havoc

```
at)) Ω)
nat)) (pos? ⚖ ((λ ((x : nat)) (+ x (/ 10 x)))
                (pos? ⚖ x))))
```

β•

```
(• nat)
```

CPCF

```
((λ ((y : nat)) Ω)
 (pos? ⚖ ((λ ((x : nat)) (+ x (/ 10 x)))
          (if0 (• nat) (• nat pos?) (blame HAVOC))))
```

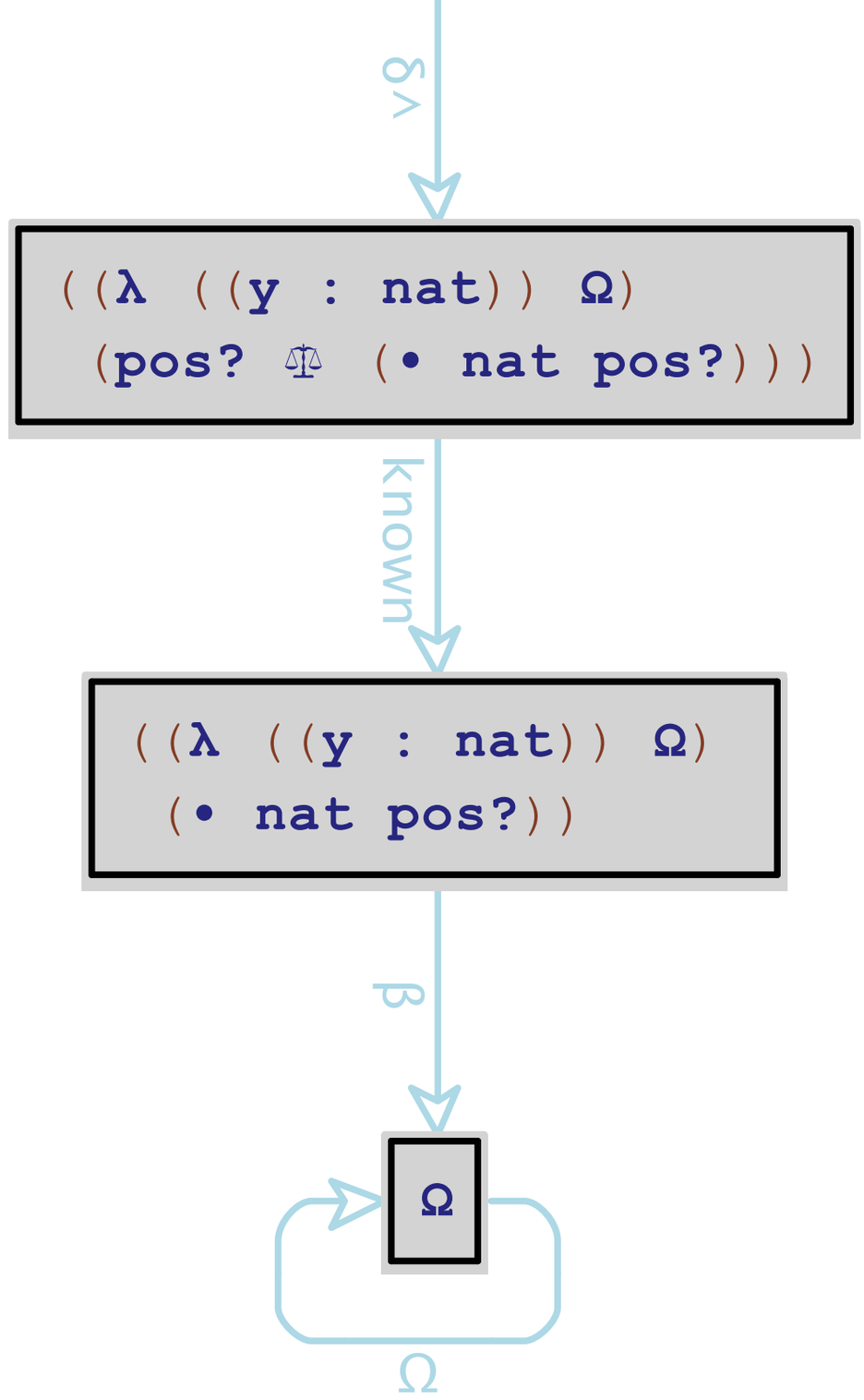
```
((λ ((y : nat)) Ω)
 (pos? ⚖ ((λ ((x : nat)) (+ x (/ 10 x)))
          (blame HAVOC))))
```

```
((λ ((y : nat)) Ω)
 (pos? ⚖ ((λ ((x :
          (• nat pos
```

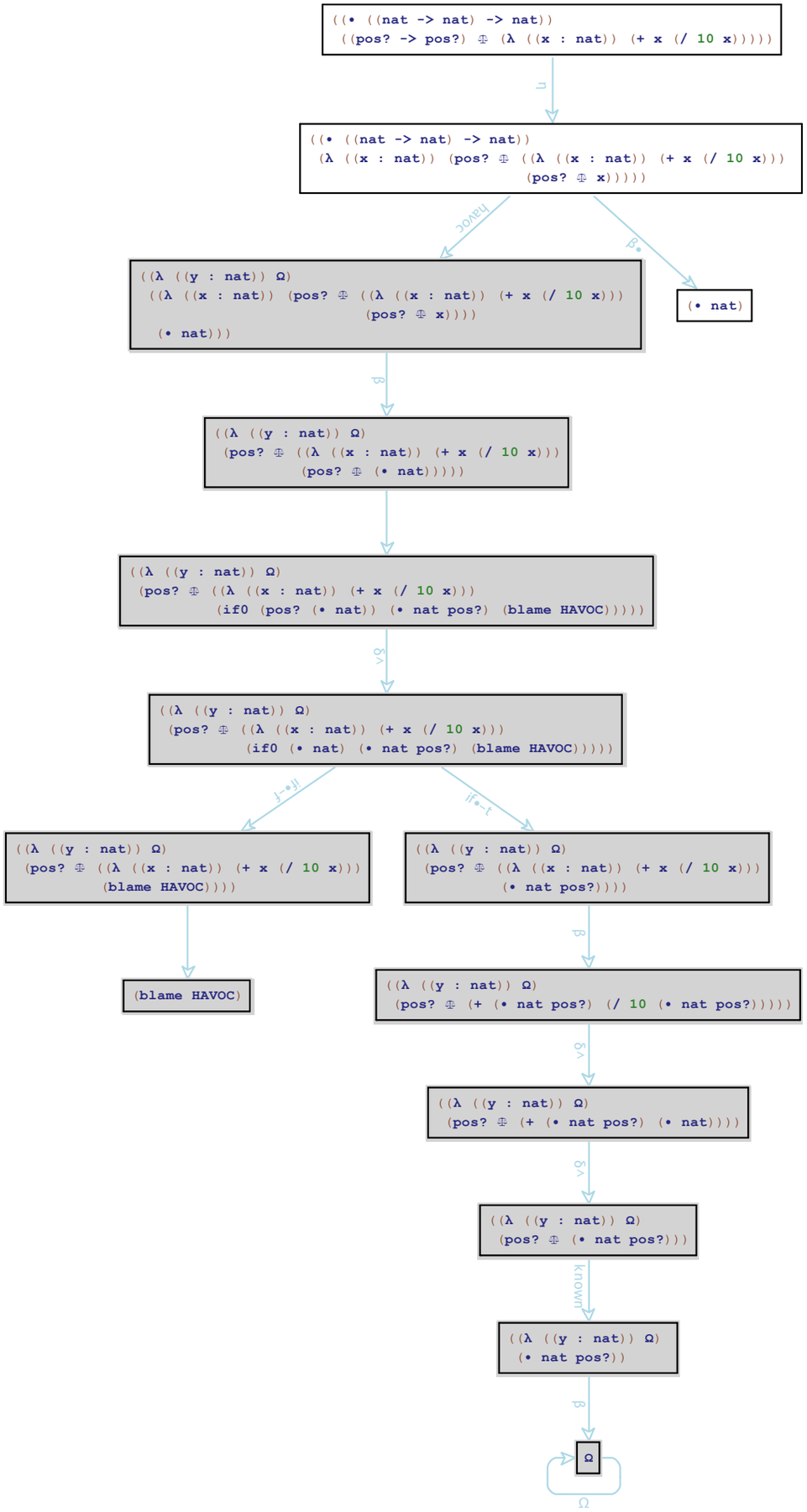
```
(blame HAVOC)
```

```
((λ ((y : nat)) Ω)
 (pos? ⚖ (+ (• nat pos
```

'CPCF



'CPCF


((• ((nat -> nat) -> nat))
((pos? -> pos?) ⊕ (λ ((x : nat)) (+ x (/ 10 x)))))
μ
((• ((nat -> nat) -> nat))
(λ ((x : nat)) (pos? ⊕ ((λ ((x : nat)) (+ x (/ 10 x)))
(pos? ⊕ x)))))
havoc
β•
((λ ((y : nat)) Ω)
((λ ((x : nat)) (pos? ⊕ ((λ ((x : nat)) (+ x (/ 10 x)))
(pos? ⊕ x))))
(• nat)))
(• nat)
β
((λ ((y : nat)) Ω)
(pos? ⊕ ((λ ((x : nat)) (+ x (/ 10 x)))
(pos? ⊕ (• nat)))))
((λ ((y : nat)) Ω)
(pos? ⊕ ((λ ((x : nat)) (+ x (/ 10 x)))
(if0 (pos? (• nat)) (• nat pos?) (blame HAVOC)))))
v.s
((λ ((y : nat)) Ω)
(pos? ⊕ ((λ ((x : nat)) (+ x (/ 10 x)))
(if0 (• nat) (• nat pos?) (blame HAVOC)))))
if-•-f
if-•-t
((λ ((y : nat)) Ω)
(pos? ⊕ ((λ ((x : nat)) (+ x (/ 10 x)))
(blame HAVOC))))
(blame HAVOC)
((λ ((y : nat)) Ω)
(pos? ⊕ ((λ ((x : nat)) (+ x (/ 10 x)))
(• nat pos?))))
β
((λ ((y : nat)) Ω)
(pos? ⊕ (+ (• nat pos?) (/ 10 (• nat pos?)))))
δ-v
((λ ((y : nat)) Ω)
(pos? ⊕ (+ (• nat pos?) (• nat))))
δ-v
((λ ((y : nat)) Ω)
(pos? ⊕ (• nat pos?)))
known
((λ ((y : nat)) Ω)
(• nat pos?))
β
Ω
Ω

ESSENTIAL IDEAS:

1) CONTRACTS AS SYMBOLIC VALUES
2) CHECKS REFINE VALUES
3) REFINEMENTS INFLUENCE COMPUTATION
4) PARTITION VERIFICATION W/ BLAME
5) HEAP OF INVARIANTS -> SMT

'CPCF

```
((λ ((x : nat)) (if0 x 1 (/ 10 x))) (• nat))
```

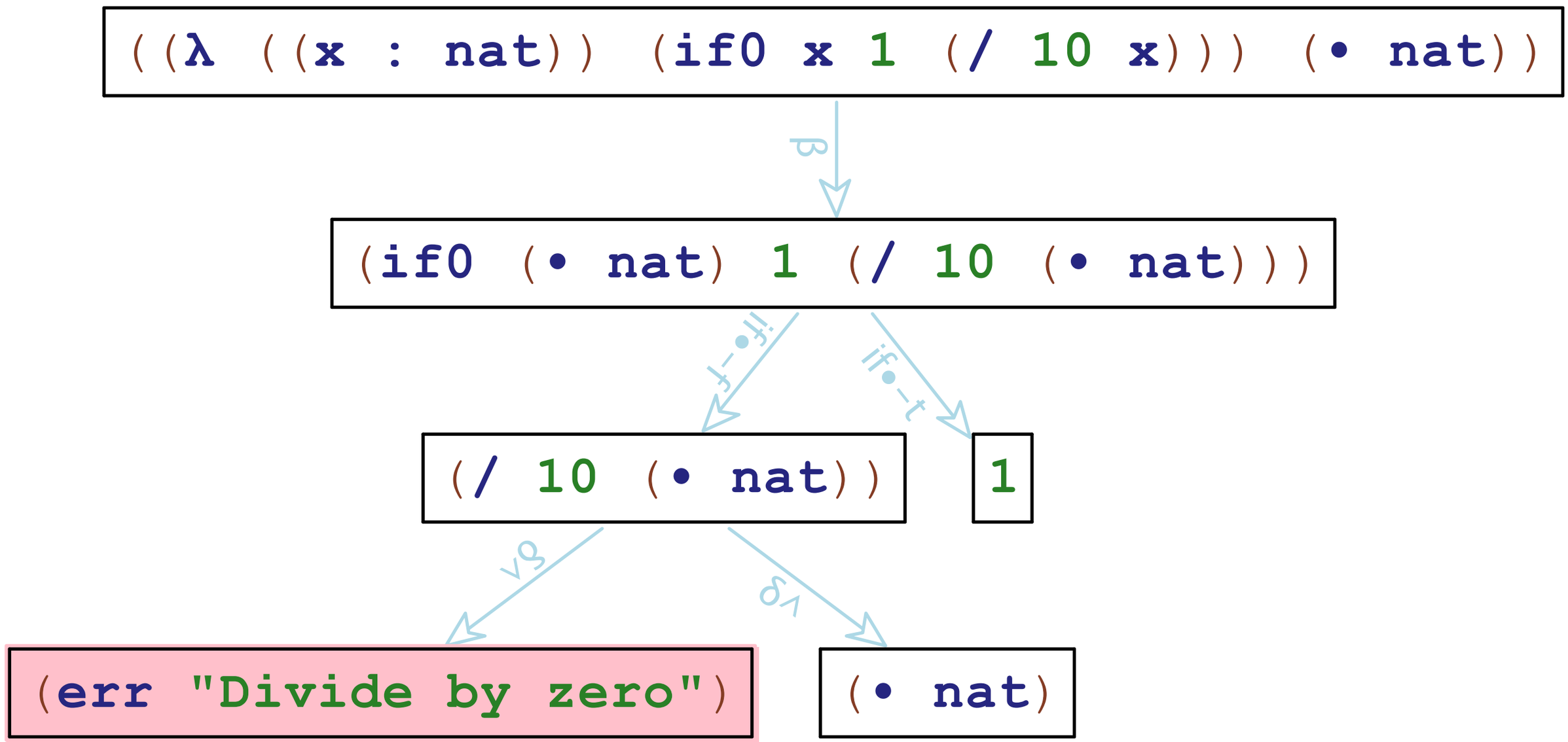
'CPCF

```
((λ ((x : nat)) (if0 x 1 (/ 10 x))) (• nat))
```

$\downarrow \beta$

```
(if0 (• nat) 1 (/ 10 (• nat)))
```

CPCF


((λ ((x : nat)) (if0 x 1 (/ 10 x))) (• nat))
β
(if0 (• nat) 1 (/ 10 (• nat)))
(/ 10 (• nat))
1
(err "Divide by zero")
(• nat)

'CPCF

```
(((λ ((x : nat)) (if0 x 1 (/ 10 x))) (• nat))
 #())
```

'CPCF

```
(((λ ((x : nat)) (if0 x 1 (/ 10 x))) (• nat))
 #())
```

β

```
((if0 (& 2) 1 (/ 10 (& 2)))
 #((1 ↦ (λ ((x : nat)) (if0 x 1 (/ 10 x))))
   (2 ↦ nat)))
```

1-Off!

'CPCF

```
(1
 #((1 ↦ (λ ((x : nat)) (if0 x 1 (/ 10 x))))
   (2 ↦ (nat zero?))))
```

'CPCF

if0-f

```
((/ 10 (& 2))
 #((1 ↦ (λ ((x : nat)) (if0 x 1 (/ 10 x))))
   (2 ↦ (nat pos?)))
```

```
((• nat)
 #((1 ↦ (λ ((x : nat)) (if0 x 1 (/ 10 x))))
   (2 ↦ (nat pos?))
   (3 ↦ /)
   (4 ↦ 10)))
```

CPCF

```
(((λ ((x : nat)) (if0 x 1 (/ 10 x))) (• nat))
 #())
```

β

```
((if0 (& 2) 1 (/ 10 (& 2)))
 #((1 ↦ (λ ((x : nat)) (if0 x 1 (/ 10 x))))
   (2 ↦ nat)))
```

if0-t

```
(1
 #((1 ↦ (λ ((x : nat)) (if0 x 1 (/ 10 x))))
   (2 ↦ (nat zero?))))
```

if0-f

```
((/ 10 (& 2))
 #((1 ↦ (λ ((x : nat)) (if0 x 1 (/ 10 x))))
   (2 ↦ (nat pos?))))
```

```
((• nat)
 #((1 ↦ (λ ((x : nat)) (if0 x 1 (/ 10 x))))
   (2 ↦ (nat pos?))
   (3 ↦ /)
   (4 ↦ 10)))
```

ESSENTIAL IDEAS:

1) CONTRACTS AS SYMBOLIC VALUES
2) CHECKS REFINE VALUES
3) REFINEMENTS INFLUENCE COMPUTATION
4) PARTITION VERIFICATION W/ BLAME
5) HEAP OF INVARIANTS -> SMT

ESSENTIAL IDEAS:

1) CONTRACTS AS SYMBOLIC VALUES

2) ~~CHECKS REFINE VALUES~~ CONDITIONALS REFINE LOCATIONS

3) REFINEMENTS INFLUENCE COMPUTATION

4) PARTITION VERIFICATION W/ BLAME

5) HEAP OF INVARIANTS -> SMT

'CPCF

```
(((λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)) (• nat))
 #())
```

'CPCF

```
(((λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)) (• nat))
 #())
```

B

```
((if0 (= (& 2) 5) (= 5 (& 2)) 0)
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)))
```

'CPCF

```
(((λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)) (• nat))
 #())
```

β

```
((if0 (= (& 2) 5) (= 5 (& 2)) 0)
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)))
```

δ-=?2

```
2)) 0)
 nat)) (if0 (= x 5) (= 5 x) 0)))
```

δ-=?1

```
((if0 0 (= 5 (& 2)) 0)
 #((1 ↦ (λ ((x : nat)) (if0 (= x
   (2 ↦ nat)
```

```
(2 ↦ nat)))
```

'CPCF

$\xi_i = -\xi$

```
((if0 1 (= 5 (& 2)) 0)
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)
   (3 ↦ =)
   (4 ↦ (5 (λ ((x : nat)) (not (= x (& 2)))))))
```

```
((if0 (
 #((1 ↦
   (2 ↦
   (3 ↦
   (4 ↦
```

if0-if

```
(0
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)
   (3 ↦ =)
   (4 ↦ (5 (λ ((x : nat)) (not (= x (& 2))))))
   (5 ↦ 1)))
```

```
((= 5
 #((1 ↦
   (2 ↦
   (3 ↦
   (4 ↦
   (5 ↦
```

'CPCF

$\delta_{-=?1}$

```
0)))

))))))
```

```
((if0 0 (= 5 (& 2)) 0)
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)
   (3 ↦ =)
   (4 ↦ (5 (λ ((x : nat)) (= x (& 2))))))
```

if0-

```
0)))

)))
```

```
((= 5 (& 2))
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)
   (3 ↦ =)
   (4 ↦ (5 (λ ((x : nat)) (= x (& 2)))))
   (5 ↦ 0)))
```

```
0))))
```

```
((= 5 (& 2))
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)
   (3 ↦ =)
   (4 ↦ (5 (λ ((x : nat)) (= x (& 2)))))
   (5 ↦ 0)))
```

'CPCF

```
(0
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)
   (3 ↦ =)
   (4 ↦ (5 (λ ((x : nat)) (= x (& 2)))))
   (5 ↦ 0)
   (6 ↦ =)
   (7 ↦ 5)))
```

```
0))))
```

```
((= 5 (& 2))
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)
   (3 ↦ =)
   (4 ↦ (5 (λ ((x : nat)) (= x (& 2)))))
   (5 ↦ 0)))
```

CPCF

Z3

```
x2, x4, x5 : nat
x4=5
x4=x2
x5=0
x2=5?
```

```
(0
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5)
   (2 ↦ nat)
   (3 ↦ =)
   (4 ↦ (5 (λ ((x : nat)) (= x (& 2)))))
   (5 ↦ 0)
   (6 ↦ =)
   (7 ↦ 5)))
```

CPCF

```
(((λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)) (• nat))
 #())
```

β

```
((if0 (= (& 2) 5) (= 5 (& 2)) 0)
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)))
```

δ-=?1

```
((if0 0 (= 5 (& 2)) 0)
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)
   (3 ↦ =)
   (4 ↦ (5 (λ ((x : nat)) (= x (& 2))))))
```

if0-f

```
((= 5 (& 2))
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)
   (3 ↦ =)
   (4 ↦ (5 (λ ((x : nat)) (= x (& 2)))))
   (5 ↦ 0)))
```

δ-=?2

```
((if0 1 (= 5 (& 2)) 0)
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)
   (3 ↦ =)
   (4 ↦ (5 (λ ((x : nat)) (not (= x (& 2)))))))
```

if0-t

```
(0
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)
   (3 ↦ =)
   (4 ↦ (5 (λ ((x : nat)) (not (= x (& 2))))))
   (5 ↦ 1)))
```

```
(0
 #((1 ↦ (λ ((x : nat)) (if0 (= x 5) (= 5 x) 0)))
   (2 ↦ nat)
   (3 ↦ =)
   (4 ↦ (5 (λ ((x : nat)) (= x (& 2)))))
   (5 ↦ 0)
   (6 ↦ =)
   (7 ↦ 5)))
```

ESSENTIAL IDEAS:

1) CONTRACTS AS SYMBOLIC VALUES

2) ~~CHECKS REFINE VALUES~~ CONDITIONALS REFINE LOCATIONS

3) REFINEMENTS INFLUENCE COMPUTATION

4) PARTITION VERIFICATION W/ BLAME

5) HEAP OF INVARIANTS -> SMT

BEYOND 'CPCF

'RACKET

$$
\begin{array}{lcl}
P, Q & ::= & \vec{M}\,E \\
M, N & ::= & (\mathsf{module}\ f\ C\ V) \\
E, E' & ::= & f^{\ell} \mid X \mid A \mid E\ E^{\ell} \mid \mathsf{if}\ E\ E\ E \mid O\ \vec{E}^{\ell} \mid \mu X.E \\
 & \mid & \mathsf{mon}^{\ell,\ell}_{\ell}(C, E) \\
U & ::= & n \mid \mathsf{tt} \mid \mathsf{ff} \mid (\lambda X.E) \mid \bullet \mid (V, V) \mid \mathsf{empty} \\
V & ::= & U / \mathcal{C} \\
C, D & ::= & X \mid C \mapsto \lambda X.C \mid \mathsf{flat}(E) \\
 & \mid & \langle C, C \rangle \mid C \vee C \mid C \wedge C \mid \mu X.C \\
O & ::= & \mathsf{add1} \mid \mathsf{car} \mid \mathsf{cdr} \mid \mathsf{cons} \mid + \mid = \mid o? \mid \ldots \\
o? & ::= & \mathsf{nat?} \mid \mathsf{bool?} \mid \mathsf{empty?} \mid \mathsf{cons?} \mid \mathsf{proc?} \mid \mathsf{false?} \\
A & ::= & V \mid \mathcal{E}[\mathsf{blame}^{\ell}_{\ell}]
\end{array}
$$

Theorem: verified modules can't be blamed.

SOFT TYPING

OCCURRENCE TYPING

H.O. RECURSION SCHEMES

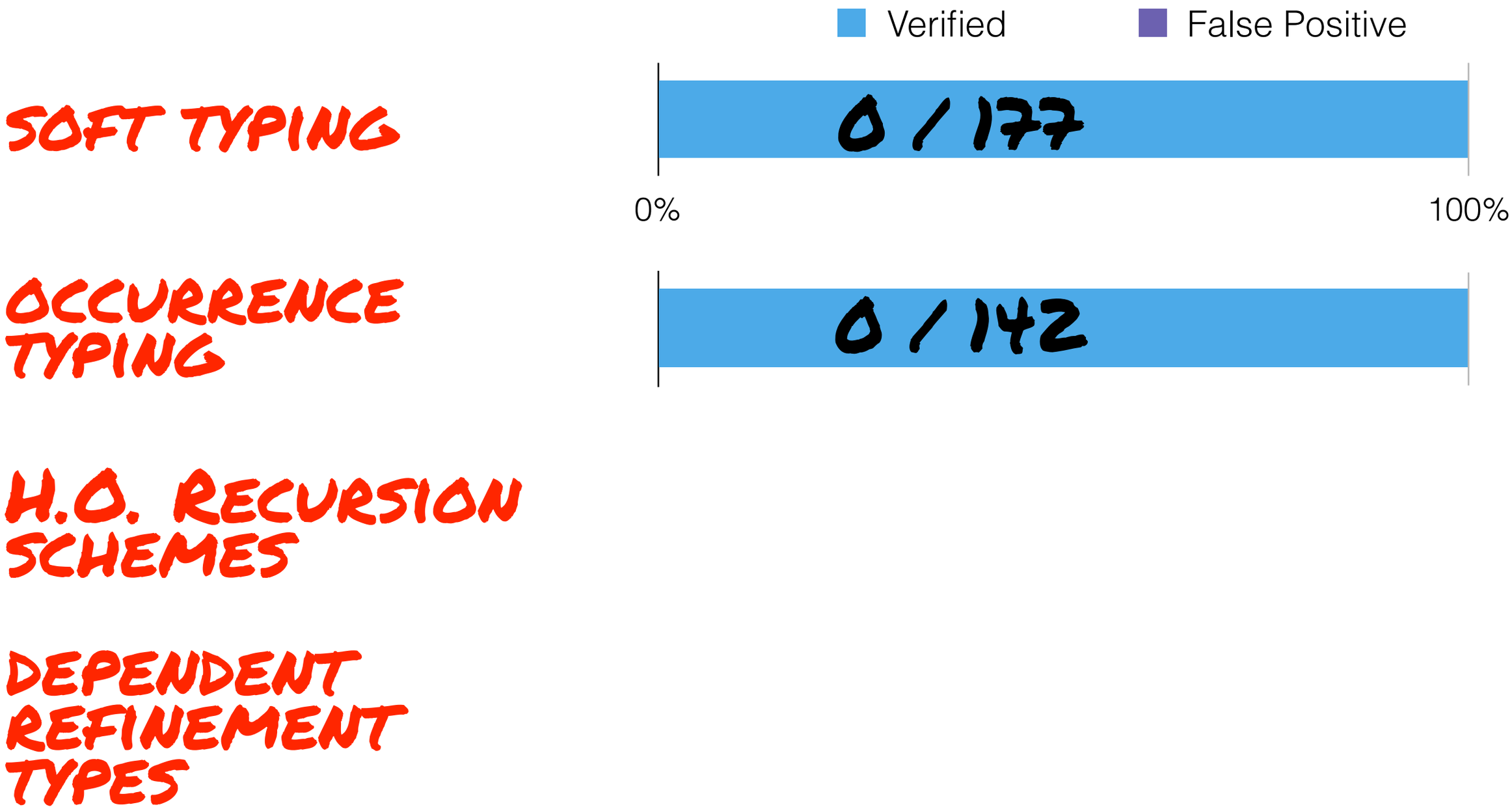
DEPENDENT REFINEMENT TYPES

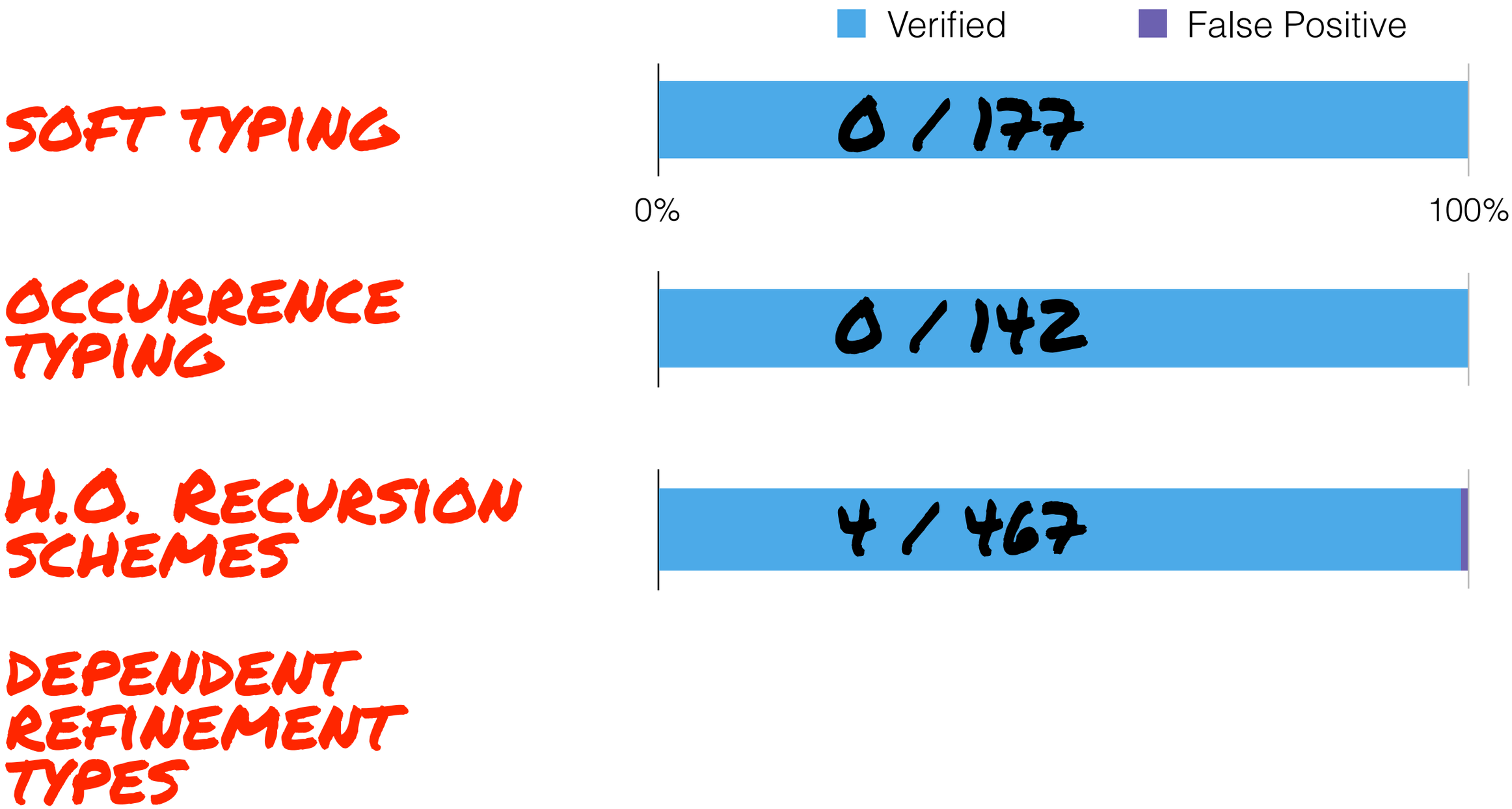
SOFT TYPING

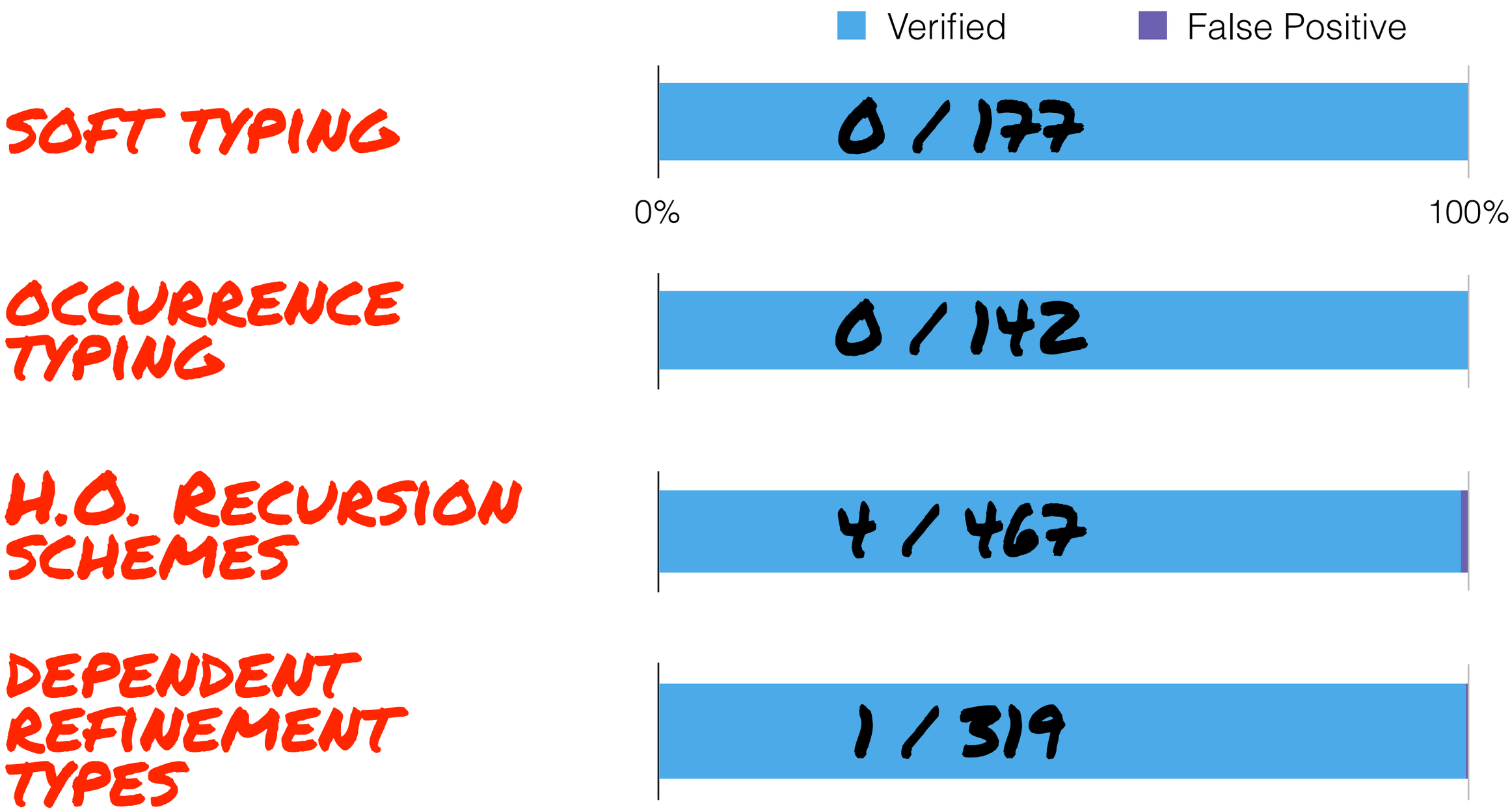
OCCURRENCE TYPING

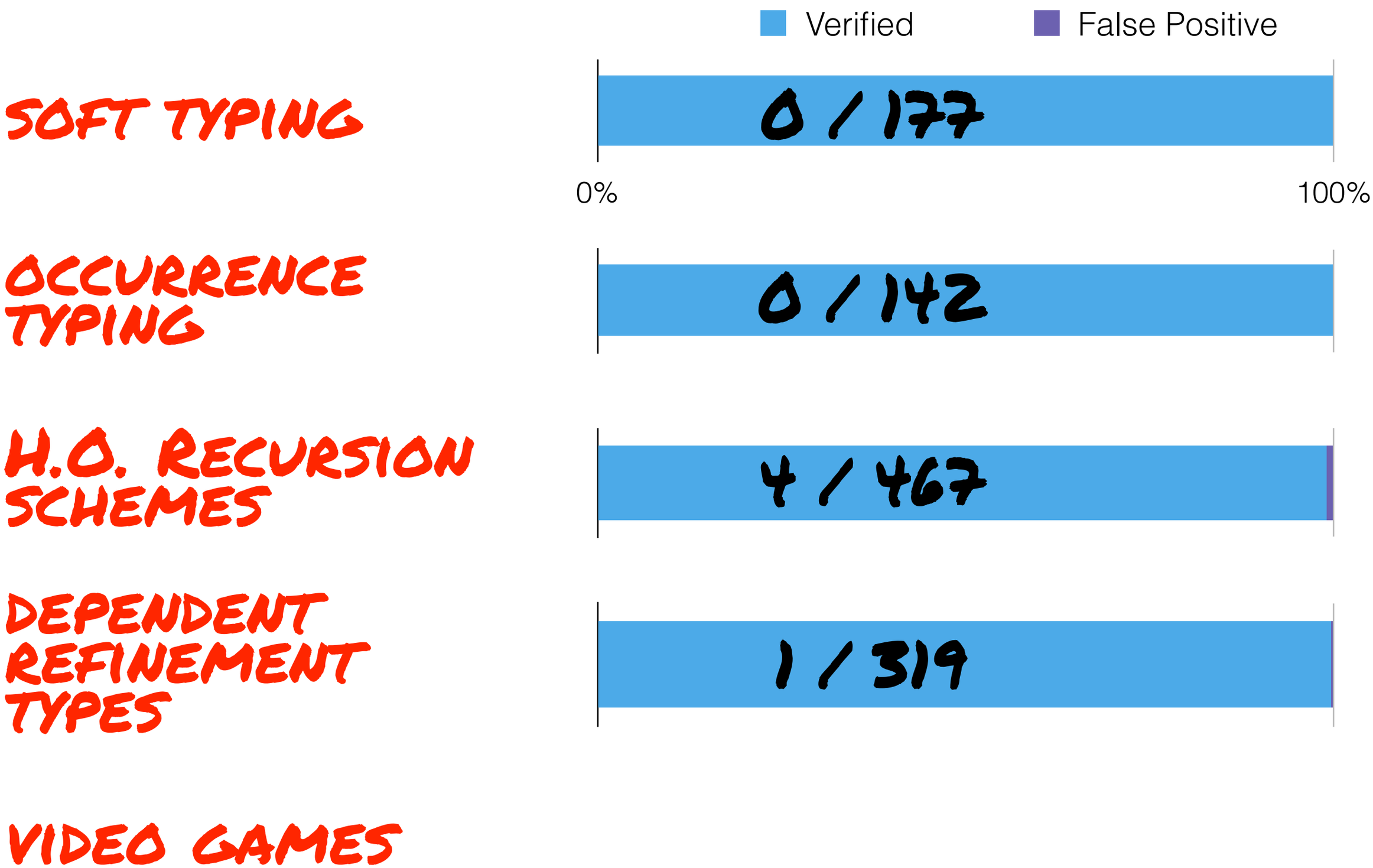
H.O. RECURSION SCHEMES

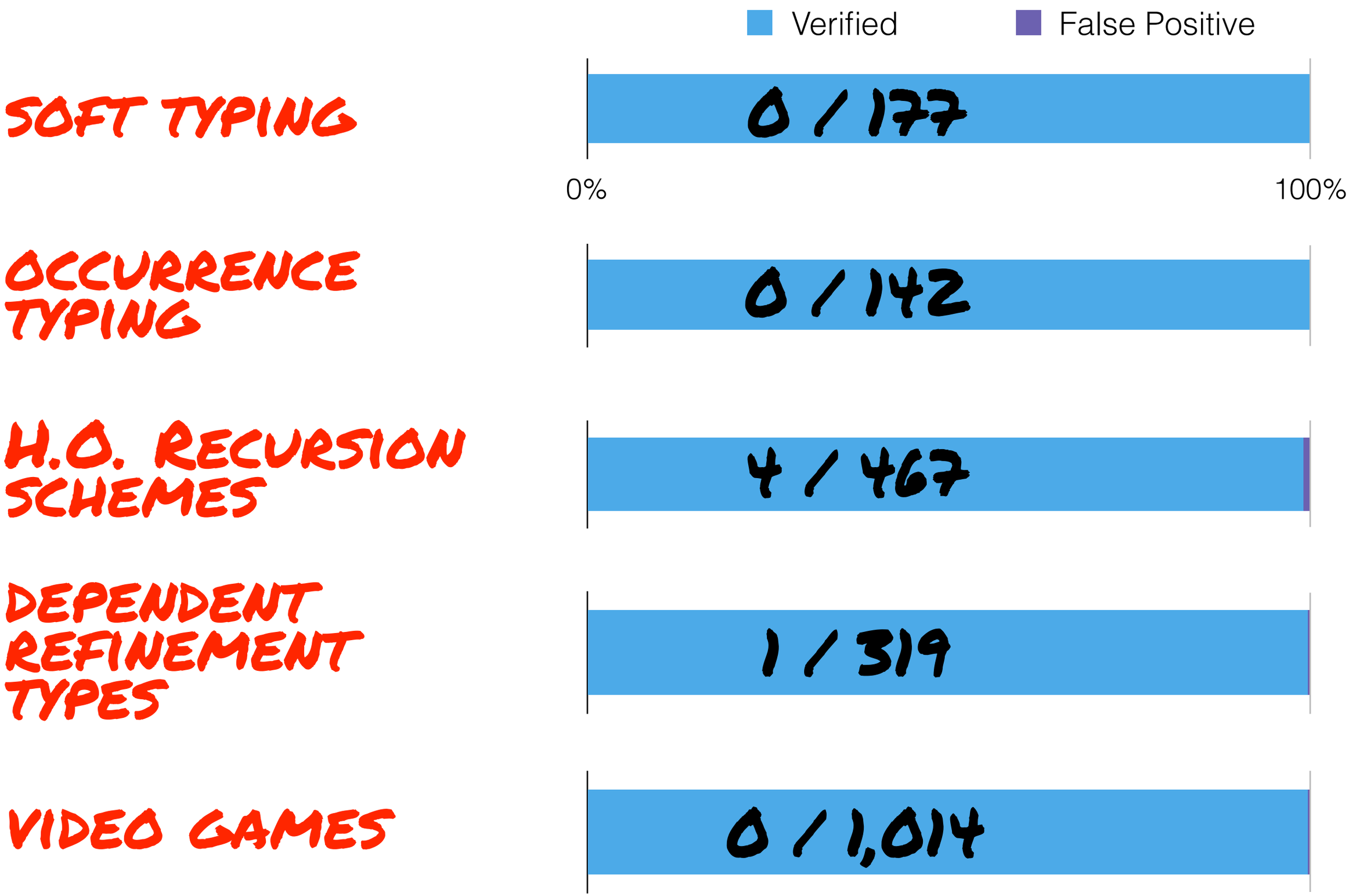
DEPENDENT REFINEMENT TYPES


Verified
False Positive
SOFT TYPING
0 / 177
0%
100%
OCCURRENCE TYPING
0 / 142
H.O. RECURSION SCHEMES
DEPENDENT REFINEMENT TYPES


Verified
False Positive
SOFT TYPING
0 / 177
0%
100%
OCCURRENCE TYPING
0 / 142
H.O. RECURSION SCHEMES
4 / 467
DEPENDENT REFINEMENT TYPES


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


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

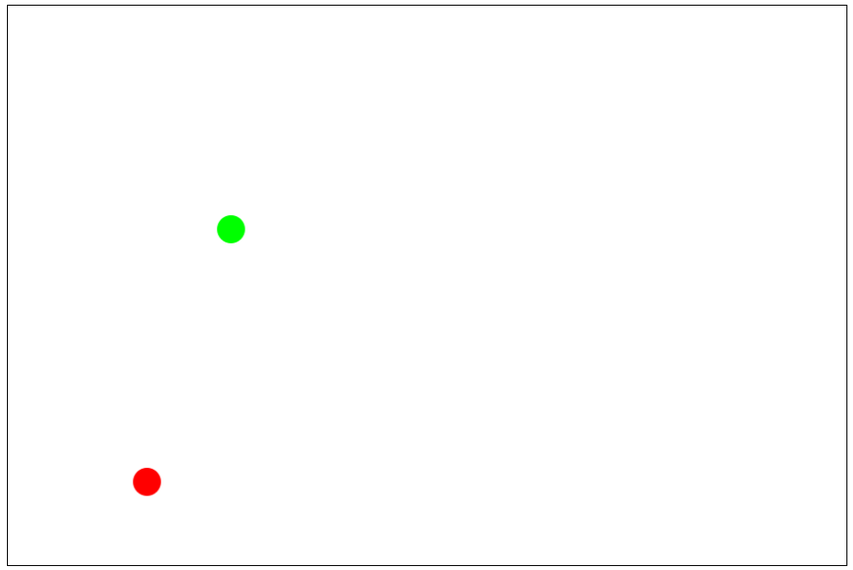

Verified
False Positive
SOFT TYPING
0 / 177
0%
100%
OCCURRENCE TYPING
0 / 142
H.O. RECURSION SCHEMES
4 / 467
DEPENDENT REFINEMENT TYPES
1 / 319
VIDEO GAMES
0 / 1,014

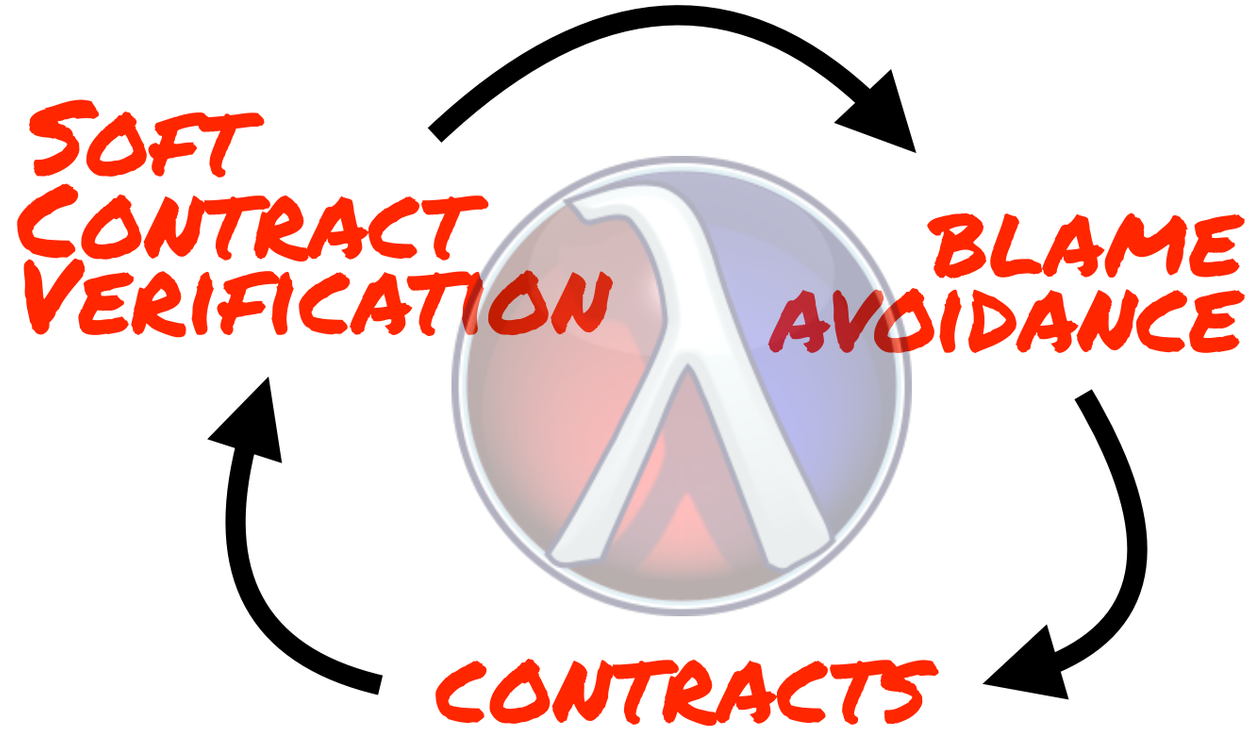
TAKE AWAY

TAKE AWAY

- CONTRACTS FAIL AT FIRST ORDER;
  ONLY NEED A F.O. SOLVER FOR VERIFICATION

- RUNTIME ENFORCEMENT MECHANISMS
  => STATIC VERIFICATION TECHNIQUES





THEOREM:

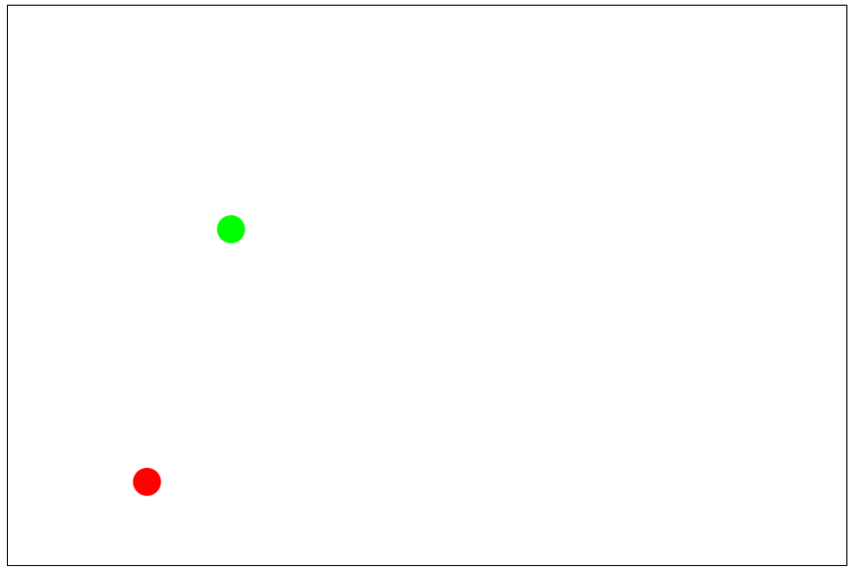
VERIFIED MODULES CAN'T BE BLAMED.

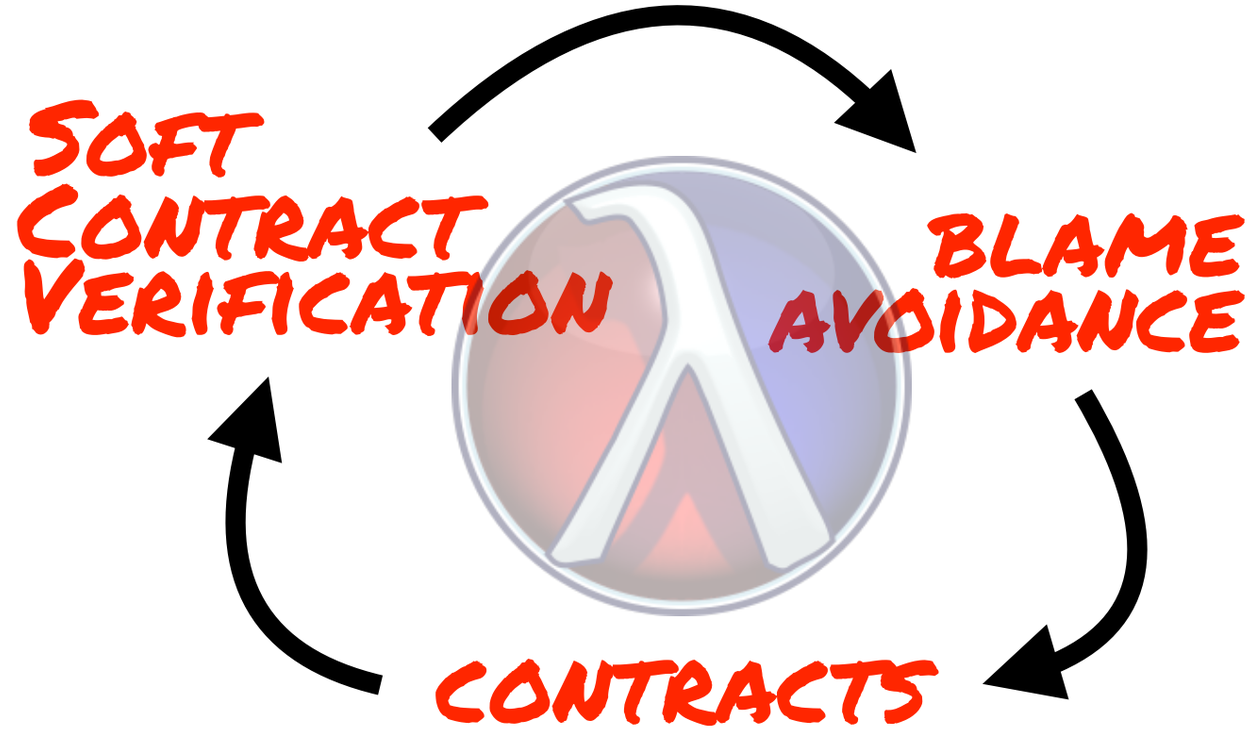
ESSENTIAL IDEAS:

1) CONTRACTS AS SYMBOLIC VALUES
2) ~~CHECKS REFINE VALUES~~ CONDITIONALS REFINE LOCATIONS
3) REFINEMENTS INFLUENCE COMPUTATION
4) PARTITION VERIFICATION W/ BLAME
5) HEAP OF INVARIANTS -> SMT

PLUM

ter.ps/softcontract





THEOREM:

VERIFIED MODULES CAN'T BE BLAMED.

ESSENTIAL IDEAS:

1) CONTRACTS AS SYMBOLIC VALUES
2) ~~CHECKS REFINE VALUES~~ CONDITIONALS REFINE LOCATIONS
3) REFINEMENTS INFLUENCE COMPUTATION
4) PARTITION VERIFICATION W/ BLAME
5) HEAP OF INVARIANTS -> SMT

PLUM

ter.ps/softcontract