Applying Random Testing to a Base Type Environment

Experience Report

Vincent St-Amour

Neil Toronto

PLT

ICFP 2013 - September 27th, 2013

$$\Gamma_{\text{base}} \vdash e : \tau$$

$$\Gamma_{\text{base}} \vdash \overset{\Big\downarrow}{e} : \tau$$

$$\Gamma_{\text{base}} \vdash e : \overset{\Downarrow}{\tau}$$

$$\Gamma_{\text{base}} \overset{\Big\downarrow}{\vdash} e : \tau$$

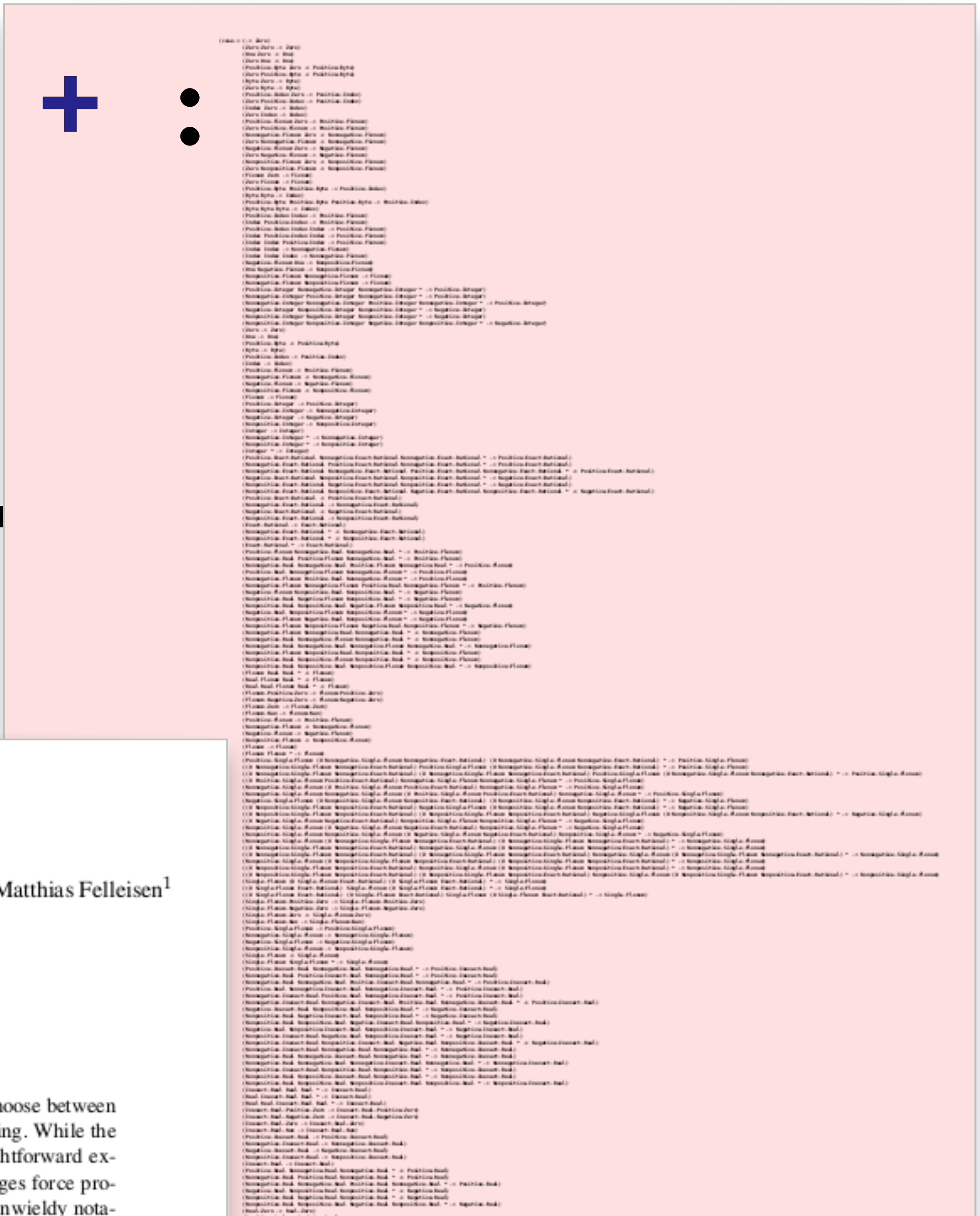
$$\Gamma_{\text{base}} \vdash e : \tau$$

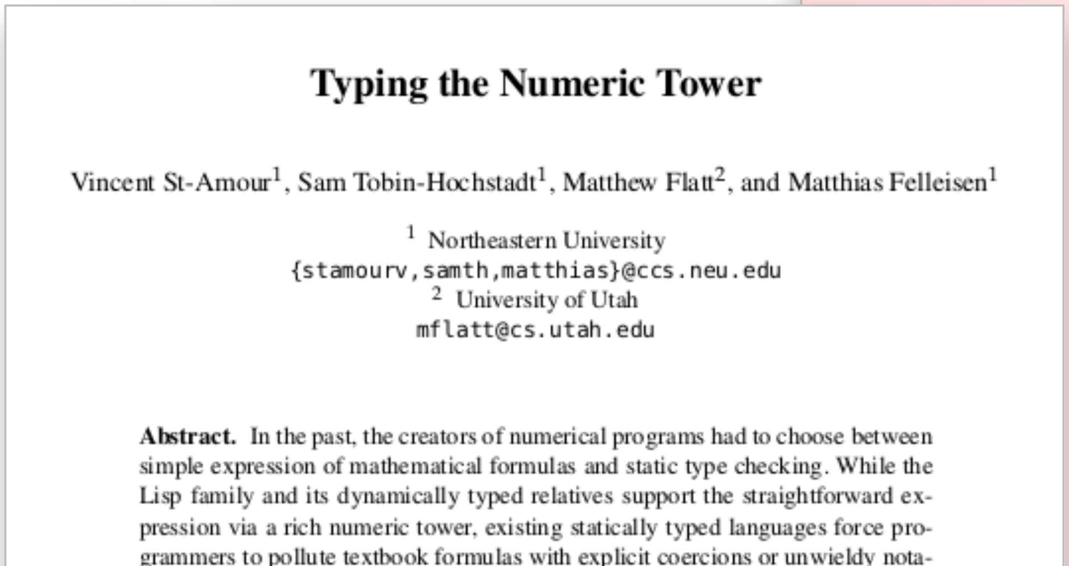
```
first : (Listof A) → A
```

$$\Gamma_{\text{base}} \vdash e : \tau$$

```
string-append : String * → String
```

$\Gamma_{base} \vdash$





Typing the Numeric Tower

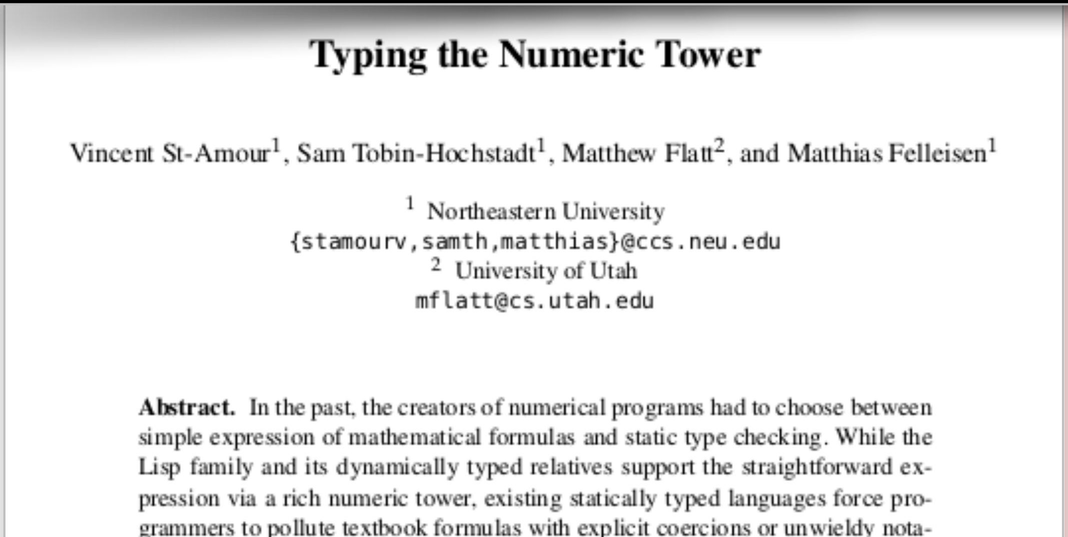
Vincent St-Amour[1], Sam Tobin-Hochstadt[1], Matthew Flatt[2], and Matthias Felleisen[1]

[1] Northeastern University
{stamourv,samth,matthias}@ccs.neu.edu
[2] University of Utah
mflatt@cs.utah.edu

Abstract. In the past, the creators of numerical programs had to choose between simple expression of mathematical formulas and static type checking. While the Lisp family and its dynamically typed relatives support the straightforward expression via a rich numeric tower, existing statically typed languages force programmers to pollute textbook formulas with explicit coercions or unwieldy nota-

+ :

```
/* This macro is used to implement most all
   binary math and comparison functions (!): */
#define GEN_BIN_THING(rettype, name, scheme_name, \
                      iop, fop, fsop, bn_op, rop,
                      wrap, combineinf, \
                      ...
```

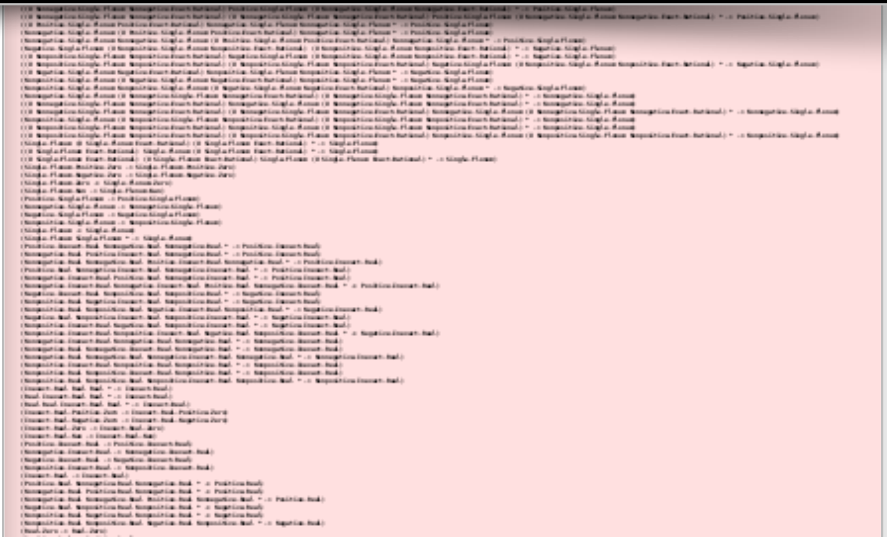


Typing the Numeric Tower

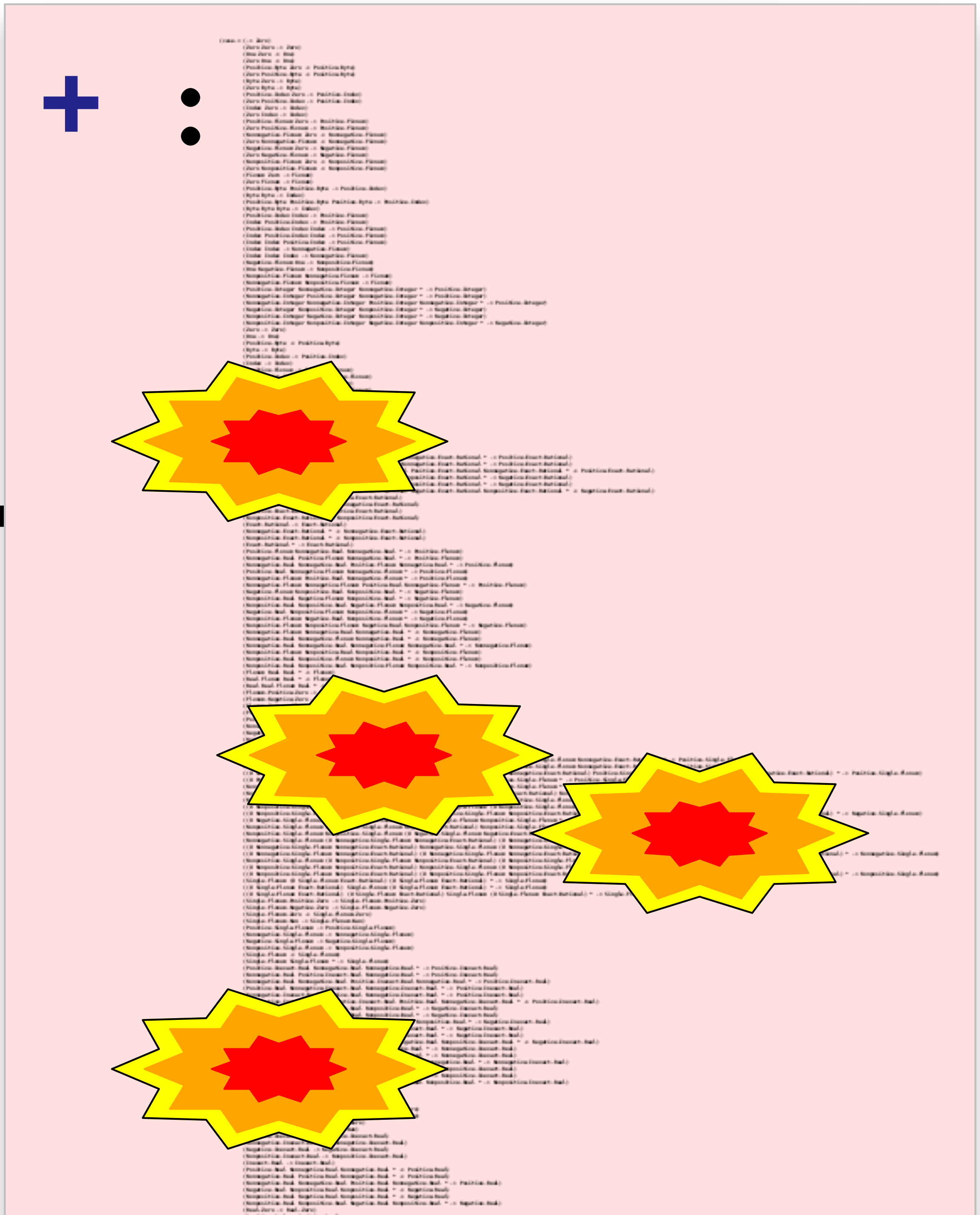
Vincent St-Amour[1], Sam Tobin-Hochstadt[1], Matthew Flatt[2], and Matthias Felleisen[1]

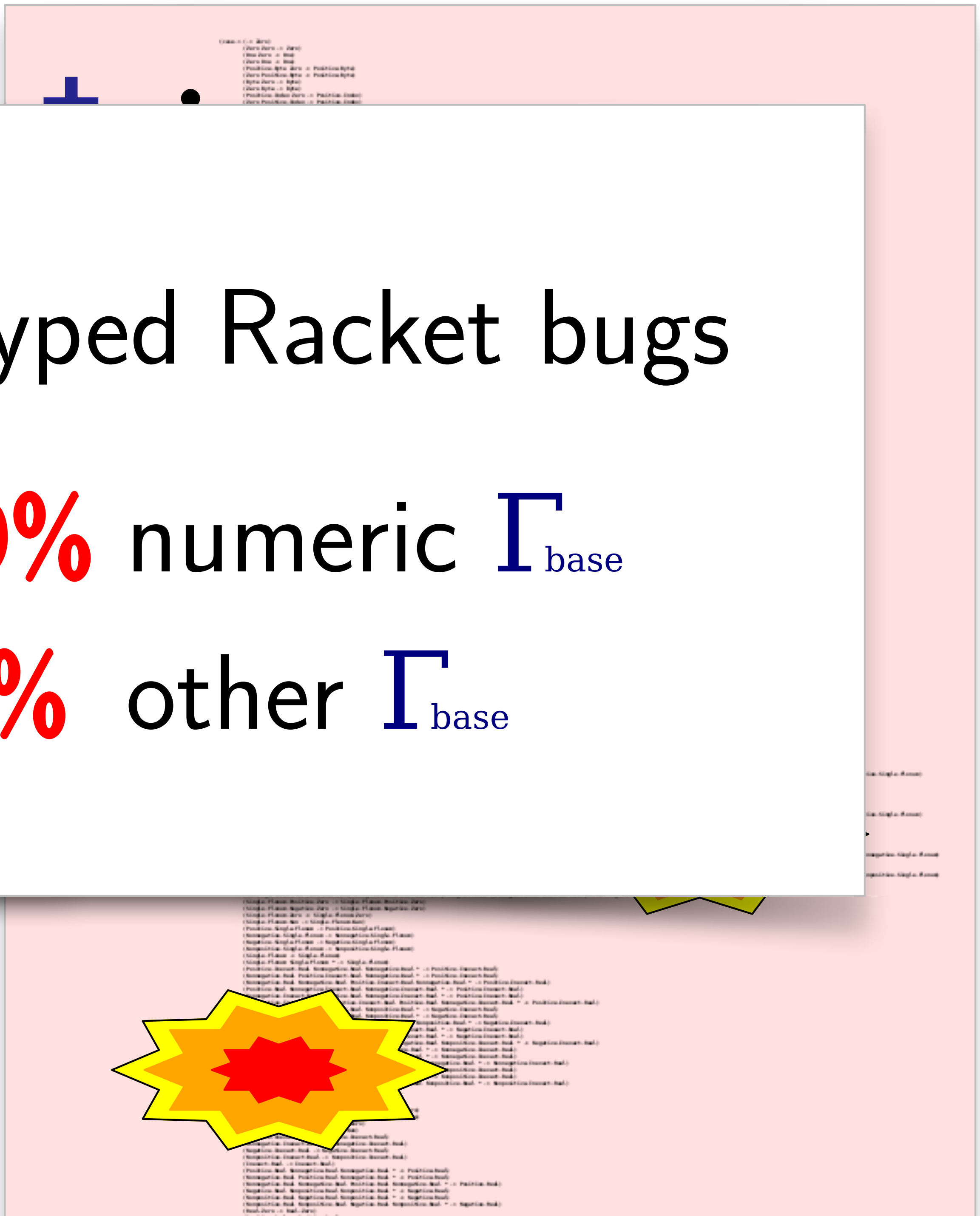
[1] Northeastern University
{stamourv,samth,matthias}@ccs.neu.edu
[2] University of Utah
mflatt@cs.utah.edu

Abstract. In the past, the creators of numerical programs had to choose between simple expression of mathematical formulas and static type checking. While the Lisp family and its dynamically typed relatives support the straightforward expression via a rich numeric tower, existing statically typed languages force programmers to pollute textbook formulas with explicit coercions or unwieldy nota-



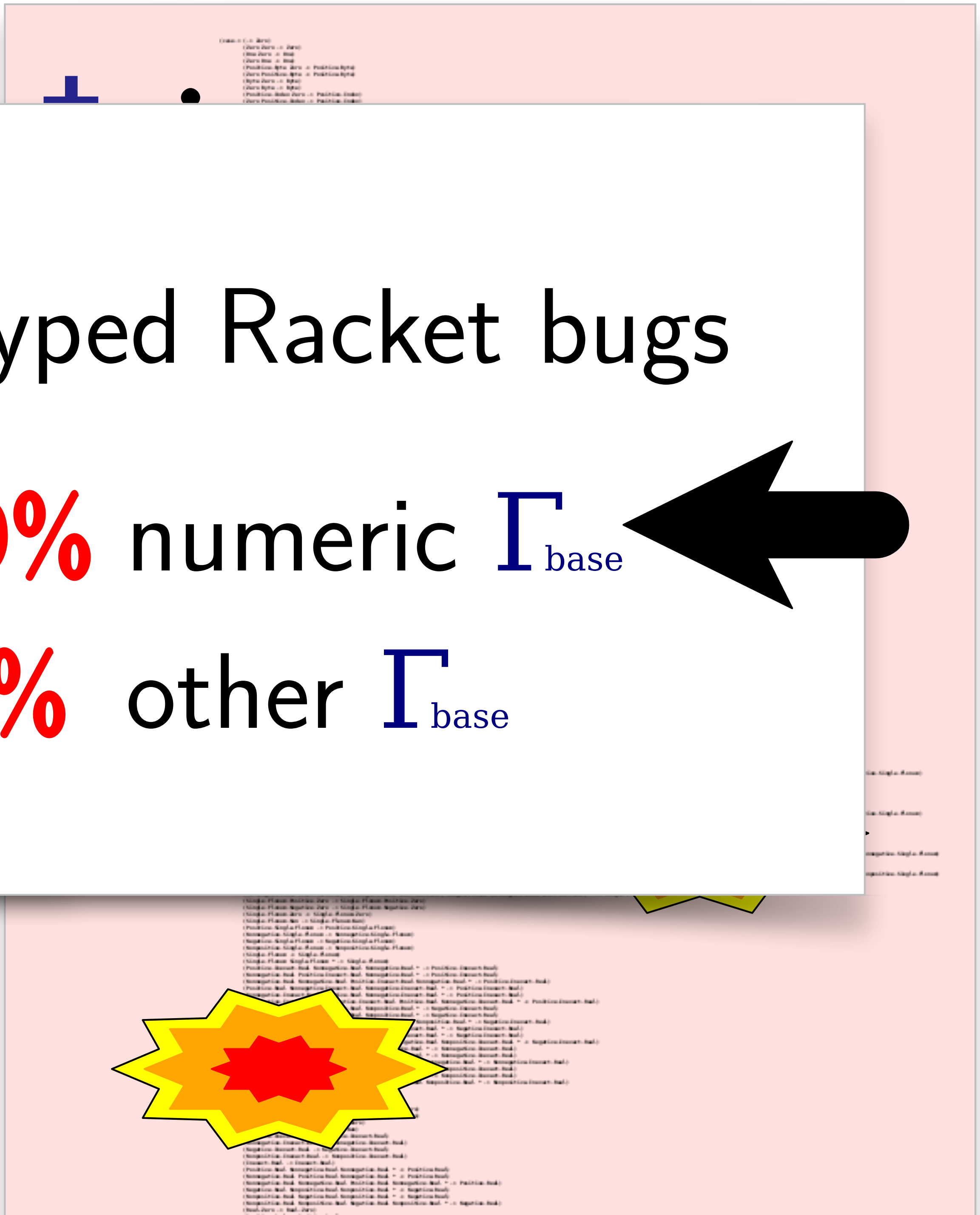
$\Gamma_{\text{base}} \vdash$





18% of Typed Racket bugs

$$\left\{\begin{array}{l}\textbf{10.9\%}\ \text{numeric}\ \Gamma_{\text{base}} \\ \textbf{6.8\%}\ \text{other}\ \Gamma_{\text{base}}\end{array}\right.$$



18% of Typed Racket bugs

$$\left\{\begin{array}{l}\textbf{10.9\%}\ \text{numeric}\ \Gamma_{\mathrm{base}} \\ \textbf{6.8\%}\ \text{other}\ \Gamma_{\mathrm{base}}\end{array}\right.$$

+ :

Use random testing

What do these bugs look like?

How do we find them?

How well did that work?

What do these bugs look like?

A Type Environment Bug

```
(: sinh (case→
         [Float-Zero     → Float-Zero]
         [Positive-Float → Positive-Float]
         [Negative-Float → Negative-Float]
         ...))
```

A Type Environment Bug

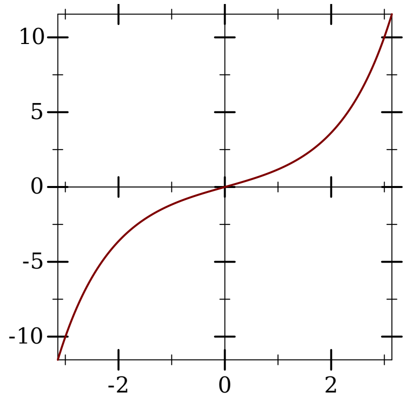
$\tau_1 \cap \tau_2 \cap \ldots$

```
(: sinh (case→
         [Float-Zero      → Float-Zero]
         [Positive-Float → Positive-Float]
         [Negative-Float → Negative-Float]
         ...))
```

A Type Environment Bug

$\tau_1 \cap \tau_2 \cap \ldots$

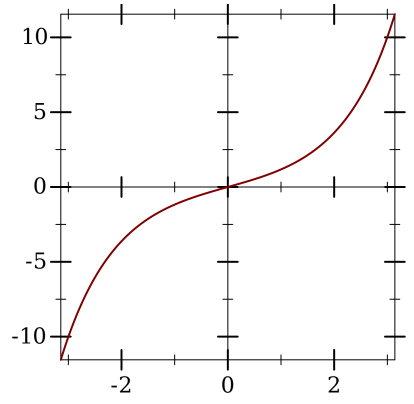
```
(: sinh (case→
          [Float-Zero      → Float-Zero]
          [Positive-Float → Positive-Float]
          [Negative-Float → Negative-Float]
          ...))
```



A Type Environment Bug

τ₁ ∩ τ₂ ∩ …

```
(: sinh (case→
          [Float-Zero       → Float-Zero]
          [Positive-Float → Positive-Float]
          [Negative-Float → Negative-Float]
          ...))
```



```
(: sinh
   (case-> (Zero -> Zero)
           ((U Nonnegative-Flonum
               Positive-Exact-Rational)
             -> Nonnegative-Flonum)
           ((U Negative-Exact-Rational
               Nonpositive-Flonum)
             -> Nonpositive-Flonum)
           (Flonum-Nan -> Flonum-Nan)
           (Flonum -> Flonum)
           (Single-Flonum-Nan
             -> Single-Flonum-Nan)
           (Nonnegative-Single-Flonum
             -> Nonnegative-Single-Flonum)
           (Nonpositive-Single-Flonum
             -> Nonpositive-Single-Flonum)
           (Single-Flonum -> Single-Flonum)
           (Nonnegative-Inexact-Real
             -> Nonnegative-Inexact-Real)
           (Nonpositive-Inexact-Real
             -> Nonpositive-Inexact-Real)
           (Inexact-Real -> Inexact-Real)
           (Nonnegative-Real -> Nonnegative-Real)
           (Nonpositive-Real -> Nonpositive-Real)
           (Real -> Real)
           (Float-Complex -> Float-Complex)
           (Single-Flonum-Complex
             -> Single-Flonum-Complex)
           (Inexact-Complex -> Inexact-Complex)
           (Number -> Number)))
```

19 cases

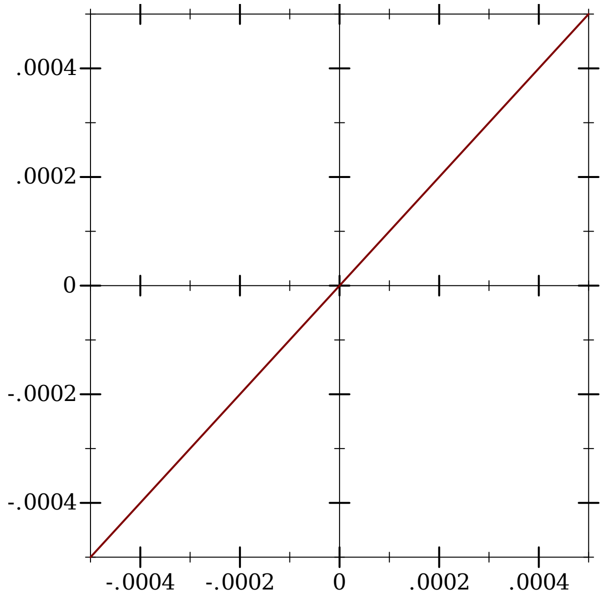
(integers, complexes, exact rationals)

A Type Environment Bug

```
(: sinh (case→
          [Float-Zero     → Float-Zero]
          [Positive-Float → Positive-Float]
          [Negative-Float → Negative-Float]
          ...))
```

A Type Environment Bug

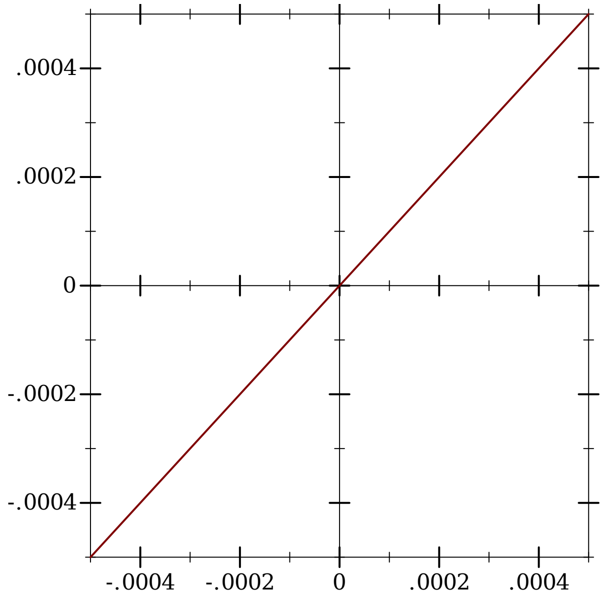
```
(: sinh (case→
         [Float-Zero      → Float-Zero]
         [Positive-Float → Positive-Float]
         [Negative-Float → Negative-Float]
         ...))
```



```
(sinh 1.2535e-17)
⇒ 0.0 : Float-Zero
```

A Type Environment Bug

```
(: sinh (case→
         [Float-Zero        → Float-Zero]
         [Nonnegative-Float → Nonnegative-Float]
         [Nonpositive-Float → Nonpositive-Float]
         ...))
```



```
(sinh 1.2535e-17)
⇒ 0.0 : Float-Zero
```

A Type Environment Bug

```
(: * (case→
       ...
       [Positive-Real Positive-Real
         → Positive-Real]
       ...))
```

A Type Environment Bug

```
(: * (case→
       ...
       [Positive-Real Positive-Real
         → Positive-Real]
       ...))
```

```
(* 5/1241 4.9406564584125e-324)
⇒ 0.0 : Float-Zero
```

A Type Environment Bug

```
(: * (case→
       ...
       [Nonnegative-Real Nonnegative-Real
         → Nonnegative-Real]
       ...))
```

```
(* 5/1241 4.9406564584125e-324)
⇒ 0.0 : Float-Zero
```

A Type Environment Bug

```
(: * (case→
       ...
       [Nonnegative-Real Nonnegative-Real
         → Nonnegative-Real]
       ...))
```

```
(* 5/1241 4.9406564584125e-324)
⇒ 0.0 : Float-Zero

(* +inf.0 0.0)
⇒ +nan.0 : Float-Nan
```

A Type Environment Bug

```
(: * (case→
      ...
      [Nonnegative-Real Nonnegative-Real
        → (U Nonnegative-Real Float-Nan)]
      ...))
```

```
(* 5/1241 4.9406564584125e-324)
⇒ 0.0 : Float-Zero

(* +inf.0 0.0)
⇒ +nan.0 : Float-Nan
```

How do we find them?

Use random testing

PLT Redex

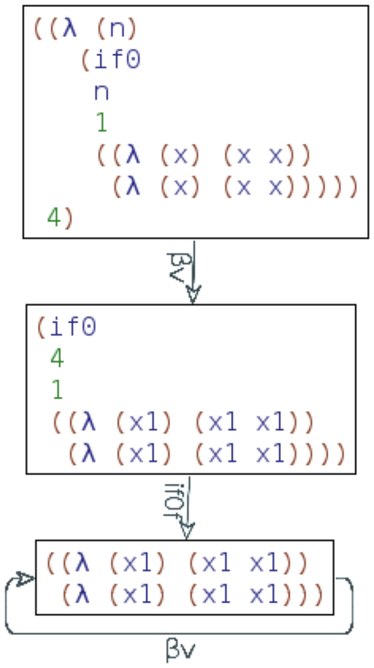
PLT Redex

```
(define-language λv
  [e (e e ...)
     (if0 e e e)
     x
     v]
  [v (λ (x ...) e)
     number]
  [x (variable-except λ if0)])
```

PLT Redex

```
(define-language λv
  [e (e e ...)
     (if0 e e e)
     x
     v]
  [v (λ (x ...) e)
     number]
  [x (variable-except λ if0)])
```

```
(define red
  (reduction-relation λv ...))
```



```
((λ (n)
   (if0
    n
    1
    ((λ (x) (x x))
     (λ (x) (x x)))))
 4)
```

```
(if0
 4
 1
 ((λ (x1) (x1 x1))
  (λ (x1) (x1 x1))))
```

```
((λ (x1) (x1 x1))
 (λ (x1) (x1 x1)))
```

PLT Redex

```
(define-language λv
  [e (e e ...)
     (if0 e e e)
     x
     v]
  [v (λ (x ...) e)
     number]
  [x (variable-except λ if0)])
```

```
(redex-check λv v
  (number? (term v)))
```

```
counterexample found
 after 4 attempts:
(λ () 1)
```

PLT Redex

```
(define-language λv
  [e (e e ...)
     (if0 e e e)
     x
     v]
  [v (λ (x ...) e)
     number]
  [x (variable-except λ if0)])
```

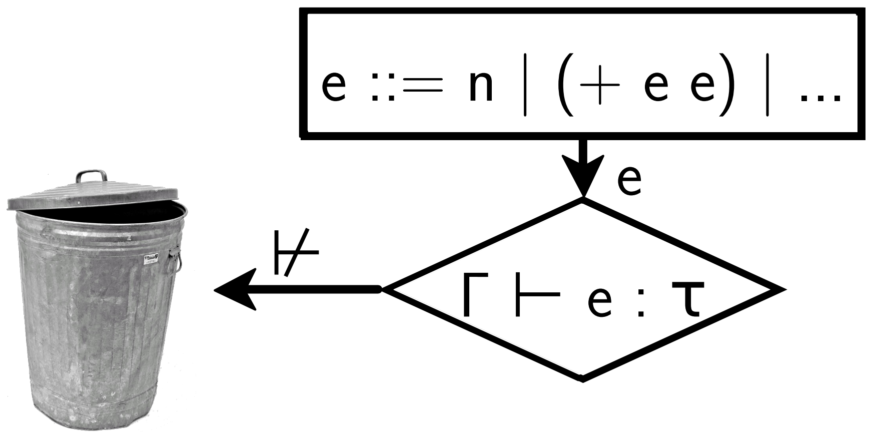
```
(redex-check λv v
  (> (n-google-results
       (term v))
     20))
```

```
counterexample found
 after 15 attempts:
(λ (x y) (+ (λ () 3) 2))
```

Testing Type Preservation

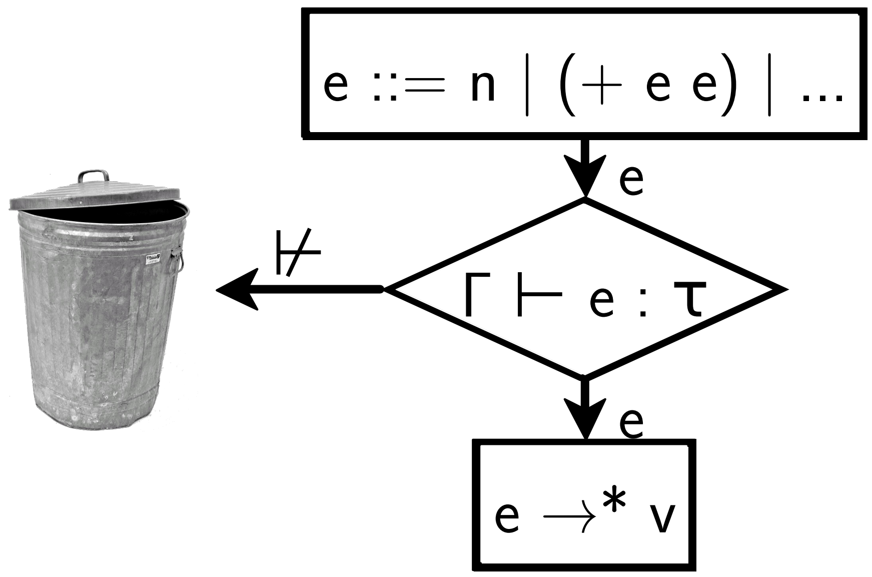
$\boxed{\mathsf{e ::= n \mid (+\ e\ e) \mid \ldots}}$ Generate arithmetic expressions

Testing Type Preservation



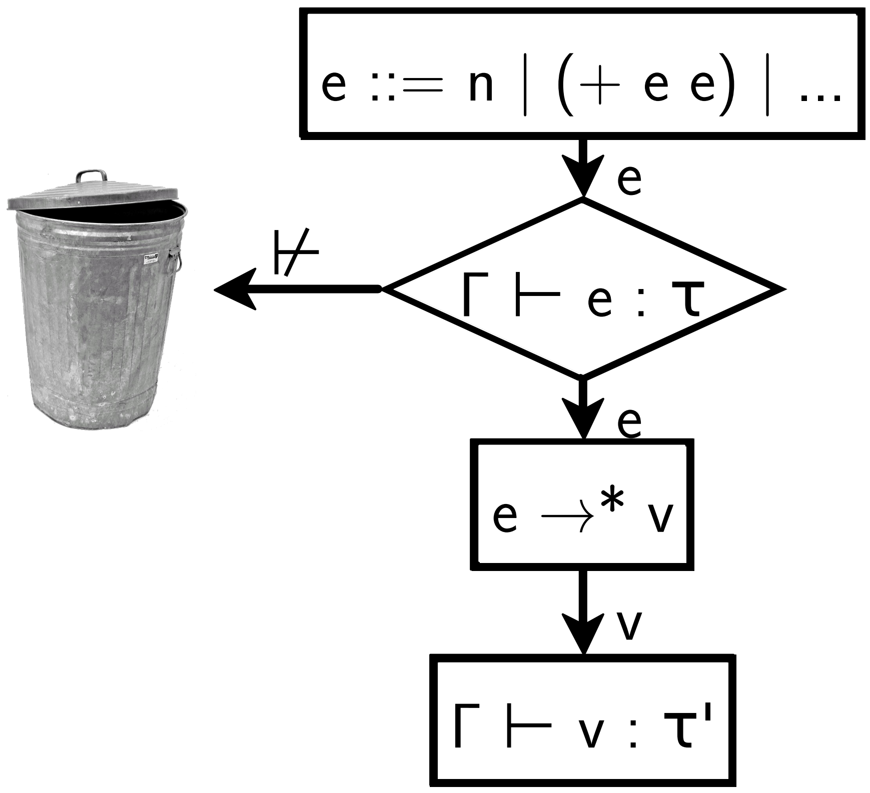
Typecheck using Typed Racket

Testing Type Preservation



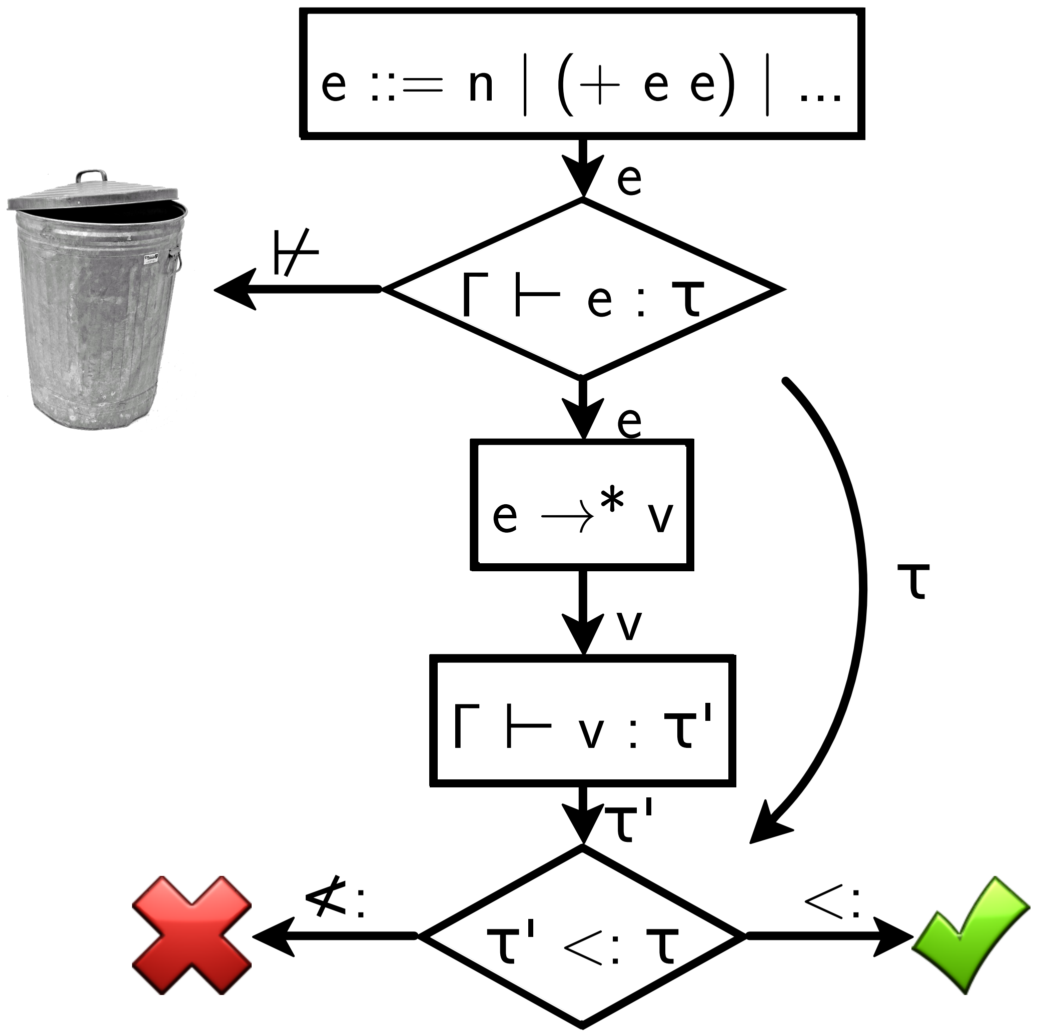
Evaluate using Typed Racket

Testing Type Preservation



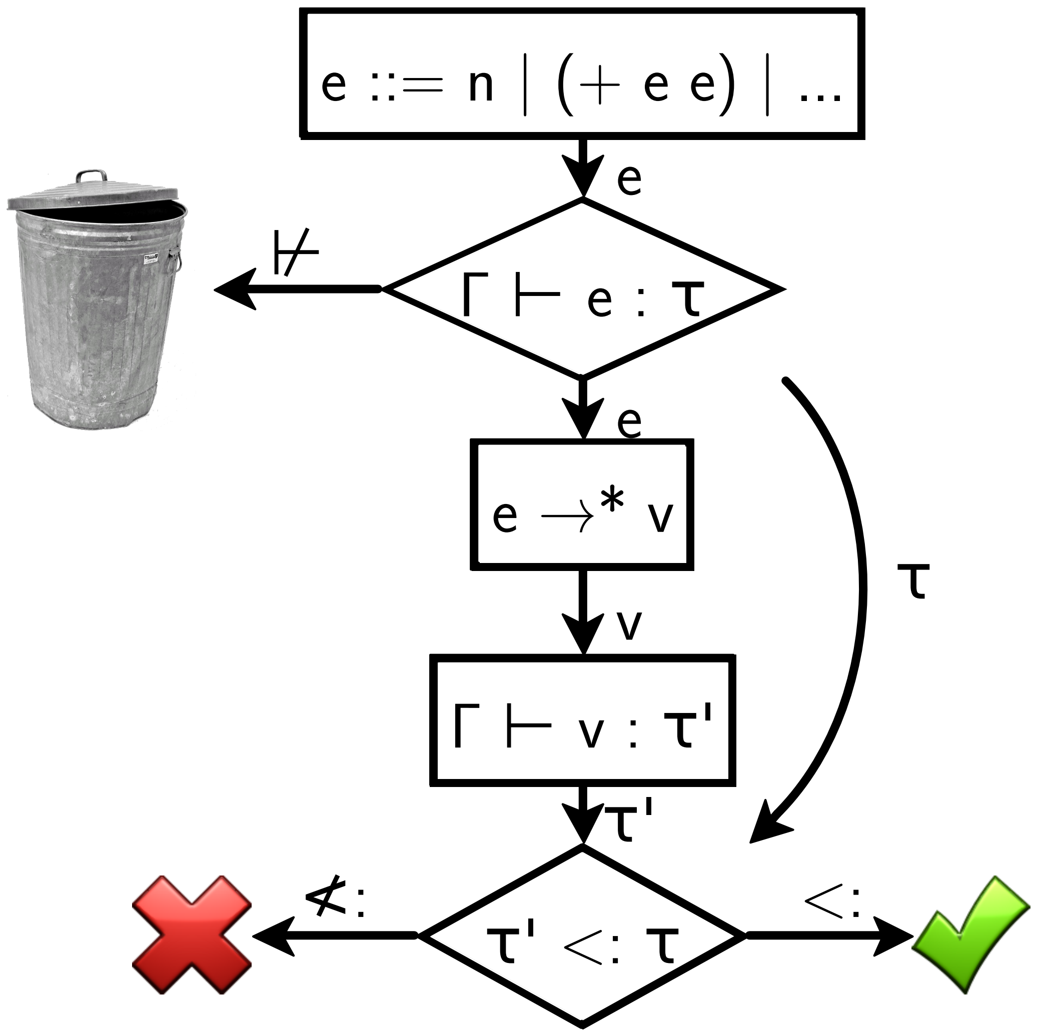
Typecheck the result

Testing Type Preservation



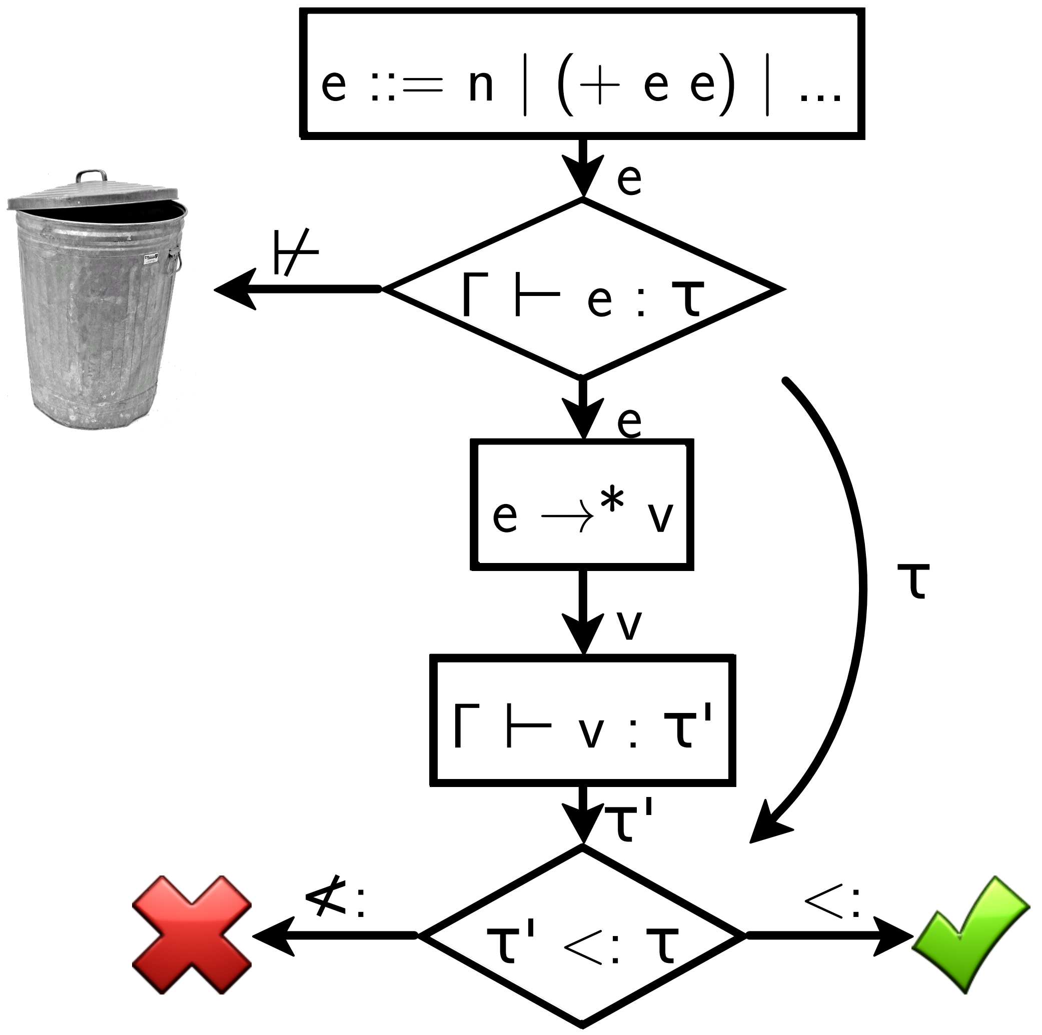
Check consistency

Testing Type Preservation



```
(sinh 1.2535e-17)
```

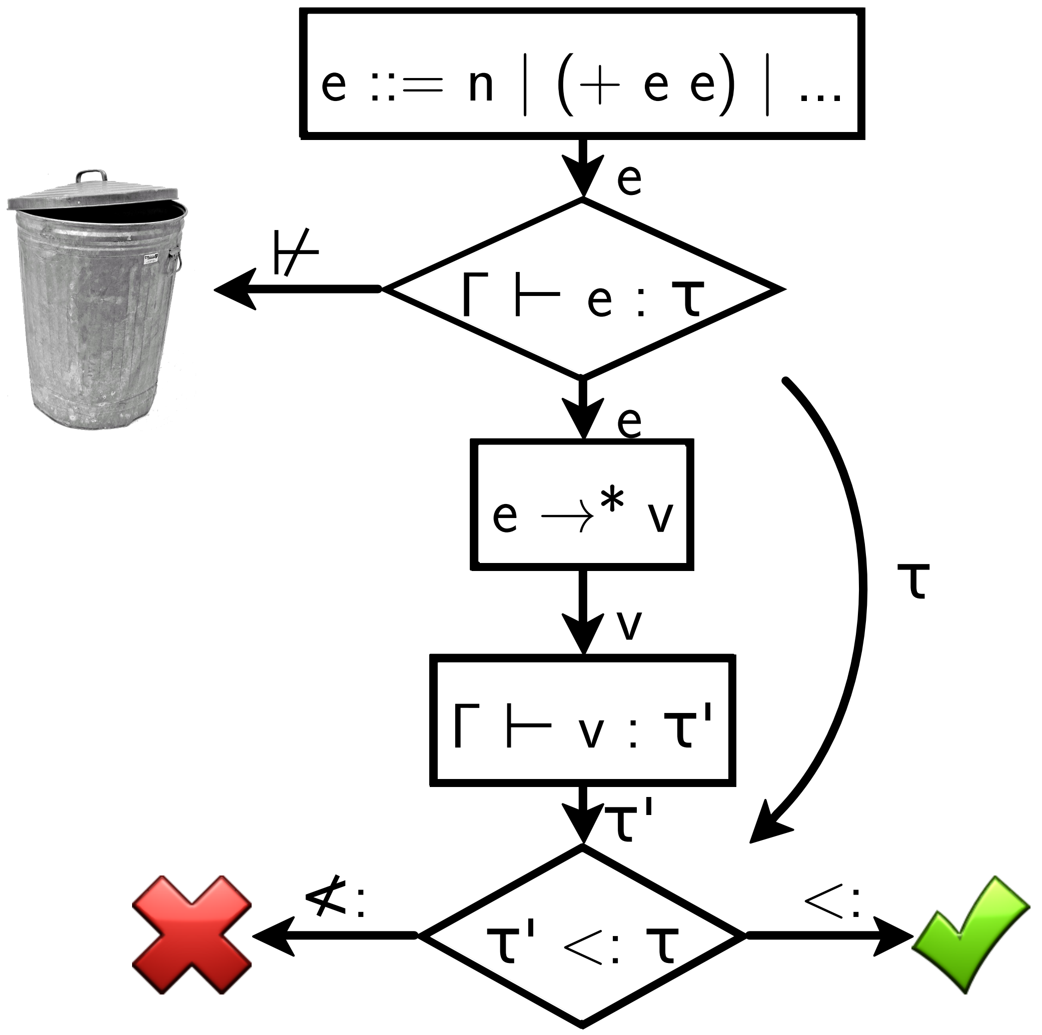
Testing Type Preservation



`(sinh 1.2535e-17)`

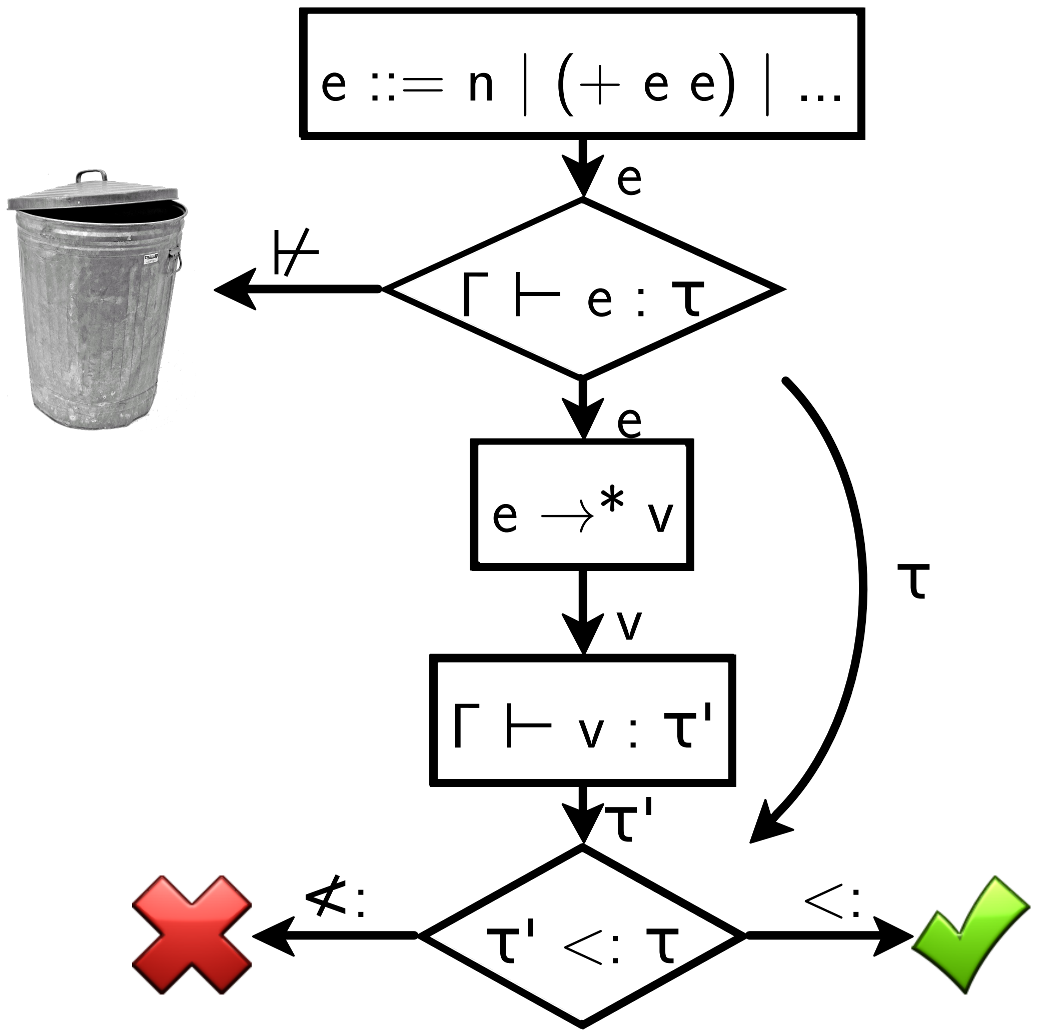
`Positive-Float`

Testing Type Preservation



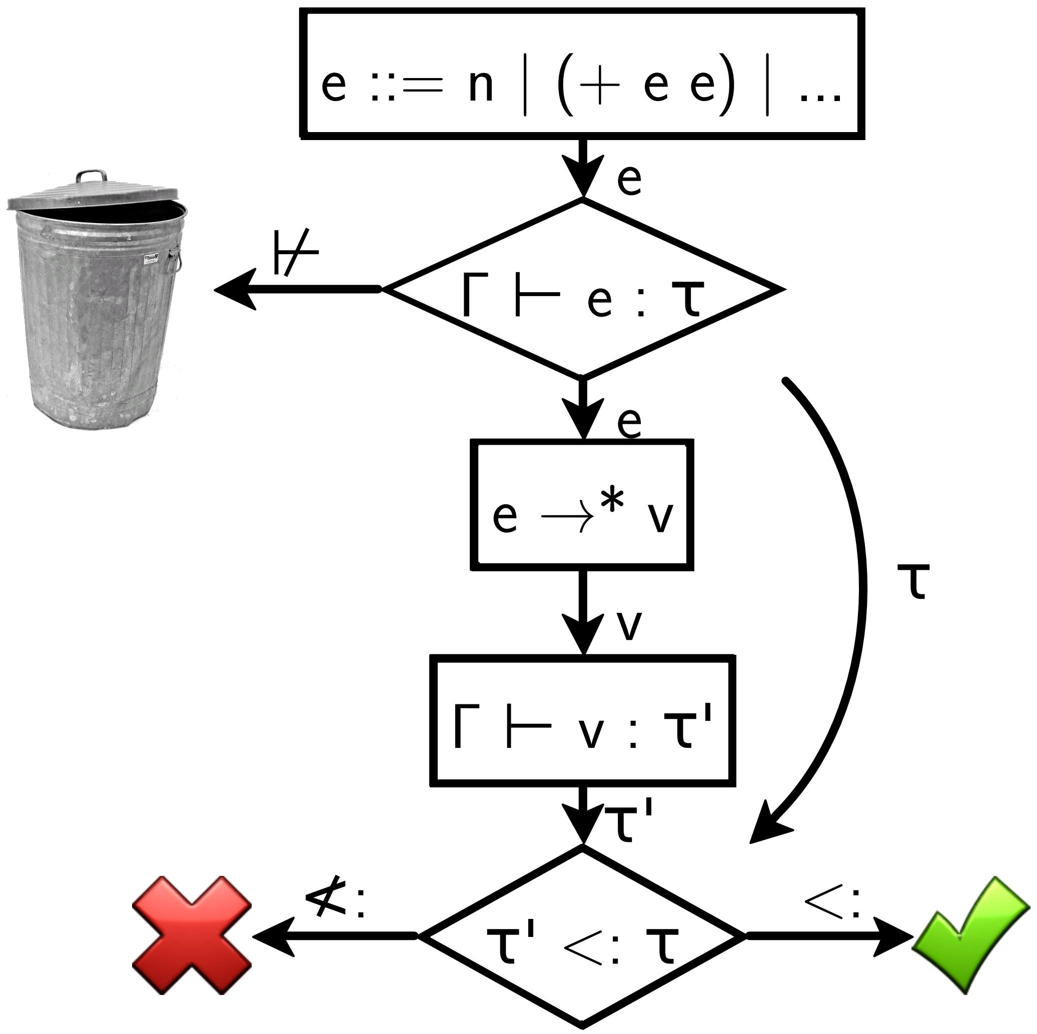
```
(sinh 1.2535e-17)

Positive-Float

0.0
```

Testing Type Preservation



```
(sinh 1.2535e-17)

Positive-Float

0.0

Float-Zero
```

Testing Type Preservation



(sinh 1.2535e-17)

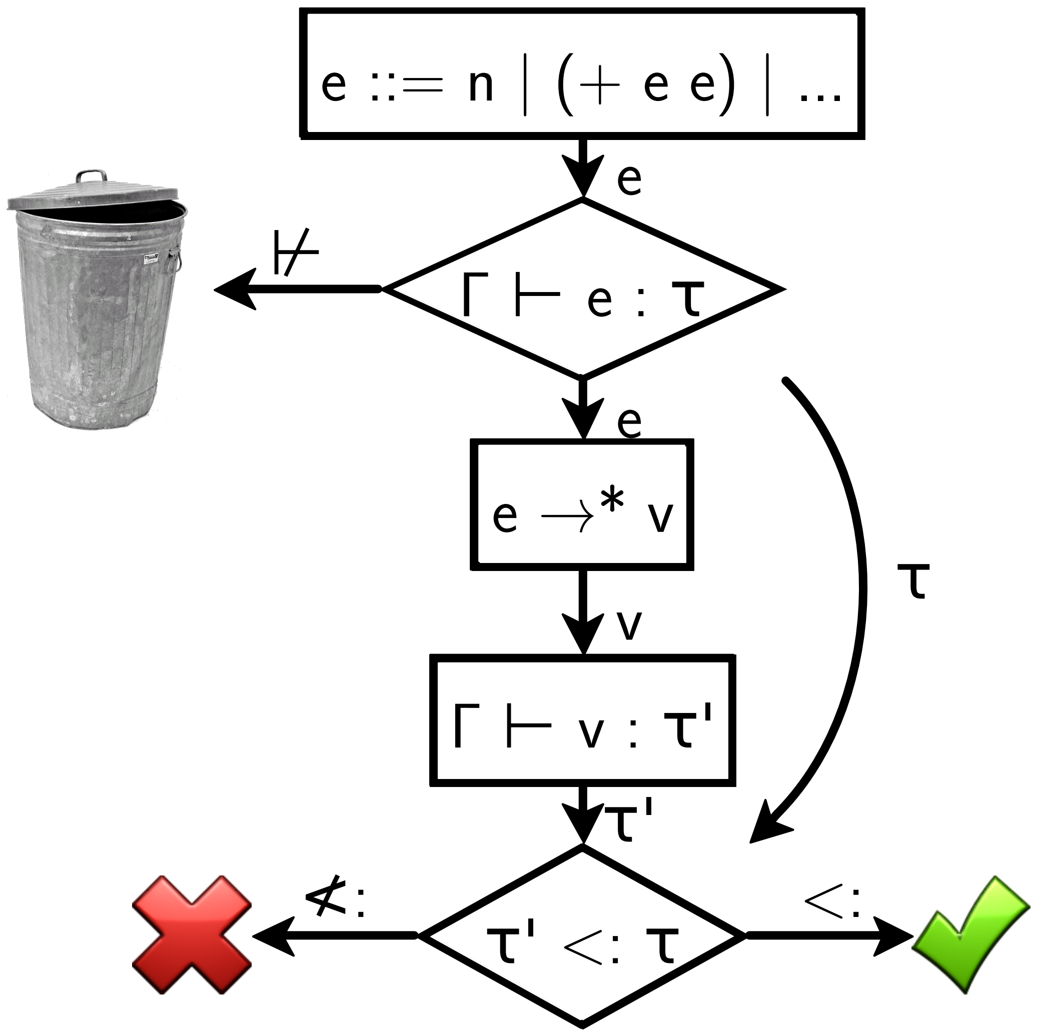
Positive-Float

0.0

Float-Zero

Float-Zero ≮: Positive-Float

Testing Type Preservation



```
(sinh 1.2535e-17)
```

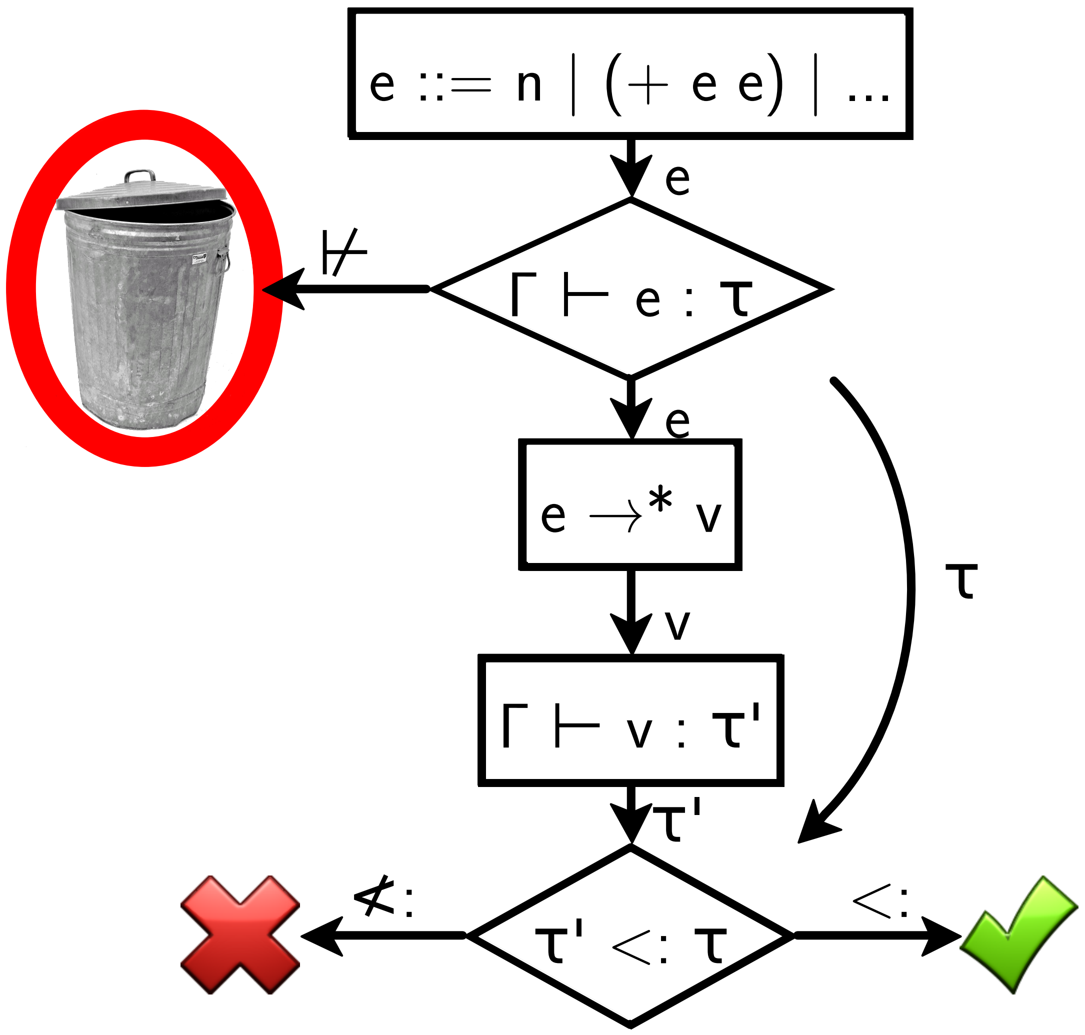
```
Positive-Float
```

```
0.0
```

```
Float-Zero
```

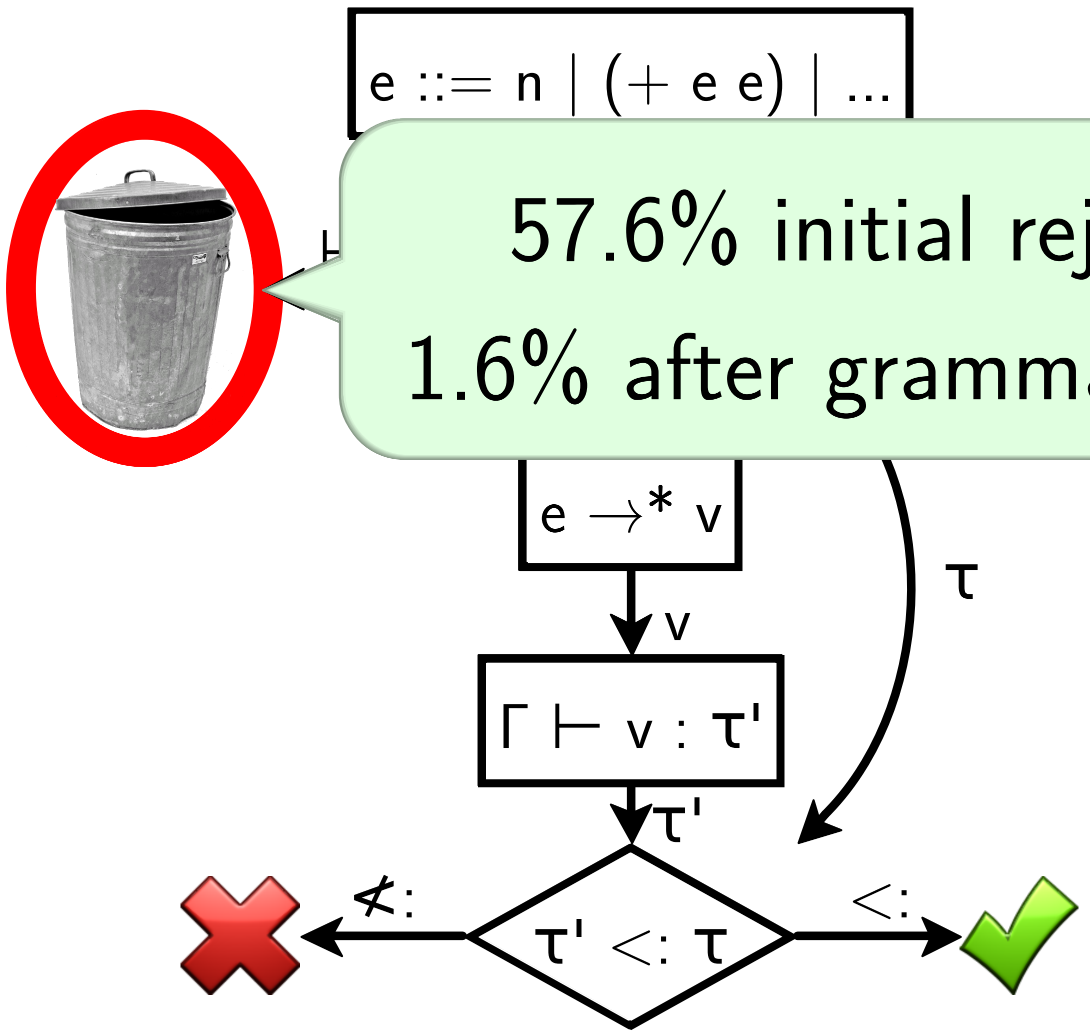
```
Float-Zero ≮ sitive-Float
```

Testing Type Preservation



```
(sinh 1.2535e-17)

Positive-Float

0.0

Float-Zero

Float-Zero ≮ sitive-Float
```

Testing Type Preservation



57.6% initial rejection rate

1.6% after grammar engineering

```
(sinh 1.2535e-17)
```

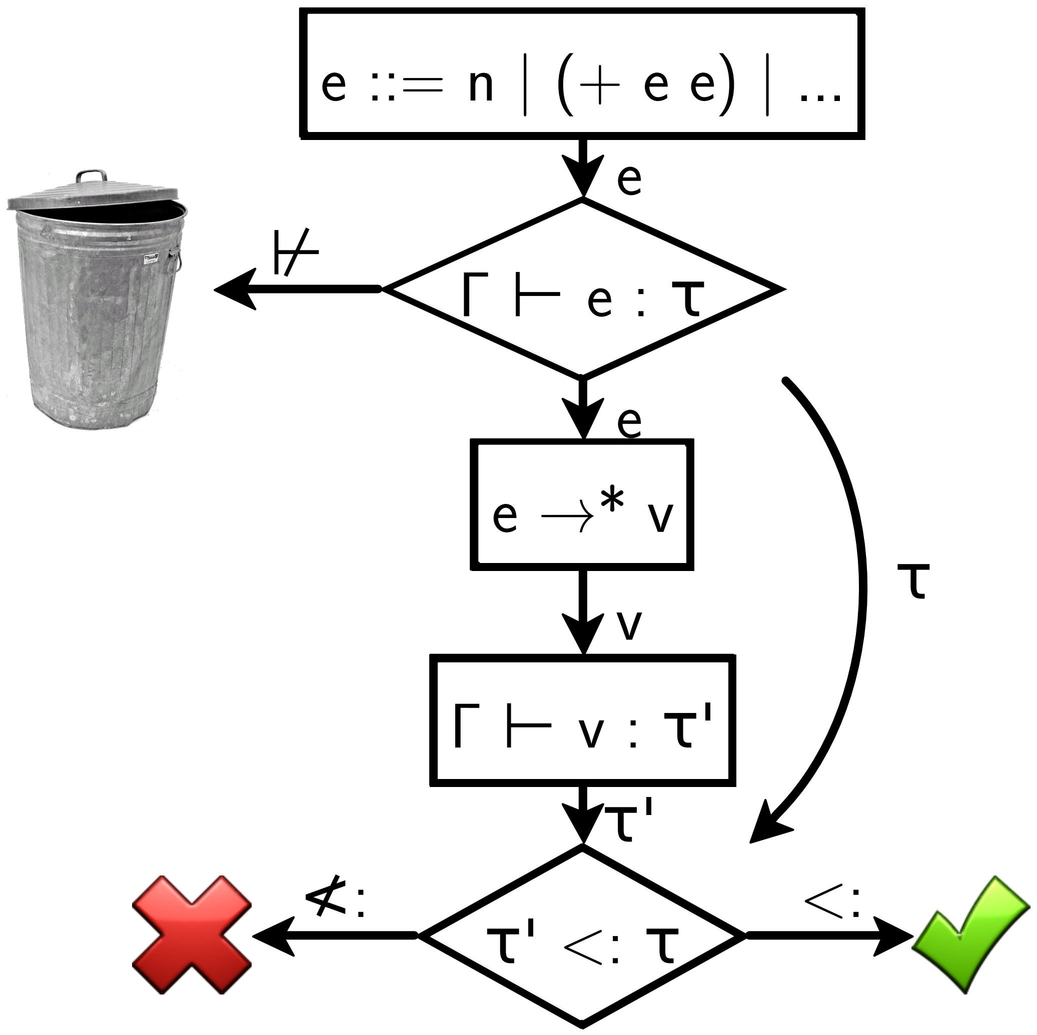
```
oat
```

```
0.0
```

```
Float-Zero
```

```
Float-Zero ≮ sitive-Float
```

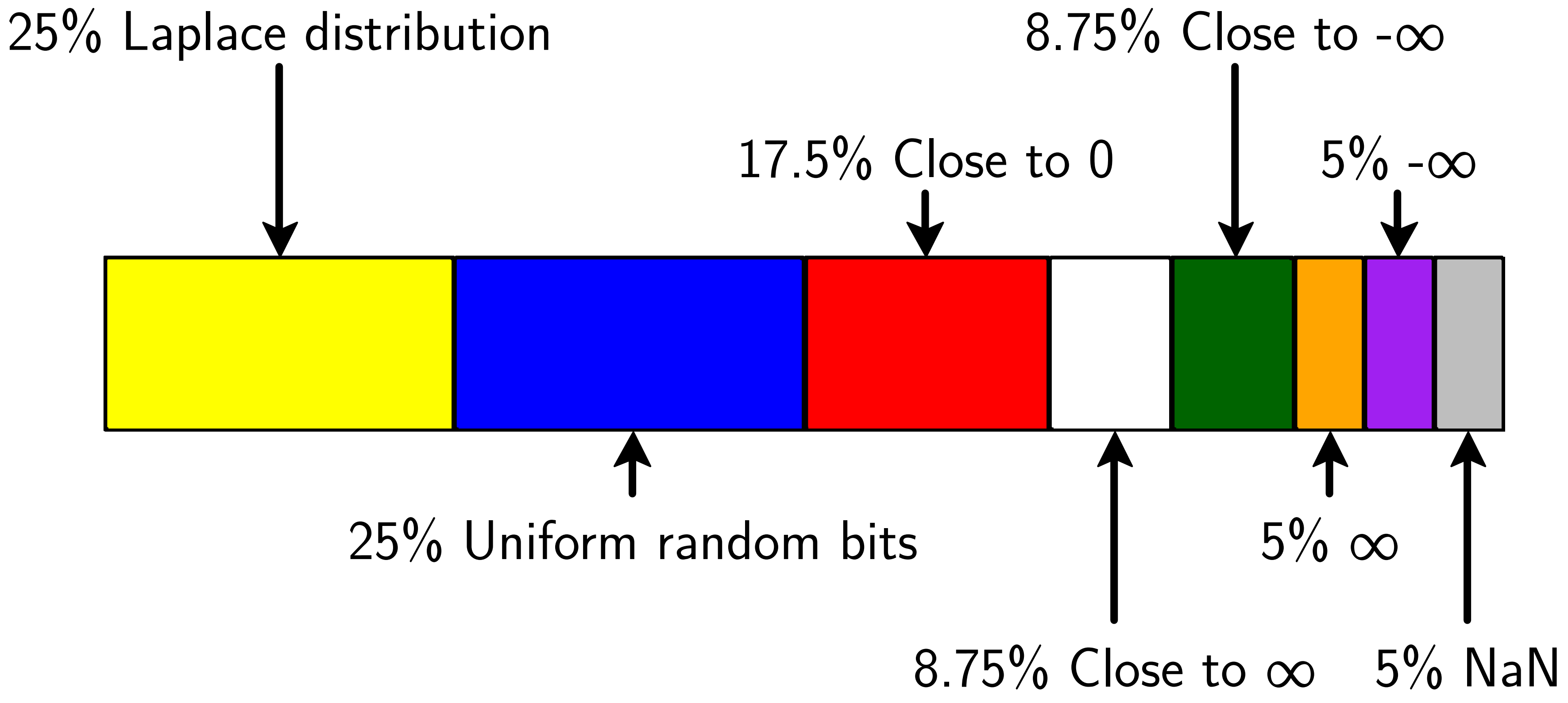
Testing Type Preservation



```
(sinh 1.2535e-17)

Positive-Float

0.0

Float-Zero

Float-Zero ≮: Positive-Float
```

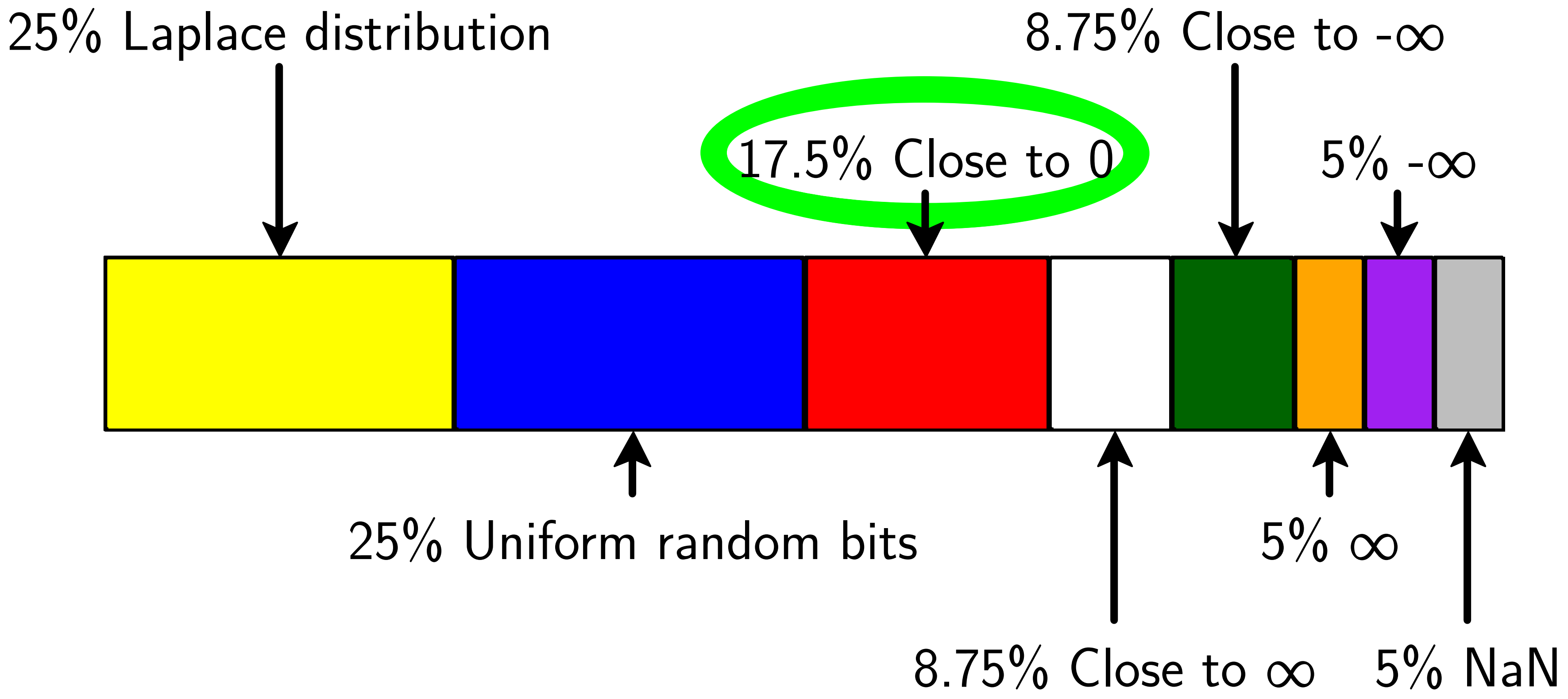
Testing Type Preservation

Random floating-point number generation



Testing Type Preservation

Random floating-point number generation



How well did that work?

Finding Bugs ✓

- Existing 10+ kloc test suite
  - Found bugs anyway
- Small random test cases
  - Smaller than user bug reports
  - Even without test case reduction

Confidence When Refactoring

- Fact: programs evolve over time
- Follow changes with random testing

Success stories

- NaN refactoring
- Optimizer rewrite

The Take-Away

Type environments have bugs too!

Random testing can help.

Redex makes random testing easy.

The Take-Away

Type environments have bugs too!

Random testing can help.

Redex makes random testing easy.

Thank You