$\Gamma_{\text{base}} \vdash e : \tau$
$\Gamma_{\text{base}} \vdash e : \tau$
\[ \Gamma_{\text{base}} \vdash e : \tau \]
Γ_{base} \vdash e : \tau
Γ_{base} \Gamma e : \tau
\[
\Gamma \vdash \text{base} : (\text{Listof } A) \rightarrow A
\]

\[
\Gamma \vdash \text{e} : \tau
\]

\[
\text{string-append} : \text{String } * \rightarrow \text{String}
\]
Typing the Numeric Tower

Vincent St-Amour¹, Sam Tobin-Hochstadt¹, Matthew Flatt², and Matthias Felleisen¹

¹ Northeastern University
{stamour, santh, matthias}@cs.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 notation.
/* 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, \ 
...
\[ \Gamma \vdash e : \tau \]

\[ + \]

Base
18% of Typed Racket bugs

\{ 
\begin{align*}
10.9\% & \text{ numeric } \Gamma_{\text{base}} \\
6.8\% & \text{ other } \Gamma_{\text{base}} 
\end{align*}
\}
18% of Typed Racket bugs

\[ \begin{cases} 
10.9\% \text{ numeric } \Gamma_{\text{base}} \\
6.8\% \text{ other } \Gamma_{\text{base}} 
\end{cases} \]
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

\( \tau_1 \cap \tau_2 \cap ... \)

\((: \sinh (\text{case} \rightarrow \)

[\text{Float-Zero} \rightarrow \text{Float-Zero}] 
[\text{Positive-Float} \rightarrow \text{Positive-Float}] 
[\text{Negative-Float} \rightarrow \text{Negative-Float}] 
...
))\)
A Type Environment Bug

\( \tau_1 \cap \tau_2 \cap \ldots \)

\( (: \text{ sinh } \ (\text{case}\to\)

[Float-Zero \rightarrow \text{Float-Zero}]

[Positive-Float \rightarrow \text{Positive-Float}]

[Negative-Float \rightarrow \text{Negative-Float}]

\ldots)) \)
A Type Environment Bug

\( \tau_1 \cap \tau_2 \cap \ldots \)

\( (: \text{sinh} (\text{case} \rightarrow \)

[Float-Zero \rightarrow Float-Zero]
[Positive-Float \rightarrow Positive-Float]
[Negative-Float \rightarrow Negative-Float]
\ldots)) \)

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

\[ (: \text{sinh} (\text{case}\to \]
\[
\begin{align*}
[\text{Float-Zero} & \to \text{Float-Zero}] \\
[\text{Nonnegative-Float} & \to \text{Nonnegative-Float}] \\
[\text{Nonpositive-Float} & \to \text{Nonpositive-Float}] \\
\ldots
\end{align*}
\]
\]

\( (\text{sinh} 1.2535e-17) \Rightarrow 0.0 : \text{Float-Zero} \)
A Type Environment Bug

(: * (case→
    ...
    ...
    [Positive-Real Positive-Real
     → Positive-Real]
    ...
))
A Type Environment Bug

\[(: \ast (\text{case} \rightarrow \ldots \text{[Positive-Real Positive-Real} \rightarrow \text{Positive-Real}] \ldots))\]

\((\ast \ 5/1241 \ 4.9406564584125e-324)\) \Rightarrow 0.0 : \text{Float-Zero}\]
A Type Environment Bug

\[
(: \ast (\text{case} \rightarrow \\
\ldots \\
[\text{Nonnegative-Real} \ \text{Nonnegative-Real} \\
\rightarrow \text{Nonnegative-Real}] \\
\ldots))
\]

\[
(* \ 5/1241 \ 4.9406564584125e-324) \\
\Rightarrow 0.0 : \text{Float-Zero}
\]
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
(define-language \(\texttt{\lambda v}\n\[
[\texttt{e (e e ...)} \\
(\texttt{if0 e e e}) \\
\texttt{x} \\
\texttt{v}] \\
[\texttt{v (\lambda (x ... e)}} \\
\texttt{number}] \\
[\texttt{x (variable-except \lambda if0))}]\)
(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 ...))
(define-language λv
  [e (e e ...)
    (if0 e e e)
    x
    v]
  [v (λ (x ...) e)
    number]
  [x (variable-except λ if0)])

(redux-check λv v
  (number? (term v)))

counterexample found after 4 attempts:
(λ () 1)
(define-language \(\lambda\)v
  \[v (\lambda (x ... ) e)\]
  \[v (\lambda (x ... ) e)\]
  \[x (variable-except \lambda if0)]\)

(redex-check \(\lambda\)v v
  (> (n-google-results
      (term v))
    20))

counterexample found
after 15 attempts:
(\(\lambda (x y) (+ (\lambda () 3) 2)\))
Testing Type Preservation

e ::= n | (+ e e) | ...

Generate arithmetic expressions
Testing Type Preservation

e ::= n | (+ e e) | ...

Typecheck using Typed Racket
Testing Type Preservation

\[ e ::= n \mid (+\ e\ e) \mid \ldots \]

\[ \Gamma \vdash e : \tau \]

Evaluate using Typed Racket
Testing Type Preservation

e ::= n | (+ e e) | ...

\[ \Gamma \vdash e : \tau \]

\[ e \rightarrow^* v \]

\[ \Gamma \vdash v : \tau' \]

Typecheck the result
**Testing Type Preservation**

\[ e ::= n \mid (+ e e) \mid \ldots \]

\[ \Gamma \vdash e : \tau \]

\[ e \rightarrow^* v \]

\[ \Gamma \vdash v : \tau' \]

\[ \tau' \preceq_\prec \tau \]

Check consistency
Testing Type Preservation

\[ e ::= n \mid (+ e e) \mid ... \]

\[ \Gamma \vdash e : \tau \]

\[ e \rightarrow^* v \]

\[ \Gamma \vdash v : \tau' \]

\[ \tau' <: \tau \]

\[ (\sinh 1.2535e-17) \]
Testing Type Preservation

```
e ::= n | (+ e e) | ...
```

\[ \Gamma \vdash e : \tau \]

\[ e \rightarrow^* v \]

\[ \Gamma \vdash v : \tau' \]

\[ \tau' <: \tau \]

\[ (\sinh 1.2535e-17) \]

Positive-Float
Testing Type Preservation

\[
e ::= n \mid (+ \ e \ e) \mid ...
\]

\[
\Gamma \vdash e : \tau
\]

\[
e \rightarrow^* v
\]

\[
\Gamma \vdash v : \tau'
\]

\[
\tau' <: \tau
\]

\[
(sinh 1.2535e-17)
\]

Positive-Float

0.0
Testing Type Preservation

\[ e ::= n \mid (+ \, e \, e) \mid \ldots \]

\[ \Gamma \vdash e : \tau \]

\[ e \rightarrow^* v \]

\[ \Gamma \vdash v : \tau' \]

\[ \tau' <: \tau \]

\( \sinh 1.2535e-17 \)

Positive-Float

0.0

Float-Zero
Testing Type Preservation

\[ e ::= n \mid (+ e e) \mid \ldots \]

\[ \Gamma \vdash e : \tau \]

\[ e \rightarrow^* v \]

\[ \Gamma \vdash v : \tau' \]

\[ \tau' \ll : \tau \]

\[ (\sinh 1.2535e-17) \]

Positive-Float

0.0

Float-Zero

Float-Zero \ll : Positive-Float
Testing Type Preservation

\[ e ::= n \mid (+ e e) \mid \ldots \]

\[
\Gamma \vdash e : \tau
\]

\[
e \rightarrow^* v
\]

\[
\Gamma \vdash v : \tau'
\]

\[
\tau' \preceq \tau
\]

\[
\tau' \npreceq \tau
\]

\[
(sinh 1.2535e-17)
\]

Positive-Float

0.0

Float-Zero

Float-Zero

Positive-Float
Testing Type Preservation

e ::= n \mid (+ e e) \mid ...

Γ ⊢ e : τ

Γ ⊢ v : τ'

τ' <: τ

(sinh 1.2535e-17)

Positive-Float

0.0

Float-Zero

Float-Zero < Positive-Float

Positive-Float
Testing Type Preservation

e ::= n | (+ e e) | ...

\[ e \rightarrow^* v \]

\[ \Gamma \vdash v : \tau' \]

\( \tau' \llcorner \)

\( (\sinh 1.2535e-17) \)

57.6% initial rejection rate
1.6% after grammar engineering

Positive-Float

0.0

Float-Zero

\( \llcorner \)

Float-Zero

Sitive-Floa
Testing Type Preservation

e ::= n | (+ e e) | ...

\[ \Gamma \vdash e : \tau \]

\[ e \rightarrow^* v \]

\[ \Gamma \vdash v : \tau' \]

\[ \tau' <: \tau \]

\[ (\sinh 1.2535e-17) \]

Positive-Float

0.0

Float-Zero

Float-Zero

Positive-Float
Testing Type Preservation

Random floating-point number generation

- 25% Laplace distribution
- 8.75% Close to \(-\infty\)
- 17.5% Close to 0
- 5% \(-\infty\)
- 5% \(+\infty\)
- 25% Uniform random bits
- 8.75% Close to \(+\infty\)
- 5% NaN
Testing Type Preservation

Random floating-point number generation

- 25% Laplace distribution
- 25% Uniform random bits
- 17.5% Close to 0
- 8.75% Close to $\infty$
- 5% Close to $-\infty$
- 5% NaN
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