Usable Language-Based Verification

Niki Vazou
University of Maryland

Formal Verification

Prove Properties of Software

Formal Verification

Prove Properties of Real Software

Formal Verification

Prove Properties of Real Software

“A plane will not crash”

Formal Verification

Prove Properties of Real Software

"Your passwords are SAFE"

Formal Verification, in Practice

Prove Properties ~~of Real Software~~ in Verification Specific Tools

Verification Specific Tools

```
Lemma plus_Snm_nSm : forall n m, S n + m = n + S m.
intros.
simpl in |- *.
rewrite (plus_comm n m).
rewrite (plus_comm n (S m)).
trivial with arith.
Qed.
```

"Plus is indeed associative!"

Verification Specific Tools

Difficult to use for Real Programs

Difficult to install

Special Syntax

No Compiler Optimizations

No Parallelism

Very Few Users!

My goal:

Verification of Real Programs

Verification of Real Programs

My approach:

Turn a Language into a Verifier

Reuse language's syntax, runtime, and users!

Verification of Real Programs

My approach:

Turn a Language into a Verifier

Target Language: Haskell

Haskell

+

Refinement Types

=

Haskell

```
take :: [a] -> Int -> [a]
```

```
> take [1,2,3] 2
> [1,2]
```

Haskell

```
take :: [a] -> Int -> [a]
```

```
> take [1,2,3] 500
> ???
```

Refinement Types

```
take :: xs:[a] -> {i:Int | i < len xs} -> [a]
```


```
take :: xs:[a] -> {i:Int | i < len xs} -> [a]
```

```
> take [1,2,3] 500
> Type Error!
```

Use Haskell's syntax, runtime, and **users**!

Used in **Industry**: to speed up code

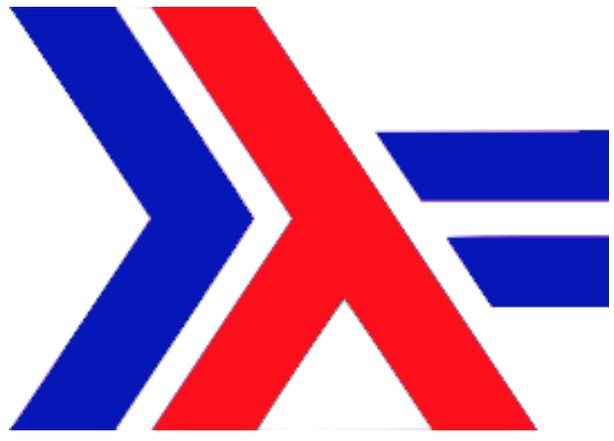
Used in **Education**: in Haskell courses!

Haskell

Functional Programing Language

Based on

Strong Typing + λ-calculus



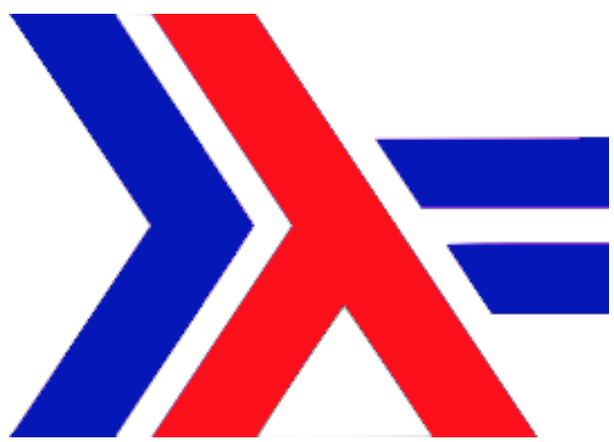
Haskell

Practical Programing Language

Fast Memory Manipulation

with C wrappers

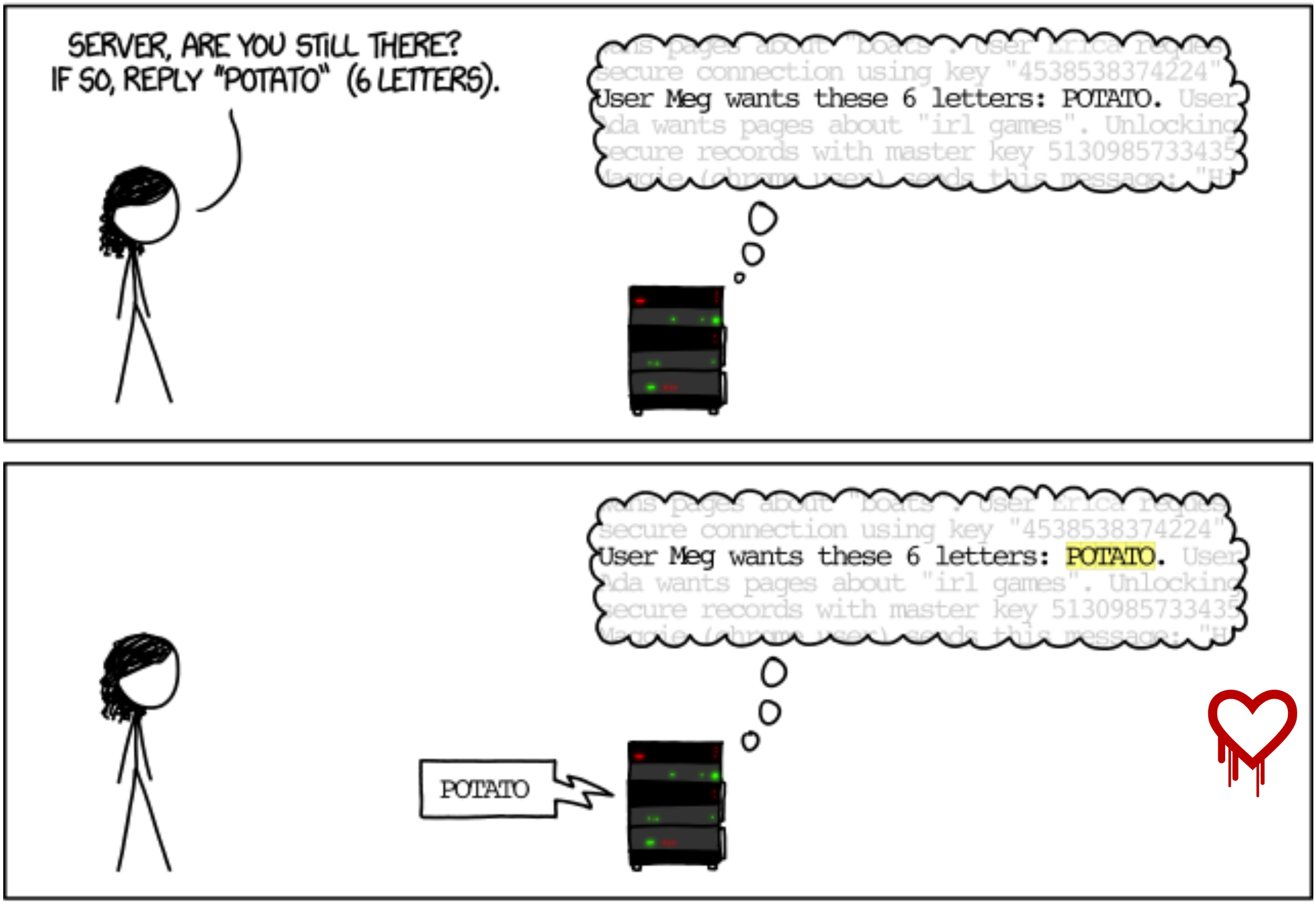
Allowing overread bugs



The Heartbleed Bug

Buffer overread in OpenSSL. 2015

HOW THE HEARTBLEED BUG WORKS:



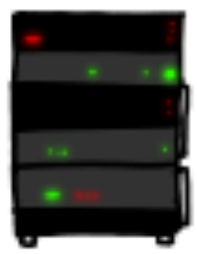
SERVER, ARE YOU STILL THERE? IF SO, REPLY "BIRD" (4 LETTERS).

User Olivia from London wants pages about "man bees in car why". Note: Files for IP 375.381. 283.17 are in /tmp/files-3843. User Meg wants these 4 letters: BIRD. There are currently 346 connections open. User Brendan uploaded the file selfie.jpg (contents: 834ba962e2cab9ff89bd3bff8

HMM...

User Olivia from London wants pages about "man bees in car why". Note: Files for IP 375.381. 283.17 are in /tmp/files-3843. User Meg wants these 4 letters: BIRD. There are currently 346 connections open. User Brendan uploaded the file selfie.jpg (contents: 834ba962e2cab9ff89bd3bff8

BIRD



SERVER, ARE YOU STILL THERE?

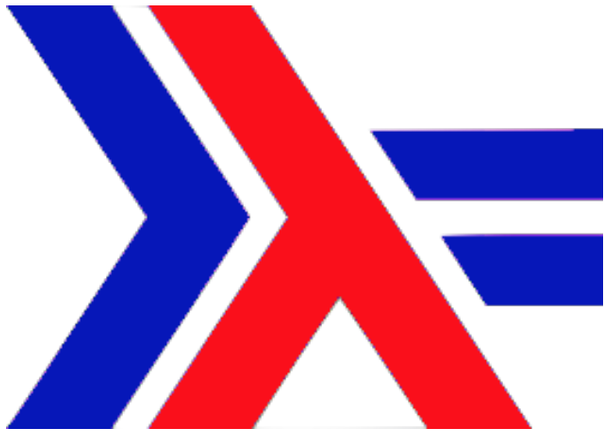
User Meg wants these 500 letters: HAT. Lucas

SERVER, ARE YOU STILL THERE? IF SO, REPLY "HAT" (500 LETTERS).

User Meg wants these 500 letters: HAT.

HAT. Lucas requests the "missed connections" page. Eve (administrator) wants to set server's master key to "14835038534". Isabel wants pages about "snakes but not too long". User Karen wants to change account password to "

in



```
module Data.Text where
take :: t:Text -> i:Int -> Text
```

```
> take "hat" 500
> *** Exception: Out Of Bounds!
```

Runtime Checks

```
take :: t:Text -> i:Int -> Text
take t i | i < len t
  = Unsafe.take t i
take t i
  = error “Out Of Bounds!”
```

Safe, but slow

No Checks

```
take :: t:Text -> i:Int -> Text
take t i | i < len t
  = Unsafe.take t i
take t i
  = error "Out Of Bounds!"
```

Fast, but unsafe!

No Checks

```
take :: t:Text -> i:Int -> Text
take t i | i < len t
  = Unsafe.take t i
take t i
  = error “Out Of Bounds!”
```

```
> take "hat" 500
> “hat\58456\2594\SOH\NUL…
```

Overread

Static Checks

```
take :: t:Text -> i:Int -> Text
take t i | i < len t
  = Unsafe.take t i
take t i
  = error “Out Of Bounds!”
```

Static Checks

```
take :: t:Text -> i:{i < len t} -> Text
take t i | i < len t
  = Unsafe.take t i
take t i
  = error “Out Of Bounds!”
```

Static Checks

```
take :: t:Text->i:{i < len t}->Text
take t i | i < len t
  = Unsafe.take t i
take t i
  = error "Out Of Bounds!"
```

Static Checks

```
take :: t:Text->i:{i < len t}->Text
take t i
 = Unsafe.take t i
```

Static Checks

```
take :: t:Text->i:{i < len t}->Text
take t i
 = Unsafe.take t i
```

```
> take "hat" 500
```

Type Error

Refinement Types

Checks valid arguments, under facts.

Checks valid arguments, under facts.

```
take :: t:Text->{v|v < len t}->Text

heartbleed = let x = "hat"
             in  take x 500
```

```
len x = 3 => v = 500 => v < len x
```

Checks valid arguments, under facts.

```
take :: t:Text->{v | v < len t}->Text

heartbleed = let x = "hat"
             in  take x 500
```

```
len x = 3 => v = 500 => v < len x
```

Checks valid arguments, under facts.

```
take :: t:Text->{v | v < len t}->Text

heartbleed = let x = "hat"
             in take x 500
```

```
len x = 3 => v = 500 => v < len x
```

Checks valid arguments, under facts.

```
take :: t:Text->{v | v < len t}->Text

heartbleed = let x = "hat"
             in  take x 500
```

```
len x = 3 => v = 500 => v < len x
```

Checks valid arguments, under facts.

```
take :: t:Text->{v | v < len t}->Text
heartbleed = let x = "hat"
             in  take x 500
```

```
len x = 3 => v = 500 => v < len x
```

Checks valid arguments, under facts.

```
take :: t:Text->{v | v < len t}->Text

heartbleed = let x = "hat"
             in  take x 500
```

```
len x = 3 => v = 500 => v < len x
```

Checks valid arguments, under facts.

```
take :: t:Text->{v | v < len t}->Text

heartbleed = let x = "hat"
             in  take x 500
```

SMT-query

```
len x = 3 => v = 500 => v < len x
```

Checks valid arguments, under facts.

```
take :: t:Text->{v|v < len t}->Text

heartbleed = let x = "hat"
             in  take x 500
```

SMT-invalid

`len x = 3 => v = 500 => v < len x`

Checks valid arguments, under facts.

```
take :: t:Text->{v|v < len t}->Text

heartbleed = let x = "hat"
             in  take x 500
```

Checker reports **Error**

```
len x = 3 => v = 500 => v < len x
```

Checks valid arguments, under facts.

```
take :: t:Text->{v | v < len t}->Text

heartbleed = let x = "hat"
             in  take x 500
```

Checker reports **Error**

```
len x = 3 => v = 500 => v < len x
```

Checks valid arguments, under facts.

```
take :: t:Text->{v | v < len t}->Text

heartbleed = let x = "hat"
             in  take x 2
```

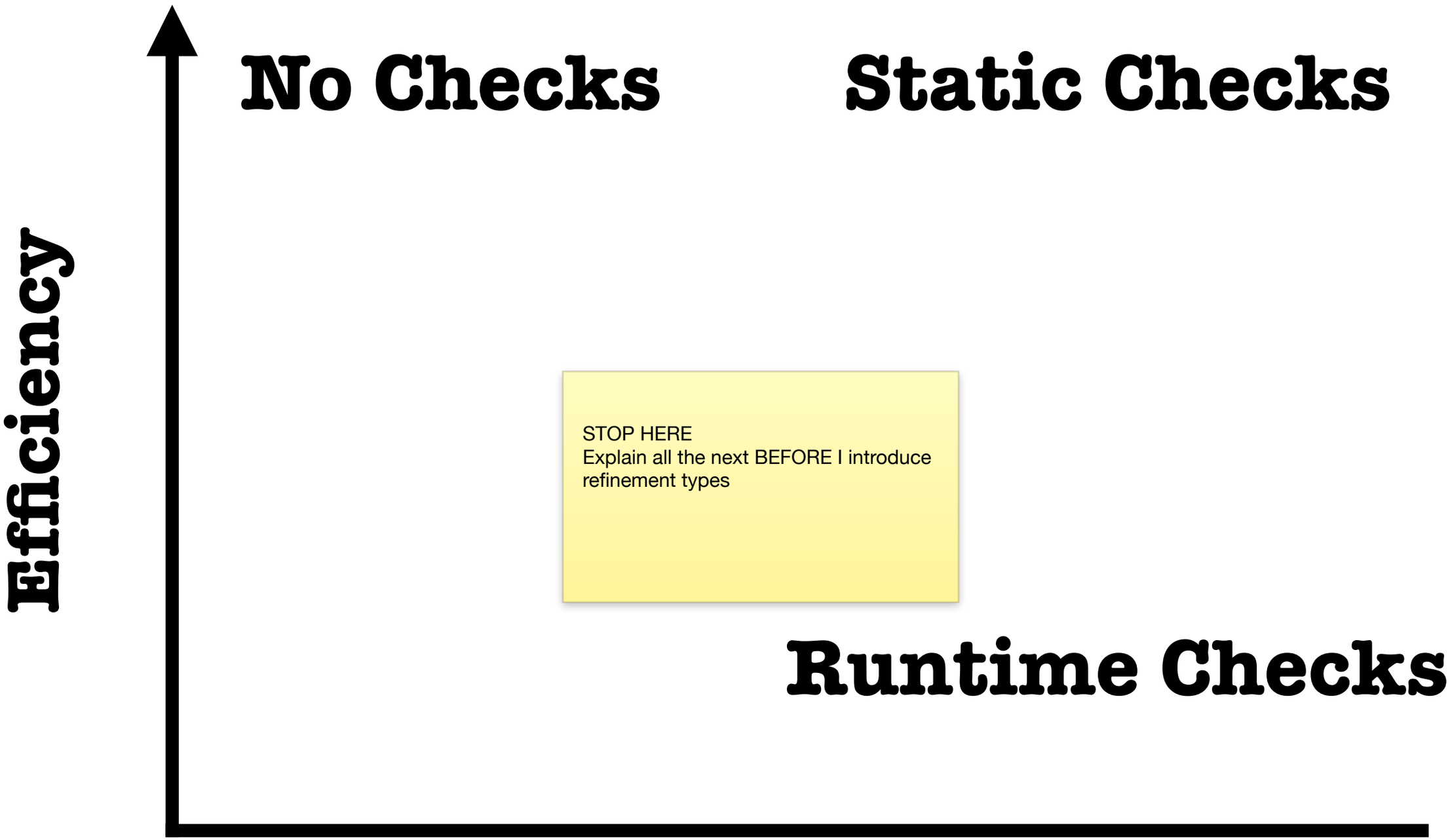
Checker reports **OK**

SMT-valid

```
len x = 3 => v = 2 => v < len x
```

Checks valid arguments, under facts.

Static Checks



STOP HERE
Explain all the next BEFORE I introduce refinement types

Static Checks

Safe & Fast Code!

Static Checks

Safe & Fast Code!

Application: Speed up Parsing

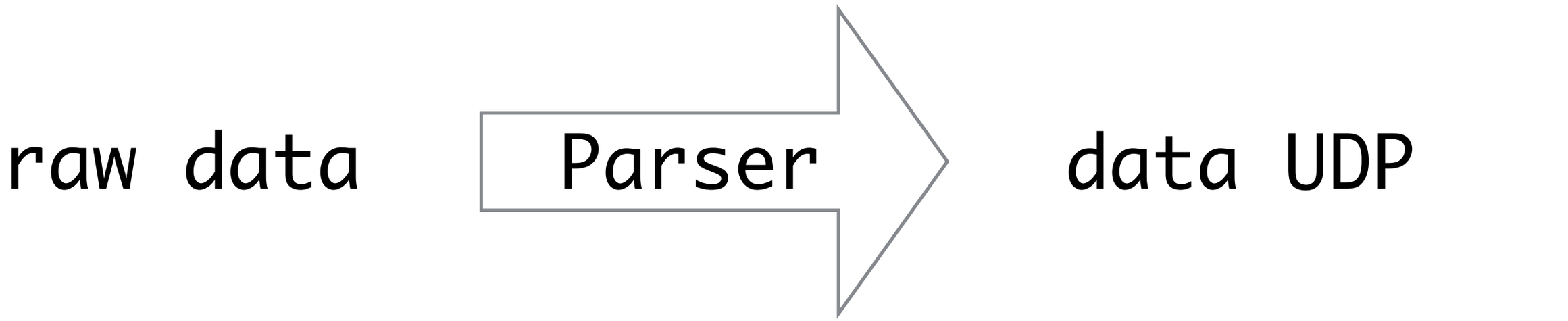
Application: Speed up Parsing

Simplified application of Liquid Haskell

by Gabriel Gonzalez

Application: Speed up Parsing

UDP:User Datagram Protocol



Application: Speed up Parsing

```
data UDP = UDP
    { udpSrcePort :: Text   -- 2 chars
    , udpDestPort :: Text   -- 2 chars
    , udpLength   :: Text   -- 2 chars
    , udpChecksum :: Text   -- 2 chars
    }
```

Application: Speed up Parsing

```
udpP :: Text -> UDP
udpP bs =
  let (udp1, bs1) = splitAt 2 bs
  let (udp2, bs2) = splitAt 2 bs1
  let (udp3, bs3) = splitAt 2 bs2
  let (udp4, bs4) = splitAt 2 bs3
  in UDP (udp1 upd2 udp3 upd4)
```

Safe but Slow (4 runtime checks)

Solution: Merge checks

Application: Speed up Parsing

```haskell
udpP :: Text -> Maybe UDP
udpP bs =
  if length bs >= 8 then
    let (udp1, bs1) = US.splitAt 2 bs
    let (udp2, bs2) = US.splitAt 2 bs1
    let (udp3, bs3) = US.splitAt 2 bs2
    let (udp4, bs4) = US.splitAt 2 bs3
    in Just (UDP udp1 upd2 udp3 upd4)
  else Nothing
```

Safe and Fast (1 runtime check!)

Application: Speed up Parsing

```
udpP :: Text -> Maybe UDP
udpP bs =
  if length bs >= 8 then
    let (udp1, bs1) = US.splitAt 2 bs
    let (udp2, bs2) = US.splitAt 2 bs1
    let (udp3, bs3) = US.splitAt 4 bs2
    let (udp4, bs4) = US.splitAt 2 bs3
    in Just (UDP udp1 upd2 udp3 upd4)
  else Nothing
```

Safe and Fast, bur error prone

Application: Speed up Parsing

```
udpP :: Text -> Maybe UDP
udpP bs =
  if length bs >= 8 then
    let (udp1, bs1) = US.splitAt 2 bs
    in Just (UDP udp1 upd2 udp3 upd4)
  else Nothing
```

```
splitAt :: i:Int -> t:{i < len t} ->
 (tl:{i = len tl}, tr:{len tr = len t - i})
```

Enforce Static Checks!

Application: Speed up Parsing

Error

```
udpP :: Text -> Maybe UDP
udpP bs =
  if length bs >= 8 then
    let (udp1, bs1) = US.splitAt 2 bs0
    let (udp2, bs2) = US.splitAt 2 bs1
    let (udp3, bs3) = US.splitAt 4 bs2
    let (udp4, bs4) = US.splitAt 2 bs3
    in Just (UDP udp1 upd2 udp3 upd4)
  else Nothing
```

Enforce Static Checks!

Application: Speed up Parsing

OK

```
udpP :: Text -> Maybe UDP
udpP bs =
  if length bs >= 8 then
    let (udp1, bs1) = US.splitAt 2 bs
    let (udp2, bs2) = US.splitAt 2 bs
    let (udp3, bs3) = US.splitAt 2 bs
    let (udp4, bs4) = US.splitAt 2 bs
    in Just (UDP udp1 upd2 udp3 upd4)
  else Nothing
```

Haskell code!
- syntax
- compiler

No proofs!
- SMT automated!

Provably Correct & Faster (x6) Code!

Application: Speed up Parsing

```
udpP :: Text -> Maybe UDP
udpP bs =
  if length bs >= 8 then
    let (udp1, bs1) = US.splitAt 2 bs
    let (udp2, bs2) = US.splitAt 2 bs1
    let (udp3, bs3) = US.splitAt 2 bs2
    let (udp4, bs4) = US.splitAt 2 bs3
    in Just (UDP udp1 upd2 udp3 upd4)
 else Nothing
```

Provably Correct & Faster (x6) Code!

Haskell Code: Verification with SMT

Provably Correct & Faster (x6) Code!

Haskell Code: Verification with SMT

Verification with SMT

If p is safe indexing

```
{t:Text | i < len t }
```

Verification with SMT

If p from decidable theories

$$\{t : a \mid p\}$$

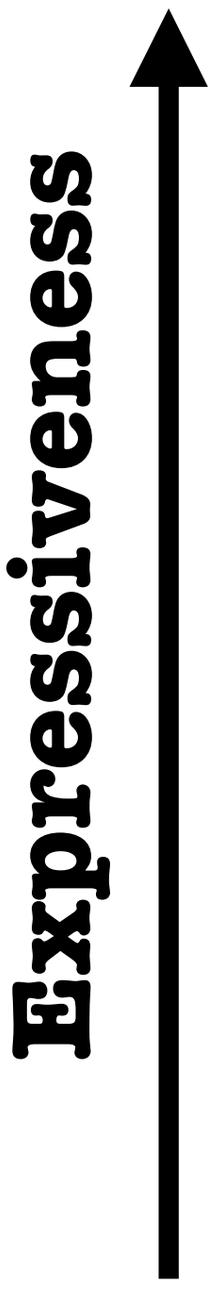
Boolean Logic

(QF) Linear Arithmetic

Uninterpreted Functions ...

What other properties can we express?

Verification with SMT



Critical Properties (Goal):
"The plane will not crash!"

Math Theorems (POPL'18):
"If f(x) < f(x+1), then f is monotonic"

Safe Indexing (Haskell'14):
"Your passwords are safe!"

Verification with SMT

Refinement Reflection, POPL'18

for Theorem Proving in Haskell!

"If `f(x) < f(x+1)`, then `f` is monotonic"

Fibonacci in Haskell

```
fib :: i:{Int|0≤i} -> {v:Int|0<v∧i≤v}
fib i
  | i≤1          = 1
  | otherwise = fib (i-1) + fib (i-2)
```

Fibonacci in Haskell

```
fib :: i:{Int|0≤i} -> {v:Int|0<v∧i≤v}
fib i
  | i≤1          = 1
  | otherwise = fib (i-1) + fib (i-2)
```

How to express **theorems** about functions?

```
\forall i. 0 ≤ i => fib i ≤ fib (i+1)
```

How to express **theorems** about functions?

Step 1: Definition

In SMT `fib` is "Uninterpreted Function"

```
\forall i j. i = j => fib i = fib j
```

How to connect logic fib with target fib?

How to connect logic fib with target fib?

```
fib :: i:{Int|0≤i} -> {v:Int|0<v∧i≤v}
fib i
  | i≤1          = 1
  | otherwise = fib (i-1) + fib (i-2)
```

Not Decidable!

SMT Axiom

```
\forall i.
      if i≤1 then fib i = 1
      else fib i = fib (i-1) + fib (i-2)
```

How to connect logic fib with target fib?

```
fib :: i:{Int|0≤i} -> {v:Int|0<v∧i≤v}
fib i
  | i≤1          = 1
  | otherwise = fib (i-1) + fib (i-2)
```

Refinement Reflection

```
fib :: i:{Int|0≤i} -> {v:Int| v=fib i ∧
    if i≤1 then fib i = 1
    else fib i = fib (i-1) + fib (i-2)
}
```

Refinement Reflection

Step 1: Definition

Step 2: Reflection

```
fib :: i:{Int|0≤i} -> {v:Int| v=fib i ∧
    if i≤1 then fib i = 1
    else fib i = fib (i-1) + fib (i-2)
}
```

Refinement Reflection

Step 1: Definition

Step 2: Reflection

```
fib :: i:{Int|0≤i} -> {v:Int| v=fib i ∧
    if i≤1 then fib i = 1
    else fib i = fib (i-1) + fib (i-2)
}
```

Step 3: Application

```
fib 0 :: {v:Int| v=fib 0 ∧ fib 0 = 1}
```

Application is Type Level Computation

```
fib 0
```

```
fib 0 = 1
```

Application

```
fib 0
```

```
fib 1
```

```
fib 2
```

```
fib i
```

Type Level Computation

```
fib 0 = 1
```

```
fib 1 = 1
```

```
fib 2 = fib 1 + fib 0
```

```
?
```

?
```
if i≤1 then fib i = 1
else fib i = fib (i-1) + fib (i-2)
```

Application　Type Level Computation

```
fib 0
```

```
fib 0 = 1
```

```
fib 1
```

```
fib 1 = 1
```

```
fib 2
```

```
fib 2 = fib 1 + fib 0
```

```
if 1<i then
```

```
fib i
```

```
fib i = fib (i-1) + fib (i-2)
```

```
if i≤1 then fib i = 1
else fib i = fib (i-1) + fib (i-2)
```

Application

```
fib 0
```

```
fib 1
```

```
fib 2
```

```
if 1<i then
```

```
fib i
```

```
fib (i+1)
```

Type Level Computation

```
fib 0 = 1
```

```
fib 1 = 1
```

```
fib 2 = fib 1 + fib 0
```

```
fib i = fib (i-1) + fib (i-2)
```

```
fib (i+1) = fib i + fib (i-1)
```

Theorem Proving

```
fibUp :: i:Nat -> {fib i ≤ fib (i+1)}
fibUp i
 | i == 0
 =    fib 0 <. fib 1
 *** QED
 | i == 1
 =    fib 1 <=. fib 1 + fib 0 <=. fib 2
 *** QED
 | otherwise
 =    fib i
 <=. fib i + fib (i-1)
 <=. fib (i+1)
 *** QED
```

Reflection for Theorem Proving

Theorems are refinement types.

Proofs are functions.

Check that functions prove theorems.

Proofs are functions.

```
fibUp :: i:Nat -> {fib i ≤ fib (i+1)}
```

Proofs are functions.

```
fibUp :: i:Nat -> {fib i ≤ fib (i+1)}
```

Let's call them!

```
fibUp 4 :: {fib 4 ≤ fib 5}
fibUp i :: {fib i ≤ fib (i+1)}
fibUp (j-1) :: {fib (j-1) ≤ fib j}
```

Proofs are functions. Let's call them!

```
fibMono :: i:Nat -> j:{Nat| i < j}
        -> {fib i ≤ fib j}

fibMono i j
 | i + 1 == j
 =    fib i
 <=. fib (i+1) ? fibUp i
 ==. fib j
 *** QED
 | otherwise
 =    fib i
 <=. fib (j-1) ? fibMono i (j-1)
 <=. fib j     ? fibUp (j-1)
 *** QED
```

Proofs are functions. Let's call them!

```
fibMono :: i:Nat -> j:{Nat| i < j}
        -> {fib i ≤ fib j}

fibMono i j
 | i + 1 == j
 =    fib i
 <=. fib (i+1) ? fibUp i
 ==. fib j
 *** QED
 | otherwise
 =    fib i
 <=. fib (j-1) ? fibMono i (j-1)
 <=. fib j     ? fibUp (j-1)
 *** QED
```

Proofs are functions. Let's abstract them!

```
fibMono :: i:Nat -> j:{Nat| i < j}
        -> fib:(Nat -> Int)
        -> (k:Nat -> {fib k ≤ fib (k+1)})
        -> {fib i ≤ fib j}

fibMono i j fib fibUp
  | i + 1 == j
  =    fib i
  <=. fib (i+1) ? fibUp i
  ==. fib j
  *** QED
  | otherwise
  =    fib i
  <=. fib (j-1) ? fibMono i (j-1)
  <=. fib j     ? fibUp (j-1)
  *** QED
```

Refinement Reflection for **Expressiveness**

Refinement Reflection for **Expressiveness**

Application (Haskell'17):
String Matching Parallelization

Refinement Reflection for **Expressiveness**

Idea:

Encode HO specs in FO SMT decidable logic

Metatheory:

THEOREM 4.1. *[Soundness of λ^R]*

- ***Denotations*** *If* $\Gamma; R \vdash p : \tau$ *then* $\forall \theta \in [\![\Gamma]\!].\theta \cdot p \in [\![\theta \cdot \tau]\!]$.
- ***Preservation*** *If* $\emptyset; \emptyset \vdash p : \tau$ *and* $p \hookrightarrow^\star w$, *then* $\emptyset; \emptyset \vdash w : \tau$.

Refinement Reflection, POPL'18

by **Vazou**, Tondwalkar, Choudhury, Scott, Newton, Wadler, and Jhala

E PROOF OF SOUNDNESS

We prove Theorem 4.1 of § D by reduction to Soundness of λ^U [Vazou et al. 2014a].

Theorem E.1. *[Denotations] If* $\Gamma \vdash p : \tau$ *then* $\forall \theta \in [\![\Gamma]\!].\theta \cdot p \in [\![\theta \cdot \tau]\!]$.

Proof. We use the proof from [Vazou et al. 2014b] and specifically Lemma 4 that is identical to the statement we need to prove. Since the proof proceeds by induction in the type derivation, we need to ensure that all the modified rules satisfy the statement.

- T-Exact Assume $\Gamma \vdash e : \{v : B \mid \{\_|r\} \wedge v = e\}$. By inversion $\Gamma \vdash e : \{v : B \mid \{\_|r\}\}$(1). By (1) and IH we get $\forall \theta \in [\![\Gamma]\!].\theta \cdot e \in [\![\theta \cdot \{v : B \mid \{\_|r\}\}]\!]$. We fix a $\theta \in [\![\Gamma]\!]$ We get that if $\theta \cdot e \hookrightarrow^\star w$, then $\theta \cdot \{\_|r\}[v/w] \hookrightarrow^\star$ True. By the Definition of = we get that $w = w \hookrightarrow^\star$ True. Since $\theta \cdot (v = e)[v/$ $e \in [\![\theta \cdot \{v : B \mid \{\_|r\} \wedge v =$ $\{v : B \mid \{\_|r\} \wedge v = e\}]\!]$.
- T-Let Assume $\Gamma \vdash$ let rec τ (2), and $\Gamma \vdash \tau$ (3). By IH (2'). By (1') and by the type (3) $\forall \theta \in [\![\Gamma]\!].\theta \cdot p$ [fix x e
- T-Refl Assume $\Gamma \vdash$ refle e in $p : \tau$. By IH, $\forall \theta \in [\![\Gamma]\!]$ are closed under evaluatio
- T-Fix In Theorem 8.3 from and Paolini 2004]) we pro

Theorem E.2. *[Preservation] If* $\emptyset \vdash p : \tau$ *and* $p \hookrightarrow^\star w$ *then* $\emptyset \vdash w : \tau$.

Proof. In [Vazou et al. 2014b] proof proceeds by iterative application of Type Preservation Lemma 7. Thus, it suffices to ensure Type Preservation in λ^R, which it true by the following Lemma. □

Lemma E.3. *If* $\emptyset \vdash p : \tau$ *and* $p \hookrightarrow p'$ *then* $\emptyset \vdash p' : \tau$.

Proof. Since Type Preservation in λ^U is proved by induction on the type derivation tree, we need to ensure that all the modified rules satisfy the statement.

- T-Exact Assume $\emptyset \vdash p : \{v : B \mid \{\_|r\} \wedge v = p\}$. By inversion $\emptyset \vdash p : \{v : B \mid \{\_|r\}\}$. By IH we get $\emptyset \vdash p' : \{v : B \mid \{\_|r\}\}$. By rule T-Exact we get $\emptyset \vdash p' : \{v : B \mid \{\_|r\} \wedge v = p'\}$. Since subtyping is closed under evaluation, we get $\emptyset \vdash \{v : B \mid \{\_|r\} \wedge v = p'\} \leq \{v : B \mid \{\_|r\} \wedge v = p\}$. By rule T-Sub we get $\emptyset \vdash p' : \{v : B \mid \{\_|r\} \wedge v = p\}$.

By 25, the above set is not empty, and hence τ is valid under d. □

Example: Fibonacci is increasing In § 2 we verified that under a definition d that includes fib, the term fibUp proves

$$n : \text{Nat} \rightarrow \{\text{fib } n \leq \text{fib } (n+1)\}$$

Thus, by Theorem D.3 we get

$$\forall n.0 \leq n \hookrightarrow^\star \text{True} \Rightarrow \text{fib } n \leq \text{fib } (n+1) \hookrightarrow^\star \text{True}$$

for all assignments $\sigma \models p$.

Embedding Functions As λ^S is a first-order logic, we embed λ-abstraction and application using the uninterpreted functions lam and app. We embed λ-abstractions using lam as shown in

Corollary F.3. *If* $\Gamma \vdash e :$ Bool, *e reduces to a value and* $\Gamma \vdash e \rightsquigarrow p$, *then for every* $\theta \in [\![\Gamma]\!]$ *and every* $\sigma \in \theta^\perp$, $\theta^\perp \cdot e \hookrightarrow^\star$ True *iff* $\sigma^\beta \models p$.

referring to the proofs in [Vazou et al. 2014b].

Liquid Haskell on papers

Expressiveness

Refinement Reflection, POPL'18

by **Vazou**, Tondwalkar, Choudhury, Scott, Newton, Wadler, and Jhala

Refinement Types for Haskell, ICFP'14

by **Vazou**, Seidel, Jhala, Vytiniotis, and Peyton-Jones

Liquid Haskell on papers

Expressiveness

Refinement Reflection, POPL'18

by **Vazou**, Tondwalkar, Choudhury, Scott, Newton, Wadler, and Jhala

Refinement Types for Haskell, ICFP'14

by **Vazou**, Seidel, Jhala, Vytiniotis, and Peyton-Jones

Liquid Haskell on papers

Expressiveness

Refinement Reflection, POPL'18

by **Vazou**, Tondwalkar, Choudhury, Scott, Newton, Wadler, and Jhala

Refinement Types for Haskell, ICFP'14

by **Vazou**, Seidel, Jhala, Vytiniotis, and Peyton-Jones

Liquid Haskell on papers

Expressiveness

Refinement Reflection, POPL'18
by **Vazou**, Tondwalkar, Choudhury, Scott, Newton, Wadler, and Jhala

Bounded Refinement Types, ICFP'15
by **Vazou**, Bakst, and Jhala

Abstract Refinement Types, ESOP'13
by **Vazou**, Rondon, and Jhala

Refinement Types for Haskell, ICFP'14
by **Vazou**, Seidel, Jhala, Vytiniotis, and Peyton-Jones

Liquid Haskell on github

ucsd-progsys / liquidhaskell

Unwatch 22 | Star 473 | Fork 75

Code | Issues 201 | Pull requests 5 | Projects 0 | Wiki | Insights | Settings

Liquid Types For Haskell

Edit

Add topics

8,382 commits | 94 branches | 19 releases | 38 contributors

Branch: develop | New pull request | Create new file | Upload files | Find file | Clone or download

nikivazou Merge pull request #1223 from ucsd-progsys/HaskellFunInRefs ... Latest commit 82f5baa 5 days ago

benchmarks	move tests into pos	a month ago
devel	Fix travis error	6 days ago

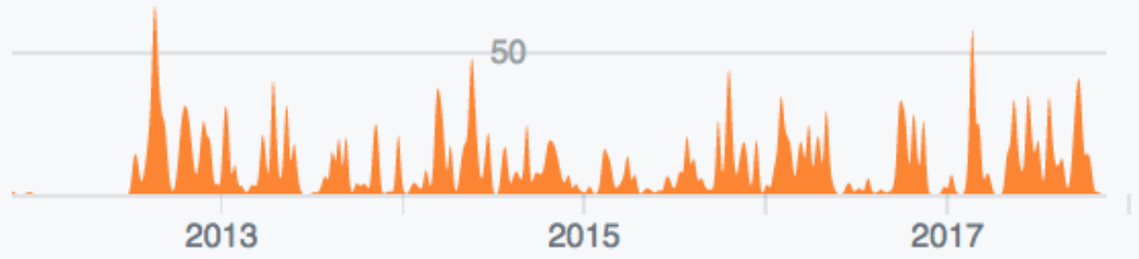
Liquid Haskell on github

Jan 15, 2012 – Jan 25, 2018

Contributions: **Commits**

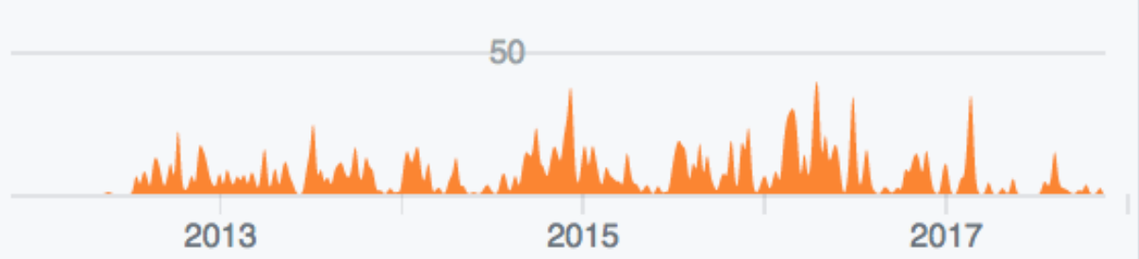
ranjitjhala — #1

3,093 commits 474,271 ++ 304,536 --



nikivazou — #2

2,113 commits 358,364 ++ 294,962 --



gridaphobe — #3

814 commits 114,731 ++ 79,008 --

spinda — #4

155 commits 7,430 ++ 5,350 --

Liquid Haskell dev team

Niki Vazou & Ranjt Jhala,
Liquid Haskell dev team

Liquid Haskell in the real world

Education

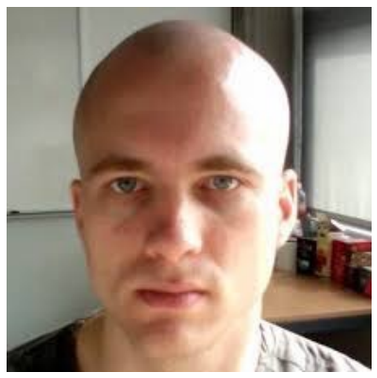
Will Kunkel,
undergrad @UMD

Rachel Xu & Xinyue Zhang,
undergrads @Bryn Mawr College

Niki Vazou & Ranjt Jhala,
Liquid Haskell dev team

Industry

Gabriel Gonzalez
Awake Security



Edsko de Vries
Well-Typed

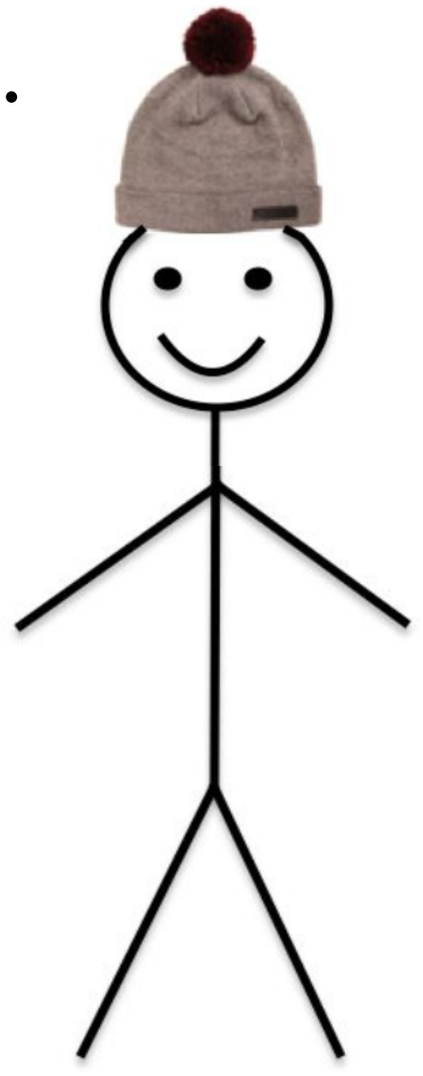
Liquid Haskell in Education

This is Will Kunkel.

Will took my undergrad course on Haskell.

Will learnt Liquid Haskell in 2 weeks.

Will was **3rd** in POPL'18 **Student Research Competition** with his project "Comparing Liquid Haskell and Coq"



Will is smart.

Be like Will.

Liquid Haskell in Education

Two winners in POPL'18 Student Research Competition

"Comparing Liquid Haskell and Coq"

Will Kunkel,
undergrad @UMD

"Comparing Dependent Haskell, Liquid Haskell and F*"

Rachel Xu & Xinyue Zhang,
undergrads @Bryn Mawr College

Liquid Haskell in Education

Two winners in POPL'18 Student Research Competition

"Comparing Liquid Haskell and Coq"

Will Kunkel,
undergrad @UMD

"Comparing Dependent Haskell, Liquid Haskell and F*"

Rachel Xu & Xinyue Zhang,
undergrads @Bryn Mawr College

Liquid Haskell in the real world

Education

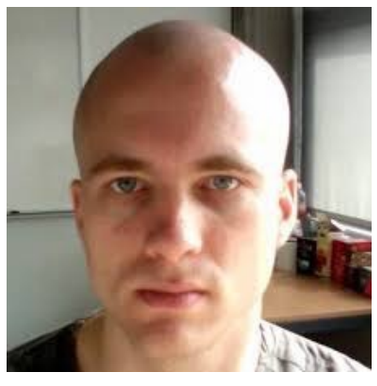
Will Kunkel,
undergrad @UMD

Rachel Xu & Xinyue Zhang,
undergrads @Bryn Mawr College

Niki Vazou & Ranjt Jhala,
Liquid Haskell dev team

Industry

Gabriel Gonzalez
Awake Security



Edsko de Vries
Well-Typed

Liquid Haskell in Industry

Gabriel Gonzalez

AWAKE

Static checks during parsing.

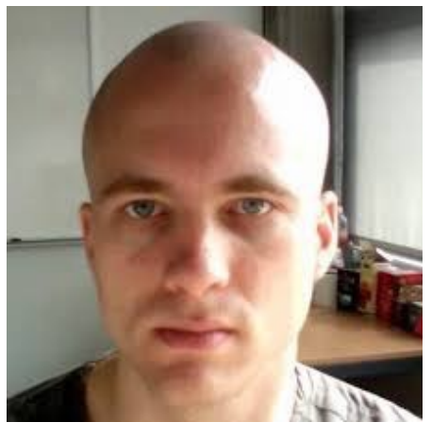
Provably Correct & Faster (x6) Code!

Huge speedup for internet traffic parsing.

Liquid Haskell in Industry

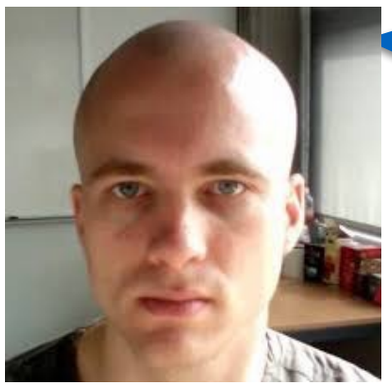
Gabriel Gonzalez

AWAKE

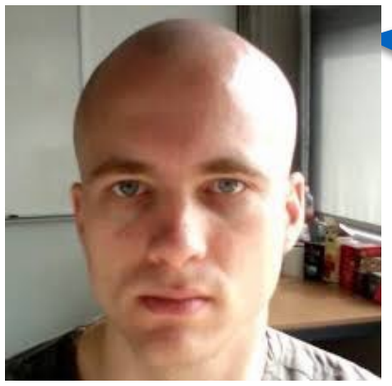


Well-Typed

Edsko de Vries

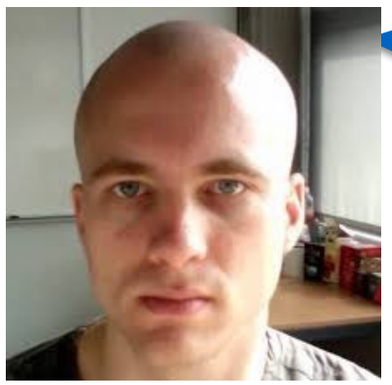


I am a Haskell consultant at a **cryptocurrency/blockchain** company.
We have a blockchain algorithm written **in paper** & a **Haskell implementation**.
We want Liquid Haskell to connect them.



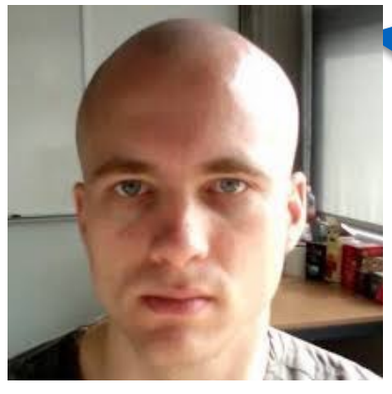
I am a Haskell consultant at a **cryptocurrency/blockchain** company. We have a blockchain algorithm written **in paper** & a **Haskell implementation**. We want Liquid Haskell to connect them.

Awesome! Lmk if you need anything!



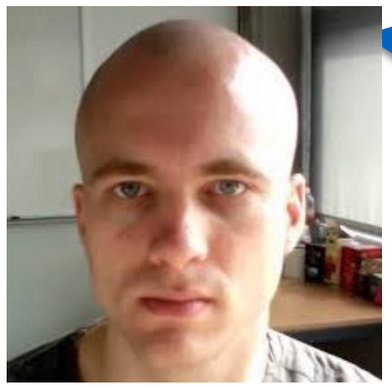
I am a Haskell consultant at a **cryptocurrency/blockchain** company. We have a blockchain algorithm written **in paper** & a **Haskell implementation**. We want Liquid Haskell to connect them.

Awesome! Lmk if you need anything!



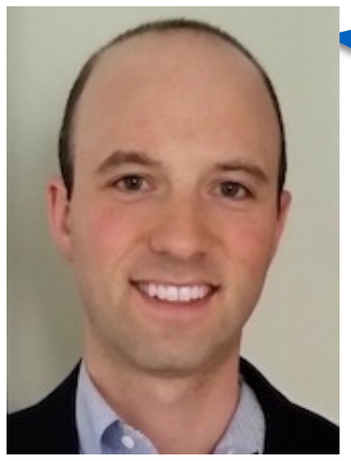
I want **interactive proof generation**!

More user suggestions



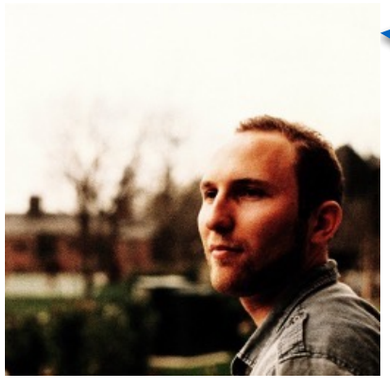
Edsko de Vries
industrial rep.

I want **interactive proof generation**!



Richard Eisenberg
ghc devs rep.

Can we use Liquid Haskell proofs for **compiler optimizations**?

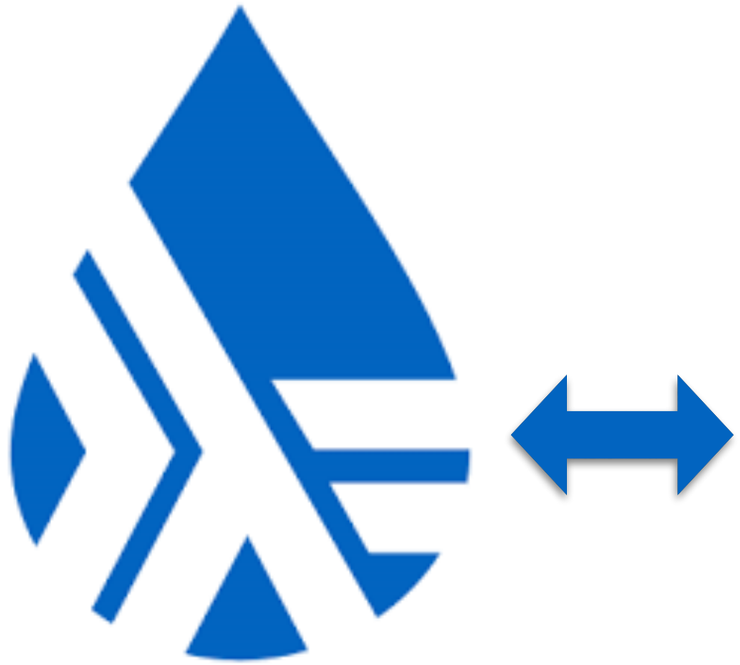


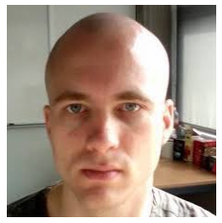
Leo Lambropoulos
Coq rep.

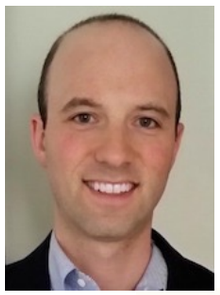
I do not trust it! Liquid Haskell should generate **proof certificates**!

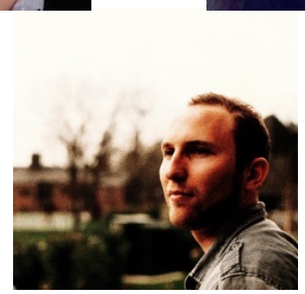
Liquid Haskell has many users

7,765 downloads

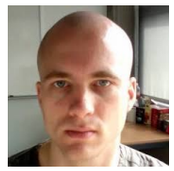


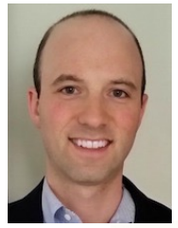




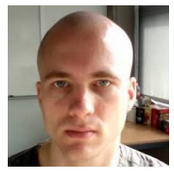


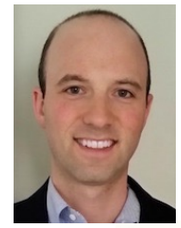
many users, but two main devs

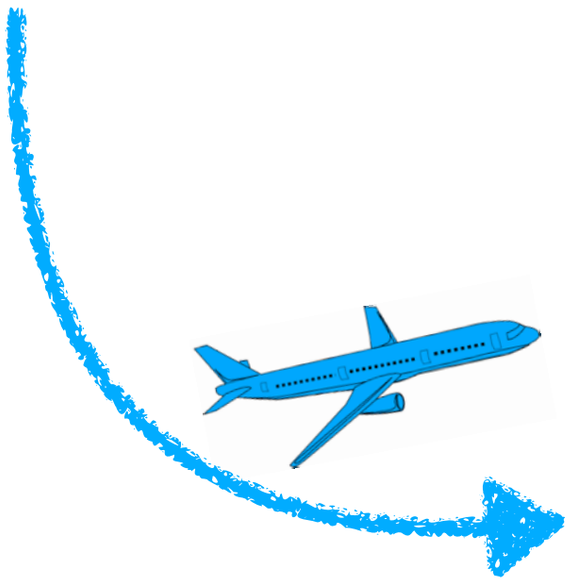




many users, but two main devs





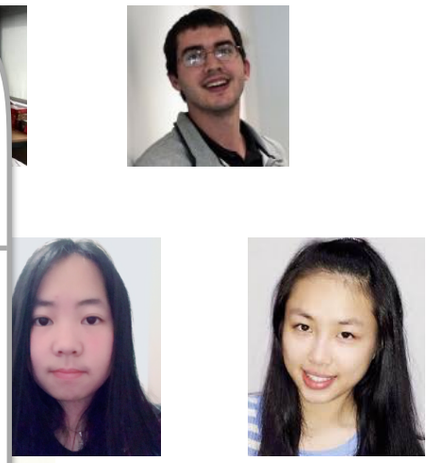


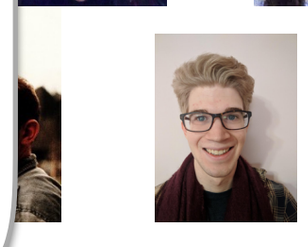
many users, but two main devs

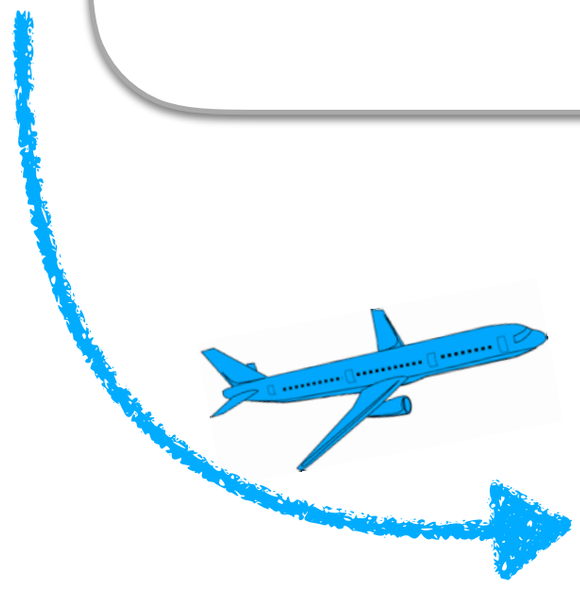
EMERGENCY ALERTS

Emergency Alert

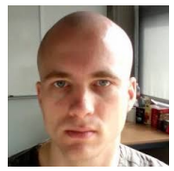
Liquid Haskell is not updated!
SEEK IMMEDIATE SHELTER.







many users, but two main devs



Goal: Expand the Liquid Haskell Team

Future work in Liquid Haskell

I. Proof Assistance

II. Compiler Interaction

III. Real-world applications

I. Proof Assistance

Goal: Coq-like IDE

for case splitting, sub-goal detection

I. Proof Assistance

Idea: Proofs are Haskell programs

Program Synthesis for Proof Generation

Error Reporting for Failure Diagnosis

I. Proof Assistance

Idea: Proofs are Haskell programs

Program Synthesis for Proof Generation

Error Reporting for Failure Diagnosis

II. Compiler Interaction

Goals:

more precise error messages

compiler optimizations

II. Compiler Interaction

Idea:

Translate Refinement to Dependent Types

Challenge:

Dependent Types require explicit proofs

Future work in Liquid Haskell

I. Proof Assistance

II. Compiler Interaction

III. Real-world applications

Vision:
Embed verification in Haskell programming

Vision:
Embed verification in Haskell programming

Specs are just comments **inside** the languages, ...
thus, learning effort is small.

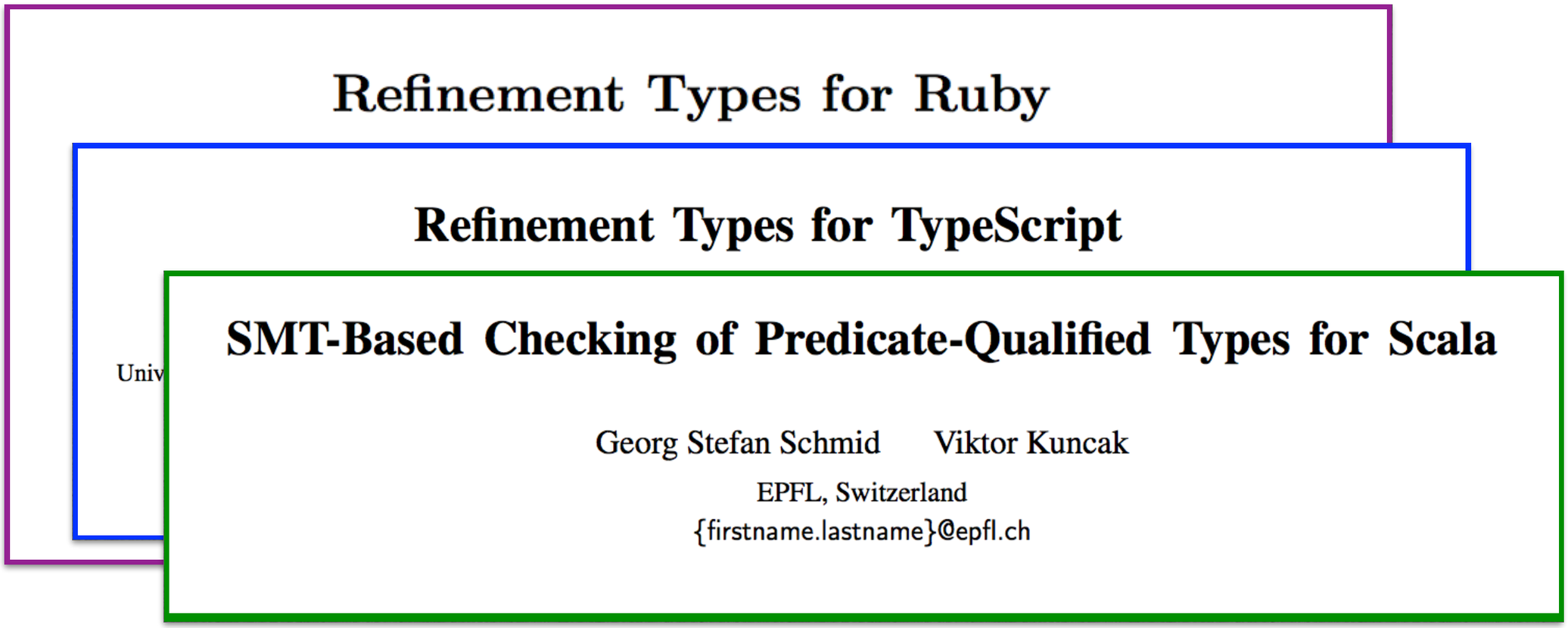
Semi-**automatically** machine checked, via SMTs.

For runtime **optimizations**, by the user or the compiler.

Vision:
Embed verification in ~~Haskell~~ programming

mainstream

Ruby, **JavaScript**, **Scala**, ...



Embed verification in Haskell programming

Fast & Safe Code

Static Checks

Theorem Proving

Refinement Reflection

Applications

Education & Industry

Thanks!

Embed verification in Haskell programming