

Understandable and Useful Error Messages for Liquid Types

Alcides Fonseca, Catarina Gamboa, João David,
Guilherme Espada, Paulo Canelas

80% Complaining about how hard it is to produce useful error messages for:

- LiquidHaskell
- Aeon
- LiquidJava
- MLVP

20% Suggestions to improve error messages

- Base Types: `int`, `bool`, `float`, ...
- Refinement Types: `{x:Int |  $x > 3$ }`
- Dependent Types: `(x:Int) -> {y:Int |  $y > x$ }`



<https://ucsd-progsys.github.io/liquidhaskell-blog/>

```
1  {-@ notEven :: {v:Int | v mod 2 = 1} @-}
2  notEven :: Int
3  notEven = 2
```

Error: Liquid Type Mismatch
Inferred type
VV : {v : Int | v == 2}
not a subtype of Required type
VV : {VV : Int | VV mod 2 == 1}

Unsafe

```
1 module Evens where
2
3
4
5
6 {-@ get :: {xs:[Int] | len xs > 0} ->
7     {n:Int | n < len xs} ->
8     Int @-}
9 get :: [Int] -> Int -> Int
10 get xs n = tail xs !! (n - 1)
```

Error: Liquid Type Mismatch
Inferred type
VV : {v : Int | v == n - 1}

not a subtype of Required type
VV : {VV : Int | 0 <= VV
&& VV < len (tail xs)}

In Context
xs : {v : [Int] | len v > 0
&& len v >= 0}

n : {n : Int | n < len xs}

- Everything in the context may be relevant
- Developers need to select what may be relevant
- There is a lot of redundancy (e.g., $len\ v > 0 \ \&\&\ len\ v \geq 0$)
- *Developers need to filter relevant assertions*
- *To understand errors, developers need to simulate the logical implications from the context.*
- *Developers are doing the work of the SMT solver!*

```
1 {-@ f :: {x:Int | x > 0} -> Int -> {y:Int | (y > 0)} -> {z:Int | z = x + y} @-}  
2 f :: Int -> Int -> Int -> Int  
3 f x y z = x + y
```

Error: Liquid Type Mismatch

Inferred type

VV : {v : Int | v = x + y}

not a subtype of Required type

VV : {VV : Int | VV = x + z}

In Context

z : {z : Int | z > 0}

x : {v : Int | v > 0}

y : Int

- Liquid Type Checking relies heavily on rewrites:
 - Internal rewrite to ANF
 - Refinements renamed in function calls

```
def minimizeGoal(m:Int) -> r:Bool | m > -10 {
  fib(m) > 30
}
```

SMT-based Program Synthesis

**Fitness Function for
Property-Based Testing**

```
def f(i:Int, j:Int) -> k:Int | k > i+j and minimizeGoal(k) {
  ??
}
```



?? :: {k:Int | **k > i+j && **k > -10**}**

Another indirection layer!

@Refinement("x > 10")

int x = 11;

x = x - 1;

Context:

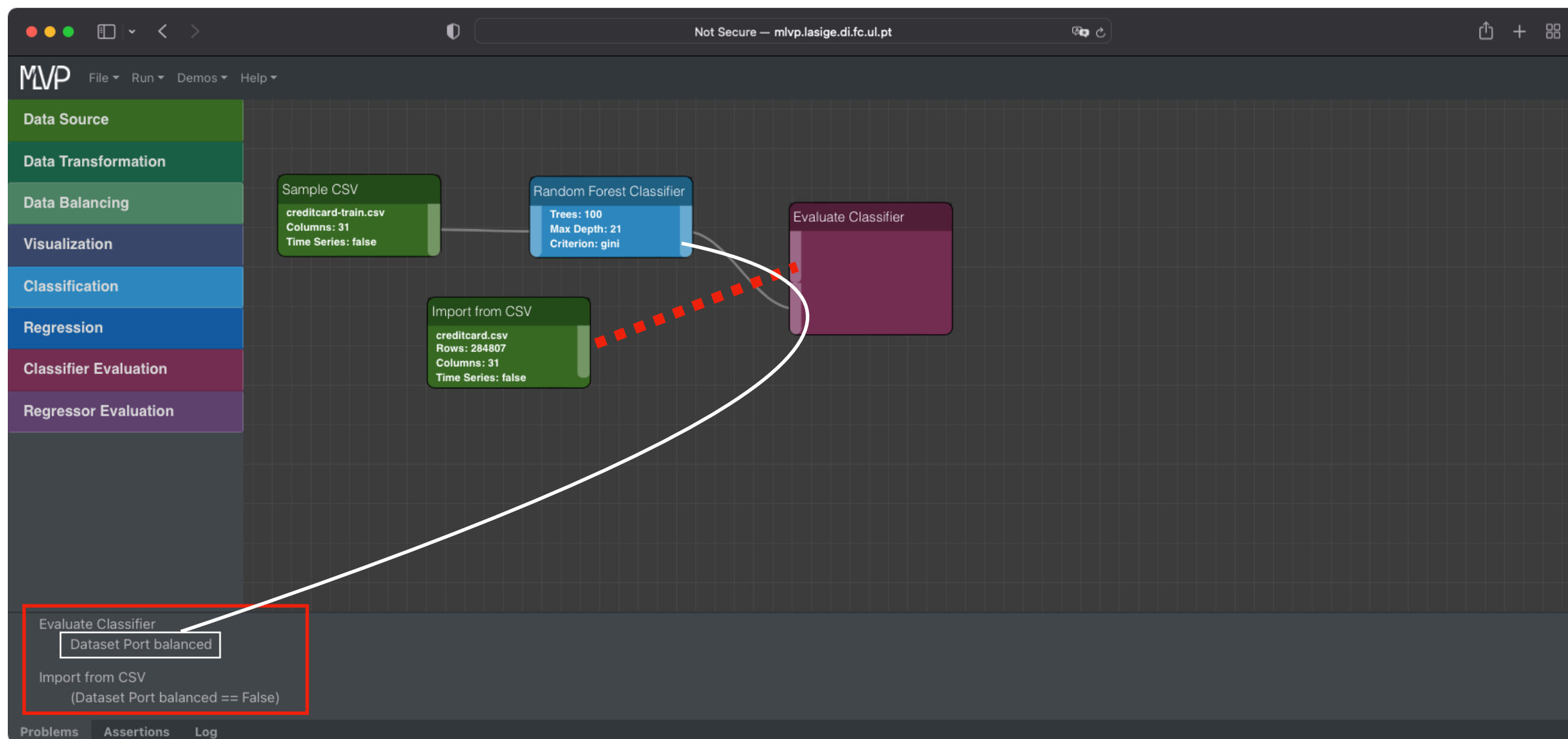
x_global | x_global > 10

x_local_1 | x_local_1 == 11

x_local_2 | x_local_2 == x_local - 1

$x == 11 - 1 \rightarrow x > 10$ ❌

Another indirection layer in the context!



Many indirection layers!

1. Show only relevant context
2. Substitute useless variables
3. Simplify redundant assertions
4. Evidence the minimal failing assertion
5. Allow to interactively expand the context
6. Evidence assertion provenance (colors?)

The screenshot displays the Lean 4 VSCode Extension interface. The main editor shows a Lean 4 file named `A.lean` with the following code:

```

1  import MathLib
2
3  theorem base : ∀ x y : Nat, x + y = y + x := by
4    intros x y
5    induction x with
6    | zero => simp
7    | succ x x_ih => {
8      let d := AddCommSemigroup.add_comm (Nat.succ x)
9      sorry
10     sorry
11   }
12
13

```

The right-hand pane shows the **Lean Infoview** for the current theorem. It displays the **Tactic state** and the **Expected type** for the current goal.

Tactic state:

```

case succ
y x : N
x_ih : x + y = y + x
├ Nat.succ x + y = y + Nat.succ x

```

Expected type:

```

y x : N
x_ih : x + y = y + x
├ ?m.195

```

Messages (1):

```

A.lean:8:13
function expected at
  AddCommSemigroup.add_comm (Nat.succ x) (sorryAx N false)
term has type
  Nat.succ x + sorryAx N false = sorryAx N false + Nat.succ x

```

The bottom status bar shows the current branch is `master*` and there are 2 commits and 1 change. A **Live Share** button is also visible.

Lean 4 VSCode Extension

```

▼ Source code
open data Nat : Set | zero | suc Nat
def wow {A : Type} {B : A -> Type} (a b : A) (x : B a) (y : B b) : Nat
example def test1 (A B : Type) (x : A) (y : B) => wow A B x y
example def test2 (A B : Type) (x : A) (y : B) => wow A B x x

▼ Trace
▶ Nat
▶ wow
▶ test1
▼ test2
  ▶ telescope
    ▼ wow A B x x : {?}
      ▶ wow A B x
      ▶ x : {?}
      • Nat = {?Nat?}
      ▼ {?} = Nat : ooType w
        ▼ {?} = Nat
          • Set 0 = {?}
          • Hole solved!

-----
      • wow {?Type h u?} {\ (_ : {?Type h u?}) => {?A?}} A B x x : Nat
    • A = A
  
```

Tesla Zhang (<https://twitter.com/OwoTizusa/status/1424159921172410372>)

```
def trans {A : Set} {a b c : A} (p : Eq a b) (q : Eq b c) :
  hcomp2d p q (idp a)
def pmap {A B : Set} (f : A → B) {a b : A} (p : Eq a b)
  : Eq (f a) (f b) ⇒ path (\ i ⇒ f (p.at i))

open data Nat : Set | zero | suc Nat
open data Int : Set
| pos Nat
| neg Nat { | zero ⇒ pos zero }

def addN (a b : Nat) : Nat
| zero, a ⇒ a
| a, zero ⇒ a
| suc a, b ⇒ suc (addN a b)
| a, suc b ⇒ suc (addN a b)
```

Tesla Zhang (<https://twitter.com/OwoTizusa/status/1396787442015817730>)

Understandable and Useful Error Messages for Liquid Types

Alcides Fonseca 🧑💻 , Catarina Gamboa 🧑💻 , João David,
Guilherme Espada 🧑💻 , Paulo Canelas 🧑💻