

# Refinement Reflection: Complete Verification with SMT

Friday  
@13:40

Niki Vazou, Anish Tondwalkar, Vikraman Choudhury, Ryan Scott, Ryan Newton,  
Philip Wadler, and Ranjit Jhala

## How to turn Haskell\* into a theorem prover

\*your favorite refinement typed programming language

```
-- math
{-@ fMono :: f:(Nat -> Int)
    -> (z:Nat -> {f z < f (z + 1) })
    -> x:Nat -> y:{Nat | x < y}
    -> {f x < f y}
    / [y]

    @-}

fMono f thm x y
| x + 1 < y
= f x <. f (x+1) ==. f y
*** QED
| otherwise
= f x <. f (y-1) ? fMono f thm x (y-1)
  <. f y      ? thm
*** QED
```

```
-- list laws
{-@ assoc :: xs:[a] -> ys:[a] -> zs:[a]
    -> {xs ++ (ys ++ zs) == (xs ++ ys) ++ zs}

    @-}

assoc [] _ _ = trivial
assoc (_:xs) ys zs = assoc xs ys zs
```

```
-- allDistr ≡ (∀x.p x ∧ q x) ⇒ ((∀x.p x) ∧ (∀x.q x))
{-@ allDistr :: p:(a -> Bool) -> q:(a -> Bool)
    -> (x:a -> ({p x}, {q x}))
    -> ((x:a -> {p x}), (x:a -> {q x}))

    @-}

allDistr _ _ andx = (pf, pf)
  where pf x = case andx x of (px,py) -> px && py
```