Solid Code with Liquid Types!

Alessio Ferrarini & Pablo Castellanos

IMDEA Software Institute, Madrid, Spain
{alessio.ferrarini, pablo.castellanos}@imdea.org

The untyped* world

```
def sumlist(xs):
    total = 0
    for x in xs:
        total += x
    return total
```

*There is no such thing as an untyped language

The untyped* world

```
>>> sumlist([1, 2, 3])
6
```

The untyped* world

```
>>> sumlist(“Hello Cadiz”)

unsupported operand type(s)
for +=: 'int' and 'str'
```

Defensive programming

```
def sumlist(xs):
    if not isinstance(xs, list):
        raise ValueError("no thanks!")
    for x in xs:
        if not isinstance(x, (int, float)):
            raise ValueError("no thanks!")
     ...
```

The typed world

```
sumList :: [Int] → [Int]
sumList []     = 0
sumList (x:xs) = x + sumList xs
```

The typed world

```
sumList “Hello Cadiz”

No instance for ‘Num Char’ arising
from a use of ‘sum’
```

Are we typed enough?

```
head :: [Int] → Int
head (x:_) = x
```

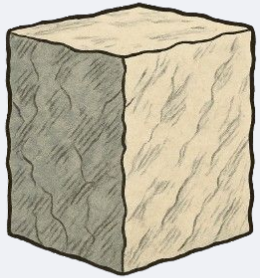
Sort of

```
head []

*** Exception: <file>:n:m-r:
Non-exhaustive patterns in function head
```

Many types

```
data NEList a
  = Singleton a
  | Cons a (NEList a)

head :: NEList a → NEList a
head (Singleton a) = a
head (Cons a xs)   = a
```

Refinement types and Liquid Haskell



```
Int
```

```
{ v:Int | v % 2 = 0 }
```

Refinement type = Base type + predicate

In Liquid Haskell predicates are expressions **(no quantifiers)**

A family of types

```
head :: { xs:[Int] | len xs > 0 } → Int
head (x:_) = x
```

With subtyping

$$\frac{\forall v.\phi_1(v) \Rightarrow \phi_2(v)}{\{v : T \mid \phi_1(v)\} \preceq \{v : T \mid \phi_2(v)\}}$$

Ok! Let’s see them in action!

ACKNOWLEDGMENTS

Partially funded by the European Union (GA 101039196). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union or the European Research Council. Neither the European Union nor the European Research Council can be held responsible for them.