TDD, as in Type-Directed Development

Clément Delafargue

2014-04-11

λ

VIA 9GAG.COM

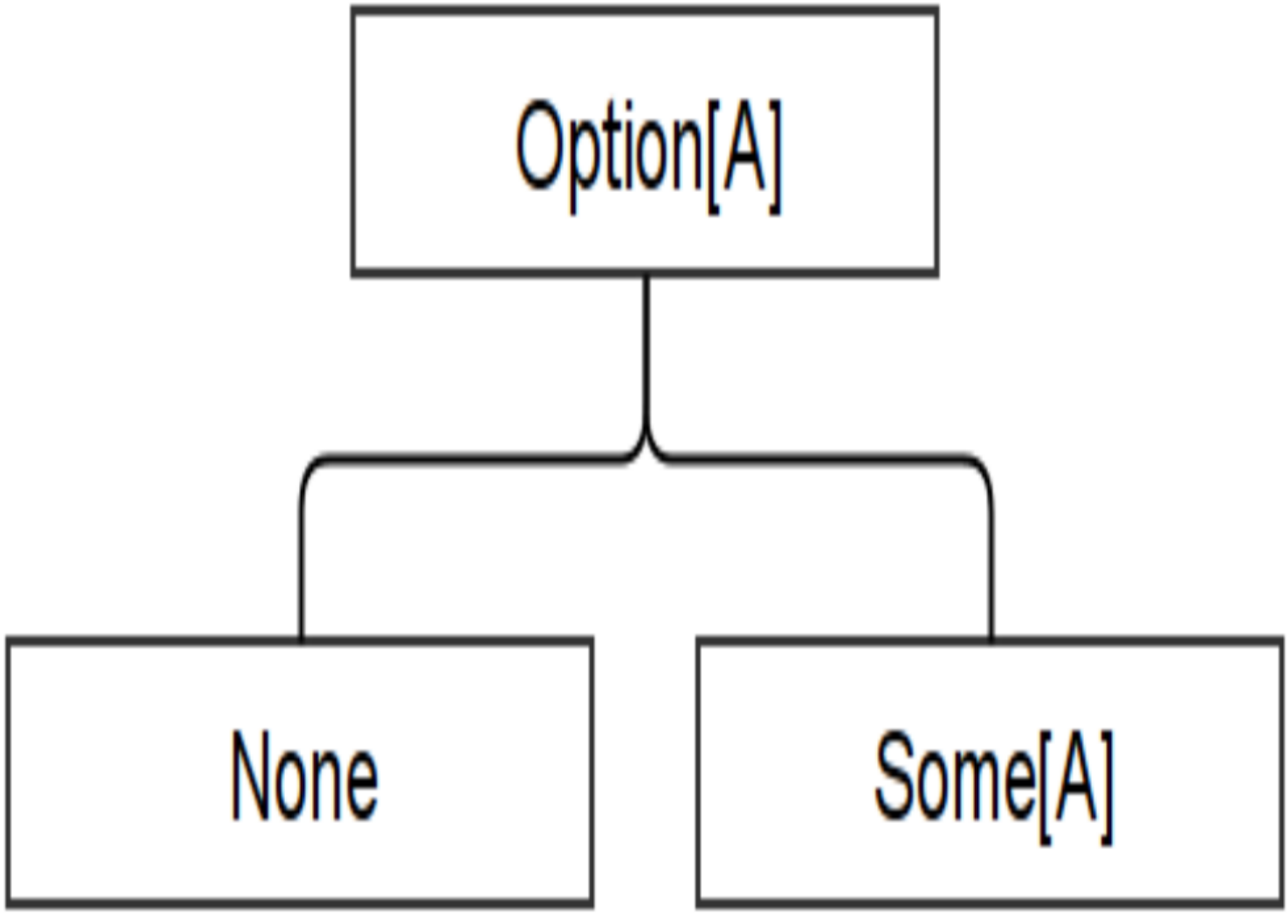
I'M NOT A SMART MAN

```
public int addFiveAction(
  params: Map<String, String>) {

    String nbS = params.get("number");
    if(!nbS.isEmpty()) {
        int nb = Integer.parseInt(nbS);
        return nb + 5;
    } else {
        return 0;
    }
}
```

```
addFiveAction({ "number": "12" });
    // 17

addFiveAction({ "yolo": "12" });
    // java.lang.NullPointerException

addFiveAction({ "number": "yolo" })
    // java.lang.NumberFormatException
```

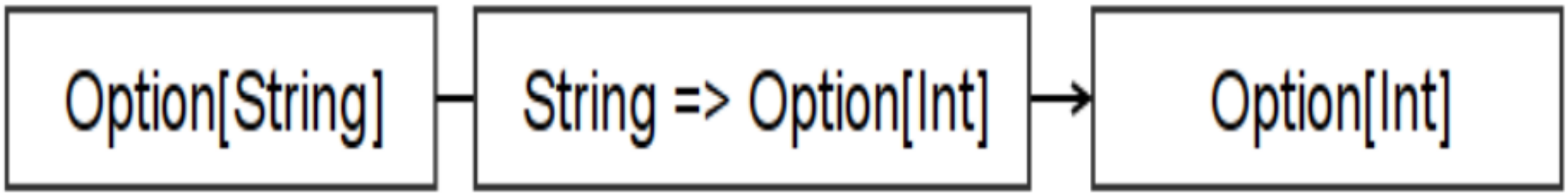
```
int addFiveAction(
  params: Map<String, String>) {
    String nbS = params.get("number");

    if(nbS != null) {
        if(!nbS.isEmpty()) {
            try {
                int nb = Integer.parseInt(nbS)
                return nb;
            } catch(NumberFormatException e) {
                return 0;
            }
        }
    } else {
        return 0;
    }
}
```

```
int addFiveAction(
  params: Map<String, String>) {
    String nbS1 = params.get("n1");
    String nbS2 = params.get("n2");

    if(nbS1 != null) {
        if(!nbS1.isEmpty()) {
            try {
                int nb1 = Integer.parseInt(nbS1)
                if(nbS2 != null) {
                    if(!nbS2.isEmpty()) {
                        try {
                            int nb2 = Integer.parseInt(nbS2);
                            return nb1 + nb2;
                        } catch(NumberFormatException e) {
                            return 0;
                        }
                    }
                }
            } catch(NumberFormatException e) {
                return 0;
            }
        }
    } else {
        return 0;
    }
}
```


Option[A]
None
Some[A]

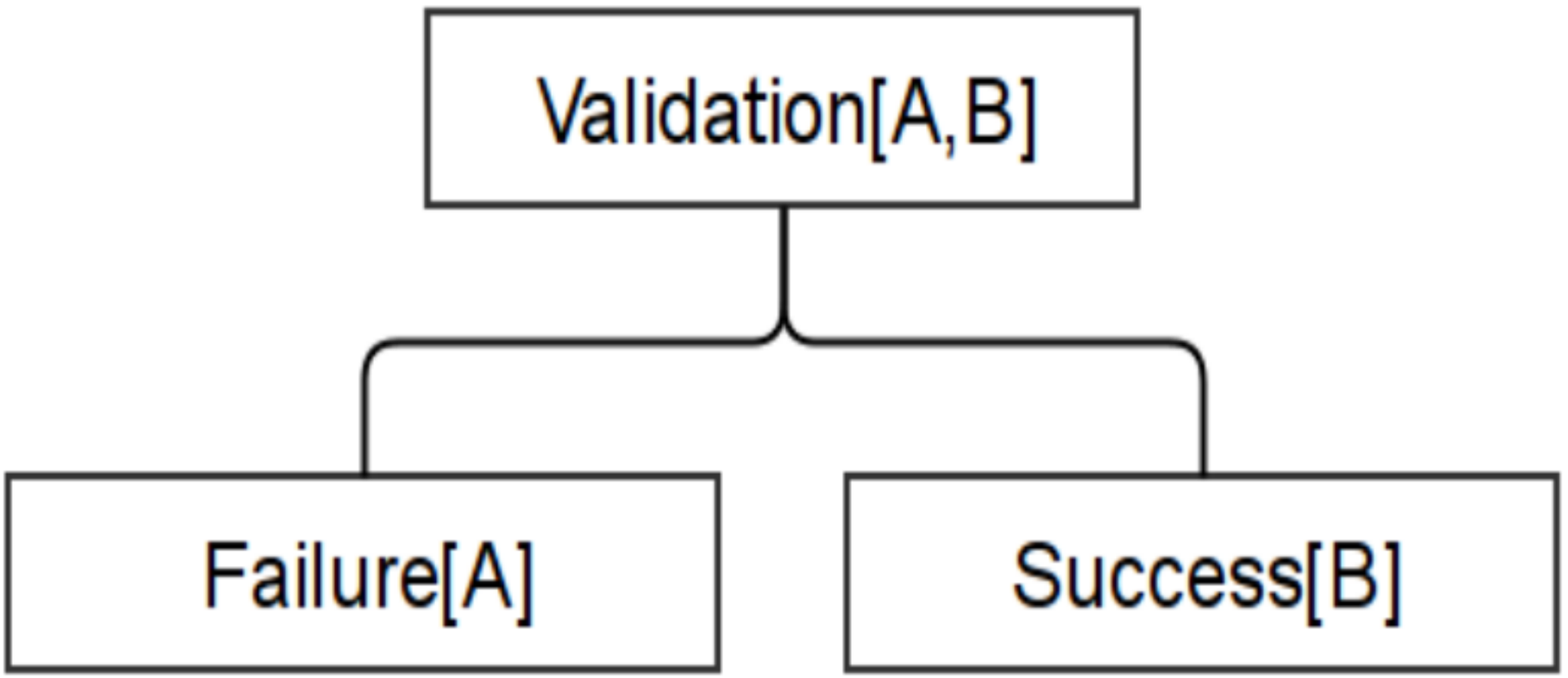
```
def parseInt(str: String):
  Option[Int]

def get(
  index: String,
  vals: Map[String, String]
): Option[String]
```

```
def getInt(
    index: String,
    vals: Map[String, String]
): Option[Int]
```



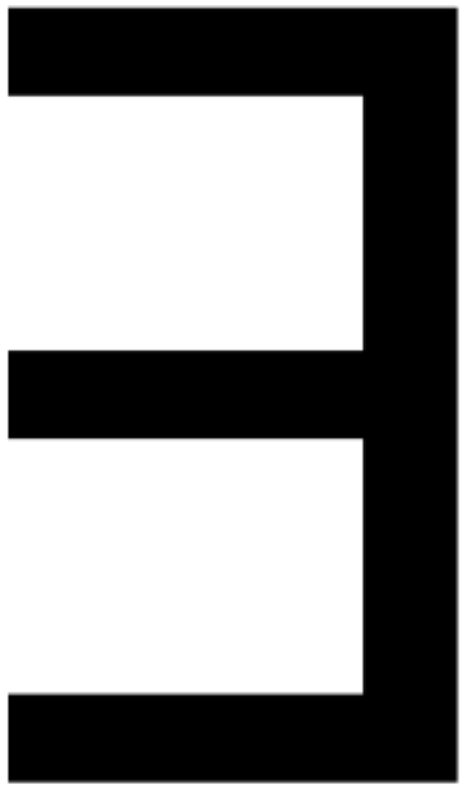
```
def addNumbersAction(
  params: Map[String, String]
): Int = {
    val i1 = getInt("n1", params)
    val i2 = getInt("n2", params)
    i1.getOrElse(0) + i2.getOrElse(0)
}
```

Non-empty list

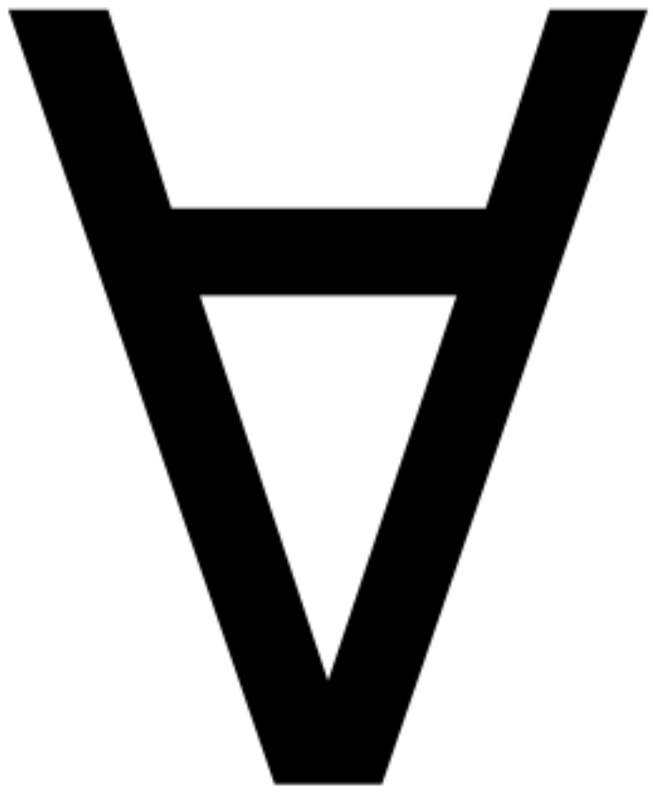

Validation[A,B]
Failure[A]
Success[B]

MASTER OF THE OBVIOUS

Why not tests?



« there exists »



« for all »

Type inference

```
val myVar =
    "obviously a string"

val myVar = List( "obviously"
        , "a"
        , "list"
        , "of"
        , "strings"
        )
```

Type ⇔ Property

Program ⇔ Proof


POLICE BOX
POLICE BOX

Expressive type systems

Parametricity

```
def reverse(
  xs: List[A]
): List[A]
```

Type-Directed Development

Confidence

Small bites

Types are the
best doc

Hoogle

Types can't
always prove
everything

And that's ok

```
reverseProp ::
    Eq a =>
    [a] -> [a] -> Bool
reverseProp xs ys =
    reverse (xs ++ ys) ==
        reverse ys ++ reverse xs
```

```
λ> quickCheck reverseProp
+++OK, passed 100 tests.
```

Types *then*

Property-based tests *then*

Unit tests

Thanks

Parametricity

http://haskell.org

http://scala.org

http://rustlang.org