Computer Science 3MI3 – Scala tidbits

Mark Armstrong

October 19th, 2020

Contents

Example of `Option[Either[A,B]]` recursive functions

In the first assignment, the final interpreter requires the return type of `Option[Either[Int,Boolean]]`. The idea is that `Option[Either[Int,Boolean]]` contains

- the value `None` which represents failure,
- for each integer `n`, the value `Some(Left(n))`,
- for each boolean `b`, the value `Some(Right(b))`

so this return type can be used for an interpreter which might return an integer, might return a boolean, or might fail.

This type can be somewhat awkward to work with, at least until we discuss the concept of *combining* such values (also known as *monadic bind*'s.)

For the moment, this example code may help you with your own definitions for the assignment.

Here, we work with that same type of `Option[Either[Int,Boolean]]` to define `sum` and `product` methods, with the idea that

- the `sum` of two integers is their sum in the usual sense,
- the `sum` of two booleans is the result of "logically or-ing" them,
- and the `sum` of two values of different types is undefined.

The `product` is similar, but uses logical and for the boolean case.

The important takeaway here is how we (using pattern matching)

1. "unwrap" the argument values `x` and `y`,
2. then in the cases we have two integers or two booleans, perform the relevant operation before
3. "rewrap"-ing the result; and
4. for the other cases,
   - (having a `None` for one or both of the arguments,
   - or having an integer for one argument and a boolean for the other)

   which we summarise with the "catch-all" case using underscores, we return `None`, representing failure.

```
def sum(x: Option[Either[Int,Boolean]],
        y: Option[Either[Int,Boolean]]):
         ↪  Option[Either[Int,Boolean]] =
  (x, y) match {
    case (Some(Left(m)), Some(Left(n))) => Some(Left(m + n))
    case (Some(Right(a)), Some(Right(b))) => Some(Right(a ||
     ↪  b))
    case (_ , _) => None
  }

def product(x: Option[Either[Int,Boolean]],
            y: Option[Either[Int,Boolean]]):
             ↪  Option[Either[Int,Boolean]] =
  (x, y) match {
    case (Some(Left(m)), Some(Left(n))) => Some(Left(m * n))
    case (Some(Right(a)), Some(Right(b))) => Some(Right(a &&
     ↪  b))
    case (_ , _) => None
  }
```

You may have noticed the extreme similarity between these two methods. And indeed, you may see similarity between these methods and the code you may write for the assignment.

If you are considering how we can *generalise* these operations in order to avoid repeating ourselves as I have above (having rewritten or copied/pasted the pattern matching, changing on the first two righthand sides) stay tuned; as mentioned earlier, we will discuss these concepts later on in the course.