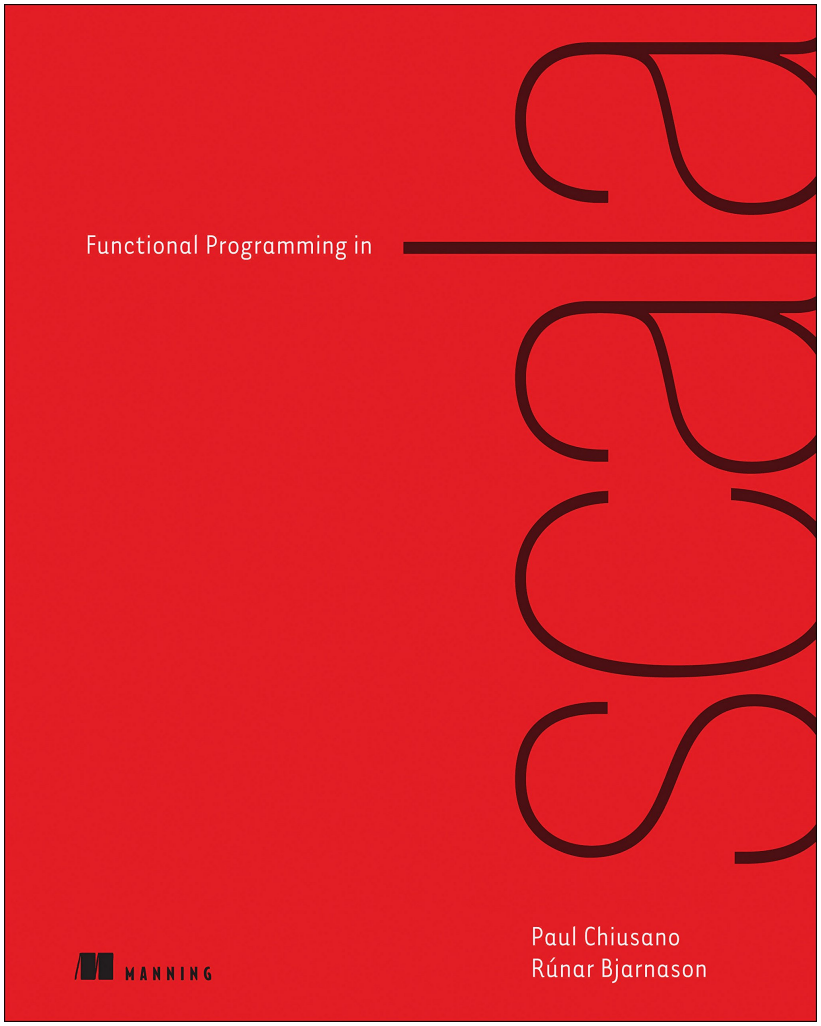
The Hidden Nature of Data

BOBKonf 2017

Martin Kühl, @mkhl

Motivation



Paul Chiusano and Rúnar Bjarnason

> **One might object that algebraic data types violate encapsulation** by making public the internal representation of a type. [...] **Exposing the data constructors of a type is often fine**, and the decision to do so is approached much like any other decision about what the public API of a data type should be.

Paul Chiusano
@pchiusano

I'm biased of course, but this is one of my favorite sidebars from #fpinscala

The rules of the game really are different in FP
The rules of the game really are different in FP
gist.github.com

But

Stackless Scala With Free Monads, Rúnar Bjarnason

```
sealed trait Free[S[+_],+A] {
  private case class FlatMap[S[+_],A,+B](
    a: Free[S,A],
    f: A => Free[S,B]) extends Free[S,B]
}
```

So when is exposing our data constructors **not** fine?
And what does that mean for when we should use pattern matching?

Example

Example: List

```
case object Nil extends List[Nothing]
case class Cons[+A](head: A, tail: List[A]) extends List[A]

sealed trait List[+A] {
  def fold[B](z: B)(f: (A, B) => B): B = this match {
    case Nil         => z
    case Cons(h, t) => f(h, t.fold(z)(f))
  }

  def append[B >: A](that: List[B]): List[B] =
      this.fold(that)(Cons(_, _))
```

Example: List + Append

```
case object Nil extends List[Nothing]
case class Cons[+A](head: A, tail: List[A]) extends List[A]
private case class Append[+A](left: List[A], right: List[A]) extends L

sealed trait List[+A] {
  def fold[B](z: B)(f: (A, B) => B): B = this match {
    case Nil           => z
    case Cons(h, t)    => f(h, t.fold(z)(f))
    case Append(l, r) => l.fold(r.fold(z)(f))(f)
  }

  def append[B >: A](that: List[B]): List[B] = (this, that) match {
    case (_, Nil) => this
    case (Nil, _) => that
    case _        => Append(this, that)
  }
```

Problems

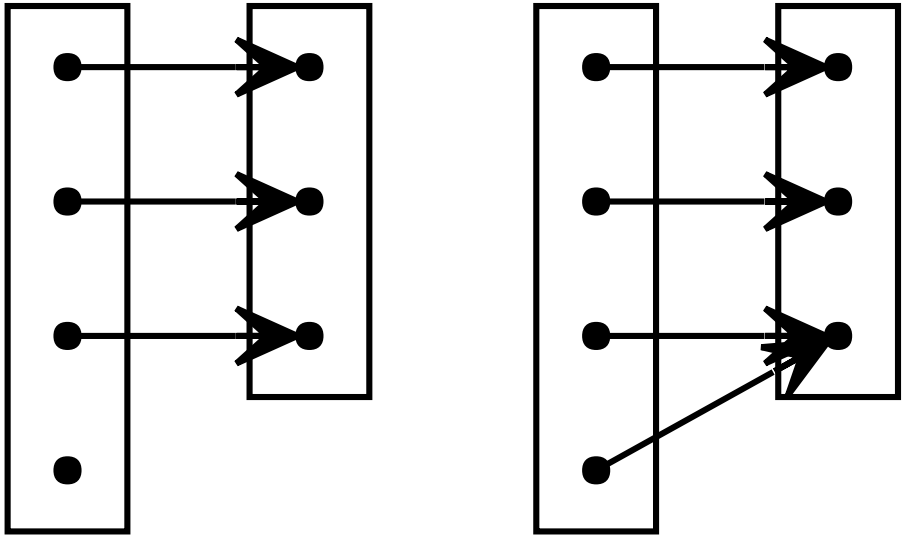
It breaks pattern matching

```
[warn] /path/to/Service.scala:28: match may not be exhaustive.
[warn] It would fail on the following input: Append(_, _)
[warn]     def go(as: List[A], count: Int): Int = as match {
[warn]                                            ^
```

It violates “No Confusion”

No Junk, No Confusion

— Joseph A. Goguen



It changes our algebra

Advice

> Be conservative in what you do, be liberal in what you accept from others.
>
> — Jon Postel

Is your library providing a language or interpreting one?

If you are providing a language, try to expose its structure

If you are interpreting a language, try consuming interfaces

If you are introducing interfaces, don’t replace existing structures

Changes to existing structures **should** affect downstream code

Pattern matching can be **nested**

Pattern matching can be **nested**

```
case (Some(Foo(x)), Right(Bar(y))) => …
```

More flexibility in defining types

```
case class Prog[A](get: Free[Lang, A])
```

VS.

```
type Prog[A] = Free[Lang, A]
```

Pattern matching
Try to use and support it

Thank you!

Questions?

Comments?

@mkhl

Martin Kühl

martin.kuehl@innoq.com

innoQ