Typelevel computations with Scala

Ilya Murzinov

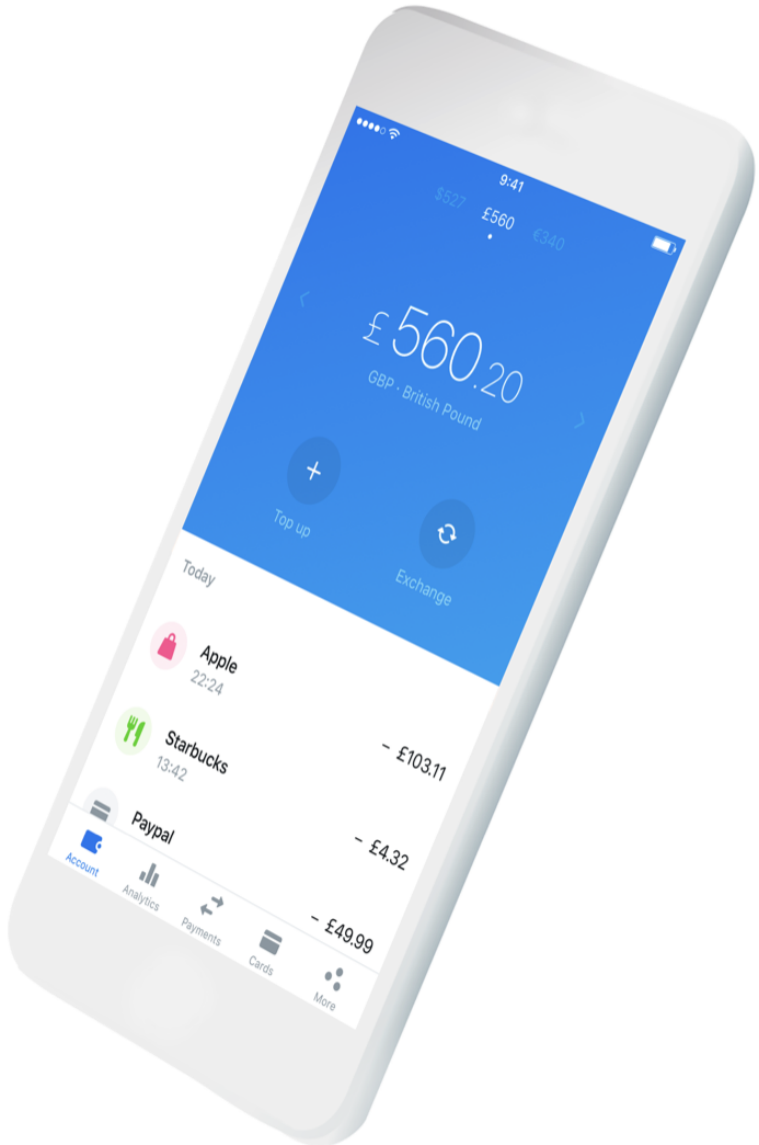
https://twitter.com/ilyamurzinov

https://github.com/ilya-murzinov

https://ilya-murzinov.github.io/slides/scalaspb2017

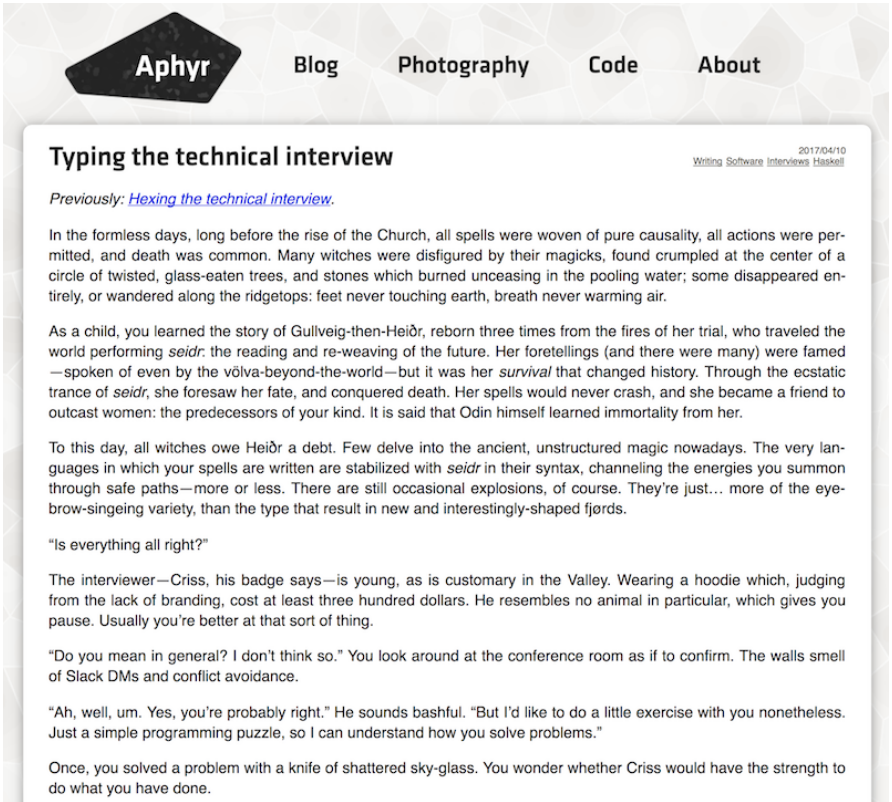
Revolut

Beyond Banking



Why?

We can do amazing things in Haskell



Aphyr Blog Photography Code About

Typing the technical interview

2017/04/10
Writing Software Interviews Haskell

Previously: Hexing the technical interview.

In the formless days, long before the rise of the Church, all spells were woven of pure causality, all actions were permitted, and death was common. Many witches were disfigured by their magicks, found crumpled at the center of a circle of twisted, glass-eaten trees, and stones which burned unceasing in the pooling water; some disappeared entirely, or wandered along the ridgetops: feet never touching earth, breath never warming air.

As a child, you learned the story of Gullveig-then-Heiðr, reborn three times from the fires of her trial, who traveled the world performing *seidr*: the reading and re-weaving of the future. Her foretellings (and there were many) were famed—spoken of even by the völva-beyond-the-world—but it was her *survival* that changed history. Through the ecstatic trance of *seidr*, she foresaw her fate, and conquered death. Her spells would never crash, and she became a friend to outcast women: the predecessors of your kind. It is said that Odin himself learned immortality from her.

To this day, all witches owe Heiðr a debt. Few delve into the ancient, unstructured magic nowadays. The very languages in which your spells are written are stabilized with *seidr* in their syntax, channeling the energies you summon through safe paths—more or less. There are still occasional explosions, of course. They're just… more of the eyebrow-singeing variety, than the type that result in new and interestingly-shaped fjørds.

"Is everything all right?"

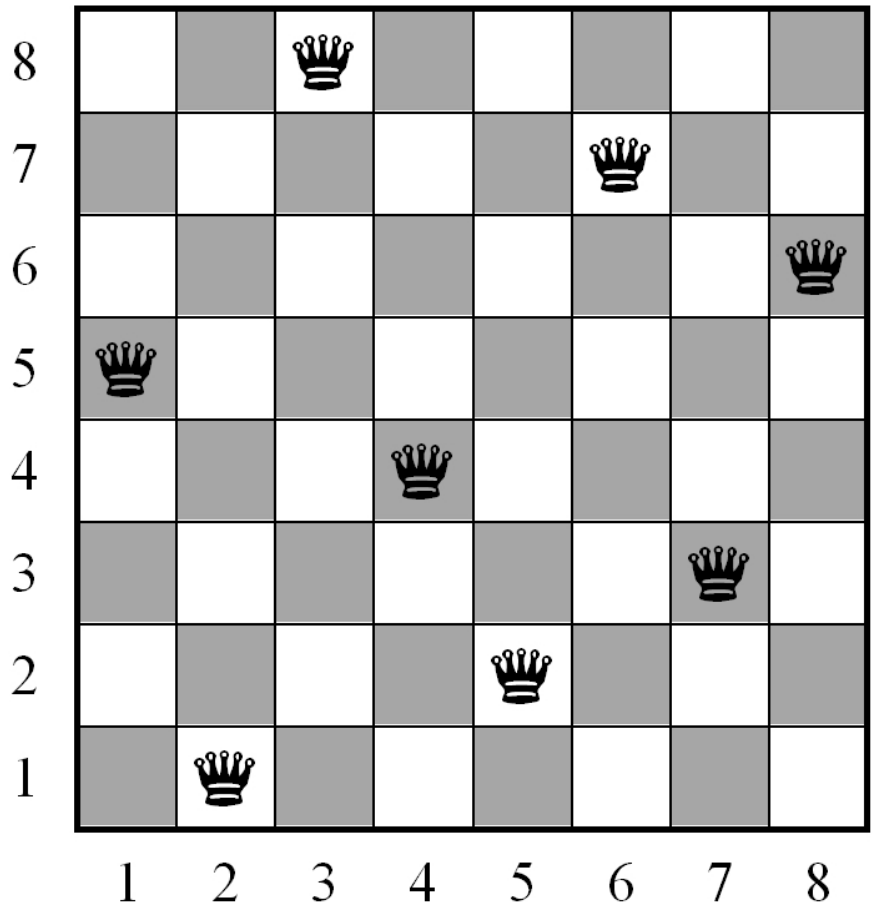
The interviewer—Criss, his badge says—is young, as is customary in the Valley. Wearing a hoodie which, judging from the lack of branding, cost at least three hundred dollars. He resembles no animal in particular, which gives you pause. Usually you're better at that sort of thing.

"Do you mean in general? I don't think so." You look around at the conference room as if to confirm. The walls smell of Slack DMs and conflict avoidance.

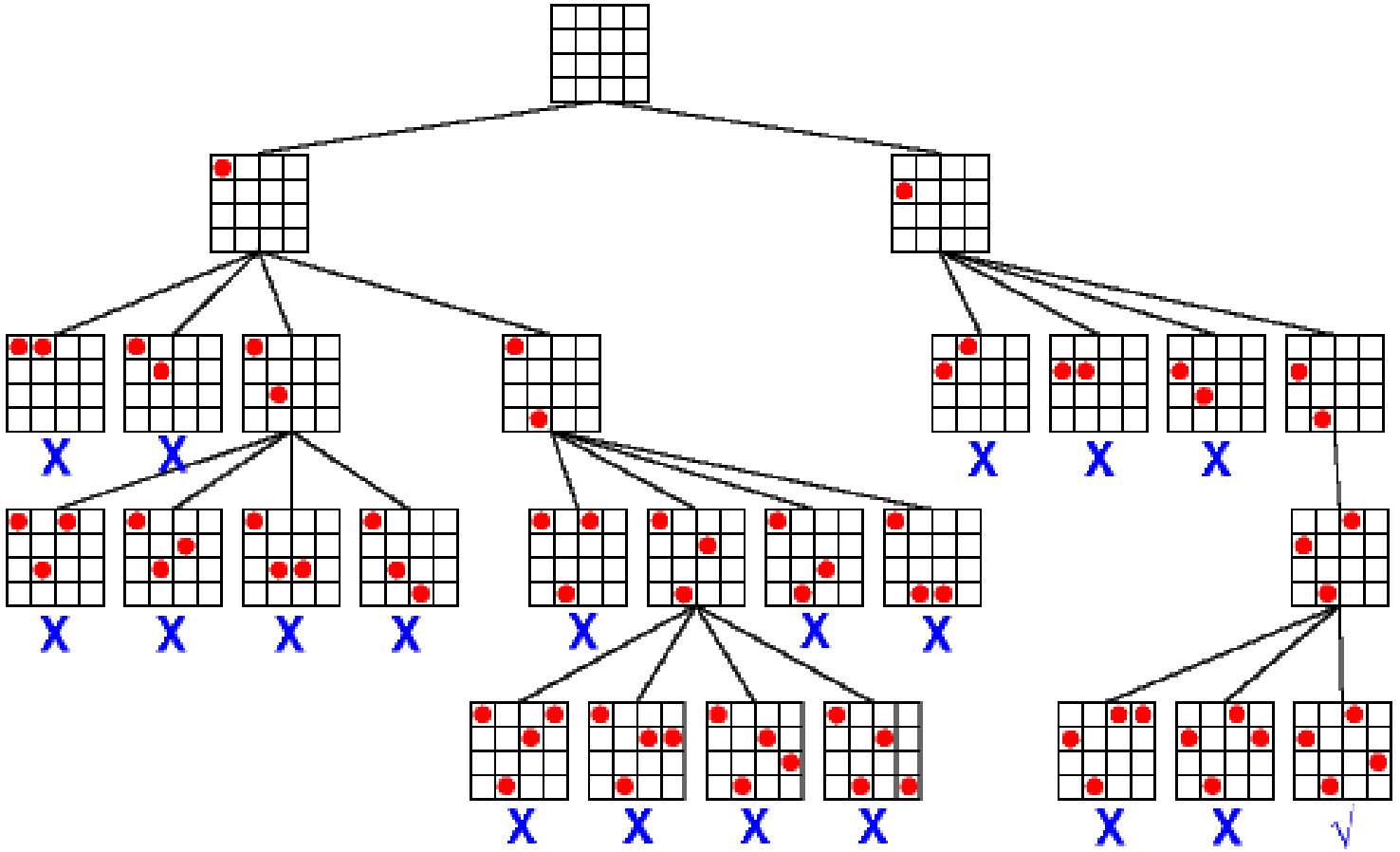
"Ah, well, um. Yes, you're probably right." He sounds bashful. "But I'd like to do a little exercise with you nonetheless. Just a simple programming puzzle, so I can understand how you solve problems."

Once, you solved a problem with a knife of shattered sky-glass. You wonder whether Criss would have the strength to do what you have done.

N queens problem



Algorithm



What we need for solution

- Natural numbers

What we need for solution

- Natural numbers
- Lists

What we need for solution

- Natural numbers
- Lists
- Booleans

What we need for solution

- Natural numbers
- Lists
- Booleans
- Functions

What we need for solution

- Natural numbers
- Lists
- Booleans
- Functions
- **The way to operate with all above**

Natural numbers

```
trait Nat
trait Z extends Nat
trait Succ[N <: Nat] extends Nat
```

Natural numbers

```
trait Nat
trait Z extends Nat
trait Succ[N <: Nat] extends Nat
```

```
type _0 = Z
type _1 = Succ[_0]
type _2 = Succ[_1]
type _3 = Succ[_2]
type _4 = Succ[_3]
type _5 = Succ[_4]

// and so on
```

Typelevel functions

```
trait Nat {
  type Add[A <: Nat]
}
```

Typelevel functions

```
trait Nat {
  type Add[A <: Nat]
}
```

```
trait Z extends Nat {
  type Add[A <: Nat] = A
}
```

Typelevel functions

```
trait Nat {
  type Add[A <: Nat]
}
```

```
trait Z extends Nat {
  type Add[A <: Nat] = A
}
```

```
trait Succ[N <: Nat] extends Nat {
  type Add[A <: Nat] = Succ[Z#Add[A]]
}
```

Typeclasses

Typeclasses

```
def print[A](a: A)(implicit e: Encoder[A]): String = e.print(a)
```

Typeclasses

```
def print[A](a: A)(implicit e: Encoder[A]): String = e.print(a)
```

```
scala> print(42)
42
```

Typeclasses

```
def print[A](a: A)(implicit e: Encoder[A]): String = e.print(a)
```

```
scala> print(42)
42
```

Deep down in some imported library:

```
trait Encoder { // <-- typeclass
  def print[A](a: A)
}

implicit val encoder = new Encoder[Int] {
  def print(i: Int) = i.toString
}
```

The Add typeclass

```
class Add[A, B] { type Out }
```

The Add typeclass

```
class Add[A, B] { type Out }
```

```
implicit def a0[A]: Add[_0, A] { type Out = A } = ???
```

The Add typeclass

```
class Add[A, B] { type Out }
```

```
implicit def a0[A]: Add[_0, A] { type Out = A } = ???
```

```
implicit def a1[A]: Add[A, _0] { type Out = A } = ???
```

The Add typeclass

```
class Add[A, B] { type Out }
```

```
implicit def a0[A]: Add[_0, A] { type Out = A } = ???
```

```
implicit def a1[A]: Add[A, _0] { type Out = A } = ???
```

```
implicit def a2[A, B, C](implicit
  a: Add[A, B] { type Out = C }
): Add[Succ[A], B] { type Out = Succ[C] } = ???
```

How to use it

How to use it

```
def implicitly[A](implicit a: A) = a
```

How to use it

```
def implicitly[A](implicit a: A) = a
```

```
scala> implicitly[Add[_1, _2]]
scala.NotImplementedError: an implementation is missing
    at scala.Predef$.$qmark$qmark$qmark(Predef.scala:252)
```

How to use it

```
def implicitly[A](implicit a: A) = a
```

```
scala> implicitly[Add[_1, _2]]
scala.NotImplementedError: an implementation is missing
    at scala.Predef$.$qmark$qmark$qmark(Predef.scala:252)
```

```
scala> :t implicitly[Add[_1, _2]]
Add[_1,_2]
```

The Aux pattern

The Aux pattern

```
class Add[A, B] { type Out }
object Add {
  type Aux[A, B, C] = Add[A, B] { type Out = C }
```

The Aux pattern

```
class Add[A, B] { type Out }
object Add {
  type Aux[A, B, C] = Add[A, B] { type Out = C }
```

```
  def apply[A, B](implicit a: Add[A, B]): Aux[A, B, a.Out] = ???
}
```

The Aux pattern

```
class Add[A, B] { type Out }
object Add {
  type Aux[A, B, C] = Add[A, B] { type Out = C }
```

```
  def apply[A, B](implicit a: Add[A, B]): Aux[A, B, a.Out] = ???
}
```

```
scala> :t Add[_1, _2]
Add[Succ[Z],Succ[Succ[Z]]]{type Out = Succ[Succ[Succ[Z]]]}
```

The Aux pattern

```
implicit def dummy(
  implicit
  a: Add[_1, _2],
  b: Add[_3, _4],
  c: Add[a.Out, b.Out]
) = ???
```

The Aux pattern

```
implicit def dummy(
  implicit
  a: Add[_1, _2],
  b: Add[_3, _4],
  c: Add[a.Out, b.Out]
) = ???
```

```
error: illegal dependent method type: parameter may only be
       referenced in a subsequent parameter section
  a: Add[_1, _2]
```

The Aux pattern

```
implicit def dummy(
  implicit
  a: Add[_1, _2],
  b: Add[_3, _4],
  c: Add[a.Out, b.Out]
) = ???
```

```
error: illegal dependent method type: parameter may only be
       referenced in a subsequent parameter section
  a: Add[_1, _2]
```

```
implicit def dummy[R1, R2](
  implicit
  a: Add.Aux[_1, _2, R1],
  b: Add.Aux[_3, _4, R2],
  c: Add[R1, R2]
): c.Out = ???
```

The real typeclass

```
trait Threatens[Q1 <: Queen[_, _], Q2 <: Queen[_, _]]
  { type Out <: Bool }

object Threatens {
  type Aux[Q1 <: Queen[_, _], Q2 <: Queen[_, _], R <: Bool] =
    Threatens[Q1, Q2] { type Out = R }

  implicit def t0[X1 <: Nat, Y1 <: Nat, X2 <: Nat, Y2 <: Nat,
                  EqX <: Bool, EqY <: Bool, EqXY <: Bool,
                  DX <: Nat, DY <: Nat, EqD <: Bool](
      implicit
      eqX: Eq.Aux[X1, X2, EqX],
      eqY: Eq.Aux[Y1, Y2, EqY],
      or0: Or.Aux[EqX, EqY, EqXY],
      dx: AbsDiff.Aux[X1, X2, DX],
      dy: AbsDiff.Aux[Y1, Y2, DY],
      eqD: Eq.Aux[DX, DY, EqD],
      res: Or[EqXY, EqD]):
      Aux[Queen[X1, Y1], Queen[X2, Y2], res.Out] = ???
}
```

What typeclasses are required for solution?

```
trait First[L <: List] { type Out }
trait Concat[A <: List, B <: List] { type Out <: List }
trait ConcatAll[Ls <: List] { type Out <: List }
trait AnyTrue[L] { type Out <: Bool }
trait Not[A <: Bool] { type Out <: Bool }
trait Or[A <: Bool, B <: Bool] { type Out <: Bool }
trait Eq[A <: Nat, B <: Nat] { type Out <: Bool }
trait Lt[A <: Nat, B <: Nat] { type Out <: Bool }
trait AbsDiff[A <: Nat, B <: Nat] { type Out <: Nat }
trait Range[A <: Nat] { type Out <: List }
trait Apply[F <: Func, A] { type Out }
trait Map[F <: Func, L <: List] { type Out <: List }
trait MapCat[F <: Func, L <: List] { type Out <: List }
trait AppendIf[B <: Bool, A, L <: List] { type Out <: List }
trait Filter[F <: Func, L <: List] { type Out <: List }
trait `QueensInRow[Y <: Nat, N <: Nat] { type Out <: List }
trait Threatens[Q1 <: Queen[_, _], Q2 <: Queen[_, _]]
  { type Out <: Bool }
trait Safe[Config <: List, Q <: Queen[_, _]] { type Out <: Bool }
trait AddQueen[N <: Nat, X <: Nat, Config <: List]
  { type Out <: List }
trait AddQueenToAll[N <: Nat, X <: Nat, Configs <: List]
  { type Out <: List }
trait AddQueensIf[P <: Bool, N <: Nat, X <: Nat, Configs <: List]
  { type Out <: List }
trait AddQueens[N <: Nat, X <: Nat, Configs <: List]
  { type Out <: List }
trait Solution[N <: Nat] { type Out <: List }
```

Implicit resolution is a search process

-Xlog-implicits

Diverging implicit expansion

```
[error] somefile.scala:XX:YY: diverging implicit expansion
        for type T
[error] starting with method m0 in class C
[error]   implicitly[T]
[error]              ^
[error] one error found
[error] (compile:compileIncremental) Compilation failed
```

Diverging implicit expansion

Diverging implicit expansion

"A couple of years ago when I was working through some issues like this I found that the easiest way to figure out what the divergence checker was doing was just to **throw some println**s **into the compiler and publish it locally.**"

(c) Travis Brown on stackoverflow

Diverging implicit expansion

```
trait S
trait V
trait T[A]
trait C[A, B]
implicit def a0[A, B](implicit ta: T[A], tb: T[B]): T[C[A, B]] = ???

implicit def a1(implicit a: T[C[V, C[V, V]]]): T[S] = ???

implicit val a2: T[V] = ???
```

Diverging implicit expansion

```
trait S
trait V
trait T[A]
trait C[A, B]
implicit def a0[A, B](implicit ta: T[A], tb: T[B]): T[C[A, B]] = ???

implicit def a1(implicit a: T[C[V, C[V, V]]]): T[S] = ???

implicit val a2: T[V] = ???
```

```
implicitly[T[C[S, V]]]

T[C[S, V]]
T[S]
T[C[V, C[V, V]]] // <- more complex
```

Diverging implicit expansion

```
trait S
trait V
trait T[A]
trait C[A, B]
implicit def a0[A, B](implicit ta: T[A], tb: T[B]): T[C[A, B]] = ???

implicit def a1(implicit a: T[C[V, C[V, V]]]): T[S] = ???

implicit val a2: T[V] = ???
```

```
implicitly[T[C[S, V]]]

T[C[S, V]]
T[S]
T[C[V, C[V, V]]] // <- more complex
```

```
[error] divexp.scala:20:13: diverging implicit expansion
        for type d.this.T[d.this.C[d.this.S,d.this.V]]
[error] starting with method a0 in class d
[error]   implicitly[T[C[S, V]]]
```

Shapeless to the rescue

```
trait S
trait V
trait T[A]
trait C[A, B]

implicit def a0[A, B](implicit
  ta: shapeless.Lazy[T[A]],
  tb: T[B]
): T[C[A, B]] = ???

implicit def a1(implicit a: T[C[V, C[V, V]]]): T[S] = ???
implicit val a2: T[V] = ???

implicitly[T[C[S, V]]]
```

The solution for 4x4

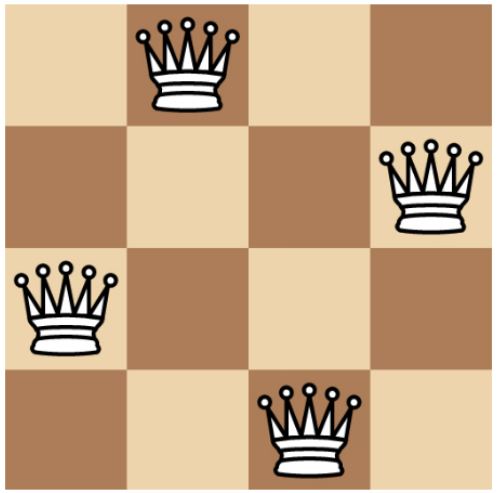
```
q.Solution[q.Succ[q.Succ[q.Succ[q.Succ[q.Z]]]]]{type Out = q.Cons[
  q.Cons[q.Queen[q.Succ[q.Succ[q.Succ[q.Z]]],q.Succ[q.Z]],
    q.Cons[q.Queen[q.Succ[q.Succ[q.Z]],q.Succ[q.Succ[q.Succ[q.Z]]]],
      q.Cons[q.Queen[q.Succ[q.Z],q.Z],
        q.Cons[q.Queen[q.Z,q.Succ[q.Succ[q.Z]]],q.Nil]
      ]
    ]
  ],
  q.Cons[
    q.Cons[q.Queen[q.Succ[q.Succ[q.Succ[q.Z]]],q.Succ[q.Succ[q.Z]]],
      q.Cons[q.Queen[q.Succ[q.Succ[q.Z]],q.Z],
        q.Cons[q.Queen[q.Succ[q.Z],q.Succ[q.Succ[q.Succ[q.Z]]]],
          q.Cons[q.Queen[q.Z,q.Succ[q.Z]],q.Nil]
        ]
      ]
    ],
    q.Nil
  ]
]}
```

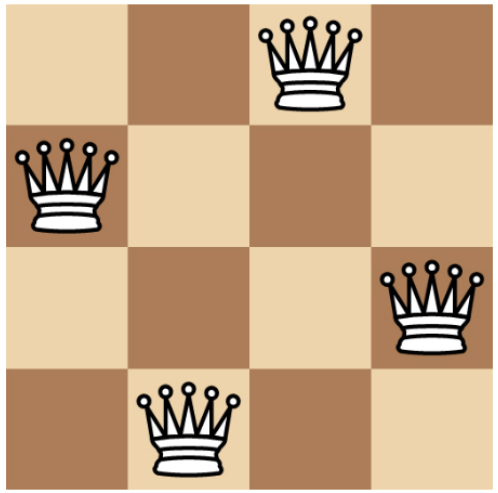
The solution for 4x4

```
Queen[_3, _1] :: Queen[_2, _3] :: Queen[_1, _0] :: Queen[_0, _2]
Queen[_3, _2] :: Queen[_2, _0] :: Queen[_1, _3] :: Queen[_0, _1]
```

The solution for 4x4

```
Queen[_3, _1] :: Queen[_2, _3] :: Queen[_1, _0] :: Queen[_0, _2]
Queen[_3, _2] :: Queen[_2, _0] :: Queen[_1, _3] :: Queen[_0, _1]
```





How can we use all this?

How can we use all this?

- for fun

How can we use all this?

- for fun
- for typeclass derivation with Shapeless

How can we use all this?

- for fun
- for typeclass derivation with Shapeless
- … and even encode dependent types in Scala

Dependent types in Scala

Dependent types in Scala

```
trait Sized[+A] {
  type Size
  def value: List[A]

  def concat[B >: A](other: Sized[B])(implicit
    a: Add[Size, other.Size]
  ): Sized[B] { type Size = a.Out } = new Sized[B] {
      type Size = a.Out
      def value = other.value ::: Sized.this.value
    }
}
```

Dependent types in Scala

```
object SNil extends Sized[Nothing] {
  type Size = _0
  def value = Nil
}
```

Dependent types in Scala

```
object SNil extends Sized[Nothing] {
  type Size = _0
  def value = Nil
}
```

```
class SCons[+A](head: A, tail: Sized[A]) extends Sized[A] {
  def value = head :: tail.value
}
object SCons {
  type Aux[+A, S] = SCons[A] {type Size = S}
  def apply[A](head: A, tail: Sized[A])(implicit
    a: Add[tail.Size, _1]
  ): Aux[A, a.Out] = new SCons[A](head, tail) {type Size = a.Out}
}
```

Summary

Scala type system/compiler is smart enough to do work for us.

Summary

Scala type system/compiler is smart enough to do work for us.

Even as complex as solving N queens problem.

Summary

Scala type system/compiler is smart enough to do work for us.

Even as complex as solving N queens problem.

With discussed techniques and patterns we can encode such the logic quite easily.

Summary

Scala type system/compiler is smart enough to do work for us.

Even as complex as solving N queens problem.

With discussed techniques and patterns we can encode such the logic quite easily.

And even readable to some extent.

Summary

Scala type system/compiler is smart enough to do work for us.

Even as complex as solving N queens problem.

With discussed techniques and patterns we can encode such the logic quite easily.

And even readable to some extent.

But it's still very hard to debug.

References

- "The Type Astronaut's Guide to Shapeless" by Dave Gurnell

References

- "The Type Astronaut's Guide to Shapeless" by Dave Gurnell
- "Hacking on scalac — 0 to PR in an hour" by Miles Sabin

References

- "The Type Astronaut's Guide to Shapeless" by Dave Gurnell
- "Hacking on scalac — 0 to PR in an hour" by Miles Sabin
- "Typing the technical interview" by Kyle Kingsbury, a.k.a "Aphyr"

References

- "The Type Astronaut's Guide to Shapeless" by Dave Gurnell
- "Hacking on scalac — 0 to PR in an hour" by Miles Sabin
- "Typing the technical interview" by Kyle Kingsbury, a.k.a "Aphyr"
- These slides

References

- "The Type Astronaut's Guide to Shapeless" by Dave Gurnell
- "Hacking on scalac — 0 to PR in an hour" by Miles Sabin
- "Typing the technical interview" by Kyle Kingsbury, a.k.a "Aphyr"
- These slides
- Solution of N queens problem on type level

Questions?

Thanks!