---

title: Typelevel computations with Scala author: Ilya Murzinov patat: incrementalLists: true wrap: true theme: emph: [vividRed]

```
trait List

trait Nil extends List
trait Cons[H, T <: List] extends List

type ::[H, T <: List] = Cons[H, T]

// Cons[A, B] ---> A :: B
```

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

type _0 = Z
type _1 = Succ[Z]
type _2 = Succ[_1]
//...
```

...

```
type L = _1 :: _2 :: _3 :: Nil
```

```
trait List {
  type Concat[That <: List] <: List
```

```
  }
```

...

```
trait Nil extends List {
  type Concat[That <: List] = That
}
trait Cons[H, T <: List] extends List {
  type First = H
  type Tail = T
  type Concat[That <: List] = Cons[H, T#Concat[That]]
}
```

...

```
scala> implicitly[(_1 :: Nil)#First =:= _1]

scala> implicitly[Nil#Concat[Nil] =:= Nil]

scala> implicitly[Cons[_1, Nil]#Tail#First =:= _1]
<console>:15: error: type First is not a member of
Nil
       implicitly[Cons[_1, Nil]#Tail#First =:= _1]
```

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

type =:=[A, B]
implicit def typeEq[A]: =:=[A, A] = ???
```

```
trait First[L <: List] { type Out }
implicit def f[H, T <: List]: First[Cons[H, T]] {type
Out = T} = ???
```

...

```
scala> :t implicitly[First[_1 :: _2 :: Nil]]
First[_1 :: (_2 :: Nil)]

scala> :t implicitly[First[Nil]]
<console>:15: error: could not find implicit value
for
                    parameter e: First[Nil]
       implicitly[First[Nil]]
                  ^
```

```
trait Dummy[L <: List]
implicit def dummy[L <: List](
  implicit
  f: First[L],
  eq: f.Out =:= _1
): Dummy[L] = ???

error: illegal dependent method type: parameter may
only be
       referenced in a subsequent parameter section
  f: First[L]
  ^
```

```
trait First[L <: List] { type Out }
```

```
object First {
  def apply[L <: List](implicit f: First[L]): Aux[L,
f.Out] = f

^^^^^^^^^^^^^
  type Aux[L <: List, H] = First[L] { type Out = H }
  implicit def f[H, T <: List]: Aux[Cons[H, T], H] =
???
}
```

...

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

```
implicit def dummy[L <: List, H](
  implicit
  f: First.Aux[L, H],
  eq: H =:= _1
): Dummy[L] = ???
```

...

```
scala> :t implicitly[Dummy[_1 :: Nil]]
Dummy[_1 :: Nil]

scala> :t implicitly[Dummy[_2 :: Nil]]
error: could not find implicit value for
       parameter e: Dummy[_2 :: Nil]
       implicitly[Dummy[_2 :: Nil]]
```

```scala
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]]]
```

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" seems to be scalac's version of ¯|(ツ)/¯"

(c) Daniel Spiewak

"I found that the easiest way to figure out what the divergence checker was doing was just to throw some printlns into the compiler and publish it locally"

(c) Travis Brown

```
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]]]
```

```
[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]]]
[error]              ^
[error] one error found
[error] (compile:compileIncremental) Compilation
failed
```

```
import shapeless._

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

implicit def a0[A, B](implicit a: Lazy[T[A]], b:
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]]]
```

What to do with all this

- programming with dependent types
- typeclass derivations

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"

Questions

This talk repo

Solution of N queens problem on type level

The Type Astronaut's Guide to Shapeless

Hacking on scalac — 0 to PR in an hour

Typing the technical interview

Patat - Presentations Atop The ANSI Terminal