Shape Safety in Tensor Programming is Easy for a Theorem Prover

Project: https://github.com/tribbloid/shapesafe

Content: https://github.com/tribbloid/shapesafe-demo

-- Peng Cheng - tribbloid@{github | twitter}

About Me 

Maintainer of DataPassports, Computing Engine ... ***2015*** **~**

- Only read Scala 'cause of Apache Spark
- Only read type theory 'cause math seems more stable than programming
- Not a Python fan, but only use PyTorch 'cause it is not TensorFlow

Maintainer of splain plugin ... ***2021*** **~**

- Partially integrated with scala compiler 2.13.6+ `-Vimplicit -Vtype-diff`

Overview

- Why type-safe linear algebra?
- How to push it to extreme?
- Why scala?
- What's next?

Why type-safe linear algebra?

Type Safe Linear Algebra in Scala - Google Chrome

Type Safe Linear Algebra in Sc ×

etrain.github.io/2015/05/28/type-safe-linear-algebra-in-scala

Evan R. Sparks   Archive   Blog   Publications   Research

Type Safe Linear Algebra in Scala

Published
28 May 2015

Thanks to Scala, lately I've been appreciating type safety more and more when programming. It was an adjustment coming from Python, R, and C, but the performance benefits and the fact that I can be pretty sure that when my code compiles, it will run properly means that I can deploy code with much higher confidence.

However, there's one area of my development life where type safety hasn't done much for me – specifically numerical linear algebra and, by consequence, machine learning. In this post I'll explain what that problem is, and propose a solution to backport type safety onto linear algebra operations in Scala, in a non-intrusive way.

The Problem

Anyone who has taken a basic linear algebra class or played around with numerical code knows about dimension alignment - in python it looks like this:

```
>>> np.random.rand(2,2) * np.random.rand(3,1)
```

1 of 7   Considine, Famelis, Paull - 2019 - Kotlin ∇ A shape-safe DSL for differentiable progra...   57.0%

Kotlin∇
A shape-safe DSL for differentiable programming

Breandan Considine McGill University
Michalis Famelis Université de Montréal
Liam Paull Université de Montréal

Abstract

Kotlin is a statically-typed programming language with support for embedded domain-specific languages, asynchronous programming, and multi-platform compilation. In this work, we present an algebraically-based implementation of automatic differentiation (AD) with shape-safe tensor operations, written in pure Kotlin. Our approach differs from existing AD frameworks in that Kotlin∇ is the first shape-safe AD library fully compatible with the Java type system, requir-

1 of 6   Typesafe Abstractions for Tensor Operations
Chen - 2017 - Typesafe abstractions for tensor operations (short paper).pdf   58.6%

Typesafe Abstractions for Tensor Operations (Short Paper)

Tongfei Chen
Johns Hopkins University, USA
tongfei@cs.jhu.edu

arXiv:1710.06892v1 [cs.PL] 18 Oct 2017

Abstract

We propose a typesafe abstraction to tensors (i.e. multidimensional arrays) exploiting the type-level programming capabilities of Scala through heterogeneous lists (HList), and showcase typesafe abstractions of common tensor operations and various neural layers such as convolution or recurrent neural networks. This abstraction could lay the foundation of future typesafe deep learning frameworks that runs on Scala/JVM.

CCS Concepts • **Software and its engineering → Software libraries and repositories;**

Keywords Scala, tensor, heterogeneous list, deep learning

ACM Reference Format:
Tongfei Chen. 2017. Typesafe Abstractions for Tensor Operations (Short Paper). In *Proceedings of 8th ACM SIGPLAN International Scala Symposium (SCALA'17).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3136000.3136001

1 Introduction

Recently the machine learning community saw a surge of libraries that handle tensors. Examples include Python libraries such as NumPy [Walt et al. 2011], Theano [Theano Development Team 2016], TensorFlow [Abadi et al. 2015], PyTorch[1], DyNet [Neubig et al. 2017], or Java libraries like Nd4j[2]). These libraries provide abstractions to tensor operations on CPUs or GPUs, and some support automatic differentiation on computational graphs. For tensors in these libraries (belongs to just one type NdArray/Tensor), *specific meaning* is implicitly assigned to each of the dimensions. Correctly manipulating the different dimensions of tensors could be difficult, rendering the whole program prone to runtime errors and hard to reason or maintain: we could mistakenly added up an 1-D tensor and 2-D tensor, or performed matrix multiplication between a 2-D tensor and a 3-D tensor. These errors are only discovered at runtime, or in some libraries, even ignored because of implicit broadcasting. Programmers must keep track of the axes of the tensors themselves (usually as comments in code), rather than leveraging the type system to guide the programmers.

We would like to have a *typing mechanism* for tensors that not only encodes *the rank of the tensor*, but also encodes *what each axis means*. Scala, being a statically-checked type-safe language that is highly capable of type-level programming and DSL construction, as is demonstrated in the popular library Shapeless[3], makes it an ideal language for implementing such typeful and typesafe tensor abstractions.

We describe the design and implementation of such a typesafe abstraction that addresses the typesafety problem of other tensor libraries, and release a prototype.[4]

2 Typesafe Tensors

We propose a typesafe tensor abstraction, in which the axes are encoded by a heterogeneous list (HList) type parameter.

```
trait Tensor[D, A <: HList]
```

D is the type of the elements this tensor holds (e.g. Float, Double, etc.), whereas types in the HList type parameter A are phantom types, i.e., they only serve as labels.

Basic constructs such as scalars, vectors and matrices could be represented as follows (types A, B etc. are labels / names to axes).

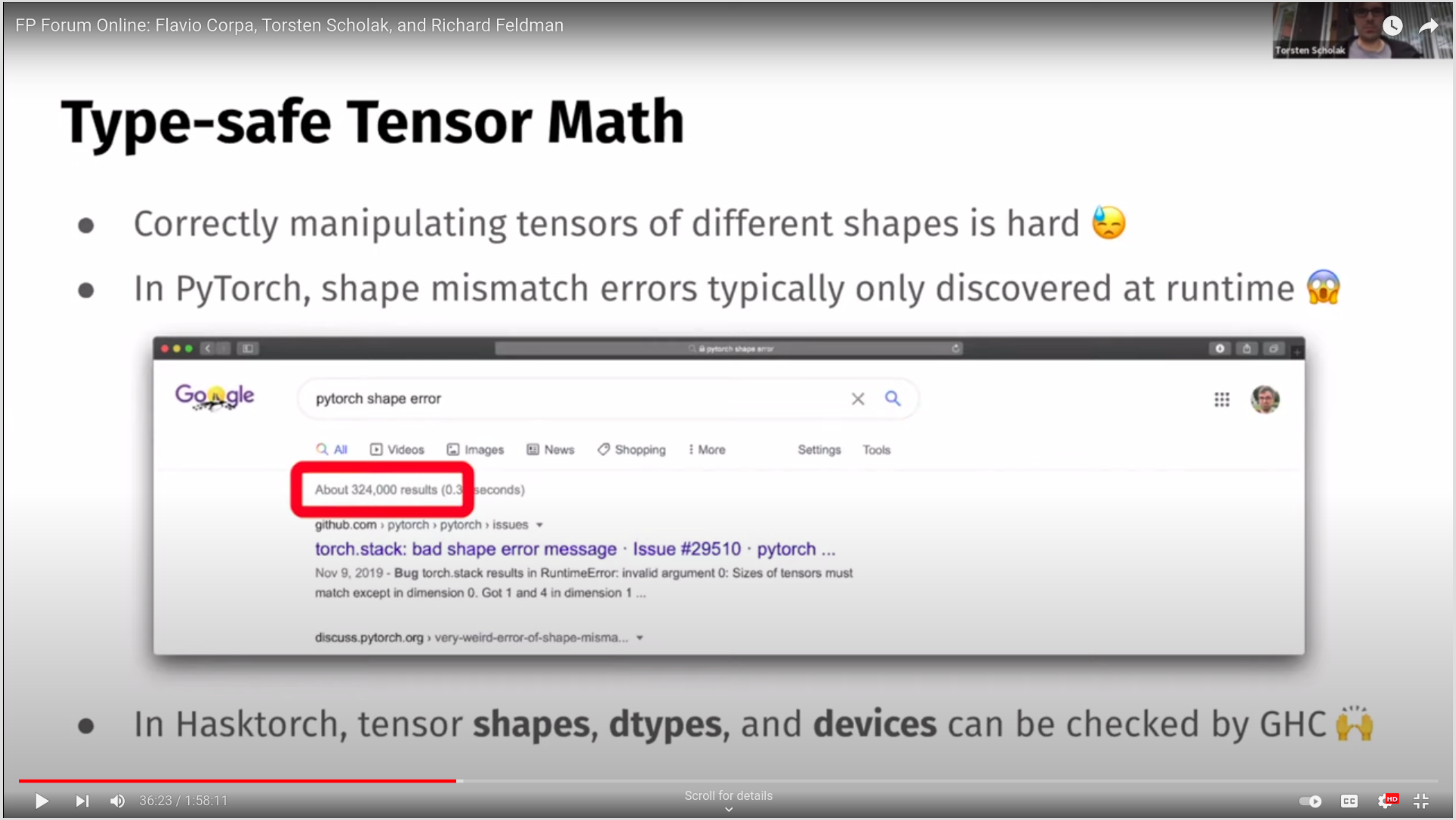
```
type Scalar = Tensor[Float, HNil]
type Vector[A] = Tensor[Float, A :: HNil]
type Matrix[A, B] = Tensor[Float, A :: B :: HNil]
```

Looking at more concrete examples from deep learning applications, images in computer vision, or sentences in which each word is mapped to a word embedding (i.e. vector representation of that word in $\mathbb{R}^d$ space) in natural language processing could be encoded as follows. (See Fig. 1)

```
type Image =
```

[1] github.com/pytorch/pytorch.
[2] github.com/deeplearning4j/nd4j.

Designed For Human (who make mistakes)



OR is it?

18th July, 2021
ICML
Long talk

Neural Architecture Search Without Training

Joseph Mellor, Jack Turner, Amos Storkey, Elliot J. Crowley.

A low-cost measure for scoring networks at initialisation which allows us to perform architecture search in a matter of seconds instead of hours or days.

19th Apr, 2021
ASPLOS
Distinguished Paper

Neural Architecture Search as Program Transformation Exploration

Jack Turner, Elliot J. Crowley, Michael O'Boyle.

A compiler-oriented approach to neural architecture search which allows us to generate new types of convolution.

6th Dec, 2020
NeurIPS
Spotlight

Bayesian Meta-Learning for the Few-Shot Setting via Deep Kernels

- (source: https://arxiv.org/abs/2102.06599)

How to push it to extreme?

Curry-Howard(-Lambek) isomorphism

proof system	$\Longleftrightarrow$	**functional programming** ***(since 1930)***
Proposition P		`type P <: Any` (in MLTT) `type P <: Prop` (in CiC)
Proposition $P(a \in A)$		`type P[_ <: A]`, `a: {type P}`
Inductive Proposition $P(a_{i+1}, P(a_i))$		`type ::[HEAD, TAIL <: P] <: P`

... with quantiifiers

proof system	$\iff$	functional programming *(since 1970)*
$\forall a \in A, P(a)$		`def p[AS <: A with Singleton](a: AS): P[AS]`, `def p[AS <: A with Singleton](a: AS): AS#P`, `def p(a: A): a.P`
$\exists b \in B, P(b)$		`def p[B]: P {val b <: B, type P}`
Axiom		`def axiom[X](.): P[X]{.}`
Theorem		`def theorem[X](., lemma1: X => X#L1): P[X]{.}`

... in shapesafe impl

<table>
<tr><th>proof system</th><th>$\Longleftrightarrow$</th><th>functional programming (after η-expansion)</th></tr>
<tr><td>$\forall a \in A, P(a)$</td><td></td><td><code>def p[AS <: A with Singleton]: AS |- P[AS]</code>,
<code>def p[AS <: A with Singleton]: AS |- AS#P</code></td></tr>
<tr><td>$\exists b \in B, P(b)$</td><td></td><td><code>def p[B]: P {val b <: B, type P}</code></td></tr>
<tr><td>Axiom</td><td></td><td><code>def axiom[X]: X |- P[X]{.}</code></td></tr>
<tr><td>Theorem</td><td></td><td><code>def theorem[X](lemma1: X => X#L1): X |- P[X]{.}</code></td></tr>
</table>

Scala 2.10 - 2015

```
class Matrix[A,B](val mat: DenseMatrix[Double]) {

  def *[C](other: Matrix[B,C]): Matrix[A,C] = new Matrix[A,C](mat*other.mat)
}

class N
class D
class K

val x = new Matrix[N,D](DenseMatrix.rand(100,10))
val y = new Matrix[N,K](DenseMatrix.rand(100,2))

val z1 = x * x //Does not compile!
val z2 = x.t * y //Compiles! Returns a Matrix[D,K]
```

Scala 3.1 - 2021

```scala
import scala.compiletime.ops.int._

class Matrix[A, B]():

  def *[C](other: Matrix[B, C]): Matrix[A, C] = new Matrix[A, C]()
  def conv[C, D](other: Matrix[C, D]): Matrix[A - C + 1, B - D + 1] =
     new Matrix[A - C + 1, B - D + 1]()

val x = new Matrix[100, 100]()
val y = new Matrix[3, 3]()
val w = x.conv(y)

val z1 = w * new Matrix[100, 1]() //Does not compile!
val z2 = w * new Matrix[98, 1]() //Compiles!
```

Extreme 1

- Weird operands (Seriously, who is going to write compile-time type evaluation for einsum?)

```
type EinSum[Matrix[N1 -> D1, N2 -> D2]] = {
  if (N1 =:= N2) {
    if (D1 =:= D2) {
      Matrix[N1 -> D1]
    }
    else {
      error(...)
    }
  }
  else {
    Matrix[N1 -> D1, N2 -> D2]
  }
}
```

Extreme 2

- Symbolic reasoning (`p.apply` + `=:=` can only go in 1 direction)

```scala
class Matrix[A, B]():

  def +(other: Matrix[A, B]): Matrix[A, B]
  def transpose: Matrix[B, A]
  def flatten: Matrix[A * B, 1]

type A <: Int
type B <: Int
val m = new Matrix[A, B]()
m.flatten + (m.transpose.flatten) // Oops
```

Learn from

- forward mode

```
lemma and_is_comm (p q: Prop) (h: p ∧ q): q ∧ p :=
    and.intro (h.right) (h.left)
```

- tactic mode

```
lemma and_is_comm' (p q: Prop) (h: p ∧ q): q ∧ p :=
begin
  apply and.intro,
  exact h.right,
  exact h.left,
end
```

(source: LEAN for Scala programmers - Part 4 - Juan Pablo Romero Méndez)

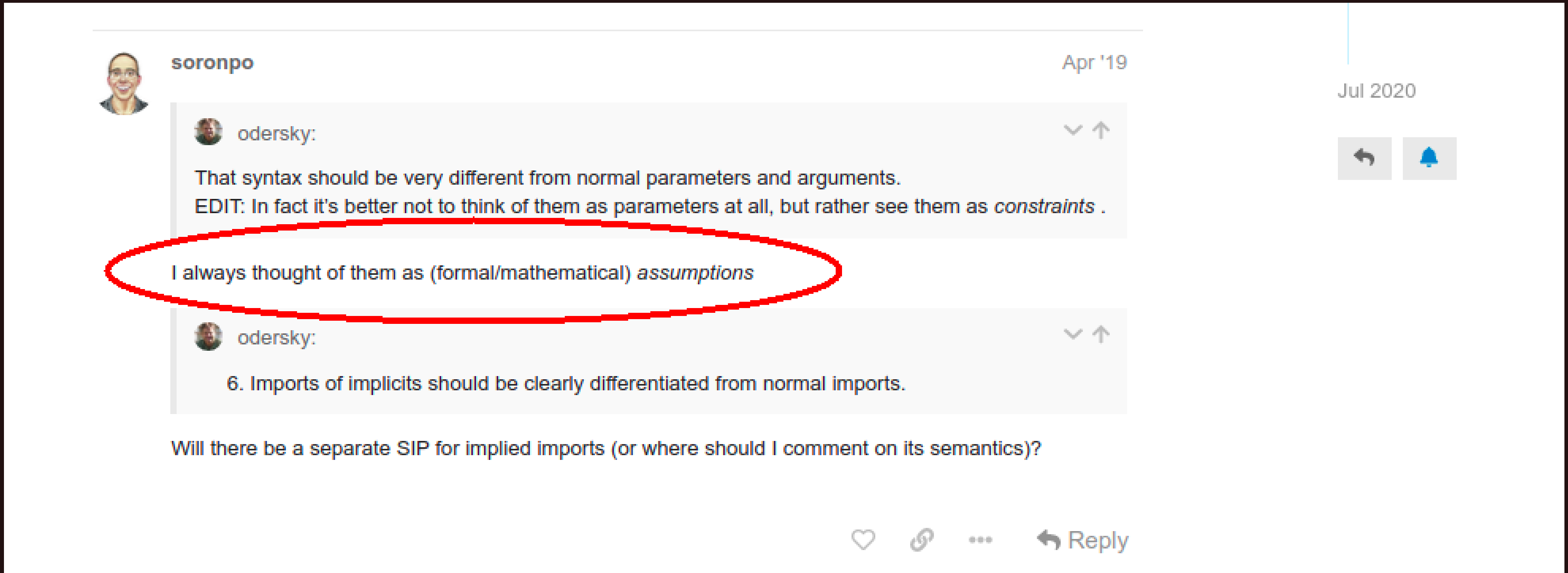
Full auto mode

```
lemma and_is_comm' (p q: Prop) (h: p ∧ q): q ∧ p :=
begin
  assumption,
  assumption,
  assumption,
end
```

=:=

```
begin
  assumption | ? => q ∧ p | .intro: (q, p)? => q ∧ p
  assumption | p ∧ q => ? | .right: p ∧ q => p?
  assumption | p ∧ q => ? | .left:  p ∧ q => q?
end
```

... in Scala!


soronpo Apr '19

odersky:

That syntax should be very different from normal parameters and arguments.
EDIT: In fact it's better not to think of them as parameters at all, but rather see them as *constraints* .

I always thought of them as (formal/mathematical) *assumptions*

odersky:

6. Imports of implicits should be clearly differentiated from normal imports.

Will there be a separate SIP for implied imports (or where should I comment on its semantics)?

Reply

Jul 2020

(source: Oron Port's response to "Principles for Implicits in Scala 3" by Odersky)

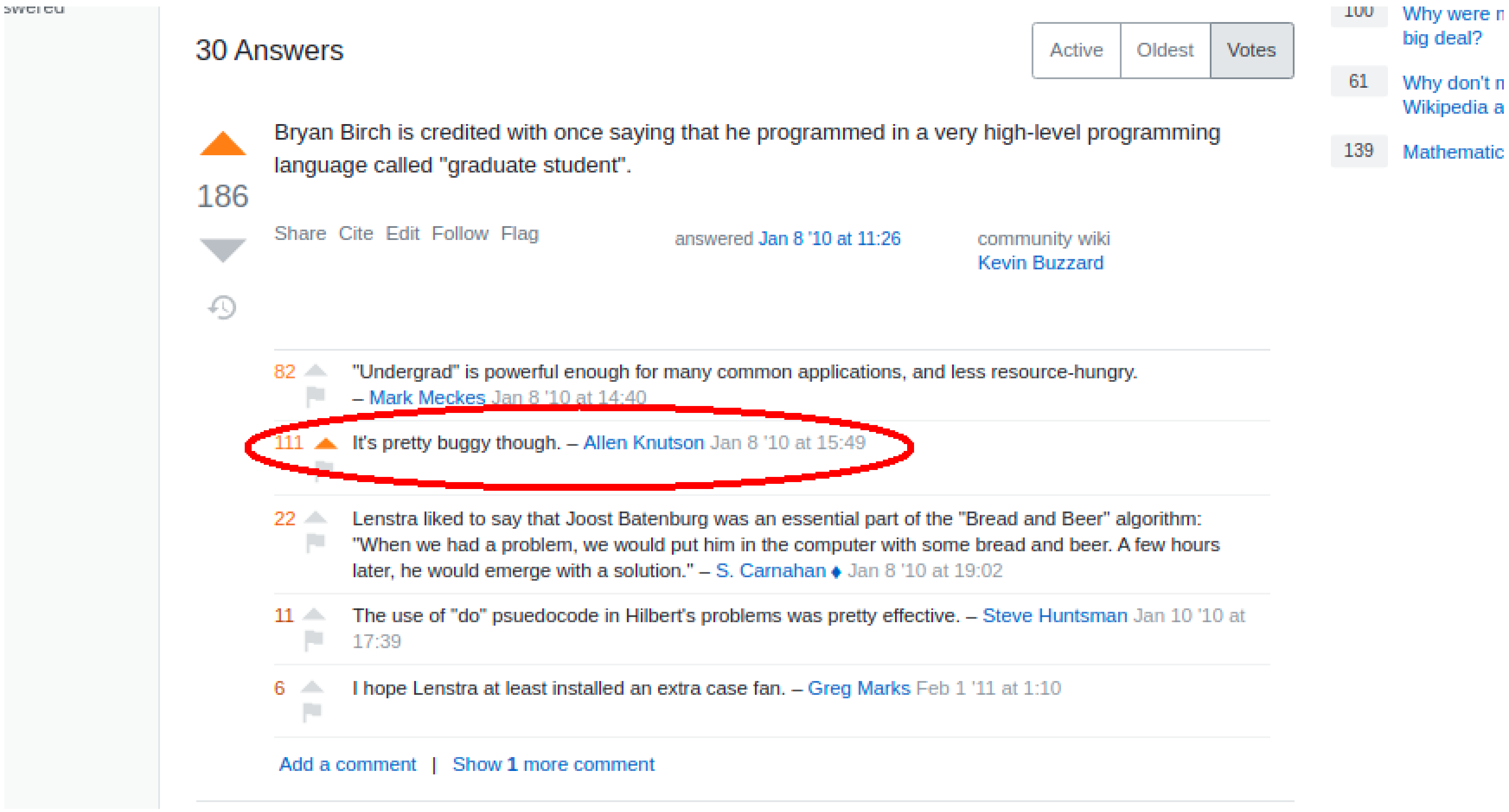
Curry-Howard isomorphism, rewritten

proof system	$\Longleftrightarrow$	functional programming (full auto!)
$\forall a \in A, P(a)$		`def p[AS <: A with Singleton]: AS |- P[AS]` , `def p[AS <: A with Singleton]: AS |- AS#P`
$\exists b \in B, P(b)$		`def p[BS <: B](implicit b: BS): BS |- BS#P`
Axiom		`def axiom[X]: X |- P[X]{.}`
Theorem		`def theorem[X](implicit lemma1: X => X#L1): X |- P[X]{.}`

<Life Coding>

Why Scala

-- Specifically, why scala 2?



swered

30 Answers

Active | Oldest | Votes

Bryan Birch is credited with once saying that he programmed in a very high-level programming language called "graduate student".

186

Share Cite Edit Follow Flag

answered Jan 8 '10 at 11:26

community wiki
Kevin Buzzard

82 "Undergrad" is powerful enough for many common applications, and less resource-hungry. – Mark Meckes Jan 8 '10 at 14:40

111 It's pretty buggy though. – Allen Knutson Jan 8 '10 at 15:49

22 Lenstra liked to say that Joost Batenburg was an essential part of the "Bread and Beer" algorithm: "When we had a problem, we would put him in the computer with some bread and beer. A few hours later, he would emerge with a solution." – S. Carnahan ♦ Jan 8 '10 at 19:02

11 The use of "do" psuedocode in Hilbert's problems was pretty effective. – Steve Huntsman Jan 10 '10 at 17:39

6 I hope Lenstra at least installed an extra case fan. – Greg Marks Feb 1 '11 at 1:10

Add a comment | Show 1 more comment

100 Why were r
big deal?

61 Why don't r
Wikipedia a

139 Mathematic

Stack of AI-HPC-Hardware co-evolution

/\ More Abstract

- Distributed Computing --- Apache {Spark | Flink}
- [*SOME Deep Learning Library?*]
- IR/Multi-stage Compilation --- reflection, LMS
- Hardware Design --- CHISEL / SpinalHDL

\/ More Concrete

But try to avoid ...

- Diverging implicit
  - Circular reference `def theorem[A](using a: A): A`
  - Expanding reference `def theorem[A <: F[_], B](using a: F[A, B]): A`
- Summoning proof on long algebraic data type

```
SquashByName[ / ]#On[
  SquashByName[ / ]#On[
    SquashByName[ + ]#On[
      SquashByName[ - ]#On[
        SquashByName[ / ]#On[
          SquashByName[ + ]#On[
            SquashByName[ - ]#On[
              SquashByName[ / ]#On[
                SquashByName[ + ]#On[
                  SquashByName[ - ]#On[
                    1024 >< 1024 |<<- (i >< j) >< (3 >< 3 |<<- (i >< j))...
```

What's next?

- Gradually typed (`UncheckedShape`), but still no symbolic reasoning
- Fairly stable, but not released or in production
- Not a computing library! Cannot empower ML engineers directly, but serve as a bridge to get there
- Need Scala 3 support!

Moving to Scala 3?

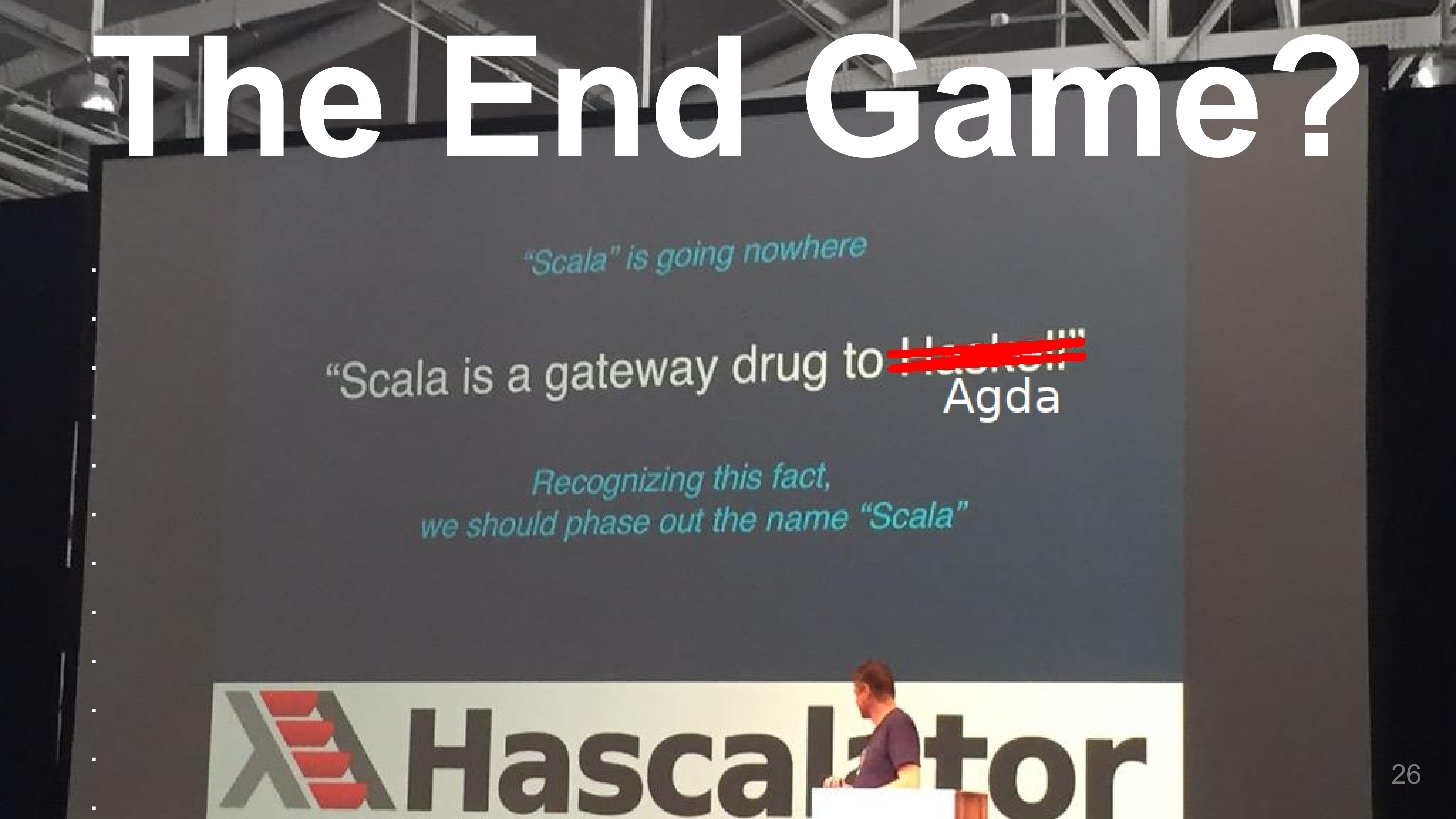
Pros

- Implicit search tree traversal is aggressively cached
- No more unpredictable diverging implicit recovery
- Theoretically sound resolving of `<:<` and `=:=`

Cons

- No type projection (may be added back)
- No shapeless Record type
  (may be integrated into *Programmatic Structural Types*)
  (can be expressed in match type in a very inefficient way)

The End Game?


"Scala" is going nowhere
"Scala is a gateway drug to ~~Haskell~~"
Agda
Recognizing this fact,
we should phase out the name "Scala"

Proudly Supported By

DataPassports ™, the Data Privacy & Trust Company

- We are hiring! Looking for a director of engineering

Splain Open-Source Team

- Looking for scala compiler guru / type theorist as technical advisor
- 1.0.0RC is close, looking for test users