Juan Manuel Serrano Hidalgo

Embedding of external DSLs in Scala: why and how!

lambda DAYS
27-28 MAY 2024
KRAKÓW | POLAND

HABLA COMPUTING

https://www.hablapps.com

Universidad Rey Juan Carlos

https://github.com/jserranohidalgo/urjc-pd

@juanshac | juanmanuel.serrano@hablapps.com | urjc.es

ScalaMAD: Scala Programming @ Madrid

Madrid, Spain

2,334 members · Public group

Organized by **Juan Manuel Serrano** and **4 others**

https://www.meetup.com/Scala-Programming-Madrid/

LAMBDA WORLD

We are the largest Functional programming event in South Europe.

2 - 4 October 2024. Cadiz, Spain

ORGANISED BY

https://lambda.world

Goals

- Why is declarative programming and **embedded DSLs**, in particular, so important? A case study with `jq` and `Scala`

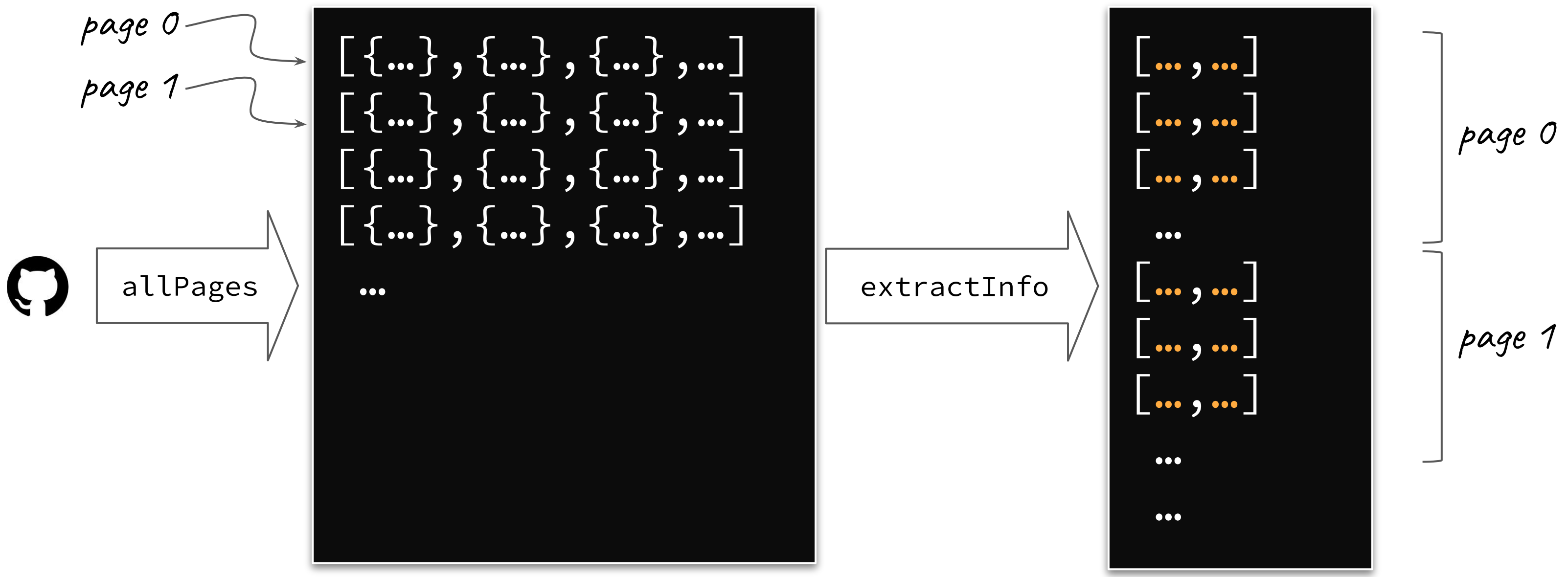
- How to proceed? Leverage **tagless-final** and its step-by-step methodology

- Which new features of **Scala 3** can we exploit for implementing embedded DSLs?

HABLA COMPUTING

```
> curl https://api.github.com/repos/hablapps/doric/commits?page=0
```


```
[
  {
    "sha": "0421e65de2a91d6ec4c95d940f8c11e31e0051e9",
    "node_id": "C_kwDOFM9sW9oAKDA0MjFlNjVkZTJhOTFkNmVjNGM5NWQ5NDBmOGMxMWUzMWUwMDUxZTk",
    "commit": {
      "author": {
        "name": "Alfonso Roa",
        "email": "alfonso.roa@habla.dev",
        "date": "2023-05-16T15:34:51Z"
      },
      "committer": {
        "name": "GitHub",
        "email": "noreply@github.com",
        "date": "2023-05-16T15:34:51Z"
      },
      "message": "Documentation update to show new version (#356)",
      "tree": {
        "sha": "e12ca0921f1538cad98c820029c566bcfa5dd4b9",
        "url": "https://api.github.com/repos/hablapps/doric/git/trees/e12ca0921f1538cad98c820029c566bcfa5dd4b9
      },
      "url": "https://api.github.com/repos/hablapps/doric/git/commits/0421e65de2a91d6ec4c95d940f8c11e31e0051e9
      "comment_count": 0,
      "verification": {
        "verified": true,
        "reason": "valid",
        "signature": "-----BEGIN PGP SIGNATURE-----\n\nwsBcBAABCAAQBQJkY6KbCRBK7hj4Ov3rIwAADVUIAFuOp/iO3EpsU8G
v0yeCHkjKJgL7WVNY+lLarXcd70RuHG3xDKEEEKNsEauVZT\n/u4+KE+Y/60jewwrd8JLFSMSAA27U0ovmVSnHZ/HHcSARwSyjJjcsUt9Ngu2j
        "payload": "tree e12ca0921f1538cad98c820029c566bcfa5dd4b9\nparent de16314ab9db01237fda4735fe1b10186c55
rsion (#356)\n\n"
      }
    },
    "url": "https://api.github.com/repos/hablapps/doric/commits/0421e65de2a91d6ec4c95d940f8c11e31e0051e9",
    "html_url": "https://github.com/hablapps/doric/commit/0421e65de2a91d6ec4c95d940f8c11e31e0051e9",
    "comments_url": "https://api.github.com/repos/hablapps/doric/commits/0421e65de2a91d6ec4c95d940f8c11e31e005
    "author": {
      "login": "alfonsorr",
      "id": 15030451,
      "node_id": "MDQ6VXNlcjE1MDMwNDUx",
      "avatar_url": "https://avatars.githubusercontent.com/u/15030451?v=4",
      "gravatar_id": "",
      "url": "https://api.github.com/users/alfonsorr",
      "html_url": "https://github.com/alfonsorr",
      "followers_url": "https://api.github.com/users/alfonsorr/followers",
      "following_url": "https://api.github.com/users/alfonsorr/following{/other_user}",
      "gists_url": "https://api.github.com/users/alfonsorr/gists{/gist_id}",
```



page 0
page 1
allPages
[{…},{…},{…},…]
[{…},{…},{…},…]
[{…},{…},{…},…]
[{…},{…},{…},…]
…
extractInfo
[…,…]
[…,…]
[…,…]
…
[…,…]
[…,…]
[…,…]
…
…
page 0
page 1

FS2

circe

```scala
def allPages[F[_]: Async](repo: String): Stream[F, Json] =

  def newPage(i: Int): Stream[F, Json] =
    stream(Uri.unsafeFromString(s"$repo/commits?page=$i"))

  def go(i: Int, s: Stream[F,Json]): Pull[F, Json, Unit] =
    s.pull.uncons.flatMap:
       case Some((hd,tl)) =>
         hd(0) match
           case IsArray(Vector()) => Pull.done
           case _ => Pull.output(hd) >> go(i+1, tl ++ newPage(i))
       case None => Pull.done

  go(1, newPage(0)).stream
```

HABLA COMPUTING

FS2

circe

```
def extractInfo[F[_]]: Stream[F, Json] => Stream[F, Json] =
    ???
```

FS2 + circe

```
def extractInfo[F[_]]: Pipe[F, Json, Json] =
    ???
```

FS2

```
def extractInfo[F[_]]: Pipe[F, Json, Json] =
    _.flatMap: page =>
         Stream(
             (root.each.author.login.json.getAll(page) zip
              root.each.commit.author.date.json.getAll(page))
                 .map(Json.arr(_, _))*
         )
```

./jq (a JSON **q**uery language)

Get the name ***of*** the author
of the commit (***index***)

and the login ***of*** the author

```
def extractInfo:
  .[] | [.commit.author.date, .author.login]
```

Iterate over
the commits
of the array …

and for each commit …
(***pipe***)

And collect that into an
array

HABLA COMPUTING

FS2 + CIRCE + Monocle

General purpose
Interoperability
…

./jq

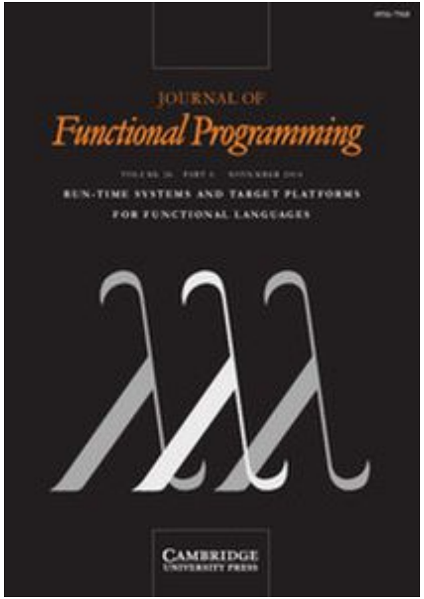
Simplicity
Conciseness
Understandability
…

FS2 + CIRCE + ./jq

```
def extractInfo[F[_]]: Pipe[F, Json, Json] =
  //   .[] |    [ .commit.author.date,   .author.login ]
  iterator | arr(i“commit.author.date”, i“author.login”)
```

Finally tagless, partially evaluated: Tagless staged interpreters for simpler typed languages

Oleg Kiselyov et al.



1 Fixed Domain

2 Domain Abstraction

3 Dynamic typing

4 Static typing

HABLA COMPUTING

```scala
def extractInfo[F[_]]: Pipe[F, Json, Json] =
  iterator | arr(i"author.login", i"commit.author.date")
```

Syntax

```scala
class Fs2_Jq[F[_]]:

    type Filter = Pipe[F, Json, Json]

    def iterator: Filter = ???

    extension (f1: Filter)
        def |(f2: Filter): Filter = ???

    def arr(filters: Filter*): Filter = ???

    extension (sc: StringContext)
        def i(args: Filter*): Filter = ???
```

HABLA COMPUTING

```scala
def extractInfo[F[_]]: Pipe[F, Json, Json] =
  iterator | arr(i"author.login", i"commit.author.date")
```

```scala
class Fs2_Jq[F[_]]:

    type Filter = fs2.Pipe[F, Json, Json]

    def iterator: Filter =
        _.flatMap:
            case IsArray(v) => Stream(v*)
            case _ => ??? // TBD

    ...
```

HABLA COMPUTING

jq▶play [Try it!]

```
.author.login
```

```
.author | .login
```

```
.[“author”] | .[“login”]
```

Identity filter

String constant filter

jq▶play [Try it!]

```
[.author.login, .commit.author.date]
```

concatenation filter

Deep syntax

Surface syntax

```
class Fs2_Jq[F[_]]:

    /* Primitives */

     def concat(f1: Filter, f2: Filter): Filter = ???
     def array(f: Filter): Filter = ???
     …

    /* Syntactic sugar */

     def arr(f: Filter*): Filter =
        array(f.reduce(_ concat _))


     …
```

jq▶play [Try it!]

```
def map(f): [.[] | f]
```

Built-in functions

Deep syntax

Standard lib

Surface syntax

```
class Fs2_Jq[F[_]]:

    /* Primitives */
     …

    /* Built-in functions */

     def map(f: Filter): Filter =
         array(iterator | f)
     …

    /* Syntactic sugar */
     …
```

```
def extractInfo: Flow[Json, Json, NotUsed] =
    iterator | arr(i"author.login", i"commit.author.date")
```

```
object Akka_Jq:
    type Filter = Flow[Json, Json, akka.NotUsed]

    /* Primitive filters */
    def id: Filter = ???
    def str(s: String): Filter = ???
    def iterator: Filter = ???
    …

    /* derived functions */
    def map(f: Filter): Filter = …

    /* Syntactic sugar */
    def arr(f: Filter*): Filter = …
    extension (sc: StringContext)
        def i(args: Filter*): Filter = …
```

1

Fixed domain

2

Domain abstraction

3

Dynamic typing

4

Static typing

Class of Jq Representations

Semantic domain

Deep syntax

```
trait Jq[R]:

    /* primitive filters */

     def id: R
     def str(s: String): R
     def iterator: R
     def array(f: R): R

     extension (f1: R)
        def |(f2: R): R
        def concat(f2: R): R
        def index(f2: R): R

    /* built-in functions */

     def map(f: R): R =
        array(iterator | f)
```

Boilerplate :(

Surface syntax

```
object syntax:

  def iterator[R: Jq]: R = Jq[R].iterator

  ...

  def arr[R](f: R*)(using J: Jq[R]): R = …

  extension [R](sc: StringContext)(using J: Jq[R])
    def i(args: R*): R = …
```

Semantics as given instances

```
object Fs2:

    type Filter[F[_]] = Pipe[F, Json, Json]

    given [F[_]]: Jq[Filter[F]] with
        def id: Filter[F] = …
        def str(s: String): Filter[F] = …
        …
```

Programs for particular domains

```
import Fs2.given, Jq.syntax._

def extractInfo[F[_]]: Pipe[F, Json, Json] =
  iterator | arr(i"author.login", i"commit.author.date")
```

Generic programs as ad-hoc polymorphic functions

```
import Jq.syntax._

def extractInfo[R: Jq]: R =
    iterator | arr(i"author.login", i"commit.author.date")
```

Polymorphic & context function types

Generic programs as values

```
val extractInfo: [R] => Jq[R] ?=> R =
    [R] => (_ : Jq[R]) ?=>
        iterator | arr(i"author.login", i"commit.author.date")
```

HABLA COMPUTING

1

Semantic functions

2

Full tagless-final

3

Dynamic typing

4

Static typing

jq▶play [Try it!]

```
> jq '.[]'
```

*Will encounter **errors** when fed with non-arrays (or non-objects)*

HABLA COMPUTING

jq▶play [Try it!]

And we can catch errors

```
try error("halt") catch .
```

We can throw custom errors

```scala
trait Jq[R]:

    def error(msg: String): R
    ...

    extension (f1: R)
        def `catch`(f2: R): R

object Jq:

    enum Error:
        case CannotIterateOver(j: Json)
        case CannotIndexObjectWith(j: Json)
        case CannotIndex(j: Json)
        case Custom(msg: String = "")
```

Possible errors

```scala
object Fs2:

    type Filter[F[_]] =
        Pipe[F, io.circe.Json, io.circe.Json | Jq.Error]

    given [F[_]]: Jq[Filter[F]] with

        def iterator: Filter[F] =
            _.flatMap:
                case IsArray(v) => Stream(v*)
                case j => Stream(Jq.Error.CannotIterateOver(j))

        def error(msg: String): Filter[F] =
            _.map: _ =>
                Jq.Error.Custom(msg)

         ...
```

jq▶play [Try it!]

We **may** run into errors

```
.[]
```

```
~~"hi"~~ | .[]
```

We **will always** run into errors: malformed programs

HABLA COMPUTING

1

Semantic functions

2

Full tagless-final

3

Dynamic typing

4

Static typing

HABLA COMPUTING

A filter that outputs **strings**

```
“hi” | .[]
```

A combinator that takes the output of the first filter and feeds it into the input of the second one, provided they **match**

A filter that is applied to input **arrays**

$$\frac{\texttt{s : String}}{\texttt{s : Filter[A, JString]}}$$

JString *does **not** conform to* JArray!

“hi” | .[]

$$\frac{\texttt{f1: Filter[A, B]} \quad \texttt{f2: Filter[B, C]}}{\texttt{f1 | f2 : Filter[A, C]}}$$

$$\frac{}{\texttt{.[] : Filter[JArray[A], A]}}$$

---

f1 : Filter[Json, **JString**]

JString *does conform to* JSon!

f1 | f2

f1: Filter[A, **B**] **f2**: Filter[**B**, C]

---

f1 | f2 : Filter[A, C]

---

f2 : Filter[**Json**, B]

HABLA COMPUTING

Phantom type

```scala
sealed trait Filter[-I <: JsonT, +O <: JsonT]

enum JsonT:
    case JNullT()
    case JBoolT()
    case JNumberT()
    case JStringT()
    case JArrayT[+t <: JsonT]()
    case JObjectT()
```

$$\frac{\mathbf{s} : \texttt{String}}{\mathbf{s} : \texttt{Filter[A, } \textbf{JString}\texttt{]}}$$

$$\frac{}{\mathbf{.[]} : \texttt{Filter[}\textbf{JArray[A]}\texttt{, A]}}$$

```
trait Jq[R[+_]]:

    def str[A <: JsonT](s: String): R[Filter[A, JStringT]]

    def iterator[A <: JsonT]: R[Filter[JArrayT[A], A]]

    extension [A <: JsonT, B <: JsonT](f1: R[Filter[A, B]])
        def |[C <: JsonT](f2: R[Filter[B, C]]): R[Filter[A, C]]

    ...
```

$$\frac{\mathbf{f1}\texttt{: Filter[A, } \textbf{B}\texttt{]} \quad \mathbf{f2}\texttt{: Filter[}\textbf{B}\texttt{, C]}}{\mathbf{f1\ |\ f2} : \texttt{Filter[A, C]}}$$

HABLA COMPUTING

```
“hi” | .[]
```

```
def program[R[+_]: Jq] =
    str("1") | iterator
```

```
-- [E007] Type Mismatch Error: ----------------------------
   |    str("1") | iterator
   |               ^^^^^^^^
   |  Found:    R[Filter[JArrayT[JsonT], Nothing]]
   |  Required: R[Filter[JStringT, JsonT]]
```

HABLA COMPUTING

Constant type constructor, as a type lambda

```
object Fs2:

    type Filter[F[_]] =
        Pipe[F, io.circe.Json, io.circe.Json | Jq.Error]

    given [F[_]]: Jq[[_] =>> Filter[F]] with

        def id[A <: JsonT]: Filter[F] = …
        def str[A <: JsonT](s: String): Filter[F] = …
        def iterator[A <: JsonT]: Filter[F] = …
        ...
```

Type erasure

jq▶play [Try it!]

We can index **objects** with **strings**

```
.["key"]
```

We can index **arrays** with **integers**

```
.[1]
```

Ill-typed program!

```
.[true]
```

We should be able to index over arrays as well

```
trait Jq[R[+_]]:

    extension [A <: JsonT](
        f1: R[Filter[A, JObjectT]]
    )
        def index(f2: R[Filter[A, JStringT]]): R[Filter[A, JsonT]]

    ...
```

In case that we are indexing arrays, the key should be an integer

If indexing arrays, the output type should match the type of its elements

```scala
trait Jq[R[+_]]:

    extension [A <: JsonT, t <: JsonT, B <: JArrayT[T] | JObjectT](
        f1: R[Filter[A, B]]
    )
        def index(f2: R[Filter[A, IndexType[B]]]):
           R[Filter[A, OutputType[B]]]

    ...
```

```scala
type OutputType[T1 <: JsonT] <: JsonT =
  T1 match
    case JArrayT[t] => t
    case JObjectT => JsonT
```

```scala
type IndexType[T <: JsonT] <: JsonT =
  T match
    case JObjectT => JStringT
    case JArrayT[_] => JNumberT
```

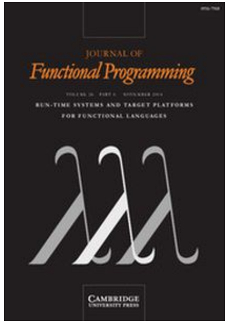
HABLA COMPUTING

Conclusion

- Why is **declarative programming** so important?
  - Because they allow us to use higher-level abstractions
- Why are **embedded DSLs** so important?
  - Because they allow us to exploit higher-level abstractions in the context of a general purpose environment

Conclusion

- How to proceed? Leverage **tagless-final** and its step-by-step methodology
  - Proceed step-by-step, starting from fixed semantics, then to type-classes, and finally to type systems
  - Apply the KISS principle
  - Ultimate goal: functional architectures built on DSLs



HABLA COMPUTING

Conclusion

- Which new features of **Scala 3** can we exploit?

Extension methods
Conversion *type class*

1 Fixed Domain

Type classes,
polymorphic function types,
context function types, ...

2 Domain Abstraction

3 Dynamic typing

Union types,
enums

4 Static typing

Match types,
Type lambdas

Conclusion

- Which new features of **Scala 3** can we exploit?
    - Going further: Scala 3 macros & structural types

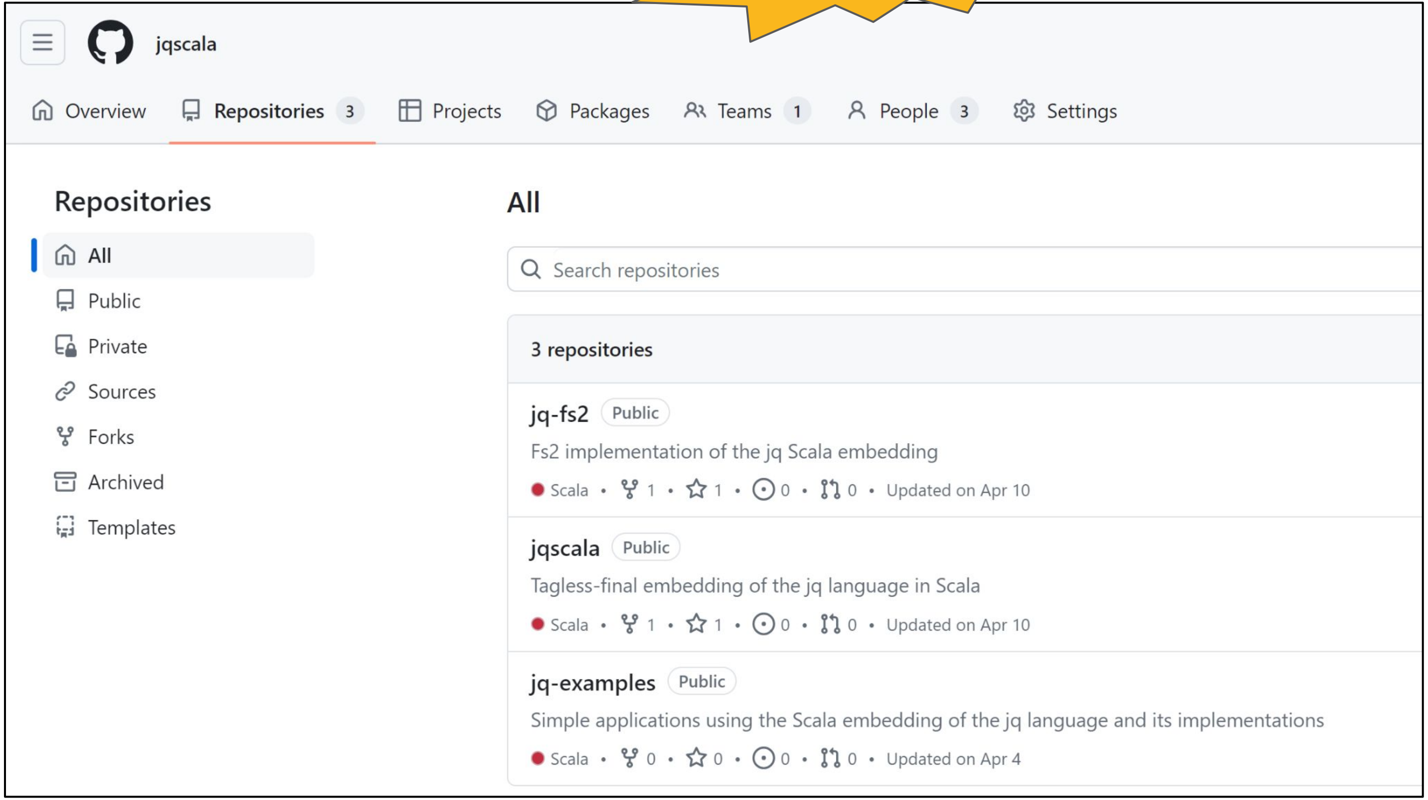
```
def extractInfo[R: Jq]: R =
    iterator | arr(id.author.login, id.commit.author.date)
```

```
def extractInfo[R: Jq]: R =
    obj(“a” -> 1.int, “b” -> 2.int).c
```

HABLA COMPUTING

References

Contributions welcomed!



HABLA COMPUTING

References

http://okmij.org/ftp/

Recent changes
June, 1 2020

rss

Shortcuts

- BER MetaOCaml
- tagless-final
- extensible effects
- streams
- Iteratee, Enumerator
- Freer monad
- Shonan Challenge
- strymonas
- simple generators
- LogicT
- shift tutorial
- generators, yield
- delimcc
- call/cc

Computation
fixpoints; CK macros; Having an Effect; monads; programming as collaborative reference; Turing machines; IO monad realized in 1965; ...

Types
type arithmetic's; lightweight static guarantees; Hindley-Milner type inference course; unusual polymorphism; eliminating existentials; ...

Meta-programming
staging; typed compilation; language-integrated query; Staged Haskell; BER MetaOCaml; HPC; generating Gaussian Eliminators; FFT; stencil; ...

Continuations
implementations; tutorials; shift/reset in CBV and CBN; `control/prompt`; delimited and undelimited; generators; zipper; call/cc; deriving recursion from iteration; delimited dynamic binding; ...

Logic
logical Frameworks; Twelf; impredicativity; strengthening in logical frameworks; eigen-variables: variables or constants?; type soundness proofs for calculi with delimited control; recursively enumerating binary arithmetic relations; ...

Algorithms and Data Structures
pretty-printing; arithmetic compression; scheduling; transforming cyclical structures; shuffling; the Credit Card transform; secure counting; proving correctness of algorithms; tree annotation; beyond Church encoding; Boehm-Berarducci isomorphism

Programming Languag
Haskell; ML; Scheme; Prolog

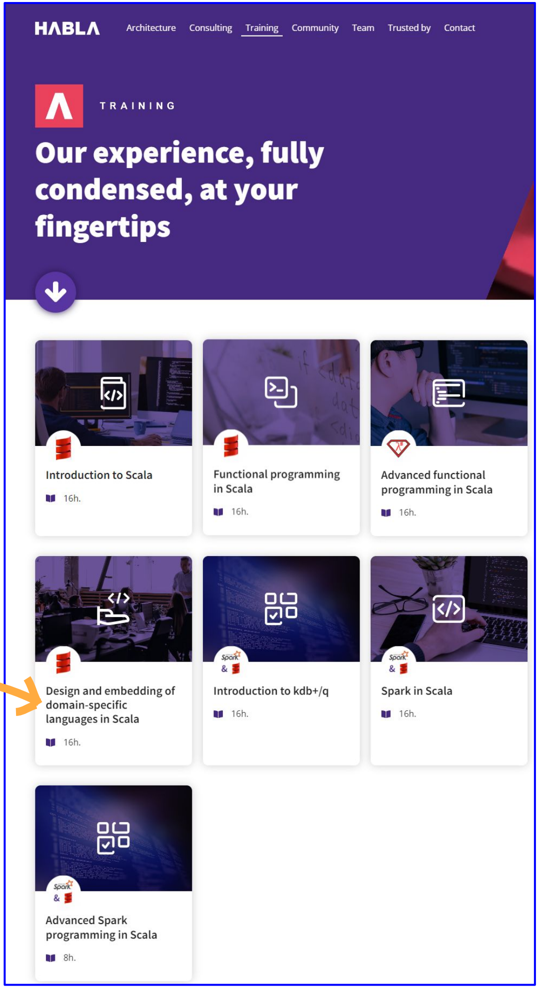
Haskell
logical type programming; de
monads; regions; type-level e

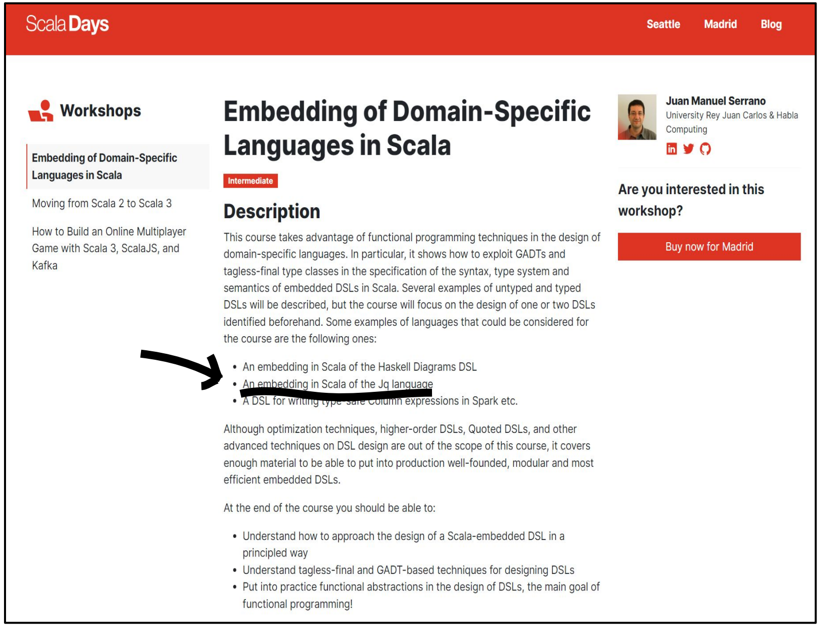
Lambda-calculus
calculators; multiple predece
...

ML
music of streams; code gener
typeclasses; generators; ...

Scheme
XML; Web; macros; text and binary parsing; utilities; database interfaces; papers; ...

XML
SXML; Iteratee parsers; SSAX; parsing; SXSLT; SXPath; typed SXML; ...





HABLA COMPUTING

References

hablapps / **tagless-final-tutorial**

*Throughout example of tagless-final vs. **MTL***

hablapps / **lambdas**

Tagless-final utilities

hablapps / **doric**

*A fixed-domain, **typed** extension of Spark column DataFrames*

References

- Jq:
  - **https://jqlang.github.io/jq/**
- Tomas Mikula's libretto: illustrates an alternative to tagless-final
  - https://github.com/TomasMikula/libretto
- Optics as an alternative to jq:
  - **https://chrispenner.ca/posts/traversal-systems**
- "Some Proposed Changes for Better Support of Type Classes"
  - https://github.com/dotty-staging/dotty/blob/typeclass-experiments/docs/_docs/reference/experimental/typeclasses.md

lambda DAλS

27-28 MAY 2024

KRAKÓW | POLAND

Thanks for your attention!

Juan M. Serrano

@juanshac | juanmanuel.serrano@hablapps.com | urjc.es