Declarative Programming with Algebra

André van Delft
Anatoliy Kmetyuk

FLOPS 2016, Kochi Japan
4 March 2016

Overview

- Introduction
  - Programming is Still Hard
  - Some History
  - Algebra of Communicating Processes
- SubScript
  - GUI application
  - DB query execution
  - Twitter Client
- Conclusion

Programming is Still Hard

Mainstream programming languages: imperative

- good in batch processing
- not good in parsing, concurrency, event handling
- Callback Hell

Neglected idioms

- Non-imperative choice: BNF, YACC
- Data flow: Unix pipes

Math!

Algebra of Communicating Processes - 1

Bergstra & Klop, Amsterdam, 1982 - ...

ACP~ Boolean Algebra

+	choice
·	sequence
0	deadlock
1	empty process

atomic actions a,b,…
parallelism
communication
disruption, interruption
time, space, probabilities
money
...

Algebra of Communicating Processes - 2

Less known than CSP, CCS

Specification & Verification

- Communication Protocols
- Production Plants
- Railways
- Coins and Coffee Machines
- Money and Economy

Strengths

- Familiar syntax
- Precise semantics
- Reasoning by term rewriting
- Events as actions

Algebra of Communicating Processes - 3

$$
\begin{aligned}
x+y &= y+x \\
(x+y)+z &= x+(y+z) \\
x+x &= x \\
(x+y)\cdot z &= x\cdot z+y\cdot z \\
(x\cdot y)\cdot z &= x\cdot(y\cdot z)
\end{aligned}
$$

$$
\begin{aligned}
0+x &= x \\
0\cdot x &= 0 \\
1\cdot x &= x \\
x\cdot 1 &= x
\end{aligned}
$$

$$
\begin{aligned}
(x+1)\cdot y &= x\cdot y + 1\cdot y \\
&= x\cdot y + y
\end{aligned}
$$

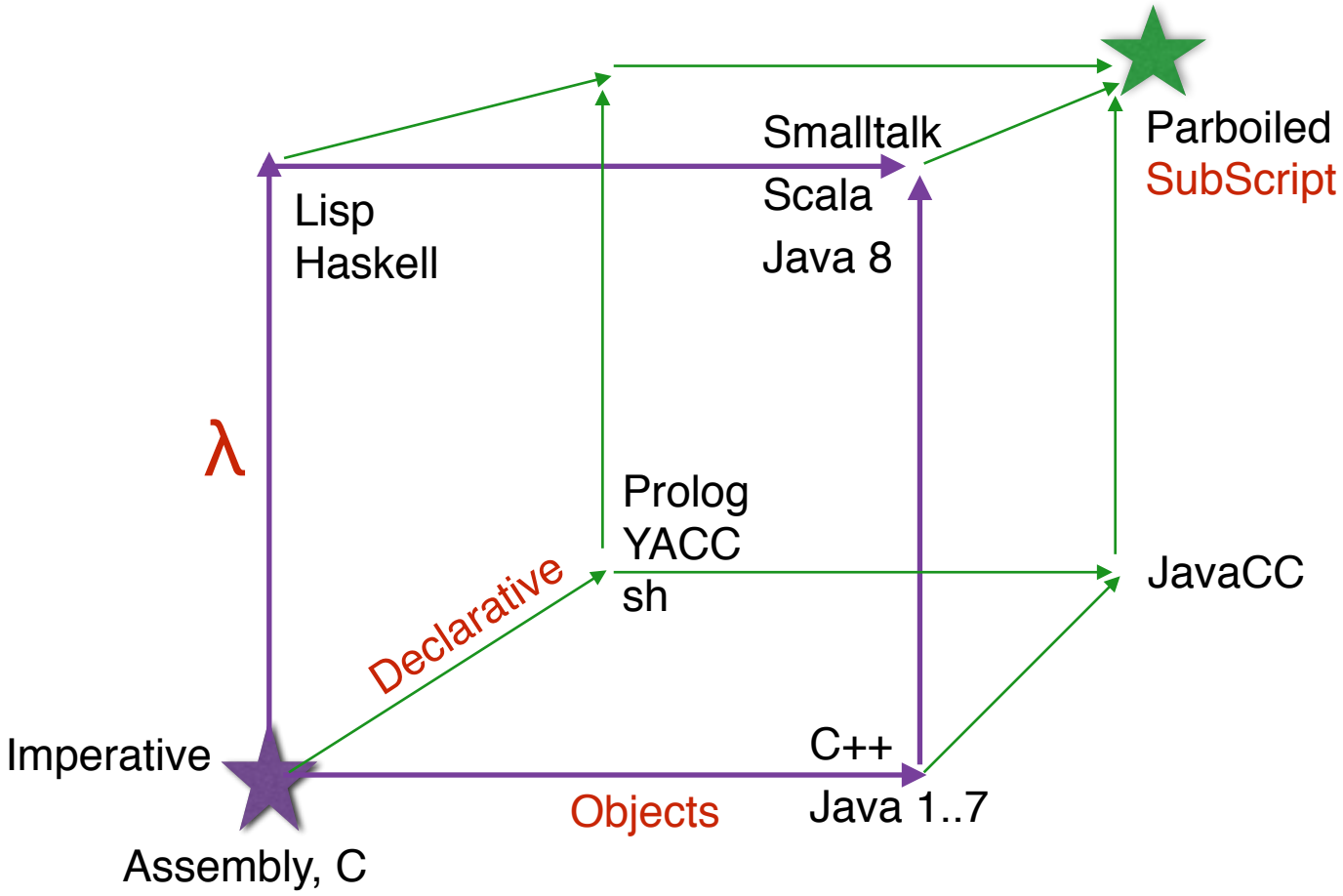
Algebra of Communicating Processes - 4

$$x \| y = x \between y + y \between x + x \mid y$$

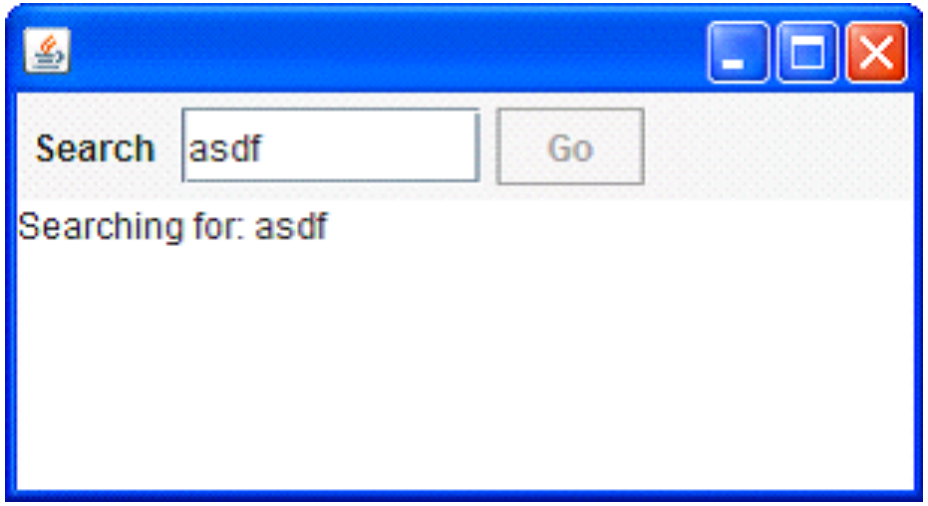
$$(x+y) \between z = \ldots$$
$$a \cdot x \between y = \ldots$$
$$1 \between x = \ldots$$
$$0 \between x = \ldots$$

$$(x+y) \mid z = \ldots$$
$$\ldots = \ldots$$

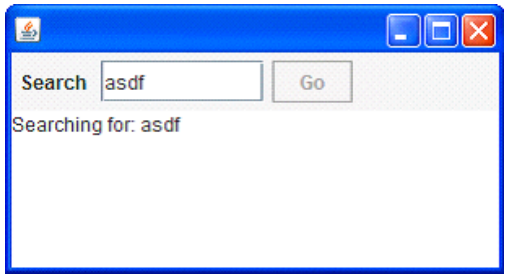
Programming Paradigms



GUI application - 1

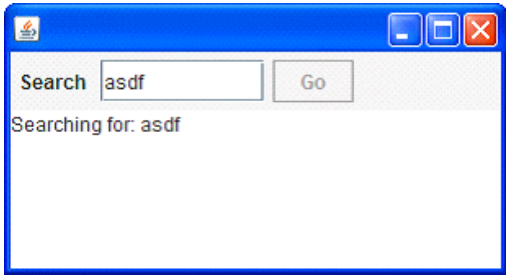


- Input Field
- Search Button
- Searching for…
- Results



GUI application - 2

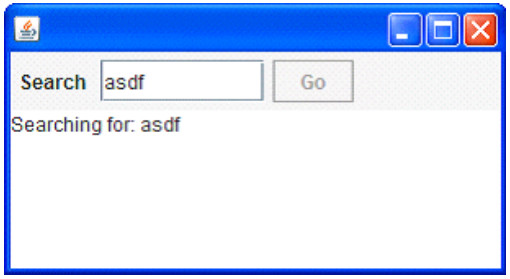
```
val searchButton = new Button("Go") {
  reactions.+= {
    case ButtonClicked(b) =>
      enabled = false
      outputTA.text = "Starting search..."
      new Thread(new Runnable {
        def run() {
          Thread.sleep(3000)
          SwingUtilities.invokeLater(new Runnable{
            def run() {outputTA.text="Search ready"
                       enabled = true
            }})
        }}).start
  }
}
```



GUI application - 3

```
live =      searchButton
       @gui: {:outputTA.text="Starting search..":}
             {* Thread.sleep(3000) *}
       @gui: {:outputTA.text="Search ready":}
             ...
```

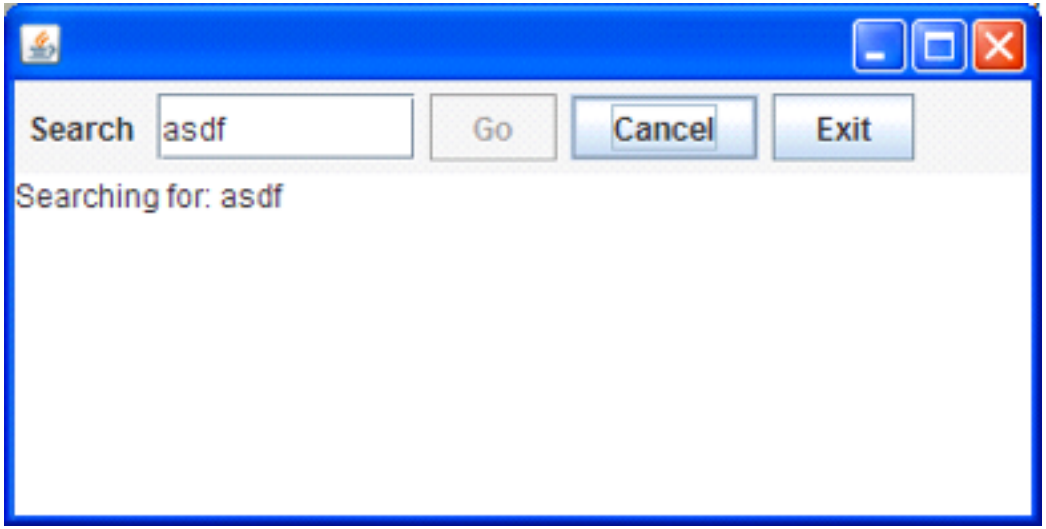
- Sequence operator: white space and `;`
- `gui`: code executor for
  - `SwingUtilities.InvokeLater+InvokeAndWait`
- `{* … *}`: by executor for `new Thread`



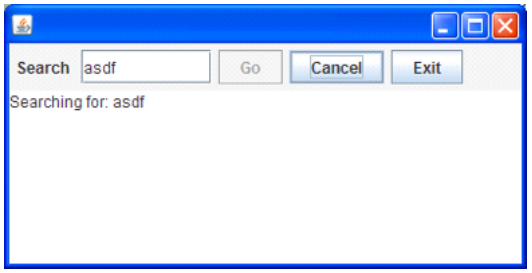
GUI application - 4

```
live              = searchSequence...

searchSequence    = searchCommand
                    showSearchingText
                    searchInDatabase
                    showSearchResults

searchCommand     = searchButton
showSearchingText = @gui: {:outputTA.text = "…":}
showSearchResults = @gui: {:outputTA.text = "…":}
searchInDatabase  = {* Thread.sleep(3000) *}
```

GUI application - 5



- Search: button or Enter key
- Cancel: button or Escape key
- Exit: button or ; ; “Are you sure?”…
- Search only allowed when input field **not** empty
- Progress indication



GUI application - 6

```
live             = searchSequence... || exit

searchCommand    = searchButton + Key.Enter
cancelCommand    = cancelButton + Key.Escape
exitCommand      =   exitButton + windowClosing
exit             =   exitCommand @gui: confirmExit ~~(b:Boolean)~~> while !b
cancelSearch     = cancelCommand @gui: showCanceledText

searchSequence   = searchGuard searchCommand
                   showSearchingText searchInDatabase showSearchResults
                   / cancelSearch

searchGuard      = if !searchTF.text.isEmpty then break? anyEvent:searchTF ...

searchInDatabase = {*Thread.sleep(3000)*} || progressMonitor
progressMonitor  = {*Thread.sleep( 250)*}
                   @gui:{:searchTF.text+=here.pass:} ...
```

SubScript Features

"Scripts" – process refinements as class members

```
script a = b; {:c:}
```

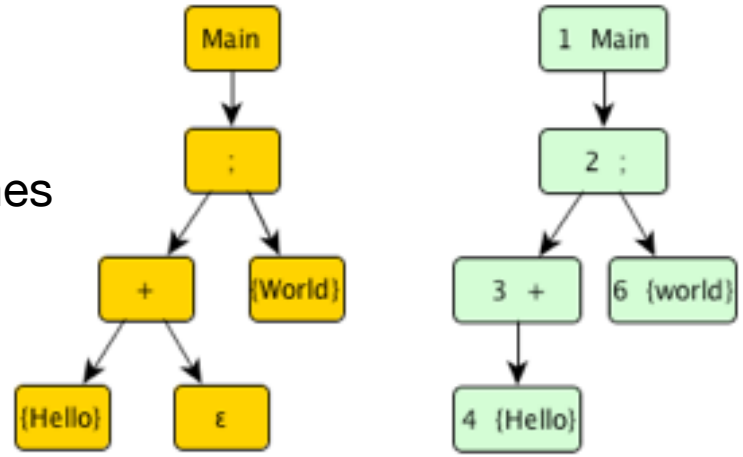
- Much like methods: override, implicit, named args, varargs, ...
- Invoked from Scala: `_execute(a, aScriptExecutor)`
  Default executor: `_execute(a)`
- Body: process expression
  Operators: `+ ; & | && || /` ...
  Operands: script call, code fragment, if, while, ...
- Output parameters: ?, ...
- Shared scripts:
  `script send,receive = {}`

Implementation - 1

- Branch of Scalac: 1300 lines (scanner + parser + typer)

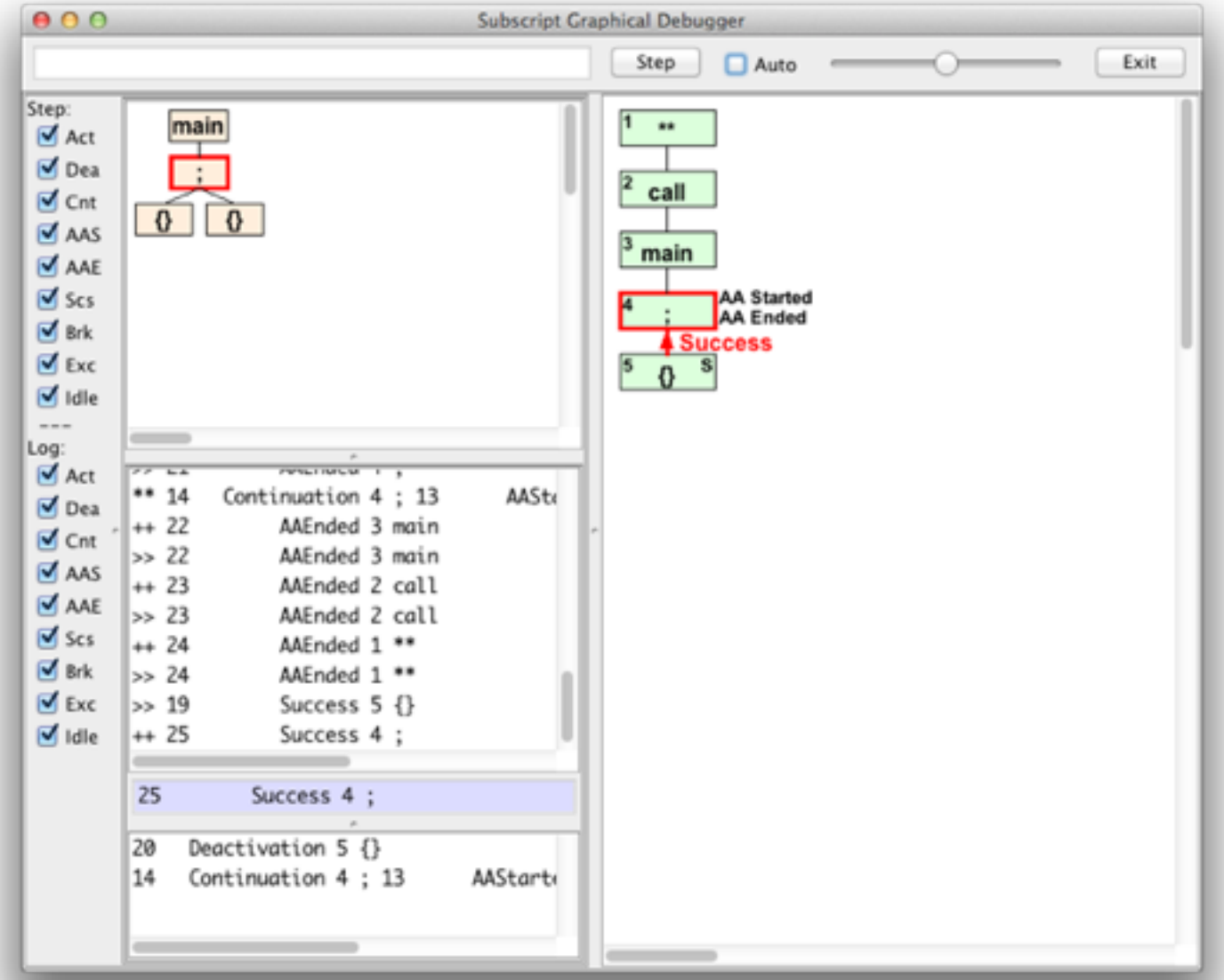
```
script Main = {!Hello!} + ε; {!World!}

import subscript.DSL._
def Main = _script('Main) {
             _seq(_alt(_normal{here=>Hello}, _empty),
                       _normal{here=>World}          )
           }
```

- Virtual Machine: 2000 lines
  - static script trees
  - dynamic Call Graph



- Swing event handling scripts: 260 lines
- Graphical Debugger: 550 lines (10 in SubScript)

Debugger - 1


Subscript Graphical Debugger
Step
Auto
Exit
Step:
Act
Dea
Cnt
AAS
AAE
Scs
Brk
Exc
Idle
Log:
Act
Dea
Cnt
AAS
AAE
Scs
Brk
Exc
Idle
main
;
{}
{}
** 14 Continuation 4 ; 13
++ 22 AAEnded 3 main
>> 22 AAEnded 3 main
++ 23 AAEnded 2 call
>> 23 AAEnded 2 call
++ 24 AAEnded 1 **
>> 24 AAEnded 1 **
>> 19 Success 5 {}
++ 25 Success 4 ;
25 Success 4 ;
20 Deactivation 5 {}
14 Continuation 4 ; 13
1 **
2 call
3 main
4 ;
AA Started
AA Ended
Success
5 {}
S

One-time Dataflow - 2

- Script result type `script confirmExit:Boolean = ...`
- Result values `$: Try[T]`
- Result propagation `call^  {:result:}^`
- Data Flow `x ~~> y`
- Exception Flow `x~/~> y`
- Ternary `x ~~> y +~/~> z`

- Matching flow:

```
x ~~(b:Boolean    )~~> y1
  +~~(i:Int if i<10)~~> y2
  +~~( _          )~~> y3
 +~/~(e:IOException)~~> z1
 +~/~(e:  Exception)~~> z2
 +~/~(e:  Throwable)~~> z3
```

Example: Slick 3

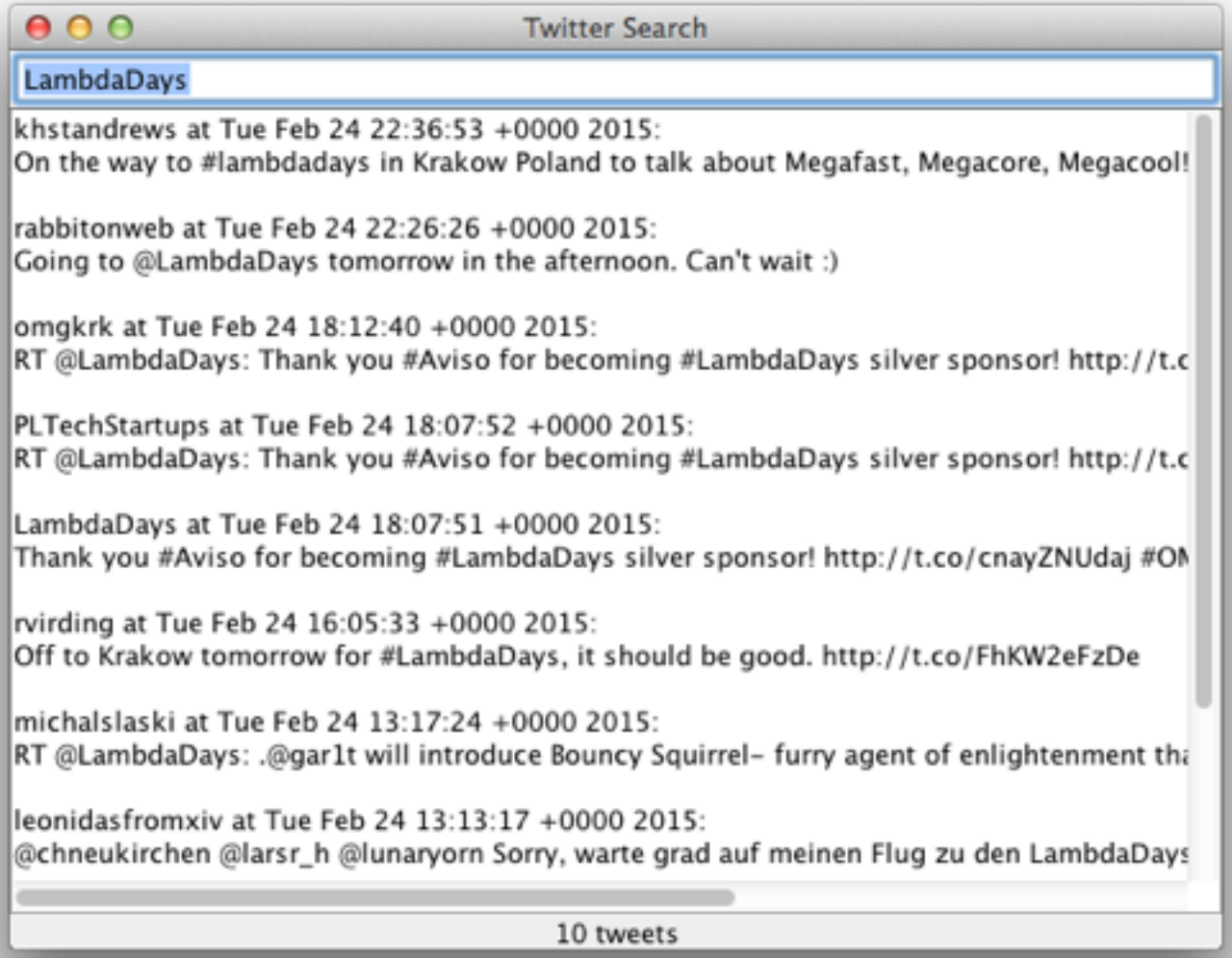
Reactive Streams for Asynchronous Database Access in Scala

http://www.infoq.com/news/2015/05/slick3

```
val q = for (c<-coffees) yield c.name
val a = q.result
val f: Future[Seq[String]] = db.run(a)

f.onSuccess { case s => println(s"Result: $s") }
```

```
val q = for (c<-coffees) yield c.name

q ~~(s)~~> println: s"Result: $s"
```

Example: Twitter Search Client - 1



Example: Twitter Search Client - 2

```
class PureController(val view: View) extends Controller with Reactor {

  def start() = {initialize; bindInputCallback}

  def bindInputCallback = {
    listenTo(view.searchField.keys)

    val fWait   = InterruptableFuture {Thread sleep keyTypeDelay}
    val fSearch = InterruptableFuture {searchTweets}

   reactions += {case _                          => fWait  .execute()
     .flatMap   {case _                          => fSearch.execute()}
     .onComplete{case Success(tweets)            => Swing.onEDT{view. ...()}
                 case Failure(e:CancelException) => Swing.onEDT{view. ...()}
                 case Failure( e               ) => Swing.onEDT{view. ...()}
} } } }
```

Example: Twitter Search Client - 3

```
class SubScriptController(val view: View) extends Controller {
  def start() = _execute(_live())



  script..
    live         = initialize; [mainSequence/..?]...

    mainSequence = anyEvent: view.searchField
                   waitForDelay
                   searchInBG ~~(ts:Seq[Tweet])~~> updateTweetsView:ts
                              +~/~(t: Throwable )~~> setErrorMsg:t

    waitForDelay = {* Thread sleep keyTypeDelay *}
    searchInBG   = {* searchTweets *}

    updateTweetsView(ts: Seq[Tweet]) = @gui: view.setTweets: ts
    setErrorMsg     (t : Throwable ) = @gui: view.setErrorMsg: t
}
```

Example: Twitter Search Client - 4

```
class SubScriptController(val view: View) extends Controller {
  def start() = _execute(_live())
  val fWait   = InterruptableFuture {Thread sleep keyTypeDelay}
  val fSearch = InterruptableFuture {searchTweets}

  script..
    live         = initialize; [mainSequence/..?]...

    mainSequence = anyEvent: view.searchField
                   fWait
                   fSearch    ~~(ts:Seq[Tweet])~~> updateTweetsView:ts
                            +~/~(t: Throwable )~~> setErrorMsg:t



    updateTweetsView(ts: Seq[Tweet]) = @gui: view.setTweets: ts
    setErrorMsg     (t : Throwable ) = @gui: view.setErrorMsg: t
}
```

Example: Twitter Search Client - 4

```
implicit script future2script[T](f:InterruptableFuture[T]): T
= @{f.execute()
     .onComplete {case aTry => there.executeForTry(aTry)}}: {.   .}
```

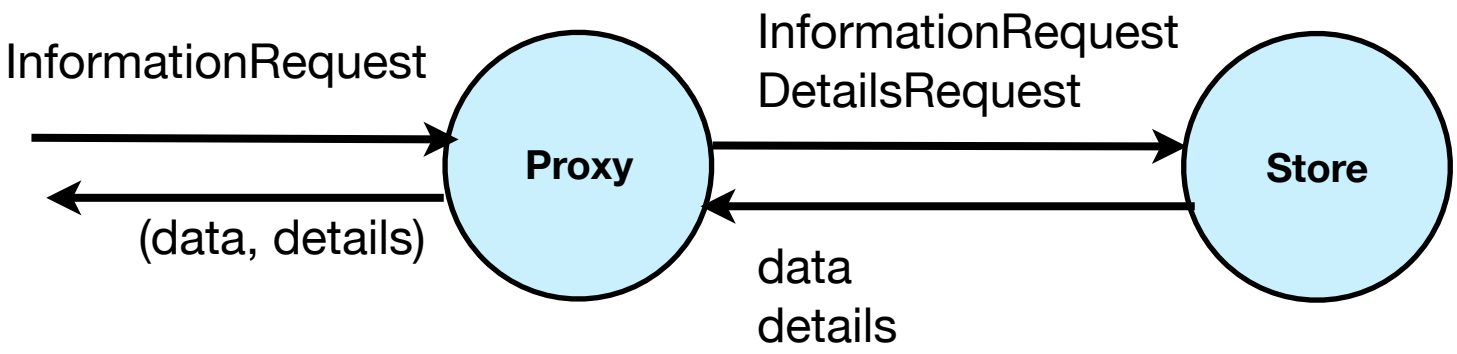
```
implicit def script2future[T](s:Script[T]): InterruptableFuture[T]
= { ... }
```

SubScript Actors: Ping Pong

```
class Ping(another: ActorRef) extends Actor {

  override def receive: PartialFunction[Any,Unit] = {case _ =>}

     another ! "Hello"
     another ! "Hello"
     another ! "Terminal"
}
```

```
class Pong extends SubScriptActor {

  implicit script str2rec(s:String) = << s >>

  script ..
    live = "Hello" ... || "Terminal" ; println: "Over"
}
```

SubScript Actors: DataStore - 1




```
class DataStore extends Actor {

  def receive = {
    case InformationRequest(name) => sender ! getData   (name)
    case DetailsRequest    (data) => sender ! getDetails(data)
  }

}
```

SubScript Actors: DataStore - 2

```
class DataProxy(dataStore: ActorRef) extends Actor {

  def waitingForRequest = {
    case req: InformationRequest =>
      dataStore ! req
      context become waitingForData(sender)
  }

  def waitingForData(requester: ActorRef) = {
    case data: Data =>
      dataStore ! DetailsRequest(data)
      context become waitingForDetails(requester, data)
  }

  def waitingForDetails(requester: ActorRef, data: Data) = {
    case details: Details =>
      requester ! (data, details)
      context become waitingForRequest
  }
}
```

SubScript Actors: DataStore - 4

```
class DataProxy(dataStore: ActorRef) extends SubScriptActor {

  script live =
    << req: InformationRequest ==> {dataStore ? req}
                  ~~(data:Data)~~> {dataStore ? DetailsRequest:data}
            ~~(details:Details)~~> {! sender  ! (data, details)!}
    >>
    ...
}
```

Open Source Project

- subscript-lang.org
  github.com/scala-subscript
- 10^4...10^5 actions per second
- Simple implementation: 6000 lines, 50%
  - Scalac branch ~~> Parboiled + Macro's
  - VM
  - scripts for actors, swing

- Jetbrains - IntelliJ Plugin
- ScalaParse + Dotty

FastParse & ScalaParse

- http://www.lihaoyi.com/fastparse/
- Better error messages than Parboiled2

- Inspiration for SubScript:
  - ^ - normal result value
  - ^^ - result values into List
  - ^^1, ^^2 - result values into tuple

```
script..
   s = var i= 0
       var j=10
       while(i<3) [^i^^1 ^j^^2]^^  {! i+=1; j-=1 !}

 test(1) {runScript(s).$ shouldBe Success(List((0,10),(1,9),(2,8)))}
```

github.com/scala-subscript/koans

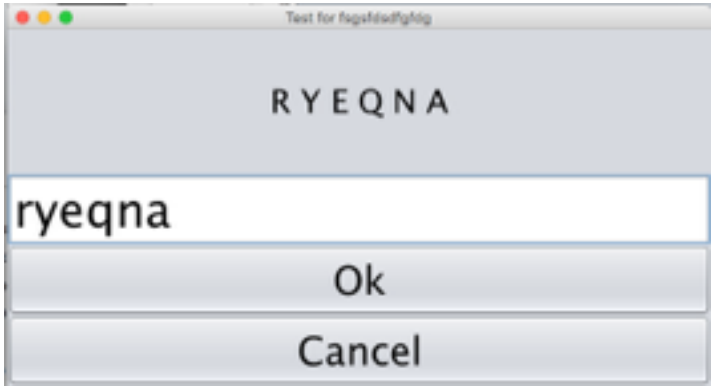
```
package subscript.koans

import subscript.language
import subscript.Predef._

import subscript.koans.util.KoanSuite

class AboutSubScript extends KoanSuite {
  koan(1)(
    """
    | Imports, scripts and atomic actions:
    |
    | To use SubScript in a file, it should have these two import statements:
    |
    | `import subscript.language`
    | ...
    """
  ) {
    var    flag = false
    script foo  = {! flag = true !}

    test(1) { runScript(foo); flag shouldBe __ }
  }
```

github.com/scala-subscript/examples

- helloworld
- lookupframe
- life
- filedownloader
- subscript-twitter-search
- pingpong
- taskprocessor

github.com/scala-subscript/eye-test

```
script..
   live = mainTestProcess^ / cancelBtn

   mainTestProcess = eyeTest("Right")^^1
                     eyeTest("Left" )^^2

   eyeTest(eyeName: String)
   = let testArea.font       = new Font("Ariel", java.awt.Font.PLAIN, 20)
     let testArea.text       = s"<html>Look with your $eyeName eye.</html>"
     let answerField.enabled = false
     sleep: 250
     Key.Enter + okBtn
     doTest( if(eyeName=="Right") previousScoreRight else previousScoreLeft )^
```



Conclusion

- Easy and efficient programming
- Still much to do: JS, NodeJS, ACP style communication, ...
- and to discover: arXiv paper "Some New Directions in ACP Research"
- To join the project: andre.vandelft@gmail.com
- Sponsors also welcome