BEAUTY AND THE BEAST

Eta Haskell for JVM
JAREK RATAJSKI

@jarek000000

Loves programming since the first line I wrote for C64

Anarchitect @ Engenius GmbH

Java developer (since 1999) with a functional heart.
ORIGIN
Lots of developers that love Haskell
but when it comes to business...
WE ONLY DO JAVA HERE
return List.ofAll(iterableRead)
  .foldLeft(HashMap.<String, Topic>empty(),
             (existingMap, newElement) ->
             existingMap.put(newElement)
             existingMap.get(n)
             .getOrElse(addMessage);
};

} catch (IOException e) {
    return HashMap.<String, Topic>empty();
}
WRITE **HASKELL USING SCALA**

```scala
implicit val treeApplicative: Applicative[Tree] = new Applic
def point[A](a: => A): Tree[A] = Tree(a, Seq.empty)

def ap[A, B](fa: => Tree[A])(tf: => Tree[A => B]): Tree[B]
  val Tree(f, tfs) = tf
  Tree(f(fa.root), fa.children.map(t => t.map(f)) ++ tfs.m
}
```
WRITE HASKELL USING HASKELL (ETA)
deploy to JVM
work with Java
TYPELEAD

Company founded to create eta and in the future provide commercial support for it.
I am not associated with Typelead.
Whatever I say show here are mine own studies, with some help of typelead developers and community.

I am neither experienced haskell nor eta developer.

What I say might be wrong or may not reflect the reality or the future.

I tried to do my best
We talk about half-finished product.
ETA 1.2.3 INTRO
quicksort [] = []
quicksort (x:xs) = quicksort left ++ [x] ++ quicksort right
  where
    left = [ y | y <- xs, y < x ]
    right = [ y | y <- xs, y >= x ]

main = do
  let result = quicksort arr
  putStrLn $ show result
  putStrLn $ show result
  where
    arr = [1,7,9,12,90,1,-1,22,0]

$ eta Main.hs

$ java -jar Main.jar

[-1,0,1,1,7,9,12,22,90]
ETLAS (CABAL FOR ETA)

$ etlas init
$ etlas build
$ etlas run
For more info see Eta tour page
ETA SPECIAL

ETA =~= GHC FOR JVM
backend for GHC -> great compatibility
STG machine

Fibb.fibbtcoinner
:: forall a_a7U9 a1_a7UA.
   (GHC.Classes.Eq a_a7U9, GHC.Num.Num a_a7U9, GHC.Num.Num a_a7U9 -> a1_a7UA -> a1_a7UA -> a1_a7UA

[GblId,
Arity=6,
Caf=NoCafRefs,
Str=<S(C(C(S)))L),U(C(C1(U)),A)<L,U(A,C(C1(U)),A,A,A,A,C(U))>
Unf=OtherCon [] =
   \r srt:SRT:[] [$dEq_s8MJ
   $dNum_s8MK
   $dNum1_s8ML
   eta_s8MM
   eta1_s8MN
   eta2_s8MO]
SPINELESS, TAGLESS G-MACHINE
...It defines how the Haskell evaluation model should be efficiently implemented on standard hardware. ...
STG =~= (bytocode or llvm)
1ST PHASE HS TO STG

Eta compiler in a phase .hs to STG

..is simply a GHC code! (forked)
2ND PHASE - STG TO BYTECODE / JVM
C IMPORTS

GHC supports native (C language) calls. (for instance used in Base packages)
Eta rewrites those parts to use JVM calls.

original GHC Float.hs fragment

```haskell
foreign import ccall unsafe "isFloatNaN" isFloatNaN :: Float -> Bool
foreign import ccall unsafe "isFloatInfinite" isFloatInfinite :: Float -> Bool
foreign import ccall unsafe "isFloatDenormalized" isFloatDenormalized :: Float -> Bool
foreign import ccall unsafe "isFloatNegativeZero" isFloatNegativeZero :: Float -> Bool
foreign import ccall unsafe "isFloatFinite" isFloatFinite :: Float -> Bool
```

Eta Float.hs fragment

```haskell
foreign import java unsafe "@static java.lang.Float.isNaN" isFloatNaN :: Float -> Bool
foreign import java unsafe "@static java.lang.Float.isInfinite" isFloatInfinite :: Float -> Bool
foreign import java unsafe "@static eta.base.Utils.isFloatDenormalized" isFloatDenormalized :: Float -> Bool
foreign import java unsafe "@static eta.base.Utils.isFloatNegativeZero" isFloatNegativeZero :: Float -> Bool
foreign import java unsafe "@static eta.base.Utils.isFloatFinite" isFloatFinite :: Float -> Bool
```
ETLAS

Haskell GHC developers use cabal (or stack).

Etlass is eta tool which is ~ cabal. It uses .cabal file format with extensions.
HACKAGE
Tons of libraries for haskell.

De facto standard.
ETA HACKAGE PATCHES
Project typelead/hackage == patches for common hackage projects.
Mostly 1 to 1 native C to Java calls changes.

https://github.com/typelead/eta-hackage/blob/master/patches/text-1.2.2.2.patch

{-# INLINE equal #-}

-foreign import ccall unsafe "_hs_text_memcpy" memcpyI
+foreign import java unsafe "@static eta.text.Utils.memcpy" m
  :: MutableByteArray# s -> CSize -> ByteArray# -> CSize -

-foreign import ccall unsafe "_hs_text_memcmp" memcmp
+foreign import java unsafe "@static eta.text.Utils.memcmp" m
  :: ByteArray# -> CSize -> ByteArray# -> CSize -> CSize -
SUPPORTS COMPILE EXTENSIONS
{-# LANGUAGE FlexibleContexts, DataKinds, TypeFamilies, RankNT
--  whatever
Eta is as close as you can get with Haskell/GHC on JVM
Lots of crazy haskell codes that use GHC extensions work on Eta without any problems.
BASIC OPTIMISATIONS
Naive fibonacci

\[ \text{fibnaive } n = \text{fibnaive } (n-1) + \text{fibnaive } (n-2) \]
```python
fibtcoinner 0 sum presum = sum
fibtcoinner n sum presum = fibtcoinner (n-1) (sum + presum)
fibtco n = fibbtcoinner n 1 0
```
private static BigInteger fibonacci(int n, BigInteger sum, Big if ( n== 0) {
    return sum;
} else {

    return fibonacci(n-1, sum.add(presum), sum);
}
}
How much java stands?
$$\text{fibtcoinner } n \text{ sum presum } = \text{fibtcoinner } (n-1) \text{ (sum + presum) sum presum }$$

$$\text{fibtco } n = \text{fibbtcoinner } n \ 1 \ 0$$
First results....
[1, 1, 2, 4, 8, 16, 32, 64, 128, 256, 512...]
BUG #603

It took couple of nights to fix this bug. I've Learned haskell...
while(var8) {
    Main.sat_s7YH var12 = new Main.sat_s7YH(var3);
    var1.R1 = var2;
    Closure var13 = Classes.zeze().enter(var1).apply2(var1, (if (!((var13 instanceof False))) {
        return ((Closure)var10).evaluate(var1);
    })

    Main.sat_s7YM var14 = new Main.sat_s7YM(var4, (Closure)var10);
    Main.sat_s7YL var15 = new Main.sat_s7YL(var3, (Closure)var10)
    var9 = var15;    //assign n-1
    var10 = var14;   //assign new sum
    var11 = var14;   //assign presum
    var8 = true;
}
while( n > 0 ) {
    n = n - 1;
    newSum = presum + sum
    sum = newSum
    presum = sum
}
I've seen you've fixed the bug, what have you done?

I've swapped two lines

Don't ask!!!

And why...?
while( n > 0 ) {
    n = n - 1;
    newSum = presum + sum
    presum = sum  // swapped
    sum = newSum  // swapped
Fix
Fix compiler of Haskell written in Haskell (ghc) while learning haskell.
MOMENT I SAW
CODEGEN MONAD
withContinuation unknownCall contCode lastCode

JumpToIt label cgLocs mLne -> do
    traceCg (str "cgIdApp: JumpToIt")
    codes <- getNonVoidArgCodes args
    emit $ multiAssign cgLocs codes
    let sorted =
        codes <- getNonVoidArgCodes $ arg <$> sorted
        emit $ multiAssign (from <$> sorted) codes
        <$> maybe mempty
        <$> maybe mempty
        (\(target, targetLoc) ->
            storeLoc targetLoc (iconst (locFt targetLoc mLne mLne)
        <$> goto label
        <$> goto label

+data LocalDep = LocalDep Int Int
+{-
while(var8) {
    Main.sat_s7YH var12 = new Main.sat_s7YH(var3);
    var1.R1 = var2;
    Closure var13 = Classes.zeze().enter(var1).apply2(var1, (if (!(var13 instanceof False))) {
        return ((Closure)var10).evaluate(var1);
    }
    Main.sat_s7YM var14 = new Main.sat_s7YM(var4, (Closure)var10);
    Main.sat_s7YL var15 = new Main.sat_s7YL(var3, (Closure)var10);
    var11 = var10; //assign presum
    var10 = var14; //assign new sum
    var9 = var15; //assign n-1
    var8 = true;
}
HOW MUCH ETA STANDS???
main = print $ show $ fibtco 100000
import Control.Monad.Trans.Cont

fibCps::Int->Cont r Int
fibCps 0 = return 1
fibCps 1 = return 1
fibCps n = do
    n1 <- fibCps $ n-1
    n2 <- fibCps $ n-2
    return $ n1 + n2

main = do
    let result = trampoline $ runCont (fibCps 100) id
    putStrLn $ show result
PERFORMANCE
- JMH
- Quick sort implementations exported and called from java
- naive and real quicksort
- compared to same solutions in Java (using vavr.io)
- not very professional - just to get some overview
Naive quicksort Eta

```plaintext
quicksort [] = []
quicksort (x:xs) = quicksort left ++ [x] ++ quicksort right
where
left = [ y | y <- xs, y < x ]
right = [ y | y <- xs, y >= x ]
```

Naive quicksort Java/vavvr

```java
private List<Integer> qsort(List<Integer> input) {
    if (!input.isEmpty()) {
        final int middle = input.head();
        final List<Integer> left = input.tail().filter(
        final List<Integer> right = input.tail().filter(
        return qsort(left).appendAll(qsort(right).prepe
    } else {
        return input;
    }
```
Real quicksort ETA

```haskell
qvsort :: (G.Vector v a, Ord a) => v a -> v a
qvsort = G.modify go where
  go xs | M.length xs < 2 = return ()
  | otherwise = do
    p <- M.read xs (M.length xs `div` 2)
    j <- M.unstablePartition (< p) xs
    let (l, pr) = M.splitAt j xs
    k <- M.unstablePartition (== p) pr
    go l; go $ M.drop k pr

myvsort :: [Int] -> [Int]
myvsort li =
  let vec = V.fromList li :: (V.Vector Int)
      sorted = qvsort vec :: (V.Vector Int)
      converted = V.toList sorted :: [Int]
```

Real quicksort Java (*)

```java
list.sort(); // :-)
```
Results

Eta / Java Quicksort performance

<table>
<thead>
<tr>
<th></th>
<th>Eta</th>
<th>Java/Vavr</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Quicksort</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Real Quicksort</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

op/s
VS OTHER HASKELLS

12 Queens
{-# LANGUAGE BangPatterns #-}

-- solution by Oystein Kolsrud
-- https://www.youtube.com/watch?v=I2tMmsZC1ZU

okToAdd :: Int -> [Int] -> Bool
okToAdd q qs = all (okToAddDirection q qs) [succ, pred, id]
  where
    okToAddDirection q qs f = and $ zipWith (/=) (tail

extendSolution n qs = map (\q -> q:qs) $ filter (\q -> okTo

allSolutions !n 0 = [[]]
allSolutions !n k = concatMap (extendSolution n) (allSoluti
<table>
<thead>
<tr>
<th>Implementation</th>
<th>Task</th>
<th>Solutions</th>
<th>Time (real)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frege</td>
<td>12</td>
<td>14200</td>
<td>(*45.816s)</td>
</tr>
<tr>
<td></td>
<td>Queens Solutions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Eta</td>
<td>12</td>
<td>14200</td>
<td>(*26.472s)</td>
</tr>
<tr>
<td></td>
<td>Queens Solutions</td>
<td></td>
<td></td>
</tr>
<tr>
<td>GHC</td>
<td>12</td>
<td>14200</td>
<td>9.806s</td>
</tr>
<tr>
<td></td>
<td>Queens Solutions</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Unfair benchmark - both frege and eta were measured with JVM startup time.
JAVA
INTEROPABILITY
data JColor = JColor @java.awt.Color deriving Class
foreign import java unsafe "getGreen" getGreen :: Java JColor Int
Java is a Monad.

-- Execute a Java action in the IO monad.
java :: Java c a -> IO a

-- Execute a Java action in the IO monad with respect to the
-- given object.
javaWith :: (Class c) => c -> Java c a -> IO a

-- Execute a Java action in the Java monad of another class
-- with respect to the given object.
(<.>) :: (Class c) => c -> Java c a -> Java b a

withObject :: (Class c) => c -> Java c a -> Java b a

-- Chain Java actions.
(>-) :: (Class b) => Java a b -> Java b c -> Java a c
FOREIGN EXPORT

foreign export java "@static eta.example.MyExportedClass.sort"
  sort :: JIntArray -> JIntArray
STYLES OF INTEROPERABILITY
Example: WAI Servlet

```haskell
case pathInfo req of
    ["state"] -> appState (unsafePerformIO $ newMVar 0) r
    ["stream" ] -> appStream req respond
    ["request-info"] -> appShowReq req respond
    ["static-file"] -> appFile filePath req respond
    _                -> appSimple req respond
```
CLASSES IN JAVA LOGIC IN HASKELL

- Types defined in java
- Haskell functions work on Java objects
- Support and use of Java frameworks, serializations, db bindings, jsons.
Hint: in 2018 most of Java frameworks do not need classical/ugly JavaBeans anymore.
@JsonDeserialization
public class Ball extends GameObject {
    private static final long serialVersionUID = 1L;
    public final Vector2D speed;

    @JsonCreator
    public Ball(float x, float y, Vector2D speed) {
        super(x, y);
        this.speed = speed;
    }
}
@JsonDeserialize
public class GameState implements Serializable {
    private static final long serialVersionUID = 1L;
    public final GamePhase phase;
    public final Ball ball;
    public final Players players;
    public final long updateTime;

    @JsonCreator
    public GameState(final Ball ball, final Players players, final long updateTime) {
        this.ball = ball;
        this.players = players;
    }
}
foreign import java unsafe "@new" newGameState :: Ball.Ball -
foreign import java unsafe "@field phase" phase :: GameState -
foreign import java unsafe "@field ball" ball :: GameState ->
foreign import java unsafe "@field players" players :: GameState -
foreign import java unsafe "@field updateTime" updateTime :: GameSta
push::GameState->Int64->J.Random->IO GameState
push state time rnd
| (aPhase == GamePhase.started ) = pushStarted state
| otherwise = return state
where aPhase = phase state
Linguistic determinism
from http://postcogtopics.blogspot.com/2016/
private Tuple2<Ball, Players> bouncePlayer1(final Players players) {
    if (this.x < 0 && speed.x < 0) {
        if (isTouchingPaddle(players.player1.paddle, this.
            return Tuple.of(new Ball(0f, this.y, this.speed.x),
        } else {
            return Tuple.of(Ball.randomDirection(rnd), players)
    }
    return Tuple.of(this, players);
}

private Tuple2<Ball, Players> bouncePlayer2(final Players players) {
    if (this.x > 1.0f && speed.x > 0) {
        if (isTouchingPaddle(players.player2.paddle, this.
            return Tuple.of(new Ball(1f, this.y, this.speed.x),
        } else {
            return Tuple.of(Ball.randomDirection(rnd), players)
    }
    return Tuple.of(this, players);
}
bouncePlayerInternal :: Ball -> Players.Players -> J.Random -> (Lens'
bouncePlayerInternal ball players rnd lens opLens xposition
   | (isTouchingPaddle paddle thisY) = return (newBall xpos
   | otherwise = do
     randomBall <- randomDirection rnd
     return ( randomBall, set opLens opponentScored playerView

where
  thisX = xObj ball
  thisY = yObj ball
  thisSpeed = speed ball
  speedX = Vector2D.x thisSpeed
  playerView = view lens players
  opponentScored = Player.incScore $ view opLens players
  paddle = Player.paddle playerView
ballBounceP :: Ball.Ball -> Players.Players -> J.Random -> IO
Players.Players
POINTER REF WAY
Data in haskell, businell logic in haskell. Java as Controller.
We need to expose haskell *objects* to java.
data Color = Color {red :: Int, 
g
data Cell = Dead | Alive {color :: Color}

type Row = Array Int Cell
type Plane = Array Int Row
type GOLState = StablePtr Plane

initEmptyXP :: Int -> Int -> IO GOLState
initEmptyXP wi hi = newStablePtr $ makePlane wi hi

c= "\n
public static int newState(int var0) {
    return ((StablePtr)Runtime.evalIO(new Ap2Upd(TopHandle}
}
PROBLEMS

- lot of imports to write for every simple java class
  - this will be fixed thanks to ffi tool
- it took me a while to find out how to pass state between haskell and java
- other bug found (and resolved)
- java monad / io monad - not totally intuitive (for a newbie)
ETA VS FREGE
I used Frege very shortly.

- Frege is more mature
- Interoperation with Java is easier with Frege
- Frege will not be close GHC in the near future
  - at the semantics level
  - at the base libraries level
ETA FOR YOU
ETA NOW
Eta is 0.7.0b2 is not production quality yet
If You think of eta in production soon -> talk to Typelead.

They want to provide commercial support - ask them for conditions.
If you are haskell developer that wants to evaluate haskell on JVM

Try it now!
If you are JVM / JavaDeveloper that wants to learn and play with Haskell

Play now!
ETA COMMUNITY
Small.
Great!
You can help! There are lot of small things to do.
Future of eta lies in your hands