SCALABLE AND RELIABLE VIDEO TRANSCODING IN HASKELL

Alfredo Di Napoli

Haskell Exchange 2015

ABOUT ME

Full story at: **http://goo.gl/qkKwKm**


```
launchMissile :: IO ()
```

discover. develop. share.

IRIS connect (contd.)

IRIS' GREEK ZOO

HERMES' EVOLUTION

October, 2013

October, 2015

https://github.com/iconnect/hermes/tree/74aeab22be600053dd841277

HERMES' CHALLENGES

abstraction is the (media key)

What's the elephant in the room?

Why not use Cloud Haskell?

2 DAY COURSE

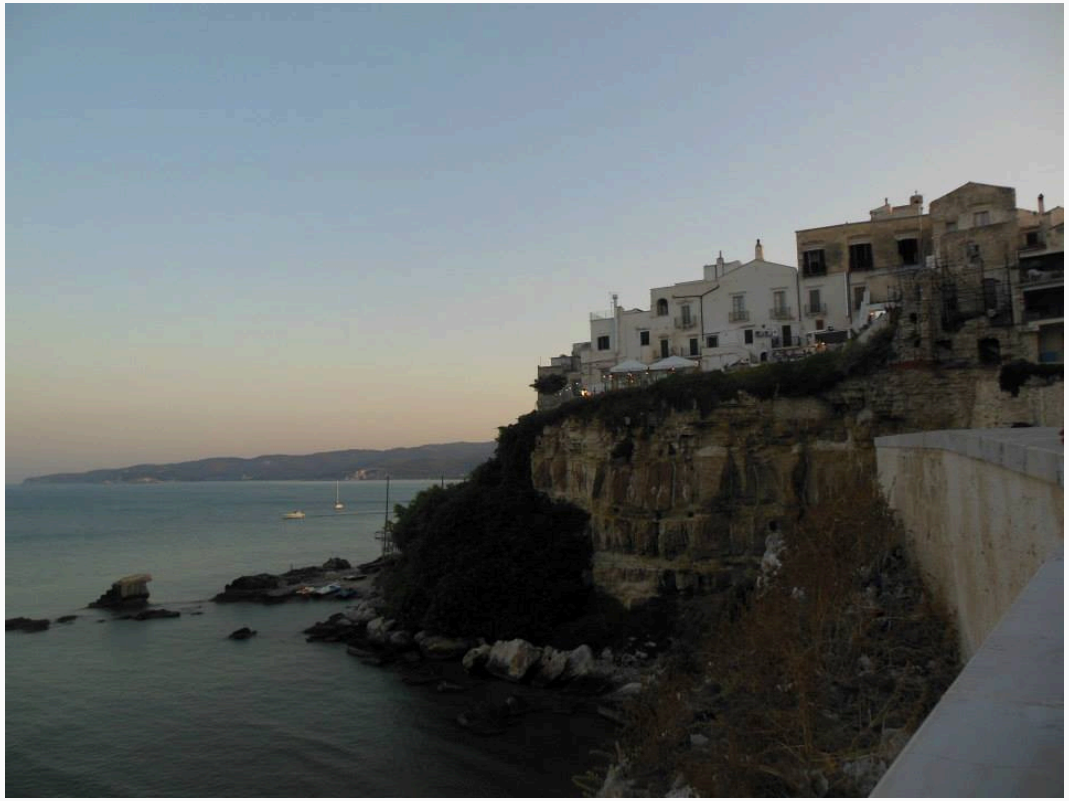
Well-Typed's Advanced Haskell

After a couple of months (it was July 2013) Well Typed was hiring. I decided to take pot luck and I applied.

To maximise my chances, I applied to a couple of other positions for Haskell jobs.

Despite the rejections, **I was actually able to face an entire interview doing nothing but Haskell!**

August 2013, Vieste - Italy



Got rejected by WT, but they said "A client of us might be searching soon..."

LANDING THE TECH JOB I LOVED

On the 29th of August, I applied for a Haskell job @ Iris Connect. Took a train to Brigthon, did the interview and was offered the position. **I was officially an Haskeller!**

TAKEHOME LESSONS

1. Don't be afraid to take leaps into the dark
2. Life is about opportunities, seize them
3. Try to contribute to a "famous" Haskell OSS
4. Constantly "sharpen your saw"
5. Be receptive, do networking

A sharing and collaboration CPD platform for teachers via video recording, feedback and introspection.

1. Initially build with RoR, it was rewritten from scratch in Haskell (backend) and RoR + Angular.js (frontend)
   1.1 Effort initially started by my colleage Chris Dornan and Well Typed

2. The backend is composed by two main projects:
   2.1 The frontend-facing API server which holds the model and the business logic (**Atlas**)
   2.2 The video transcoding system (**Hermes**), a highly distributed and fault tolerant system, built on top of RabbitMQ.

WHY HASKELL?

Because software development is a marathon, not a sprint.

"It took me more time writing the specs that implementing the feature itself."

Because we are like Shlemiel the painter.

"I can't help it," says Shlemiel. "Every day I get farther and farther away from the paint can!"

1. The more time it pass, the farther we get from our “paint can”, the mental model we built of the system.
2. In large scale systems, you can have parts that won't be touched for *years*!
   2.1 How do you defend yourself when the refactoring or feature time comes?
3. A rich, strong and expressive type system can be your ultimate ally against complexity
   3.1 Things like `newtype`s and `ADT`s can help you cure common “diseases” like *Boolean Blindness*

As universe expands, so does the entropy in your software: use types to keep it at bay!

Some "pros" of working in Haskell

1. Refactoring is a dream
2. EDSLs are a piece of cake
3. Makes impossible states unrepresentable

REFACTORING IS A DREAM

1. The type system naturally guides you
2. In Haskell we tend to write small and generic functions
   - 2.1 Cfr. Bob Martin's "Clean Code"
   - 2.2 Most of the time they don't even break as they are written to work on polimorphic types
   - 2.3 Code reuse = profit!

So ultimately is not just about the strong type system, is about Haskell's (and Haskellers) natural tendency towards **composition** and **parametricity**.

EDSLS ARE A PIECE OF CAKE

```
fromPreset :: MediaFile -> MediaFile
           -> Maybe Atlas.VideoFilter
           -> VideoPreset -> Maybe VideoRotation
           -> LogLevel -> [T.Text]
fromPreset filename outFilePath flt vpres vi ll =
  let cli = ffmpegCLI $ mconcat [
              i $ toTextIgnore filename
            , loglevel ll
            , fromVideoPreset vpres
            , isVideoRotated vi <?> resetRotateMetadata
            , yuv420p
            , vf [rotateMb vi]
            , isJust flt <?> vf_technicolor
            , o_y_ext (toTextIgnore outFilePath) (Left vpres)
            ]
  in T.words cli
```

MAKES IMPOSSIBLE STATES UNREPRESENTABLE

Real world scenario:

```haskell
-- | Creates a new Supervisor.
-- Maintains a map <ThreadId, ChildSpec>
newSupervisor :: IO Supervisor

-- | Start an async thread to supervise its children
supervise :: Supervisor -> IO ()

-- | forkIO-inspired function
forkSupervised :: Supervisor
               -> RestartStrategy
               -> IO ()
               -> IO ThreadId
```

Example usage:

```
main = do
  sup <- newSupervisor
  supervise sup
  _ <- forkSupervised sup OneForOne $ do
         threadDelay 1000000
         print "Done"
```

Can you spot a potential bug?

Nothing in the types is forcing us to call `supervise` before actually supervising some thread!

```haskell
main = do
  sup <- newSupervisor
  -- Wrong! We forgot to start the supervisor...
  _ <- forkSupervised sup OneForOne $ do
         threadDelay 1000000
         print "Done"
```

As Haskellers, we can certainly do better!

PHANTOM TYPES TO THE RESCUE!

Phantom Types allow us to "embed" constrain on our types, together with smart constructors.

```
data Uninitialised
data Initialised

data Supervisor_ a = Supervisor_ {
      -- record fields (omitted)
      }

type SupervisorSpec = Supervisor_ Uninitialised
type Supervisor = Supervisor_ Initialised
```

Let's now slightly change our API to be this:

```haskell
-- | Creates a new Supervisor.
newSupervisor :: IO SupervisorSpec

-- | Start an async thread to supervise its children
supervise :: SupervisorSpec -> IO Supervisor
```

What did we get? Let's try to run the "wrong" snippet again...

```
main = do
  sup <- newSupervisor
  _ <- forkSupervised sup OneForOne $ do
         threadDelay 1000000
         print "Done"
```

What did we get? Let's try to run the "wrong" snippet again...

```
main = do
  sup <- newSupervisor
  _ <- forkSupervised sup OneForOne $ do
         threadDelay 1000000
         print "Done"
```

GHC will complain:

```
Couldn't match type Control.Concurrent.Supervisor.Uninitialised
         with Control.Concurrent.Supervisor.Initialised
Expected type: Supervisor
Actual type: Control.Concurrent.Supervisor.SupervisorSpec
```

1. This is because now we require a `Supervisor` to be initialised first
2. The type system prevented us making a silly mistake
   2.1 Failed with a very useful error message
3. Profit!

This is just a small example (this is only one of the possible solutions), but the benefits are real.

`https://github.com/adinapoli/threads-supervisor`

SNAGS OF WORKING IN HASKELL

1. Slow(ish) Compilation
2. Cabal Hell

slow(ish) compilation

SLOW COMPILATION: THE CAVEAT

1. It's a problem all non-interpreted languages have to deal with
2. GHC indeed does incremental compilation, building only what's changed
3. It's even slower if..
   3.1 You have TH (Template Haskell) in your code
   3.2 You are building with profiling enabled

If you want faster feedback loop, consider using ghci

It's the aggregate of more than one problem, which most of the time results in "I couldn't install package X (easily)"

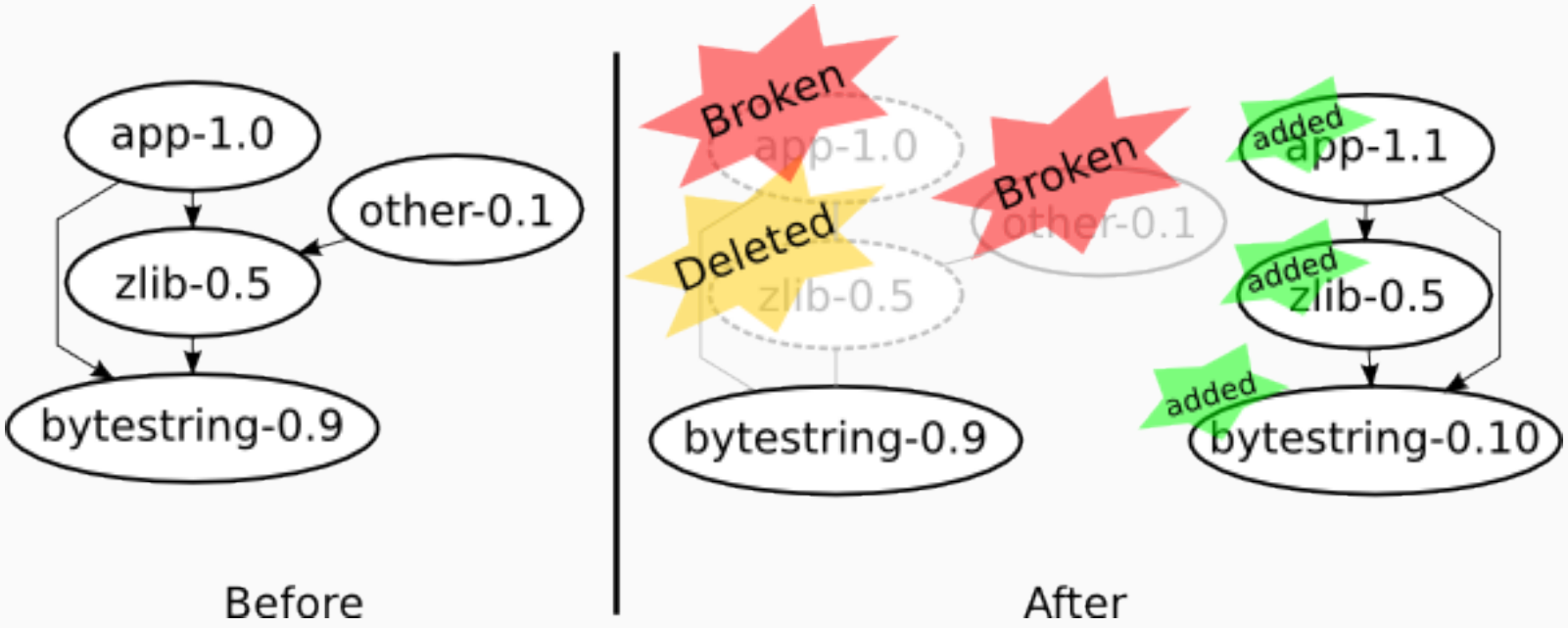


Figure 1: Image courtesy of Well Typed Ltd

CABAL HELL - THE SILVER LININGS

1. Sandboxes mitigate the issue

```
cabal sandbox init
cabal install
```

2. “Package aggregates” can help

   Stackage

   HaskellLTS

   Nix and NixOS

3. Broader solutions are in the pipeline

   Edward Z. Yang’s “Backpack”

Thank you.

Questions?

My road to Haskell

http://www.alfredodinapoli.com/posts/2014-04-27-my-road-to-haskell.html

Don Stewart - Haskell in the large

http://code.haskell.org/~dons/talks/dons-google-2015-01-27.pdf

Joel Spolsky - Back to Basics

http://www.joelonsoftware.com/articles/fog0000000319.html