01

November 2020

CHANGE HERE for the FP LINE

Monthly Digital Functional Programming Magazine

{- 1 Introduction

We are delighted to introduce the inaugural issue of ***Bind The Gap***!

Bind The Gap (BTG) is a functional programming magazine that reflects our thoughts and vision on the topics of current interest.

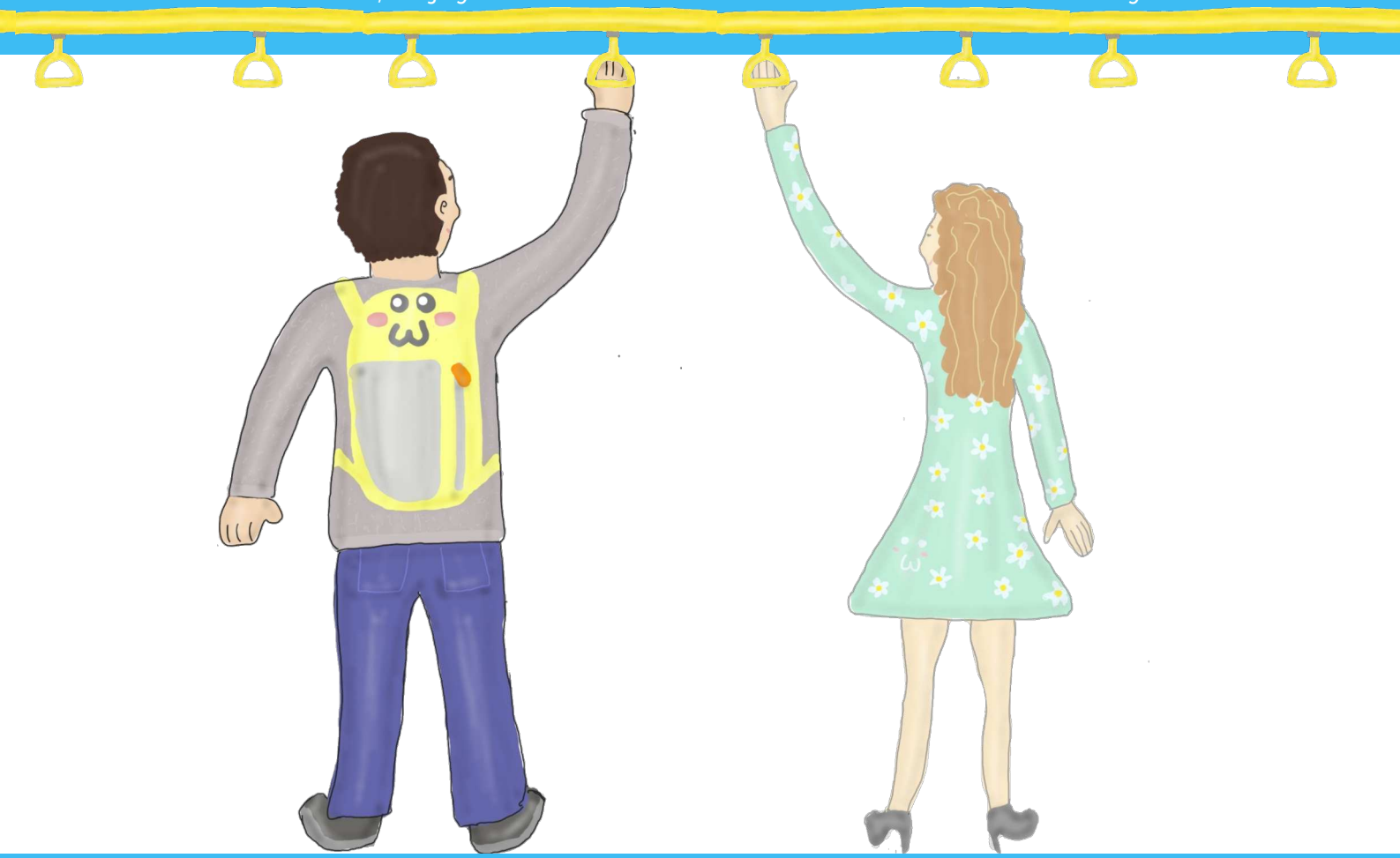
The idea of creating a magazine came to us while we were discussing news in the Haskell world, as we always do. We thought that if we have a platform where we would be able to expand interesting discoveries and events, and add our thoughts to it; it would be actually attractive to those who share the same passion in FP and Haskell in particular.

The goals of Bind The Gap are to elucidate Haskell news, comment on actual issues, meet and introduce wonderful Haskellers, popularise Haskell and FP, involve people in more open discussions, and share our vision and positions.

The "pilot" issue of the magazine is focused on the most eminent news that we are excited about. It also introduces the permanent columns that shed light on the different aspects of Haskell: GHC proposals, beginners information, challenges and more!

Thanks a lot to everyone involved, particularly: **Simon Peyton Jones** for kindly accepting our interview; **Alexander Granin** for his book and interview; **Impure Pics** for the incredible *"Pure Gold by Impure Pics"* section; **Cate Roxl** for proofreading. We appreciate your time and collaboration!

Without further ado, enjoy the first issue of *BTG* and let us know how you liked it!



Dmitrii Kovanikov <> Veronika Romashkina,
Editors-in-Chief
-}

MAP LINE

LOCO-MOTIVE

Interview with *Simon Peyton Jones* about **Haskell Foundation**

3-7

LET US IN NOVATE

The of-lambda GHC proposal overview

8-9

WHINE SOMMELIER

Review of the aeson library

10-12

THE -WALL STREET ANALYTICS

Analytics of the -Wunused-packages warning

13

LIBRARIAN

Review the newtype helpers proposal to base

14

FOLD LINE

PURE GOLD BY IMPURE PICS

Illustrations by *Impure Pics*

15

NEWS FROM ALL AROUND THE HELLO WORLD

The best ways to create projects in Haskell

16-17

WHAT'S THE READIO?

Interview with *Alexander Granin* about his book **Functional Design and Architecture**

18-20

FUN FOR THE WHOLE TYPE FAMILY

FP Humour, challenges, puzzles and surveys

21-23

LOCO MOTIVE

November started with a piece of excellent news for Haskell. Wonderful Simon Peyton Jones officially announced the creation of the Haskell Foundation (HF) [1] at the beginning of this month. This is a crucial step towards a brighter, and more importantly, a more definite future for the language and its users. We couldn't walk past such an exciting occasion, so we decided to get more information about the Haskell Foundation – look closer on what it means for us and the language, understand how it works, and find out if any of us could be part of this.

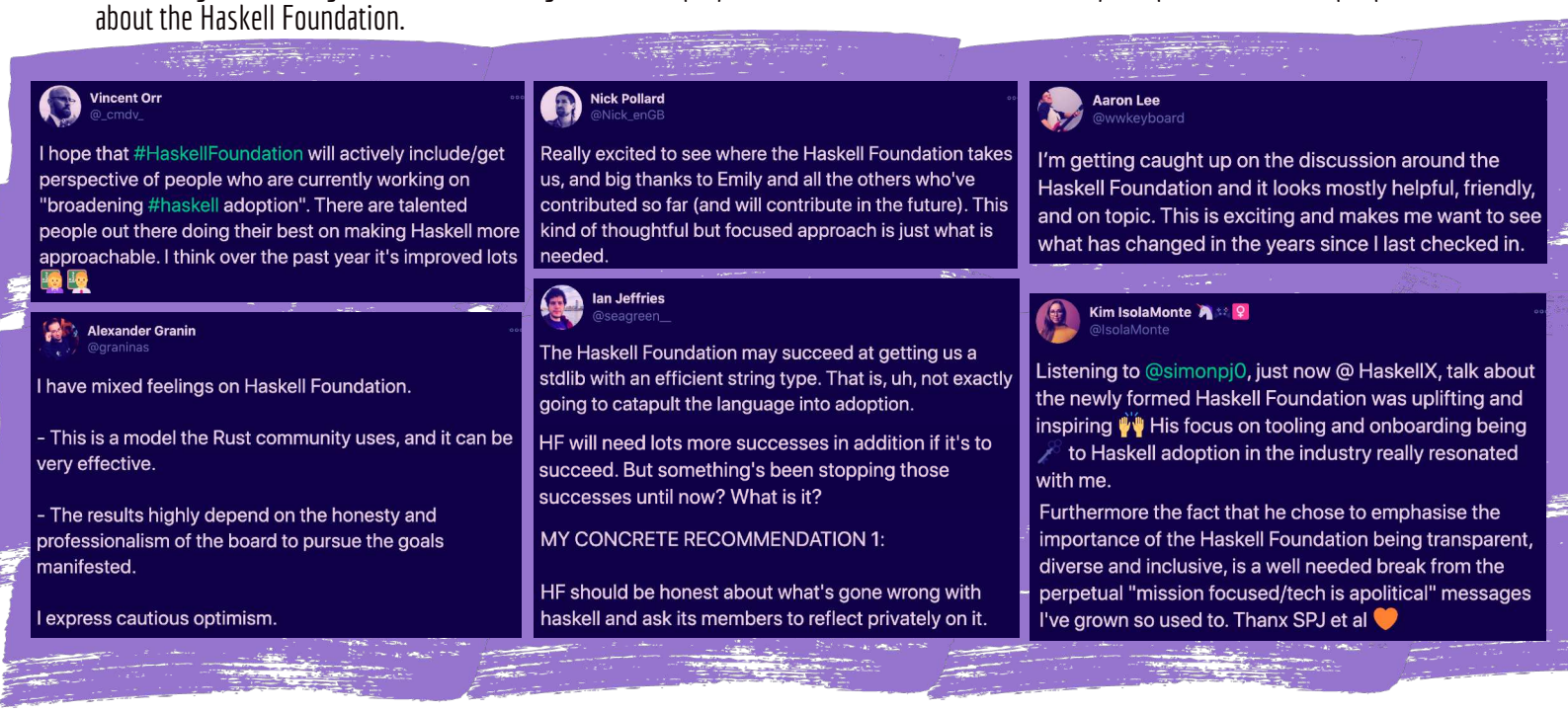
Here is the official description of what the organisation represents:

> The Haskell Foundation is an independent, non-profit organisation dedicated to broadening the adoption of Haskell, by supporting its ecosystem of tools, libraries, education, and research.

HF is supposed to become a glue for different subcommunities within the entire Haskell ecosystem. The primary goal is to enable diverse and separated groups to work more efficiently by helping each other and collaborating on a more fundamental level.

HF was met with the support from companies that use Haskell as the primary language, and they decided to sponsor the organisation. This shows the great significance of such structure to consumers. HF could use their donations to hire different people for solving long-standing problems. For example, one of the things on the priority radar is fixing the Windows support, which would improve the user experience for the specific group, and companies that were limited by these restrictions.

HF became a trending and hot topic after the announcement. There are very positive vibes from all people, which proves that something like the organisation is the long-awaited hope for all Haskellers. We've collected a few opinions on what people think about the Haskell Foundation.



We spoke to Simon Peyton Jones to get more insights on HF and got the opportunity to find out fascinating points from the Haskell-star himself!

[1]: https://haskell.foundation/

Simon Peyton Jones

Haskell and GHC lead designer and developer, principal researcher at Microsoft Research in Cambridge, chair of the Computing at School

> It's really important that we, the community, feel that the foundation is something we are doing together, not that it is something that somebody is doing to us.

Q: *How and when did the work on HF begin?*

Simon: It all started fairly recently, actually. Around May this year (2020), I started talking to a small group of people involved in: myself, folks at Tweag and the folks behind the reborn Skills Matter. So we got together and started thinking, *"Maybe something like the Haskell Foundation would be really helpful to have?"* And then we wrote a white paper about it.

Within a couple of weeks, we started to share the paper and ideas about it with the members of the community. So we very rapidly grew, because to everybody we spoke to, we offered to join the working group and involved them in future conversations. Eventually, it wasn't long before we were starting holding fortnightly discussions.

I think we started with the Haskell.Org Committee. We first spoke to Jasper, the Chair of the committee, and the other members. Then we went around the other obvious stakeholder groups and the Haskell committees: the Core Libraries' committee, the GHC Steering committee, Cabal, and Stackage.

It all took a bit of time because, at that stage, we didn't want to publish something and then have everybody only read about it in the newspaper and think, *"Oh, I don't know about that."* We'd rather say: if you would like to be part of this, come and help shape it. Because we really didn't have the answer. It was more like, "We have some challenges and some ideas about things we might do about it; would you like to share with us in doing something together that will help address those challenges?" So it was a process of accretion, a larger and larger group of people to be involved in, that was the idea.

Q: *You covered the reasons for creating HF in the launch talk, but could you briefly share your opinion of them here? Why did you decide that it's time for HF?*

Simon: The Haskell community is like my family. I used to know every single person in it. And now it's much, much bigger. And that's great because it means that more people are using Haskell for more real things, but it also brings the challenges of scale. So I ended up trying to distil them down to four underserved needs:

1. **The full user experience, the user journey.** When somebody starts using Haskell, do they find the tools smooth? Do they work together well? Is it well documented? Is the training adequate? And so forth.

> I think there were lots of people who are interested in pieces of that picture, but no one who really stands in the shoes of the user and as well gives the voice to the user who is going through that journey.

2. **Technical infrastructure and glue**. We have lots of individual groups that are really efficient at building Haskell tools: e.g. **GHC** – my own true love – Haskell compiler is one of them; Cabal, Stack, Hackage, Haddock, the Haskell IDE and many more. There are lots of pieces, but they are served by individual groups that work really hard and very professionally. But somehow we lack connective tissue to make sure that all these pieces of any structure will work together well. Sometimes we accidentally mess each other up, and that's really sad when that happens.

3. **Community glue**. We've got quite a rich, diverse community, but no one is responsible for glueing us together. And I often feel that volunteers can get an incredible amount of work done. But also volunteers need a certain amount of coordination. They work together better if there's someone, or some group, or some mechanism for them to cooperate.

4. **Resources and funding**. The Haskell community and its ecosystem have got to the stage where we have so much to do that it's really hard for volunteers to sustain the required amount of effort and attention to detail. And so what do we need? We need some people whose full-time day job is to look after some of these boring but useful things. But if it is going to be their day job, somebody's got to pay for it. Who's going to do that? One possibility is to have a credible foundation, a non-profit organisation to which willing partners can give money, feeling confident that it will be well spent in the service of the user journey. And up to now, we haven't really had such a credible donee. That's part of what I hope the foundation might do.

So there were four particular things that we thought might be helped by having something like the foundation.

Q: *Did movements like "Simple Haskell" or "Boring Haskell" have their effect on creating HF?*

Simon: I don't think those movements did specifically. But the strong point of the residence is that they were concerned with the users' journey. They're rooted in the idea of "let's not make Haskellers run away by pointy-headed people like Simon who think that impredicative types are really cool". But let's make sure that we have a vehicle that is suitable for building large scale software in production. "Simple Haskell" or "Boring Haskell" movement is one approach to doing that. I'm not sure that everybody agrees with that approach, but it's one. And it clearly is an approach that's started in "let's make it useful, but let's concentrate on usage rather than on demand or supply". So to that extent, totally on board. But I wouldn't say that those two movements were specifically at the root of the logic of the foundation.

Q: *How did you choose the people who are willing to help the foundation?*

Simon: Essentially, we tried to contact every stakeholder who had an influential role in the Haskell community. I'm sure we left people outside; I am cautious about saying this. It's really hard to be comprehensive. We've got Haskell.org first, then the Core Libraries committee and the GHC steering committee, the Cabal and the Stackage folk, Haskell IDE team, the Haskell Weekly News, the Hackage trustees, and the Haskell admins.

We wrote a list, in which we tried to include everybody who we thought had some kind of collective identity and made a contribution to the Haskell community. That doesn't scale very well because it's lots of individual conversations. And we flipped a bit by going public at the Haskell Exchange. Now we're hoping to hold those kinds of conversations more broadly with the entire Haskell community.

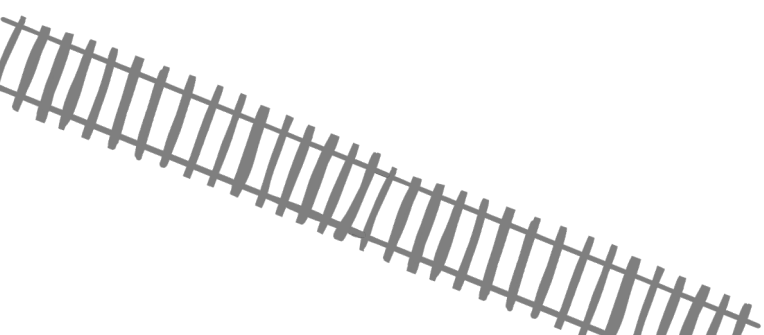
I don't know how to say this strongly enough, but it's really important that we, the community, feel that the foundation is something we are doing together, not that it is something that somebody is doing to us. It is not cast in stone. It is not a piece of concrete. It is shapeable and can be shaped. We are trying really hard to be transparent in decision making, governance and how you can get involved. I'm not saying they're necessarily successful in everyone's perception. But the goal is that it should be a **community-driven process**.

Q: *How will the success of the Haskell Foundation be evaluated? How would we be able to say "HF is doing what it is supposed to do"?*

Simon: I often ask this when people submit a research proposal. I say, "*How would I know if your project has succeeded?*" I want crisp criteria, but I don't think I can offer you a crisp criterion for whether the HF succeeds. I want the Haskell community to be friendly, open, inclusive and welcoming. And, of course, that's a matter of degree. Probably many people would say that it is already ideal, and some people would say that it isn't. And it's a continuum, and different people have different views. But I really would like the foundation to make a major contribution in that direction. I would want no one to bypass Haskell because its tooling is clunky, or it is on ramps or too forbidding.

So **increase in adoption**, I suppose, would be one criterion. The second one is that **more companies are visibly using Haskell** or maybe even signed up as sponsors, although supporters of the foundation. But none of these is "we succeeded, or we didn't". It's not black and white, I'm afraid.

The single most important thing for me is that HF acts as a glueing entity that makes us all feel that we're contributing to a shared endeavour in which we have a voice and from which we get goodness rather than another free-standing, well-meaning organisation that's sitting on the side somewhere.



Q: *Does this mean that you would be really happy if more people would use Haskell and find it more approachable?*

Simon: Yeah, yeah, it does! I mean exactly that.

My life's work is the idea that ***purely functional programming*** is an approach to the entire enterprise of writing programs, of telling computers what you want them to do; that is radical and elegant and worthy of greater attention than it has in the past received.

That idea has been seeping into mainstream programming brains over the last 30 years quite effectively. And Haskell has been an excellent vehicle for making that happen. So in the end, I don't know whether it's that everybody will be programming in Haskell, but I think there's room for more mindshare for Functional Programming. And therefore, the HF promoting Haskell is not a narrow thing: *"We want them to use our product"*. It's a broad thing — we want them to get the idea of purely functional programming and use it to make their lives better.

My life's work is the idea that purely functional programming is an approach to the entire enterprise of writing programs. That is radical, elegant and worthy of greater attention.

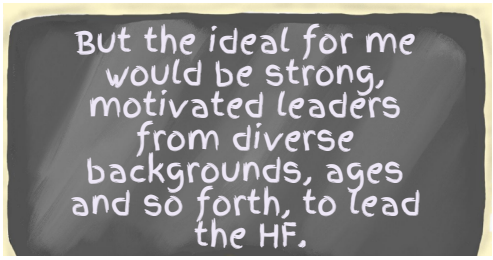
Q: *What are your personal plans for the foundation? How actively are you going to be involved in the Board decisions?*

Simon: That's a slightly hard one to answer.

There's a question of how bootstrapping the HF gets going. I played a role in that because I'm kind of visible in the Haskell community and I feel honoured to be in that position and privileged to be able to play a small role in helping to get it going. To bootstrap our process, as I described in the launch talk, the working group asked **Ed Kmett**, **Simon Marlow** and myself to appoint an Interim Board.

Besides us, the Interim Board also includes other well known and trusted people in the Haskell community: **Chris Dornan**, **Gabriele Keller**, **Jasper Van Der Jeugt**, **Stephanie Weirich**, **Lennart Augustsson**. The role of the Interim Board is to appoint the permanent Foundation Board, which will include 12 members. And we're seeking nominations at the moment.

I am currently chair of the Interim Board, but I'm definitely going to continue to be involved in HF in the longer term. The Haskell community is my second family, and I'm not going to bow out of it. But nor would I like to feel that I'm essential to it either. Otherwise, it's a bad single point of failure. I'll be delighted if we found a board that wouldn't include me, but would be well led and strong. I've got that far to decide whether I'll self nominate to be on the foundation board, let alone whether the Interim Board will appoint me. *(*laughs*)*



I would be thrilled to bow out at that stage to be a quiet voice behind the scenes, which we shall see.

Q: *Haskell Foundation sounds like a lot of work, and it took a lot of time and effort. How did you manage to do all that with full-time jobs?*

Simon: Yeah, that's true! But that's also true for a number of people in the Haskell community, who work on GHC, Cabal, Stack, Haskell IDE and so forth. The Haskell community is built out of volunteers. There is an old phrase, *"If you want to get something done, ask a busy person."*

So I think it's just more of the same that all of the people who have contributed to the foundation are busy people, but they've given freely of their time.

Emily Pillmore and **Tim Sears** have been particularly generous in giving their time to this, because the thing is: it's easy to have a meeting, and everybody says *"we should do this, that and the other"*, but somebody's got to actually do it! Emily and Tim have been among the people who've done well. I'm really grateful to them, but it is a broadly based thing. So it isn't just that three people have done everything, some people have worked on the website, **Jasper** led the affiliation track, etc. It's just how volunteer movements work. And we should be very grateful to the people who've devoted their time to making it happen.

Q: *Do you have any plans on how to make the process of onboarding and encouraging volunteers more enjoyable for everyone?*

Simon: I think that's a major topic for the HF. And I don't think I've got any quick answers for you now. But there is a track which **Rebecca** is involved in. She made a recent post on the HF discuss mailing list, which everybody reading this magazine should be subscribed to, and in which she is describing some aspects of volunteer onboarding.

There's a little working group thinking about how to make the process of being able to volunteer for things in the Haskell ecosystem not to big up the foundation, but just to make the Haskell ecosystem better, how to volunteer and how to make that process easy and rewarding. It has to be fun! And you have to get some level of recognition and a little or no aggro *(*laughs*)*.

I think if anybody reading this article is interested in that, it would be great just to join the working group.

Q: *What are the criteria for diversity in the board?*

Simon: We tried hard to write down criteria for nominations. So I'd encourage readers to look at the call for nominations on the HF site because a fair amount of work goes into trying to say *"we don't want to stick our fingers in the air"*. We want to have some written criteria so that nominees can say, *"I think I meet these criteria"*. For example, they meet A, B and C, maybe not D, so then the Interim Board can evaluate against those criteria.

Diversity is certainly one of them. But we haven't set quotas; we haven't said *"We will have three people from the USA and two from Europe or we have two people with fair hair and four people with brown hair"*. We've just listed a number of constituencies that we would like to see represented on various axes: gender, geography, age, stakeholder groups to represent users or suppliers, the academic versus industry. And we wrote down as many as we could think of. Any one person is not going to represent all of those. And inevitably, 12 people are not going to provide a completely balanced representation of all possible axes here.

So, no, we haven't established quotas. We're just going to do our best to make choices that balance those probably conflicting criteria as best as we can.

Q: *Among the values of HF is transparency. How much of this value extends to the board members selection process?*

Simon: I don't think we'll make nominations public. We didn't say that we would nor do we want to give rise to a new ad hominem personal debate about why you appointed X instead of Y, I'm not sure that would be helpful to us.

So I'm hoping that because we went to some effort to get the Interim Board appointed through a whole other layer of mechanism, we got people in the board that everyone will be prepared to say, *"OK, I haven't seen all the nominations, but I just trust this group to have done at least a reasonable job"*. And then that board, in turn, will appoint successors according to criteria it will establish probably closely based on the ones we have.

The diversity of all of these dimensions is really important, this whole point, if the foundation comes to be seen as being run by a particular interest group, it will fail. So, yeah, tricky *(*laughs*)*, because I'm sure it will not be as perfectly diverse as we would like.

Many thanks to Simon for such great thoughts and revelations shared with us today! Both Simon and we encourage you to subscribe to **HF-announce** and **HF-discuss** mailing lists [1] to get updates about news in HF.

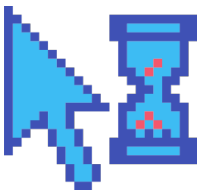
Moreover, don't forget to nominate yourself for a **board member** [2] or even as **Executive Director**! And remind your friends to do so. As Simon pointed, it would be crucial to have a diverse board, so it doesn't matter how experienced you are in Haskell, we need both beginners and experienced industrial users to represent different groups in the board. If you have passion for the language and want to help achieve HF goals, join in!

We wish all the best to the noble idea of Haskell Foundation, and are looking forward to its future successes!

[1]: https://haskell.foundation/en/contact/
[2]: https://haskell.foundation/board-nominations/

~PROPOSALS TO IMPROVE GHC AND HASKELL LANGUAGE~

For today's theme, we want to discuss one of the proposals, potentially affecting lots of us — `\` of -lambda expressions with guards and multiple clauses (-XMultiWayLambda) [1] by **Jakob Brünker**.



Most Haskell developers use the handy **LambdaCase** GHC feature daily, which allows writing clean pattern-matching for the single argument functions, like in the example below:

```
fromMaybe :: a -> Maybe a -> a
fromMaybe def = \case
    Just x  -> x
    Nothing -> def
```

If you haven't heard about **LambdaCase**, the LambdaCase in the wild [2] blog post gives a solid overview of this extension with samples.

However, the power of **LambdaCase** is not always enough because it has a particular application area. And admittedly, GHC has a proposal for an alternative improved version — **MultiWayLambda** extension. The proposal introduces the *\of* syntax similar to *\case* but has more control, with the logical consequence of deprecating **LambdaCase** and **MultiWayIf** extensions as less expressible after the newer addition.

To give you an impression of the new syntax, let's look at how a simple function can utilise this new feature for good. Consider the definition of zipWith:

```
zipWith
    :: (a  ->  b  ->  c)
    -> [a] -> [b] -> [c]
zipWith _ [] _ = []
zipWith _ _ [] = []
zipWith f (x:xs) (y:ys) =
    f x y : zipWith f xs yx
```

You can see how the name of the function is repeated multiple times. This becomes problematic and noisy for functions with long names and/or lots of arguments. The *of-lambda* proposal offers a nicer way to implement such functions:

```
zipWith
    :: (a  ->  b  ->  c)
    -> [a] -> [b] -> [c]
zipWith f = \of
    [] _ -> []
    _ [] -> []
    (x:xs) (y:ys) ->
        f x y : zipWith f xs ys
```

Note that currently, you can't use **LambdaCase** to achieve the same level of clarity. It would only allow you to do so with the last argument. Alternatively, you can use *case-of* expressions as well, but here lambda saves us a few intermediate variable names. Moreover, you can't smoothly pattern match on multiple arguments with *case-of*.

But it is not all that the proposed extension can do. Another benefit of the *of-lambda* syntax is that it also supports guards-only expressions, unlike *\case*. The ultimate goal of this proposal is to consolidate two existing GHC extensions **LambdaCase** and **MultiWayIf** into a new single extension, making those two deprecated and probably rolled out eventually.

While users may be excited about this new feature, how it can simplify both the user code and the GHC codebase, there are a few concerns.

At some point, the discussion around the proposal tended to deprecate the existing syntax of **LambdaCase** by forcing the use of parentheses around patterns. **LambdaCase** is one of the most beloved and popular extensions — the most desirable extension to be enabled by default according to the 2020 State of Haskell Survey [3] results and already the 6th most commonly used extension [4] in default-extension on Hackage. So it's no surprise that not everyone is okay with deprecating such a widely used feature. These changes affect a massive number of people, at least in the way that a lot of work needs to be made from the users' side — to align with the changes, even if **LambdaCase** is precisely enough for your case and there is no need for an immense extension.

If these discussions stick, after the *of-lambda* proposal is accepted and implemented, users will need to rewrite the previous fromMaybe example with extra () around the patterns:

```
fromMaybe :: a -> Maybe a -> a
fromMaybe def = \case
    (Just x) -> x
    Nothing  -> def
```

The required changes are understandable. But we all love **LambdaCase** because it allows us to write clean and concise code, and it is not always worth it to give up the comfortability. Notwithstanding the fact that deprecating old syntax will break tons of packages. Which is, compared to the first argument of slightly better ergonomics, sounds quite huge.

A comment under the GHC proposal [5], asking not to deprecate LambdaCase, collected 54 upvotes, meaning there are a lot of people who also don't support the removal of this feature. This wish doesn't contradict with having an improved mechanism of lambdas; instead, users want to keep the useful tool that they use ubiquitously.

We will keep you updated about the changes in this proposal, and hope that spreading this information wider and speaking with Haskell users directly would help with gaining an understanding of the real needs of the people, which can be acted upon accordingly.

In any case, there is a lesson to learn from this proposal. Innovating and improving the language is a noble deed. However, it is more efficient and healthy for the tool to include its users in the decision-making process rather than just notify post-factum.

Ultimately, such massive behaviour changes could only happen if the following measures are to be taken:

Possible breakages are discussed in a more public place, and the voice of real users is heard and taken into consideration

In case if breakage is inevitable, there should be tools for automatic code refactoring

If you have thoughts about this proposal, speak up. Maybe we all can come up with a better compromise that would work for the greater intersection of people. We know that maintaining GHC breakages is tough and time-consuming, and it can actually be avoided by other approaches to the new extension. In the case of agreement between users and implementers, we could have a brand new feature and keep enjoying our beloved **LambdaCase**.

What a wonderful world it would be!

[1]: https://github.com/ghc-proposals/ghc-proposals/pull/302
[2]: https://storm-country.com/blog/LambdaCase
[3]: https://taylor.fausak.me/2020/11/22/haskell-survey-results/
[4]: https://gist.github.com/nomeata/3d1a75f8ab8980f944fc8c845d6fb9a9
[5]: https://bit.ly/lambdacase-comment

WHINE Sommelier

~Degustate different libraries and check them against our appetite*~

If you need to parse JSON in Haskell, **aeson**** is the standard choice (and, frankly speaking, the only pragmatic choice). We want to see how this standard go-to elect meets the requirements to be user-friendly and ready for real-world needs. So here goes our review.

The internal architecture overview: JSON parser in **aeson** is based on the **attoparsec** library that provides fast parser combinators for parsing ByteString and Text. In addition to parsers, **aeson** contains FromJSON and ToJSON typeclasses for converting between Haskell values and the unified JSON representation in the library.

Let's review **aeson** under different angles and see how suitable it is for production needs! We are going to examine the latest version from Hackage, which is 1.5.4.1 at the moment.

* https://bit.ly/btg-lib
** https://hackage.haskell.org/package/aeson

Documentation

Documentation is an essential part of evaluating the user experience of the library. There are lots of ways to help your users by providing different types of documentation.

Aeson has a short top-level description in the .cabal file and README. However, the primary and most valuable piece of documentation is in Haddock for the main module, which is pointed out in the description. Haddock is the Haskell way to document your code, which later is available nicely on Hackage.
Module documentation is good, though it does not cover all exported functions and types. Some instances have @since annotations which are invaluable for production users. But this is applicable only for a few recent versions of **aeson**, which suggests that the annotations weren't maintained previously, and it would be tricky to get the information of the version where the function was introduced first.

Sometimes the default documentation is not quite sufficient to start writing non-trivial decoders straightaway, but multiple comprehensive blog posts can help you in this. Check out Aeson: the tutorial [1], A cheatsheet to JSON handling with aeson [2] and many others.

The documentation contains a lot of code examples, which is neat! But they are not tested automatically. Using **doctest** for **aeson** would be a considerable improvement in maintainability of doc examples and up-to-date information!

Aeson is a de-facto standard library. Another good addition to proudly carry the title would be the comparison with other JSON decoding and encoding libraries, such as **hw-json** and **waargonaut**. To be a high bar library, it is crucial to be forthright about the (dis)advantages of using **aeson** over other options, so the choice of a JSON library would be more reasonable to users' needs.

Ease of use

Next, we shall evaluate how comfortable to get started with the library and how friendly it would be in the long term for direct users.

The library provides quite a powerful API for parsing arbitrary data.

There are a lot of functions and type classes exported, which strews a flexible fundament for building encoders/decoders of different complexity. However, we'd say that you need to train quite a lot in writing parsers manually and read a few blog posts to understand the design truly and to become fluent with the API. For example, the JSON AST type Value has a naive FromJSON instance. Knowing this fact and understanding the instance behaviour is crucial for manually parsing nested JSON objects, for example, `[ {"key": 42, "val": "foo"} ]`.

The **aeson** API is stable, so there are no problems in frequent upgrades.

The library has several C files in dependencies for faster JSON parsing. Still, they are included conditionally only for supported platforms and compilers, which make aeson cross-platform and easy to depend on.

Aeson is infamous for its poor error messages, which could be challenging for the library beginners. However, there are external solutions like **aeson-better-errors** that are improving the situation.
Generally, even if **aeson** misses some of the features, there usually exist some integrated solutions in the wild. For example, **aeson** doesn't provide JSON pretty-printer out-of-the-box, but luckily, you can add one more library **aeson-pretty** to have this. In total, there are a lot of libraries that integrate with **aeson**, so the ecosystem support is vast and comprehensive.

Regarding the comfortability of module usage: to get the main **aeson** functionality, it is enough to import the Data.Aeson module. However, the library is not designed for qualified imports. And at the same time, it has name conflicts with other libraries sometimes, so you often end up with the mixed imports: qualified import of the main **aeson** module and a single unqualified import with restricted items as qualified operators doesn't look very pretty.

```
import qualified Data.Aeson as Aeson
import Data.Aeson ((.=))
```

Moreover, if you want to write a custom parser but not the instance, you need to import Data.Aeson.Types (and not even Data.Aeson.Parser), so the ergonomics suffer a bit here.

[1]: https://artyom.me/aeson
[2]: https://williamyaoh.com/posts/2019-10-19-a-cheatsheet-to-json-handling.html

Maintenance

From the maintenance point of view, **aeson** is an exemplary library.

Aeson follows *PVP (versioning policy)* and provides both lower and upper bounds for all its dependencies, so building **aeson** gives predictable and robust results.

The CHANGELOG is maintained in good faith, and it is detailed and gives sufficient information about what happened in each new release. The migration guide for the major releases is not written as clearly as it could've been, though it is possible to see what needs to be done from the neat CHANGELOG as it is.

The development of the library is active, and it has several responsible maintainers who are looking after the library constantly. If you have any problems with **aeson**, you can submit them to issue tracker, most likely to receive the answer. But it still has open issues dated 2014, and some issues don't have any comments at all. Moreover, the library has a dozen open PRs, and some of which stalled since 2016; although it is understandable for an extensive and popular across-the-board library, ideally it would be nice to have them resolved in some way.

Aeson has a CI set-up that checks its compatibility with GHC versions up to even GHC 7.8.4! So the newer releases are unlikely to break anything.

Code quality

The library code is quite clean and readable. **Aeson** compiles with -Wall without any warnings produced by GHC, though no additional warnings are enabled besides -Wall.

However, **aeson** has a lot of **HLint** hints (checked with hlint-3.2.1) and **Stan** warnings (checked with stan-0.0.1.0). For example, usages of unsafeCoerce, many usages of undefined. And the standard API operators don't have explicit fixity declarations.

Testing is also covered well in **aeson** (though, no *doctests* as mentioned before). The documentation contains a few Haddock warnings. So, even mature libraries can improve quality in many ways!

It is not a secret that **aeson** is quite a heavy library. Unfortunately, it has a lot of dependencies, and their number is only growing (e.g. the recent addition of **these** to dependencies). As a result, the library itself takes a noticeable amount of time to build. So, if you want to write a small JSON-parsing script quickly, you will have to wait a lot for the library and all its dependencies to finish compiling, so this is not for impatient people.

Fortunately, there are some options here. If you want to have a lightweight JSON parsing, you can use **microaeson**. If you need only JSON encoding, you can use the recently released **jsonifier** package that provides blazing fast JSON encoding.

Summary

Aeson showed itself as a very fine and stable library!
But as *the standard JSON parsing library*, **aeson** is not ideal, and can do better. There is a big room for improvement, and a lot of people will benefit from patches to **aeson**, so if you have ideas for refinements, feel free to contribute as maintainers seem to be considerable!

Documentation: 7.5 / 10 (Solid)

Ease of use: 6.5 / 10 (Good)

Maintenance: 9.5 / 10 (Amazing)

Code quality: 5 / 10 (Fair)

Summary: 7.1 / 10 (Solid)

The -Wall Street Analytics

GHC options, warnings and flags

In this edition of the "The -Wall Street analytics" column, we want to shed light on the -Wunused-packages [1] GHC warning.

This warning was introduced in GHC 8.10. However, it's not enabled by default, nor it's included in -Wall at the moment.
To enable it universally for your project, add the following lines to the .cabal file (we recommend putting such warnings into a common stanza [2]):

```
if impl(ghc >= 8.10)
  ghc-options: -Wunused-packages
```

If you create Haskell packages using *Summoner*, you will have this warning configured properly in your .cabal file out-of-the-box.

After enabling -Wunused-packages, GHC will warn you on having unused dependencies in your Haskell packages. The output will look like in the example below:

```
warning: [-Wunused-packages]
  The following packages were specified
  via -package or -package-id flag
  but were not needed for compilation:
    - async-2.2.1
```

Afterwards, you can go and remove the redundant dependency from the stanzas, specified in the warnings.

But why should you remove unused dependencies in the first place? This is an important thing to do, because having a dependency is not free, and it is crucial to keep it clean in your library or application for many reasons. First of all, they can be heavyweight and therefore, add a significant amount of extra time to your project compilation.

Moreover, if it is a library, users may consider using a different one instead, only because of heavy dependencies. Also, redundant dependencies can break your code implicitly when migrating to newer versions of other dependencies or even GHC itself.

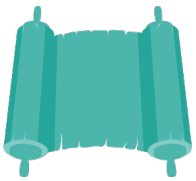
The only thing to keep in mind with this new option is that this sanity check doesn't warn on dependencies that you don't use directly, but which are used by some other dependencies of yours. So implicit dependencies are invisible for -Wunused-packages. But hey, who said it's not possible to improve the situation? ☺

[1]: https://bit.ly/wunused-packages
[2]: https://vrom911.github.io/blog/common-stanzas

LIBRARIAN

~ Discussions and review of proposals to standard libraries ~

As the standard library, **base** has a special place in the hearts of many Haskell developers. But like any library, it needs to be refined from time to time. However, as those changes need to be weighed against the fact that it is massively used, there are two sides of the coin. On the one hand, some long-awaited changes to the APIs are welcomed by its users, but on the other hand, they can break workflows and programs. That is why proposals to **base** are discussed thoroughly.



This time we would like to review the Integration of helpers to operate on Newtypes in base [1] proposal by **Hécate**. It suggests the addition of a module called Data.Newtype with the helper functions to work with newtypes easily (e.g. un for unwrapping newtypes).

Newtypes are one of the crucial Haskell features, and it enables multiple design patterns, which you can read about in the Haskell mini-patterns handbook [2]. That is why this proposal is a handy addition to the friendliness and completeness of the standard library.

The improvements suggested in the proposal are inspired by the existing newtype helpers [3] from the alternative standard library called **relude**. So the proposed changes are already battle-tested by different people and proved to be useful.

Despite the good intention behind the proposal, however, many people seem to not be in favour of these additions for different reasons. One mentioned cause against adding new functions is that un @Int is not better than coerce @_ @Int, though we don't think that many people would agree with this statement, but most probably they are not on the mailing list to do so. Another reason is that the function un not only unwraps but also wraps into a *newtype*. This can lead to unforeseen results if you are not familiar with the Coerce typeclass.

There was a brief discussion in the past about implementing unidirectional coercible [4] in Haskell, and such a feature can indeed help with many cases, including *newtypes* helpers. But the non-welcoming reaction to the proposal feels like the Haskell community again lets perfect be the enemy of good. Instead of encouragement and discussions of possible ways to improve the situation along with helping as many developers as possible, most of the people involved in the mailing list's discussion find more and more elaborate reasons to prevent this proposal from happening.

The UX of Haskell developers can be improved significantly, and it is fantastic how much we can do with what is already implemented! But we can not always afford to wait for the ultimate solution to arise. Often iterative and incremental improvements can bring a lot of invaluable benefits.

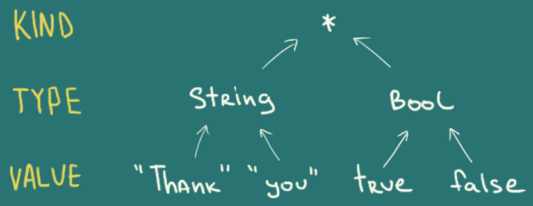
[1]: https://bit.ly/base-newtype-proposal
[2]: https://kowainik.github.io/posts/haskell-mini-patterns
[3]: https://bit.ly/relude-newtypes
[4]: https://github.com/ghc-proposals/ghc-proposals/issues/198

Pure Gold

by Impure Pics

KINDS

VALUES HAVE TYPES
TYPES ALSO HAVE THEIR OWN TYPES
TYPE OF A TYPE IS CALLED KIND



`*`

CONCRETE TYPES
INHABITED TYPES
(HAVE AT LEAST 1 VALUE)

```
Bool
[String]
Maybe String
Either String Bool
```

`* → *`

TAKES ONE TYPE OF KIND *
RETURNS A TYPE OF KIND *

```
[]
Maybe
Either String
```

`* → * → *`

GIVEN TWO TYPES OF KIND *
CREATES A TYPE OF KIND *

```
Either
(,)
Map
```

`* → (* → *) → * → *`

```
StateT s m a
```

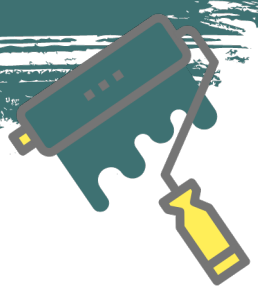
JUST BASIC EXAMPLES OF KINDS
IN THE WILD THERE ARE OTHER KINDS
E.G. CONSTRAINT KINDS, DATA KINDS

GHC IS IN PROCESS OF UNIFYING TYPES AND KINDS
THE NEW NAME FOR THE STAR * KIND IS "TYPE"

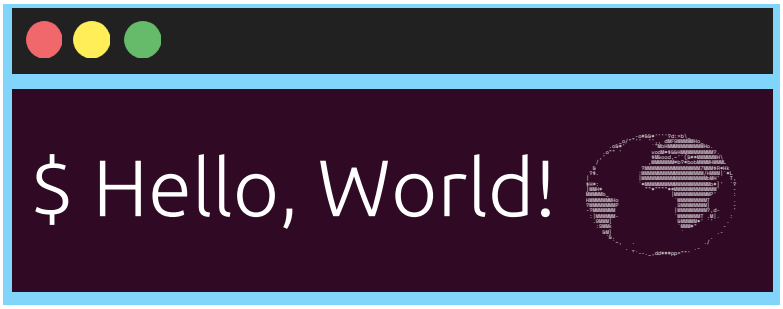
Impure PICS

FP Artist & Content maker

Distilling functional programming for the good of all



NEWS FROM *all around the*



Beginner-friendly notes, tools, tips and tricks. All you need to get learning Haskell easier.

How to play with Haskell

Haskell is a unique language that motivates people to learn something new each day. While experimenting with lots of different concepts, approaches and new ideas, we always find ourselves in need of a magic wand that can quickly create a buildable project template which will contain all necessary things, so we can focus on the idea and leave all the boring stuff to some tool.

Haskell has several ways of starting using the language locally:

- Define and evaluate expressions directly in **GHCi** — an interactive command-line REPL.
- Write definitions in a file (**module**), and load it in GHCi or compile directly to binary using GHC.
- Scaffold a buildable **project** with library, executables, tests and benchmarks.
- Create a runnable **script**.

At the beginning of your journey, you usually use the first two options, as they have the least overhead, and don't require you to know much besides the language syntax itself.

GHCi is an extremely useful tool for experimenting with various functions, calling them with different arguments to see the results, inspect types of expressions, and so on. But creating longer multi-line definitions in GHCi is awkward and doesn't suit for an active-development mode.

Here Haskell files (aka modules) come to play. You can define your own functions and even custom data types quickly in a separate file using your editor, and then load modules in GHCi to play with your functions in a familiar manner of Haskell REPL.

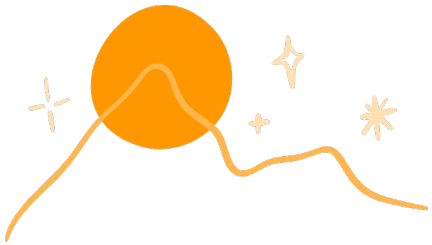
However, when starting to do something more sophisticated, a single module is not enough anymore. You often find yourself in need of creating a complete project with various modules and metadata content. This is the time for making a Haskell project.

or Package Project

Project and *package* are two concepts that come close to each other and sometimes could be mixed up due to their similar nature. Let's discuss the difference. The Haskell **project** is a more broad concept than the *package*. Any **package** is a *project*, though a project can contain one to many *packages*. If the *project* has more than one *package*, it is called a **multipackage project**.

Hackage — the Haskell central package repository (similar to npm) – has only a notion of *packages*. So even if you have a *project* with multiple *packages* and want to make it public and easily accessible to everyone from Hackage, you need to upload all *packages* separately.

Each *package*, in turn, comprises **stanzas** – Haskell *package* units. The examples of *stanzas* are **library**, **executable**, **test**, and **benchmark**. Each *package* can have zero or more units of each *stanza* type, but it must contain at least one *stanza*.



Most Haskellers create a few projects per month for personal usage. No matter if you are only learning the language or you are a library maintainer, the frequent need to create a ridge for your next project instantly is not much to ask for. Besides the Haskell code itself, the project usually contains .cabal files with the packages metadata (name, version, description), changelog, license, CI configuration, etc. You can imagine how creating all those files from scratch each time you need to scaffold a new project is a tedious job.

The Haskell ecosystem has a few options on how you can do that. It depends on the build tool you use, add-ons that you always want to have, and personal taste.

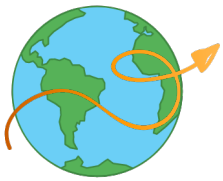
cabal init — the init command of the cabal-install [1] build tool. This command creates a package with a single executable by default. If you want to make it more configurable, you can use the global cabal configuration file, or search for the necessary CLI options that do what is needed (e.g. add tests or benchmarks, create a library, change the default license, etc.).

stack new — the new command of the Stack [2] build tool. This command creates a library with executable and tests and uses the latest stable snapshot of Haskell packages from Stackage.

Stack templates — creates a project based on some custom template with the configurable variables. This option is useful for creating multiple projects of a similar structure so that you can utilise templates for CLI tools, web-backend services, etc.

Summoner [3] — the CLI and TUI tool to scaffold fully configured batteries-included production-level Haskell projects. Use the summon new my-project-name command to create a new Haskell project interactively and choose all options you want. The options include all possible project configurations that previous options supply, plus additional features, like CI, appropriate options and more comprehensive metadata.

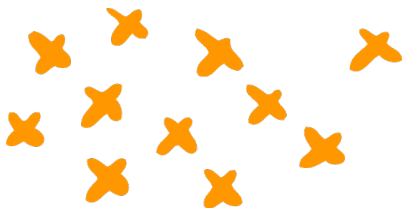
Summoner doesn't come with the default set of Haskell tools, unlike Cabal or Stack. However, you can install Summoner easily either by installing from Hackage, downloading a binary directly from GitHub releases or via package managers Ubuntu PPA and Homebrew.



Sometimes you don't need the full power of a project. You just want to have a single file (that maybe uses a few external libraries) to experiment or do some small task. For instance, analyse the content of local directories, fetch and parse JSON, process some CSV files, etc. For this task, you can use a feature called **Haskell scripts** — executable and runnable standalone Haskell modules.

A script starts with **shebang** — a special line to tell how to execute the script. Shebang is followed by the Haskell comment, containing the description of all external libraries. The description format is different for Cabal and Stack build tools. And then, the Haskell code itself.

Remembering the required script header can be difficult, but you can create a basic script with all the boilerplate using Summoner, and start hacking in seconds!



Happy Haskell hacking everyone, and let's make the process of learning/experimenting/creating with Haskell as easy as possible!

[1]: https://cabal.readthedocs.io/en/3.4/index.html
[2]: https://docs.haskellstack.org/en/stable/README/
[3]: https://kowainik.github.io/projects/summoner

Recently, Alexander Granin shared great news about his book "**Functional Design and Architecture**" – the print version is ready, and it looks fabulous! So, if you don't have a copy yet, now is the best time to purchase it.

FDaA is a gigantic piece of work by Alexander. The book is about building real-world Haskell applications focused on the *Hierarchical Free Monads approach*. The ultimate benefit is that this resource walks readers through all steps of creating applications, explaining different essential Software Development patterns and concepts along the way. This makes the book unique. And we suggest Haskell engineers familiarise themselves with the outstanding approaches of the book, even if they already have experience with other ways to develop the products.

Alexander Granin, speaker, researcher, author Expert Haskell and C++ developer

> ***Before committing into some approach, I want to know how to incorporate it into my application, what are the consequences I can expect long term, why should I use it, and what are other options I probably need to examine. I want to be sure that this approach won't make my code fragile with time.***

We spoke to Alexander about his work and he shared a lot of interesting details.

Q: *When did you come up with the idea for the book? How long have you been working on it?*

Alexander: It was 2015 when I realised that I couldn't really build programs in Haskell. I knew Haskell well and could use some of its sophisticated features, and I had some experience with many popular Haskell libraries. But it turned out that just knowing Haskell is not sufficient if you want to build a relatively big and complex product. Especially if you are going to work on the code with the team, and keep your project alive and maintainable for a long time. This task has been solved in the mainstream world, where developers collected a lot of practices and methodologies and organised them into the *Software Engineering* discipline. And this is what I felt was lacking in the Haskell community.

In short, Software Engineering is a set of well-understood and well-tested high-level approaches, application architectures, design ideas and best practices. Its part, *Software Design*, tells us how to compose a simple, testable, maintainable software with low coupling and low risks. If you are a senior software engineer, you can't just hack the code — you should think about separation of concerns, interfaces, subsystems, design and semantics of the code, and how the project will be evolving with time. This was a different mindset than we saw in the Haskell community 5 years ago. Not only because it resembles mainstream reasoning, but it also requires a certain level of understanding of how the software industry in the real world works.

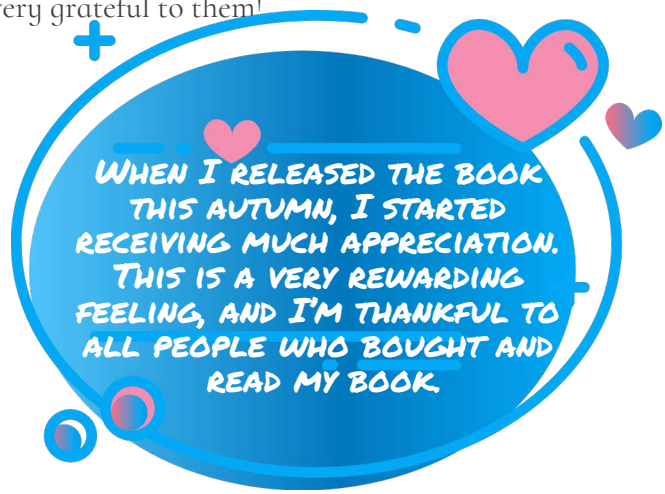
I started thinking about all this in 2015. Doing some investigations on this theme, and I wanted to find a set of ready approaches and patterns in Haskell similar to, let's say, in C#. There were a few, but nothing organised and elaborated systematically. No good showcase projects, no well-described success stories, and even the very

discipline of Software Engineering was kind of a persona non grata. Design patterns? General development principles? Avoiding shiny new stuff implemented several days ago in GHC? Why would one ever want this?

In 2016, I initiated work on my book. I used some ideas already existing in the community, but also I found my own ways to build a new discipline of Software Design in functional programming and Haskell. It took more than 3 years to write the book, not counting a two years pause I made from 2017 to 2019. I finished the book only in 2020, and for now, I can say it's the most *comprehensive source of knowledge* on Software Engineering in Haskell.

Q: *How were the community support and reception?*

Alexander: This is one of those questions that I can't answer honestly while staying nice. I'll try, though. I provided a lot of talks on Software Design in Haskell. I wrote a reasonable amount of code to demonstrate my ideas. I even created several real-world technologies wholly based on the methodology I'm presenting in my book. I've been talking about this stuff in chats, on forums, at conferences. For many years, I didn't feel like it was well-received. I experienced a fair amount of resistance to my ideas in the past, but fortunately, things have changed since then. In 2019, I began a Patreon program to finish the book, and many people became my supporters — I'm very grateful to them!



Q: *You mentioned that books like this are filling the gaps in Haskell resources. What do you think causes such gaps to exist, and do you think that FDaA fulfilled its purposes?*

Alexander: I strongly believe that there is an intense field of knowledge that we Haskellers were ignoring for a long time. In my book, I'm revealing only a part of it, but for sure, it's not the end. In fact, there are several other attempts to fill the same gap. There was **Real World Haskell** which is vastly outdated today. *Matt Parsons* is writing his own book **Production Haskell**. These and other materials are trying to show that Haskell is ready for real-world challenges.

We have a lot of scientific papers, I guess, 10 times more than in other communities.

But if you compare the number of books, Haskell will lose dramatically: 20-30 books about Haskell versus 500+ books on any popular language (like JavaScript or Python).
We have a lot of exciting articles and posts all around the web, but these posts don't provide you with a complete picture.

Before committing into some approach, I want to know how to incorporate it into my application, what are the consequences I can expect long term, why should I use it, and what are other options I probably need to examine. I want to be sure that this approach won't make my code fragile with time. Nope. The vast majority of the existing materials discuss some niceness and coolness of different aspects of Haskell, in isolation from the real needs of the industry. It's all about curiosity, not about professional software development.

My book goes much further than other resources. It connects two realities: a Haskell world of deep functional concepts and the real world with its problems and difficulties. One should be able to take my book and build a working piece of software, on time, in budget, with some guarantees to not being trashed after a while.

Q: *How approachable and interesting is the book for people who use the mtl approach of architecture? Would it help enlarge vision on the mtl style as well?*

Alexander: Mtl (also called *Final Tagless* sometimes) is one of the oldest approaches to structure the code. It's probably not an exaggeration to say that every complex codebase uses this approach to some degree. I understand why people choose it, and it works; however, my opinion is that it doesn't satisfy several essential requirements. You may read this as "there are subtle things which increase risks for the project".

My book proposes a different approach. I call it *Hierarchical Free Monads*. The idea is to have a notion of a functional interface similar to what we have in OOP with OOP interfaces. The separation of concerns should be complete and unbreachable because once you allow people to do dangerous stuff, you'll have a bunch of it in the big system. But the book is not agitating to forget mtl completely. In fact, there are sections where I'm talking about how to use mtl on top of a Free Monad solution, and provide reasons for that.
The book contains a lot of other approaches and design patterns, and it will definitely help

broaden one's horizons. And I believe it is very approachable to any Haskeller starting from the *intermediate* level. I did my best to explain things like no one before. This is not about the deepness of explanation, but rather about style and choosing the right words. I read a lot, I write a lot, and I see how many technical materials miss the needed things and describe what would better be omitted. This is even worse in the Haskell world.

Unfortunately, writing good documentation requires a very specific skill of, well, writing. Words, when used right, can do magical things. And you know, to write a good book you need to follow three simple rules. Unfortunately, nobody knows them.

> I don't go too deep by leveraging a very smart, math-like Haskell. I tried hard to avoid that because I value simplicity very much

Q: *Would you suggest your book to people with OOP background (with basic knowledge of Haskell syntax, maybe)?*

Alexander: Probably? It will be a little bit tough to follow the text if one has a shallow understanding of Haskell. On the one hand, the book is not about Haskell itself; it doesn't introduce the language. I tried to keep the narration as high level as I could because I wanted to focus on the high-level stuff rather than bolts and pieces. On the other hand, I don't go too deep by leveraging a very smart, math-like Haskell. I tried hard to avoid that because I value simplicity very much. There are some complex concepts used in the last chapters, though.

I can't really say how well the reading is, but I would recommend learning from Haskell books first. **Get programming with Haskell** by *Will Kurt* will be quite enough. Plus, maybe, some parts of **Haskell in Depth** by *Vitaly Bragilevsky* for a deeper understanding of the concepts.

Q: *Did you hear any "success stories" out of this book design already?*

Alexander: I knew I wanted my book to be practical. **Practical** means *overall goal is to have a working program in the end*. It also means "*every moving piece is in the right place; not a single thing is introduced just to satisfy curiosity*". I could say that being practical means that you can take some idea and derive it for your project. Best practices. Ready patterns. Useful approaches that are proven to be working in a real environment. When I started my book, I couldn't even imagine that I'd get great success with my ideas. But there are several companies in which I used exactly the approach from the book. This helped us a lot.

You might have heard about this success story, but let me maybe remind you. I'm working in **Juspay**, a well-founded Indian financial company. Its core logic is written on top of several Free Monadic frameworks. I was a person who designed those frameworks, and I've shown why it's beneficial to go this path. Many different projects were done using those frameworks, thanks to the great work of business logic developers. Juspay even open sourced those frameworks: PureScript **Presto** and PureScript **Presto.Backend**.

But let me share a secret. We're going to open source our Haskell framework as well — EulerHS. It is an older brother of my Hydra framework, which I created for the book as a showcase project.

In fact, they are like twins. We use EulerHS in our production, and we think it is a remarkable technology to share with the community. I'm personally responsible for that, so keep your eyes on my updates.

We are grateful to Alexander for his time and answers!
We are sure that many of the readers are intrigued by this book as well, so we are delighted to tell you that Alexander kindly shared a special promo code for you on 25% of discount on this book.

https://bit.ly/btg-promo-fdaa

Read good books, folks!

Humourmorphism

An applicative walks into a bar, approaches two sitting monads and asks, "Can I join you?"

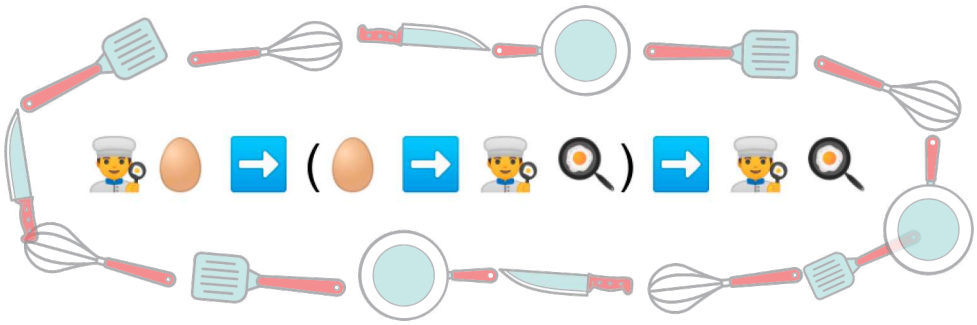
CHALLENGE yo self

Implement the **reverse** function using the **sort** function (and maybe some others).

Send us your solutions to ***xrom.xkov@gmail.com***, or tag ***@bind_the_gap*** on your solution in Twitter and we will highlight the most elegant and creative solutions in the following issue!

TYPE emojination

Guess the standard function by the following type, written in emojis:



SURVEYVOR

~ Monthly BTG survey, important community surveys and results ~

The survey of the month is the annual **State of the Haskell Survey**, created and maintained by *Taylor Fausak*. A lot of people submitted their answers, and you can read the summary of all responses here:

https://taylor.fausak.me/2020/11/22/haskell-survey-results/

If you interested in *cluster analysis* of the survey results, check out the following blog post:

https://www.ariis.it/static/articles/2020-haskell-survey-analysis/page.html

We also have a survey! But much smaller. This time we want to hear what you would like to see in **Bind The Gap** and how did you like this pilot issue. Please, use the link below to share your feedback.

https://bit.ly/btg-survey-nov2020

Service Information

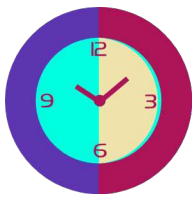
Date: Nov 13, 2020

Time: 21:59 UTC

When you get a recycling bin delivered you have to first take it out of the cardboard box and then put the cardboard in the bin and this is like a real life sequenceA.

@locallycompact

The next station



Event Sourcing

The upcoming month brings us the following Haskell community events:

Dec 2: Haskell Wednesday: Live coding tutorial & free time to mingle
by Berlin Haskell Users Group
https://bit.ly/events-dec-live-coding

Dec 7: Haskell Meetup #20
by Haskell Milano
https://bit.ly/events-dec-milano

Dec 9: #18 REMOTE Haskell Tutorial & Weihnachtsedition
by women in tech – Tübingen
https://bit.ly/events-dec-remote-haskell

Dec 9: Leverage the power of logic programming using souffle-haskell
by London Haskell
https://bit.ly/events-dec-london

The *Berlin Functional Programming Group* will have multiple events in **December**
https://bit.ly/events-bfpg

Closing Words

The pilot issue of **Bind The Gap** is brought to you by **Kowainik** —
Dmitrii Kovanikov and **Veronika Romashkina**.
We hope you enjoyed our first issue!

Besides BTG, we do a lot of open-source development, tutorials and guides writing, mentorship. You can visit our website to read more about our work:

https://kowainik.github.io/

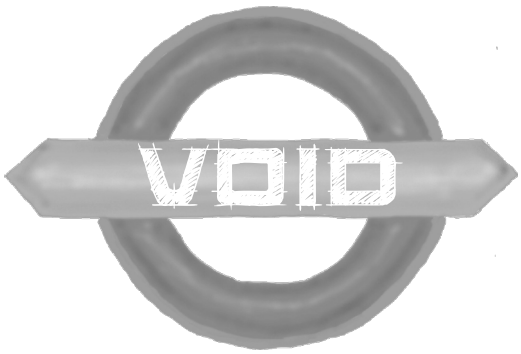
We have plenty of ideas and plans for future issues. Work on the magazine takes a lot of time and effort. So your support is highly appreciated! You can support our work and editions of BTG in particular on Ko-Fi or via GitHub Sponsorship:

https://ko-fi.com/kowainik
https://github.com/sponsors/vrom911
https://github.com/sponsors/chshersh

If you want to help with the magazine in any way, or you would like to have your own rubric and produce themed content monthly, feel free to reach out to us! You can contact us in Twitter **@bind_the_gap** or by dropping an email to **xrom.xkov@gmail.com**

See you in *December*, folks!

Way out →

This is 

where this issue terminates

ALL CHANGE PLEASE

Please remember to take all your

Monads

with you when you leave the train