Haskell for a New Decade

Stephen Diehl
@smdiehl

Berlin Functional Programming Group
A community of people that think about the future, rather than just about the present.
Software is Terrible and Getting Worse

- Late stage Moore’s Law.
- End-user software is bloated, slow, insecure, and increasingly tied to a legacy stack our industry cannot afford to change.
- Software is only getting more expensive to write.
- We are forced to live with the bad decisions of the past forever.
- Our industry has gotten sloppy.
- Open source programming culture has changed. We sold out.
- Software is a pop culture.
Medicine in 1300
Software "engineering" in 2020

Software is still at the blood-letting & leeching stage.
The Timescales of Progress

- Lambda calculus was discovered/invented in 1930.
- Second-order polymorphic lambda calculus was discovered by Girard in 1972.
- Ideas that are invented in functional languages find their way into other languages at about a 10-20 year lag.
- Proebsting's Law: Improvements to compiler technology double the performance of typical programs every 18 years.

Is it meaningful to ask if there is progress in programming languages? Or is our field merely a function of the fashions, whims, and tastes of our era.

Time scale of progress is sloooow!
The Past

• 1989 - A very different time in software.
  ○ Berlin Wall fell
  ○ Apple Computer Macintosh SE/30 released
  ○ Intel 80486DX released
  ○ MS-DOS 4.01
  ○ Lotus Notes released
  ○ CERN "Information Management: A Proposal" (i.e. The Internet)
  ○ First Haskell language proposal.

Haskell is an OLD language!
The Present

- 2020 - Some things have changed.
  - GHC 8.10.1
  - A stable industrial-strength compiler
  - Fast concurrent runtime
  - Build tooling
  - Extensible compiler plugins
  - Pseudo-dependent types
  - Improving editor integration
  - Thriving package ecosystem
A language community is an economic engine.

It’s a fallacy to reduce all progress to a single spoke on the flywheel. There needs to be uniform torque!

Research alone won’t move the wheel. Community alone won’t move the wheel

A PL community is a group of people who share in the success of common ideas.
What if anything is Haskell good for?

- Compilers?
- Document preparation?
- Prototypes for papers?
- Proof assistants?
- Financial services?
- Static analysis?
- Crypto money laundering?

Success?

For a 30 year old language, it’s very difficult to find many success stories!
Complete absence of death has multiple readings

Zombie Immortality

Time Lord Immortality
Future: Stagnation and Sclerosis

- The Standard ML fate.
- A curiosity left for students at university to study and then never use.
- The barrier to entry for GHC development remains impossibly high.
- The cartesian product of the semantics for all language extensions exceeds human cognitive capacity to maintain.
- Hackage ecosystem bitrots into an archive of abandonware.
Future: Steady State

- A small group of students learn Haskell at university. They leave university and go write Java in industry.
- Hobbyists at German meetups meet several times a year and talk about the latest functional pearl.
- The language is blacklisted as unfit for production at large companies because laziness is the boogeyman.
- Another dozen papers on building red-black trees with dependent types.
- We continue the debate about the word *monad* being renamed to “warm fuzzy thing” after every new monad tutorial is written.
- High turnover of small companies.
Future: Growth

What does success look like?

How do we do to get there?
A New Decade!

Open source development drastically outperforms the market.

Open source: open as in open door, not as in open bar.

Leave the campground better than when you found it.

A new generation of FPers!

Ideas and trends that will carry the ecosystem into the future.

What Haskell open source should I work on?
The Haskell-like Family Tree

1G

GHC Haskell

2G

Purescript
Idris
Agda

3G

Idris2

Language Evolution = Progress

- Namespace records.
- Implicit Prelude.
- Numerical tower.
- Modules.
- Dependent core.
Language Family

- We don’t know what we don’t know about the functional language design space.
- Most of the design space is unexplored.
- More compiler experiments are a good thing.
- All knowledge is derived from experience.
- Efficient backend code generation dominates the fate of functional languages.

Is 2020 the decade of practical dependent types?
Algebraic Effect Systems

• There are better models than MTL for effect handling.
  ○ [link](https://hackage.haskell.org/package/polysemy)
  ○ [link](https://hackage.haskell.org/package/fused-effects)

• Lots of optimisation and testing to do.
• Very few experiments programming with effects in the large.

```haskell
example :: Members ' [Trace, State Example, Error MyError] r => Sem r ()
example = do
  modify $ \s -> s {x = 1}
  trace "Logging message"
  throw MyError
```
GHC

- A **big** old quirky codebase.
- Everyone should build GHC from source at least once. [https://ghc.dev](https://ghc.dev)
- Subscribe to ghc-devs. You will learn a lot about how the sausage gets made. [https://mail.haskell.org/pipermail/ghc-devs/](https://mail.haskell.org/pipermail/ghc-devs/)
- GHC development sustainability worries me a lot. I don’t know how/if the language will continue in its current situation.
- GHC Haskell cannot remain *all things to all people* forever. The needs of different subsets of the community will eventually diverge under the current specification.
Compiler Performance

- On its current trajectory GHC may cross threshold of being almost unusably slow for large industrial codebases.
- Majority of compilation time is in the simplifier. All the big wins in GHC compile-times are to be had in optimising the simplifier.
- Compilation allocates a lot of memory.
- Shouldn’t need to allocate u-6tb1.metal instances with TBs of RAM just to compile a project.

*Pending crisis for industrial users?*
GRIN

- A whole-program optimization framework for multiple functional languages.
- A compiler back-end for lazy and strict functional languages.
- Retargetable and reusable functional compilation framework based on LLVM for functional languages.

https://github.com/grin-compiler/grin
https://github.com/grin-compiler/ghc-grin
https://github.com/grin-compiler/ghc-wpc
Editor Tooling

- ghcup is markedly improved.
- Cabal > 3.0 is markedly improved.
- Linking directly against the GHC & Cabal APIs used to a nightmare.
- [https://hackage.haskell.org/package/hie-bios](https://hackage.haskell.org/package/hie-bios)
- [https://hackage.haskell.org/package/ghc-lib](https://hackage.haskell.org/package/ghc-lib)
- Early work on Language Server (hls)
- HIE Files

```haskell
if impl(ghc >= 8.10)
  ghc-options: -fwrite-ide-info
              -hiedir=.hie
```
Machine Learning can learn from ML

- The Python ML ecosystem is reinventing functional compilers on top of Python for graph compilers.
  - JAX
  - Swift for TensorFlow
  - Torchscript
- There is a place for existing FP to enter this ecosystem with the interface being shared intermediate representations.
- MLIR (https://mlir.llvm.org)
- hasktorch/hasktorch
- google-research/dex-lang

Source: https://github.com/OpenTensors/ecosystem/blob/master/graph.png
Computational Integrity Proofs

- Niche branch of computer science research being actively developed.
- Non-interactive zero-knowledge proofs (zkSNARKs)
- Small and computationally efficient zero-knowledge proofs of arbitrary computations.
- Currently quite computationally expensive proof construction, will become viable for large scale computation mid-decade at the current rate.
- A complete Haskell framework for circuit construction evaluation: [https://github.com/adjoint-io/zkp](https://github.com/adjoint-io/zkp)

*My company maintains these libraries.*
Industry

- There is effectively no Haskell job market.
- The perception of Haskell in industry at large is not good.
- Large corporates are generally too risk averse.
- Early companies and SMEs.
- Start more companies, create more jobs!

Very small intersection

Entrepreneurs
Executives

Haskellers
Haskell 2030

Programming communities always like to believe our best days are ahead of us and our worst days behind us. But it’s the now that’s the issue and always has been.

Never accept the status quo in programming.

Dream of a better future in 2030.

Start building that future today.
Thank You

@smdiehl
@sdiehl

https://gist.github.com/sdiehl/5e0e25d4a35d25abd3408d542ccfd45