

Haskell Communities and Activities Report

<http://tinyurl.com/haskcar>

Thirty Third Edition — November 2017

Chris Allen	Mihai Maruseac (ed.)	Moritz Angermann
Francesco Ariis	Christopher Anand	Emil Axelsson
Gershon Bazerman	Heinrich Apfelmus	Jost Berthold
Ingo Blechschmidt	Doug Beardsley	Emanuel Borsboom
Jan Bracker	Sasa Bogicevic	Joachim Breitner
Rudy Braquehais	Jeroen Bransen	Erik de Castro Lopo
Manuel M. T. Chakravarty	Björn Buckwalter	Olaf Chitil
Alberto Gómez Corona	Eitan Chatav	Tobias Dammers
Kei Davis	Nils Dallmeyer	Richard Eisenberg
Tom Ellis	Dimitri DeFigueiredo	Dennis Felsing
Olle Fredriksson	Maarten Faddegon	Marc Fontaine
PÁLI Gábor János	Phil Freeman	Michael Georgoulopoulos
Andrew Gill	Ben Gamari	Mark Grebe
Gabor Greif	Mikhail Glushenkov	Jennifer Hackett
Jurriaan Hage	Adam Gundry	Bastiaan Heeren
Sylvain Henry	Martin Handley	Kei Hibino
Guillaume Hoffmann	Joey Hess	Nicu Ionita
Patrik Jansson	Graham Hutton	Anton Kholomiov
Oleg Kiselyov	Dzianis Kabanau	Yasuaki Kudo
Harendra Kumar	Ivan Krišto	David Lettier
Ben Lippmeier	Rob Leslie	Rita Loogen
Tim Matthews	Andres Löh	Simon Michael
Andrey Mokhov	Gilberto Melfe	Antonio Nikishaev
Henrik Nilsson	Dino Morelli	Ulf Norell
Ivan Perez	Wisnu Adi Nurcahyo	Bryan Richter
Herbert Valerio Riedel	Jens Petersen	Alexey Radkov
Michael Schröder	Sibi Prabakaran	Christian Höner zu Siederdisen
Jeremy Singer	Jeremy Shaw	Chris Smith
Michael Snoyman	Gideon Sireling	Lennart Spitzner
Yuriy Syrovetskiy	David Sorokin	Henk-Jan van Tuyl
Michael Walker	Jonathan Thaler	Chris Wong
Li-yao Xia	Ingo Wechsung	Yuji Yamamoto
Brent Yorgey	Kazu Yamamoto	Stack Builders
	Marco Zocca	

Preface

This is the 33rd edition of the Haskell Communities and Activities Report. This report has 143 entries, many more than in the previous edition. Of these, 38 projects have received substantial updates and 19 entries are completely new. As usual, fresh entries – either completely new or old entries which have been revived after a short temporarily disappearance – are formatted using a blue background, while updated entries have a header with a blue background.

Since my goal is to keep only entries which are under active development (defined as receiving an update in a 3-editions sliding window), contributions from 2015 and before have been completely removed. However, they can be resurfaced in the next edition, should a new update be sent for them. For the 30th edition, for example, we had around 20 new entries which resurfaced. We also had 2 entries resurface in this report after they have been dropped in the past two reports. We hope to see more entries revived and updated in the next edition.

A call for new HCAR entries and updates to existing ones will be issued on the Haskell mailing lists in late February/early March.

Work on a new, modern HCAR pipeline continues to develop. Once that is done, the next edition will look different, allowing for more expressivity. Details will follow on the usual communication channels, once they become available.

Now enjoy the current report and see what other Haskellers have been up to lately. Any feedback is very welcome, as always.

Mihai Maruseac, LeapYear Technologies Inc., US
<hcar@haskell.org>

Contents

1	Community	7
1.1	Haskell' — Haskell 2020	7
1.2	Haskellers	7
2	Books, Articles, Tutorials	8
2.1	Oleg's Mini Tutorials and Assorted Small Projects	8
2.2	Learning Haskell	8
2.3	School of Haskell	9
2.4	Programming in Haskell - 2nd Edition	9
2.5	Haskell Programming from first principles, a book for all	10
2.6	Stack Builders Tutorials	10
2.7	Haskell MOOC	10
3	Implementations	11
3.1	The Glasgow Haskell Compiler	11
3.2	Frege	12
3.3	The Helium Compiler	13
3.4	Specific Platforms	13
3.4.1	Fedora Haskell SIG	13
3.4.2	Debian Haskell Group	14
3.5	Related Languages and Language Design	14
3.5.1	Agda	14
3.5.2	Disciple	14
4	Libraries, Tools, Applications, Projects	16
4.1	Language Extensions and Related Projects	16
4.1.1	Dependent Haskell	16
4.1.2	generics-sop	16
4.1.3	Supermonads	17
4.1.4	Reifying type families	17
4.2	Build Tools and Related Projects	18
4.2.1	Cabal	18
4.2.2	The Stack build tool	18
4.2.3	Stackage: the Library Dependency Solution	19
4.2.4	Stackgo	19
4.2.5	hsinstall	20
4.2.6	yesod-rest	20
4.2.7	Haskell Cloud	20
4.2.8	Hackagebot 2.0	20
4.3	Repository Management	21
4.3.1	Darcs	21
4.3.2	git-annex	21
4.3.3	Octohat	21
4.3.4	openssh-github-keys (Stack Builders)	22
4.4	Debugging and Profiling	22
4.4.1	Hoed – The Lightweight Algorithmic Debugger for Haskell	22
4.4.2	ghc-heap-view	23
4.4.3	ghc-vis	23
4.4.4	Hat — the Haskell Tracer	24
4.4.5	LeanCheck	24
4.4.6	Speculate	25
4.4.7	Extrapolate	25

4.5	Development Tools and Editors	26
4.5.1	Haskell for Mac	26
4.5.2	haskell-ide-engine, a project for unifying IDE functionality	26
4.5.3	HyperHaskell – The strongly hyped Haskell interpreter	27
4.5.4	CodeWorld	27
4.5.5	Haskell Indexer	28
4.5.6	Brittany	28
4.6	Formal Systems and Reasoners	29
4.6.1	The Incredible Proof Machine	29
4.6.2	Exference	29
4.7	Education	30
4.7.1	Holmes, Plagiarism Detection for Haskell	30
4.7.2	DSLsofMath	30
4.7.3	Interactive Domain Reasoners	30
4.8	Text and Markup	31
4.8.1	lhs2 \TeX	31
4.8.2	Lentil	31
4.8.3	Unicode things	32
4.8.4	Fast Unicode Normalization	32
4.8.5	Ginger	32
4.9	Web	33
4.9.1	WAI	33
4.9.2	Warp	33
4.9.3	Yesod	33
4.9.4	Happstack	34
4.9.5	Snap Framework	34
4.9.6	MFlow	35
4.9.7	PureScript	35
4.9.8	Sprinkles	35
4.9.9	nginx-haskell-module	36
4.9.10	Hapistrano	37
4.9.11	Template Toolkit	37
4.10	Databases	37
4.10.1	Persistent	37
4.10.2	Opaleye	38
4.10.3	Squeal	38
4.10.4	Riak bindings	38
4.10.5	Haskell Relational Record	39
4.10.6	YeshQL	39
4.11	Data Structures, Data Types, Algorithms	39
4.11.1	Algebraic graphs	39
4.11.2	Conduit	40
4.11.3	Transactional Trie	41
4.11.4	Concurrent Trie	41
4.11.5	Random access zipper	41
4.11.6	Generic random generators	42
4.11.7	Generalized Algebraic Dynamic Programming	42
4.11.8	Earley	44
4.11.9	Transient	44
4.12	Parallelism	44
4.12.1	Eden	44
4.12.2	Auto-parallelizing Pure Functional Language System	45
4.12.3	Déjà Fu: Concurrency Testing	46
4.12.4	The Remote Monad Design Pattern	46
4.12.5	concurrent-output	47
4.13	Systems programming	47
4.13.1	Haskell for Mobile development	47
4.13.2	haskus-system	48

4.13.3	Haskino	48
4.13.4	STM32-Zombie	49
4.13.5	Feldspar	49
4.14	Mathematics, Simulations and High Performance Computing	49
4.14.1	sparse-linear-algebra	49
4.14.2	aivika	50
4.14.3	General Decimal Arithmetic	50
4.15	Graphical User Interfaces	51
4.15.1	wxHaskell	51
4.15.2	threepenny-gui	51
4.16	FRP	52
4.16.1	Yampa	52
4.16.2	reactive-banana	53
4.16.3	Functional Reactive Agent-Based Simulation	54
4.17	Graphics and Audio	55
4.17.1	diagrams	55
4.17.2	csound-expression	56
4.17.3	Chordify	56
4.17.4	The Arpeggigon	57
4.17.5	Movie Monad	57
4.17.6	Gifcurry	58
4.18	Games	59
4.18.1	Nomyx	59
4.18.2	EtaMOO	59
4.18.3	Tetris in Haskell in a Weekend	59
4.18.4	Barbarossa	60
4.18.5	tttool	60
4.19	Data Tracking	60
4.19.1	hledger	60
4.19.2	gipeda	61
4.19.3	arbt	61
4.19.4	propellor	61
4.20	Others	62
4.20.1	ADPfusion	62
4.20.2	leapseconds-announced	62
4.20.3	Haskell in Green Land	63
4.20.4	clr-haskell (Haskell interoperability with the Common Language Runtime)	63
4.20.5	Kitchen Snitch server	64
4.20.6	FRTrader	64
4.20.7	shell-conduit	64
4.20.8	Hapoid	64
4.20.9	Hanum - OSM Dynamic Attributes Linter	64
4.20.10	tldr	65
5	Commercial Users	66
5.1	Well-Typed LLP	66
5.2	Keera Studios LTD	67
5.3	Stack Builders	68
5.4	McMaster Computing and Software Outreach	69
6	Research and User Groups	70
6.1	DataHaskell	70
6.2	Haskell at Eötvös Loránd University (ELTE), Budapest	70
6.3	Artificial Intelligence and Software Technology at Goethe-University Frankfurt	71
6.4	Functional Programming at the University of Kent	71
6.5	Functional Programming at KU	72
6.6	Functional Programming Laboratory at the University of Nottingham	72
6.7	fp-syd: Functional Programming in Sydney, Australia	73

6.8	Regensburg Haskell Meetup	74
6.9	Curry Club Augsburg	74
6.10	Italian Haskell Group	74
6.11	RuHaskell – the Russian-speaking haskellers community	74
6.12	NY Haskell Users Group and Compose Conference	74
6.13	Japan Haskell User Group – Haskell-jp	75
6.14	Tokyo Haskell Meetup – Casual, English-speaking monthly meetings in Tokyo	76
6.15	Functional Programming at the Telkom University	76
6.16	Haskell Serbia	76

1 Community

1.1 Haskell' — Haskell 2020

Report by:	Herbert Valerio Riedel
Participants:	Andres Löh, Antonio Nikishae, Austin Seipp, Carlos Camarao de Figueiredo, Carter Schonwald, David Luposchainsky, Henk-Jan van Tuyl, Henrik Nilsson, Herbert Valerio Riedel, Iavor Diatchki, John Wiegley, José Manuel Calderón Trilla, Jurriaan Hage, Lennart Augustsson, M Farkas-Dyck, Mario Blažević, Nicolas Wu, Richard Eisenberg, Vitaly Bragilevsky, Wren Romano

Haskell' is an ongoing process to produce revisions to the Haskell standard, incorporating mature language extensions and well-understood modifications to the language. New revisions of the language are expected once per year.

The goal of the Haskell Language committee together with the Core Libraries Committee is to work towards a new Haskell 2020 Language Report. The Haskell Prime Process relies on *everyone* in the community to help by contributing proposals which the committee will then evaluate and, if suitable, help formalise for inclusion. Everyone interested in participating is also invited to join the `haskell-prime` mailing list.

Four years (or rather ~3.5 years) from now may seem like a long time. However, given the magnitude of the task at hand, to discuss, formalise, and implement proposed extensions (taking into account the recently enacted three-release-policy) to the Haskell Report, the process shouldn't be rushed. Consequently, this may even turn out to be a tight schedule after all. However, it's not excluded there may be an interim revision of the Haskell Report before 2020.

Based on this schedule, GHC 8.8 (likely to be released early 2020) would be the first GHC release to feature Haskell 2020 compliance. Prior GHC releases may be able to provide varying degree of conformance to drafts of the upcoming Haskell 2020 Report.

The Haskell Language 2020 committee starts out with 20 members which contribute a diversified skill-set. These initial members also represent the Haskell community from the perspective of practitioners, implementers, educators, and researchers.

The Haskell 2020 committee is a language committee; it will focus its efforts on specifying the Haskell language itself. Responsibility for the libraries laid out in the Report is left to the Core Libraries Committee (CLC). Incidentally, the CLC still has an available seat; if you would like to contribute to the Haskell 2020 Core Libraries you are encouraged to apply for this opening.

1.2 Haskellers

Report by:	Michael Snoyman
Status:	experimental

Haskellers is a site designed to promote Haskell as a language for use in the real world by being a central meeting place for the myriad talented Haskell developers out there. It allows users to create profiles complete with skill sets and packages authored and gives employers a central place to find Haskell professionals.

Haskellers is a web site in maintenance mode. No new features are being added, though the site remains active with many new accounts and job postings continuing. If you have specific feature requests, feel free to send them in (especially with pull requests!).

Haskellers remains a site intended for all members of the Haskell community, from professionals with 15 years experience to people just getting into the language.

Further reading

<http://www.haskellers.com/>

2 Books, Articles, Tutorials

2.1 Oleg's Mini Tutorials and Assorted Small Projects

Report by: Oleg Kiselyov

The collection of various Haskell mini tutorials and assorted small projects (<http://okmij.org/ftp/Haskell/>) has received a manifold addition centered on Type Equality and Overlapping Instances.

Type Equality Predicates and Assertions

We describe several forms of testing two types for equality, and asserting that two types are equal or not equal. The predicates differ in how they compare type variables, whether they allow wildcards, etc.

Described predicates:

- `TypeEq t1 t2`: simplifies to `HTrue` if the type checker regards `t1` and `t2` as the same. The result is `HFalse` if the type checker can see the two types as different. The decision is postponed if `t1` and `t2` are not instantiated enough;
- `TypeEqTotal t1 t2`: immediately yields either `HFalse` or `HTrue`, regardless of how `t1` and `t2` are instantiated. A type variable is `TypeEqTotal` only to itself;
- `t1~t2`: asserts that `t1` and `t2` are the same, and, as a *side effect*, instantiates type variables in `t1` and `t2` to make the type the same. If the types cannot be made the same, a type error is raised;
- A predicate version of `t1~t2` – that is, `TypeEq` with the side-effect of instantiation of type variables – is incoherent;

[Read the tutorial online.](#)

[How to implement type equality.](#)

For and Against Overlapping Instances

Overlapping instances are so practically appealing because they express the common pattern of adding a special case to an existing set of overloaded functions. We illustrate the pattern on a real-life example of optimizing a generic library. The example demonstrates the conflict between two practically useful features, overlapping instances and associated data types.

Overlapping instances are controversial because they straddle a contradiction. They embody “negation as failure”: the general type class instance is chosen for the given type only when all more specific instances failed to match the type. Negation-as-failure presupposes closed world, or a fixed set of instances. However,

type classes are *open*: the user may add more instances at any time, in the same or different modules.

[Read the tutorial online.](#)

Another use case: [Comparing types by their shape.](#)

[Objections to overlapping instances.](#)

Type-level type introspection, equality and matching

We describe type-level representation of types: essentially type-level `Typeable`. The library lets us check if two types are equal or dis-equal, and compare them by shape. We may use wildcards in type comparisons.

The library exhibits type-level conditional and higher-order type families such as `Member` (which takes a type-level equality predicate as an argument).

[Read the tutorial online.](#)

[Using `TTypeable` to avoid `OverlappingInstances`.](#)

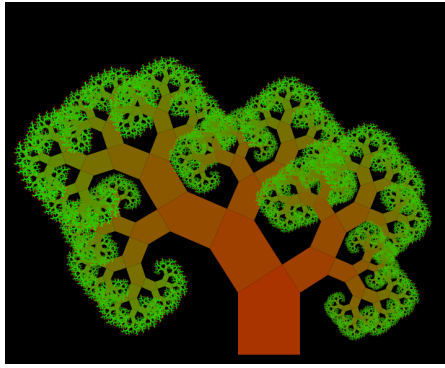
2.2 Learning Haskell

Report by:	Manuel M. T. Chakravarty
Participants:	Gabriele Keller
Status:	Work in progress with eight published chapters

Learning Haskell is a new Haskell tutorial that integrates text and screencasts to combine in-depth explanations with the hands-on experience of live coding. It is aimed at people who are new to Haskell and functional programming. *Learning Haskell* does not assume previous programming expertise, but it is structured such that an experienced programmer who is new to functional programming will also find it engaging.

Learning Haskell combines perfectly with the Haskell for Mac programming environment, but it also includes instructions on working with a conventional command-line Haskell installation. It is a free resource that should benefit anyone who wants to learn Haskell.

Learning Haskell is still work in progress with eight chapters already available. The current material covers all the basics, including higher-order functions and algebraic data types. *Learning Haskell* is approachable and fun – it includes topics such as illustrating various recursive structures using fractal graphics, such as this fractal tree.



Further chapters will be made available as we complete them.

Further reading

- *Learning Haskell* is free at <http://learn.hfm.io>
- Blog post with some background:
<http://blog.haskellformac.com/blog/learning-haskell>

2.3 School of Haskell

Report by:	Michael Snoyman
Participants:	Edward Kmett, Simon Peyton Jones and others
Status:	active

The School of Haskell has been available since early 2013. It's main two functions are to be an education resource for anyone looking to learn Haskell and as a sharing resources for anyone who has built a valuable tutorial. The School of Haskell contains tutorials, courses, and articles created by both the Haskell community and the developers at FP Complete. Courses are available for all levels of developers.

School of Haskell has been open sourced, and is available from its own domain name (schoolofhaskell.com). In addition, the underlying engine powering interactive code snippets, `ide-backend`, has also been released as open source.

Currently 3150 tutorials have been created and 441 have been officially published. Some of the most visited tutorials are *Text Manipulation*, *Attoparsec*, *Learning Haskell at the SOH*, *Introduction to Haskell - Haskell Basics*, and *A Little Lens Starter Tutorial*. Over the past year the School of Haskell has averaged about 16k visitors a month.

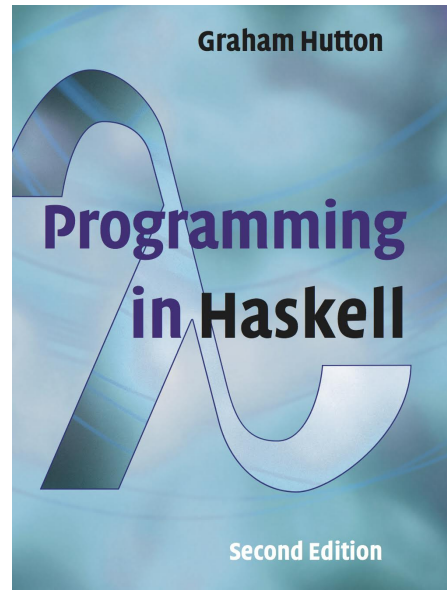
All Haskell programmers are encouraged to visit the School of Haskell and to contribute their ideas and projects. This is another opportunity to showcase the virtues of Haskell and the sophistication and high level thinking of the Haskell community.

Further reading

<https://www.schoolofhaskell.com/>

2.4 Programming in Haskell - 2nd Edition

Report by:	Graham Hutton
Status:	published September 2016



Overview

Haskell is a purely functional language that allows programmers to rapidly develop software that is clear, concise and correct. This book is aimed at a broad spectrum of readers who are interested in learning the language, including professional programmers, university students and high-school students. However, no programming experience is required or assumed, and all concepts are explained from first principles with the aid of carefully chosen examples and exercises. Most of the material in the book should be accessible to anyone over the age of around sixteen with a reasonable aptitude for scientific ideas.

Structure

The book is divided into two parts. Part I introduces the basic concepts of pure programming in Haskell and is structured around the core features of the language, such as types, functions, list comprehensions, recursion and higher-order functions. Part II covers impure programming and a range of more advanced topics, such as monads, parsing, foldable types, lazy evaluation and reasoning about programs. The book contains many extended programming examples, and each chapter includes suggestions for further reading and a series of exercises. The appendices provide solutions to selected exercises, and a summary of some of the most commonly used definitions from the Haskell standard prelude.

What's New

The book is an extensively revised and expanded version of the first edition. It has been extended with new chapters that cover more advanced aspects of Haskell, new examples and exercises to further reinforce the concepts being introduced, and solutions to selected exercises. The remaining material has been completely reworked in response to changes in the language and feedback from readers. The new edition uses the Glasgow Haskell Compiler (GHC), and is fully compatible with the latest version of the language, including recent changes concerning applicative, monadic, foldable and traversable types.

Further reading

<http://www.cs.nott.ac.uk/~pszgmh/pih.html>

2.5 Haskell Programming from first principles, a book for all

Report by:	Chris Allen
Participants:	Julie Moronuki
Status:	Content complete, in final editing

Haskell Programming is a book that aims to get people from the barest basics to being well-grounded in enough intermediate Haskell concepts that they can self-learn what would be typically required to use Haskell in production or to begin investigating the theory and design of Haskell independently. We're writing this book because many have found learning Haskell to be difficult, but it doesn't have to be. What particularly contributes to the good results we've been getting has been an aggressive focus on effective pedagogy and extensive testing with reviewers as well as feedback from readers. My coauthor Julie Moronuki is a linguist who'd never programmed before learning Haskell and authoring the book with me.

Haskell Programming is currently content complete and is approximately 1,200 pages long in the v0.12.0 release. The book is available for sale during the early access, which includes the 1.0 release of the book in PDF. We're still editing the material. We expect to release the final version of the book this winter.

Further reading

- <http://haskellbook.com>
- <https://superginbaby.wordpress.com/2015/05/30/learning-haskell-the-hard-way/>
- <http://bitemyapp.com/posts/2015-08-23-why-we-dont-chuck-readers-into-web-apps.html>

2.6 Stack Builders Tutorials

Report by:	Stack Builders
------------	----------------

At Stack Builders, we consider it our mission not only to develop robust and reliable applications for clients, but to help the industry as a whole by lowering the barrier to entry for technology that we consider important. We hope that you enjoy our tutorials – we're sure you'll find them useful. Any suggestions for future publications, don't hesitate to contact us.

Further reading

- <https://www.stackbuilders.com/tutorials/>
- <https://github.com/stackbuilders/tutorials>

2.7 Haskell MOOC

Report by:	Jeremy Singer
Participants:	Wim Vanderbauwhede
Status:	Second run of a six-week online Haskell course has just completed

The School of Computing Science at the University of Glasgow has partnered with the FutureLearn platform to deliver a six week massive open online course (MOOC) entitled *Functional Programming in Haskell*. The course goes through the basics of the Haskell language, using short videos, an online REPL, multiple choice quizzes and articles.

The second run of the course completed on 31 Oct 2017. Around 2500 people signed up for the course, 70% of whom actively engaged with the materials. Around 500 students completed the full course. The most engaging aspect of the activity was the comradely atmosphere in the discussion forums.

The course will run again, in April 2018. Visit our [site](#) to register your interest.

We hope to refine the learning materials, based on learner feedback from the course. We presented some initial experiences at the Trends in Functional Programming in Education 2017 conference.

Further reading

- <https://www.futurelearn.com/courses/functional-programming-haskell>
- https://www.cs.kent.ac.uk/people/staff/sjt/TFPIE2017/TFPIE_2017/Papers/TFPIE_2017_paper_5.pdf

3 Implementations

3.1 The Glasgow Haskell Compiler

Report by:	Ben Gamari
Participants:	the GHC developers
Status:	GHC continues to improve in performance and stability in 8.4

GHC continues to focus on stability and infrastructure to improve release reliability and predictability.

Major changes in GHC 8.4

GHC 8.4 will continue the focus on stability and performance started in 8.2 and will include a number of internal refactorings in addition to a few smattering of user-facing changes.

Libraries, source language, and type system

- Phase 2 of the [Semigroup-Monoid Proposal](#).
- Further refinement of `TypeInType` including improved error messages.

Compiler

- A new syntax tree representation based on “Trees that Grow”. This will make it easier for external users to add their own annotations to the `HsSyn` AST. In future this should allow Shayan Najd to harmonise the GHC and Template Haskell ASTs, and for the `ghc-exactprint` annotations to move into the GHC parsed AST (Shayan Najd and Alan Zimmerman).
- Further stabilization of the Backpack module system (Edward Yang)
- Greatly improved support for cross-compilation (Moritz Angerman)
- A new build system based on the ‘shake’ library. This is the culmination of nearly two years of effort, replacing GHC’s old Make-based build system (Andrey Mokhov and Zhen Zhang)
- Improved leverage of join-points: By floating out the “exit path” as a join point of a recursive function, we expose more opportunities for inlining to the simplifier. In the case of nested loops, this can make inner loops allocation-free! (see [#14152](#), Joachim Breitner)
- Serialisation performance of `Data.Typeable.TypeRep` has improved in ([#14254](#), David Feuer)
- Improved SIMD support. While previous GHC versions supported a variety of SIMD primitives, they

did not use native SIMD registers for passing arguments across function calls. This severely limited the usefulness of the primitives. Starting with 8.4, GHC will use native registers for passing vector arguments by default.

- Support for the BMI and BMI2 instruction set extensions (John Ky)
- Many, many bug fixes.

Runtime system

- Significantly improved Windows support, with improvements in exception handling, linking, crash diagnostics, GHCi responsiveness and memory allocation and protection (Tamar Christina).

Development updates and acknowledgments

The past six months have seen a great deal of activity in GHC’s infrastructure. This began in the summer on the heels of the 8.2.1 release, with a small group of developers reflecting on the various shortcomings of GHC’s current Phabricator-based continuous integration scheme. With the help of Davean this effort grew from a hypothetical reimagining of GHC’s continuous integration into a functional Jenkins configuration. This effort revealed and addressed numerous inadequacies in GHC’s current test infrastructure. As a result of this work we can now test GHC end-to-end: starting from repository clone, to source distribution tarball, to binary distribution tarball, to completed test suite run. This allows us to find regressions not just in the compiler itself but also in the sizeable mass of infrastructure dedicated to packaging and deploying it.

While Jenkins served as a good testing ground for ideas on improving GHC’s testing methodology, continued work with it revealed a number of issues:

- It lacked support for testing within ‘msys2’ on Windows
- It offered little support in ensuring build purity and reproducibly configuring build environments
- It required a significant investment of effort to setup, followed by an on-going administrative overhead.

Manuel Chakravarty of the newly formed GHC Devops Committee noted these problems and instead proposed that GHC follow the lead of Rust and consider moving to a combination of hosted CI services, Appveyor and CircleCI. With Manuel’s advocacy, the rest of the committee ultimately agreed. As of the time of writing GHC is in the final stages of moving towards this new scheme. Not only will this provide us with more reliable and easier-to-administer CI system, but it will also enable us to broaden the pool of contributors to our testing infrastructure.

GHC's performance testing infrastructure also saw some attention this summer thanks to Haskell Summer of Code student Jared Weakly. Previously, GHC's performance test suite would build a variety of test programs, measuring a variety of run-time and compile-time metrics of each. It would then compare each metric against a supplied acceptance window to identify regressions. While this simple approach served us well for years, it was far from perfect,

- inherent variability between runs and environment dependence meant that the test suite would often incorrectly identify commits as regressing
- to reduce the frequency of false-positives, acceptance window sizes grew over the years meaning that only large regressions would be identified

In his Summer of Code project Jared refactored the test suite to instead simply record the performance metrics resulting from a test suite run. This new visibility into GHC's test suite history will allow GHC developers to precisely identify regressing changes, meanwhile freeing maintainers from the need to periodically bump test suite windows.

The recent work on CI goes hand-in-hand with recent changes in GHC's release scheduling. As of GHC 8.4, GHC will be trying to hold to a six-month periodic release schedule. We hope that this will allow us to get changes into users' hands more quickly and more predictably.

In the compiler itself, Shayan Najd and Alan Zimmerman have been working hard on porting the compiler's frontend AST to use the extension mechanism proposed in Shayan's "Trees That Grow" paper. This is a significant refactoring that will allow GHC API users to extend the AST for their own purposes, significantly improving the reusability of the structure. Eventually this will allow us to split the AST types out of the 'ghc' package, allowing tooling authors, Template Haskell users, and the compiler itself to use the same AST representation.

Joachim Breitner has been working on continuing the join points work started in GHC 8.2 by Luke Maurer. Join points formalize a long-standing technique performed by GHC for eliminating thunk allocation. This formalization has allowed GHC to be better in identifying and preserving join points by directly representing them in Core. Joachim is carrying on this work by teaching the Core simplifier to float out the exit paths of a function, enabling more aggressive inlining.

Thomas Jakway has also been looking at runtime performance, introducing loop annotations in the native code generator. These annotations allow the backend to identify "hot" variables in loops, which the register allocator can use to inform its allocation decisions. Early indications suggest that this work may produce significant speedups in some programs. This work will likely be present in GHC 8.6.

Kavon Farvardin has been working on improving code-generation by the LLVM code generator. For

a long time, GHC has had to work around LLVM's lack of externally visible labels in its intermediate language. This workaround meant that many continuation blocks, which the NCG can optimize as part of a single procedure, must be broken up into multiple LLVM functions, severely limiting the optimization opportunities that LLVM sees. Kavon has been working with LLVM upstream to introduce pseudo-instructions allowing GHC to directly represent procs in LLVM IR.

Kavon has also been looking at improving the default optimization pass configuration used by GHC's LLVM backend. This should both improve compilation time as well as runtime performance.

Peter Trommler, James Clarke, and Karel Gardas have been looking after the PowerPC and SPARC native code generator backends. This work is valuable not only as it improves GHC's portability story, but also because these architectures have memory models which reveal latent bugs more readily than amd64.

Moritz Angerman has been hard at work on a number of areas of the compiler, with a general focus on portability and cross-compilation. Not only has he single-handedly rewritten much of GHC's ARM and AArch64 linker, but he is also adding cross-compilation support to Template Haskell, improving cross-compilation support in the build system, and rewriting the LLVM backend. Thanks Moritz!

As always, if you are interested in contributing to any facet of GHC, be it the runtime system, type-checker, documentation, simplifier, or anything in between, please come speak to us either on IRC (`#ghc` on `irc.freenode.net`) or `ghc-devs@haskell.org`. Happy Haskell!

Further reading

- GHC website: <https://haskell.org/ghc/>
- GHC users guide: https://downloads.haskell.org/~ghc/master/users_guide/
- ghc-devs mailing list: <https://mail.haskell.org/mailman/listinfo/ghc-devs>

3.2 Frege

Report by:	Ingo Wechsung
Participants:	Dierk König, Mark Perry, Marimuthu Madasami, Sean Corfield, Volker Steiss and others
Status:	actively maintained

Frege is a Haskell dialect for the Java Virtual Machine (JVM). It covers essentially Haskell 2010, though there are some mostly insubstantial differences. Several GHC language extensions are supported, most prominently *higher rank types*.

As Frege wants to be a *practical* JVM language, interoperability with existing Java code is essential. To

achieve this, it is not enough to have a foreign function interface as defined by Haskell 2010. We must also have the means to inform the compiler about existing data types (i.e. Java classes and interfaces). We have thus replaced the FFI by a so called *native interface* which is tailored for the purpose.

The compiler, standard library and associated tools like Eclipse IDE plugin, REPL (interpreter) and several build tools are in a usable state, and development is actively ongoing. The compiler is self hosting and has no dependencies except for the JDK.

In the growing, but still small community, a consensus developed last summer that existing differences to Haskell shall be eliminated. Ideally, Haskell source code could be ported by just compiling it with the Frege compiler. Thus, the ultimate goal is for Frege to become *the* Haskell implementation on the JVM.

Already, in the last months, some of the most offending differences have been removed: lambda syntax, instance/class context syntax, recognition of `True` and `False` as boolean literals, lexical syntax for variables and layout-mode issues. Frege now also supports code without module headers.

Frege is available under the BSD-3 license at the GitHub project page. A ready to run JAR file can be downloaded or retrieved through JVM-typical build tools like Maven, Gradle or Leiningen.

All new users and contributors are welcome!

Currently, we have a new version of code generation in alpha status. This will be the base for future interoperability with Java 8 and above.

In April, a community member submitted his masters thesis about implementation of a STM library for Frege.

Further reading

<https://github.com/Frege/frege>

3.3 The Helium Compiler

Report by:	Jurriaan Hage
Participants:	Bastiaan Heeren

Helium is a compiler that supports a substantial subset of Haskell 98 (but, e.g., $n+k$ patterns are missing). Type classes are restricted to a number of built-in type classes and all instances are derived. The advantage of Helium is that it generates novice friendly error feedback, including domain specific type error diagnosis by means of specialized type rules. Helium and its associated packages are available from Hackage. Install it by running `cabal install helium`. You should also `cabal install lvmrun` on which it dynamically depends for running the compiled code.

Currently Helium is at version 1.8.1. The major change with respect to 1.8 is that Helium is again

well-integrated with the Hint programming environment that Arie Middelkoop wrote in Java. The jar-file for Hint can be found on the Helium website, which is located at <http://www.cs.uu.nl/wiki/Helium>. This website also explains in detail what Helium is about, what it offers, and what we plan to do in the near and far future.

A student has added parsing and static checking for type class and instance definitions to the language, but type inferencing and code generating still need to be added. Completing support for type classes is the second thing on our agenda, the first thing being making updates to the documentation of the workings of Helium on the website.

3.4 Specific Platforms

3.4.1 Fedora Haskell SIG

Report by:	Jens Petersen
Participants:	Elliott Sales de Andrade, Robert-Andr� Mauchin
Status:	active

The Fedora Haskell SIG works to provide good Haskell support in the Fedora Project Linux distribution.

For the coming Fedora 27 release we added git-annex back to Fedora, thanks to great packaging efforts by Elliott and Robert-Andr . This also allowed us to enable https in pandoc and http in hakyll. Due to mock changing to default to nospawn chroots, Fedora Copr builds of Haskell packages do not currently work. We use the cabal-rpm packaging tool to create and update Haskell packages, and fedora-haskell-tools to build them.

If you are interested in Fedora Haskell packaging, please join our mailing-list and the Freenode #fedora-haskell channel. You can also follow @fedorahaskell for occasional updates.

Further reading

- o Homepage: http://fedoraproject.org/wiki/Haskell_SIG
- o Mailing-lists: <https://lists.fedoraproject.org/archives/list/haskell@lists.fedoraproject.org/> and <https://lists.fedoraproject.org/archives/list/haskell-devel@lists.fedoraproject.org/>
- o Package list: <https://admin.fedoraproject.org/pkgdb/packager/haskell-sig/>
- o Copr repos: <https://copr.fedorainfracloud.org/coprs/petersen/ghc-8.0.2> and <https://copr.fedorainfracloud.org/coprs/petersen/stack>
- o Fedora Haskell Tools: <https://github.com/fedora-haskell/fedora-haskell-tools>

3.5 Related Languages and Language Design

3.4.2 Debian Haskell Group

Report by: Joachim Breitner
Status: working

The Debian Haskell Group aims to provide an optimal Haskell experience to users of the Debian GNU/Linux distribution and derived distributions such as Ubuntu. We try to follow the Haskell Platform versions for the core packages and package a wide range of other useful libraries and programs. At the time of writing, we maintain 911 source packages.

A system of virtual package names and dependencies, based on the ABI hashes, guarantees that a system upgrade will leave all installed libraries usable. Most libraries are also optionally available with profiling enabled and the documentation packages register with the system-wide index.

The current stable Debian release (“stretch”) provides GHC 8.0.1. In Debian unstable and testing (“buster”, the next release) we ship GHC 8.0.2.

Debian users benefit from the Haskell ecosystem on 22 architecture/kernel combinations, including the non-Linux-ports KFreeBSD and Hurd.

Further reading

<http://wiki.debian.org/Haskell>

3.5.1 Agda

Report by: Ulf Norell
Participants: Ulf Norell, Nils Anders Danielsson, Andreas Abel, Jesper Cockx, Makoto Takeyama, Stevan Andjelkovic, Jean-Philippe Bernardy, James Chapman, Dominique Devriese, Peter Divianszki, Fredrik Nordvall Forsberg, Olle Fredriksson, Daniel Gustafsson, Alan Jeffrey, Fredrik Lindblad, Guilhem Moulin, Nicolas Pouillard, Andrés Sicard-Ramírez and many others
Status: actively developed

Agda is a dependently typed functional programming language (developed using Haskell). A central feature of Agda is inductive families, i.e., GADTs which can be indexed by *values* and not just types. The language also supports coinductive types, parameterized modules, and mixfix operators, and comes with an *interactive* interface—the type checker can assist you in the development of your code.

A lot of work remains in order for Agda to become a full-fledged programming language (good libraries, mature compilers, documentation, etc.), but already in its current state it can provide lots of value as a platform for research and experiments in dependently typed programming.

Some highlights from the past six months:

- o Agda 2.5.3 was released in September 2017.
- o The Agda documentation at <http://agda.readthedocs.org/en/stable/> is being continuously improved.
- o Experimental support for homotopy type theory has been added to the development branch by Andrea Vezzosi.

Further reading

The Agda Wiki: <http://wiki.portal.chalmers.se/agda/>

3.5.2 Disciple

Report by: Ben Lippmeier
Participants: Ben Lippmeier, Jacob Stanley
Status: experimental, active development

The Disciplined Disciple Compiler (DDC) is a research compiler used to investigate program transformation in the presence of computational effects. It compiles a family of strict functional core languages and supports region and effect typing. This extra informa-

tion provides a handle on the operational behaviour of code that isn't available in other languages. Programs can be written in either a pure/functional or effect-ful/imperative style, and one of our goals is to provide both styles coherently in the same language.

What is new?

DDC v0.5.1 was released in late October, and is in "working alpha" state. The main new features are:

- Copying garbage collection using the LLVM shadow stack.
- Implicit parameters, which support Haskell-like ad-hoc overloading using dictionaries.
- Floating point primitives.
- Travis continuous integration for the GitHub site.
- A new Sphinx based user guide and homepage.

We are currently working on a new indexed binary format for interface files, as re-parsing interface files is currently a bottleneck. The file format is to be provided by the Shimmer project, which has been split out into a separate repo.

Further reading

- <http://disciple.ouroborus.net>
- <http://disciple.ouroborus.net/release/ddc-0.5.1.html>
- <https://github.com/DDCSF/shimmer>

4 Libraries, Tools, Applications, Projects

4.1 Language Extensions and Related Projects

4.1.1 Dependent Haskell

Report by:	Richard Eisenberg
Status:	work in progress

I am working on an ambitious update to GHC that will bring full dependent types to the language. In GHC 8, the Core language and type inference have already been updated according to the description in our ICFP'13 paper [1]. Accordingly, *all* type-level constructs are simultaneously kind-level constructs, as there is no distinction between types and kinds. Specifically, GADTs and type families are promotable to kinds. At this point, I conjecture that any construct writable in those other dependently-typed languages will be expressible in Haskell through the use of singletons.

Building on this prior work, I have written my dissertation on incorporating proper dependent types in Haskell [2]. I have yet to have the time to start genuine work on the implementation, but I plan to do so starting summer 2017.

Here is a sneak preview of what will be possible with dependent types, although much more is possible, too!

```
data Vec :: * -> Integer -> * where
  Nil :: Vec a 0
  (:::) :: a -> Vec a n -> Vec a (1 + n)
replicate :: π n. ∀ a. a -> Vec a n
replicate @0 _ = Nil
replicate    x = x ::: replicate x
```

Of course, the design here (especially for the proper dependent types) is preliminary, and input is encouraged.

Further reading

- [1]: *System FC with Explicit Kind Equality*, by Stephanie Weirich, Justin Hsu, and Richard A. Eisenberg. ICFP '13.
<http://www.cis.upenn.edu/~eir/papers/2013/fckinds/fckinds.pdf>
- [2]: *Dependent Types in Haskell: Theory and Practice*, by Richard A. Eisenberg. PhD Thesis, 2015.
<https://github.com/goldfirere/thesis/tree/master/built>

4.1.2 generics-sop

Report by:	Andres Löh
Participants:	Andres Löh, Edsko de Vries

The `generics-sop` (“sop” is for “sum of products”) package is a library for datatype-generic programming in Haskell, in the spirit of GHC’s built-in `DeriveGeneric` construct and the `generic-deriving` package.

Datatypes are represented using a structurally isomorphic representation that can be used to define functions that work automatically for a large class of datatypes (comparisons, traversals, translations, and more). In contrast with the previously existing libraries, `generics-sop` does not use the full power of current GHC type system extensions to model datatypes as an n-ary sum (choice) between the constructors, and the arguments of each constructor as an n-ary product (sequence, i.e., heterogeneous lists). The library comes with several powerful combinators that work on n-ary sums and products, allowing to define generic functions in a very concise and compositional style.

The current release is 0.2.0.0.

A new talk from ZuriHack 2016 is available on Youtube. The most interesting upcoming feature is probably type-level metadata, making use of the fact that GHC 8 now offers type-level metadata for the built-in generics. While the feature is in principle implemented, there are still a few open questions about what representation would be most convenient to work with in practice. Help or opinions are welcome!

Further reading

- `generics-sop` package:
<https://hackage.haskell.org/package/generics-sop/>
- Tutorial (summer school lecture notes):
<https://github.com/kosmikus/SSGEP/>
- ZuriHac 2016 talk:
<https://www.youtube.com/watch?v=sQxH349HOik>
- WGP 2014 talk:
<https://www.youtube.com/watch?v=jzgfM6NFE3Y>
- Paper:
<http://www.andres-loeh.de/TrueSumsOfProducts/>

4.1.3 Supermonads

Report by:	Jan Bracker
Participants:	Jan Bracker and Henrik Nilsson
Status:	Experimental fully working version

The supermonad package provides a unified way to represent different monadic and applicative notions. In other words, it provides a way to use standard and generalized monads and applicative functors (with additional indices or constraints) without having to manually disambiguate which notion is referred to in every context. This allows the reuse of code, such as standard library functions, across all of the notions.

To achieve this, the library splits the monad and applicative type classes such that they are general enough to allow instances for all of the generalized notions and then aids constraint checking through a GHC plugin to ensure that everything type checks properly. Due to the plugin the library can only be used with GHC.

If you are interested in using the library, we have a few examples of different size in the repository to show how it can be utilized. The generated Haddock documentation also has full coverage and can be seen on the libraries Hackage page.

The project had its first release shortly before ICFP and the Haskell Symposium 2016. Since then we have added support for applicative functors in addition to monads. The support for applicative functors has not been released on Hackage at the time of writing, but is available on GitHub.

We are working on a comprehensive paper that covers all aspects of the project and its theoretical foundations. It should be available within the next year.

If you are interested in contributing, found a bug or have a suggestion to improve the project we are happy to hear from you in person, by email or over the projects bug tracker on GitHub.

Further reading

- Hackage:
<http://hackage.haskell.org/package/supermonad>
- Repository:
<https://github.com/jbracker/supermonad>
- Paper:
<http://www.cs.nott.ac.uk/~psxjb5/publications/2016-BrackerNilsson-Supermonads.pdf>
- Bug-Tracker:
<https://github.com/jbracker/supermonad/issues>
- Haskell Symposium presentation:
<https://youtu.be/HRofw58sySw>

4.1.4 Reifying type families

Report by:	Gabor Greif
Status:	experimental, comments welcome

The outcome of the compile-time evaluation of type families is currently inscrutable to the running program. `tyfam-witnesses` is a new minimal library that utilises Template Haskell to obtain the necessary artifacts for running the clauses of *closed* type families at execution time. By pattern matching on the outcome all the type-level equalities can be recovered.

For each closed type family in a series of declarations, `witnesses` adds a GADT mirroring its clauses, and a reification function that *runs* it given indexed `TypeReps`. Here is a usage example:

```
type family Elim v f where
  Elim v (v -> c) = c
  Elim v (d -> c) = d -> Elim v c
}
```

gets accompanied with

```
data ElimRefl v f where
  Elim0 :: Elim v (v -> d) ~ d
        => ElimRefl v (v -> d)
  Elim1 :: Elim v (c -> d) ~ (c -> Elim a b)
        => ElimRefl v (c -> d)
```

and a *runner* (or *reifier*)

```
reify_Elim :: TypeRep a -> TypeRep b
            => Maybe (ElimRefl a b)
```

Pattern matching on the result of the latter guides the GHC type checker and allows writing recursive functions that evaluate to an `ElimRefl v f`, which would otherwise get stuck.

The library has been introduced at the Regensburg Haskell Meetup (\rightarrow 6.8) and other conferences in Oct. 2017.

You can find it on hackage, grab it with `cabal install tyfam-witnesses`, be reminded however, that GHC v8.2 is a prerequisite for its usage.

I am interested in possible further uses and am waiting for encouragement in resolving the two remaining restrictions.

Further reading

- <https://hackage.haskell.org/package/tyfam-witnesses>
- <https://skillsmatter.com/skillscasts/10947-lightning-talk-engage-clutch-shift-gear-rofl>

4.2 Build Tools and Related Projects

4.2.1 Cabal

Report by:	Mikhail Glushenkov
Status:	Stable, actively developed

Background

Cabal is the standard packaging system for Haskell software. It specifies a standard way in which Haskell libraries and applications can be packaged so that it is easy for consumers to use them, or re-package them, regardless of the Haskell implementation or installation platform.

`cabal-install` is the command line interface for the Cabal and Hackage system. It provides a command line program `cabal` which has sub-commands for installing and managing Haskell packages.

Recent Progress

We've recently produced new point releases of Cabal/`cabal-install` from the 1.24 branch. Among other things, Cabal 1.24.2.0 includes a fix necessary to make soon-to-be-released GHC 8.0.2 work on macOS Sierra.

Almost 1500 commits were made to the `master` branch by 53 different contributors since the 1.24 release. Among the highlights are:

- Convenience, or `internal libraries` – named libraries that are only intended for use inside the package. A common use case is sharing code between the test suite and the benchmark suite without exposing it to the users of the package.
- Support for `foreign libraries`, which are Haskell libraries intended to be used by foreign languages like C. Foreign libraries only work with GHC 7.8 and later.
- Initial support for building Backpack packages. Backpack is an exciting new project adding an ML-style module system to Haskell, but on the package level. See [here](#) and [here](#) for a more thorough introduction to Backpack.
- `./Setup configure` now accepts an argument specifying the component to be configured. This is mainly an internal change, but it means that `cabal-install` can now perform component-level parallel builds (among other things).
- A lot of improvements in the `new-build` feature (a.k.a. nix-style local builds). Git HEAD version of `cabal-install` is now recommended if you use `new-build`. For an introduction to `new-build`, see [this chapter](#) of the manual.
- Special support for the Nix package manager in `cabal-install`. See [here](#) for more details.
- `cabal upload` now uploads a package candidate by default. Use `cabal upload --publish` to upload a final version. `cabal upload --check` has been removed in favour of package candidates.

- An `--index-state` flag for requesting a specific version of the package index.
- New `cabal reconfigure` command, which re-runs `configure` with most recently used flags.
- New `autogen-modules` field for modules built automatically (like `Paths_PACKAGE_NAME`).
- New version range operator `^>=`, which is equivalent to `>=` intersected with an automatically-inferred major version bound. For example, `^>= 2.0.3` is equivalent to `>= 2.0.3 && < 2.1`.
- An `--allow-older` flag, dual to `--allow-newer`.
- New Parsec-based parser for `.cabal` files has been merged, but not enabled by default yet.
- The manual has been converted to reST/Sphinx format, improved and expanded.
- Hackage Security has been enabled by default.
- A lot of bug fixes and performance improvements.

Looking Forward

The next Cabal/`cabal-install` versions will be released either in early 2017, or simultaneously with GHC 8.2 (April/May 2017). Our main focus at this stage is getting the `new-build` feature to the state where it can be enabled by default, but there are many other areas of Cabal that need work.

We would like to encourage people considering contributing to take a look at the [bug tracker on GitHub](#) and the [Wiki](#), take part in discussions on tickets and pull requests, or submit their own. The bug tracker is reasonably well maintained and it should be relatively clear to new contributors what is in need of attention and which tasks are considered relatively easy. For more in-depth discussion there is also the `cabal-devel` mailing list.

Further reading

- Cabal homepage: <https://www.haskell.org/cabal/>
- Cabal on GitHub: <https://github.com/haskell/cabal>

4.2.2 The Stack build tool

Report by:	Emanuel Borsboom
Status:	stable

Stack is a modern, cross-platform build tool for Haskell code. It is intended for Haskellers both new and experienced.

Stack handles the management of your toolchain (including GHC - the Glasgow Haskell Compiler - and, for Windows users, MSYS), building and registering libraries, building build tool dependencies, and more. While it can use existing tools on your system, Stack has the capacity to be your one-stop shop for all Haskell tooling you need.

The primary design point is reproducible builds. If you run `stack build` today, you should get the same result running `stack build` tomorrow. There are some

cases that can break that rule (changes in your operating system configuration, for example), but, overall, Stack follows this design philosophy closely. To make this a simple process, Stack uses curated package sets called snapshots.

Stack has also been designed from the ground up to be user friendly, with an intuitive, discoverable command line interface.

Since its first release in June 2015, many people are using it as their primary Haskell build tool, both commercially and as hobbyists. New features and refinements are continually being added, with regular new releases.

Binaries and installers/packages are available for common operating systems to make it easy to get started. Download it at <http://haskellstack.org/>.

Further reading

<http://haskellstack.org/>

4.2.3 Stackage: the Library Dependency Solution

Report by:	Michael Snoyman
Status:	new

Stackage began in November 2012 with the mission of making it possible to build stable, vetted sets of packages. The overall goal was to make the Cabal experience better. Five years into the project, a lot of progress has been made and now it includes both Stackage and the Stackage Server. To date, there are over 1900 packages available in Stackage. The official site is <https://www.stackage.org>.

The Stackage project consists of many different components, linked to from the Stackage Github repository <https://github.com/fpco/stackage#readme>. These include:

- Stackage Nightly, a daily build of the Stackage package set
- LTS Haskell, which provides major-version compatibility for a package set over a longer period of time
- Stackage Server, which runs on stackage.org and provides browsable docs, reverse dependencies, and other metadata on packages
- Stackage Curator, a tool for running the various builds

The Stackage package set has first-class support in the Stack build tool (→ 4.2.2). There is also support for cabal-install via cabal.config files, e.g. <https://www.stackage.org/lts/cabal.config>.

There are dozens of individual maintainers for packages in Stackage. Overall Stackage curation is handled by the “Stackage curator” team, which consists of Michael Snoyman, Adam Bergmark, Dan Burton, Jens Petersen, Luke Murphy, Chris Dornan, and Mihai Maruseac.

Stackage provides a well-tested set of packages for end users to develop on, a rigorous continuous-integration system for the package ecosystem, some basic guidelines to package authors on minimal package compatibility, and even a testing ground for new versions of GHC. Stackage has helped encourage package authors to keep compatibility with a wider range of dependencies as well, benefiting not just Stackage users, but Haskell developers in general.

If you’ve written some code that you’re actively maintaining, don’t hesitate to get it in Stackage. You’ll be widening the potential audience of users for your code by getting your package into Stackage, and you’ll get some helpful feedback from the automated builds so that users can more reliably build your code.

Since the last HCAR, we have moved Stackage Nightly to GHC 8.2.1, as well as released LTS 9 based on GHC 8.0.2. We continue to make releases of LTS 6, 7 and 8 concurrently, which are based on GHC 7.10.3, 8.0.1 and 8.0.2, respectively.

4.2.4 Stackgo

Report by:	Sibi Prabakaran
Status:	active

A browser plugin (currently supported for Firefox/Google Chrome) to automatically redirect Haddock documentation on Hackage to corresponding Stackage pages, when the request is via search engines like Google/Bing etc. For the case where the package hasn’t been added yet to Stackage, no redirect will be made and the Hackage documentation will be available. This plugin also tries to guess when the user would want to go to a Hackage page instead of the Stackage one and tries to do the right thing there.

Compared to the previous version, stackgo now has the ability to redirect to a specific resolver in Stackage. You can set a custom resolver from the Settings page. By default, it will always redirect to the latest lts resolver.

Further reading

- <https://github.com/psibi/stackgo>
- <https://addons.mozilla.org/en-US/firefox/addon/stackgo>
- <https://chrome.google.com/webstore/detail/ojjalokgookadeklnfjglbnlbaiackn>

4.2.5 hinstall

Report by:	Dino Morelli
Status:	stable, actively developed

This is a utility to install Haskell programs on a system using stack. Although stack does have an `install` command, it only copies binaries. Sometimes more is needed, other files and some directory structure. `hinstall` tries to install the binaries, the LICENSE file and also the resources directory if it finds one.

Installations can be performed in one of two directory structures. FHS, or the Filesystem Hierarchy Standard (most UNIX-like systems) and what I call “bundle” which is a portable directory for the app and all of its files. They look like this:

- o bundle is sort-of a self-contained structure like this:

```
$PREFIX/  
$PROJECT-$VERSION/  
  bin/...  
  doc/LICENSE  
  resources/...
```

- o fhs is the more traditional UNIX structure like this:

```
$PREFIX/  
  bin/...  
  share/  
  $PROJECT-$VERSION/  
  doc/LICENSE  
  resources/...
```

There are two parts to `hinstall` that are intended to work together. The first part is a Haskell shell script, `util/install.hs`. Take a copy of this script and check it into a project you’re working on. This will be your installation script. Running the script with the `-help` switch will explain the options. Near the top of the script are default values for these options that should be tuned to what your project needs.

The other part of `hinstall` is a library. The install script will try to install a `resources` directory if it finds one. the `HSInstall` library can then be used in your code to locate the resources at runtime.

Note that you only need the library if your software has data files it needs to locate at runtime in the installation directories. Many programs don’t have this requirement and can ignore the library altogether.

Source code is available on darcsHub, Hackage and Stackage

Further reading

- o `hinstall` on darcsHub
<http://hub.darcs.net/dino/hinstall>
- o `hinstall` on Hackage
<https://hackage.haskell.org/package/hinstall>
- o `hinstall` on Stackage
<https://www.stackage.org/package/hinstall>

4.2.6 yesod-rest

Report by:	Sibi Prabhakaran
Status:	active

A Yesod scaffolding site with Postgres backend. It provides a JSON API backend as a separate subsite. The primary purpose of this repository is to use Yesod as a API server backend and do the frontend development using a tool like React or Angular. The current code includes a basic example using React and Babel which is bundled finally by webpack and added in the handler `getHomeR` in a type safe manner.

The future work is to integrate it as part of `yesod-scaffold` and make it as part of `stack template`.

Further reading

- o <https://github.com/psibi/yesod-rest>
- o <https://github.com/yesodweb/yesod-scaffold/issues/136>

4.2.7 Haskell Cloud

Report by:	Gideon Sireling
------------	-----------------

Haskell Cloud is a `Source-to-Image` builder for building Haskell source into a runnable Docker image. It can be used directly with `s2i`, or deployed on `OpenShift`.

Using the Haskell Cloud builder, existing Haskell projects can be uploaded, built, and run from the cloud with minimal changes. A choice of pre-installed frameworks is available - see the `Wiki` for details.

Further reading

- o <https://bitbucket.org/accusoft/haskell-cloud>

4.2.8 Hackagebot 2.0

Report by:	Chris Wong
------------	------------

Hackagebot 2.0 is an IRC bot that announces new Hackage uploads to the `#haskell` channel. It is a complete rewrite of the original Hackagebot, and improves on many aspects of stability and correctness.

In particular:

- o It combines multiple announcements into a single message, instead of sending them individually;
- o It does not leak memory;
- o It has full Unicode support, and does not scramble non-ASCII text.

The code is hosted on GitHub.

Further reading

<https://github.com/lfairy/hircine/tree/master/bots>

4.3 Repository Management

4.3.1 Darcs

Report by:	Guillaume Hoffmann
Participants:	darcs-users list
Status:	active development

Darcs is a distributed revision control system written in Haskell. In Darcs, every copy of your source code is a full repository, which allows for full operation in a disconnected environment, and also allows anyone with read access to a Darcs repository to easily create their own branch and modify it with the full power of Darcs' revision control. Darcs is based on an underlying theory of patches, which allows for safe reordering and merging of patches even in complex scenarios. For all its power, Darcs remains a very easy to use tool for every day use because it follows the principle of keeping simple things simple.

As the release of the next major version is getting closer, the codebase of Darcs is undercoming many refactorings. Visible changes include a better support of encodings, improved display of patch dependencies, per-file conflict marking, a more efficient annotate command, and improvements in shell completion.

SFC and donations Darcs is free software licensed under the GNU GPL (version 2 or greater). Darcs is a proud member of the Software Freedom Conservancy, a US tax-exempt 501(c)(3) organization. We accept donations at <http://darcs.net/donations.html>.

Further reading

- o <http://darcs.net>
- o <http://darcs.net/Releases/2.12>
- o <http://hub.darcs.net>

4.3.2 git-annex

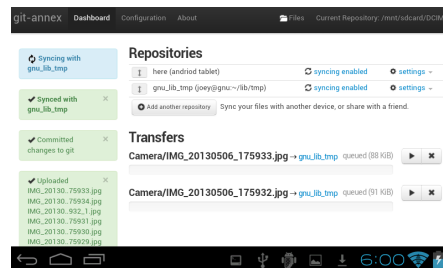
Report by:	Joey Hess
Status:	stable, actively developed

git-annex allows managing files with git, without checking the file contents into git. While that may seem paradoxical, it is useful when dealing with files larger than git can currently easily handle, whether due to limitations in memory, time, or disk space.

As well as integrating with the git command-line tools, git-annex includes a graphical app which can be used to keep a folder synchronized between computers. This is implemented as a local webapp using yesod and warp.

git-annex runs on Linux, OSX and other Unixes, and has been ported to Windows. There is also an incomplete but somewhat usable port to Android.

Five years into its development, git-annex has a wide user community. It is being used by organizations for purposes as varied as keeping remote Brazilian communities in touch and managing Neurological imaging data. It is available in a number of Linux distributions, in OSX Homebrew, and is one of the most downloaded utilities on Hackage. It was my first Haskell program.



At this point, my goals for git-annex are to continue to improve its foundations, while at the same time keeping up with the constant flood of suggestions from its user community, which range from adding support for storing files on more cloud storage platforms (around 20 are already supported), to improving its usability for new and non technically inclined users, to scaling better to support Big Data, to improving its support for creating metadata driven views of files in a git repository.

At some point I'd also like to split off any one of a half-dozen general-purpose Haskell libraries that have grown up inside the git-annex source tree.

Further reading

<http://git-annex.branchable.com/>

4.3.3 Octohat

Report by:	Stack Builders
Participants:	Juan Carlos Paucar, Sebastian Estrella, Juan Pablo Santos
Status:	Working, well-tested minimal wrapper around GitHub's API

Octohat is a comprehensively test-covered Haskell library that wraps GitHub's API. While we have used it successfully in an open-source project to automate granting access control to servers, it is in very early development, and it only covers a small portion of GitHub's API.

Octohat is available on Hackage, and the source code can be found on GitHub.

We have already received some contributions from the community for Octohat, and we are looking forward to more contributions in the future.

Further reading

- o <https://github.com/stackbuilders/octohat>
- o [Octohat announcement](#)
- o [Octohat update](#)

4.3.4 openssh-github-keys (Stack Builders)

Report by:	Stack Builders
Participants:	Justin Leitgeb
Status:	active

It is common to control access to a Linux server by changing public keys listed in the `authorized_keys` file. Instead of modifying this file to grant and revoke access, a relatively new feature of OpenSSH allows the accepted public keys to be pulled from standard output of a command.

This package acts as a bridge between the OpenSSH daemon and GitHub so that you can manage access to servers by simply changing a GitHub Team, instead of manually modifying the `authorized_keys` file. This package uses the `Octohat` wrapper library for the GitHub API which we released.

`openssh-github-keys` is still experimental, but we are using it on a couple of internal servers for testing purposes. It is available on [Hackage](#) and contributions and bug reports are welcome in the [GitHub repository](#).

While we don't have immediate plans to put `openssh-github-keys` into heavier production use, we are interested in seeing if community members and system administrators find it useful for managing server access.

Further reading

<https://github.com/stackbuilders/openssh-github-keys>

4.4 Debugging and Profiling

4.4.1 Hoed – The Lightweight Algorithmic Debugger for Haskell

Report by:	Maarten Faddegon
Status:	active

Hoed is a lightweight algorithmic debugger that is practical to use for real-world programs because it works with any Haskell run-time system and does not require trusted libraries to be transformed.

To locate a defect with Hoed you annotate suspected functions and compile as usual. Then you run your program, information about the annotated functions is collected. Finally you connect to a debugging session using a webbrowser.

Using Hoed

Let us consider the following program, a defective implementation of a parity function with a test property.

```
isOdd :: Int -> Bool
isOdd n = isEven (plusOne n)

isEven :: Int -> Bool
isEven n = mod2 n == 0
```

```
plusOne :: Int -> Int
plusOne n = n + 1

mod2 :: Int -> Int
mod2 n = div n 2

prop_isOdd :: Int -> Bool
prop_isOdd x = isOdd (2*x+1)

main :: IO ()
main = print0 (prop_isOdd 1)

main :: IO ()
main = quickcheck prop_isOdd
```

Using the property-based test tool QuickCheck we find the counter example 1 for our property.

```
./MyProgram
*** Failed! Falsifiable (after 1 test): 1
```

Hoed can help us determine which function is defective. We annotate the functions `isOdd`, `isEven`, `plusOne` and `mod2` as follows:

```
import Debug.Hoed.Pure

isOdd :: Int -> Bool
isOdd = observe "isOdd" isOdd'
isOdd' n = isEven (plusOne n)

isEven :: Int -> Bool
isEven = observe "isEven" isEven'
isEven' n = mod2 n == 0

plusOne :: Int -> Int
plusOne = observe "plusOne" plusOne'
plusOne' n = n + 1

mod2 :: Int -> Int
mod2 = observe "mod2" mod2'
mod2' n = div n 2

prop_isOdd :: Int -> Bool
prop_isOdd x = isOdd (2*x+1)

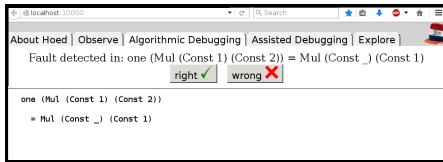
main :: IO ()
main = print0 (prop_isOdd 1)
```

And run our program:

```
./MyProgram
False
Listening on http://127.0.0.1:10000/
```

Now you can use your webbrowser to interact with Hoed.

There is a classic algorithmic debugging interface in which you are shown computation statements, these are function applications and their result, and are asked to judge if these are correct. After judging enough computation statements the algorithmic debugger tells you where the defect is in your code.



In the explore mode, you can also freely browse the tree of computation statements to get a better understanding of your program. The observe mode is inspired by HOOD and gives a list of computation statements. Using regular expressions this list can be searched. Algorithmic debugging normally starts at the top of the tree, e.g. the application of `isOdd` to $(2*x+1)$ in the program above, using explore or observe mode a different starting point can be chosen.

To reduce the number of questions the programmer has to answer, we added a new mode Assisted Algorithmic Debugging in version 0.3.5 of Hoed. In this mode (QuickCheck) properties already present in program code for property-based testing can be used to automatically judge computation statements

Further reading

- <http://wiki.haskell.org/Hoed>
- <http://hackage.haskell.org/package/Hoed>

4.4.2 ghc-heap-view

Report by:	Joachim Breitner
Participants:	Dennis Felsing
Status:	active development

The library `ghc-heap-view` provides means to inspect the GHC's heap and analyze the actual layout of Haskell objects in memory. This allows you to investigate memory consumption, sharing and lazy evaluation.

This means that the actual layout of Haskell objects in memory can be analyzed. You can investigate sharing as well as lazy evaluation using `ghc-heap-view`.

The package also provides the GHCi command `:printHeap`, which is similar to the debuggers' `:print` command but is able to show more closures and their sharing behaviour:

```
> let x = cycle [True, False]
> :printHeap x
_bco
> head x
True
> :printHeap x
let x1 = True : _think x1 [False]
in x1
> take 3 x
[True,False,True]
> :printHeap x
```

```
let x1 = True : False : x1
in x1
```

The graphical tool `ghc-vis` (\rightarrow 4.4.3) builds on `ghc-heap-view`.

Since version 0.5.10, `ghc-heap-view` supports GHC 8.2.

Further reading

- <http://www.joachim-breitner.de/blog/archives/548-ghc-heap-view-Complete-referential-opacity.html>
- <http://www.joachim-breitner.de/blog/archives/580-GHCi-integration-for-GHCi-HeapView.html>
- <http://www.joachim-breitner.de/blog/archives/590-Evaluation-State-Assertions-in-Haskell.html>

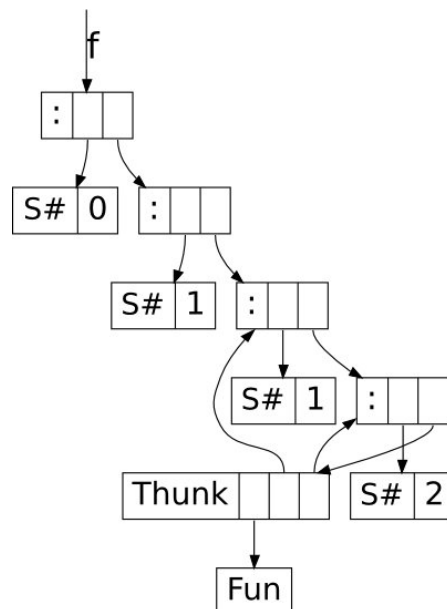
4.4.3 ghc-vis

Report by:	Joachim Breitner
Status:	active development

The tool `ghc-vis` visualizes live Haskell data structures in GHCi. Since it does not force the evaluation of the values under inspection it is possible to see Haskell's lazy evaluation and sharing in action while you interact with the data.

`Ghc-vis` supports two styles: A linear rendering similar to GHCi's `:print`, and a graph-based view where closures in memory are nodes and pointers between them are edges. In the following GHCi session a partially evaluated list of fibonacci numbers is visualized:

```
> let f = 0 : 1 : zipWith (+) f (tail f)
> f !! 2
> :view f
```



At this point the visualization can be used interactively: To evaluate a thunk, simply click on it and immediately see the effects. You can even evaluate thunks

which are normally not reachable by regular Haskell code.

Ghc-vis can also be used as a library and in combination with GHCi's debugger.

Further reading

<http://felsin9.de/nnis/ghc-vis>

4.4.4 Hat — the Haskell Tracer

Report by: Olaf Chitil

Hat is a source-level tracer for Haskell. Hat gives access to detailed, otherwise invisible information about a computation.

Hat helps locating errors in programs. Furthermore, it is useful for understanding how a (correct) program works, especially for teaching and program maintenance. Hat is not a time or space profiler. Hat can be used for programs that terminate normally, that terminate with an error message or that terminate when interrupted by the programmer.

You trace a program with Hat by following these steps:

1. With *hat-trans* translate all the source modules of your Haskell program into tracing versions. Compile and link (including the Hat library) these tracing versions with *ghc* as normal.
2. Run the program. It does exactly the same as the original program except for additionally writing a trace to file.
3. After the program has terminated, view the trace with a tool. Hat comes with several tools for selectively viewing fragments of the trace in different ways: *hat-observe* for Hood-like observations, *hat-trail* for exploring a computation backwards, *hat-explore* for freely stepping through a computation, *hat-detect* for algorithmic debugging, ...

Hat is distributed as a package on Hackage that contains all Hat tools and tracing versions of standard libraries. Hat 2.9.4 works with recent versions of the Glasgow Haskell compiler for Haskell programs that are written in Haskell 98 plus a few language extensions such as multi-parameter type classes and functional dependencies.

Although Hat is distributed as a cabal package that can be installed with *stack*, it currently does not support working with *stack* projects; instead it provides an old-fashioned build tool *hat-make*.

Note that all modules of a traced program have to be transformed, including trusted libraries (transformed in trusted mode). For portability all viewing tools have a textual interface; however, many tools require

an ANSI terminal and thus run on Unix / Linux / OS X, but not on Windows.

In the longer term we intend to transfer the lightweight tracing technology that we use in Hoed (→ 4.4.1) also to Hat.

Further reading

- o Initial website: <http://projects.haskell.org/hat>
- o Hackage package: <http://hackage.haskell.org/package/hat>

4.4.5 LeanCheck

Report by: Rudy Braquehais
Participants: Colin Runciman
Status: Actively maintained, v0.6.7

LeanCheck is an enumerative property-based testing library with a very small core of only 180 lines of code. Its enumeration is size-bounded so the number of tests is easier to control than with SmallCheck. LeanCheck is somewhat stable and has been around for a while, but this is its first announcement on HCAR.

It is used like so:

```
> import Test.LeanCheck
> check $ \x y -> x + y == y + (x :: Int)
+++ OK, passed 200 tests.
> check $ \x y -> x - y == y - (x :: Int)
*** Failed! Falsifiable (after 2 tests):
0 1
```

LeanCheck has support for higher-order properties (those taking functions as arguments). For example:

```
> import Test.LeanCheck.Function
> check $ \p q xs -> filter p (filter q xs)
>                               == filter q (filter p xs)
>                               :: [Int]
+++ OK, passed 200 tests.
> check $ \f p xs -> map f (filter p xs)
>                               == filter p (map f xs)
>                               :: [Bool]
*** Failed! Falsifiable (after 20 tests):
\x -> case x of False -> False; True -> False
\x -> case x of False -> False; True -> True
[True]
```

The function `filter` commutes with itself, but not with `map`.

LeanCheck works on properties whose argument types are instances of the `Listable` typeclass. It is very easy to define `Listable` instances for user-defined types. For example, take “Hutton’s Razor”:

```
data Expr = Val Int | Add Expr Expr
  deriving (Show, Eq)
```

Its `Listable` instance can be given by

```
instance Listable Expr where
  tiers = cons1 Val \/ cons2 Add
```

or automatically derived using Template Haskell by

```
deriveListable ''Expr
```

LeanCheck is available on Hackage under a BSD3-style license. All you need to do to get it is:

```
$ cabal install leancheck
```

Further reading

- <https://hackage.haskell.org/package/leancheck>
- <https://github.com/rudymatela/leancheck>

4.4.6 Speculate

Report by:	Rudy Braquehais
Participants:	Colin Runciman
Status:	Actively maintained, v0.2.9

Speculate is a library that uses testing to automatically discover and conjecture properties about Haskell functions. Those properties can contribute to understanding, documentation, validation, design refinement and regression testing.

A quick example, discovering properties about addition and multiplication:

```
> import Test.Speculate
> speculate args
> { constants = [ constant "+" (+)
>                , constant "*" (*) ] }
    x + y == y + x
    x * y == y * x
(x + y) + z == x + (y + z)
(x * y) * z == x * (y * z)
(x + x) * y == x * (y + y)

x <= x * x
```

Speculate is similar to QuickSpec, but uses a different algorithm to produce *inequalities* and *conditional equations*. See the documentation for further details and examples.

Speculate is available on Hackage under a BSD3-style license. All you need to do to get it is:

```
$ cabal install speculate
```

Further reading

- <https://hackage.haskell.org/package/speculate>
- <https://github.com/rudymatela/speculate>
- The Haskell Symposium 2017 paper about Speculate:
<https://matela.com.br/paper/speculate.pdf>

4.4.7 Extrapolate

Report by:	Rudy Braquehais
Participants:	Colin Runciman
Status:	Actively maintained, v0.2.4

Extrapolate is a property-based testing library capable of reporting generalized counter-examples to properties. Extrapolate works on top of LeanCheck ([→ 4.4.5](#)).

Here is an example:

```
> import Test.Extrapolate
> import Data.List (nub)
> check $ \xs -> nub xs == (xs :: [Int])
*** Failed! Falsifiable (after 3 tests):
[0,0]
```

Generalization:

```
x:x:_
```

Conditional Generalization:

```
x:xs when elem x xs
```

The above property about `nub` not only fails for the list `[0,0]` but also for any list that has repeated elements.

The generalization of failing cases informs the programmer more fully and more immediately what characterizes failures. This information helps the programmer to locate more confidently and more rapidly the causes of failure in their program.

Extrapolate's generalization of counter-examples is similar to SmartCheck's. However, when generalizing, Extrapolate allows for repeated variables and side-conditions.

Extrapolate is available on Hackage under a BSD3-style license. All you need to do to get it is:

```
$ cabal install extrapolate
```

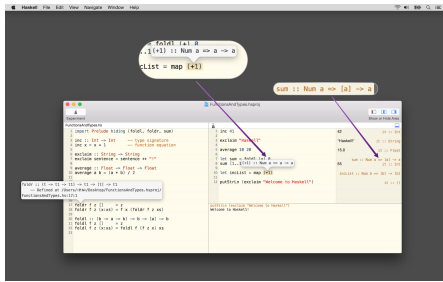
Further reading

- <https://hackage.haskell.org/package/extrapolate>
- <https://github.com/rudymatela/extrapolate>
- Paper about Extrapolate published in the IFL 2017 draft proceedings:
<https://matela.com.br/paper/extrapolate.pdf>

4.5 Development Tools and Editors

4.5.1 Haskell for Mac

Report by: Manuel M. T. Chakravarty
Status: Available & actively developed



Haskell for Mac is an easy-to-use, innovative programming environment and learning platform for Haskell on OS X. It includes its own Haskell distribution and requires no further set up. It features interactive Haskell playgrounds to explore and experiment with code. Playground code is not only type-checked, but also executed while you type, which leads to a fast turn around during debugging or experimenting with new code.

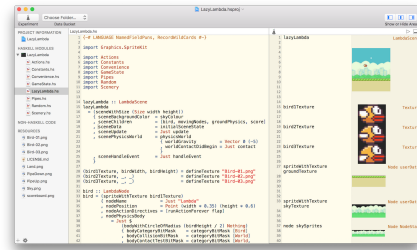
Integrated environment. Haskell for Mac integrates everything needed to start writing Haskell code, including an editor with syntax highlighting and smart identifier completion. Haskell for Mac creates Haskell projects based on standard Cabal specifications for compatibility with the rest of the Haskell ecosystem. It includes the Glasgow Haskell Compiler (GHC) and over 200 of the most popular packages of LTS Haskell package sets. Matching command line tools and extra packages can be installed, too.

Type directed development. Haskell for Mac uses GHC's support for deferred type errors so that you can still execute playground code in the face of type errors. This is convenient during refactoring to test changes, while some code still hasn't been adapted to new signatures. Moreover, you can use type holes to stub out missing pieces of code, while still being able to run code. The system will also report the types expected for holes and the types of the available bindings.

Interactive HTML, graphics & games. Haskell for Mac comes with support for web programming, network programming, graphics programming, animations, and much more. Interactively generate web pages, charts, animations, or even games (with the OS X SpriteKit support). Graphics are also live and change as you modify the program code.

The screenshot below is from the development of a Flappy Bird clone in Haskell. Watch the Haskell for Mac developer live code Flappy Bird in Haskell in 20min at the end of the Compose :: Melbourne 2016 keynote at <https://speakerdeck.com/mchakravarty/playing-with-graphics-and-animations-in-haskell>. You can find more information about writing games in

Haskell in this blog post: <http://blog.haskellformac.com/blog/writing-games-in-haskell-with-spritekit>.



Haskell for Mac has recently gained auto-completion of identifiers, taking into account the current module's imports. It now also features a graphical package installer for LTS Haskell and support for GHC 8. Moreover, a new type class, `Presentable`, enables custom rendering of user-defined data types using images, HTML, and even animations.

Haskell for Mac is available for purchase from the Mac App Store. Just search for "Haskell", or visit our website for a direct link. We are always available for questions or feedback at support@haskellformac.com.

Further reading

The Haskell for Mac website: <http://haskellformac.com>

4.5.2 `haskell-ide-engine`, a project for unifying IDE functionality

Report by: Chris Allen
Participants: Alan Zimmerman, Moritz Kiefer, Michael Sloan, Gracjan Polak, Daniel Gröber, others welcome
Status: Open source, just beginning

`haskell-ide-engine` is a backend for driving the sort of features programmers expect out of IDE environments. `haskell-ide-engine` is a project to unify tooling efforts into something different text editors, and indeed IDEs as well, could use to avoid duplication of effort.

There is basic support for getting type information and refactoring, more features including type errors, linting and reformatting are planned. People who are familiar with a particular part of the chain can focus their efforts there, knowing that the other parts will be handled by other components of the backend. Integration for Emacs and Leksah is available and should support the current features of the backend. Work has started on a Language Server Protocol transport, for use in VS Code. `haskell-ide-engine` also has a REST API with Swagger UI. Inspiration is being taken from the work the Idris community has done toward an interactive editing environment as well.

Help is very much needed and wanted so if this is a problem that interests you, please pitch in! This is not a project just for a small inner circle. Anyone who

wants to will be added to the project on github, address your request to @alanz.

Further reading

- o <https://github.com/haskell/haskell-ide-engine>
- o <https://github.com/Microsoft/language-server-protocol>
- o <https://mail.haskell.org/pipermail/haskell-cafe/2015-October/121875.html>
- o <https://www.fpcomplete.com/blog/2015/10/new-haskell-ide-repo>
- o https://www.reddit.com/r/haskell/comments/3pt560/ann_haskellide_project/
- o https://www.reddit.com/r/haskell/comments/3qbgmo/fp_complete_the_new_haskellide_repo/

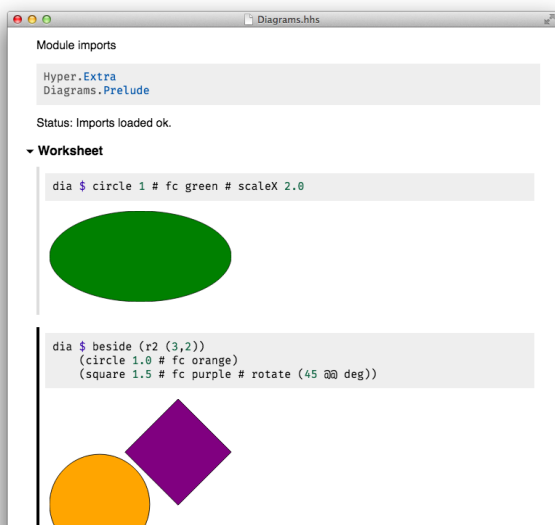
4.5.3 HyperHaskell – The strongly hyped Haskell interpreter

Report by:	Heinrich Apfelmus
Status:	available, active development

HyperHaskell is a graphical Haskell interpreter, not unlike GHCi, but hopefully more awesome. You use worksheets to enter expressions and evaluate them. Results are displayed graphically using HTML.

HyperHaskell is intended to be *easy to install*. It is cross-platform and should run on Linux, Mac and Windows. Internally, it uses the GHC API to interpret Haskell programs, and the graphical front-end is built on the Electron framework. *HyperHaskell* is open source.

HyperHaskell's main attraction is a `Display` class that supersedes the good old `Show` class. The result looks like this:



Status

HyperHaskell is currently *Level α* . The latest stable release is 0.1.0.2. Compared to the previous report, no new release has been made, but basic features are working. It is now possible to interpret statements in the IO monad and to bind variables, greatly enhancing the usefulness of the interpreter.

Support for the Nix package manager has been implemented by Rodney Lorrimar. I am looking for help in setting up binary releases on the Windows platform!

Future development

Programming a computer usually involves writing a program text in a particular language, a “verbal” activity. But computers can also be instructed by gestures, say, a mouse click, which is a “nonverbal” activity. The long term goal of *HyperHaskell* is to blur the lines between programming “verbally” and “non-verbally” in Haskell. This begins with an interpreter that has graphical representations for values, but also includes editing a program text while it’s running (“live coding”) and interactive representations of values (e.g. “tangible values”). This territory is still largely uncharted from a purely functional perspective, probably due to a lack of easily installed graphical facilities. It is my hope that *HyperHaskell* may provide a common ground for exploration and experimentation in this direction, in particular by offering the `Display` class which may, perhaps one day, replace our good old `Show` class.

A simple form of live coding is planned for *Level β* , and I am experimenting with interactive music programming.

Further reading

- o Project homepage and downloads: <https://github.com/HeinrichApfelmus/hyper-haskell>

4.5.4 CodeWorld

Report by:	Chris Smith
Status:	actively used

CodeWorld is a web-based educational programming environment using Haskell, and appropriate for all ages. It provides a simple mathematical model for geometric figures, animations, and interactive and multi-player games. The language scales between a graphical block-based language for primary students, a simplified variant of Haskell, and the full-fledged Haskell language for older students and universities. In addition to the tools, CodeWorld also provides learning resources for teachers and independent learners. CodeWorld is actively used for Haskell programming classes and activities, by universities, primary and secondary schools, and non-profit organizations and programs.

Features

- A cloud-based programming environment available from anywhere, to write and run code directly in the browser.
- A full-featured Haskell editor with syntax highlighting, rainbow brackets, formatting, and auto-complete.
- A simple graphics model for composable geometry and animations.
- Integrated debugging tools that intelligently link program output to the lines of code responsible.
- The world's simplest framework for single and multi-player (networked) games.

Recent changes

In the summer of 2017, CodeWorld hosted four students through the Summer of Haskell program to work on improving debugging tools, error messages, collaborative coding experiences, and exporting projects to video and mobile applications. Some contributions are still being merged. Other recent changes include the addition of a model for simple multi-player networked games and QuickCheck support for the full Haskell mode. Simultaneously, we've been busy developing a packaged curriculum for early secondary students, ages 11-14, using functional programming as a framework for creative mathematics.

Availability

CodeWorld is freely available. The hosted web site at <http://code.world> is open to the public. Source code for the project is available under the Apache license. Teaching materials and resources are released as they are completed under a Creative Commons license.

Further reading

<https://github.com/google/codeworld>

4.5.5 Haskell Indexer

Report by:	Ivan Krišto
Participants:	Robin Palotai, Kristoffer S�holm
Status:	stable, actively developed

Haskell Indexer is a [Kythe](https://github.com/google/kythe) extension for working with Haskell source code. Kythe is language-agnostic ecosystem for building tools that work with code. An example is code search with cross-reference support: <https://cs.chromium.org/>. Haskell Indexer makes it possible to use Kythe-based tools with Haskell. With Haskell Indexer it's possible to list all use-sites of any given function (get reverse-references) and explore the code without any IDE setup.

A portion of GHC and Stackage is indexed and available at <http://stuff.codereview.me/>.

```
254
255 -- | Produce an environment with a custom set of fonts.
256 -- The default fonts are still loaded as fall back.
257 createEnv :: (Read n, RealFloat n)
258           => AlignmentFns -- ^ Alignment functions to use.
259           -> n -- ^ The output image width in backend coordinates.
260           -> n -- ^ The output image height in backend coordinates.
261           -> FontSelector n -> DEnv n
262 createEnv alignFns w h fontSelector = DEnv
263   { envAlignmentFns = alignFns
264   , envFontStyle = def
265   , envSelectFont = fontSelector
266   , envOutputSize = (w,h)
267   , envUsedGlyphs = M.empty
268   }
269
270 -- | Produce a default environment with just the sans-serif fonts.
```

• Definitions:

- Chart-diagrams-1.8.2/Graphics/Rendering/Chart/Backend/Diagrams.hs
262: createEnv alignFns w h fontSelector = DEnv

• References:

- Chart-diagrams-1.8.2/Graphics/Rendering/Chart/Backend/Diagrams.hs
123: let env = createEnv vectorAlignmentFns w h fontSelector
257: createEnv :: (Read n, RealFloat n)
279: return (createEnv alignFns w h fontSelector)

Haskell Indexer is in active use and development. As of today, it's possible to get it from [GitHub](https://github.com/google/haskell-indexer) – the Hackage release is work in progress. The future plans include improving the cross-reference support, adding cross-language linking with C and better handling of Template Haskell.

Further reading

- <https://github.com/google/haskell-indexer>
- <https://kythe.io/>

4.5.6 Brittany

Report by:	Lennart Spitzner
Status:	work in progress

Brittany is a Haskell source code formatting tool. It is based on `ghc-exactprint` and thus uses the `ghc` parser, in contrast to tools based on `haskell-src-exts` such as `hindent` or `haskell-formatter`.

The goals of the project are to:

- support the full `ghc-haskell` syntax including syntactic extensions;
- retain newlines and comments unmodified (to the degree possible when code around them gets reformatted);
- be clever about using horizontal space while not overflowing it if it cannot be avoided;
- have linear complexity in the size of the input text / the number of syntactic nodes in the input.
- support horizontal alignments (e.g. different equations/pattern matches in the some function's definition).

In contrast to other formatters `brittany` internally

works in two steps: Firstly transforming the syntax tree into a document tree representation, similar to the document representation in general-purpose pretty-printers such as the *pretty* package, but much more specialized for the specific purpose of handling a Haskell source code document. Secondly this document representation is transformed into the output text document. This approach allows to handle many different syntactic constructs in a uniform way, making it possible to attain the above goals with a manageable amount of work.

Brittany is work in progress; currently only type signatures and function bindings are transformed, and not all syntactic constructs are supported. Nonetheless Brittany is safe to try/use as there are checks in place to ensure that the output is syntactically valid.

Brittany requires `ghc-8.*`, and is available on Hackage and on Stackage.

Further reading

- <https://github.com/lspitzner/brittany>

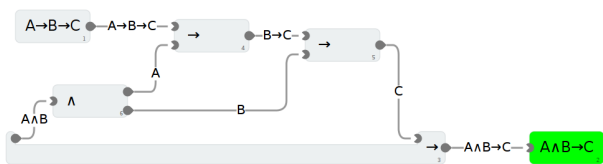
4.6 Formal Systems and Reasoners

4.6.1 The Incredible Proof Machine

Report by:	Joachim Breitner
Status:	active development

The Incredible Proof Machine is a visual interactive theorem prover: Create proofs of theorems in propositional, predicate or other, custom defined logics simply by placing blocks on a canvas and connecting them. You can think of it as Simulink mangled by the Curry-Howard isomorphism.

It is also an addictive and puzzling game, I have been told.



The Incredible Proof Machine runs completely in your browser. While the UI is (unfortunately) boring standard JavaScript code with a spaghetti flavor, all the logical heavy lifting is done with Haskell, and compiled using GHCJS.

Further reading

- <http://incredible.nomeata.de> The Incredible Proof Machine
- <https://github.com/nomeata/incredible> Source Code
- http://www.joachim-breitner.de/blog/682-The_Incredible_Proof_Machine Announcement blog post

4.6.2 Exference

Report by:	Lennart Spitzner
Status:	experimental, active development

Exference is a tool aimed at supporting developers writing Haskell code by generating expressions from a type, e.g.

Input:

```
(Show b) => (a -> b) -> [a] -> [String]
```

Output:

```
\ f1 -> fmap (show . f1)
```

Input:

```
(Monad m, Monad n)
=> ([a] -> b -> c) -> m [n a] -> m (n b)
-> m (n c)
```

Output:

```
\ f1 -> liftA2 (\ hs i ->
  liftA2 (\ n os -> f1 os n) i (sequenceA hs))
```

The algorithm does a proof search specialized to the Haskell type system. In contrast to Djinn, the well known tool with the same general purpose, Exference supports a larger subset of the Haskell type system - most prominently type classes. The cost of this feature is that Exference makes no promise regarding termination (because the problem becomes an undecidable one; a draft of a proof can be found in the pdf below). Of course the implementation applies a time-out.

There are two primary use-cases for Exference:

- In combination with typed holes: The programmer can insert typed holes into the source code, retrieve the expected type from `ghc` and forward this type to Exference. If a solution, i.e. an expression, is found and if it has the right semantics, it can be used to fill the typed hole.
- As a type-class-aware search engine. For example, Exference is able to answer queries such as `Int -> Float`, where the common search engines like `hoogle` or `hayoo` are not of much use.

The current implementation is functional and works well. The most important aspect that still could use improvement is the performance, but it would probably take a slightly improved approach for the core algorithm (and thus a major rewrite of this project) to make significant gains.

The project is actively maintained; apart from occasional bug-fixing and general maintenance/refactoring there are no major new features planned currently.

Try it out by on IRC(freenode): `exferenceBot` is in `#haskell` and `#exference`.

Further reading

- <https://github.com/lspitzner/exference>
- <https://github.com/lspitzner/exference/raw/master/exference.pdf>

4.7 Education

4.7.1 Holmes, Plagiarism Detection for Haskell

Report by:	Jurriaan Hage
Participants:	Brian Vermeer, Gerben Verburg

Holmes is a tool for detecting plagiarism in Haskell programs. A prototype implementation was made by Brian Vermeer under supervision of Jurriaan Hage, in order to determine which heuristics work well. This implementation could deal only with Helium programs. We found that a token stream based comparison and Moss style fingerprinting work well enough, if you remove template code and dead code before the comparison. Since we compute the control flow graphs anyway, we decided to also keep some form of similarity checking of control-flow graphs (particularly, to be able to deal with certain refactorings).

In November 2010, Gerben Verburg started to reimplement Holmes keeping only the heuristics we figured were useful, basing that implementation on `haskell-src-externs`. A large scale empirical validation has been made, and the results are good. We have found quite a bit of plagiarism in a collection of about 2200 submissions, including a substantial number in which refactoring was used to mask the plagiarism. A paper has been written, which has been presented at CSERC'13, and should become available in the ACM Digital Library.

The tool will be made available through Hackage at some point, but before that happens it can already be obtained on request from Jurriaan Hage.

Contact

J.Hage@uu.nl

4.7.2 DSLsofMath

Report by:	Patrik Jansson
Participants:	Cezar Ionescu, Daniel Heurlin
Status:	active development

“Domain Specific Languages of Mathematics” is a project at Chalmers and UGOT which lead to a new BSc level course of the same name, including accompanying material for learning and applying classical mathematics (mainly basics of real and complex analysis). The main idea is to encourage the students to approach mathematical domains from a functional programming perspective: to identify the main functions and types involved and, when necessary, to introduce new abstractions; to give calculational proofs; to pay attention to the syntax of the mathematical expressions; and, finally, to organize the resulting functions and types in domain-specific languages.

The second instance of the course was carried out Jan-March 2017 at Chalmers and the course material is available on [github](#). The next step (ongoing work) is to write up the lecture notes as a “book” during the autumn, in preparation for the next instance of the course early 2018. Contributions and ideas are welcome!

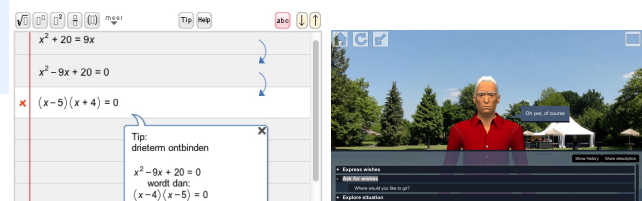
Further reading

- [DSLsofMath](#) (github organisation)
- [TFPIE 2015 paper](#)
- [Latest snapshot of the DSLsofMath Lecture Notes](#) (work in progress)
- [Exam 2017 with solutions](#)

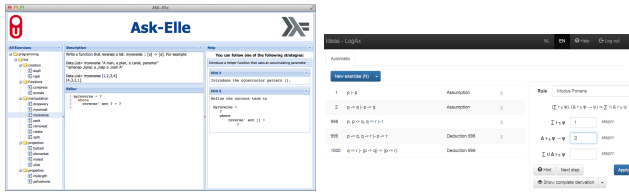
4.7.3 Interactive Domain Reasoners

Report by:	Bastiaan Heeren
Participants:	Johan Jeuring, Alex Gerdes, Josje Lodder, Hieke Keuning, Ivica Milovanovic
Status:	experimental, active development

IDEAS (Interactive Domain-specific Exercise Assistants) is a joint research project between the Open University of the Netherlands and Utrecht University. The project’s goal is to use software and compiler technology to build state-of-the-art components for intelligent tutoring systems (ITS), learning environments, and applied games. The ‘ideas’ software package provides a generic framework for constructing the expert knowledge module (also known as a domain reasoner) for an ITS or learning environment. Domain knowledge is offered as a set of feedback services that are used by external tools such as the digital mathematical environment (first/left screenshot) and the Math-Bridge system. We have developed several domain reasoners based on this framework, including reasoners for mathematics, linear algebra, statistics, propositional logic, for learning Haskell (the Ask-Elle programming tutor) and evaluating Haskell expressions, and for practicing communication skills (the serious game *Communicate!*, second/right screenshot).



We have continued working on the domain reasoners that are used by our programming tutors. The Ask-Elle functional programming tutor lets you practice introductory functional programming exercises in Haskell. We have extended this tutor with QuickCheck properties for testing the correctness of student programs, and for the generation of counterexamples. We have analysed the usage of the tutor to find out how many student submissions are correctly diagnosed as right or wrong.



We have just started with the [Advise-Me project](#) (Automatic Diagnostics with Intermediate Steps in Mathematics Education), which is a Strategic Partnership in EU's Erasmus+ programme. In this project we develop innovative technology for calculating detailed diagnostics in mathematics education, for domains such as 'Numbers' and 'Relationships'. The technology is offered as an open, reusable set of feedback and assessment services. The diagnostic information is calculated automatically based on the analysis of intermediate steps.

We are continuing our research in various directions. We are investigating feedback generation for [axiomatic proofs](#) for propositional logic. We also want to add student models to our framework and use these to make the tutors more adaptive, and develop authoring tools to simplify the creation of domain reasoners.

The library for developing domain reasoners with feedback services is available as a [Cabal source package](#). We have written a [tutorial](#) on how to make your own domain reasoner with this library. The [domain reasoner](#) for mathematics and logic has been released as a separate package.

Further reading

- Bastiaan Heeren, Johan Jeuring, and Alex Gerdes. [Specifying Rewrite Strategies for Interactive Exercises](#). *Mathematics in Computer Science*, 3(3):349–370, 2010.
- Bastiaan Heeren and Johan Jeuring. [Feedback services for stepwise exercises](#). *Science of Computer Programming, Special Issue on Software Development Concerns in the e-Learning Domain*, volume 88, 110–129, 2014.
- Alex Gerdes, Bastiaan Heeren, Johan Jeuring, and Thomas Binsbergen. [Ask-Elle: an Adaptable Programming Tutor for Haskell Giving Automated Feedback](#). *Journal of Artificial Intelligence in Education* 2016.
- Josje Lodder, Bastiaan Heeren, and Johan Jeuring. [Generating hints and feedback for Hilbert-style axiomatic proofs](#). To appear in SIGCSE 2017.

4.8 Text and Markup

4.8.1 lhs2TeX

Report by:	Andres Löh
Status:	stable, maintained

This tool by Ralf Hinze and Andres Löh is a preprocessor that transforms literate Haskell or Agda code into L^AT_EX documents. The output is highly customizable by means of formatting directives that are interpreted by lhs2TeX. Other directives allow the selective inclusion of program fragments, so that multiple versions of a program and/or document can be produced from a common source. The input is parsed using a liberal parser that can interpret many languages with a Haskell-like syntax.

The program is stable and can take on large documents.

The current version is 1.19 and has been released in April 2015. Development repository and bug tracker are on GitHub. The tool is mostly in plain maintenance mode, although there are still vague plans for a complete rewrite of lhs2TeX, hopefully cleaning up the internals and making the functionality of lhs2TeX available as a library.

Further reading

- <http://www.andres-loeh.de/lhs2tex>
- <https://github.com/kosmik/lhs2tex>

4.8.2 Lentil

Report by:	Francesco Ariis
Status:	working

Lentil helps the programmers who litter their code with TODOs and FIXMEs.

Lentil goes through a project and outputs all issues in a pretty format, referencing their file/line position. As today it recognises Haskell, Javascript, C/C++, Python, Ruby, Pascal, Perl, Shell and Nix source files, plus plain .txt files.

Lentil syntax allows you to put [tag]s in your issues, which can then be used to filter/extract/export data.

Current version is 0.1.12.0, which introduces new flag-words, recognised languages (html, elm, coffee-script, typescript) and export formats (xml).

Further reading

- manual: <http://ariis.it/static/articles/lentil/page.html>
- decentralised issue tracking: <http://ariis.it/static/articles/decentralised-lentil/page.html>

4.8.3 Unicode things

Report by:	Antonio Nikishaev
Status:	work in progress

Many programming languages offer non-existing or very poor support for Unicode. While many think that Haskell is not one of them, this is not completely true. The way-to-go library of Haskell's string type, `Text`, only provides codepoint-level operations. Just as a small and very elementary example: two “Haskell café” strings, first written with the ‘é’ character, and the second with the ‘e’ character followed by a combining acute accent character, obviously have a correspondence for many real-world situations. Yet they are entirely different and unconnected things for `Text` and its operations.

And even though there is `text-icu` library offering proper Unicode functions, it has a form of FFI bindings to C library (and that is painful, especially for Windows users). More so, its API is very low-level and incomplete.

`Prose` is a work-in-progress pure Haskell implementation of Unicode strings. Right now it's completely unoptimized. Implemented parts are normalization algorithms and segmentation by graphemes and words.

`Numerals` is pure Haskell implementation of CLDR (Common Language Data Repository, Unicode's locale data) numerals formatting.

Contributions and comments are always welcome!

Further reading

- <http://lelf.lu/prose>
- <https://github.com/lelf/prose>
- <https://github.com/lelf/numerals>

4.8.4 Fast Unicode Normalization

Report by:	Harendra Kumar
Status:	Working

Unicode strings need to be converted to a normalized form using the Unicode Character Database before they can be compared for equivalence. `unicode-transforms` is a pure Haskell implementation of Unicode normalization. The alternative is the `text-icu` package which provides this functionality as Haskell bindings to the ICU C++ implementation. `unicode-transforms` supports all forms of normalization (NFC/NFD/NFKC/NFKD) and supports the latest version of the Unicode standard (Unicode 9).

One of the goals of `unicode-transforms` was high performance. We have successfully achieved this goal for decompose (NFD/NFKD) forms, achieving performance close to, and in one benchmark even better than the C++ implementation (i.e. the `text-icu` package). Compose (NFC/NFKC) implementation is not yet optimized, and though the performance of compose is decent it is not at par with the C++ implementation.

This is still open for anyone looking for a challenge to beat C++.

This library can potentially be integrated with the `text` package allowing us to keep text in a standard normalized form by default, thus freeing the Haskell programmers from worrying about explicit normalization. The library is available on Hackage under BSD3 license.

Further reading

<https://github.com/harendra-kumar/unicode-transforms>

4.8.5 Ginger

Report by:	Tobias Dammers
Status:	Active development

Ginger is a Haskell implementation of the Jinja2 HTML template language. Unlike most existing Haskell templating solutions, Ginger expands templates at runtime, not compile time; this is a deliberate design decision, intended to support a typical rapid-cycle web development workflow. Also unlike most existing Haskell HTML DSLs, Ginger is completely unaware of the DOM, and does not enforce well-formed HTML. Just like Jinja2, however, it does distinguish HTML source and raw values at the type level, meaning that HTML encoding is automatic and (mostly) transparent, avoiding the most common source of XSS vulnerabilities. For a quick impression of what Ginger syntax looks like:

```
<section class="page">
  <h1>{{ page.title }}</h1>
  {% if page.image %}
    
  {% endif %}
  <section class="teaser">
    {{ page.teaser }}
  </section>
  <section class="content">
    {{ page.body|markdown }}
  </section>
  <section class="page-meta">
    Submitted by {{ page.author }} on
    {{page.submitted|formatDate('%Y-%m-%d')}}
  </section>
</section>
```

All the important features of Jinja2 have been implemented, and the library is fully usable for production work. Some features of the original Jinja2 have been left out because the author considers them Pythonisms; others are missing simply because they haven't been implemented yet. Additionally, some features have been added that are missing in Jinja2, such as lambdas, being able to use macros as functions or filters, ‘do’ expressions, output indenting constructs, and “script mode”, switching Ginger into a syntax that is closer

to a untyped scripting language than a template language.

Improvement that haven't made it yet include TemplateHaskell support (which would allow programmers to compile Ginger templates directly into the binary, and perform template compilation at compile time rather than runtime), a built-in caching mechanism, and more configuration options. Contributions of any kind are very welcome.

Further reading

- <https://ginger.tobiasdammers.nl/>
- <https://github.com/tdammers/ginger>
- <http://hackage.haskell.org/package/ginger>
- <http://jinja2.pocoo.org> (the original Jinja2, not my work)

4.9 Web

4.9.1 WAI

Report by:	Kazu Yamamoto
Participants:	Michael Snoyman, Greg Weber
Status:	stable

WAI (Web Application Interface) is an application interface between web applications and handlers in Haskell. The `Application` data type is defined as follows:

```
type Application
= Request
-> (Response -> IO ResponseReceived)
-> IO ResponseReceived
```

That is, a WAI application takes two arguments: a `Request` and a function to send a `Response`. So, the typical behavior of WAI application is processing a request, generating a response and passing the response to the function.

Historically speaking, this interface made possible to develop handlers other than HTTP. The WAI applications can run through FastCGI (`wai-handler-fastcgi`), run as stand-alone (`wai-handler-webkit`), etc. But the most popular handler is based on HTTP, of course. The major HTTP handler for WAI is Warp which now provides both HTTP/1.1 and HTTP/2. TLS (`warp-tls`) is also available. New transports such as WebSocket (`wai-websocket`) and Event Source (`wai-extra`) can be implemented, too.

It is possible to develop WAI applications directly. For instance, Hoogle and Mighttpd2 take this way. However, you may want to use web application frameworks such as Apiary, MFlow, rest, Servant, Scotty, Spock, Yesod, etc.

WAI also provides `Middleware`:

```
type Middleware = Application -> Application
```

WAI middleware can inspect and transform a request, for example by automatically gzipping a response or logging a request (`wai-extra`).

Since the last HCAR, no major changes were made.

Further reading

- <https://groups.google.com/d/forum/haskell-wai>

4.9.2 Warp

Report by:	Kazu Yamamoto
Participants:	Michael Snoyman
Status:	stable

Warp is a high performance, easy to deploy HTTP handler for WAI (→ 4.9.1). Its supports both HTTP/1.1 and HTTP/2.

Since the last HCAR, some bugs of Warp have been fixed. Also, Warp TLS incorporated the `tls-session-manager` package which enables TLS session cache in the server side.

Further reading

- “Warp: A Haskell Web Server”
 - the May/June 2011 issue of IEEE Internet Computing
 - Issue page:
<http://www.computer.org/portal/web/csdl/abs/mags/ic/2011/03/mic201103toc.htm>
 - PDF: http://steve.vinoski.net/pdf/IC-Warp_a_Haskell_Web_Server.pdf
- “Warp”
 - The Performance of Open Source Applications
 - HTML:
<http://www.aosabook.org/en/posa/warp.html>

4.9.3 Yesod

Report by:	Michael Snoyman
Participants:	Greg Weber, Luite Stegeman, Felipe Lessa
Status:	stable

Yesod is a traditional MVC RESTful framework. By applying Haskell's strengths to this paradigm, Yesod helps users create highly scalable web applications.

Performance scalability comes from the amazing GHC compiler and runtime. GHC provides fast code and built-in event-based asynchronous IO.

But Yesod is even more focused on scalable development. The key to achieving this is applying Haskell's type-safety to an otherwise traditional MVC REST web framework.

Of course type-safety guarantees against typos or the wrong type in a function. But Yesod cranks this up a notch to guarantee common web application errors won't occur.

- declarative routing with type-safe urls — say goodbye to broken links
- no XSS attacks — form submissions are automatically sanitized
- database safety through the Persistent library (→ 4.10.1) — no SQL injection and queries are always valid
- valid template variables with proper template insertion — variables are known at compile time and treated differently according to their type using the shakespearean templating system.

When type safety conflicts with programmer productivity, Yesod is not afraid to use Haskell’s most advanced features of Template Haskell and quasi-quoting to provide easier development for its users. In particular, these are used for declarative routing, declarative schemas, and compile-time templates.

MVC stands for model-view-controller. The preferred library for models is Persistent (→ 4.10.1). Views can be handled by the Shakespeare family of compile-time template languages. This includes Hamlet, which takes the tedium out of HTML. Both of these libraries are optional, and you can use any Haskell alternative. Controllers are invoked through declarative routing and can return different representations of a resource (html, json, etc).

Yesod is broken up into many smaller projects and leverages Wai (→ 4.9.1) to communicate with the server. This means that many of the powerful features of Yesod can be used in different web development stacks that use WAI such as Scotty and Servant.

Yesod has been in API stability for some time. The 1.4 release was made in September of 2014, and we are still backwards-compatible to that. Even then, the 1.4 release was almost a completely backwards-compatible change. The version bump was mostly performed to break compatibility with older versions of dependencies, which allowed us to remove approximately 500 lines of conditionally compiled code. Notable changes in 1.4 include:

- New routing system with more overlap checking control.
- yesod-auth works with your database and your JSON.
- yesod-test sends HTTP/1.1 as the version.
- Type-based caching with keys.

The Yesod team is quite happy with the current level of stability in Yesod. Since the 1.0 release, Yesod has maintained a high level of API stability, and we intend to continue this tradition. Future directions for Yesod are now largely driven by community input and patches. We’ve been making progress on the goal of easier client-side interaction, and have high-level interaction with languages like Fay, TypeScript, and CoffeeScript. GHCJS support is in the works.

The Yesod site (<http://www.yesodweb.com/>) is a great place for information. It has code examples, screencasts, the Yesod blog and — most importantly

— a book on Yesod.

To see an example site with source code available, you can view Haskellers (→ 1.2) source code: (<https://github.com/snoyberg/haskellers>).

Further reading

<http://www.yesodweb.com/>

4.9.4 Happstack

Report by:	Jeremy Shaw
------------	-------------

Happstack is a diverse collection of libraries for creating web applications in Haskell. Libraries include support for type-safe routing, HTML templating, form validation, authentication and more.

In the last six months we have added two new experimental packages: `happstack-servant` and `happstack-websockets`. `happstack-servant` makes it easy to use Happstack with the new `servant` framework. `happstack-websockets` provides support for using `websockets`.

Further reading

- <http://www.happstack.com/>
- <http://www.happstack.com/docs/crashcourse/index.html>

4.9.5 Snap Framework

Report by:	Doug Beardsley
Participants:	Gregory Collins, Greg Hale, and others
Status:	active development

The Snap Framework is a web application framework built from the ground up for speed, reliability, stability, and ease of use. The project’s goal is to be a cohesive high-level platform for web development that leverages the power and expressiveness of Haskell to make building websites quick and easy.

Snap continues to see active use in industry and continued support from the development team. If you would like to contribute, get a question answered, or just keep up with the latest activity, stop by the `#snapframework` IRC channel on Freenode.

Further reading

- <http://snapframework.com>
- snap-server test coverage: https://snapframework.github.io/snap-code-coverage/snap-server/hpc-ghc-8.0.1/hpc_index.html
- Snaplet Directory: <http://snapframework.com/snaplets>
- io-streams: <http://hackage.haskell.org/package/io-streams>

4.9.6 MFlow

Report by:	Alberto Gómez Corona
Status:	active development

MFlow is a Web framework of the kind of other functional, stateful frameworks like WASH, Seaside, Ocsigen or Racket. MFlow does not use continuation passing properly, but a backtracking monad that permits the synchronization of browser and server and error tracing. This monad is on top of another “Workflow” monad that adds effects for logging and recovery of process/session state. In addition, MFlow is RESTful. Any GET page in the flow can be pointed to with a REST URL.

The navigation as well as the page results are type safe. Internal links are safe and generate GET requests. POST request are generated when formlets with form fields are used and submitted. It also implements monadic formlets: They can modify themselves within a page. If JavaScript is enabled, the widget refreshes itself within the page. If not, the whole page is refreshed to reflect the change of the widget.

MFlow hides the heterogeneous elements of a web application and expose a clear, modular, type safe DSL of applicative and monadic combinators to create from multipage to single page applications. These combinators, called widgets or enhanced formlets, pack together javascript, HTML, CSS and the server code.

A paper describing the MFlow internals has been published in The Monad Reader issue 23.

Further reading

- o MFlow as a DSL for web applications <https://www.fpcomplete.com/school/to-infinity-and-beyond/older-but-still-interesting/MFlowDSL1>
- o MFlow, a continuation-based web framework without continuations <http://themonadreader.wordpress.com/2014/04/23/issue-23>
- o How Haskell can solve the integration problem <https://www.fpcomplete.com/school/to-infinity-and-beyond/pick-of-the-week/how-haskell-can-solve-the-integration-problem>
- o Towards a deeper integration: A Web language: <http://haskell-web.blogspot.com.es/2014/04/towards-deeper-integration-web-language.html>
- o Perch <https://github.com/agocorona/haste-perch>
- o hplayground demos <http://tryplayg.herokuapp.com>
- o haste-perch-hplaygroun tutorial <http://www.airpair.com/haskell/posts/haskell-tutorial-introduction-to-web-apps>
- o react.js a solution for a problem that Haskell can solve in better ways <http://haskell-web.blogspot.com.es/2014/11/browser-programming-reactjs-as-solution.html>
- o MFlow demo site: <http://mflowdemo.herokuapp.com>

4.9.7 PureScript

Report by:	Phil Freeman
Status:	active, looking for contributors

PureScript is a small strongly typed programming language that compiles to efficient, readable JavaScript. The PureScript compiler is written in Haskell.

The PureScript language features Haskell-like syntax, type classes with functional dependencies, rank-n types, and row polymorphism with a form of polymorphic labels.

PureScript features a comprehensive standard library, and a large number of other libraries and tools under development, covering data structures, algorithms, Javascript integration, web services, game development, testing, asynchronous programming, FRP, graphics, audio, UI implementation, and many other areas. It is easy to wrap existing Javascript functionality for use in PureScript, making PureScript a great way to get started with strongly-typed pure functional programming on the web. PureScript is currently used successfully in production in commercial code.

The PureScript compiler can be downloaded from purescript.org, or compiled from source from Hackage.

Further reading

<https://github.com/purescript/purescript/>

4.9.8 Sprinkles

Report by:	Tobias Dammers
Status:	Active development

Sprinkles is a “zero programming web development framework”, intended for building content-centric server-site websites in a declarative fashion.

As such, Sprinkles sits in a unique position in between static site generators such as Jekyll, Hekyll, etc., and fully dynamic CMS solutions like Ghost, WordPress, etc.

A Sprinkles website consists of a project file in YAML format, a set of templates (using Ginger, a Haskell implementation of the Jinja HTML template language), and static files such as stylesheets, images, and client-side scripts. Data can be loaded from SQL databases, local files, HTTP, or local shell scripts, and Sprinkles supports a wide range of input formats, including JSON, YAML, HTML, Markdown, LaTeX, and even DOCX. The heart of a project file is a list of routes, each specifying a list of backend data sources to query, the output of which then gets bound to a template variable.

Unlike a static site generator, Sprinkles generates its HTML output on the fly, so there is no build step, you just deploy your updated project files to the server, restart the Sprinkles process, and your new site is live. Unlike a classic CMS, however, there is no admin area; instead, you manage your data externally.

This approach has a few advantages, most notably, it makes Sprinkles more secure, because no admin area is exposed that could be exploited to escalate from compromising the public-facing part to gain write access; write access is implemented separately, via separate channels (typically an SSH connection).

Various workflows are possible, e.g.:

- hand-editing markdown files or even Word documents
- putting your data in an SQL database, using a graphical SQL frontend to modify things, and pointing Sprinkles at the database
- using an existing website as your data source, but building a Sprinkles frontend on top
- using github as your data source (the Ginger homepage at <https://ginger.tobiasdammers.nl/> does this for the user guide section)
- using something like <https://form.io/> to manage your data, and pointing Sprinkles at an API endpoint
- ...

Sprinkles comes with a built-in Warp server, and this is the preferred deployment option, but CGI, SCGI and FastCGI are also supported. A “bake” mode has been added recently that turns Sprinkles into a static site generator, so a Jekyll-style workflow is now also possible.

Further reading

- <https://sprinkles.tobiasdammers.nl/>
- <https://github.com/tdammers/sprinkles>
- <http://hackage.haskell.org/package/sprinkles>

4.9.9 nginx-haskell-module

Report by:	Alexey Radkov
Status:	stable, actively developed

The `nginx-haskell-module` allows using typed Haskell handlers from within custom Nginx configurations. It provides several Nginx directives that correspond to where Haskell code is applied and what kind of task it solves:

- `haskell_run` — run Haskell handler synchronously and return a string-like result;
- `haskell_run_async` — run Haskell handler asynchronously in the early rewrite Nginx phase and return a string-like result when ready;
- `haskell_content` — run Haskell handler for rendering content;
- `haskell_static_content` — optimized content rendering;
- `haskell_unsafe_content` — optimized (in another way) content rendering;
- `haskell_run_service` — run Haskell handler as a service, meaning that this implements not a request-

driven task but rather timer-driven and is directly bound to the Nginx event loop.

Here, “a string-like result” is an umbrella term for various Haskell string-like implementations like `String` and `ByteString`, perhaps wrapped inside `IO Monad`. Typed Haskell handlers give a strong guarantee that underlying code will cause no IO specific side effects thus making sure that Nginx will not stop working due to unpredictably long-running IO code, which is extremely crucial for this popular web server.

On the other hand, when effectful code is desirable, it can be declared with directive `haskell_run_async`, which runs Haskell code asynchronously and won’t stop the Nginx world. Moreover, the module provides directive `haskell_run_service` for running custom asynchronous “services” that are not bound to requests: this makes it possible to program interesting solutions like Nginx-specific service discovery (see a reference to an advanced example below). An asynchronous service results are available in request-specific Nginx configuration areas (e.g. `location`) via an Nginx variable that holds a string, which, on a deeper level, may wrap a typed data (or JSON, or other representations).

Basically, a service results variable is stored in a worker-specific memory, but with directive `haskell_service_var_in_shm` it gets its place in a shared memory and becomes available from all Nginx workers. In conjunction with directive `haskell_service_var_update_callback`, this allows programming custom ad-hoc solutions when a Haskell service’s results are needed in another Nginx module. On the Github project page there is an advanced example `dynamicUpstreams` that implements such an approach.

Starting from version 1.4.0 of the module, services that share their results in shared memory become *shared* themselves. This means that only one worker process really runs a shared service, while services on the other workers wait on locks until the active worker exits or dies.

The module provides other two directives `haskell_var_nocacheable` and `haskell_var_compensate_uri_changes` that make Nginx `error_page` redirections almost Turing-complete. It is achieved by letting them loop without limits on number of cycles (by the second directive) and ensuring that loop-specific variables get updated between iterations (by the first directive). The technique of Turing-complete `error_page` redirections is used in an advanced example `labeledMediaRouting`.

Further reading

- [Project page](#)
- [Module ngx-export on Hackage](#)
- [An advanced example in a blog article](#)

4.9.10 Hapistrano

Report by:	Stack Builders
Status:	active

Hapistrano deploys Haskell applications to a new directory marked with a timestamp on the remote host. It creates this new directory quickly by placing a Git repository for caching purposes on the remote server.

When the build process completes, it switches a symlink to the current release directory, and optionally restarts the web server. By default, Hapistrano keeps the last five releases on the target host filesystem and deletes previous releases to avoid filling up the disk.

Hapistrano has been almost completely rewritten and new features have been added. The reason for such a rewrite was that we wished to make configuration easier and to avoid re-building our software on target hosts when possible, thus making CI cycles that deploy our software much quicker.

Hapistrano is available on [Hackage](#), and contribution are welcome in the [GitHub repository](#).

Further reading

- <https://github.com/stackbuilders/hapistrano>
- [Hapistrano update](#)

4.9.11 Template Toolkit

Report by:	Dzianis Kabanau
Status:	active, working

Haskell implementation of popular Perl template processing system.

Perl Template Toolkit is mainly used for web development.

This port includes such features as:

- Perl-like variables: scalar, array, hash
- Variables interpolation and autovivification.
- Conditional directives
- Loops and loop controls
- Processing of external templates and internal sub-template blocks
- Many virtual methods and filters

In the next release it is planned to include support for custom methods, filters and functions in templates.

Further reading

<https://hackage.haskell.org/package/template-toolkit>

4.10 Databases

4.10.1 Persistent

Report by:	Michael Snoyman
Participants:	Greg Weber, Felipe Lessa
Status:	stable

Since the last HCAR, persistent has mostly experienced bug fixes, including recent fixes and increased backend support for the new flexible primary key type.

Haskell has many different database bindings available, but most provide few useful static guarantees. Persistent uses knowledge of the data schema to provide a type-safe interface to the database. Persistent is designed to work across different databases, currently working on Sqlite, PostgreSQL, MongoDB, MySQL, Redis, and ZooKeeper.

Persistent provides a high-level query interface that works against all backends.

```
selectList [PersonFirstName == . "Simon",  
            PersonLastName == . "Jones"] []
```

The result of this will be a list of Haskell records.

Persistent can also be used to write type-safe query libraries that are specific. `esqueleto` is a library for writing arbitrary SQL queries that is built on Persistent.

Future plans

Persistent is in a stable, feature complete state. Future plans are only to increase its ease for the places where it can be easily used:

- Declaring a schema separately from a record, possibly leveraging GHC's new annotations feature or another pattern

Persistent users may also be interested in Groundhog, a similar project.

Persistent is recommended to Yesod ([→ 4.9.3](#)) users. However, there is nothing particular to Yesod or even web development about it. You can have a type-safe, productive way to store data for any kind of Haskell project.

Further reading

- <http://www.yesodweb.com/book/persistent>
- <http://hackage.haskell.org/package/esqueleto>
- <http://www.yesodweb.com/blog/2014/09/persistent-2>
- <http://www.yesodweb.com/blog/2014/08/announcing-persistent-2>

4.10.2 Opaleye

Report by:	Tom Ellis
Status:	stable, active

Opaleye is an open-source library which provides an SQL-generating embedded domain specific language. It allows SQL queries to be written within Haskell in a typesafe and composable fashion, with clear semantics.

The project was publically released in December 2014. It is stable and actively maintained, and used in production in a number of commercial environments. Professional support is provided by Purely Agile.

Just like Haskell, Opaleye takes the principles of type safety, composability and semantics very seriously, and one aim for Opaleye is to be “the Haskell” of relational query languages.

In order to provide the best user experience and to avoid compatibility issues, Opaleye specifically targets PostgreSQL. It would be straightforward produce an adaptation of Opaleye targeting other popular SQL databases such as MySQL, SQL Server, Oracle and SQLite. Offers of collaboration on such projects would be most welcome.

Opaleye is inspired by theoretical work by David Spivak, and by practical work by the HaskellDB team. Indeed in many ways Opaleye can be seen as a spiritual successor to HaskellDB. Opaleye takes many ideas from the latter but is more flexible and has clearer semantics.

Further reading

<http://hackage.haskell.org/package/opaleye>

4.10.3 Squeal

Report by:	Eitan Chatav
Status:	experimental, active development

The `squeal-postgresql` package is a library providing a deep embedding of PostgreSQL in Haskell. Squeal makes use of lots of features and ideas in Haskell such as:

- DataKinds
- OverloadedLabels
- Indexed monads
- Generics
- GADTs

The point is not to use them in particularly new ways, but for very practical purposes. DataKinds are used to embed the SQL type and schema system into Haskell. OverloadedLabels are used to refer to tables and columns. Indexed monads are used to track changing schema types. Generics are used to easily marshal data between Haskell product and record types and SQL parameter and row types. GADTs are used for joins, aliased expressions, heterogeneous lists and more.

However, in spite of these perhaps being formidable sounding concepts, it is much easier to learn them when

there is a familiar pattern you can use, and that’s SQL. I believe that if a Haskeller knows SQL, then Squeal can be a gateway to learning these concepts. That’s because Squeal tries to be (mostly) faithful to SQL.

If the Squeal expressions are basically SQL, then why not just use SQL? Squeal adds strong, static typing and decomposability, which helps to develop and maintain an application. Change your schema and the compiler will tell you where you must change your queries and manipulations. Queries and manipulations with common subparts can be factored for reusability. You can even write typed schema migration definitions using Squeal.

Squeal is still experimental and I would love to get some real users and contributors. Send me bug reports and feature requests. There are still many missing features that I have plans to include.

Further reading

- <https://hackage.haskell.org/package/squeal-postgresql>
- <https://github.com/morphismtech/squeal>

4.10.4 Riak bindings

Report by:	Antonio Nikishaev
Status:	active development

`riak` is a Haskell binding to the Riak database. While stable and working, it has had only `riak-1.*` support. The author of this report entry has been recently working on fixing bugs and adding new `riak-2.*` features. Notable ones are: bucket types, high-level CRDT (Conflict-free replicated data types) support, basic search operations.

Further reading

- <http://hackage.haskell.org/package/riak>
- <https://github.com/markhibberd/riak-haskell-client>

4.10.5 Haskell Relational Record

Report by:	Kei Hibino
Participants:	Shohei Murayama, Shohei Yasutake, Sho Kuroda and Kazu Yamamoto
Status:	stable, active

Haskell Relational Record (HRR) is open-source libraries which provides a pragmatic embedded domain specific language to generate SQL. It supports a large part of the SQL standard which includes outer joins and aggregations with type safety, and also supports pragmatic but type-unsafe features like placeholder, correlation and direct SQL embedding interfaces. Direct SQL embedding interfaces allow database-system-dependent SQL code fragments.

HRR also provides template-haskell which generates record types and relation types definitions from relational-database schemas. Supported schemes of relational-database engines are PostgreSQL, MySQL, SQLite, IBM DB2, Microsoft SQL Server and OracleSQL.

HRR is publicly developed on github since 2013, and its release is publicly announced to Haskell community in December 2014. HRR has been in use at Asahi Net (Internet Service Provider in Japan) since March 2013, and more than three years of production use demonstrates its stability and usability. HRR is actively developed, and technical support is provided in Haskell-jp ([→ 6.13](#)).

Further reading

<http://khibino.github.io/haskell-relational-record/>

4.10.6 YeshQL

Report by:	Tobias Dammers
Status:	Stable, active development

YeshQL is a library to bridge the Haskell / SQL gap by implementing a quasi-quoter that allows programmers to write SQL queries in plain SQL, adding meta-information as structured SQL comments. The latter allows the quasi-quoter to generate a type-safe API for these queries. YeshQL uses JDBC for the database backends, but doesn't depend on any particular JDBC driver.

The approach was stolen from the YesQL library for Clojure, and adapted to be more idiomatic in Haskell. An example code snippet might look like this:

```
withTransaction db $ \conn -> do
  pageID:_ <- [yesh1|
    -- :: (Integer)
    -- :title:Text
    -- :body:Text
    INSERT INTO pages (title, body)
    VALUES (:title, :body)
```

```
RETURNING id
|]
  conn title body
[yesh1|
  -- :: Integer
  INSERT
  INTO page_owners (page_id, owner_id)
  VALUES (:pageID, :userID)
|]
  conn pageID currentUserID
return pageID
```

Contributions of any kind are more than welcome.

Further reading

- o <https://bitbucket.org/tdammers/yeshql>
- o <http://hackage.haskell.org/package/yeshql>
- o <https://github.com/krisajenkins/yesql> (not my work)

4.11 Data Structures, Data Types, Algorithms

4.11.1 Algebraic graphs

Report by:	Andrey Mokhov
Participants:	Arseniy Alekseyev, Neil Mitchell
Status:	usable, active development

Alga is a library for algebraic manipulation of graphs in Haskell. The underlying theory is presented [here](#).

Main idea

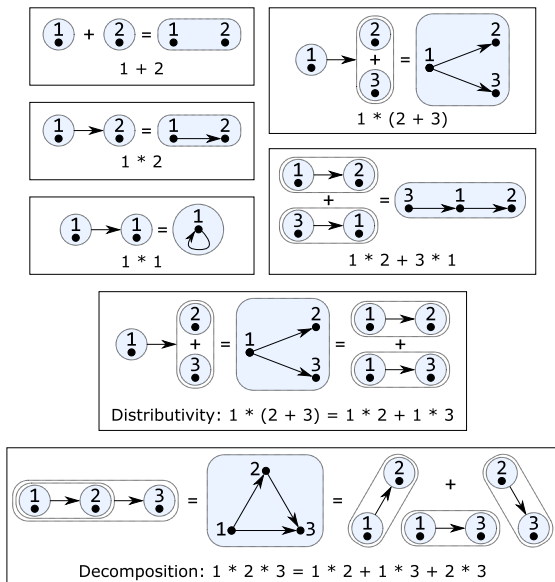
Consider the following data type, which is defined in the top-level module `Algebra.Graph` of the library:

```
data Graph a = Empty
  | Vertex a
  | Overlay (Graph a) (Graph a)
  | Connect (Graph a) (Graph a)
```

We can give the following semantics to the constructors in terms of the pair (V, E) of vertices and edges:

- o **Empty** constructs the empty graph (\emptyset, \emptyset) .
- o **Vertex** x constructs a single vertex, i.e. $(\{x\}, \emptyset)$.
- o **Overlay** $x y$ overlays graphs (V_x, E_x) and (V_y, E_y) constructing $(V_x \cup V_y, E_x \cup E_y)$.
- o **Connect** $x y$ connects graphs (V_x, E_x) and (V_y, E_y) constructing $(V_x \cup V_y, E_x \cup E_y \cup V_x \times V_y)$.

This figure shows examples of graph construction, where $+$ and $*$ stand for **Overlay** and **Connect**.



We can give an algebraic semantics to the graph construction primitives by defining the type class:

```
class Graph g where
  type Vertex g
  empty :: g
  vertex :: Vertex g -> g
  overlay :: g -> g -> g
  connect :: g -> g -> g
```

Instances of the type class obey the following laws:

- $(+, \text{empty})$ is an idempotent commutative monoid.
- $(*, \text{empty})$ is a monoid.
- $*$ distributes over $+$, e.g. $x * (y + z) = x * y + x * z$.
- $*$ can be *decomposed*: $x * y * z = x * y + x * z + y * z$.

This algebraic structure corresponds to unlabelled directed graphs: every expression represents a graph, and every graph can be represented by an expression. The library defines several law-abiding instances and polymorphic graph manipulation functions.

Current status

The library is documented, tested and usable. It is under active development, so the API is subject to change.

Further reading

- <http://hackage.haskell.org/package/algebraic-graphs>
- <https://github.com/snowleopard/alga-paper>

4.11.2 Conduit

Report by:
Status:

Michael Snoyman
stable

The conduit package is one of the most popular approaches to solving the streaming data problem in Haskell. It provides a composable, resource-safe, and constant memory solution to many common problems. With the well developed conduit ecosystem around it, you can easily deal with a variety of data sources (files, memory, HTTP, TCP), file formats (XML, YAML, JSON, CSV), advanced abstractions around parallel processing and concurrency, and have access to a wide range of helpful library functions to choose from.

Since the last HCAR, conduit has remained in an API stable mode. There are some tentative plans for a future breaking change which will standardize operator, function, and type naming around the recent reskinning described in <http://www.snoyman.com/blog/2016/09/proposed-conduit-reskin>. However, we strongly want to do this in a way that minimizes API breakage, so are taking such a change slowly.

Under the surface, conduit makes use of coroutines and an inlined free monad transformer approach to allow for a high level of flexibility in conduit composition. This is as opposed to other approaches, like list-t or stream fusion, which trade in some of that flexibility for performance. Some of these ideas were explored in <http://www.yesodweb.com/blog/2016/02/first-class-stream-fusion>. The end result is that, for most I/O based applications, conduit provides a great trade-off. For fully CPU-bound operations, you'll likely want to consider using something less flexible but higher performance.

Conduit is intended to be a replacement to usage of lazy I/O in Haskell code, allowing us to work on large data sets, ensure resources are cleaned up promptly, and retain composable programs. Please see the aforementioned tutorial for many examples of how this works.

The conduit package is designed to work well with the resourcet package, which allows for guaranteeing resource finalization in continuation-based monads. This is one of the main simplifications that conduit achieved versus previous streaming approaches, such as the enumerator package and other left-fold iterator approaches.

Since its initial release, conduit has been through many design iterations, all the while keeping to its initial core principles. The conduit API has remained stable on version 1.2, which includes a lot of work around performance optimizations, including a stream fusion implementation to allow much more optimized runs for some forms of pipelines, and the codensity transform

to provide better behavior of monadic bind.

Additionally, much work has gone into `conduit-combinators` and `streaming-commons`. The former provides a "batteries included" approach to conduit, containing a wide array of common functionality for both chunked data (like `ByteString`, `Text`, and `Vector`) and unchunked data. The latter contains common functionality useful to most streaming data frameworks, made available so that other libraries in this solution space can share a common code base.

There is a rich ecosystem of libraries available to be used with conduit, including cryptography, network communications, serialization, XML processing, and more.

Many conduit libraries are available via Hackage, Stackage Nightly, and LTS Haskell (just search for the word conduit). The main repository includes a tutorial on using the package.

Conduit includes a stream fusion framework for optimizing away intermediate data creation. Unfortunately, these optimizations do not always trigger due to problems with rewrite rule firing. There is some consideration around making this stream fusion a first-class entity that developers can rely upon. See <http://www.yesodweb.com/blog/2016/02/first-class-stream-fusion> and <https://github.com/snoyberg/foreach>.

Further reading

- <https://haskell-lang.org/library/conduit>
- <https://github.com/snoyberg/conduit#readme>
- <https://www.stackage.org/package/conduit>
- <https://www.stackage.org/package/conduit-combinators>
- <http://hackage.haskell.org/packages/archive/pkg-list.html#cat:conduit>

4.11.3 Transactional Trie

Report by:	Michael Schröder
Status:	stable

The transactional trie is a contention-free hash map for Software Transactional Memory (STM). It is based on the lock-free concurrent hash trie.

“Contention-free” means that it will never cause spurious conflicts between STM transactions operating on different elements of the map at the same time. Compared to simply putting a `HashMap` into a `TVar`, it is up to 8x faster and uses 10x less memory.

Further reading

- <http://hackage.haskell.org/package/ttrie>
- <http://github.com/mcschroeder/thesis>, in particular chapter 3, which includes a detailed discussion of the transactional trie’s design and implementation, its limitations, and an evaluation of its performance.

4.11.4 Concurrent Trie

Report by:	Michael Schröder
Status:	stable

A non-blocking concurrent map implementation based on the *lock-free concurrent hash trie* (aka *Ctrie*).

A hash trie is a tree whose leaves store key-value bindings and whose nodes are implemented as arrays. The concurrent trie extends the hash trie by adding indirection nodes above every array node. Indirection nodes have the property that they stay in the trie even if the nodes above or below them change. When inserting an element into the trie, instead of directly modifying an array node, an updated copy of the array node is created and an atomic compare-and-swap operation on the indirection node is used to switch out the old array node for the new one. If the compare-and-swap operation fails, the insert is retried from the beginning. This simple scheme, where indirection nodes act as barriers for concurrent modification, ensures that there are no lost updates or race conditions of any kind, while keeping all operations completely lock-free.

A more thorough discussion, including proofs of linearizability and lock-freedom, can be found in the papers by Prokopec et al.

Further reading

- <http://hackage.haskell.org/package/ctrie>
- "Cache-Aware Lock-Free Concurrent Hash Tries", Aleksander Prokopec, Phil Bagwell, Martin Odersky
- "Concurrent Tries with Efficient Non-Blocking Snapshots", Aleksander Prokopec, Nathan G. Bronson, Phil Bagwell, Martin Odersky

4.11.5 Random access zipper

Report by:	Li-yao Xia
Status:	Experimental, work in progress

The Random Access Zipper (RAZ) is a data structure for representing sequences with efficient indexing and edits.

The paper introducing it (with an implementation in OCaml) reported performance that is competitive with the more common Finger Tree structure.

I have translated it in Haskell, and started implementing the same interface as `Data.Sequence` from containers in `Data.Raz.Sequence`.

I reproduced the benchmarks from the paper as well as those of `containers`. On average, Haskell’s `raz` is slightly slower than OCaml’s, but faster than `containers` for many operations.

Future work

The `Data.Raz.Sequence` module remains to be finished to fully match `Data.Sequence`.

There are certainly lots of optimizations opportunities.

Raz requires randomness, which results in some awkward types in Haskell on the one hand (or, internal (ab)use of `unsafePerformIO` to implement a pure interface), but this explicitness can be informative on the other hand, though I am not yet certain how that information may be usefully exploited.

Further reading

- Random Access Zippers: Simple, Purely-Functional Sequences, K. Headley, M. A. Hammer.
<https://arxiv.org/abs/1608.06009>
- <https://hackage.haskell.org/package/raz>
- <https://github.com/Lysxia/raz.haskell>

4.11.6 Generic random generators

Report by:	Li-yao Xia
Status:	Experimental, active development

Description

The generic-random library automatically derives random generators for most datatypes. It can be used in testing for example, in particular to define instances of `QuickCheck's Arbitrary`.

The module `Generic.Random.Generic` leverages `GHC.Generics` to handle common boilerplate in instances of `Arbitrary` for simple datatypes.

However, for recursive datatypes, a naive generator is likely to have problematic issues: non-termination, inconveniently biased distributions (too large, too small, too full). `Generic.Random.Data` derives Boltzmann samplers, introduced by Duchon et al. (2004). They produce finite values of a given type and about a given size (the number of constructors) in linear time; the distribution is uniform when conditioned to a fixed size: two values with the same size occur with the same probability. An implementation can now be found in the `boltzmann-samplers` library (it has been split out of `generic-random`).

Status

Slowness of Boltzmann samplers I found out that the FEAT library, which can derive random generators for the same class of datatypes producing values of a given size exactly and uniformly distributed, has *much* better performance as well.

A more detailed explanation can be found at <https://github.com/Lysxia/generic-random/issues/6>.

In theory, Boltzmann samplers have the better asymptotic complexity, but they come with an overhead that appears hard to get rid of; `boltzmann-samplers` only catches up to FEAT on data sizes that seem too large to be practical (thousands of constructors).

Due to that, I have lost some motivation to go forward with `boltzmann-samplers`. I still remain open to discussion and suggestions.

Generic and applicative interface In spite of the above issues, I've started a rework of `boltzmann-samplers` using `GHC.Generics` rather than SYB, and a more general way to obtain generators from an applicative specification.

The branch can currently be found at: <https://github.com/Lysxia/boltzmann-samplers/tree/generics>.

Further reading

- Boltzmann Samplers for the Random Generation of Combinatorial Structures P. Duchon, P. Flajolet, G. Louchard, G. Schaeffer.
<http://algo.inria.fr/flajolet/Publications/DuFILoSc04.pdf>
- <https://hackage.haskell.org/package/generic-random>
- <https://github.com/Lysxia/generic-random>
- <https://hackage.haskell.org/package/testing-feat>

4.11.7 Generalized Algebraic Dynamic Programming

Report by:	Christian Höner zu Siederdisen
Participants:	Sarah J. Berkemer
Status:	usable, active development

Generalized Algebraic Dynamic Programming (gADP) provides a solution for high-level dynamic programs. We treat the formal grammars underlying each DP algorithm as an algebraic object which allows us to *calculate* with them. gADP covers dynamic programming problems of various kinds: (i) we include linear, context-free, and multiple context-free languages (ii) over sequences, trees, and sets; and (iii) provide abstract algebras to combine grammars in novel ways.

Below, we describe the highlights our system offers in more detail:

Grammars Products

We have developed a theory of algebraic operations over linear and context-free grammars. This theory allows us to combine simple “atomic” grammars to create more complex ones.

With the compiler that accompanies our theory, we make it easy to experiment with grammars and their products. Atomic grammars are user-defined and the algebraic operations on the atomic grammars are embedded in a rigorous mathematical framework.

Our immediate applications are problems in computational biology and linguistics. In these domains, algorithms that combine structural features on individual inputs (or tapes) with an alignment or structure between tapes are becoming more commonplace. Our

ADPfusion and the generalized Algebraic Dynamic Programming methodology have been extended to handle these kinds of grammars.

Further reading

- <http://www.bioinf.uni-leipzig.de/Software/gADP/>
- <http://dx.doi.org/10.1109/TCBB.2014.2326155>
- http://dx.doi.org/10.1007/978-3-319-12418-6_8

4.11.8 Earley

Report by:	Olle Fredriksson
Participants:	Spiros Boosalis, Oleg Grenrus, Tero Keinänen
Status:	maintained

Earley is a parsing library that can parse all context-free grammars, including tricky ones for example with left-recursion. The grammars are specified in applicative style.

A new feature in the Earley library is language generation. Given a grammar and a list of allowed input tokens, Earley can generate the members of the language that the grammar generates. The following example shows the language generated by a Roman numerals grammar limited to the tokens 'V', 'I', and 'X'.

```
language (generator romanNumeralsGrammar "VIX")
= [(0, ""), (1, "I"), (5, "V"), (10, "X"), (20, "XX"),
  (11, "XI"), (15, "XV"), (6, "VI"), (9, "IX"),
  (4, "IV"), (2, "II"), (3, "III"), (19, "XIX"),
  (16, "XVI"), (14, "XIV"), (12, "XII"), (7, "VII"),
  (21, "XXI"), (25, "XXV"), (30, "XXX"),
  (31, "XXXI"), (35, "XXXV"), (8, "VIII"),
  (13, "XIII"), (17, "XVII"), (26, "XXVI"),
  (29, "XXIX"), (24, "XXIV"), (22, "XXII"),
  (18, "XVIII"), (36, "XXXVI"), (39, "XXXIX"),
  (34, "XXXIV"), (32, "XXXII"), (23, "XXIII"),
  (27, "XXVII"), (33, "XXXIII"), (28, "XXVIII"),
  (37, "XXXVII"), (38, "XXXVIII")]
```

Further reading

<https://github.com/ollef/Earley>

4.11.9 Transient

Report by:	Alberto Gómez Corona
Status:	active development

Transient is a monad/applicative/Alternative with batteries included that brings the power of high level effects in order to reduce the learning curve and make the Haskell programmer productive. Effects include event handling/reactive, backtracking, extensible state, indeterminism, concurrency, parallelism, thread control and distributed computing, publish/subscribe and client/server side web programming among others.

All effects can be combined while maintaining algebraic and monadic composability using standard applicative, alternative and monadic combinators.

What is new in this report is:

- Restoring execution state from checkpoint
- Nodes can connect using websockets or relay communications
- Secure communications with TLS
- Optimize local calls
- Exception management using backtracking
- HTML rendering support templates and template edition !!??

Future work: Relay communications, Programmer-defined serialization, Server side HTML rendering

Further reading

- [gitter chat](#)
- [Transient tutorial](#)
- [distributed Transient, GIT repository](#)
- [Transient GIT repository](#)
- [An EDSL for Hard-working IT programmers](#)
- [The hardworking programmer II: practical backtracking to undo actions](#)
- [Publish-suscribe variables](#)
- [Moving processes between nodes](#)
- [Parallel non-determinism](#)
- [streaming, distributed streaming, mapReduce with distributed datasets](#)

4.12 Parallelism

4.12.1 Eden

Report by:	Jost Berthold and Rita Loogen
Participants:	in Madrid: Yolanda Ortega-Mallén, Mercedes Hidalgo, Lidia Sánchez-Gil, Fernando Rubio, Alberto de la Encina, in Marburg: Florian Fey, Rita Loogen, Lukas Schiller, in Sydney: Jost Berthold
Status:	ongoing



Eden extends Haskell with a small set of syntactic constructs for explicit process specification and creation, including automatic process communication (via head-strict lazy lists) and synchronization. Higher-level coordination is achieved by defining *skeletons*, ranging from a simple parallel map to sophisticated master-worker schemes.

Eden's interface supports a simple definition of arbitrary communication topologies using *Remote Data*.

The remote data concept can also be used to compose skeletons in an elegant and effective way, especially in distributed settings. A *PA-monad* enables the *eager* execution of user defined sequences of *Parallel Actions* in Eden.

Survey and standard reference: Rita Loogen, Yolanda Ortega-Mallén, and Ricardo Peña: *Parallel Functional Programming in Eden*, Journal of Functional Programming 15(3), 2005, pages 431–475.

Tutorial: Rita Loogen: *Eden - Parallel Functional Programming in Haskell*, in: V. Zsóok, Z. Horváth, and R. Plasmeijer (Eds.): CFP 2011, Springer LNCS 7241, 2012, pp. 142-206. (see also: www.mathematik.uni-marburg.de/~eden/?content=cefp)

Implementation

Eden is currently implemented by modifications to the Glasgow-Haskell Compiler (extending its runtime system to use multiple communicating instances). Apart from MPI or PVM in cluster environments, Eden supports a shared memory mode on multicore platforms, which uses multiple independent heaps but does not depend on any middleware. Building on this runtime support, the Haskell package *edenmodules* defines the language, and *edenskels* provides a library of parallel skeletons.

As an alternative, a library implementation of Eden was developed in a recent master thesis by Florian Fey. The library implementation uses the *network-fancy* library for networking, and the *packman* library for serialising Haskell heap objects.

Source code for Eden is available from <http://hex.mathematik.uni-marburg.de:8080/>, and <http://github.com/jberthold/ghc>. The Eden libraries (Haskell-level) are also available via Hackage.

The available source code works with recent GHC versions, but due to other commitments and limited time, no packaged releases were prepared for GHC-8.0 and GHC-8.2 and libraries. We will be happy to provide support, and delighted to receive contributions — please contact us.

Tools and libraries

The Eden trace viewer tool *EdenTV* provides a visualisation of Eden program runs on various levels. Activity profiles are produced for processing elements (machines), Eden processes and threads. In addition message transfer can be shown between processes and machines. *EdenTV* is written in Haskell and is freely available on the Eden web pages and on hackage. Eden's thread view can also be used to visualise ghc eventlogs. Another trace viewer tool, *Eden-Tracelab*, has been developed by Bastian Reitemeier.

It is capable of visualising large trace files, without being constrained by the available memory, see brtmr.de/2015/10/17/introducing-eden-tracelab.html.

The Eden skeleton library contains various skeletons for parallel processing, including parallel maps, workpools, divide-and-conquer, and common process topologies. Take a look on the Eden pages or the had-dock documentation on hackage for an overview.

Further reading

<http://www.mathematik.uni-marburg.de/~eden>

4.12.2 Auto-parallelizing Pure Functional Language System

Report by:	Kei Davis
Participants:	Dean Prichard, David Ringo, Loren Anderson, Jacob Marks
Status:	active

The main project goal is the demonstration of a lightweight, higher-order, polymorphic, pure functional language implementation in which we can experiment with automatic parallelization strategies, varying degrees of default function and constructor strictness, and lightweight instrumentation.

We do not consider speculative or eager evaluation, but do plan to infer strictness by program analysis, so potential parallelism is dictated by the specified degree of default strictness, language constructs for parallelism, and program analysis.

Our approach is similar to that of the [Intel Labs Haskell Research Compiler](#): we use GHC as a front-end to generate STG, then exit to our own back-end compiler; additionally, we have native *Mini-Haskell* and STG front-ends. As in their case we do not attempt to use the GHC runtime. Our implementation is *lightweight* in that we are not attempting to support or recreate the vast functionality of GHC and its runtime. This approach is also similar to [Don Stewart's](#) except that we generate C instead of Java.

Current Status

Currently we have a fully functioning serial implementation and a primitive proof-of-design parallel implementation. The most recent major development was the bridge between GHC and our system. Thus we can now compile and run Haskell programs with simple primitive and algebraic data types using GHC or our own mini-Haskell front-end.

Additionally, we have developed a new strictness analysis technique that is currently being implemented.

Immediate Plans

We are currently developing a more realistic parallel runtime, and writing up and implementing the new strictness analysis technique.

Undergraduate/post-graduate Internships

If you are a United States citizen or permanent resident alien studying computer science or mathematics at the undergraduate level, or are a recent graduate, with strong interests in Haskell programming, compiler/runtime development, and pursuing a spring, fall, or summer internship at Los Alamos National Laboratory, USA, this could be for you.

We don't expect applicants to necessarily already be highly accomplished Haskell programmers—such an internship is expected to be a combination of further developing your programming/Haskell skills and putting them to good use. If you're already a strong C hacker we could use that too.

The application process requires a bit of work so don't leave enquiries until the last day/month. Dates for terms beyond summer 2018 are best guesses based on prior years.

Term	Application Opening	Deadline
Summer 2018	OPEN NOW	Jan 2018
Fall 2018	Jan 2018	May 31 2018
Spring 2019	July 2018	Oct 2018

Email kei (at) lanl (dot) gov if interested in more information, and feel free to pass this along.

Further reading

Email same address as above for the Trends in Functional Programming 2016 paper about this project.

Intern Loren Anderson did an interesting Haskell exercise while here for this [mathematics paper](#).

4.12.3 Déjà Fu: Concurrency Testing

Report by: Michael Walker
Status: actively developed

Déjà Fu is a concurrency testing tool for Haskell built on top of the “concurrency” library, which provides a typeclass abstraction over a large subset of the “base” concurrency API, such as:

- Threading
- Capabilities
- Yielding and delaying
- IORefs, MVars, and Software Transactional Memory
- Relaxed memory for IORef operations

- Atomic IORef primitives
- Exceptions

In the last six months, `dejaFu` has gained a new module for property-testing stateful properties, improved reporting of uncaught exceptions, and numerous bug fixes and performance improvements.

A new 1.0.0.0 release is actively being developed which will bring with it support for bound threads, many small API improvements, and significant performance improvements in the normal case.

Further reading

- <http://hackage.haskell.org/package/dejaFu>
- <http://hackage.haskell.org/package/concurrency>
- The 2015 Haskell Symposium paper is available at <http://bit.ly/1N2Lkw4>; and a more up-to-date technical report is available at <http://bit.ly/1SMHx4U>.
- There are a number of blog posts on the functionality and implementation at <https://www.barrucadu.co.uk>.

4.12.4 The Remote Monad Design Pattern

Report by: Andrew Gill
Participants: Justin Dawson, Mark Grebe, James Stanton, David Young
Status: active

The **remote monad design pattern** is a way of making Remote Procedure Calls (RPCs), and other calls that leave the Haskell eco-system, considerably less expensive. The idea is that, rather than directly call a remote procedure, we instead give the remote procedure call a service-specific monadic type, and invoke the remote procedure call using a monadic “send” function. Specifically, a **remote monad** is a monad that has its evaluation function in a remote location, outside the local runtime system.

By factoring the RPC into sending invocation and service name, we can group together procedure calls, and amortize the cost of the remote call. To give an example, Blank Canvas, our library for remotely accessing the JavaScript HTML5 Canvas, has a `send` function, `lineWidth` and `strokeStyle` services, and our remote monad is called `Canvas`:

```
send      :: Device -> Canvas a -> IO a
lineWidth :: Double   -> Canvas ()
strokeStyle :: Text    -> Canvas ()
```

If we wanted to change the (remote) line width, the `lineWidth` RPC can be invoked by combining `send` and `lineWidth`:

```
send device (lineWidth 10)
```

Likewise, if we wanted to change the (remote) stroke color, the `strokeStyle` RPC can be invoked by combining `send` and `strokeStyle`:

```
send device (strokeStyle "red")
```

The key idea is that remote monadic commands can be locally combined before sending them to a remote server. For example:

```
send device (lineWidth 10 >> strokeStyle "red")
```

The complication is that, in general, monadic commands can return a result, which may be used by subsequent commands. For example, if we add a monadic command that returns a Boolean,

```
isPointInPath :: (Double,Double) -> Canvas Bool
```

we could use the result as follows:

```
send device $ do
  inside <- isPointInPath (0,0)
  lineWidth (if inside then 10 else 2)
  ...
```

The invocation of `send` can also return a value:

```
do res <- send device (isPointInPath (0,0))
  ...
```

Thus, while the monadic commands inside `send` are executed in a remote location, the results of those executions need to be made available for use locally.

We had a paper in the 2015 Haskell Symposium that discusses these ideas in more detail, and more recently, we have improved the packet mechanism to include an analog of the applicative monad structure, allowing for even better bundling. We have also improved the error handling capabilities. These ideas are implemented up in the `hackage` package `remote-monad`, which captures the pattern, and automatically bundled the monadic requests.

Further reading

<http://ku-fpg.github.io/practice/remotemonad>

4.12.5 concurrent-output

Report by:	Joey Hess
Status:	stable, actively developed

A common problem with concurrent programs is that output to the console has to be buffered or otherwise dealt with to avoid multiple threads writing over top of one-another. This is particularly a problem for progress displays, and the output of external processes. The `concurrent-output` library aims to be a simple solution to this problem.

It includes support for multiple console regions, which different threads can update independently. Rather than the complexity of using a library such as `ncurses` to lay out the screen, `concurrent-output`'s regions are compositional; it acts as a kind of miniature

tiling window manager. This makes it easy to generate progress displays similar to those used by `apt` or `docker`.

`STM` is used extensively in the implementation, which simplified what would have otherwise been a mess of nested locks. This made concurrent-output extensible using `STM` transactions. See [this blog post](#).

Concurrent-output is used by `git-annex`, `propellor`, and `xdcc` and patches have been developed to make both `shake` and `stack` use it.

4.13 Systems programming

4.13.1 Haskell for Mobile development

Report by:	Moritz Angermann
Participants:	zw3rk.com and obsidian.systems
Status:	in review

The set of languages to choose from for mobile development is limited to those languages that can target those ecosystems.

There have been ongoing efforts to make Haskell a viable choice for Mobile Development for many years, via the path of cross compilation.

A major obstacle for cross compilation with Haskell is Template Haskell. Up until recently Template Haskell was available only for `stage2` compilers, while cross compilers are `stage1`. One notable exception is `GHCJS`, which had Template Haskell support quite some time already via an out of process Template Haskell solution. With the addition of `-fexternal-interpreter` via `iserv` in recent `GHCs`, which implements a very similar system, it is possible to add TH support to `stage1` cross compilers.

With the addition of linker for `mach-o/aarch64`, `elf/aarch64`, and improvements to the `elf/armv7` linker in `GHC`, as well as a proxy mechanism that allows `GHC` to communicate with a remote `iserv` on a different host it is now possible to compile large chunks of TH code with cross compilers targeting supported linker platforms. File and Process IO pose interesting problems.

The necessary changes to `GHC` are all under review and will hopefully make it into `GHC 8.4` in time.

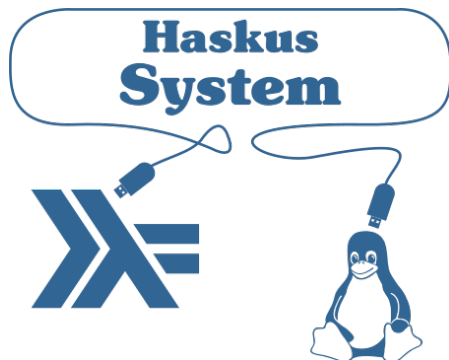
There are some additional challenges in the Haskell ecosystem (e.g. `cabal` is not very cross compilation aware) that need to be addressed before Haskell for Mobile works out of the box.

A periodically force-pushed snapshot of the `GHC HEAD` plus the open `differentials` is available, as well as `build scripts` that build `ghc` for `iOS` and `Android` using a custom `toolchain`, derived from the `ghc-ios-scripts`. `GHCSlave` apps for `iOS` and `Android` that wrap `iserv`.

Further information covering the ongoing development will be published at <https://medium.com/@zw3rk>.

4.13.2 haskus-system

Report by: Sylvain Henry
Status: active



`haskus-system` is a system programming framework directly on top of the Linux kernel system calls. It doesn't rely on usual interfaces (e.g., `libc`, `libdrm`, `libinput`, `X11`, `wayland`, etc.) to communicate with the kernel. Everything is done in Haskell.

Notable changes since the last HCAR:

- Building a system is much easier. We just have to declare the Linux version to use and some other options in a `system.yaml` file and a tool called `haskus-system-build` automatically downloads, builds and configures the dependencies (Linux, SysLinux).
- The same tool can also be used to build ISO images and to execute the system with QEMU.
- The user manual has been enhanced and a new demo is available showing a simple terminal-like application.

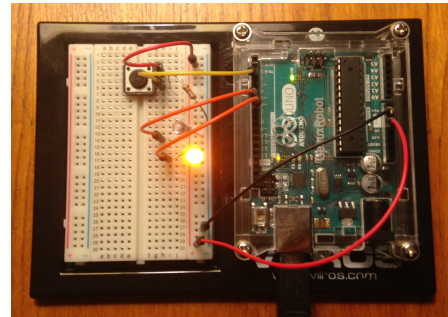
The source code is freely available (BSD3 license).

Further reading

www.haskus.org/system

4.13.3 Haskino

Report by: Andrew Gill
Participants: Mark Grebe
Status: active



Haskino is a Haskell development environment for programming the Arduino microcontroller boards in a high level functional language instead of the low level C language normally used.

This work started with Levent Erkök's `hArduino` package. The original version of Haskino, extended `hArduino` by applying the concepts of the strong remote monad design pattern to provide a more efficient way of communicating, and generalizing the controls over the remote execution. In addition, it added a deep embedding, control structures, an expression language, and a redesigned firmware interpreter to enable standalone software for the Arduino to be developed using the full power of Haskell.

The current version of Haskino continues to build on this work. Haskino is now able to directly generate C programs from our Arduino Monad. This allows the same monadic program to be quickly developed and prototyped with the interpreter, then compiled to C for more efficient operation. In addition, we have added scheduling capability with lightweight threads and semaphores for inter-thread synchronization.

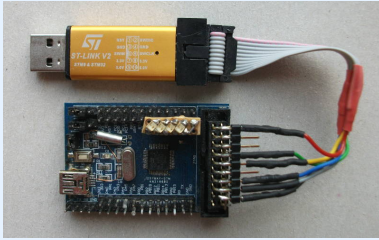
The development has been active over the past year. A paper was published at PADL 2016 for original version, and there is a paper accepted for presentation at TFP 2016 for the new scheduled and compiled version.

Further reading

- <https://github.com/ku-fpg/haskino>
- <https://github.com/ku-fpg/haskino/wiki>

4.13.4 STM32-Zombie

Report by: Marc Fontaine
Status: active



The *STM32-Zombie* project turns a STM32Fxxx micro controller into a Haskell programmable and flexible IO peripheral of your PC. The STM32Fxxx micro controller family features a variety of powerful IO peripherals like GPIO ports, USART, SPI, I2C, USB, ADC, timers, real time clock, etc. and *STM32-Zombie* allows a Haskell program to control the complete set of built-in micro controller peripherals.

The project is called *STM32-Zombie* because it shuts down the controllers brain (the ARM CPU) and turns it into a remote controlled zombie. It works without any c-code, cross-compiler tool-chain, or firmware. The STM32Fxxx peripherals use memory mapped control registers and the on-chip-debugging interface of the controller allows Haskell to access all of the controllers address space and registers. With help of the DMA controller even hard-real-time applications, like high-frequency sampling or generating of high-frequency output patterns, are possible.

Minimal hardware requirements, for trying out this project, are a mini STM32F103 breakout board and a STLink V2 USB dongle simulator. Both parts are available for less then \$2 each. My test setup are mini STM32F103 breakout boards. I have not tested the library with other members of the STM32Fxxx family but the library is probably a good starting point for work on other STM32Fxxx controllers.

The cabal package contains examples for LED blinking, serial port, SPI, high-frequency ADC sampling, control of WS1228B RGB LED strips, and more.

The library uses the naming conventions of the *STM32F10x Firmware Library* and provides a mid-level abstraction of the hardware.

While access to all of the peripherals is possible, the abstraction layer is still work-in-progress and does not cover all of the peripherals yet.

Comments, suggestions, experience reports, and patches are welcome. The project is open to any kind of contribution.

Further reading

- <https://github.com/MarcFontaine/stm32hs>
- <http://hackage.haskell.org/package/STM32-Zombie>

4.13.5 Feldspar

Report by: Emil Axelsson
Status: active development

Feldspar is a domain-specific language for digital signal processing (DSP). The language is embedded in Haskell and has been developed as a collaboration, in different phases, between Chalmers University of Technology, ELTE University, SICS Swedish ICT AB and Ericsson AB.

The motivating application of Feldspar is telecoms processing, but the language is intended to be useful for DSP and numeric code in general as well as for programming embedded systems. The aim is to allow functions to be written in functional style in order to raise the abstraction level of the code and to enable more high-level optimizations.

The currently recommended Feldspar implementation is RAW-Feldspar. Its README file is a good starting point for getting to know Feldspar.

RAW-Feldspar provides libraries for numeric and array processing operations, and supports file handling, calls to external C libraries, concurrency, etc. It also comes with a code generator producing C code for running on embedded targets.

For reference, the original Feldspar implementation is available in the packages

- `feldspar-language` – language front end
- `feldspar-compiler` – C back end

Ongoing work involves using RAW-Feldspar to implement more high-level libraries for streaming and interactive programs. Two examples of such libraries are:

- `zeldspar` – a Ziria-like EDSL
- `feldspar-synch` – a synchronous data-flow library
- `raw-feldspar-mcs` is a library built on top of RAW-Feldspar that generates code for running on NUMA architectures such as the Parallella.

There is also ongoing work to support hardware/software co-design in RAW-Feldspar.

Further reading

- Official home page: <http://feldspar.github.io>

4.14 Mathematics, Simulations and High Performance Computing

4.14.1 sparse-linear-algebra

Report by: Marco Zocca
Participants:
Status: Actively developed

This library provides common numerical analysis functionality, without requiring any external bindings. It is not optimized for performance yet, but it serves as

an experimental platform for scientific computation in a purely functional setting.

Currently it offers :

- iterative linear solvers of the Krylov subspace type, e.g. variants of conjugate gradient such as Conjugate Gradient Squared, BiConjugate Gradient and BiCGSTAB
- linear eigensolvers, based on the QR algorithm and the Rayleigh iteration
- matrix factorizations (namely, LU and QR)
- a number of utility functions such vector and matrix norms, computation of the matrix condition number, Givens' rotation and Householder reflection matrices, and partitioning/stacking/reshaping operations.

The initial motivation for this was on one hand the lack of native Haskell tools for numerical computation, and on the other a curiosity to reimagine scientific computing through a functional lens.

The implementation relies on nested `IntMap`'s from `containers`.

Currently, a new backend based on `accelerate` is under development, along with a generalized interface based on typeclasses, which will allow to decouple algorithms from datastructures. Eventually, the `accelerate`-based backend and interface will be provided as two distinct packages, and `sparse-linear-algebra` will import from both and provide a new reference implementation.

A usage tutorial on the major functionality is available in the README file, and all interface functions are commented throughout the Haddock documentation.

`sparse-linear-algebra` is freely available on Hackage under the terms of a GPL-3 license; development is tracked on GitHub, and all suggestions and contributions are very much welcome.

Further reading

- <https://github.com/ocramz/sparse-linear-algebra>
- <https://hackage.haskell.org/package/sparse-linear-algebra>

4.14.2 aivika

Report by:	David Sorokin
Status:	stable

Aivika is a collection of open-source simulation libraries written in Haskell. It is mainly focused on discrete event simulation but has a partial support of system dynamics and agent-based modeling too.

A key idea is that many simulation activities can be modeled based on abstract computations such as monads, streams and arrows. The computations are composing units, which we can construct simulation models from and then run.

Aivika consists of a few packages. The basic package introduces the simulation computations. There are

other packages that allow automating simulation experiments. They can save the simulation results in files, plot charts and histograms, collect the statistics summary and so on. There are also packages for distributed parallel simulation and nested simulation based on the generalized version of Aivika.

The core of Aivika is quite stable and well-tested. The libraries work on Linux, OS X and Windows. They are licensed under BSD3 and available on Hackage.

There are plans to find new application fields for the libraries. The core libraries solve a very general task and definitely can be applied to other fields too.

Further reading

<http://hackage.haskell.org/package/aivika>

4.14.3 General Decimal Arithmetic

Report by:	Rob Leslie
Status:	experimental, active development

Haskell's `Float` and `Double` types are often used for floating-point arithmetic, but sometimes they should not be. Although the Haskell Language Report does not require it, these types are typically implemented using a binary floating-point representation, and consequently do not always have an exact correspondence with the way we write a value using decimal notation. For example, the value 0.1 cannot be represented exactly as a binary floating-point value, because there is no integer solution to make $c \times 2^q = 10^{-1}$.

This can lead to subtle errors in computations, as well as the attendant grief and frustration when our programs fail to meet user expectations. It is why Mike Cowlshaw, editor of the IEEE 754 standard for floating-point arithmetic, writes: "While suitable for many purposes, binary floating-point arithmetic should not be used for financial, commercial, and user-centric applications or web services because the decimal data used in these applications cannot be represented exactly using binary floating-point."

Mike Cowlshaw has an entire section of his website devoted to *General Decimal Arithmetic* (URL below), including a complete specification that describes the decimal arithmetic included in the updated IEEE 754-2008 standard. Decimal floating-point arithmetic works just like its binary cousin, except it uses a radix value of 10 internally instead of 2. This allows decimal floating-point values to be represented exactly; there is a one-to-one correspondence between the way a value is written in decimal notation and the way it is represented internally. Consequently there are fewer surprises when performing arithmetic with such values.

The *decimal-arithmetic* package is being developed to provide a Haskell implementation of the *General Decimal Arithmetic* specification. It offers a versatile `Decimal` type constructor that is parameterized with

both a precision and a rounding algorithm; all arithmetic using this type will be restricted to the given precision, using the given rounding algorithm whenever the result of a computation exceeds the precision.

Several type aliases are provided as a convenience. For example, `Decimal64` is a decimal floating-point type with 16 digits of precision (comparable to `Double`) that rounds half even. It is special because it also has a `Binary` instance with a 64-bit encoded representation using the *decimal64* interchange format. Similar types `Decimal32` (cf. `Float`) and `Decimal128` are also provided.

All the usual numeric, fractional, real, and floating-point type classes have been fully implemented, allowing decimal types to be used as drop-in substitutes for `Float` and `Double`. In addition, a low-level arithmetic monad is provided within which primitive operations can be performed with complete control over the outcome of exceptional conditions such as inexact results or division by zero.

The package is currently in a usable state; extensive testing is encouraged. While the package includes a moderate test suite, future work will focus on ensuring the implementation passes all of the test cases provided on Mike Cowlshaw's website.

Further reading

- <http://speleotrove.com/decimal/>
- <https://hackage.haskell.org/package/decimal-arithmetic>

4.15 Graphical User Interfaces

4.15.1 wxHaskell

Report by:	Henk-Jan van Tuyl
Status:	active development



wxHaskell 0.93.0.0 development is in progress, with, amongst others, an adaptation to Cabal 2.0. A `cabal.project` file is added to the GitHub repository and the test sets are converted to Cabal packages, to be able to compile everything with the experimental command `"cabal new-build all"`. Preparations have been made for the wxWidgets 3.1 binding, but more work is needed for this.

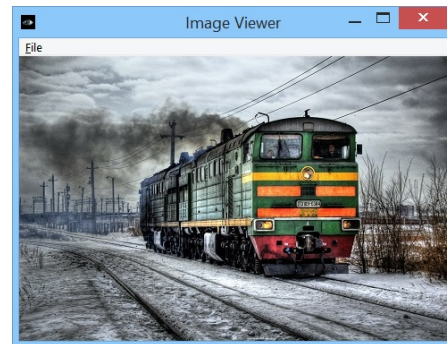
New project participants are welcome.

wxHaskell is a portable and native GUI library for Haskell. The goal of the project is to provide an indus-

trial strength GUI library for Haskell, but without the burden of developing (and maintaining) one ourselves.

wxHaskell is therefore built on top of wxWidgets: a comprehensive C++ library that is portable across all major GUI platforms; including GTK, Windows, X11, and MacOS X. Furthermore, it is a mature library (in development since 1992) that supports a wide range of widgets with the native look-and-feel.

A screen printout of a sample wxHaskell program:



Further reading

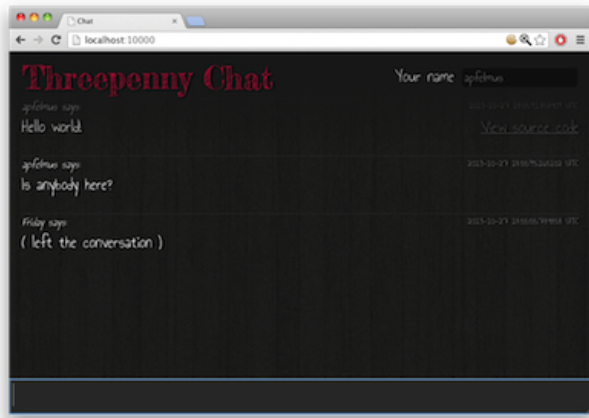
<https://wiki.haskell.org/WxHaskell>

4.15.2 threepenny-gui

Report by:	Heinrich Apfelmus
Status:	active development

Threepenny-gui is a framework for writing graphical user interfaces (GUI) that uses the web browser as a display. Features include:

- *Easy installation.* Everyone has a reasonably modern web browser installed. Just install the library from Hackage and you are ready to go. The library is cross-platform.
- *HTML + JavaScript.* You have all capabilities of HTML at your disposal when creating user interfaces. This is a blessing, but it can also be a curse, so the library includes a few layout combinators to quickly create user interfaces without the need to deal with the mess that is CSS. A foreign function interface (FFI) allows you to execute JavaScript code in the browser.
- *Functional Reactive Programming (FRP)* promises to eliminate the spaghetti code that you usually get when using the traditional imperative style for programming user interactions. Threepenny has an FRP library built-in, but its use is completely optional. Employ FRP when it is convenient and fall back to the traditional style when you hit an impasse. You can download the library from Hackage or Stackage and use it right away to write that cheap GUI you need for your project. Here a screenshot from the example code:



For a collection of real world applications that use the library, have a look at the gallery on the homepage.

Status

The latest release is version 0.8.2.0. I would like to thank co-maintainer Simon Jakobi for his diligent and timely releases.

Compared to the previous report, several small additions to the API have been made, the documentation has been improved, and compatibility with the latest GHC versions has been ensured. Moreover, I given a tutorial about the library on HaL 2017, and the slides are now available in the source repository.

Future development

The library is still in flux, API changes are likely in future versions.

In the future, I hope to improve the functional reactive programming (FRP) aspects of the framework, for instance by using the `reactive-banana` library as a basis. (→ 4.16.2)

Further reading

- Project homepage:
<http://wiki.haskell.org/Threepenny-gui>
- Example code:
<https://github.com/HeinrichApfelmus/threepenny-gui/tree/master/samples#readme>
- Application gallery:
<http://wiki.haskell.org/Threepenny-gui#Gallery>

4.16 FRP

4.16.1 Yampa

Report by:

Ivan Perez

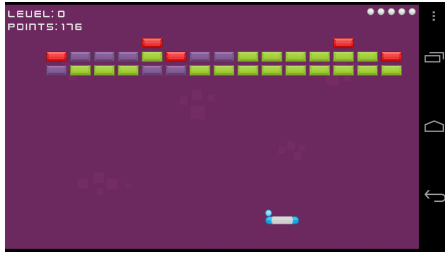
Yampa (Github: <http://git.io/vTvxQ>, Hackage: <http://goo.gl/JGwycF>), is a Functional Reactive Programming implementation in the form of a EDSL to define *Signal Functions*, that is, transformations of input signals into output signals (aka. *behaviours* in other FRP dialects).

Yampa systems are defined as combinations of Signal Functions. Yampa includes combinators to create constant signals, apply pointwise (or time-wise) transformations, access the running time, introduce delays and create loopbacks (carrying present output as future input). Systems can be dynamic: their structure can be changed using *switching* combinators, which apply a different signal function at some point in the future. Combinators that deal with collections enable adding, removing, altering, pausing and unpausing signal functions at will.

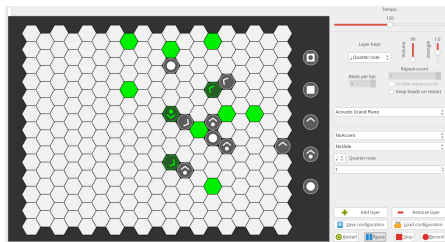
A suitable thinking model for FRP in Yampa is that of signal processing, in which components (signal functions) transform signals based on their present value and a component's internal state. Components can, therefore, be serialized, applied in parallel, etc. Yampa's signal functions implement the Arrow and ArrowLoop typeclasses, making it possible to use both arrow notation and arrow combinators.

Yampa combinators guarantee *causality*: the value of an output signal at a time t can only depend on values of input signals at times $[0, t]$. Efficiency is provided by limiting history only to the immediate past, and letting signals functions explicitly carry *state* for the future. Unlike other implementations of FRP, Yampa enforces a strict separation of effects and pure transformations: all IO code must exist outside Signal Functions, making systems easier to reason about and debug.

Yampa has been used to create both free/open-source and commercial games. Examples of the former include Frag (<http://goo.gl/8bfSmz>), a basic reimplement of the Quake III Arena engine in Haskell, and Haskanoid (<http://git.io/v8eq3>), an arkanoid game featuring SDL graphics and sound with Wiimote & Kinect support, which works on Windows, Linux, Mac, Android, iOS and web browsers (thanks to GHCJS). Examples of the latter include Keera Studios' Magic Cookies!, a Haskell puzzle board game for iOS and Android available on iTunes (<https://goo.gl/6gB6sb>) and Google Play (<https://goo.gl/0A8z6i>).



Guerric Chupin (ENSTA ParisTech), under the supervision of Henrik Nilsson (Functional Programming Lab, University of Nottingham ([→ 6.6](#))) has developed Arpeggigon ([→ 4.17.4](#)) (<https://gitlab.com/chupin/arpeggigon>), an interactive cellular automaton for composing groove-based music. The aim was to evaluate two reactive but complementary frameworks for implementing interactive time-aware applications. Arpeggigon uses Yampa for music generation, Gtk2HS for Graphical User Interface, jack for handling MIDI I/O, and Keera Hails to implement a declarative MVC architecture, based on *Reactive Values and Relations* (RVRs). The results have been written up in an application paper, *Funky Grooves: Declarative Programming of Full-Fledged Musical Applications*, presented at PADL 2017. The code and an extended version of the paper are publicly available (<https://gitlab.com/chupin/arpeggigon>). Arpeggigon has also been demonstrated at FARM 2017, the Haskell eXchange 2017, and Haskell in Leipzig 2017.



Yampa is under active development, with many Haskellers participating on github and sending their contributions. The last few versions of Yampa have featured a cleaner API, better documentation and new Signal Function combinators. Our github repository includes development branches with features that have been used to extend Yampa for custom games. Some of these features have been presented in the paper “Back to the Future: time travel in FRP”, presented at the Haskell Symposium 2017, and will be included in future versions. Yampa will now be extended with testing and debugging features, which allow us to express temporal assertions about FRP systems using Temporal Logic, and to use QuickCheck to test those properties, as described in the paper “Testing and Debugging Functional Reactive Programming”, presented at ICFP 2017.

Extensions to Arrowized Functional Reactive Programming are an active research topic. Last year

we published, together with Manuel Bärenz, a monadic arrowized reactive framework called Dunai (<https://git.io/vXsw1>), and a minimal FRP implementation called BearRiver. BearRiver provides all the core features of Yampa, as well as additional extensions. We have demonstrated the usefulness of our approach and the compatibility with existing Yampa games by using BearRiver to compile and execute the Haskanoid and Magic Cookies! for Android without changing the code of such games. These games are also available for iOS and other platforms.

The Functional Programming Laboratory at the University of Nottingham ([→ 6.6](#)) is working on other extensions to make Yampa more general and modular, increase performance, enable new use cases and address existing limitations. To collaborate with our research, please contact Ivan Perez () and Henrik Nilsson ().

We encourage all Haskellers to participate on Yampa’s development by opening issues on our Github page (<http://git.io/vTvXQ>), adding improvements, creating tutorials and examples, and using Yampa in their next amazing Haskell games. We thank the kind users who have already sent us their contributions.

4.16.2 reactive-banana

Report by:	Heinrich Apfelmus
Status:	active development



Reactive-banana is a library for functional reactive programming (FRP). FRP offers an elegant and concise way to express interactive programs such as graphical user interfaces, animations, computer music or robot controllers. It promises to avoid the spaghetti code that is all too common in traditional approaches to GUI programming.

The goal of the library is to provide a solid foundation.

- Programmers interested in implementing FRP will have a *reference* for a *simple semantics* with a working implementation. The library stays close to the semantics pioneered by Conal Elliott.
- The library features an *efficient implementation*. No more spooky time leaks, predicting space & time usage should be straightforward.

The library is meant to be used in conjunction with existing libraries that are specific to your problem domain. For instance, you can hook it into any event-based GUI framework, like wxHaskell or Gtk2HS. Several helper packages like reactive-banana-wx provide a small amount of glue code that can make life easier.

Status

The latest release is version 1.1.0.1. Thanks to the efforts of our co-maintainer Oliver Charles, the library is available on Stackage. (→ 4.2.3) The library API is stable and feature complete.

Future development

There still remains some work to be done to improve the constant factor performance of the library. Also, the library does not yet compile well to JavaScript with GHC/JS, as there are some issues with garbage collection.

Several other improvements have been suggested on the issue tracker, most notably renaming the `Event` type to `Events`, as the plural provides a more apt description of the semantics of this type.

Further reading

- Project homepage:
<http://wiki.haskell.org/Reactive-banana>
- Example code:
<http://wiki.haskell.org/Reactive-banana/Examples>

It is of most importance to us to keep our code pure - except from the reactive Yampa-loop all our code is pure and does not make use of the IO-Monad.

As use-cases we implemented a number of various well known ABS models, some of which have time-semantics like timed transitions or recurring events according to an exponential random-distribution and others which have none. For the former one we implemented an EDSL on top of Yampas reactive primitives and for non time-semantic models we provide a monadic version of our low-level EDSL which allows implementing ABS in a very imperative style.

Further research will go into validation & verification of ABS in our library using QuickCheck and algebraic reasoning.

The code is freely available but not stable as it currently serves for prototyping for gaining insights into the problems faced when implementing ABS in Haskell. The plan is to release the final implementation at the end of the PhD as a stable and full-featured library on Hackage.

If you are interested in the on-going research please contact Jonathan Thaler (jonathan.thaler@nottingham.ac.uk).

4.16.3 Functional Reactive Agent-Based Simulation

Report by:	Jonathan Thaler
Status:	Experimental, active development

Implementations of Agent-Based Simulation (ABS) have so far been reduced to the context of Object-Oriented (OO). We investigate how ABS can be implemented in a pure functional language like Haskell. The fundamental problem is that unlike in OO there are no objects and no implicit aliases through which to access and change data: method calls are not available in FP.

We solve the problem of how to represent an agent and how agents can interact with each other. We build on the concept of Functional Reactive Programming for which we use the library Yampa.

This allows us to represent agents as signal-functions with different types as input and output. In each time-step an agent gets fed in an input and creates an output which is the input for the next time-step, creating a feedback. The input- and output-types contain incoming and outgoing messages and various events like start, termination, kill.

Also we build on the facilities Yampa provides for time-flow of a system be it continuous or discrete. For interactions between Agents we implemented messages, which are interactions over simulated time and a novel concept which we termed 'conversations', which are restricted interactions during which the simulation-time is halted.

Further reading

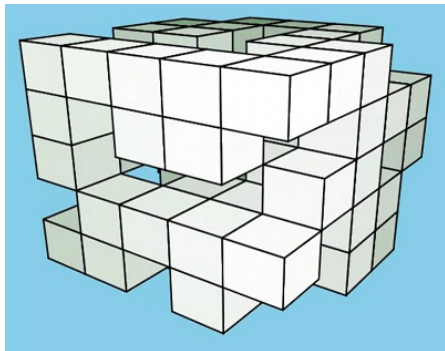
Repository: <https://github.com/thalerjonathan/phd/tree/master/coding/libraries/frABS>

4.17 Graphics and Audio

4.17.1 diagrams

Report by:	Brent Yorgey
Participants:	many
Status:	active development

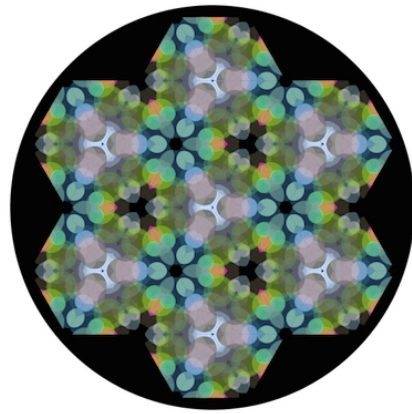
The diagrams framework provides an embedded domain-specific language for declarative drawing. The overall vision is for diagrams to become a viable alternative to DSLs like MetaPost or Asymptote, but with the advantages of being *declarative*—describing what to draw, not how to draw it—and *embedded*—putting the entire power of Haskell (and Hackage) at the service of diagram creation. There is always more to be done, but diagrams is already quite fully-featured, with a comprehensive user manual and a growing set of tutorials, a large collection of primitive shapes and attributes, many different modes of composition, paths, cubic splines, images, text, arbitrary monoidal annotations, named subdiagrams, and more.



What's new

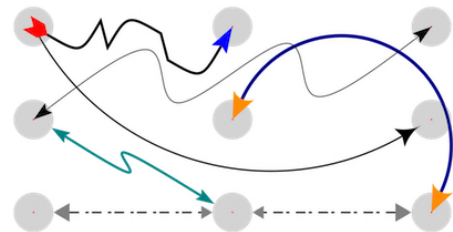
Diagrams 1.4 was released at the end of October, and mostly just adds new features, such as:

- B-spline support, and B-spline to cubic Bezier conversion
- Boolean operations on paths, such as union and intersection
- CSG support for 3D diagrams
- New techniques and tools for drawing 2D projections of 3D diagrams, illustrated above
- Constraint-based layout



Contributing

There is plenty of exciting work to be done; new contributors are welcome! Diagrams has developed an encouraging, responsive, and fun developer community, and makes for a great opportunity to learn and hack on some “real-world” Haskell code. Because of its size, generality, and enthusiastic embrace of advanced type system features, diagrams can be intimidating to would-be users and contributors; however, we are actively working on new documentation and resources to help combat this. For more information on ways to contribute and how to get started, see the Contributing page on the diagrams wiki: <http://haskell.org/haskellwiki/Diagrams/Contributing>, or come hang out in the #diagrams IRC channel on freenode.



Further reading

- <http://projects.haskell.org/diagrams>
- <http://projects.haskell.org/diagrams/gallery.html>
- <http://haskell.org/haskellwiki/Diagrams>
- <http://github.com/diagrams>
- <http://ozark.hendrix.edu/~yorgey/pub/monoid-pearl.pdf>
- <http://www.youtube.com/watch?v=X-8NckD2vOw>

4.17.2 csound-expression

Report by: Anton Kholomiov
Status: active, experimental

The csound-expression is a Haskell framework for electronic music production. It's based on very efficient and feature rich software synthesizer Csound. The Csound is a programming language for music production. With CE we can generate the Csound code out of high-level functional description.

This period was the period of the application of the library to the real world cases. Making more music with it and using it on the stage for the live performances. It became a bit more stable. There were some minor adjustments and fixes that came up during usage of the library. For example now it's possible to use arbitrary number of inputs/outputs for JACK instruments.

The library was successfully used on the stage for live performances by several artists on festivals and concerts. Library was used for real time music production. Some compositions used the generative algorithmic music techniques. Like playing bass randomly within fixed scale or introducing random variations in the drum patterns. Here is the link to the live recording from the mixing console of the track with live generated electronics: <https://soundcloud.com/kailash-project/kotlin-skazka-2017>.

To make it easier for beginners I've created non-haskell friendly tutorial. The library itself is huge. But tutorial covers most frequently used tools which I use to create my compositions. The tutorial was published at the Csound Journal (http://csoundjournal.com/issue23/Csound_expression_paper.html).

I'm glad to say that one track made with csound-expression was played at the Csound conference 2017 which was held at the Uruguay. Here is the link to the track: <https://soundcloud.com/anton-kho/early-morning-i-can-see-3-planets-in-the-sky> Or in conference proceedings: <http://csound.com/icsc2017/pieces.html>.

You can listen to the music that was made with Haskell and the library csound-expression:

- o <https://soundcloud.com/anton-kho>
- o <https://soundcloud.com/kailash-project>

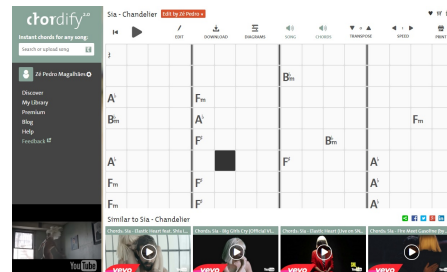
The library is available on Github and Hackage. See the packages csound-expression, csound-sampler and csound-catalog.

Further reading

- o <https://github.com/spell-music/csound-expression>
- o <http://hackage.haskell.org/package/csound-expression>
- o <http://csound.github.io/>

4.17.3 Chordify

Report by: Jeroen Bransen
Participants: W. Bas de Haas, José Pedro Magalhães, Dion ten Heggeler, Tijmen Ruizendaal, Gijs Bekenkamp, Hendrik Vincent Koops
Status: actively developed



Chordify is a music player that extracts chords from musical sources like Youtube, Deezer, Soundcloud, or your own files, and shows you which chord to play when. The aim of Chordify is to make state-of-the-art music technology accessible to a broader audience. Our interface is designed to be simple: everyone who can hold a musical instrument should be able to use it.

Behind the scenes, we have a chord extraction pipeline that is written in Haskell with some bindings to C libraries. We use Kiss FFT for computing audio features, which give information about the frequencies in the audio signal. These features are the inputs for a deep convolutional neural network that is trained and evaluated with bindings to the Tensorflow library. Finally a Hidden Markov Model, which is also trained on datasets of audio with manually annotated chords, is used to pick the final chord for each beat. This model encapsulates the rules of tonal harmony.

We have a distributed backend based on Cloud Haskell, allowing us to easily scale up when the demand increases. Our library currently contains about 6.5 million songs, and about 7,500 new songs are *Chordified* every day. We have also released an iOS app that allows iPhone and iPad users to interface with our technology more easily, and an Android version is being developed.

Chordify is a proud user of Haskell, but we have also encountered some problems and limitations of the language and the libraries. These include:

- o A hard-to-find memory leak, where the memory usage of one of our live systems grew slowly over time. After many failed debugging and profiling attempts, this turned out to be a library that was a bit too lazy in evaluating its data. Using a different library with slightly stricter evaluation solved this problem.
- o The signal processing libraries that we tried are not efficient and complete enough. At Chordify we want to do fast audio processing, for which Haskell implementations are not available or nowhere near the performance of C libraries.

The code for our old backend, called HarmTrace, is available on [Hackage](#), and we have [ICFP'11](#), [ISMIR'12](#), [ISMIR'16](#) and [DLM'17](#) publications describing some of the technology behind Chordify.

Further reading

<https://chordify.net>

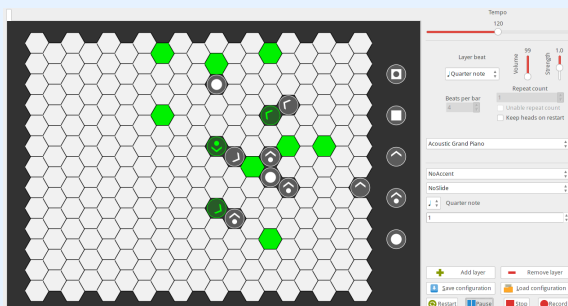
4.17.4 The Arpeggigon

Report by: Henrik Nilsson
Participants: Henrik Nilsson, Gueric Chupin, Jin Zhan
Status: Experimental: working but not yet feature complete

The Arpeggigon is a Functional Reactive Musical Automaton developed by Henrik Nilsson and summer interns Gueric Chupin (ENSTA ParisTech) and Jin Zhan (University of Nottingham) at the Functional Programming Laboratory, University of Nottingham (→6.6). It was developed as a case study for evaluating the combination of two reactive but complementary frameworks for implementing interactive time-aware applications: (FRP) as embodied by Yampa and Reactive Values and Relations (RVR). Hybrid (mixed continuous and discrete time) aspects were of particular interest.

We choose a musical application as music is a domain with hybrid temporal aspects at its core. The goal was to develop a full-fledged application in terms of look and feel and interoperability with standard MIDI software and hardware as would be found in a studio setting. To that end, we opted for using the standard GUI framework GTK+ for the user interface and the Jack framework for MIDI connectivity. Additionally, we wanted an application that is interesting and fun in its own right.

The Arpeggigon was inspired by the hardware *recTogon*: a “chain reactive performance arpeggiator”. It’s essentially a cellular automaton based around the Harmonic Table: a way to arrange musical notes on a hexagonal grid. *Play heads* bounce between *tokens* arranged on the hexagonal grid. Whenever a play head hits a token, a note corresponding to the position of the token on the grid is played. The picture below shows what the Arpeggigon looks like:



The case study demonstrates that FRP in combination with RVR is a compelling combination for building this kind of applications. FRP, in the synchronous dataflow tradition, aligns with the temporal and declarative nature of music, while RVR allows declarative interfacing with external components as needed for full-fledged musical applications.

The results have been written up in an application paper, “Funky Grooves: Declarative Programming of Full-Fledged Musical Applications”, that was presented at PADL 2017. There is also an extended technical report version and videos as well as slides from talks (PADL, FARM, Haskell eXchange, etc).

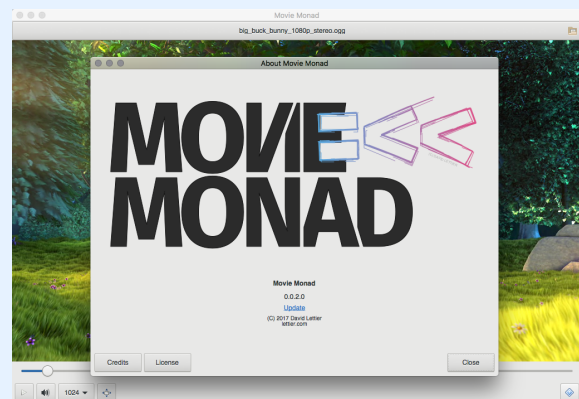
The code is publicly available on [GitLab](#).

Further reading

- o Paper: <http://eprints.nottingham.ac.uk/38747/>
- o Technical Report: <http://eprints.nottingham.ac.uk/38657/>
- o Haskell eXchange 2017 talk (video): <https://skillsmatter.com/skillscasts/10883-the-arpeggigon-a-functional-reactive-musical-automaton>
- o London Haskell Meetup talk (slides): <http://www.cs.nott.ac.uk/~psznhn/talks.html#lhmjune2017>
- o Computerphile overview: <https://www.youtube.com/watch?v=v0HIkFR1EN4>
- o Video Demo: <https://www.youtube.com/watch?v=yJteVN8OQYk>
- o Repository: <https://gitlab.com/chupin/arpeggigon>

4.17.5 Movie Monad

Report by: David Lettier
Status: Actively Maintained



Movie Monad is an open source desktop video player written in Haskell. Its run time dependencies include GTK+ and GStreamer. To interface with GTK+ and GStreamer, Movie Monad uses the *haskell-gi* bindings.

Features include the usual playback and volume controls. Users can play both local and remote files from the web.

While there already exists a wide array of video players, there is still room for Movie Monad with its intuitive interface and unobtrusive feature set. Movie Monad falls somewhere between mpv's minimalism and VLC's verbosity. The main reason for the project's existence, however, is to showcase Haskell's capabilities of targeting the desktop by using Haskell to build a nontrivial, multimedia GUI application.

The project is still ongoing and actively maintained. Originally, Movie Monad was web based. It used Fay, Clay, and Blaze to generate the JavaScript, CSS, and HTML needed to run the program with Electron. This year, however, the project was switched to GTK+ and GStreamer for portability and performance reasons.

You can obtain Movie Monad from GitHub and Hackage. A Linux AppImage can be downloaded from the GitHub releases page and should be listed on AppImageHub soon. If you need to build Movie Monad yourself, easy to follow instructions are provided on how to build Movie Monad for either Linux or macOS.

There are three new features scheduled to be released sometime soon. These include subtitle support, drag-and-drop playback, and a command-line interface to play a video file without first having to select it using the GUI. If you see a feature Movie Monad should have, be sure to leave your feedback on the GitHub issues page.

Further reading

<https://github.com/lettier/movie-monad>

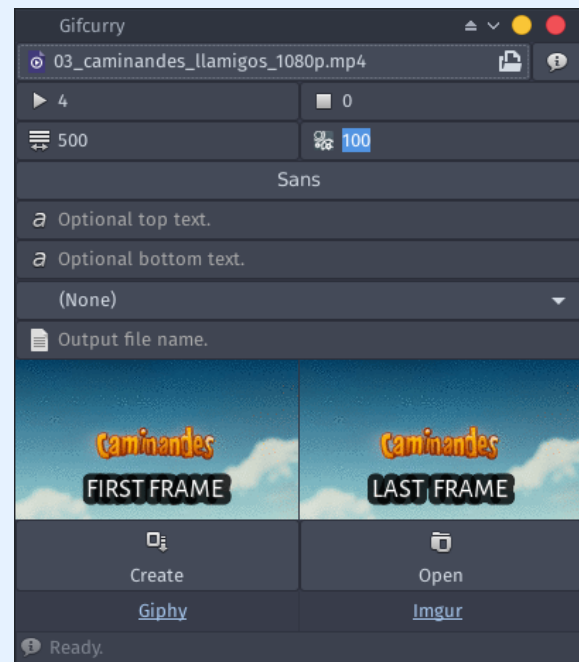
4.17.6 Gifcurry

Report by:

David Lettier

Status:

Actively Maintained



Gifcurry is an open source video to GIF maker written in Haskell. Powered by the core library, Gifcurry has both a command-line and graphical user interface. Features include controls for width, start, duration, output quality, and font selection. Users have the option to place text at the top and/or the bottom of the output GIF. Platforms supported include Linux and macOS.

Gifcurry is actively maintained and continuously improved. Soon to be a two year old project, Gifcurry has had 15 releases at the time of this writing. The most recent version was released in October of 2017.

Featured in eTechRoom, TECHFOUR, and OMG! Ubuntu!, Gifcurry has been downloaded over 1,200 times from Hackage. You can find Gifcurry on GitHub, Hackage, Arch Linux AUR, and AppImageHub. The Gifcurry AppImage is a portable executable that requires no installation and should run on most Linux based desktops.

Future improvements include a better GUI preview and the ability to crop the final GIF. As of right now, Gifcurry provides a preview by showing the first and last frame. This will change to showing a looping clip of the video segment selected.

Further reading

<https://github.com/lettier/gifcurry>

4.18 Games

4.18.1 Nomyx

Report by:	Corentin Dupont
Status:	stable, actively developed

Nomyx is a unique game where you can change the rules of the game itself, while playing it! In fact, changing the rules is the goal of the game. Changing a rule is considered as a move. The players can submit new rules or modify existing ones, thus completely changing the behavior of the game through time. The rules are managed and interpreted by the computer. They must be written in the Nomyx language, based on Haskell. This is the first complete implementation of a Nomic game on a computer.

At the beginning, the initial rules are describing:

- How to add new rules and change existing ones. For example a unanimity vote is necessary to have a new rule accepted.
- How to win the game. For example you win the game if you have 5 rules accepted.
The V1.0 has been released!
- a completely new web GUI
- a library of ready-made rules
- add client command line interface
- a REST API
- a new event programming library called 'Imprevu'
- container-based deployment

The game has been presented in Zurichac 2017, where we playing several live matches. A lot of learning material is available, including videos, a tutorial, a FAQ, and API documentation.

If you like Nomyx, you can help! There is a development mailing list (check the website).

Further reading

<http://www.nomyx.net>

4.18.2 EtaMOO

Report by:	Rob Leslie
Status:	experimental, active development

EtaMOO is a new, experimental MOO server implementation written in Haskell. MOOs are network accessible, multi-user, programmable, interactive systems well suited to the construction of text-based adventure games, conferencing systems, and other collaborative software. The design of EtaMOO is modeled closely after LambdaMOO, perhaps the most widely used implementation of MOO to date.

Unlike LambdaMOO which is a single-threaded server, EtaMOO seeks to offer a fully multi-threaded environment, including concurrent execution of MOO

tasks. To retain backward compatibility with the general MOO code expectation of single-threaded semantics, EtaMOO makes extensive use of software transactional memory (STM) to resolve possible conflicts among simultaneously running MOO tasks.

EtaMOO fully implements the MOO programming language as specified for the latest version of the LambdaMOO server, with the aim of offering drop-in compatibility. Several enhancements are also planned to be introduced over time, such as support for 64-bit MOO integers, Unicode MOO strings, and others.

Recent development has brought the project to a largely usable state. A major advancement was made by integrating the *vcache* library from Hackage for persistent storage — a pairing that worked especially well given EtaMOO's existing use of STM. Consequently, EtaMOO now has a native binary database backing with continuous checkpointing and instantaneous crash recovery. Furthermore, EtaMOO takes advantage of *vcache*'s automatic value cache with implicit structure sharing, so the entire MOO database need not be held in memory at once, and duplicate values (such as object properties) are stored only once in persistent storage.

Further development has incorporated optional support for the lightweight object WAIF data type as originally described and implemented for the LambdaMOO server. The *vcache* library was especially useful in implementing the persistent shared WAIF references for EtaMOO.

Future EtaMOO development will focus on feature parity with the LambdaMOO server, full Unicode support, and several additional novel features.

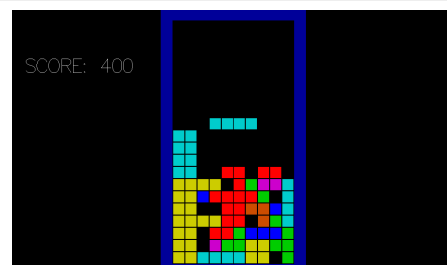
Latest development of EtaMOO can be seen on GitHub, with periodic releases also being made available through Hackage.

Further reading

- <https://github.com/verement/etamoo>
- <https://hackage.haskell.org/package/EtaMOO>
- <https://en.wikipedia.org/wiki/MOO>

4.18.3 Tetris in Haskell in a Weekend

Report by:	Michael Georgouloupoulos
Status:	actively developed



I made a Tetris in Haskell, while learning the basics of the language, in order to gain some hands-on experience, and also to convince myself that Haskell is

a practical language that's worth the time investment and the steep learning curve.

I am now convinced that that is the case. In fact, I'm amazed at how concise and readable Haskell code can be, and I can already acknowledge Haskell as a tool for productivity, predictability and reliability, which are without a doubt, properties most software developers could benefit from.

I have documented this experience as a series of thoughts from the point of view of a beginner, in the form of a blog post titled "Tetris in Haskell in a Weekend"

I also documented the project's evolution in small increments as a git repository that might be of interest to other beginners. The repository can be accessed via github, and contributions are welcome

Further reading

<https://github.com/mgeorgoulopoulos/TetrisHaskellWeekend>

4.18.4 Barbarossa

Report by:	Nicu Ionita
Status:	actively developed

Barbarossa is a UCI chess engine written completely in Haskell. UCI is one of the two most used protocols used in the computer chess scene to communicate between a chess GUI and a chess engine. This way it is possible to write just the chess engine, which then works with any chess GUI.

I started in 2009 to write a chess engine under the name Abulafia. In 2012 I decided to rewrite the evaluation and search parts of the engine under the new name, Barbarossa.

My motivation was to demonstrate that even in a domain in which the raw speed of a program is very important, as it is in computer chess, it is possible to write competitive software with Haskell. The speed of Barbarossa (measured in searched nodes per second) is still far behind comparable engines written in C or C++. Nevertheless Barbarossa can compete with many engines - as it can be seen on the CCRL rating lists, where it is currently listed with a strength of about 2200 ELO.

Barbarossa uses a few techniques which are well known in the computer chess scene:

- in evaluation: material, king safety, piece mobility, pawn structures, tapped evaluation and a few other less important features
- in search: principal variation search, transposition table, null move pruning, killer moves, futility pruning, late move reduction, internal iterative deepening.

I still have a lot of ideas which could improve the strength of the engine, some of which address a higher

speed of the calculations, and some, new chess related features, which may reduce the search tree.

The engine is open source and is published on github. The last released version is Barbarossa v0.4.0 from December 2016.

Further reading

- <https://github.com/nionita/Barbarossa/releases>
- <http://www.computerchess.org.uk/ccrl/404/>

4.18.5 tttool

Report by:	Joachim Breitner
Status:	active development

The Ravensburger Tiptoi[®] pen is an interactive toy for kids aged 4 to 10 that uses OiD technology to react when pointed at the objects on Ravensburger's Tiptoi books, games, puzzles and other toys. It is programmed via binary files in a proprietary, undocumented data format.

We have reverse engineered the format, and created a tool to analyze these files and generate our own. This program, called `tttool`, is implemented in Haskell, which turned out to be a good choice: Thanks to Haskell's platform independence, we can easily serve users on Linux, Windows and OS X.

The implementation makes use of some nice Haskell idioms such as a monad that, while parsing a binary, creates a hierarchical description of it and a writer monad that uses laziness and `MonadFix` to reference positions in the file "before" these are determined.

Further reading

- <https://github.com/entropia/tip-toi-reveng>
- <http://tttool.entropia.de/> (in German)
- <http://funktionale-programmierung.de/2015/04/15/monaden-reverse-engineering.html> (in German)

4.19 Data Tracking

4.19.1 hledger

Report by:	Simon Michael
Status:	stable, actively developed

`hledger` is a set of cross-platform tools (and Haskell libraries) for tracking money, time, or any other commodity, using double-entry accounting and a simple text file format. `hledger` aims to be a reliable and practical tool for daily use, and provides command-line, curses-style, and web interfaces. It is a largely compatible Haskell reimplement of John Wiegley's Ledger program. `hledger` is released under GNU GPLv3+.

In November 2015, the immediate plans were to improve docs and help, improve parser speed and memory efficiency, integrate a separate parser for Ledger files

built by John Wiegley, `hledger-ui` improvements, and work towards the 1.0 release.

All but one of these goals have been achieved:

- o docs have been reorganized, with more focussed manuals available in multiple versions, formats and as built-in help
- o `hledger` has migrated from `parsec` to `megaparsec` and from `String` to `Text`, parsers have been simplified, memory usage is 30% less on large files, speed is slightly improved all around
- o the `ledger4` parser is not yet integrated
- o `hledger-ui` has acquired many new features making it more useful (file editing, filtering, historical/period modes, quick period browsing..)
- o 1.0 has been released!

Also,

- o `hledger-web` is more robust and more mobile-friendly
- o `hledger-api`, a simple web API server, has been added
- o a new "timedot" file format allows retroactive/approximate time logging
- o we now support GHC 8 and GHC 7.10, dropping GHC 7.8 and 7.6 support. (GHC 7.8 support requires a maintainer).
- o `hpack` is now used for maintaining cabal files
- o our benchmarking tool has been spun off as the `quickbench` package
- o the `hledger.org` website is simpler, clearer, and more mobile-friendly
- o a call for help was sent out last month, and contributor activity has increased.

Future plans include:

- o support the 1.0 release
- o improve the website and docs
- o grow the user & developer community
- o clean up, automate, improve and scale our processes
- o improve quality, reduce waste
- o add the `ledger4` parser
- o add budget/goal-tracking features
- o improve `hledger-ui` usability and features; live reloading

`hledger` is available from the `hledger.org` website, from Github, Hackage, and Stackage, and is packaged for a number of systems including Homebrew, Debian, Ubuntu, Gentoo, Fedora, and NixOS.

Further reading

<http://hledger.org>

4.19.2 gipeda

Report by:	Joachim Breitner
Status:	active development

Gipeda is a a tool that presents data from your program's benchmark suite (or any other source), with nice tables and shiny graphs. Its name is an abbreviation for "Git performance dashboard" and highlights that it is aware of git, with its DAG of commits.

Recent commits

a day ago	898bbdf	Remove dead function <code>patSynTyDetails</code>	275	👍	0	👎	0
2 days ago	04e8366	ELFix86_64: map object file sections separately into the low 2GB	275	👍	1	👎	0
10 days ago	e2b579e	Parser: revert some error messages to what they were before 7.10	275	👍	1	👎	0
15 days ago	03b3804	Add Data.Semigroup and Data.List.NonEmpty (re #10385)	277	👍	0	👎	1

Gipeda powers the GHC performance dashboard at <http://perf.haskell.org>, but it builds on Shake and creates static files, so that hosting a gipeda site is easily possible.

Further reading

<https://github.com/nomeata/gipeda>

4.19.3 arbtt

Report by:	Joachim Breitner
Status:	working

The program `arbtt`, the automatic rule-based time tracker, allows you to investigate how you spend your time, without having to manually specify what you are doing. `arbtt` records what windows are open and active, and provides you with a powerful rule-based language to afterwards categorize your work. And it comes with documentation!

The program works on Linux, Windows, and MacOS X.

Further reading

- o <http://arbtt.nomeata.de/>
- o <http://www.joachim-breitner.de/blog/archives/336-The-Automatic-Rule-Based-Time-Tracker.html>
- o http://arbtt.nomeata.de/doc/users_guide/

4.19.4 propellor

Report by:	Joey Hess
Status:	actively developed

Propellor is a configuration management system for Linux that is configured using Haskell. It fills a similar role as Puppet, Chef, or Ansible, but using Haskell instead of the ad-hoc configuration language typical of such software. Propellor is somewhat inspired by the functional configuration management of NixOS.

A simple configuration of a web server in Propellor looks like this:

```
webServer :: Host
webServer = host "webserver.example.com"
    & ipv4 "93.184.216.34"
    & staticSiteDeployedTo "/var/www"
    'requires' Apt.serviceInstalledRunning "apache2"
    'onChange' Apache.reloaded
staticSiteDeployedTo :: FilePath → Property DebianLike
```

There have been many benefits to using Haskell for configuring and building Propellor, but the most striking are the many ways that the type system can be

used to help ensure that Propellor deploys correct and consistent systems. Beyond typical static type benefits, GADTs and type families have proven useful. For details, see [the blog](#).

An eventual goal is for Propellor to use type level programming to detect at compile time when a host has eg, multiple servers configured that would fight over the same port. Moving system administration toward using types to prove correctness properties of the system.

Propellor recently has been extended to support FreeBSD, and this led to Propellor properties including information about the supported OSes in their types. That was implemented using singletons to represent the OS, and functions over type level lists. For details, see [this blog post](#).

Propellor has also been extended to be able to create bootable disk images. This allows it to not only configure existing Linux systems, but manage their entire installation process.

Further reading

<http://propellor.branchable.com/>

4.20 Others

4.20.1 ADPfusion

Report by:	Christian Höner zu Siederdisen
Status:	usable, active development

ADPfusion provides a low-level domain-specific language (DSL) for the formulation of dynamic programs with emphasis on computational biology and linguistics. We follow ideas established in algebraic dynamic programming (ADP) where a problem is separated into a grammar defining the search space and one or more algebras that score and select elements of the search space. The DSL has been designed with performance and a high level of abstraction in mind.

ADPfusion grammars are abstract over the type of terminal and syntactic symbols. Thus it is possible to use the same notation for problems over different input types. We directly support grammars over strings, sets (with boundaries, if necessary), and trees. Linear, context-free and multiple context-free languages are supported, where linear languages can be asymptotically more efficient both in time and space. ADPfusion is extendable by the user without having to modify the core library. This allows users of the library to support novel input types, as well as domain-specific index structures. The extension for tree-structured inputs is implemented in exactly this way and can serve as a guideline.

As an example, consider a grammar that recognizes palindromes. Given the non-terminal p , as well as parsers for single characters c and the empty input ϵ , the production rule for palindromes can be formulated as $p \rightarrow c p c \mid \epsilon$.

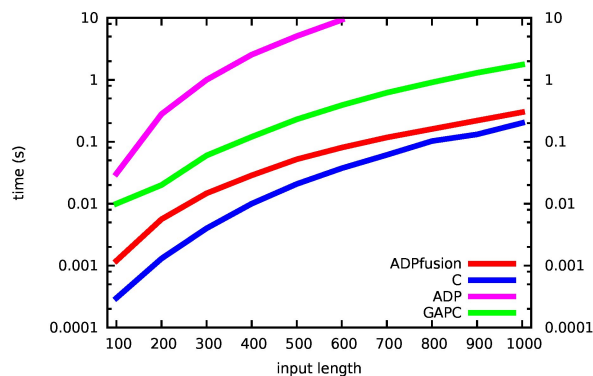
The corresponding ADPfusion code is similar:

```
p (f <<< c % p % c ||| g <<< e ... h)
```

We need a number of combinators as “glue” and additional evaluation functions f , g , and h . With $f c_1 p c_2 = p \ \&\& \ (c_1 \equiv c_2)$ scoring a candidate, $g e = \text{True}$, and $h xs = \text{or } xs$ determining if the current substring is palindromic.

This effectively turns the grammar into a memo-function that then yields the optimal solution via a call to `axiom p`. Backtracking for co- and sub-optimal solutions is provided as well. The backtracking machinery is derived automatically and requires the user to only provide a set of pretty-printing evaluation functions.

As of now, code written in ADPfusion achieves performance close to hand-optimized C, and outperforms similar approaches (Haskell-based ADP, GAPC producing C++) thanks to stream fusion. The figure shows running times for the *Nussinov algorithm*.



The entry on generalized Algebraic Dynamic Programming ([→ 4.11.7](#)) provides information on the associated high-level environment for the development of dynamic programs.

Further reading

- <http://www.bioinf.uni-leipzig.de/Software/gADP>
- <http://hackage.haskell.org/package/ADPfusion>
- <http://dx.doi.org/10.1145/2364527.2364559>

4.20.2 leapseconds-announced

Report by:	Björn Buckwalter
Status:	stable, maintained

The leapseconds-announced library provides an easy to use static LeapSecondMap with the leap seconds announced at library release time. It is intended as a quick-and-dirty leap second solution for one-off analyses concerned only with the past and present (i.e. up until the next as of yet unannounced leap second), or for applications which can afford to be recompiled against an updated library as often as every six months.

Version 2017.1 of leapseconds-announced was released to support the change from LeapSecondTable to

LeapSecondMap in time-1.7. It contains all leap seconds up to 2017-01-01. A new version will be uploaded if/when the IERS announces a new leap second.

Further reading

<https://hackage.haskell.org/package/leapseconds-announced>

4.20.3 Haskell in Green Land

Report by:	Gilberto Melfe
Participants:	Luís Gabriel Lima, Francisco Soares-Neto, Paulo Lieuthier, Fernando Castor, João Paulo Fernandes
Status:	mostly stable, with ongoing extensions

In the Haskell in Green Land initiative we attempt to understand the energy behavior of programs written in Haskell. It is particularly interesting to study Haskell in the context of energy consumption since Haskell has mature implementations of sophisticated features such as laziness, partial function application, software transactional memory, tail recursion, and a kind system. Furthermore, recursion is the norm in Haskell programs and side effects are restricted by the type system of the language.

We analyze the energy efficiency of Haskell programs from two different perspectives:

- strictness: by default, expressions in Haskell are lazily evaluated, meaning that any given expression will only be evaluated when it is first necessary. This is different from most programming languages, where expressions are evaluated strictly and possibly multiple times;
- concurrency: previous work has demonstrated that concurrent programming constructs can influence energy consumption in unforeseen ways.

Concretely, we have addressed the following high-level research question: To what extent can we save energy by refactoring existing Haskell programs to use different data structure implementations or concurrent programming constructs?

In order to address this research question, we conducted two complementary empirical studies:

- we analyzed the performance and energy behavior of several benchmark operations over 15 different implementations of three different types of data structures considered by the Edison Haskell library;
- we assessed three different thread management constructs and three primitives for data sharing using nine benchmarks and multiple experimental configurations.

Overall, experimental space exploration comprises more than 2000 configurations and 20000 executions.

We found that small changes can make a big difference in terms of energy consumption. For example, in one of our benchmarks, under a specific configuration, choosing one data sharing primitive over another can yield 60% energy savings. Nonetheless, there is no universal winner.

In addition, the relationship between energy consumption and performance is not always clear. Generally, especially in the sequential benchmarks, high performance is a proxy for low energy consumption. Nonetheless, when concurrency comes into play, we found scenarios where the configuration with the best performance (30% faster than the one with the worst performance) also exhibited the second worst energy consumption (used 133% more energy than the one with the lowest usage).

To support developers in better understanding this complex relationship, we have extended two existing tools from the Haskell world:

- the powerful benchmarking library Criterion;
- the profiler that comes with the Glasgow Haskell Compiler.

Originally, such tools were devised for performance analysis and we have adapted them to make them *energy-aware*.

Further reading

The data for this study, the source code for the implemented tools and benchmarks as well as a paper describing all the details of our work can be found at green-haskell.github.io.

Furthermore, we have referenced the following papers:

- Pinto, Gustavo and Castor, Fernando and Liu, Yu David: *Understanding Energy Behaviors of Thread Management Constructs*, Proceedings of the 2014 ACM International Conference on Object Oriented Programming Systems Languages & Applications
- Luís Gabriel Lima and Gilberto Melfe and Francisco Soares-Neto and Paulo Lieuthier and João Paulo Fernandes and Fernando Castor, *Haskell in Green Land: Analyzing the Energy Behavior of a Purely Functional Language*, Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER'2016)

4.20.4 clr-haskell (Haskell interoperability with the Common Language Runtime)

Report by:	Tim Matthews
Participants:	José Iborra López
Status:	experimental, actively developed

clr-haskell is a project to enable the use of code within the common language runtime (.NET / Mono / CoreCLR) from GHC Haskell.

This project provides 2 primary flavours for a developer to interop between the CLR & Haskell:

The Haskell-er's strongly typed flavour. Takes advantage of the latest GHC extensions to provide a way of encoding an OO type system within the Haskell type system.

The .NET dev's inline flavour. Provides the ability to call directly into valid C# / F# syntax via quasi-quoted template Haskell.

Further reading

<https://gitlab.com/tim-m89/clr-haskell>

4.20.5 Kitchen Snitch server

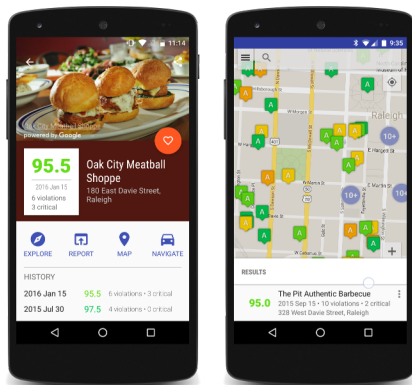
Report by:	Dino Morelli
Participants:	Betty Diegel
Status:	stable, actively developed

This project is the server-side software for Kitchen Snitch, a mobile application that provides health inspection scores, currently for the Raleigh-Durham area in NC, USA. The data can be accessed on maps along with inspection details, directions and more.

The back-end software provides a REST API for mobile clients and runs services to perform regular inspection data acquisition and maintenance.

Kitchen Snitch has been in development for over a year and is running on AWS. The mobile client and server were released for public use in April of 2016 after a beta-test period.

Some screenshots of the Android client software in action:



Getting Kitchen Snitch:

The mobile client can be installed from the Google Play Store. There is also a landing page <http://getks.honuapps.com/>.

The Haskell server source code is available on darcs-hub at the URLs below.

Further reading

- ks-rest <http://hub.darcs.net/dino/ks-rest>
- ks-download <http://hub.darcs.net/dino/ks-download>
- ks-library <http://hub.darcs.net/dino/ks-library>

4.20.6 FRTrader

Report by:	Dimitri DeFigueiredo
Status:	active

FRTrader is a functional reactive bitcoin trading bot. It uses the reactive-banana FRP library and currently has bindings to the GDAX bitcoin exchange. The bot uses as a multi-threaded architecture that makes it easy to plug in extra exchanges and to trade on multiple exchanges simultaneously.

The code is available on [github](#).

The following [talk](#) recorded at BayHac 2017 explains the design rationale and the use of FRP in trading.

We welcome contributions and hope to add bindings to other exchanges soon. Also, feel free to make a huge trading profit! Enjoy!

4.20.7 shell-conduit

Report by:	Sibi Prabakaran
Participants:	Chris Done
Status:	active

shell-conduit allows writing shell scripts with conduit. It uses template Haskell to bring all the executables in the PATH as top level functions which can be used to launch them as a process.

Further reading

- <https://github.com/psibi/shell-conduit>
- <http://chrisdone.com/posts/shell-conduit>

4.20.8 Hapoid

Report by:	Wisnu Adi Nurcahyo
Status:	Under development

Hapoid is an Portable Object Translation file Linter for Bahasa Indonesia.

Currently, this project is a prototype but it will continue to be developed.

Further reading

<https://github.com/wisn/hapoid>

4.20.9 Hanum - OSM Dynamic Attributes Linter

Report by:	Wisnu Adi Nurcahyo
Status:	Prototype

Hanum is an OpenStreetMap dynamic attributes linter with custom presets.

Some contributors to OSM may just want to fix wrong attributes on the OSM data. This means that they might not want to see any path or shape of the data. This linter thus acts as a library which allows data validation and creating new editors for OSM.

The linter filters error, thus reducing possible conflicts when submitting data to OSM. To further enhance the capabilities, custom presets and rules can be created for each individual country.

Motivation

In Indonesia, especially in Kalimantan there are many areas where OSM has invalid data. For example, SMAN 1 Bintang Ara, which is a senior high school, shows on OSM as having a *school : type_idn* attribute of *elementary school*. Furthermore, it also lacks an address attribute.

A similar example is found in Papua: there are locations where the *admin_level* attribute is not a number as it should be, as well as areas where *addr : full* has invalid formatting.

With Hanum, we can define custom rules and use these presets to improve data quality.

Further reading

<https://github.com/wisn/hanum>

4.20.10 tldr

Report by:	Sibi Prabakaran
Status:	active

tldr is a command line client for the TLDR pages. The TLDR pages are a community effort to simplify the beloved man pages with practical examples.

```
estbl:jane (~) -> tldr tar
tar
Archiving utility.
Often combined with a compression method, such as gzip or bzip.

- Create an archive from files:
  tar cf {{target.tar}} {{file1 file2 file3}}

- Create a gzipped archive:
  tar czf {{target.tar.gz}} {{file1 file2 file3}}

- Extract an archive in a target folder:
  tar xf {{source.tar}} -C {{folder}}

- Extract a gzipped archive in the current directory:
  tar xzf {{source.tar.gz}}

- Extract a bzipped archive in the current directory:
  tar xjf {{source.tar.bz2}}

- Create a compressed archive, using archive suffix to determine the compression program:
  tar caf {{target.tar.xz}} {{file1 file2 file3}}

- List the contents of a tar file:
  tar tvf {{source.tar}}
```

New development includes the support for the new syntax of TLDR pages. The code for that is available in the new-format branch and will be merged with HEAD once the relevant upstream changes are finalized and merged.

Further reading

<https://github.com/psibi/tldr-hs>

5 Commercial Users

5.1 Well-Typed LLP

Report by:	Adam Gundry
Participants:	Duncan Coutts, Andres Löh and others

Well-Typed is a Haskell services company. We provide commercial support for Haskell as a development platform, including consulting services, training, bespoke software development and technical support. For more information, please take a look at our website or drop us an e-mail at (info@well-typed.com).

One of our main responsibilities is the maintenance of GHC (→ 3.1), thanks to support from Microsoft Research and others. At the beginning of the year we were delighted to welcome Reid Barton and David Feuer to join the team alongside Ben Gamari. Work is now well underway on the 8.2.1 release. If your company is interested in supporting work on GHC, either by contributing to the general maintenance pool or funding work on specific issues, please get in touch with us. We are also able to provide technical support with toolchain issues such as non-standard GHC configurations.

We endeavour to make our work available as open source software where possible, and contribute back to existing projects we use, although inevitably much of our work is proprietary to our clients and not publicly available. Here is a non-exhaustive list of open-source projects to which we contribute:

- Well-Typed is cooperating with Tweag I/O and LeapYear Technologies to improve bindings to the Java Native Interface (JNI) and Spark, the Apache framework for writing distributed applications (a generalization of Hadoop).
- Duncan Coutts has been further improving `new-build`, the Nix-style local builds feature for Cabal (→ 4.2.1).
- Duncan Coutts, Austin Seipp, Ben Gamari and other contributors are working on the `binary-serialise-cbor` library, an improved version of (large parts of) the `binary` package.
- Andres Löh recently released a new version of the generic programming library `generics-sop` (→ 4.1.2), taking advantage of the new type-level metadata support in GHC 8 to provide new possibilities for generic programming, such as record subtyping.
- Adam Gundry continues his work on record field overloading, with GHC 8.2.1 including modifications to the `OverloadedLabels` extension and a new facility for polymorphism over record field selectors.

We are looking forward to the next Haskell eXchange in London from 12–13th October 2017, and we are planning another free Hackathon on 14th–15th October. Together with Skills Matter, we will be offering public Haskell training courses in London from 26th–

30th June and around the Haskell eXchange (10th–11th, 16th–17th October). Registration is now open for both the courses and eXchange, so if you would like to come, register now!

We'll also once again be participating in ZuriHac (Zurich, June 2017), and ICFP and affiliated events (Oxford, September 2017).

If you are interested in getting information about Well-Typed events (such as conferences or courses we are participating in or organising), you can subscribe to our events mailing list. We are also always looking for new clients and projects, so if you have something we could help you with, or even would just like to tell us about your use of Haskell, please drop us an e-mail.

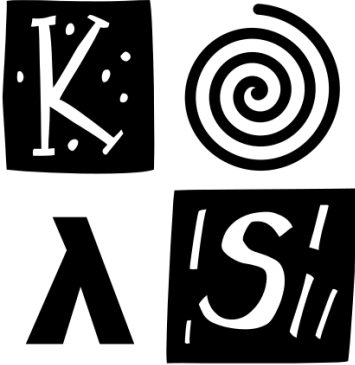
Further reading

- Home page: <https://www.well-typed.com>
- Blog: <https://www.well-typed.com/blog/>
- Training page: https://www.well-typed.com/services_training
- Events mailing list: <https://www.well-typed.com/cgi-bin/mailman/listinfo/events>
- JNI bindings: <https://github.com/tweag/inline-java>
- Spark bindings: <https://github.com/tweag/sparkle>
- `binary-serialise-cbor`: <https://github.com/well-typed/binary-serialise-cbor/>
- `generics-sop`: <https://github.com/well-typed/generics-sop/>
- GHC proposal for overloaded record fields: <https://github.com/ghc-proposals/ghc-proposals/pull/6>
- Haskell eXchange 2017: <https://skillsmatter.com/conferences/8522-haskell-exchange-2017>
- Skills Matter courses: <https://skillsmatter.com/explore?q=haskell>

5.2 Keera Studios LTD

Report by:

Ivan Perez



Keera Studios Ltd. is a Haskell development studio that focuses on games and mobile apps for both iOS and Android.

This year we have completed a set of development tools for mobile Haskell games that enables compiling, testing, packaging and deploying mobile games with no effort. Our framework is versatile enough to accommodate not only games, but also mobile apps using standard widget toolkits available on mobile platforms, in order to provide a natural look and feel.

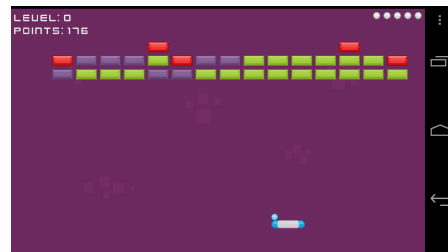
Our development toolkit has been designed to be trivial to run in different environments. Currently we use Travis CI to compile, package, sign and automatically upload our games to Google Play for Android (and tweet when a major release is out!). When developing locally, our toolchain maximizes caching, and is able to recompile a mobile app and deploy it to a phone (connected via USB) in less than 45 seconds, or to the online store in less than one minute.

We top our development framework with a novel tool for testing and debugging mobile games: Haskell TitanTM (Testing Infrastructure for Temporal AbstractionNs). Haskell TitanTM is designed to take advantage of Haskell's referential transparency to deliver fully reproducible game runs that can be saved, replayed, paused, played backwards, modified and debugged. Our GUI tool communicates with the game running on a phone or a computer, and uses Temporal Logic to specify game assertions. Players can record a game run with minimal overhead and send it over the internet to our servers, which we can use to reproduce bugs deterministically on the same architecture. Our tool also integrates well with QuickCheck, which we use to test game assertions and find counterexamples. Effectively, we can see QuickCheck play.

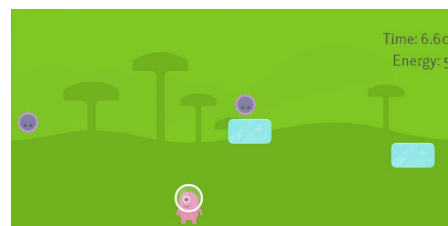
We have published Magic Cookies!, the first commercial game for iOS and Android written in Haskell, available on iTunes (<https://goo.gl/6gB6sb>) and on Google PlayTM (<https://goo.gl/cM1tD8>). Currently holding a 4.8/5 star review, Magic Cookies! been in the Top 100 of paid board games on Google Play US, and Top 125 in the European Union, number 7 for paid braingames in Taiwan, number 15 in Italy, and Top 50 in 110 countries.



We have developed a breakout clone that runs on Windows, Linux, Mac, iOS, Android and Web (<http://goo.gl/53pK2x>), using *hardware acceleration* and *parallelism*. This game has been released on Google Play for Android, is distributed pre-compiled for Ubuntu via Launchpad, and can be played directly on the browser, compiled via GHCJS. The desktop version additionally supports Nintendo Wiimotes and Kinect. In recent days, thanks to the collaborations of many volunteer Haskellers, this game has seen updated documentation, better support for all OSs, and now includes 9 new levels.



We are currently developing three games for both Android and iOS. The first, a novel game designed to challenge user's ability to track multiple objects moving simultaneously in different directions. The second, codenamed pang-a-lambda, is inspired by the classic Super Pang and takes advantage of newer developments in the FRP implementation Yampa to include declarative physics simulations and time transformations, including the possibility of reversing time. Both of these games are now being finalized.



The third, a visual interactive game about the war in Ukraine, tries to depict the everyday life of people affected by the war in an effort to raise awareness about the current state of affairs, how people are being treated and what they have lost. This game invites the player to see the human side of people who remain invisible to their own society, only to become an unnamed shield standing at the conflict line. On the basis of this game, our company has been accepted in Facebook’s acceleration programme for new apps and startups FbStart, and being granted thousands of dollars in digital goods and services for support and advertising.

We have developed GALE, a DSL for graphic adventures, together with an engine and a basic IDE that allows non-programmers to create their own 2D graphic adventure games without any knowledge of programming. Supported features include multiple character states and animations, multiple scenes and layers, movement bitmasks (used for shortest-path calculation), luggage, conversations, sound effects, background music, and a customizable UI. The IDE takes care of asset management, generating a fully portable game with all the necessary files. The engine is multi-platform, working seamlessly on Linux, Windows, Android and iOS. This work has been described in the paper “GALE: A functional Graphic Adventure Library and Engine”, presented at FARM 2017.

We have also started the Haskell Game Programming project (<http://git.io/vlxtJ>), which contains documentation and multiple examples of multimedia, access to gaming hardware, physics and game concepts. We have developed a battery of Haskell mobile demos, covering SDL multimedia (including demos for multi-touch and accelerometers), communication with Java via C/C++, Facebook/Twitter status sharing, access to each mobile ecosystem (for instance, to use Android built-in shared preferences storage system, or for in-app payments), and use to native mobile widgets on Android and iOS.

All of this proves that Haskell truly is a viable option for *professional game and app development*, for both mobile and desktop. Our novel testing and debugging tools show that Haskell can, in certain respects, be more suitable than other languages for game and app programming, especially for (heterogeneous) mobile devices. In combination with our samples and our compilation, packaging and deployment tool, we have a complete, pain-free mobile app development suite.

Our GUI applications are created using Keera Hails, our open-source *reactive* programming library (<http://git.io/vTvXg>). Keera Hails provides integration with GTK+, network sockets, files, FRP Yampa signal functions and other external resources. Experimental integration with wxWidgets, Qt, Android and iOS (using each platform’s default widget system, communicating via FFI), and HTML DOM (via GHCJS) is also available. We have used Keera Hails for our Graphic Adventure IDE, the open-source posture monitor Keera Posture (<http://git.io/vTvXy>), as well as

multiple other commercial and open-source applications. This reactive framework is straightforward to adapt to new platforms, and we have recently shown a reactive Haskell application that works on iOS, Android, Windows, Linux, Mac and Web, just by choosing different backends (<https://goo.gl/nFUA2u>).

Last year, Guericc Chupin (ENSTA Paris-Tech) and Henrik Nilsson (University of Nottingham (→6.6)) published Arpeggigon (→4.17.4) (<https://gitlab.com/chupin/arpeggigon>), an interactive cellular automaton for composing groove-based music, which combines the FRP implementation Yampa and our reactive framework Keera Hails. The former, in the synchronous dataflow tradition, aligns with the temporal and declarative nature of music, while the latter allows declarative interfacing with external components as needed for full-fledged musical applications. Their results have been written up in an application paper, *Funky Grooves: Declarative Programming of Full-Fledged Musical Applications*, presented at PADL 2017, and demonstrated at FARM 2017, the Haskell eXchange 2017, and Haskell in Leipzig 2017.

Videos and details of our work are published regularly on Facebook (<https://fb.me/keerastudios>), on Twitter (<https://www.twitter.com/keerastudios>), and on our company website (<http://www.keera.co.uk>). If you want to use Haskell in your next game, desktop or mobile application, or to receive more information, please contact us at .

5.3 Stack Builders

Report by:	Stack Builders
Status:	software consultancy



stackbuilders

Stack Builders is an international software consultancy, aimed to push the boundaries of the software industry. As consultants, we use functional programming, especially Haskell, to build amazing products. The company has offices in New York, United States, and Quito, Ecuador.

In addition to our Haskell software consultancy services, we are actively involved with the Haskell community:

- We organize **Quito Lambda**, a monthly meetup about functional programming in Quito, Ecuador (Now also available for live streaming). The talks are conducted in Spanish for this meetup.
- We maintain several packages in Hackage including *cassava-megaparsec*, *dotenv*, *hapistrano*, *inflections*,

octohat, openssh-github-keys, stache, and twitter-feed.

- We talk about Haskell at universities and events such as Lambda Days and BarCamp Rochester.
- We write blog posts and [tutorials](#) about Haskell.

For more information, take a look at our [website](#) or get in touch with us at info@stackbuilders.com.

Further reading

<http://www.stackbuilders.com/>

5.4 McMaster Computing and Software Outreach

Report by:	Christopher Anand
Status:	active

McMaster Computer Science Outreach visits schools in Ontario, Canada to teach basic Computer Science topics and discuss the impacts of the Information Revolution, teaching children from six to sixteen, using Elm for the programming portion of our programming. Elm looks a lot like Haskell, but does not have user-definable type classes and is strict. We presented a paper about our tools and curriculum at TFPIE 2017, https://www.cs.kent.ac.uk/people/staff/sjt/TFPIE2017/TFPIE_2017/Programme.html.

We would like to thank Google for igniteCS funding, which allowed us to run seven-week workshops in schools in high-needs schools. Many of their contributions are in our Hall of Fame: <http://outreach.mcmaster.ca/menu/fame.html>.

6 Research and User Groups

6.1 DataHaskell

Report by:	Marco Zocca
Participants:	Nikita Tchayka, Mahdi Dibaiee, John Vial, Stefan Dresselhaus, Michał Gajda, and many others
Status:	Ongoing

The DataHaskell community was initiated in September 2016 as a gathering place for scientific computing, machine learning and data science practitioners and Haskell programmers; we observe a growing interest in using functional composition, domain-specific languages and type inference for implementing robust and reusable data processing pipelines.

DataHaskell revolves around a [Gitter chatroom](#) and a [GitHub organization](#). The community is slowly but steadily growing; new interested people join the chatroom discussion (which now counts more than 230 unique users) every few days, and 20 or so are active on an average week.

We have a [documentation page](#) that serves both as a knowledge base of related Haskell packages and frameworks and to coordinate development, along with a [package benchmarking repository](#).

After an informal survey we concluded that large part of our userbase seems to be lacking most:

- an IDE for exploratory data analysis,
- a generic ‘data-frame’ for fast import and manipulation of heterogeneous tabular data,
- a native numerical back-end.

The notebook-IDE situation has improved, thanks for example to an updated iHaskell and the new, native [Haskell.do](#) editor. Dataframes are a more subtle topic, that require domain-specific optimizations, and we are also actively working on that after adopting the [analyze](#) package.

Some of us met in person at ZuriHac (Zürich) and ICFP 2017 (Oxford). An informal DataHaskell workshop took place on Saturday 9, 2017 at ICFP, during which a series of lightning talks showed various applications of the current state of the library ecosystem, e.g. rendering mathematics-heavy code on Hackage, high-performance numerical computing, probabilistic EDSLs and notebook usage for exploratory data analysis.

The workshop was well-received and we are evaluating various options for upcoming meetings, e.g. to be hosted at either functional programming or data science conferences.

We cherish the open and multidisciplinary nature of our community, and welcome all new users and contributions.

Further reading

datahaskell.org

6.2 Haskell at Eötvös Loránd University (ELTE), Budapest

Report by:	PÁLI Gábor János
Status:	ongoing

Education

There are many different courses on functional programming – mostly taught in Haskell – at Eötvös Loránd University, Faculty of Informatics. Currently, we are offering the following courses in that regard:

- Functional programming for first-year Hungarian undergraduates in Software Technology and second-year Hungarian teacher of informatics students, both as part of their official curriculum.
- An additional semester on functional programming with Haskell for bachelor’s students, where many of the advanced concepts are featured, such as algebraic data types, type classes, functors, monads and their use. This is an optional course for Hungarian undergraduate and master’s students, supported by the Eötvös József Collegium.
- Functional programming for Hungarian and foreign-language master’s students in Software Technology. The curriculum assumes no prior knowledge on the subject in the beginning, then through teaching the basics, it gradually advances to discussion of parallel and concurrent programming, property-based testing, purely functional data structures, efficient I/O implementations, embedded domain-specific languages, and reactive programming. It is taught in both one- and two-semester formats, where the latter employs the Clean language for the first semester.

In addition to these, there is also a Haskell-related course, Type Systems of Programming Languages, taught for Hungarian master’s students in Software Technology. This course gives a more formal introduction to the basics and mechanics of type systems applied in many statically-typed functional languages.

For teaching some of the courses mentioned above, we have been using an interactive online evaluation and testing system, called ActiveHs. It contains several dozens of systematized exercises, and through that,

some of our course materials are available there in English as well.

Our homebrew online assignment management system, "BE-AD" keeps working on for the fourth semester starting from this September. The BE-AD system is implemented almost entirely in Haskell, based on the Snap web framework and Bootstrap. Its goal to help the lecturers with scheduling course assignments and tests, and it can automatically check the submitted solutions as an option. It currently has over 700 users and it provides support for 12 courses at the department, including all that are related to functional programming. This is still in an alpha status yet so it is not available on Hackage as of yet, only on GitHub, but so far it has been performing well, especially in combination with ActiveHs.

Further reading

- Haskell course materials (in English): http://pnyf.inf.elte.hu/fp/Index_en.xml
- Agda tutorial (in English): <http://people.inf.elte.hu/pgj/agda/tutorial/>
- ActiveHs: <http://hackage.haskell.org/package/activehs>
- BE-AD: <http://github.com/andorp/bead>

6.3 Artificial Intelligence and Software Technology at Goethe-University Frankfurt

Report by:	Nils Dallmeyer
Participants:	Manfred Schmidt-Schauß

Semantics of Functional Programming Languages. Extended call-by-need lambda calculi model the semantics of Haskell. We analyze the semantics of those calculi with a special focus on the correctness of program analyses and program transformations. In our recent research, we use Haskell to develop automated tools to show correctness of program transformations, where the method is syntax-oriented and computes so-called forking and commuting diagrams by a combination of several unification algorithms that operate on a meta-representation of the language expressions and transformations.

We therefore developed variants of unification on the meta-representation: An expressive variant that covers all the specifics of normal-order reduction rules. We also developed (less expressive) extensions of nominal unification, which can, however, deal directly with alpha-equivalence: one variant that can deal with recursive lets, another variant that permits variable-variables, and a third one that deals with context-variables and a meta-form of distinct-variable-conditions.

Improvements In recent research we analyzed whether program transformations are optimizations, i.e. whether they improve the time and/or space resource behavior. We showed that common subexpression elimination is an improvement, also under polymorphic typing. We developed methods for better reasoning about improvements in the presence of sharing, i.e. in call-by-need calculi. We also developed a simulation-based method to validate time-improvements respecting the sharing structure. We showed for different transformations that they are space improvements and developed a tool for specific space analyses. Ongoing work is to enhance the techniques to (preferably automatically) verify that program transformations are improvements.

Concurrency We analyzed a higher-order functional language with concurrent threads, monadic IO, MVars and concurrent futures which models Concurrent Haskell. We proved that this language conservatively extends the purely functional core of Haskell. In a similar program calculus we proved correctness of a highly concurrent implementation of Software Transactional Memory (STM) and developed an alternative implementation of STM Haskell which performs quite early conflict detection.

Grammar based compression This research topic focuses on algorithms on grammar compressed data like strings, matrices, and terms. We implemented several algorithms as a Haskell library.

Further reading

<http://www.ki.informatik.uni-frankfurt.de/research/HCAR.html>

6.4 Functional Programming at the University of Kent

Report by:	Olaf Chitil
------------	-------------

The Functional Programming group at Kent is a subgroup of the Programming Languages and Systems Group of the School of Computing. We are a group of staff and students with shared interests in functional programming. While our work is not limited to Haskell – we use for example also Erlang and ML – Haskell provides a major focus and common language for teaching and research.

Our members pursue a variety of Haskell-related projects, several of which are reported in other sections of this report. In September Joanna Sharrad joined us as a new PhD student to work with Meng Wang and Olaf Chitil on type error debugging. For her undergraduate work on delta debugging of type errors she already won the second place in the Student Research Competition at ICFP 2017. Stephen Adams is working

on advanced refactoring of Haskell programs, extending HaRe. Andreas Reuleaux is building in Haskell a refactoring tool for a dependently typed functional language. Maarten Faddegon built the lightweight tracer and algorithmic debugger Hoed for real-world Haskell programs and successfully defended his PhD thesis. Olaf Chitil develops these ideas on tracing further, with a small prototype tracer HatLight for a subset of Haskell and the aim of developing the Haskell tracer Hat further. Dominic Orchard is working on co-effectful programming and applying verification in computational science. He also develops tools in Haskell for analysing, refactoring and verifying Fortran programs. Meng Wang is working on lenses, bidirectional transformation and property-based testing (QuickCheck). Together with last years visitor Colin Runciman from the University of York, Stefan Kahrs is working on minimising regular expressions, implemented in Haskell. Scott Owens is working on verified compilers for the (strict) functional language CakeML. Simon Thompson, Scott Owens, Hugo F  r  e and Reuben Rowe are working on an EPSRC project on trustworthy refactoring. They are studying refactoring for CakeML and OCaml, informed by their previous work for Haskell and Erlang.

We are always looking for more PhD students. We are particularly keen to recruit students interested in programming tools for verification, compilation, tracing, refactoring, type checking and any useful feedback for a programmer. The school and university have support for strong candidates: more details at <http://www.cs.kent.ac.uk/pg> or contact any of us individually by email.

We are also keen to attract researchers to Kent to work with us. There are many opportunities for research funding that could be taken up at Kent, as shown in the website <http://www.kent.ac.uk/researchservices/sciences/fellowships/index.html>. Please let us know if you're interested in applying for one of these, and we'll be happy to work with you on this.

Finally, if you would like to visit Kent, either to give a seminar if you're passing through London or the UK, or to stay for a longer period, please let us know.

Further reading

- o PLAS group: <http://www.cs.kent.ac.uk/research/groups/plas/>
- o Haskell: the craft of functional programming: <http://www.haskellcraft.com>
- o Parsers and static analysis tools for Fortran code in Haskell <https://github.com/camfort/fortran-src>
- o A refactoring and verification tool for Fortran code in Haskell <https://github.com/camfort/camfort>
- o Refactoring Functional Programs: <http://www.cs.kent.ac.uk/research/groups/plas/hare.html>
- o Hoed, a lightweight Haskell tracer and debugger: <https://github.com/MaartenFaddegon/Hoed>

- o Hat, the Haskell Tracer: <http://projects.haskell.org/hat/>
- o CakeML, a verification friendly dialect of SML: <https://cakeml.org>
- o Heat, an IDE for learning Haskell: <http://www.cs.kent.ac.uk/projects/heat/>

6.5 Functional Programming at KU

Report by:	Andrew Gill
Status:	ongoing



Functional Programming continues at KU and the Computer Systems Design Laboratory in ITTC! The System Level Design Group (lead by Perry Alexander) and the Functional Programming Group (lead by Andrew Gill) together form the core functional programming initiative at KU. All the Haskell related KU projects are now focused on use-cases for the remote monad design pattern (\rightarrow 4.12.4). One example is the Haskino Project (\rightarrow 4.13.3).

Further reading

The Functional Programming Group: <http://www.ittc.ku.edu/csdl/fpg>

6.6 Functional Programming Laboratory at the University of Nottingham

Report by:	Jennifer Hackett, Martin Handley
------------	----------------------------------

The School of Computer Science at the University of Nottingham is home to the Functional Programming Laboratory, a research group focused on all theoretical and practical aspects of functional programming, together with related topics such as type theory and category theory. The lab is led by Thorsten Altenkirch and Graham Hutton, with Henrik Nilsson and Venanzio Capretta as academic staff. In addition, we currently have three post-doctoral researchers, eight postgraduate students and one intern.

Our interests are wide-ranging, including:

Mind the Gap: Reasoning about Efficiency

The Functional Programming Lab is currently home to an EPSRC grant on reasoning about the efficiency of programs. Jennifer Hackett and Graham Hutton are currently preparing a paper for ICFP that presents a cost-aware parametricity theorem that produces free theorems that can be used to reason about time costs, alongside another paper exploring an approach to reasoning about efficiency based on metric spaces. Also, Martin Handley and Graham Hutton have developed a tool that will assist programmers to reason inequationally about the cost behaviour of their programs. A paper on this tool has been submitted to the upcoming European Symposium on Programming.

Generalizing Monads and Applicative Functors

Jan Bracker and Henrik Nilsson are working on unifying generalizations of monads and applicative functors. Their work has produced the `supermonads` library (→ 4.1.3) which offers a unified way to work with these generalized notions in Haskell (details in the independent entry). In addition to the practical support of these notions they are also exploring the theoretical foundations of said notions in category theory. Future work may involve generalizing arrows as well.

Functional Reactive Programming

Functional Reactive Programming (FRP) remains an active part of our research. Henrik Nilsson and Ivan Perez have worked on monadic FRP, FRP extensions such as time transformations, property-based testing and debugging for FRP, Reactive Values and Relations (RVR). We have also worked on applications of these technologies; e.g., to games and music. In particular, together with summer interns Gueric Chupin and Jin Zhan, we developed the Arpeggigon (→ 4.17.4), a Functional Reactive Musical Automaton, as a substantial test case for FRP and RVR that is also an interesting and fun application in its own right. This has led to a number of publications and talks (PADL, ICFP, Haskell Symposium, FARM, London Haskell Meetup, Haskell eXchange, Haskell in Leipzig) as well as open source software releases. The Yampa (→ 4.16.1) FRP implementation is also actively being maintained.

Jonathan Thaler is applying Functional Reactive Programming to agent-based simulation (→ 4.16.3) with the aim to create more robust and correct implementations.

Teaching

The lab plays an active role in teaching, with Haskell playing an important role in many of the modules offered. Modules include: Programming, Programming

Paradigms, Advanced Functional Programming, Machines and Their Languages, Compilers, Introduction to Formal Reasoning, Advanced Algorithms and Data Structures, Languages and Computation and Foundations of Programming.

Other Activities

Every Friday afternoon we hold the FP Café, an informal meeting where we hold whiteboard discussions on current activities, general business or anything that sounds interesting. Visitors are welcome: if you want to come along (perhaps even to give a talk yourself), please contact us!

The lab plays a leading role in the Midlands Graduate School in the Foundations of Computing Science.

Further reading

<http://www.nottingham.ac.uk/research/groups/fp-lab/>

6.7 fp-syd: Functional Programming in Sydney, Australia

Report by:	Erik de Castro Lopo
Participants:	Ben Lippmeier, Shane Stephens, and others

We are a seminar and social group for people in Sydney, Australia, interested in Functional Programming and related fields. Members of the group include users of Haskell, Ocaml, LISP, Scala, F#, Scheme and others. We have 10 meetings per year (Feb–Nov) and meet on the fourth Wednesday of each month. We regularly get 40–50 attendees, with a 70/30 industry/research split. Talks this year have included material on compilers, theorem proving, type systems, Haskell web programming, dynamic programming, Scala and more. We usually have about 90 mins of talks, starting at 6:30pm. All welcome.

Further reading

- o <http://groups.google.com/group/fp-syd>
- o <http://fp-syd.ouroborus.net/>
- o <http://fp-syd.ouroborus.net/wiki/Past/2016>

6.8 Regensburg Haskell Meetup

Report by: Andres Löh

Since autumn 2014 Haskellers in Regensburg, Bavaria, Germany have been meeting roughly once per month to socialize and discuss Haskell-related topics.

We usually have dinner first and then move on to have a talk. Topics vary quite a bit, from introductory to advanced, from theoretical to practical, and we have been looking at other languages such as Scala or dependently typed languages as well.

There are typically between 5 and 15 attendees, and we often get visitors from Munich and Nürnberg.

New members are always welcome, whether they are Haskell beginners or experts. If you are living in the area or are visiting, please join! Meetings are announced a few weeks in advance on our meetup page: <http://www.meetup.com/Regensburg-Haskell-Meetup/>.

6.9 Curry Club Augsburg

Report by: Ingo Blechschmidt
Status: active

Since March 2015 haskellistas, scalafists, lambdroids, and other fans of functional programming languages in Augsburg, Bavaria, Germany have been meeting every four weeks in the OpenLab, Augsburg's hacker space. Usually there are ten to twenty attendees.

At each meeting, there are typically two to three talks on a wide range of topics of interest to Haskell programmers, such as latest news from the Kmettiverse and introductions to the category-theoretic background of freer monads. Afterwards we have stimulating discussions while dining together.



From time to time we offer free workshops to introduce new programmers to the joy of Haskell.

Newcomers are always welcome! Recordings of our talks are available at <http://www.curry-club-augsburg.de/>.

Further reading

<http://www.curry-club-augsburg.de/>

6.10 Italian Haskell Group

Report by: Francesco Ariis
Status: ongoing

Born in Summer 2015, the Italian Haskell Group is an effort to advocate functional programming and share our passion for Haskell through real-life meetings, discussion groups and community projects.

There have been 3 meetups (in Milan, Bologna and Florence), our plans to continue with a quarterly schedule. Anyone from the experienced hacker to the functionally curious newbie is welcome; during the rest of the year you can join us on our irc/mumble channel for haskell-related discussions and activities.

Further reading

- site: <http://haskell-ita.it/>
- IRC channel: <https://webchat.freenode.net/?channels=%23haskell.it>
- Discussion forum : https://groups.google.com/forum/#!forum/haskell_ita

6.11 RuHaskell – the Russian-speaking haskellers community

Report by: Yuriy Syrovetskiy
Status: active

RuHaskell is the Russian-speaking community of haskellers. We have a website with Haskell-related articles, a podcast, a subreddit and some Gitter chats including one for novice haskellers specially. We also organize mini-conferences about twice a year in Moscow, Russia. The 4th mini-conference has taken place in the April, 2017.

Further reading

- Short info: <https://wiki.haskell.org/RuHaskell>
- Website: ruhaskell.org
- Gitter chats: [/ruhaskell/home](https://gitter.im/ruhaskell/home)
- Twitter channel: [@ruhaskell](https://twitter.com/ruhaskell)
- Subreddit: [/r/ruhaskell](https://www.reddit.com/r/ruhaskell)

6.12 NY Haskell Users Group and Compose Conference

Report by: Gershon Bazerman
Status: ongoing

Since 2012 the NY Haskell Users Group has been hosting monthly Haskell talks and the occasional hackathon. Over fifteen-hundred members are registered on Meetup for the group, and talk attendance ranges between sixty to one hundred and twenty. NY-HUG has also been organizing, on and off, beginner-oriented hangouts where people can assemble and study

and learn together. And as of recently, NYHUG has also been the home base for organizing a Haskell Programming from First Principles study group, as well as an active Slack channel where ongoing discussion for the reading group takes place.

In 2015, the NY Haskell organizers launched the Compose Conference, which was held again in 2016, with a sibling “Compose::Melbourne” conference being held in 2016 as well. Compose is a cross-language conference for functional programmers, focused on strongly-typed functional languages such as Haskell, OCaml, F#, and SML. It aims to be both practical and educational, among other things providing opportunity for researchers to present the more applicable elements of their work to a wide audience of professional and hobbyist functional programmers. It is our hope to continue Compose and also to extend it to sibling conferences in other geographic areas as well sharing similar goals and format.

Further reading

- <http://www.meetup.com/NY-Haskell/>
- <http://www.composeconference.org/>

6.13 Japan Haskell User Group – Haskell-jp

Report by:	Yuji Yamamoto
Status:	active

Japan Haskell User Group (a.k.a Haskell-jp) is a Haskellers’ community group in Japan, established at April 2017.

Since it began, we have tried various activities to help Haskellers discuss, collect, learn, and deliver any information related to Haskell. Today, let me introduce you some of the fun activities:

Online Discussion Space

We have a Slack Workspace. Most of the messages are archived in SlackArchive. You can register too.

You can also check out our Subreddit.

We also have a public Haskell-jp Trello Board. If you have a request, suggest it in the Slack workspace.

Haskell-jp Blog

The Haskell-jp Blog is set up such that anyone can post articles broadly related to Haskell. We always welcome new posts. You can contribute by sending a pull request to <https://github.com/haskell-jp/blog>.

Some popular articles (written in Japanese):

- <https://haskell.jp/blog/posts/2017/12-ghc-show-info.html>
- <https://haskell.jp/blog/posts/2017/10-about-kind-system-part1.html>

- <https://haskell.jp/blog/posts/2017/13-about-kind-system-part2.html>

Haskell-jp Mokumoku-kai

Haskell-jp Mokumoku-kai is a monthly meetup for anyone who wants to do something with the Haskell programming language. Anyone, including non-Japanese speakers, can participate.

Both Haskell-jp Mokumoku-kai and Haskell-jp originate from Haskell Mokumoku-kai.

We have an after party which is now free for students.

The next meetup is the 48th in our series and is on Nov. 26.

Haskell-jp wiki

Haskell-jp wiki is a wiki in Japanese, powered by gitit and Heroku. Edits are unrestricted, anyone owning GitHub account can edit.

Haskell Antenna

Haskell Antenna, inspired by Haskell News, is a very experimental (visible if you’re lucky when accessing it) RSS feed aggregator. I say it is very experimental because it is run on my (@lotz84) personal Heroku account.

You can add an RSS feed to display by sending a pull request to <https://github.com/haskell-jp/antenna>.

Future plans

- Create an introduction of Haskell in Japanese.
 - In development.
 - Will be one of the biggest contents with the blog.
- Establish a community committee.
 - Haskell-jp should be casual, slack group of Haskell users.
 - But to accelerate its activities, we should clarify each member’s role and responsibility.

As written in the last edition of HCAR, our largest mission is to provide spaces via the haskell.jp domain.

Further reading

- <https://haskell.jp/blog>
- <https://haskell.jp/antenna>
- <https://wiki.haskell.jp/>
- https://www.reddit.com/r/haskell_jp/
- <https://haskell-jp.slack.com>
- <https://github.com/haskell-jp>

6.14 Tokyo Haskell Meetup – Casual, English-speaking monthly meetings in Tokyo

Report by:	Yasuaki Kudo
Status:	active

Tokyo Haskell Meetup is a group that meets mostly every month in Tokyo. English is the dominant language and we enjoy casual and lively discussions of Haskell and Functional Languages, (and others including Philosophy, Politics, Business, etc.), while helping each other learn Haskell. Our members proficiency in Haskell is very diverse – from the beginners (such as myself) to the experts.

Further reading

<https://www.meetup.com/Tokyo-Haskell-Meetup>

6.15 Functional Programming at the Telkom University

Report by:	Wisnu Adi Nurcahyo
------------	--------------------

Functional programmers are rare to find in Indonesia, especially for Haskell where they are less than 30 from hundreds of thousands programmers that the country has.

I started a functional programming group at Telkom University. My goal is to create a great community of functional programmers starting from university.

Contact

wisnu@nurcahyo.me

6.16 Haskell Serbia

Report by:	Sasa Bogicevic
Status:	Active

Haskell-serbia is website for the Haskell user group in Serbia.

The idea is to organize meetups, write tutorials and generally spread the Haskell programming language in Serbia.

It is in active development and website is up and running.

There are plans to add fresh new look to website, customize user profile pages, write more tutorials and generally get more people involved.

Further reading

<https://haskell-serbia.com>