Haskell Communities and Activities Report

http://tinyurl.com/haskcar

Thirty Second Edition — May 2017

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

Preface

This is the 32nd edition of the Haskell Communities and Activities Report. It comes shortly after the 31st edition because that one was delayed until December. It is my intention to keep the next editions on schedule.

This report has 125 entries, 67 of which were touched between December and today. Out of these, 16 are 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 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 September/early October.

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, Leap Year Technologies, US
⟨hcar@haskell.org⟩

Contents

1 Community

1.1 Haskell' — Haskell 2020

Report by:	Herbert Valerio Riedel
Participants:	Andres Löh, Antonio Nikishaev, 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 skillset. 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 three-part addition:

Implementing Explicit and Finding Implicit Sharing in Embedded DSLs

The web page collects the materials for the tutorial on sharing and common subexpression elimination in embedded domain-specific language compilers. The tutorial was presented at DSL 2011: IFIP Working Conference on Domain-Specific Languages (6-8 September 2011, Bordeaux, France). The material includes the extensively commented Haskell source code, with solutions to selected exercises.

As the running example, the tutorial uses a DSL of arithmetic expressions with the strong flavor of hardware description languages. In fact, the output of the DSL compilation is essentially a netlist (the list of primitive gates and connections among them). The tutorial reviews several ways to introduce sharing, so important in hardware and so difficult to represent in functional programming. The DSL approach advocated in the tutorial is probably the best compromise, hiding the essentially imperative nature of sharing.

Read the tutorial online.

Explicit sharing across several expressions

We demonstrate how to add explicit sharing – a `let`-like form – to a DSL without introducing large notational inconvenience and, specifically, without rewriting the code in a monadic style. Recall that in pure functional code sharing is not observable: it is an imperative feature. The challenge of explicit sharing in an embedded DSL was posed on Haskell-Cafe, back in 2008.

Read the tutorial online.

Historical note on hash consing

Hash-consing reminds one of lisp (where it has been used widely). However, hash-consing has been invented before lisp! Hash-consing – as well as hash functions, investigation of their properties and the algorithm for compiling arithmetic expressions (still used, e.g., in OCaml compiler) – were first described in the paper by A.P.Ershov published in 1958. Although published in Russian, the English translation appeared in Communications of the ACM later that year: the unheard of publishing speed for those times.

Read the tutorial online.

2.2 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.

Since the last HCAR, 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/

2.3 Haskell Programming from first principles, a book forall

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

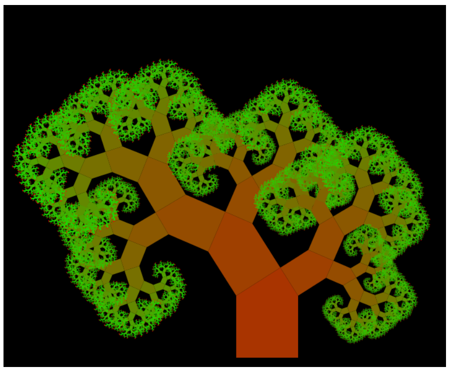
2.4 Learning Haskell

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.



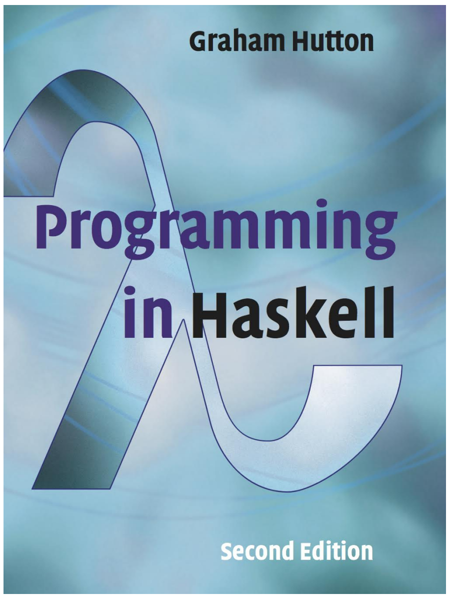
Further chapters will be made available as we complete them.

Further reading

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

2.5 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

2.6 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:	First run of a six-week online Haskell course has just completed

The School of Computing Science at the University of Glasgow 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 first run of the course completed on 30 Oct 2016. Over 6000 people signed up for the course, 50% of whom actively engaged with the materials. Around 800 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, probably in Sep–Oct 2017. Visit our site to register your interest.

We hope to refine the learning materials, based on the first run of the course. We also intend to write up our experiences as a scholarly report.

Further reading

https://www.futurelearn.com/courses/functional-programming-haskell/1

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 8.2

By the time of publication GHC will be nearing release of its 8.2.1 release. While GHC 8.0 was a feature-oriented release, the past year of GHC development has been focused on stabilization and consolidation of existing features. This has included an effort to reduce compilation times, improve code generation, and place the levity polymorphism story introduced in GHC 8.0 on a stable theoretical footing.

While priorities for the 8.4 release have not yet been discussed, they will very likely include further focus on compilation time reduction, improved error messages, and additional improvements to code generation.

Major changes in GHC 8.2

While the emphasis of 8.2 is on performance, stability, and consolidation, it also includes a number of new features.

Libraries, source language, and type system

- **Indexed Typeable representations.** While GHC has long supported runtime type reflection through the `Typeable` typeclass, its current incarnation requires careful use, providing little in the way of type-safety. For this reason the implementation of types like `Data.Dynamic` must be implemented in terms of `unsafeCoerce` with no compiler verification.

  GHC 8.2 will address this by introducing indexed type representations, leveraging the type-checker to verify many programs using type reflection. This allows facilities like `Data.Dynamic` to be implemented in a fully type-safe manner. See the paper[1] for a description of the proposed interface and the Wiki[2] for current implementation status.

- **Backpack.** Backpack has merged with GHC, Cabal and `cabal-install`, allowing you to write libraries which are parametrized by signatures, letting users decide how to instantiate them at a later point in time. If you want to just play around with the signature language, there is a new major mode `ghc -backpack`; at the Cabal syntax level, there are two new fields `signatures` and `mixins` which permit you to define parametrized packages, and instantiate them in a flexible way. More details are on the Backpack wiki page[3].

- **Levity polymorphism.** GHC 8.0 reworked GHC's kind system introducing the notion of levity to describe a type's runtime representation. However, the ability to write functions which were polymorphic over levity was purposefully withheld.

  While GHC's current compilation model doesn't allow arbitrary levity polymorphism, GHC 8.2 enables certain classes of polymorphism which were either disallowed or broken in 8.0. See the paper for details [4].

- **`deriving` strategies.** GHC now provides the programmer with a precise mechanism to distinguish between the three ways to derive typeclass instances: the usual way, the `GeneralizedNewtypeDeriving` way, and the `DeriveAnyClass` way. See the `DerivingStrategies` Wiki page for more details [5].

- **New classes in `base`.** The `Bifoldable`, and `Bitraversable` typeclasses are now included in the `base` library.

- **Unboxed sums.** GHC 8.2 has a new language extension, `UnboxedSums`, that enables unboxed representation for non-recursive sum types. GHC 8.2 doesn't use unboxed sums automatically, but the extension comes with new syntax, so users can manually unpack sums. More details can be found in the Wiki page [6].

Runtime system

- **Compact regions**. This runtime system feature allows a referentially "closed" set of heap objects to be collected into a "compact region," allowing cheaper garbage collection, sharing of heap-objects between processes, and the possibility of inexpensive serialization. See the paper [7] for details.

- **Better profiling support.** The cost-center profiler now better integrates with the GHC event-log. Heap

[1] https://research.microsoft.com/en-us/um/people/simonpj/papers/haskell-dynamic/
[2] https://ghc.haskell.org/trac/ghc/wiki/Typeable/BenGamari
[3] https://ghc.haskell.org/trac/wiki/Backpack
[4] https://www.microsoft.com/en-us/research/publication/levity-polymorphism/
[5] https://ghc.haskell.org/trac/wiki/DerivingStrategies
[6] https://ghc.haskell.org/trac/ghc/wiki/UnpackedSumTypes
[7] http://ezyang.com/papers/ezyang15-cnf.pdf

profile samples can now be dumped to the event log, allowing heap behavior to be more easily correlated with other program events. Moreover, the cost center stack output (e.g. `.prof` files) can now be produced in a machine-readable JSON format for easier integration with external tooling.

- **More robust DWARF output.** GHC 8.2 will be the first release with reliable support for DWARF debugging information. A number of bugs leading to incorrect debug information for foreign calls have been fixed, meaning it should now be safe to enable debugging information in production builds.

  With stable DWARF support comes a number of opportunities for new performance analysis and debugging tools (e.g. statistical profiling, cheap execution stacks). As GHC's debugging information improves, we expect to see tooling developed to support these applications. See the DWARF status page on the GHC Wiki[8] for further information.

- **Better support for NUMA platforms.** Machines with non-uniform memory access costs are becoming more and more common as core counts continue to rise. The runtime system is now better equipped to efficiently run on such systems.

- **Experimental changes to the scheduler** that enable the number of threads used for garbage collection to be lower than the `-N` setting.

- **Support for `StaticPointers` in GHCi.** At long last programs making use of the `StaticPointers` language extension will have first-class bytecode interpreter support, allowing such programs to be loaded into GHCi.

- **Reduced CPU usage at idle.** A long-standing regression resulting in unnecessary wake-ups in an otherwise idle program was fixed. This should lower CPU utilization and improve power consumption for some programs.

Miscellaneous

- **Compiler Determinism.** GHC 8.0.2 is the first release of GHC which produces deterministic interface files. This helps consumers like Nix and caching build systems, and presents new opportunities for compile-time improvements. See the Wiki [9] for details.

Development updates and acknowledgments

Since the 8.0 release nearly a year ago, GHC's development community has continued to grow. Over the last year we have seen sixty new developers submit patches, many of whom have become regular contributors. These include new generations of researchers, as well as, increasingly, regular Haskell users from industry and elsewhere. GHC has had an exciting few months...

Moritz Angermann has been hard at work improving cross-compilation and bringing remoting support to the `-fexternal-interpreter` feature introduced in GHC 8.0. He has also contributed a number of patches improving support for iOS and Android targets and has been pondering how to enable Template Haskell during cross-compilation.

Ryan Scott and Matthew Pickering have been fixing bugs all over the compiler, in addition to their respective work on deriving strategies and the new `COMPLETE` pragma mentioned above.

Tamar Christina has been hard at work as our resident Windows expert. In addition to introducing split sections support, reworking dynamic linking, fixing toolchain breakage due to the recent Windows 10 Creator's Update, and maintaining GHC packaging for the Chocolatey package manager, he is also an invaluable source of knowledge.

Andrey Mokhov and Jose Calderon have been continuing work on Hadrian, GHC's new build system built on Shake. Hadrian is already very usable, lacking only in support for a some non-standard build configurations, building documentation, and producing binary and source distributions. These would be great, fairly self-contained projects for any interested developer with a Haskell experience. Contact Andrey if any of these sound intriguing to you.

Michal Terepeta has been performing a long-overdue grooming of GHC's `nofib` benchmark suite. This suite is one of the primary windows into the performance of the compiler, yet has fallen into somewhat of a state of disrepair. Michal has been fixing broken tests, refactoring the build system, and generally making `nofib` a nicer place. In addition, he has been gradually cleaning up native code generator, revisiting GHC's use of the `hoopl` library. Also working in the native code generator is Thomas Jakway, who has been working on improving the register allocator's treatment of loops.

Another contributor working on runtime performance is Luke Maurer, who has contributed a rework of GHC's treatment of join points, one of the major features of 8.2. Join points are an essential optimization which allows GHC to eliminate closure allocation for some types of calls. Prior to GHC 8.2 the join points optimization was fragile due to the lack of a sound theoretical footing. With Luke's work, GHC is much better at preserving join point opportunities, leading to improved code generation in many performance critical settings. More details can be found in his paper [10].

As always, this is just a fraction of the contributions which we've seen in the past months. GHC improves

[8] https://ghc.haskell.org/trac/ghc/wiki/DWARF/Status
[9] https://ghc.haskell.org/trac/ghc/wiki/DeterministicBuilds
[10] https://www.microsoft.com/en-us/research/publication/compiling-without-continuations

through the efforts of everyone who offers patches, bug reports, code review, and discussion. If you have contributed any of these in the past year, thank you!

GHC HQ has also taken a number of steps in the past months to try to improve contributor workflow. Our new proposal process[11] officially began in December and has been extremely active ever since: the `ghc-proposals` repository now has over fifty pull requests and has facilitated hundreds of messages of great discussion. Thanks to our everyone who has participated! GHC also now accepts GitHub pull requests for small changes to the user guide and library documents, which several contributors have made use of this. We hope that this channel will make it easier for users to contribute and perhaps feel compelled to pick up larger tasks in the future.

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

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

[11] https://github.com/ghc-proposals/ghc-proposals

3.2 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.

3.3 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

3.4 Specific Platforms

3.4.1 Fedora Haskell SIG

Report by:	Jens Petersen
Status:	active

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

For the coming Fedora 26 release ghc has been updated to 8.0.2 and all the Haskell packages have been rebuilt and most updated to LTS 8.2 (except pandoc and hledger, due to missing dependencies). For current releases there is a ghc-8.0.2 Fedora Copr repo available for Fedora and EPEL 7, and also a Fedora Copr repo for stack. We use the cabal-rpm packaging tool to create and update Haskell packages, and improved fedora-haskell/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

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

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 package and package a wide range of other useful libraries and programs. At the time of writing, we maintain 995 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 ("jessie") provides the Haskell Platform 2013.2.0.0 and GHC 7.6.3, with GHC 7.10.3 being available via "jessie-backports". In Debian unstable and testing (the soon-to-be released release "stretch") we ship GHC 8.0.1.

Debian users benefit from the Haskell ecosystem on 17 architecture/kernel combinations, including the non-Linux-ports KFreeBSD and Hurd.

Further reading

http://wiki.debian.org/Haskell

3.5 Related Languages and Language Design

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:

- Agda 2.5.2 was released in December 2016.
- The Agda documentation at http://agda.readthedocs.org/en/stable/ is being continuously improved.
- Experimental support for homotopy type theory has been added to the developement branch by Andrea Vezzosi.

Release of Agda 2.5.3 is planned for summer 2017.

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 information 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 effectful/imperative style, and one of our goals is to provide both styles coherently in the same language.

What is new?

DDC is in an experimental, pre-alpha state, though parts of it do work. In September this year we released DDC 0.4.3, with the following new features:

- Completed desugaring of pattern alternatives.
- Better type inference for higher ranked types, which allows explicit dictonaries for Functor, Applicative, Monad and friends to be written easily.
- Automatic insertion of run and box casts is now more well baked.
- Automatic interrogation of LLVM compiler version and generation of matching LLVM assembly syntax.
- Added code generation for partial applications of data constructors.
- Added support for simple type synonyms.

Further reading

http://disciple.ouroborus.net

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:	Henrik Nilsson
Status:	Experimental fully working version

The supermonad package provides a unified way to represent different monadic notions. In other words, it provides a way to use standard and generalized monads (with additional indices or constraints) with each other without having to manually disambiguate which notion is referred to in every computation. To achieve this, the library represents monads as a set of two type classes that are general enough to allow instances for all of the different 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. We are currently working on providing the same kind of support for applicative functors and arrows, so that generalizations of these notions can be used as freely as the different notions of monads.

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

4.2 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_PACKAGENAME`).
- 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. Two 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, and Luke Murphy.

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 8.0.2, as well as released LTS 8 based on that same GHC release. We continue to make releases of LTS 6 and 7 concurrently, which are based on GHC 7.10.3 and 8.0.1, 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 always takes you to the latest lts page of the package.

Further reading

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

4.2.5 hsinstall

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. hsinstall 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:

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

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

- fhs is the more traditional UNIX structure like this:

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

There are two parts to hsinstall 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 hsinstall 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

- hsinstall on darcshub
  http://hub.darcs.net/dino/hsinstall
- hsinstall on Hackage
  https://hackage.haskell.org/package/hsinstall
- hsinstall on Stackage
  https://www.stackage.org/package/hsinstall

4.2.6 `cab` — A Maintenance Command of Haskell Cabal Packages

Report by:	Kazu Yamamoto
Status:	open source, actively developed

`cab` is a MacPorts-like maintenance command of Haskell cabal packages. Some parts of this program are a wrapper to `ghc-pkg` and `cabal`.

If you are always confused due to inconsistency of `ghc-pkg` and `cabal`, or if you want a way to check all outdated packages, or if you want a way to remove outdated packages recursively, this command helps you.

Since the last HCAR, Cabal 2.0 was supported thanks to Ryan Scott.

Further reading

http://www.mew.org/~kazu/proj/cab/en/

4.2.7 yesod-rest

Report by:	Sibi Prabakaran
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

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

4.3 Repository Management

4.3.1 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

- https://github.com/stackbuilders/octohat
- Octohat announcement
- Octohat update

4.3.2 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.

In April 2016, we released Darcs 2.12, with minors update in September 2016 and January 2017 to fix compiling with GHC 8 and other minor bugs. This new major release includes the new `darcs show dependencies` command (for exporting the patch dependencies graph of a repository to the Graphviz format), improvements for Git import, and improvements to `darcs whatsnew` to facilitate support of Darcs by third-party version control front ends like Meld and Diffuse.

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

- http://darcs.net
- http://darcs.net/Releases/2.12
- http://hub.darcs.net

4.3.3 git-annex

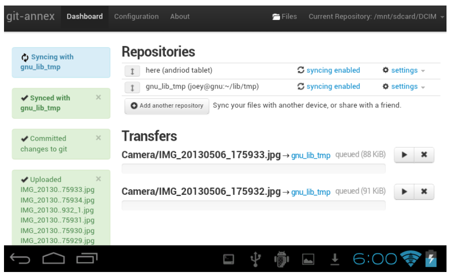
Report by:	Joey Hess
Status:	stable, actively developed

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.



Syncing with gnu_lib_tmp

Synced with gnu_lib_tmp

Committed changes to git

Uploaded
IMG_20130..75933.jpg
IMG_20130..75934.jpg
IMG_20130..932_1.jpg
IMG_20130..75931.jpg
IMG_20130..75930.jpg
IMG_20130..75929.jpg

Repositories

here (andriod tablet)	syncing enabled	settings
gnu_lib_tmp (joey@gnu:~/lib/tmp)	syncing enabled	settings

Sync your files with another device, or share with a friend.

Transfers

Camera/IMG_20130506_175933.jpg → gnu_lib_tmp queued (88 KiB)

Camera/IMG_20130506_175932.jpg → gnu_lib_tmp queued (91 KiB)

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/

4.3.4 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 = printO (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
```

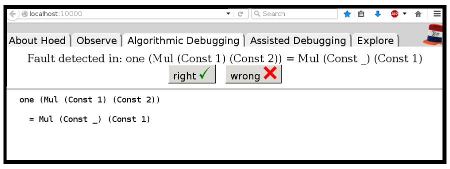
```
plusOne :: Int -> Int
plusOne = observe "plusOne" plusOne'
plusOne' n = n + 1

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 = printO (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.



Fault detected in: one (Mul (Const 1) (Const 2)) = Mul (Const _) (Const 1)

```
one (Mul (Const 1) (Const 2))
  = Mul (Const _) (Const 1)
```

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 : _thunk x1 [False]
in x1
> take 3 x
[True,False,True]
> :printHeap x
let x1 = True : False : x1
in x1
```

The graphical tool ghc-vis (→ 4.4.3) builds on ghc-heap-view.

Since version 0.5.6, ghc-heap-view supports GHC 8.

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-GHC.HeapView.html
- http://www.joachim-breitner.de/blog/archives/590-Evaluation-State-Assertions-in-Haskell.html

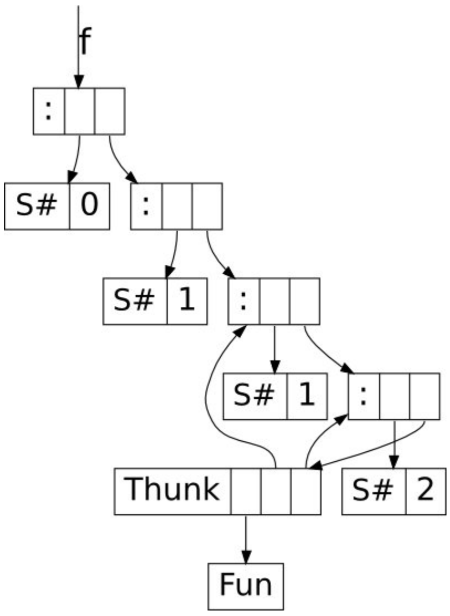
4.4.3 ghc-vis

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 works with 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. 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.

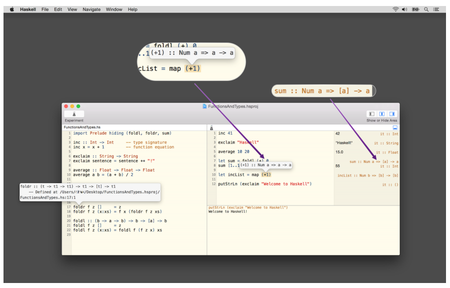
Further reading

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

4.5 Development Tools and Editors

4.5.1 Haskell for Mac

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



```
import Prelude hiding (foldl, foldr, sum)

inc :: Int -> Int       -- type signature
inc x = x + 1           -- function equation

exclaim :: String -> String
exclaim sentence = sentence ++ "!"

average :: Float -> Float -> Float
average a b = (a + b) / 2

foldr :: (t -> t1 -> t1) -> t1 -> [t] -> t1
    -- Defined at /Users/[illegible]/Desktop/FunctionsAndTypes.hsproj/
FunctionsAndTypes.hs:17:1

foldr f z []     = z
foldr f z (x:xs) = f x (foldr f z xs)

foldl :: (b -> a -> b) -> b -> [a] -> b
foldl f z []     = z
foldl f z (x:xs) = foldl f (f z x) xs
```

```
inc 41                                   42          it :: Int

exclaim "Haskell"                        "Haskell!"  it :: String

average 10 20                            15.0        it :: Float

let sum = foldl (+) 0                    sum :: Num a => [a] -> a
sum [1..1(+1) :: Num a => a -> a         55          it :: Int

let incList = map (+1)                   incList :: Num b => [b] -> [b]

putStrLn (exclaim "Welcome to Haskell")  it :: ()
```

```
putStrLn (exclaim "Welcome to Haskell")
Welcome to Haskell!
```

```
(+1) :: Num a => a -> a
cList = map (+1)
sum :: Num a => [a] -> a
```

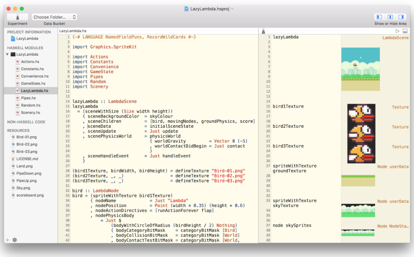
Haskell for Mac is an easy-to-use, innovative programming environment and learning platform for Haskell on OS X. It includes its own Haskell distribution and requires no further set up. It features interactive Haskell playgrounds to explore and experiment with code. Playground code is not only type-checked, 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
{-# LANGUAGE NamedFieldPuns, RecordWildCards #-}

import Graphics.SpriteKit

import Actions
import Constants
import Convenience
import GameState
import Pipes
import Random
import Scenery


lazyLambda :: LambdaScene
lazyLambda
  = (sceneWithSize (Size width height))
    { sceneBackgroundColor  = skyColour
    , sceneChildren         = [bird, movingNodes, groundPhysics, score]
    , sceneData             = initialSceneState
    , sceneUpdate           = Just update
    , scenePhysicsWorld     = physicsWorld
                              { worldGravity         = Vector 0 (-5)
                              , worldContactDidBegin = Just contact
                              }
    , sceneHandleEvent      = Just handleEvent
    }

(bird1Texture, birdWidth, birdHeight) = defineTexture "Bird-01.png"
(bird2Texture, _, _)                  = defineTexture "Bird-02.png"
(bird3Texture, _, _)                  = defineTexture "Bird-03.png"

bird :: LambdaNode
bird = (spriteWithTexture bird1Texture)
       { nodeName             = Just "Lambda"
       , nodePosition         = Point (width * 0.35) (height * 0.6)
       , nodeActionDirectives = [runActionForever flap]
       , nodePhysicsBody
           = Just $
               (bodyWithCircleOfRadius (birdHeight / 2) Nothing)
               { bodyCategoryBitMask    = categoryBitMask [Bird]
               , bodyCollisionBitMask   = categoryBitMask [World]
               , bodyContactTestBitMask = categoryBitMask [World,
```

```
lazyLambda                      LambdaScene
bird1Texture                    Texture
bird2Texture                    Texture
bird3Texture                    Texture
spriteWithTexture groundTexture Node userData
spriteWithTexture skyTexture    Node userData
node skySprites                 Node NodeSta...
```

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

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

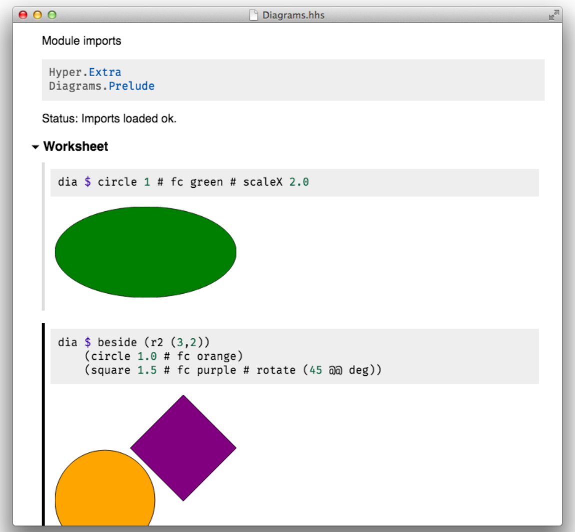
4.5.3 HyperHaskell – The strongly hyped Haskell interpreter

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:



Module imports

```
Hyper.Extra
Diagrams.Prelude
```

Status: Imports loaded ok.

Worksheet

```
dia $ circle 1 # fc green # scaleX 2.0
```

```
dia $ beside (r2 (3,2))
    (circle 1.0 # fc orange)
    (square 1.5 # fc purple # rotate (45 @@ deg))
```

Current status

HyperHaskell is currently *Level α*. Compared to the previous report, no new release has been made, but basic features are working. A new cell type, the *text cell*, has been implemented, but not yet released.

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 "nonverbally" 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*.

Further reading

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

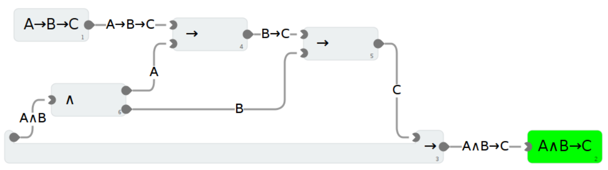
4.6 Formal Systems and Reasoners

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 spagetthi 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 $\rightarrow$ 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-exts`. 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.

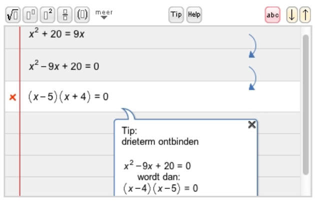
Contact

⟨J.Hage@uu.nl⟩

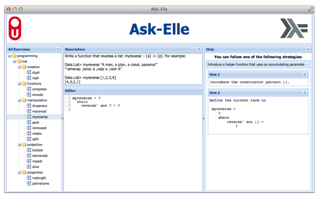
4.7.2 Interactive Domain Reasoners

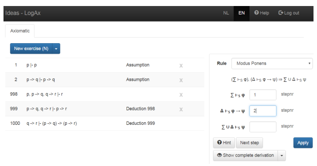
Report by:	Bastiaan Heeren
Participants:	Johan Jeuring, Alex Gerdes, Josje Lodder, Hieke Keuning, Ivica Milovanovic
Status:	experimental, active development

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.

4.7.3 DSLsofMath

Report by:	Patrik Jansson
Participants:	Cezar Ionescu, Irene Lobo Valbuena, Adam Sandberg Ericsson
Status:	active development

"Domain Specific Languages of Mathematics" is a project at Chalmers University of Technology developing a new BSc level course and accompanying material for learning and applying classical mathematics (mainly 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; and, finally, to organize the resulting functions and types in domain-specific languages.

The first instance of the course was carried out Jan-March 2016 at Chalmers and the course material is available on github.

The next step is to write up the lecture notes as a book during the autumn, in preparation for the next instance of the course early 2017.

Contributions and ideas are welcome!

Further reading

- DSLsofMath (github organisation)
- TFPIE 2015 paper
- Exam 2016 with solutions

4.8 Text and Markup

4.8.1 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 not released on hackage yet; for a description of how to build it see the repository README.

Further reading

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

4.8.2 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 LATEX 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/kosmikus/lhs2tex

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/llelf/prose
- https://github.com/llelf/numerals

4.8.4 Lentil

Report by:	Francesco Ariis
Status:	working

Lentil helps the programmers who litter their code with `TODO`s and `FIXME`s.

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

4.8.5 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

4.8.6 Ginger

Report by:	Tobias Dammers
Status:	Active, usable, not feature complete

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 %}
        <img class="page-image"
             src="{{ page.image.thumbURL }}" />
    {% 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 unityped 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://bitbucket.org/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, Warp improved some exception handling.

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 Mighttpd2 — Yet another Web Server

Report by:	Kazu Yamamoto
Status:	open source, actively developed

Mighttpd (called mighty) version 3 is a simple but practical Web server in Haskell. It provides features to handle static files, redirection, CGI, reverse proxy, reloading configuration files and graceful shutdown. Also TLS is supported.

The logic to handle static files has been transferred to Warp, an HTTP server library. So, Mighttpd became a simpler web application now.

You can install Mighttpd 3 (*mighttpd2*) from HackageDB. Note that the package name is *mighttpd2*, not *mighttpd3*, for historical reasons.

Since the last HCAR, Mighttpd provided the Tls_Chain_Files configuration option to support certificates generated by Let's encrypt and changed to return a bad request for paths including dot files for security reasons.

Further reading

- http://www.mew.org/~kazu/proj/mighttpd/en/

4.9.4 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 scalablity comes from the amazing GHC compiler and runtime. GHC provides fast code and built-in evented 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 shakesperean 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 CoffeScript. 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/

4.9.5 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

4.9.6 Snap Framework

Report by:	Doug Beardsley
Participants:	Gregory Collins, Shu-yu Guo, James Sanders, Carl Howells, Shane O'Brien, Ozgun Ataman, Chris Smith, Jurrien Stutterheim, Gabriel Gonzalez, 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.

This HCAR we are pleased to announce the release of Snap 1.0. This major milestone includes a rewrite of Snap's web server. The new web server is built on io-streams, a simple and fast stream library designed specifically for this purpose. It is also faster and more robust with 100% test coverage.

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

- Snaplet Directory:
  http://snapframework.com/snaplets
- http://snapframework.com
- io-streams:
  http://hackage.haskell.org/package/io-streams
- snap-server test coverage: https://snapframework.github.io/snap-code-coverage/snap-server/hpc-ghc-8.0.1/hpc_index.html
- Snap 1.0 release announcement::
  http://snapframework.com/blog/2016/08/07/snap-1.0-released

4.9.7 Sprinkles

Report by:	Tobias Dammers
Status:	Active, usable

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, Hakyll, 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.

Further reading

- https://sprinkles.tobiasdammers.nl/
- https://bitbucket.org/tdammers/sprinkles
- http://hackage.haskell.org/package/sprinkles

4.9.8 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

- MFlow as a DSL for web applications https://www.fpcomplete.com/school/to-infinity-and-beyond/older-but-still-interesting/MFlowDSL1
- MFlow, a continuation-based web framework without continuations http://themonadreader.wordpress.com/2014/04/23/issue-23
- 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
- Towards a deeper integration: A Web language: http://haskell-web.blogspot.com.es/2014/04/towards-deeper-integration-web-language.html
- Perch https://github.com/agocorona/haste-perch
- hplayground demos http://tryplayg.herokuapp.com
- haste-perch-hplaygroun tutorial http://www.airpair.com/haskell/posts/haskell-tutorial-introduction-to-web-apps
- 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
- MFlow demo site: http://mflowdemo.herokuapp.com

4.9.9 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, extensible records and extensible effects.

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/

4.9.10 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 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.

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:
  http://github.com/lyokha/nginx-haskell-module
- Module ngx-export on Hackage:
  http://hackage.haskell.org/package/ngx-export
- An advanced example in a blog article:
  http://lin-techdet.blogspot.com/2017/01/nginx-haskell-module-labeled-media.html

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 the places where it can be easitly 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 (→ 4.9.4) 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 YeshQL

Report by:	Tobias Dammers
Status:	Active, usable, somewhat stable

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 metainformation as structured SQL comments. The latter allows the quasi-quoter to generate a type-safe API for these queries. YeshQL uses HDBC for the database backends, but doesn't depend on any particular HDBC 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
```

YeshQL is production ready; I have used it on several real-world projects, with good success. The syntax has some breaking changes based on progressive insight, addressing all the issues mentioned in earlier reports.

Recent improvements:

- Queries can now directly return record types, as long as a suitable ToSqlRow instance exists.
- Queries can use projection functions to access subsets of arguments passed to them. This is mainly intended so that you can pass record types to queries, and the query itself extracts the fields it needs.
- FromSqlRow and ToSqlRow typeclasses can be derived automatically with TemplateHaskell.
- YeshQL can be used both as a quasi-quoter and as plain TemplateHaskell to support more use cases.
- It is now possible to generate DDL queries (not returning any results) and queries returning rowcounts instead of result sets.

Contributions of any kind are more than welcome.

Further reading

- https://bitbucket.org/tdammers/yeshql

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

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

4.11 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

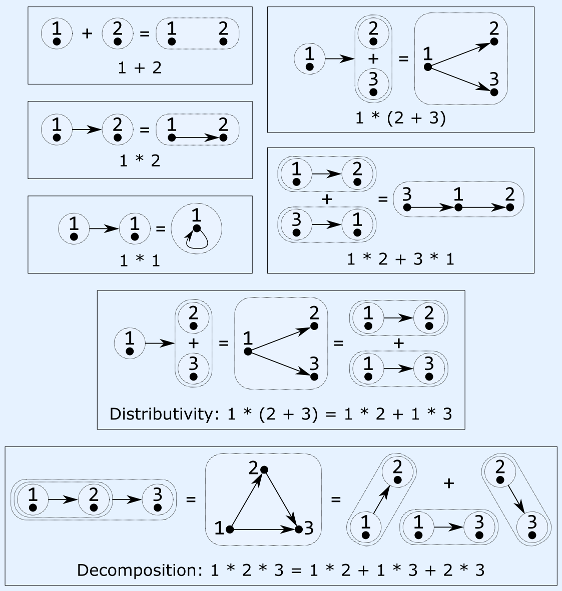
Consider the following data type, which is defined in the top-level module Algebra.Graph of the library:

```
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:

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


1 + 2
1 * (2 + 3)
1 * 2
1 * 1
1 * 2 + 3 * 1
Distributivity: 1 * (2 + 3) = 1 * 2 + 1 * 3
Decomposition: 1 * 2 * 3 = 1 * 2 + 1 * 3 + 2 * 3

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:

- $(+,$ `empty`$)$ is an idempotent commutative monoid.
- $(*,$ `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:	Michael Snoyman
Status:	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.

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 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 introducting 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

4.11.5 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.

Status

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.

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

Due to that, I have lost the motivation to go forward with this package. I still remain open to discussion and suggestions.

Admittedly, the benchmarks I did were perhaps simplistic (they can be found in the Github repo), comparing the speed of generating basic binary trees. I am unsure about how the trade-offs evolve with more complex types.

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/DuFlLoSc04.pdf
- https://hackage.haskell.org/package/generic-random
- https://github.com/Lysxia/generic-random
- https://hackage.haskell.org/package/testing-feat

4.11.6 Generalized Algebraic Dynamic Programming

Report by:	Christian Höner zu Siederdissen
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 theory will simplify building grammar-based applications by dealing with the intrinsic complexity of these algorithms.

We provide multiple types of output. LaTeX is available to those users who prefer to manually write the resulting grammars. Alternatively, Haskell modules can be created. TemplateHaskell and QuasiQuoting machinery is also available turning this framework into a fully usable embedded domain-specific language. The DSL or Haskell module use ADPfusion (→ 4.20.1) with multitape extensions, delivering "close-to-C" performance.

Set Grammars

Most dynamic programming frameworks we are aware of deal with problems over sequence data. There are, however, many dynamic programming solutions to problems that are inherently non-sequence like. Hamiltonian path problems, finding optimal paths through a graph while visiting each node, are a well-studied example.

We have extended our formal grammar library to deal with problems that can not be encoded via linear data types. This provides the user of our framework with two benefits, easy encoding of problems based on set-like inputs and construction of dynamic programming solutions. On a more general level, the extension of ADPfusion and the formal grammars library shows how to encode new classes of problems that are now gaining traction and are being studied.

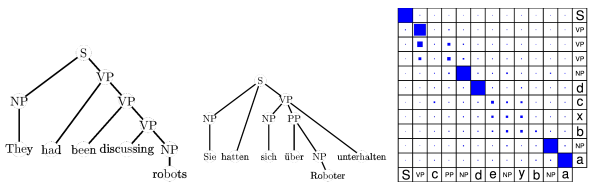
If, say, the user wants to calculate the shortest Hamiltonian path through all nodes of a graph, then the grammar for this problem is:

```
s (f <<< s % n ||| g <<< n ... h)
```

which states that a path s is either extended by a node n, or that a path is started by having just a first, single node n. Functions f and g evaluate the cost of moving to the new node. gADP has notions of sets with interfaces (here: for s) that provide the needed functionality for stating that all nodes in s have been visited with a final visited node from which an edge to n is to be taken.

Tree Grammars

Tree grammars are important for the analysis of structured data common in linguistics and bioinformatics. Consider two parse trees for english and german (from: Berkemer et al. *General Reforestation: Parsing Trees and Forests*) and the node matching probabilities we gain when trying to align the two trees:


S
VP
NP
They had been discussing robots
Sie hatten sich über Roboter unterhalten
PP
d c x b a e y

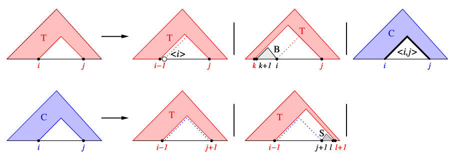
We can create the parse trees themselves with a normal context-free language on sequences. We can also compare the two sentences with, say, a Needleman-Wunsch style sequence alignment algorithm. However, this approach ignores the fact that parse trees encode grammatical structure inherent to languages. The comparison of sentences in english or german should be on the level of the structured parse tree, not the unstructured sequence of words.

Our extension of ADPfusion (→ 4.20.1) to forests as inputs allows us to deal with a variety of problems in complete analogy to sequence-based dynamic programming. This extension fully includes grammar products, and automatic outside grammars.

Automatic Outside Grammars

Our third contribution to high-level and efficient dynamic programming is the ability to automatically construct Outside algorithms given an Inside algorithm. The combination of an Inside algorithm and its corresponding Outside algorithm allow the developer to answer refined questions for the ensemble of all (suboptimal) solutions.

The image below depicts one such automatically created grammar that parses a string from the Outside in. T and C are non-terminal symbols of the Outside grammar; the production rules also make use of the S and B non-terminals of the Inside version.



One can, for example, not only ask for the most efficient path through all cities on a map, but also answer which path between two cities is the most frequented one, given all possible travel routes. In networks, this allows one to determine paths that are chosen with high likelihood.

Multiple Context-Free Grammars

In both, linguistics and bioinformatics, a number of problems exist that can only be described with formal languages that are more powerful than context-free languages, but often have the form of two or more interleaved context-free languages (say: $a^n b^n c^n$). In RNA biology, pseudoknotted structures can be modelled in this way, while in linguistics, we can model languages with crossing dependencies.

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

4.11.7 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

4.11.8 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

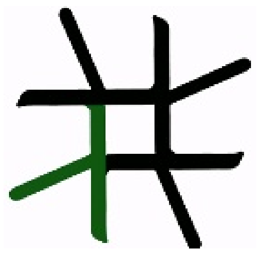
- gitter chat
- Transient tutorial
- distributed Transient, GIT repository
- Transient GIT repository
- An EDSL for Hard-working IT programmers
- The hardworking programmer II: practical backtracking to undo actions
- Publish-suscribe variables
- Moving processes between nodes
- Parallel non-determinism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

4.12 Parallelism

4.12.1 Eden

Report by:	Rita Loogen
Participants:	**in Madrid:** Yolanda Ortega-Mallén, Mercedes Hidalgo, Lidia Sánchez-Gil, Fernando Rubio, Alberto de la Encina, **in Marburg:** Mischa Dieterle, Thomas Horstmeyer, 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. While providing enough control to implement parallel algorithms efficiently, it frees the programmer from the tedious task of managing low-level details by introducing automatic communication (via head-strict lazy lists), synchronization, and process handling.



Eden's primitive constructs are process abstractions and process instantiations. Higher-level coordination is achieved by defining *skeletons*, ranging from a simple parallel map to sophisticated master-worker schemes. They have been used to parallelize a set of non-trivial programs.

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. Zsók, Z. Horváth, and R. Plasmeijer (Eds.): CEFP 2011, Springer LNCS 7241, 2012, pp. 142-206. (see also: www.mathematik.uni-marburg.de/~eden/?content=cefp)

Implementation

Eden is 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.

The source code repository for Eden releases is james.mathematik.uni-marburg.de:8080/gitweb, the Eden libraries (Haskell-level) are also available via Hackage. Please contact us if you need any support.

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. Recently, in the course of his Bachelor thesis, Bastian Reitemeier developed another trace viewer tool, *Eden-Tracelab*, which is capable of visualising large trace files, without being constrained by the available memory. Details can be found in his blogpost brtmr.de/2015/10/17/introducing-eden-tracelab.html.

The Eden skeleton library is under constant development. Currently it contains various skeletons for parallel maps, workpools, divide-and-conquer, topologies and many more. Take a look on the Eden pages.

Recent and Forthcoming Publications

- L. Schiller: *An Agglomeration Law for Sorting Networks and its Application in Functional Programming*, Workshop on Functional and (Constraint) Logic Programming (WFLP), Leipzig, September 2016.
- M. Dieterle, Th. Horstmeyer, R. Loogen, J. Berthold: *Skeleton Composition vs Stable Process Systems in Eden*, Journal of Functional Programming, Volume 26, Published online: 15 July 2016, e11, 40 pages.
- J. Berthold, H.-W. Loidl, K. Hammond: *PAEAN: Portable and scalable runtime support for parallel Haskell dialects*, Journal of Functional Programming, Volume 26, Published online: 13 July 2016, e10, 39 pages.

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 light-weight, 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. As in their case we do not attempt to use the GHC runtime. Our implementation is *light-weight* 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.

Additionally, we have developed a new strictness analysis technique that will be incorporated into the compiler. We'll miss the Static Analysis Symposium 2017 submission deadline, unfortunately.

Immediate Plans

We are currently developing a more realistic parallel runtime. The implementation of the strictness analysis is scheduled as a Master's thesis project starting May 2017.

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, 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 fall 2017 are best guesses based on prior years.

Term	Application Opening	Deadline
Fall 2017	Open	May 31 2017
Spring 2018	July 2017	Oct 2017
Summer 2018	Oct 2017	Jan 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. It provides a typeclass abstraction over a large subset of the functionality in the *Control.Concurrent* module hierarchy, and makes use of testing techniques pioneered in the imperative and object-oriented worlds.

The testing trades completeness for speed, by bounding the number of preemptions and yields in a single execution, as well as the overall length. This also allows testing of potentially non-terminating programs. All of these bounds are optional, however, and can be disabled, or changed.

A brief list of supported functionality:

- Threads: the `forkIO*` and `forkOn*` functions, although bound threads are not supported.
- Getting and setting capablities (testing default is two).
- Yielding and delaying.
- Mutable state: STM, `MVar`, and `IORef`.
- Relaxed memory for `IORef` operations: total store order (the testing default) and partial store order.
- Atomic compare-and-swap for `IORef`.

- Exceptions.
- All of the data structures in *Control.Concurrent.** and *Control.Concurrent.STM.** have typeclass-abstracted equivalents.

This is quite a rich set of functionality, although it is not complete. If there is something else you need, file an issue!

Even if you don't need the testing functionality, the concurrency abstraction (in the "concurrency" package) is useful in its own right, as it allows transparent use of concurrency with many common monad transformers.

The 0.5 and 0.6 releases of dejafu have seen significant performance improvements both in time and space, and support for testing programs by random scheduling.

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 applicate 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.

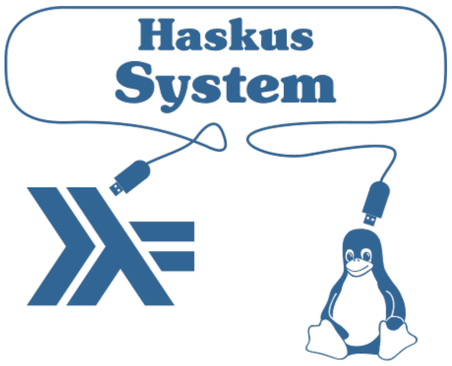
There are some additional challenges in the Haskell ecosystem (e.g. cabal is not very cross compilation aware) that need to be addressed before Haskell for Mobile works out of the box.

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 framework designed for system programming. Fundamentally it is an experiment into providing an integrated interface leveraging Haskell features (type-safety, STM, etc.) for the whole system: input, display, sound, network, etc.

It is based directly and only on the Linux kernel:

- it doesn't contain any kernel level code (device driver, etc.)
- it doesn't rely on usual interfaces (e.g., libdrm, libinput, X11, wayland, etc.) to communicate with the kernel

The source code is freely available (BSD3 license).

Notable changes since the last HCAR:

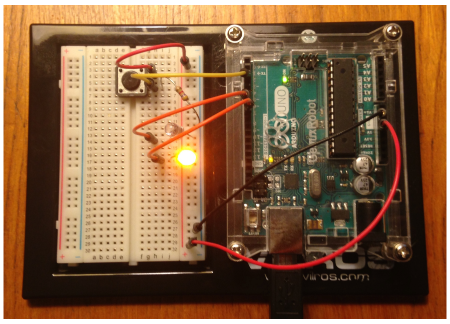
- the project has been renamed from `ViperVM` to `haskus-system`
- a lot of documentation has been written (see the website)
- several examples have been released

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/wiki

4.13.4 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`, but more efficient backends might be desirable, e.g based on Vector. One of the current development goals is completely generalizing the interface to typeclasses in order to decouple algorithms from datastructures.

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

4.15 Graphical User Interfaces

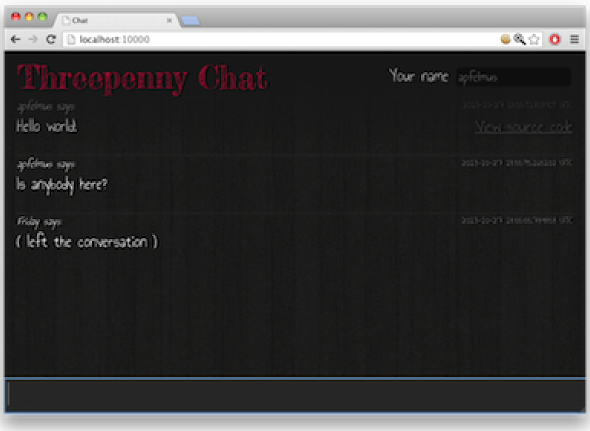
4.15.1 threepenny-gui

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 Stack and use it right away to write that cheap GUI you need for your project. Here a screenshot from the example code:



apfelmus says:
Hello world

apfelmus says:
Is anybody here?

Friday says:
(left the conversation)

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

Status

Compared to the previous report, the JavaScript FFI has been improved significantly. It is now more robust when handling exceptions or loss of connection, and has better performance by (optionally) buffering FFI calls. Moreover, thanks to the efforts of our new co-maintainer Simon Jakobi, the library is now available on Stackage and has been updated to work with the current Haskell ecosystem.

Future development

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

In the future, I hope to include some sort of GUI combinator library that abstracts away the need to go into details with HTML and CSS, as the latter can be very cumbersome for GUI design. Perhaps a solution based on the qooxdoo JavaScript framework may work.

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

4.15.2 wxHaskell

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

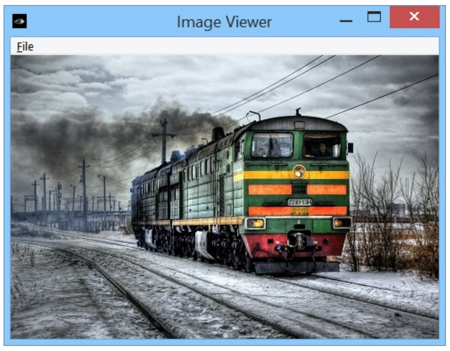
wxHaskell

wxHaskell 0.92.3.0 is released with minor improvements. Next to come, is adapting wxHaskell to wxWidgets 3.1. New project participants are welcome.

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

4.16 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).

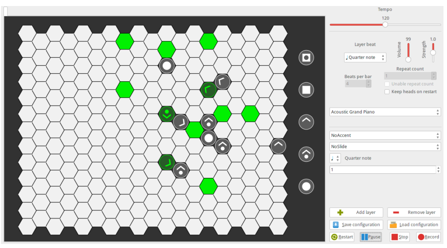
Yampa systems are defined as combinations of Signal Functions. Yampa includes combinators to create constant signals, apply pointwise (or time-wise) transformations, access the running time, introduce delays and create loopbacks (carrying present output as future input). Systems can be dynamic: their 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 reimplementation 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 and web browsers (thanks to GHCJS). Examples of the latter include Keera Studios' Magic Cookies! (https://goo.gl/0A8z6i), a board game for Android written in Haskell and avaliable on Google Play.

Guerric Chupin (ENSTA ParisTech), under the supervision of Henrik Nilsson (Functional Programming Lab, University of Nottingham) has developed Arpeggigon (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*, that will be presented at PADL 2017. The code and an extended version of the paper are publicly available (https://gitlab.com/chupin/arpeggigon).



Yampa is under active development. 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 which will be included in future versions. Furthermore, 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", accepted for 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.

The Functional Programming Laboratory at the University of Nottingham 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.

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

Having reached the milestone of version `1.*`, I consider the library API to be stable and feature complete. Compared to the previous report, thanks to the efforts of our new co-maintainer Oliver Charles, the library is now available on Stackage (→ 4.2.3).

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 GHCJS, as there are some issues with garbage collection.

Further reading

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

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-Orientation (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. 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. Currently we are implementing the full *SugarScape* model of J.M.Epstein and R.Axtell as described in their book, which serves as the initial use-case to drive the implementation of our library. In the next step we also want to implement *Agent_Zero*, the most recent Agent-Model developed by J.M.Epstein. 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).

Further reading

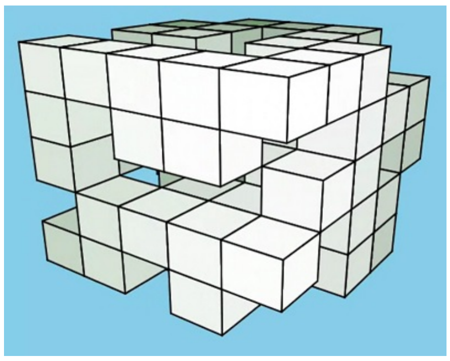
Repository: https://github.com/thalerjonathan/phd/tree/master/coding/libraries/frABS

4.17 Graphics and Audio

4.17.1 diagrams

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

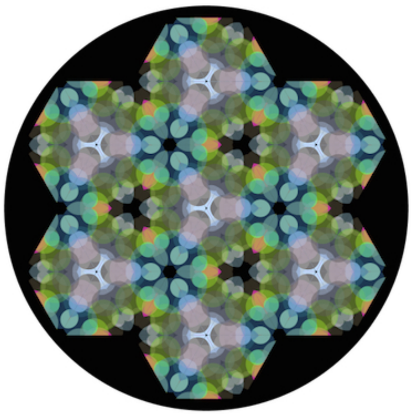
The diagrams framework provides an embedded domain-specific language for declarative drawing. The overall vision is for diagrams to become a viable alternative to DSLs like MetaPost or Asymptote, but with the advantages of being *declarative*—describing what to draw, not how to draw it—and *embedded*—putting 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

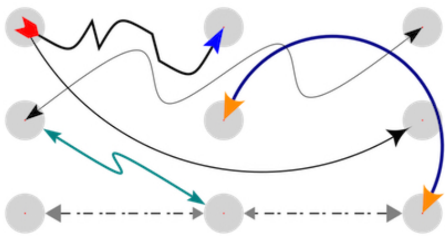
Diagrams 1.4 was released at the end of October, and mostly just adds new features, such as:

- 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.

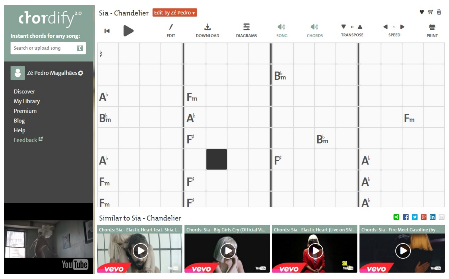


Further reading

- 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 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 use the sonic annotator for extraction of audio features. These features consist of the downbeat positions and the tonal content of a piece of music. Next, from these features a set of probable chords is constructed for each beat. Finally a Hidden Markov Model, which is 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 recently completely redesigned our Haskell backend, improving both the chord accuracy and the scalability. We now have a distributed backend based on Cloud Haskell, allowing us to easily scale up when the demand increases. Our library currently contains about 4.5 million songs, and about 6,000 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.

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

- 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.
- 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 library coverage for machine learning techniques is limited. We use TensorFlow for training and some handwritten code to import and evaluate the trained models in our own pipeline, but we believe that better approaches should be possible.

In the coming months we expect to improve our chord extraction algorithm yet again using deep learning. Recent research has shown that deep learning can improve the feature extraction phase, and our own preliminary research shows that combining chords from multiple sources, such as edits from users on our website, can improve the accuracy of such a feature extraction deep network even further.

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

Further reading

https://chordify.net

4.17.3 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.

There are improvements of the library that are worth to mention:

- Complete support for monophonic synthesizers. Previously it was possible to use them only with MIDI-devices. But now it's also possible to use them with scores and event streams.
- Many guitar bread and butter effects were implemented. It makes it easy to create your own virtual pedalboards and use them on stage if you have good audio-card with Jack-inputs.
  Also it comes with friendly DSL to compose the stompboxes algorithms with visual representation. Need distortion, delay and reverb? You can do it with couple lines of code right in the ghci:

```
let board = fxHor [uiTort1, uiAdele2 0.4 0.35, uiHall2]
dac $ fxApply (board :: Source Fx2)
```

- It greatly improves support for OSC. Several on stage OSC-synthesizers were made with it.
- The complete list of Csound table generators is implemented. With it we can load files to tables and make various functional shape (like splines, random distributions etc)
- It adds exeprimental support for Arrays. It's a Csound container data type.

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

- https://soundcloud.com/anton-kho
- 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

- https://github.com/spell-music/csound-expression
- https://github.com/anton-k/csound-expression
- http://csound.github.io/

4.18 Games

4.18.1 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

4.18.2 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 is it 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.

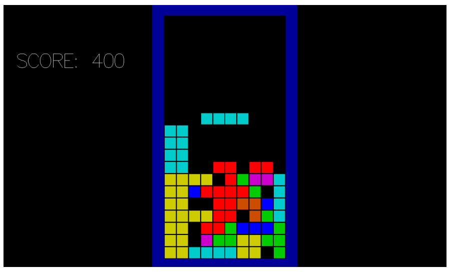
The engine is open source and is published on github. The last released version is Barbarossa v0.4.0 from December 2016.

Further reading

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

4.18.3 Tetris in Haskell in a Weekend

Report by:	Michael Georgoulopoulos
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 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 lazyness 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 reimplementation 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:

- docs have been reorganized, with more focussed manuals available in multiple versions, formats and as built-in help
- 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
- the ledger4 parser is not yet integrated
- hledger-ui has acquired many new features making it more useful (file editing, filtering, historical/period modes, quick period browsing..)
- 1.0 has been released!

Also,

- hledger-web is more robust and more mobile-friendly
- hledger-api, a simple web API server, has been added
- a new "timedot" file format allows retroactive/approximate time logging
- we now support GHC 8 and GHC 7.10, dropping GHC 7.8 and 7.6 support. (GHC 7.8 support requires a maintainer).
- hpack is now used for maintaining cabal files
- our benchmarking tool has been spun off as the quickbench package

- the hledger.org website is simpler, clearer, and more mobile-friendly
- a call for help was sent out last month, and contributor activity has increased.

Future plans include:

- support the 1.0 release
- improve the website and docs
- grow the user & developer community
- clean up, automate, improve and scale our processes
- improve quality, reduce waste
- add the ledger4 parser
- add budget/goal-tracking features
- 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.

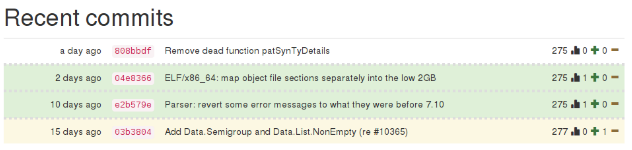
Further reading

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	808bbdf	Remove dead function patSynTyDetails	275 0 ✚ 0 −
2 days ago	04e8366	ELF/x86_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 #10365)	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. Also, it is useful not only for benchmarks: The author uses it to track the progress of his thesis, measured in area covered by the ink.

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 thanks to a contribution by Vincent Rasneur, it now also works on MacOS X.

Further reading

- http://arbtt.nomeata.de/
- http://www.joachim-breitner.de/blog/archives/336-The-Automatic-Rule-Based-Time-Tracker.html
- 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 Siederdissen
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 $p \to c\ p\ c \mid \epsilon$.

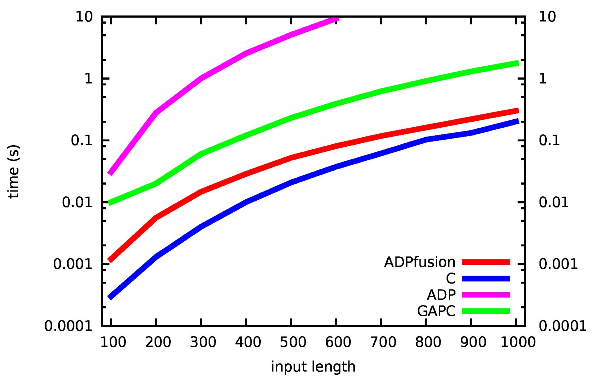
The corresponding ADPfusion code is similar:

```
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, $g\ e\ = \texttt{True}$, and $h\ xs = \texttt{or}\ xs$ 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 (→4.11.6) 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:

a) 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;

b) 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:

a) 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;

b) 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:

i) the powerful benchmarking library Criterion;

ii) 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 Kitchen Snitch server

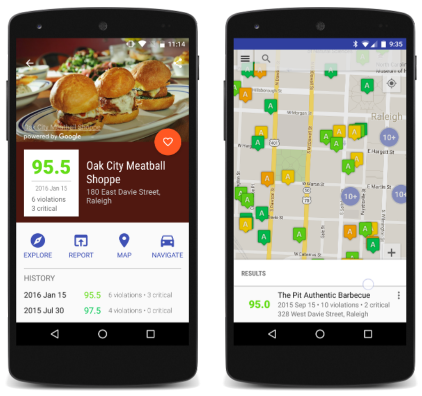
Report by:	Dino Morelli
Participants:	Betty Diegel
Status:	stable, actively developed

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 darcshub 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

4.20.5 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 Haskeller'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

4.20.6 FRTrader

Report by:	Dimitri DeFigueiredo
Status:	active

FRTrader is a new 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.

Recent development in shell-conduit added support for variadic list usage.

```
< makeDirs :: Segment ()
< makeDirs = mkdir "-p" ["dir1", "dir2", "dir3"]
```

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.

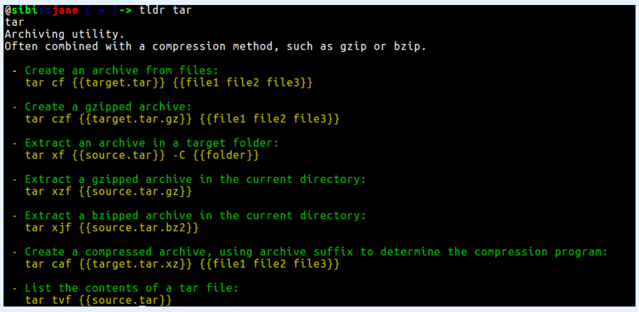
Further reading

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.



```
@sibi::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}}
```

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 game development studio that focuses on games and mobile apps for both Android and iOS.

In recent months 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 accomodate not only games, but also mobile apps using standard widget toolkits available on mobile platforms, in order to provide a natural look and feel.

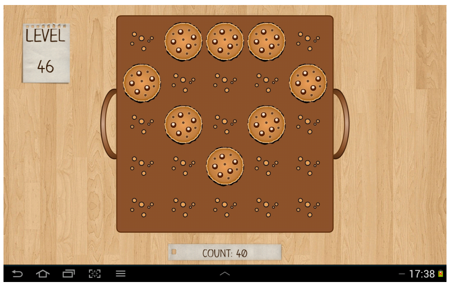
Our development toolkit has been designed to be trivial to run in different environments. Currently we use Travis CI to compile, package, sign and automatically upload our games to Google Play for Android (and tweet when a major release is out!). When developing locally, our toolchain maximizes caching, 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 Titan^{TM} (Testing Infrastructure for Temporal AbstractioNs). Haskell Titan^{TM} 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 Android written in Haskell, available on Google Play^{TM} (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. Magic Cookies! also works seamlessly on iOS.

We have also shown a breakout-like game running on a Android tablet (http://goo.gl/53pK2x), using *hardware acceleration* and *parallelism*, which has also been released on Google Play for Android. The desktop version additionally supports Nintendo Wiimotes and Kinect, and is distributed pre-compiled for Ubuntu via Launchpad. A version compiled with GHCJS has also been published for browser.

We are currently developing two 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, is currently in beta-testing phase. The second, inspired by the classic Super Pang, is currently in prototyping phase.



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 multiplatform, working seamlessly on Linux, Windows and Android.

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 viable option for *professional game development*, for both mobile and desktop games. Our novel testing and debugging tools show that Haskell can, is certain respects, be more suitable than other languages for game 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 and 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, Guerric Chupin (ENSTA ParisTech) and Henrik Nilsson (University of Nottingham) published Arpeggigon (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*, to be presented at PADL 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

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 Computing and Software 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. In 2015, we swapped out Python in our programming activities for ELM, which is a functional replacement for JavaScript. ELM looks a lot like Haskell, but does not have user-definable type classes and is strict. Thanks largely to ELM, we tripled the number of children in our workshops to 3500. Our hypothesis is that declarative programming matches the computational model instructed in basic algebra, which receives significant attention in the Ontario curriculum. But it is possible that other aspects of the language are more important. We also believe that immediate graphical feedback is important, as do many other educators, but since declarative specifications of vector graphics are significantly simpler than stateful constructions, these issues are not orthogonal.

We would like to thank Evan Czapliki for creating ELM, and assisting us.

To see what children with no programming experience can accomplish in a declarative language in a just a few hours, please visit http://outreach.mcmaster.ca/menu/fame.html. Note that grade 4 students are about ten years old.

6 Research and User Groups

6.1 DataHaskell

Report by:	Marco Zocca
Participants:	Nikita Tchayka, Mahdi Dibaiee, John Vial, Stefan Dresselhaus, 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 development team uses a Trello board to track ongoing activities; access to this tool will be granted to all interested parties.

As first steps we set up a documentation site 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.

Current DataHaskell activities are focusing on improving the ergonomics of the IHaskell notebook, and putting it to use on a Kaggle classification exercise. This will serve to highlight the merits and the gaps or inefficiencies in the current package landscape.

We cherish the open and multidisciplinary nature of our community, and welcome all new users and contributions.

Further reading

- datahaskell.org
- https://gitter.im/dataHaskell/Lobby
- https://github.com/DataHaskell
- https://trello.com/b/ucB25d5v/tasks
- http://www.datahaskell.org/docs/
- https://github.com/DataHaskell/numeric-libs-overview
- https://github.com/DataHaskell/DataIHaskell
- https://github.com/johnny555/ToolExamples/tree/master/Kaggle

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, and another variant that permits variable-variables.

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. To support reasoning on space improvements we implemented a tool in Haskell. 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 the last months Dominic Orchard, Hugo Férée and Reuben Rowe joined our group. Stephen Adams is working on advanced refactoring of Haskell programs, extending HaRe. He currently focuses on refactoring code using monads to code such that it uses the Applicative class instead. Andreas Reuleaux is building in Haskell a refactoring tool for a dependently typed functional language. Maarten Faddegon and Olaf Chitil are working on making tracing for Haskell practical and easy to use. Maarten built the lightweight tracer and algorithmic debugger Hoed and Olaf develops the Haskell tracer Hat further. Dominic Orchard is working on coeffectful 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 Férée and Reuben Rowe recently started 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

- PLAS group:
  http://www.cs.kent.ac.uk/research/groups/plas/
- Marco Gaboardi, Shin-ya Kasumata, Dominic Orchard, Flavien Breuvart and Tarmo Uustalu: Combining Effects and Coeffects via Grading. ICFP 2016.
- Maarten Faddegon and Olaf Chitil: Lightweight Computation Tree Tracing for Lazy Functional Languages. PLDI 2016.
- Haskell: the craft of functional programming:
  http://www.haskellcraft.com
- Parsers and static analysis tools for Fortran code in Haskell https://github.com/camfort/fortran-src
- A refactoring and verification tool for Fortran code in Haskell https://github.com/camfort/camfort
- Refactoring Functional Programs: http://www.cs.kent.ac.uk/research/groups/plas/hare.html
- Hoed, a lightweight Haskell tracer and debugger:
  https://github.com/MaartenFaddegon/Hoed
- Hat, the Haskell Tracer:
  http://projects.haskell.org/hat/
- CakeML, a verification friendly dialect of SML:
  https://cakeml.org
- 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 (→ 4.12.4). One example is the Haskino Project (→ 4.13.3).

Further reading

The Functional Programming Group: http://www.ittc.ku.edu/csdl/fpg

6.6 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

- http://groups.google.com/group/fp-syd
- http://fp-syd.ouroborus.net/
- http://fp-syd.ouroborus.net/wiki/Past/2016

6.7 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/.

6.8 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/

6.9 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

6.10 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
- Twitter channel: @ruHaskell
- Subreddit: /r/ruhaskell

6.11 NY Haskell Users Group and Compose Conference

Report by:	Gershom 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 attendence ranges between sixty to one hundred and twenty. NYHUG 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/

6.12 Japan Haskell User Group – Haskell-jp

Report by:	Yuji Yamamoto
Status:	active

Japan Haskell User Group (a.k.a Haskell-jp) is a new Haskell user community group in Japan. Our main goals are to promote Haskell in Japan and to support Haskell users in Japan. For the time being, our main activity is to provide a space where people who want to share Haskell-related information in Japan via haskell.jp domain and the Slack team.

We launched the website on 30th of April 2017. The website is vital to our current main activity. So the date is the birthday of Haskell-jp.

As long as I see, Haskell is not very popular in Japan, even less popular than the other languages. Among those conditions, Haskell-jp just started. We'll make all the best to make Japan a Haskellers' eutopia.

6.13 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⟩