Haskell Communities and Activities Report

http://tinyurl.com/haskcar

Twenty-Ninth Edition — November 2015

Mihai Maruseac (ed.)

Andreas Abel
Chris Allen
Christopher Anand
Francesco Ariis
Heinrich Apfelmus
Emil Axelsson
Christiaan Baaij
Carl Baatz
Doug Beardsley
Jean-Philippe Bernardy
Alexander Berntsen
Joachim Breitner
Björn Buckwalter
Erik de Castro Lopo
Lucas DiCioccio
Manuel M. T. Chakravarty
Roman Cheplyaka
Olaf Chitil
Alberto Gómez Corona
Duncan Coutts
Atze Dijkstra
Péter Diviánszky
Corentin Dupont
Richard Eisenberg
Tom Ellis
Maarten Faddegon
Andrew Farmer
Dennis Felsing
Julian Fleischer
Phil Freeman
PÁLI Gábor János
Michal J. Gajda
Andrew Gibiansky
Brett G. Giles
Andrew Gill
Alexander Granin
Mark Grebe
Daniel Gröber
Jurriaan Hage
Greg Hale
Bastiaan Heeren
Joey Hess
Nicu Ionita
Bob Ippolito
Robin KAY
Anton Kholomiov
Ian-Woo Kim
Oleg Kiselyov
Edward Kmett
Balázs Kőműves
Eric Kow
Nickolay Kudasov
Rob Leslie
Ben Lippmeier
Andres Löh
Rita Loogen
Boris Lykah
Ian Lynagh
José Pedro Magalhães
Ketil Malde
Douglas McClean
Simon Michael
Mantas Markevicius
Dino Morelli
Natalia Muska
Rishiyur Nikhil
Antonio Nikishaev
Ulf Norell
Kiwamu Okabe
Ivan Perez
Jens Petersen
Haskell Consultancy Munich
Simon Peyton Jones
Matthew Pickering
Gracjan Polak
Bryan Richter
Jeffrey Rosenbluth
Ian Ross
David Sabel
Martijn Schrage
Carter Tazio Schonwald
Tom Schrijvers
Michael Schröder
Austin Seipp
Jeremy Shaw
Christian Höner zu Siederdissen
Aditya Siram
Gideon Sireling
Jim Snow
Michael Snoyman
Kyle Marek-Spartz
Lennart Spitzner
Doaitse Swierstra
Henk-Jan van Tuyl
Bernhard Urban
Alessio Valentini
Adam Vogt
Daniel Wagner
Greg Weber
Ingo Wechsung
Kazu Yamamoto
Edward Z. Yang
Brent Yorgey
Alan Zimmerman
Marco Zocca

Preface

This is the 29th edition of the Haskell Communities and Activities Report.

Quite a lot of things changed since the previous report. Some old entries have resurfaced and contributors submitted stories in new areas of development but, sadly, several entries have become too stale and had to be removed. All entries from 2013 except the one about cabal have been removed for now but we hope to see them resurface again on the next edition. Please do revive such entries next time if you do have news on them.

To make contributing easier, we are experimenting with moving to a rolling-deadline submission where each entry can be submitted at any moment of the year. However, we will still have only 2 editions of HCAR per year, at the usual pace, only the entry submission period will be extended. This change allows us to change the pipeline for building the report (thus, the next report might look completely different). Hence, we are also considering opening the building pipeline to be used by any submitter.

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.

A few words from Mihai: The previous issue of HCAR was the last in which Alejandro helped in editing. Due to other obligations he has decided to step down. I want to thank him for the effort so far and for the contributions to the Haskell world.

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

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

Mihai Maruseac, University of Massachusetts Boston, US
⟨hcar@haskell.org⟩

Contents

1 Community

1.1 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 three additions:

Generators: `yield` = exception + non-determinism

Generators, familiar nowadays from Python (although they predate Python by about two decades) are the expressions that *yield*. They give the benefit of lazy evaluation – modular on-demand processing – in strict languages or when lazy evaluation does not apply because the computation has observable effects such as IO. The tutorial explains that *yield* decomposes into exceptions and non-determinism. The one-line definition should clarify:

```
yield :: MonadPlus m ⇒ e → EitherT e m ()
yield x = raise x `mplus` return ()
```

The exception carries the *yield*'s argument 'out of band', on the emergency channel so to speak; the non-deterministic choice *mplus* effectively lets the computation continue despite raising the exception. We have unwittingly obtained the typing of generators: MonadPlus m ⇒ EitherT e m a is the type of a generator returning the result of type a while yielding intermediate results of type e.

The tutorial *derives* the above definition by equational reasoning. It also documents the rich history of generators.

Read the tutorial online.

Icon-like generators in Haskell

The decomposition of *yield* into exceptions and non-determinism sprouts the implementations of generators in Haskell, logical programming and ML – the implementations that are correct and working on the first try. As an illustration, the tutorial shows how to use Haskell as if it were Icon. For example, the following Icon code

```
procedure findodd (s1, s2)
  every i := find (s1, s2) do
    if i % 2 = 1 then suspend i
end
```

that produces all odd indices at which string *s1* occurs within *s2* looks in Haskell as follows

```
findodd2 :: (Monad m, LogicT t, MonadPlus (t m)) ⇒
  String → String → EitherT Int (t m) ()
findodd2 s1 s2 = iterE Nothing $ do
  i ← findIM s1 s2
  if i `mod` 2 ≡ 1 then yield i else return () |])
```

Read the tutorial online.

Generator from any Foldable

Any abstract data type that implements the Data.Foldable interface implements the generator interface. In fact, any (collection) type T with the operation like *mapM_* :: Monad m ⇒ (Element T → m b) → T → m () that lets a monadic action examine its elements (of the type Element T) implements the generator interface. The claims holds regardless of the implementation of the data type, whether it is a data structure or if the elements to traverse are computed on-the-fly.

The proof of the claim is constructive, short, and trivial:

```
foldable_gen :: (MonadPlus m, F.Foldable t) ⇒
  t a → EitherT a m ()
foldable_gen = F.mapM_ yield
```

The type says it all: any Foldable is a generator.
Read the tutorial online.

2.2 School of Haskell

Report by:	Natalia Muska
Participants:	Michael Snoyman, 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.

Two new features were added to the School of Haskell. First is the addition of Disqus for commenting on each tutorial and highlighting other potentially interesting tutorials. Second is the inclusion of autorun tags. This enables users to run a snippet as soon as they open a tutorial.

Currently 3150 tutorials have been created (a 125% increase from this time last year) and 441 have been officially published (a 53% increase from this time last year). 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.fpcomplete.com/school

2.3 Haskell Programming from first principles, a book forall

Report by:	Chris Allen
Participants:	Julie Moronuki
Status:	In progress, content complete soon

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 about 70 percent complete and is 912 pages long in the v0.7.0 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 doing the main block of writing and testing of material. We've got some unannounced material that we think will excite people a bit as well.

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 four 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, but the first four chapters are already available. The latest chapter illustrates 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 Agda Tutorial

Report by:	Péter Diviánszky
Participants:	Ambrus Kaposi, students at ELTE IK
Status:	experimental

Agda may be the next programming language to learn after Haskell. Learning Agda gives more insight into the various type system extensions of Haskell, for example.

The main goal of the tutorial is to let people explore programming in Agda without learning theoretical background in advance. Only secondary school mathematics is required for the tutorial.

Further reading

http://people.inf.elte.hu/divip/AgdaTutorial/Index.html

3 Implementations

3.1 The Glasgow Haskell Compiler

Report by: Austin Seipp
Participants: many others

GHC development spurs on, with an exciting new announcement - the next release will be a super-major one, culminating in **GHC 8.0**. There are many reasons for this change, but one of the most exciting is that GHC is getting a **completely new core language**. While this adds a bit of complexity to the compiler, it paves the way to implement **Dependent Haskell** over the course of the next few years.

Major changes in GHC 8.0.1

Support for simple, implicit callstackas with source locations and implicit parameters providing callstacks/source locations, allowing you to have a lightweight means of getting a call-stack in a Haskell application.

Improved optimization diagnostics The compiler is now more liberal about issues warnings of potentially non-firing rewrite rules and other potential gotchas.

Support for wildcards in data and type family instances

Injective type families Injective TFs allow you to specify type families which are injective, i.e. have a one-to-one relationship.

Applicative do notation With the new −XApplicativeDo, GHC tries to desugar do-notation to Applicative where possible, giving a more convenient sugar for many common Applicative expressions.

Support for deriving the Lift **typeclass** - a very common need when working with Template Haskell.

A PowerPC 64bit code generator The new native codegen supports Linux/ppc64 in both big endian and little endian mode.

A beautiful new users guide Now rewritten in reStructured Text, and with significantly improved output and documentation.

Visible type application This allows you to say, for example *id*@Bool to specialize *id* to Bool → Bool. With this feature, proxies are never needed.

Kind Equalities , which form the first step to building Dependent Haskell. This feature enables promotion of GADTs to kinds, kind families, heterogeneous equality (kind-indexed GADTs), and $* :: *$.

Strict Haskell support This includes new −XStrictData and −XStrict language extensions.

Support for record pattern synonyms

Implement phase 1 of the MonadFail proposal

Overloaded record fields At long last, ORF will finally be available in GHC 8.0, allowing multiple uses of the same field name and a form of type-directed name resolution.

A huge improvement to pattern matching (including much better coverage of GADTs), based on the work of Simon PJ and Georgios Karachalias.

More Backpack improvements There's a new user-facing syntax which allows multiple modules to be defined a single file, and we're hoping to release at least the ability to publish multiple "units" in a single Cabal file.

Support for DWARF based stacktraces from Peter Wortmann, Arash Rouhani, and Ben Gamari with backtraces from Haskell code.

A better LLVM backend We're planning on a major build system change that will ship GHC 8.0 with a pre-baked copy of LLVM 3.7.0, that ships with every major Tier 1 platform.

Big GC improvements for large, 64-bit workloads , motivated by work Simon Marlow has been doing at Facebook. If you have a GC-intensive workload with large heaps, GHC 8.0.1 should hopefully lower latency and collection time for your applications.

Upcoming post 8.0 plans

Naturally, there were several things we didn't get around to this cycle, or things which are still in flight and being worked on. (And you can always try to join us if you want something done!)

Libraries, source language, type system

- A new, type-indexed type representation, data TTypeRep $(a :: k)$. See TypeableT.
- Support for **Type Signature Sections**, allowing you to write $(::ty)$ as a shorthand for $(\lambda x \to x :: ty)$.
- A DEPRECATED pragma for exports

◦ Further work on Dependent Haskell, including pi types.

Back-end and runtime system

◦ More DWARF improvements, including experimentations in new performance tools!
◦ Work continues on GHC on ARM, and more recently, GHC on AArch64! 8.0 is leagues ahead of 7.10, but there is still much to be done.
◦ We're working on integrating support for **Compact Normal Forms**, described in an ICFP 2015 paper. CNFs are designed for the use case of transmitting large data structures across applications, without paying for costly serialization.

Frontend, build-system and miscellaneous changes

◦ A planned overhaul of GHC's build system to use Shake instead of Make is still in play, although it didn't make it in time for 8.0.1
◦ A new Mac Mini will be added to our continuous integration suite, thanks to a generous donation from Futurice!
◦ Work on a new set of flags, −Wcompat, is underway, hoping to bring some order to the chaos of code-breaking language updates, and the implications these have for −Wall clean codebases.
◦ Work steams on to make GHC a **deterministic** compiler - always producing the same output object files for the same input. This has proven to be surprisingly tricky, but we just needed a smart hacker to step up and tackle it - and **Bartosz Nitka** has decided to do just that!

Development updates and getting involved

GHC is as busy as ever, and the amount of patches we review and contributors we actively get has been steadily increasing. In particular, **Ben Gamari** from Well-Typed has joined Austin in ongoing GHC maintenance, patch review, and guidance.

But GHC is far, far too large for even a few paid individuals to fully manage it. As always, if you want something done, you should get involved and talk to us!

Since the last release, we've continued to see some major effort expended by new contributors. **Matthew Pickering** has lead the way on major improvements to the state of pattern synonyms, **Ömer Sinan Ağacan** has been tirelessly working on multiple parts of the compiler, and **Thomas Miedema** has continued with seemingly endless energy to improve all areas of GHC.

Of course, it would be even better if your name were included here! So if you've got a few spare cycles - we could always use more helping hands...

Links:

◦ https://ghc.haskell.org/trac/ghc/wiki/ApiAnnotations
◦ https://ghc.haskell.org/trac/ghc/wiki/ApplicativeDo
◦ https://ghc.haskell.org/trac/ghc/wiki/Building/Shake
◦ https://ghc.haskell.org/trac/ghc/wiki/DWARF
◦ https://ghc.haskell.org/trac/ghc/wiki/DependentHaskell/Phase1
◦ https://ghc.haskell.org/trac/ghc/wiki/DistributedHaskell
◦ https://ghc.haskell.org/trac/ghc/wiki/ExplicitCallStack/ImplicitLocations
◦ https://ghc.haskell.org/trac/ghc/wiki/ExplicitTypeApplication
◦ https://ghc.haskell.org/trac/ghc/wiki/GhcApi
◦ https://ghc.haskell.org/trac/ghc/wiki/ImprovedLLVMBackend
◦ https://ghc.haskell.org/trac/ghc/wiki/InjectiveTypeFamilies
◦ https://ghc.haskell.org/trac/ghc/wiki/OverloadedRecordFields
◦ https://ghc.haskell.org/trac/ghc/wiki/PartialTypeSignatures
◦ https://ghc.haskell.org/trac/ghc/wiki/Plugins/TypeChecker
◦ https://ghc.haskell.org/trac/ghc/wiki/Records/OverloadedRecordFields
◦ https://ghc.haskell.org/trac/ghc/wiki/StaticPointers
◦ https://ghc.haskell.org/trac/ghc/wiki/Typeable
◦ https://ghc.haskell.org/trac/ghc/wiki/prelude710
◦ http://www.seas.upenn.edu/~eir/papers/2013/fckinds/fckinds-extended.pdf
◦ https://github.com/yav/type-nat-solver/raw/master/docs/paper.pdf
◦ https://github.com/yav/type-nat-solver
◦ http://ezyang.com/papers/ezyang15-cnf.pdf

3.2 Ajhc Haskell Compiler

Report by:	Kiwamu Okabe
Participants:	John Meacham, Hiroki Mizuno, Hidekazu Segawa, Takayuki Muranushi
Status:	experimental

What is it?

Ajhc is a Haskell compiler, and acronym for "A fork of jhc".

Jhc (http://repetae.net/computer/jhc/) converts Haskell code into pure C language code running with jhc's runtime. And the runtime is written with 3000 lines (include comments) pure C code. It's a magic!

Ajhc's mission is to keep contribution to jhc in the repository. Because the upstream author of jhc, John Meacham, can't pull the contribution speedily. (I think he is too busy to do it.) We should feedback jhc any changes. Also Ajhc aims to provide the Metasepi project with a method to rewrite NetBSD kernel using Haskell. The method is called Snatch-driven development http://www.slideshare.net/master_q/20131020-osc-tokyoajhc.

Ajhc is, so to speak, an accelerator to develop jhc.

Demonstrations

https://www.youtube.com/watch?v=XEYcR5RG5cA

NetBSD kernel's HD Audio sound driver has interrupt handler. The interrupt handler of the demo is re-written by Haskell language using Ajhc.

At the demo, run following operations. First, set breakpoint at the interrupt of finding headphone, and see Haskell function names on backtrace. Second, set breakpoint s_alloc() function, that allocate area in Haskell heap. Make sure of calling the function while anytime running kernel. Nevertheless, playing wav file does not break up.

The source code is found at https://github.com/metasepi/netbsd-arafura-s1 The interrupt handler source code at https://github.com/metasepi/netbsd-arafura-s1/blob/fabd5d64f15058c198ba722058c3fb89f84d08a5/metasepi/sys/hssrc/Dev/Pci/Hdaudio/Hdaudio.hs#L15.

Discussion on mailing list: http://www.haskell.org/pipermail/haskell-cafe/2014-February/112802.html

http://www.youtube.com/watch?v=n6cepTfnFoo

The touchable cube application is written with Haskell and compiled by Ajhc. In the demo, the application is breaked by ndk-gdb debugger when running GC. You could watch the demo source code at https://github.com/ajhc/demo-android-ndk.

http://www.youtube.com/watch?v=C9JsJXWyajQ

The demo is running code that compiled with Ajhc on Cortex-M3 board, mbed. It's a simple RSS reader for reddit.com, showing the RSS titles on Text LCD panel. You could watch the demo detail and source code at https://github.com/ajhc/demo-cortex-m3.

http://www.youtube.com/watch?v=zkSy0ZroRIs

The demo is running Haskell code without any OS. Also the clock exception handler is written with Haskell.

Usage

You can install Ajhc from Hackage.

```
$ cabal install ajhc
$ ajhc --version
ajhc 0.8.0.9 (9c264872105597700e2ba403851cf3b
236cb1646)
compiled by ghc-7.6 on a x86_64 running linux
$ echo 'main = print "hoge"' > Hoge.hs
$ ajhc Hoge.hs
$ ./hs.out
"hoge"
```

Please read “Ajhc User's Manual” to know more detail. (http://ajhc.metasepi.org/manual.html)

Future plans

Maintain Ajhc as compilable with latast GHC.

License

- Runtime: MIT License https://github.com/ajhc/ajhc/blob/master/rts/LICENSE
- Haskell libraries: MIT License https://github.com/ajhc/ajhc/blob/master/lib/LICENSE
- The others: GPLv2 or Later https://github.com/ajhc/ajhc/blob/arafura/COPYING

Contact

- Mailing list: http://groups.google.com/group/metasepi
- Bug tracker: https://github.com/ajhc/ajhc/issues
- Metasepi team: https://github.com/ajhc?tab=members

Further reading

- Ajhc – Haskell everywhere: http://ajhc.metasepi.org/
- jhc: http://repetae.net/computer/jhc/
- Metasepi: Project http://metasepi.org/
- Snatch-driven-development: http://www.slideshare.net/master_q/20131020-osc-tokyoajhc

3.3 The Helium Compiler

Report by:	Jurriaan Hage
Participants:	Bastiaan Heeren

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

Currently Helium is at version 1.8.1. The major change with respect to 1.8 is that Helium is again well-integrated with the Hint programming environment that Arie Middelkoop wrote in Java. The jar-file for Hint can be found on the Helium website, which is located at http://www.cs.uu.nl/wiki/Helium. This website also explains in detail what Helium is about, what it offers, and what we plan to do in the near and far future.

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

3.4 UHC, Utrecht Haskell Compiler

Report by:	Atze Dijkstra
Participants:	many others
Status:	active development

UHC is the Utrecht Haskell Compiler, supporting almost all Haskell98 features and most of Haskell2010, plus experimental extensions.

Status Current active development directly on UHC:
- Making intermediate Core language available as a compilable language on its own, used by an experimental Agda backend (Philipp Hausmann).
- The platform independent part of UHC has been made available via Hackage, as package "uhc-light" together with a small interpreter for Core files (Atze Dijkstra, interpreter still under development).
- Implementing static analyses (Tibor Bremer, Jurriaan Hage, in progress).
- Rework of the type system (Alejandro Serrano, Jurriaan Hage, just started).

Background. UHC actually is a series of compilers of which the last is UHC, plus infrastructure for facilitating experimentation and extension. The distinguishing features for dealing with the complexity of the compiler and for experimentation are (1) its stepwise organisation as a series of increasingly more complex standalone compilers, the use of DSL and tools for its (2) aspectwise organisation (called Shuffle) and (3) tree-oriented programming (Attribute Grammars, by way of the Utrecht University Attribute Grammar (UUAG) system (→ 5.3.2).

Further reading

- UHC Homepage:
  http://www.cs.uu.nl/wiki/UHC/WebHome
- UHC Github repository:
  https://github.com/UU-ComputerScience/uhc
- Attribute grammar system: http://www.cs.uu.nl/wiki/HUT/AttributeGrammarSystem

3.5 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, despite 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 this 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 month, three of the most offending syntactical differences have been removed: lambda syntax, instance/class context syntax and recognition of True and False as boolean literals.

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 develop a new backend of the compiler to support and employ Java 8 lambdas. In addition, several contributors are porting or re-implementing GHC libraries, like Array or STM.

Further reading

https://github.com/Frege/frege

3.6 Specific Platforms

3.6.1 Haskell on FreeBSD

Report by:	PÁLI Gábor János
Participants:	FreeBSD Haskell Team
Status:	ongoing

The FreeBSD Haskell Team is a small group of people who maintain Haskell software on all actively supported versions of FreeBSD. The primarily supported implementation is the Glasgow Haskell Compiler together with Haskell Cabal, although one may also find Hugs and NHC98 in the ports tree. FreeBSD is a Tier-1 platform for GHC (on both x86 and x86_64) starting from GHC 6.12.1, hence one can always download native vanilla binary distributions for each new release.

We have a developer (staging) repository for Haskell ports that currently features around 600 of many of the popular Cabal packages. Most of the updates committed to that repository are continuously integrated to the official ports tree on a regular basis. In result, the FreeBSD Ports Collection still offers many popular and important Haskell software: GHC 7.10.2, Gtk2Hs, wxHaskell, XMonad, Pandoc, Gitit, Yesod, Happstack, Snap, Agda (along with its standard library), git-annex, and so on – all of them are available on 9.3-RELEASE and 10.2-RELEASE. Note that we decided to abandon tracking Haskell Platform (although all its former components are still there as individual packages), instead we updated the packages to match their versions on Stackage (at end of August).

If you find yourself interested in helping us or simply want to use the latest versions of Haskell programs on FreeBSD, check out our development repository on GitHub (see below) where you can find the latest versions of the ports together with all the important pointers and information required for contacting or contributing.

Further reading

https://github.com/freebsd-haskell/ports

3.6.2 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 741 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 just released stable Debian release ("jessie") provides the Haskell Platform 2013.2.0.0 and GHC 7.6.3, while in Debian unstable, we ship GHC 7.8.4. GHC 7.10.2, together with all libraries, is ready to be used in Debian experimental, and should be uploaded to unstable shortly.

Debian users benefit from the Haskell ecosystem on 14 architecture/kernel combinations, including the non-Linux-ports KFreeBSD and Hurd.

Further reading

http://wiki.debian.org/Haskell

3.6.3 Fedora Haskell SIG

Report by:	Jens Petersen
Participants:	Ricky Elrod, Ben Boeckel, and others
Status:	active

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

Fedora 22 is about to be released. Updating to ghc-7.8.4 turned out to be a lot of work. Some packages now have static subpackages for portability: alex, cabal-install, pandoc, and darcs. Lots of Haskell packages were updated to their latest versions (see "Package changes" below).

Fedora 23 development is starting: we are considering if we can update to ghc-7.10 if there is a bugfix release in time, and to refresh packages to their latest versions tracking Stackage where possible. In the meantime there is a ghc-7.10.1 Fedora Copr repo available for Fedora 20+ and EPEL 7.

At the time of writing we have 314 Haskell source packages in Fedora. The cabal-rpm packaging tool has improved further with a new update command, dnf support, and various bugfixes and improvements.

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-list: https://admin.fedoraproject.org/mailman/listinfo/haskell
- Package list: https://admin.fedoraproject.org/pkgdb/users/packages/haskell-sig
- Package changes: http://git.fedorahosted.org/cgit/haskell-sig.git/tree/packages/diffs/f21-f22.compare

4 Related Languages and Language Design

4.1 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 fun as a platform for experiments in dependently typed programming.

Since the release of Agda 2.4.0 in June 2014 a lot has happened in the Agda project and community. For instance:

- There have been two Agda courses at the Oregon Programming Languages Summer School (OPLSS). In 2014 by Ulf Norell, and in 2015 by Peter Dybjer.
- Agda has moved to github: https://github.com/agda/agda.
- Agda 2.4.2 was released in September 2014, and the latest stable version is Agda 2.4.2.4, released in September 2015.
- The restriction of Agda to not use Streicher's Axiom K was proved correct by Jesper Cockx et al. in the ICFP 2014 paper *Pattern Matching without K*.
- Instance arguments are now powerful enough to emulate Haskell-style type classes.
- The reflection machinery has been extended, making it possible to define convenient reflection based tactics.
- Improved compiler performance, and a new backend targeting the Utrecht Haskell Compiler (UHC).

Release of Agda 2.4.4 is planned for early 2016.

Further reading

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

4.2 MiniAgda

Report by:	Andreas Abel
Status:	experimental

MiniAgda is a tiny dependently-typed programming language in the style of Agda (→ 4.1). It serves as a laboratory to test potential additions to the language and type system of Agda. MiniAgda's termination checker is a fusion of sized types and size-change termination and supports coinduction. Bounded size quantification and destructor patterns for a more general handling of coinduction. Equality incorporates eta-expansion at record and singleton types. Function arguments can be declared as static; such arguments are discarded during equality checking and compilation.

MiniAgda is now hosted on http://hub.darcs.net/abel/miniagda.

MiniAgda is available as Haskell source code on hackage and compiles with GHC 6.12.x – 7.8.2.

Further reading

http://www.cse.chalmers.se/~abela/miniagda/

4.3 Disciple

Report by:	Ben Lippmeier
Participants:	Ben Lippmeier, Amos Robinson, Erik de Castro Lopo, Kyle van Berendonck
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, effect and closure 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 March this year we released DDC 0.4.1, with the following new features:

- Added a bi-directional type inferencer based on Joshua DunïňĄeld and Neelakantan Krishnaswami's recent ICFP paper.
- Added a region extension language construct, and coeffect system.
- Added the Disciple Tetra language which includes infix operators and desugars into Disciple Core Tetra.
- Compilation of Tetra and Core Tetra programs to C and LLVM.
- Early support for rate inference in Core Flow.
- Flow fusion now generates vector primops for maps and folds.
- Support for user-defined algebraic data types.
- Civilized error messages for unsupported or incomplete features.
- Most type error messages now give source locations.
- Building on Windows platforms.
- Better support for foreign imported types and values.
- Changed to Git for version control.

Further reading

http://disciple.ouroborus.net

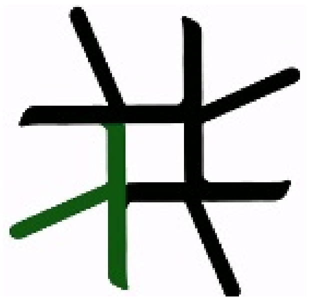
5 Haskell and . . .

5.1 Haskell and Parallelism

5.1.1 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: http://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.

A version based on GHC-7.8.2 (including binary packages and prepared source bundles) has been released in April 2014. This version fixed a number of issues related to error shut-down and recovery, and featured extended support for serialising Haskell data structures. The release of a version based on GHC-7.10 is in preparation. Previous stable releases with binary packages and bundles are still available on the Eden web pages.

The source code repository for Eden releases is http://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 http://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

- Lidia Sánchez-Gil: *On the equivalence of operational and denotational semantics for parallel functional languages*, PhD Thesis, Facultad de Informática, Universidad Complutense de Madrid, July 2015. http://eprints.ucm.es/33213/
- Bastian Reitemeier: *Analysis of Large Eden Trace Files*, Bachelor Thesis, Philipps-Universität Marburg, October 2015.
- Thomas Horstmeyer: *Smarter Communication Channels*, Pre-Symposium Proceedings of IFL 2015, Sept. 2015.
- Lukas Schiller: *A functional view of BatcherÂŠs bitonic sorting network*, Pre-Symposium Proceedings of IFL 2015, Sept. 2015.
- M. Dieterle, Th. Horstmeyer, R. Loogen, J. Berthold: *Skeleton Composition vs Stable Process Systems in Eden*, submitted for publication
- J. Berthold, H.-W. Loidl, K. Hammond: *PAEAN: Portable Runtime Support for Physically-Shared-Nothing Architectures in Parallel Haskell Dialects*, submitted for publication

Further reading

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

5.1.2 Wakarusa

Report by:	Andrew Gill
Participants:	Mark Grebe, Ryan Scott, James Stanton, David Young
Status:	active

The Wakarusa project is a domain-specific language toolkit, that makes domain-specific languages easier to deploy in high-performance scenarios. The technology is going to be initially applied to two types of high-performance platforms, GPGPUs and FPGAs. However, the toolkit will be general purpose, and we expect the result will also make it easier to deploy DSLs in situations where resource usage needs to be well-understand, such as cloud resources and embedded systems. The project is funded by the NSF.

Wakarusa is a river just south of Lawrence, KS, where the main campus of the University of Kansas is located. Wakarusa is approximately translated as “deep river”, and we use deep embeddings a key technology in our DSL toolkit. Hence the project name Wakarusa.

A key technical challenge with syntactic alternatives to deep embeddings is knowing when to stop unfolding. We are using a new design pattern, called the remote monad (→ 6.5.7), which allows a monad to be virtualized, and run remotely, to bound our unfolding. We have already used remote monads for graphics (Blank Canvas), hardware bus protocols (λ-bridge), a driver for MineCraft, an implementation of JSON-RPC, and a compiler for the Arduino (→ 6.1.8). Using the remote monad design pattern, and HERMIT, we are developing a translation framework that translates monadic Haskell to GPGPUs (building on accelerate), and monadic Haskell to Hardware (building on Kansas Lava), and monadic imperative Haskell to Arduino C.

Further reading

- https://github.com/ku-fpg/wakarusa
- http://ku-fpg.github.io/research/wakarusa/

5.2 Haskell and the Web

5.2.1 WAI

Report by:	Michael Snoyman
Participants:	Greg Weber
Status:	stable

The Web Application Interface (WAI) is an interface between Haskell web applications and Haskell web servers. By targeting the WAI, a web framework or web application gets access to multiple deployment platforms. Platforms in use include CGI, the Warp web server, and desktop webkit.

WAI is also a platform for re-using code between web applications and web frameworks through WAI middleware and WAI applications. WAI middleware can inspect and transform a request, for example by automatically gzipping a response or logging a request. The Yesod (→ 5.2.2) web framework provides the ability to embed arbitrary WAI applications as subsites, making them a part of a larger web application.

By targeting WAI, every web framework can share WAI code instead of wasting effort re-implementing the same functionality. There are also some new web frameworks that take a completely different approach to web development that use WAI, such as webwire (FRP), MFlow (continuation-based) and dingo (GUI). The Scotty (→ 5.2.11) web framework also continues to be developed, and provides a lighter-weight alternative to Yesod. Other frameworks- whether existing or newcomers- are welcome to take advantage of the existing WAI architecture to focus on the more innovative features of web development.

WAI applications can send a response themselves. For example, wai-app-static is used by Yesod to serve static files. However, one does not need to use a web framework, but can simply build a web application using the WAI interface alone. The Hoogle web service targets WAI directly.

Since the last HCAR, WAI has successfully released version 3.0, which removes dependency on any specific streaming data framework. A separate wai-conduit package provides conduit bindings, and such bindings can easily be provided for other streaming data frameworks.

The WAI community continues to grow, with new applications and web frameworks continuing to be added. We've recently started a new mailing list to discuss WAI related topics. Interested parties are strongly encouraged to join in!

Further reading

- http://www.yesodweb.com/book/wai
- https://groups.google.com/d/forum/haskell-wai

5.2.2 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 (→ 7.6.1) — 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 (→ 7.6.1). 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 (→ 5.2.1) 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 (→ 5.2.11).

The new 1.4 release of Yesod is 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 (→ 1.1) source code: (https://github.com/snoyberg/haskellers).

Further reading

http://www.yesodweb.com/

5.2.3 Scotty

Report by:	Andrew Farmer
Participants:	Andrew Farmer
Status:	active

Scotty is a Haskell web framework inspired by Ruby's Sinatra, using WAI (→ 5.2.1) and Warp (→ 5.2.4), and is designed to be a cheap and cheerful way to write RESTful, declarative web applications.

- A page is as simple as defining the verb, url pattern, and Text content.
- It is template-language agnostic. Anything that returns a Text value will do.
- Conforms to WAI Application interface.
- Uses very fast Warp webserver by default.

The goal of Scotty is to enable the development of simple HTTP/JSON interfaces to Haskell applications.

Implemented as a monad transformer stack, Scotty applications can be embedded in arbitrary MonadIOs. The Scotty API is minimal, and fully documented via haddock. The API has recently remained stable, with a steady stream of improvements contributed by the community.

Further reading

- Hackage: http://hackage.haskell.org/package/scotty
- Github: https://github.com/scotty-web/scotty

5.2.4 Warp

Report by:	Michael Snoyman

Warp is a high performance, easy to deploy HTTP server backend for WAI (→ 5.2.1). Since the last HCAR, Warp has followed WAI in its move from conduit to a lower level streaming data abstraction. We've additionally continued work on more optimizations, and improved support for power efficiency by using the auto-update package.

Due to the combined use of ByteStrings, blaze-builder, conduit, and GHC's improved I/O manager, WAI+Warp has consistently proven to be Haskell's most performant web deployment option.

Warp is actively used to serve up most of the users of WAI (and Yesod).

"Warp: A Haskell Web Server" by Michael Snoyman was published in 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

5.2.5 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 experimentally supported.

Since Warp supports HTTP/2, Mighttpd 3 is able to serve in HTTP/2 (if TLS is enabled) as well as HTTP/1.1.

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

Further reading

- http://www.mew.org/~kazu/proj/mighttpd/en/
- http://www.yesodweb.com/blog/2015/07/http2

5.2.6 Happstack

Report by:	Jeremy Shaw

Happstack is a very fine collection of libraries for creating web applications in Haskell. We aim to leverage the unique characteristics of Haskell to create a highly-scalable, robust, and expressive web framework.

Over the past year, much development has been focused on the higher-level components such as a rewrite of the `happstack-authentication` library and work on unifying the various `stripe` bindings into a single authoritative binding.

Over the next year we hope to get back to the core and focus on `hyperdrive`, a new low-level, trustworthy HTTP backend, as well as focusing on developing and deploying applications using `nixops`.

Further reading

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

5.2.7 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, 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.

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

5.2.8 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. [1].

A paper describing the MFlow internals has been published in The Monad Reader issue 23 [2]

The use of backtracking to solve "the integration problem". It happens when the loose coupling produce exceptional conditions that may trigger the rollback of actions in the context of failures, shutdowns and restart of the systems (long running processes). That has been demonstrated using MFlow in [3].

A web application can be considered as an special case of integration. MFlow pack the elements of a web aplication within composable widgets. This "deep integration" is the path followed by the software industry to create from higher level framewors to operating systems [4]

perch[5] and hplayground are two new packages that make run the page logic of MFlow in the Web Browser using Haste, the Haskell-to-JavaScript compiler. perch has the syntax of blaze-html and hplayground uses the syntax and primitives of the View Monad. Both permit the page logic of MFlow to run fully in the Web Browser. Under the same syntax, they are completely different. It generates trees by calling DOM primitives directly. While string builders are unary tree constructors, perch uses a generalized builder for n-trees. It also has combinators for the modification of elements and it can assign perch event handlers to elements and it has also JQuery-like operations. It can be used alone for the creation of client-side applications.

Perch run in Haste and has been ported to GHCJS by Arthur Fayzrakhmanov.

hplayground is a monadic functional reactive[6] framework with MFlow syntax that permits the creation of seamless client-side applications. it uses the Transient monad. hplayground can sequence perch renderings with events. It is possible to construct composable monadic and applicative formlets with validations, and event handling included these formlets may modify his own rendering.

There is a site with example Haste-perch-hplayground (made with MFlow) online[6] . There is also a tutorial for the creation of Client-side applications, that describe the structure of a small accounting application for haskell beginners[7].

Since the events are handled locally but there are no event handlers, Monadic Reactive may be a better alternative to functional Reactive in the creation of seamless Web Browser applications whenever there are many dynamic DOM updates[8].

Future work: To port hplayground to GHCJS. To manage client-side applications as nodes in the cloud using websockets and transient.

Further reading

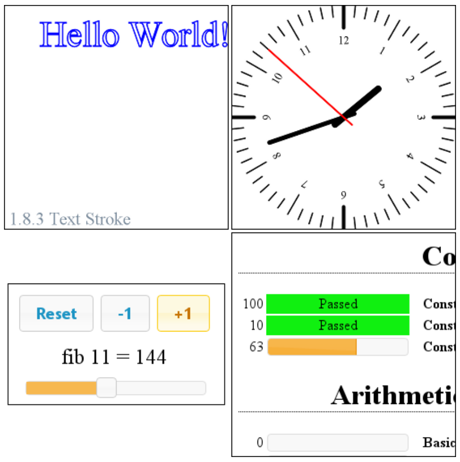
- MFlow as a DSL for web applications https://www.fpcomplete.com/school/to-infinity-and-beyond/older-but-still-interesting/MFlowDSL1
- MFlow, a continuation-based web framework without continuations http://themonadreader.wordpress.com/2014/04/23/issue-23
- How Haskell can solve the integration problem 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

5.2.9 Sunroof

Report by:	Andrew Gill
Participants:	Jan Bracker
Status:	active

Sunroof is a Domain Specific Language (DSL) for generating JavaScript. It is built on top of the JS-monad, which, like the Haskell IO-monad, allows read and write access to external resources, but specifically JavaScript resources. As such, Sunroof is primarily a feature-rich foreign function API to the browser's JavaScript engine, and all the browser-specific functionality, like HTML-based rendering, event handling, and drawing to the HTML5 canvas.

Furthermore, Sunroof offers two threading models for building on top of JavaScript, atomic and blocking threads. This allows full access to JavaScript APIs, but using Haskell concurrency abstractions, like MVars and Channels. In combination with the push mechanism Kansas-Comet, Sunroof offers a great platform to build interactive web applications, giving the ability to interleave Haskell and JavaScript computations with each other as needed.



It has successfully been used to write smaller applications. These applications range from 2D rendering using the HTML5 canvas element, over small GUIs, up to executing the QuickCheck tests of Sunroof and displaying the results in a neat fashion. The development has been active over the past 6 months and there is a drafted paper submitted to TFP 2013.

Further reading

- Homepage: http://www.ittc.ku.edu/csdl/fpg/software/sunroof.html
- Tutorial: https://github.com/ku-fpg/sunroof-compiler/wiki/Tutorial
- Main Repository: https://github.com/ku-fpg/sunroof-compiler

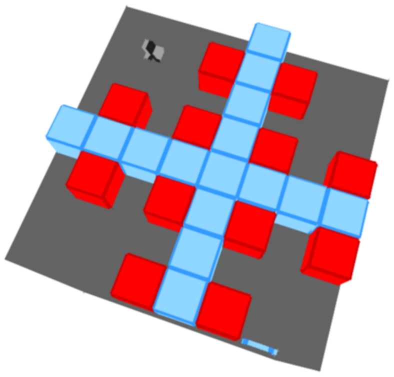
5.2.10 Blank Canvas

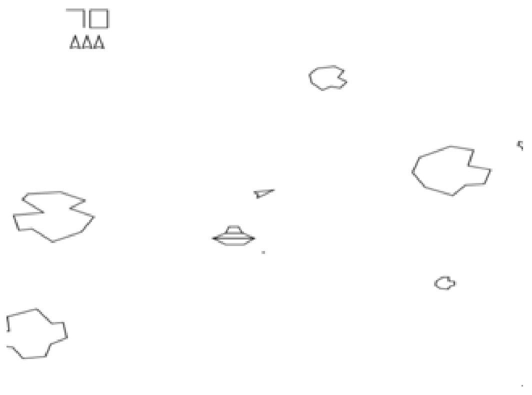
Report by:	Andrew Gill
Participants:	Justin Dawson, Mark Grebe, Ryan Scott, James Stanton, Jeffrey Rosenbluth, and Neil Sculthorpe
Status:	active

Blank Canvas is a Haskell binding to the complete HTML5 Canvas API. Blank Canvas allows Haskell users to write, in Haskell, interactive images onto their web browsers. Blank Canvas gives the user a single full-window canvas, and provides many well-documented functions for rendering images. Out of the box, Blank Canvas is pac-man complete – it is a platform for simple graphics, classic video games, and building more powerful abstractions that use graphics.

Blank Canvas was written in Spring 2012, as part of the preparation for a graduate-level functional programming class. In Fall 2012 and Fall 2013, we used Blank Canvas to teach Functional Reactive Programming. This was our first hint that the Blank Canvas library was faster than we expected, as we had hundreds of balls bouncing smoothly on the screen, much to the students' delight.

Blank Canvas has now been used by the students in four separate instances of our functional programming class. Students find it easy to understand, given the analog between the IO monad, and the remote Canvas monad, with student often choosing to use Blank Canvas for their end-of-semester project. To give two examples, one end-of-semester project was Omar Bari and Dain Vermaak's Isometric Tile Game, that can be rotated in 3D in real-time; another project was Blankeroids, a playable asteroids clone, written by Mark Grebe, on top of Yampa and yampa-canvas.





For more details, read the blank-canvas wiki.

Further reading

- https://hackage.haskell.org/package/blank-canvas
- https://github.com/ku-fpg/blank-canvas
- https://github.com/ku-fpg/blank-canvas/wiki

5.2.11 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, 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 was recently the focus of two successful Google Summer of Code projects, generously supported by the Haskell.org organization:

- The development of a searchable online database of PureScript code with type search and rendered documentation
- The addition of an exhaustivity and redundancy checker for pattern matches.

PureScript development is currently focussed on the following areas:

- Improving the performance of the compiler.
- Improving error messages.
- Enabling new backends (C++, Lua, Python, etc.)
- The development of new PureScript libraries

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

Further reading

https://github.com/purescript/purescript/

5.3 Haskell and Compiler Writing

5.3.1 MateVM

Report by:	Bernhard Urban
Participants:	Harald Steinlechner
Status:	looking for new contributors

MateVM is a method-based Java Just-In-Time Compiler. That is, it compiles a method to native code on demand (i.e. on the first invocation of a method). We use existing libraries:

hs-java for processing Java Classfiles according to *The Java Virtual Machine Specification.*

harpy enables runtime code generation for `i686` machines in Haskell, in a domain specific language style.

We believe that Haskell is suitable to implement compiler technologies. However, we have to jump between "Haskell world" and "native code world", due to the low-level nature of Just-In-Time compiler in a virtual machine. This poses some challenges when it comes to signal handling and other interesting rather low level operations. Not immediately visible, the task turns out to be well suited for Haskell although we experienced some tensions with signal handling and GHCi. We are looking forward to sharing our experience on this.

In the current state we are able to execute simple Java programs. The compiler eliminates the JavaVM stack via abstract interpretation, does a liveness analysis, linear scan register allocation and finally machine code emission. The software architecture enables easy addition of further optimization passes based on an intermediate representation.

Future plans are, to add an interpreter to gather profile information for the compiler and also do more aggressive optimizations (e.g. method inlining or stack allocation). An interpreter can also be used to enable speculation during compilation and, if such a speculation fails, compiled code can *deoptimize* to the interpreter.

Apart from that, features are still missing to comply as a JavaVM, most noteable are proper support for classloaders, floating point operations or threads. We would like to see a real base library such as GNU Classpath or the JDK running with MateVM some day. Other hot topics are Hoopl and Garbage Collection.

We are looking for new contributors! If you are interested in this project, do not hesitate to join us on IRC (`#MateVM @ OFTC`) or contact us on Github.

Further reading

- https://github.com/MateVM
- http://docs.oracle.com/javase/specs/jvms/se7/html/
- http://hackage.haskell.org/package/hs-java
- http://hackage.haskell.org/package/harpy
- http://www.gnu.org/software/classpath/
- http://hackage.haskell.org/package/hoopl-3.8.7.4
- http://en.wikipedia.org/wiki/Club-Mate

5.3.2 UUAG

Report by:	Atze Dijkstra
Participants:	ST Group of Utrecht University
Status:	stable, maintained

UUAG is the *Utrecht University Attribute Grammar* system. It is a preprocessor for Haskell that makes it easy to write *catamorphisms*, i.e., functions that do to any data type what *foldr* does to lists. Tree walks are defined using the intuitive concepts of *inherited* and *synthesized attributes*, while keeping the full expressive power of Haskell. The generated tree walks are *efficient* in both space and time.

An AG program is a collection of rules, which are pure Haskell functions between attributes. Idiomatic tree computations are neatly expressed in terms of copy, default, and collection rules. Attributes themselves can masquerade as subtrees and be analyzed accordingly (higher-order attribute). The order in which to visit the tree is derived automatically from the attribute computations. The tree walk is a single traversal from the perspective of the programmer.

Nonterminals (data types), productions (data constructors), attributes, and rules for attributes can be specified separately, and are woven and ordered automatically. These aspect-oriented programming features make AGs convenient to use in large projects.

The system is in use by a variety of large and small projects, such as the Utrecht Haskell Compiler UHC (→ 3.4), the editor Proxima for structured documents (http://www.haskell.org/communities/05-2010/html/report.html#sect6.4.5), the Helium compiler (http://www.haskell.org/communities/05-2009/html/report.html#sect2.3), the Generic Haskell compiler, UUAG itself, and many master student projects. The current version is 0.9.52.1 (January 2015), is extensively tested, and is available on Hackage. There is also a Cabal plugin for easy use of AG files in Haskell projects.

We recently implemented the following enhancements:

Evaluation scheduling. We have done a project to improve the scheduling algorithms for AGs. The previously implemented algorithms for scheduling AG computations did not fully satisfy our needs; the code we write goes beyond the class of OAGs, but the algorithm by Kennedy and Warren (1976) results in an undesired increase of generated code due to non-linear evaluation orders. However, because we know that our code belongs to the class of linear orderable AGs, we wanted to find and algorithm that can find this linear order, and thus lies in between the two existing approaches. We have created a backtracking algorithm for this which is currently implemented in the UUAG (`-aoag` flag).

Another approach to this scheduling problem that we implemented is the use of SAT-solvers. The scheduling problem can be reduced to a SAT-formula and efficiently solved by existing solvers. The advantage is that this opens up possibilities for the user to influence the resulting schedule, for example by providing a cost-function that should be minimized. We have also implemented this approach in the UUAG which uses Minisat as external SAT-solver (`-loag` flag).

We have recently worked on the following enhancements:

Incremental evaluation. We have just finished a Ph.D. project that investigated incremental evaluation of AGs. The target of this work was to improve the UUAG compiler by adding support for incremental evaluation, for example by statically generating different evaluation orders based on changes in the input. The project has lead to several publications, but the result has not yet been implemented into the UUAG compiler.

Further reading

- http://www.cs.uu.nl/wiki/bin/view/HUT/AttributeGrammarSystem
- http://hackage.haskell.org/package/uuagc

6 Development Tools

6.1 Environments

6.1.1 Haskell IDE From FP Complete

Report by:	Natalia Muska
Status:	available, stable

Since FP Complete^{TM} announced the launch of FP Haskell Center^{TM} (FPHC) in early September 2013, a lot of additions to the original IDE have been added and the pricing structure has changed dramatically. The new features and the pricing modifications are a direct result of community feedback. The changes were gradually rolled out over the past year.

As of October 1, 2014, all users of FPHC who are using it for non-commercial projects have free access under the new Open Publish model. This means that Open Publish accounts will automatically publish all projects on the FPHC site with each commit, similar to Github. This move is meant to make FPHC more valuable, and increase support for users sharing their work with the community. There are still paid subscriptions available for Commercial projects.

This is a current list of features included with the free version of FPHC:
- Create and Edit Haskell Projects,
- Open Projects from Git, FPHC, or Web
- Continuous Error and Type Information
- Hoogle and Haddock Integration
- Easy to use build system
- Vetted Stable Libraries
- Easy to Understand Error Messages
- No setup or install
- Free Community Support
- Push Projects to Git and GitHub
- Emacs Integration
- Shared Team Accounts
- Support for Sub Projects
- Multiple Repository Projects
- Deploy to FP Application Servers
- Large Project and Megarepos Support (new)
- Subscriptions include continuous refresh releases on new features, updates, bug fixes and free community support

Over the past year the feedback and activity on FPHC has been very positive. To ensure FPHC is meeting the demands of the Haskell community, FP complete is constantly seeking feedback and suggestions from users and the Haskell community.

Further reading

Visit www.fpcomplete.com for more information.

6.1.2 ghc-mod — Happy Haskell Programming

Report by:	Daniel Gröber
Status:	open source, actively developed

`ghc-mod` is both a backend program for enhancing editors and other kinds of development environments with support for Haskell, and an Emacs package providing the user facing functionality, internally called `ghc` for historical reasons. Other people have also developed numerous front ends for Vim and there also exist some for Atom and a few other proprietary editors.

This summer's two month `ghc-mod` hacking session was mostly spent (finally) getting a release supporting GHC 7.10 out the door as well as fixing bugs and adding full support for the *Stack* build tool.

Since the last report the *haskell-ide-engine* project has seen the light of day. There we are planning to adopt `ghc-mod` as a core component to use its environment abstraction.

The *haskell-ide-engine* project itself is aiming to be the central component of a unified Haskell Tooling landscape.

In the light of this `ghc-mod`'s mission statement remains the same but in the future it will be but one, important, component in a larger ecosystem of Haskell Tools.

We are looking forward to *haskell-ide-engine* making the Haskell Tooling landscape a lot less fragmented. However until this project produces meaningful results life goes on and `ghc-mod`'s ecosystem needs to be maintained.

Right now `ghc-mod` has only one core developer and a handful of occasional contributors. If *you* want to help make Haskell development even more fun come and join us!

Further reading

https://github.com/kazu-yamamoto/ghc-mod

6.1.3 haskell-ide-engine, a project for unifying IDE functionality

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

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

Features like type errors, linting, refactoring, and reformatting code 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. Inspiration is being taken from the work the Idris community has done toward an interactive editing environment as well. This is a deliberately broad scope, the initial versions will be very limited at first. The sooner we can get started the sooner we will have something concrete to criticise and improve.

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 or` hvr.

Further reading

- https://github.com/haskell/haskell-ide-engine
- 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/

6.1.4 HaRe — The Haskell Refactorer

Report by:	Alan Zimmerman
Participants:	Francisco Soares, Chris Brown, Stephen Adams, Huiqing Li, Matthew Pickering, Gracjan Polak

Refactorings are source-to-source program transformations which change program structure and organization, but not program functionality. Documented in catalogs and supported by tools, refactoring provides the means to adapt and improve the design of existing code, and has thus enabled the trend towards modern agile software development processes.

Our project, *Refactoring Functional Programs*, has as its major goal to build a tool to support refactorings in Haskell. The HaRe tool is now in its seventh major release. HaRe supports full Haskell 2010, and is integrated with (X)Emacs. All the refactorings that HaRe supports, including renaming, scope change, generalization and a number of others, are *module-aware*, so that a change will be reflected in all the modules in a project, rather than just in the module where the change is initiated.

Snapshots of *HaRe* are available from our GitHub repository (see below) and Hackage. There are related presentations and publications from the group (including LDTA'05, TFP'05, SCAM'06, PEPM'08, PEPM'10, TFP'10, Huiqing's PhD thesis and Chris's PhD thesis). The final report for the project appears on the University of Kent Refactoring Functional Programs page (see below).

There is also a Google+ community called HaRe, a Google Group called https://groups.google.com/forum/#!forum/hare and an IRC channel on freenode called *#haskell-refactorer*. IRC is the preferred contact method.

Current version of *HaRe* supports 7.10.2 and work is continuing to support GHC version 8.x forward. The new version makes use of *ghc-exactprint* library, which only has GHC support from GHC 7.10.2 onwards.

Development on the core *HaRe* is focusing is on making sure that deficiencies identified in the API Annotations in GHC used by *ghc-exactprint* are removed in time for GHC 8.0.1, so that the identity refactoring can cover more of the corner cases.

There is also a new *haskell-ide* project which will allow *HaRe* to operate as a plugin and will ease its integration into multiple IDEs.

Recent developments

- The current version is 8.2, which supports GHC 7.10.2 only, and was released in October 2015.
- Matthew Pickering has been deeply involved in the *ghc-exactprint* development, and successfully completed his Google Summer of Code project which involved bringing it up to standard, which has helped tremendously for *HaRe*.
- There is plenty to do, so anyone who has an interest is welcome to fork the repo and get stuck in.
- Stephen Adams is continuing his PhD at the University of Kent and will be working on data refactoring in Haskell.

Further reading

- http://www.cs.kent.ac.uk/projects/refactor-fp/
- https://github.com/RefactoringTools/HaRe
- https://github.com/alanz/ghc-exactprint
- http://mpickering.github.io/gsoc2015.html
- https://github.com/haskell/haskell-ide

6.1.5 ghc-exactprint

Report by:	Matthew Pickering
Participants:	Alan Zimmerman
Status:	Active, Experimental

`ghc-exactprint` aims to be a low-level foundation for refactoring tools. Unlike most refactoring tools, it works directly with the GHC API which means that it can understand any legal Haskell source file.

The program works in two phases. The first phase takes the output from the parser and converts all absolute source positions into relative source positions. This means that it is much easier to manipulate the AST as you do not have to worry about updating irrelevant parts of your program. The second phase performs the reverse process, it converts relative source positions back into absolute positions before printing the source file. The entire library is based around a free monad which keeps track of which annotations should be where. Each process is then a different interpretation of this structure.

In theory these two processes should be entirely separate but at the moment they are not entirely decoupled due to shortcomings we are addressing in GHC 8.0.

In order to verify our foundations, the program has been run on every source file on Hackage. This testing highlighted a number of bugs which have been fixed for GHC 7.10.2. Apart from a few outstanding issues with very rare cases, we can now confidently say that `ghc-exactprint` is capable of processing *any* Haskell source file.

Over the last few months Alan Zimmerman has integrated `ghc-exactprint` into HaRe(→ 6.1.4) whilst Matthew Pickering participated in Google Summer of Code to provide integration with `HLint`.

Both of these proceeded smoothly, and are now working.

`ghc-exactprint` has also been used for a proof of concept tool to migrate code forward for AMP and MRP, see link below.

Alan Zimmerman also presented `ghc-exactprint` at HIW2015, and Matthew Pickering at SkillsMatter in October. Links to the respective videos are provided below.

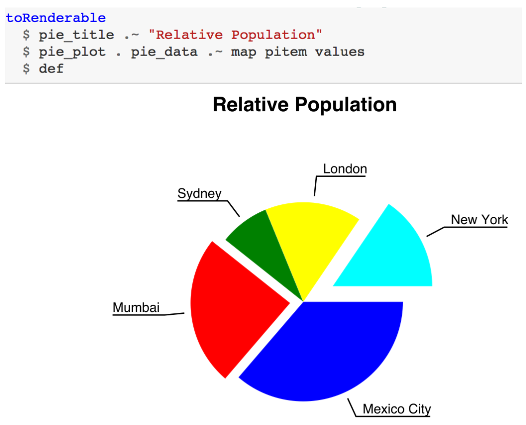
Further reading

- https://github.com/alanz/ghc-exactprint
- https://github.com/hvr/Hs2010To201x
- https://www.youtube.com/watch?v=U5_9mfQAUBo - HIW2015
- https://skillsmatter.com/skillscasts/6539-a-new-foundation-for-refactoring-ghc-exactprint - Skills Matter, (free) registration required

6.1.6 IHaskell: Haskell for Interactive Computing

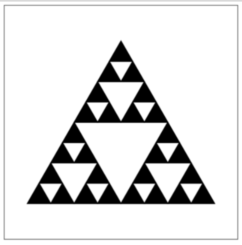
Report by:	Andrew Gibiansky
Status:	stable

IHaskell is an interactive interface for Haskell development. It provides a *notebook* interface (in the style of Mathematica or Maple). The notebook interface runs in a browser and provides the user with editable cells in which they can create and execute code. The output of this code is displayed in a rich format right below, and if it's not quite right, the user can go back, edit the cell, and re-execute. This rich format defaults to the same boring plain-text output as GHCi would give you; however, library authors will be able to define their own formats for displaying their data structures in a useful way, with the only limit being that the display output must be viewable in a browser (images, HTML, CSS, Javascript). For instance, integration with graphing libraries yields in-browser data visualizations, while integration with Aeson's JSON yields a syntax-highlighted JSON output for complex data structures.


```
toRenderable
  $ pie_title .~ "Relative Population"
  $ pie_plot . pie_data .~ map pitem values
  $ def
```

Implementation-wise, IHaskell is a language kernel backend for the Jupyter project, a *language-agnostic* protocol and set of frontends by which interactive code environments such as REPLs and notebooks can communicate with a language evaluator backend. IHaskell also provides a generic library for writing Jupyter kernels, which has been used successfully in the ICryptol project.

```haskell
-- We can draw diagrams, right in the notebook.
:extension NoMonomorphismRestriction
import Diagrams.Prelude

-- By Brent Yorgey
-- Draw a Sierpinski triangle!
sierpinski 1 = eqTriangle 1
sierpinski n =      s
                   ===
               (s ||| s) # centerX
  where s = sierpinski (n-1)

-- The `diagram` function is used to display them
diagram $ sierpinski 4
            # centerXY
            # fc black
          `atop` square 10
                 # fc white
```



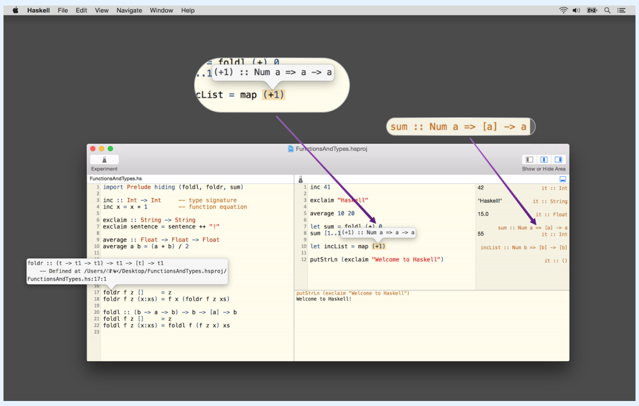
Integration with popular Haskell libraries can give us beautiful and potentially interactive visualizations of Haskell data structures. On one hand, this could range from simple things such as foldable record structures — imagine being able to explore complex nested records by folding and unfolding bits and pieces at a time, instead of trying to mentally parse them from the GHCi output. On the other end, we have interactive outputs, such as Parsec parsers which generate small input boxes that run the parser on any input they're given. And these things are just the beginning — tight integration with IPython may eventually be able to provide things such as code-folding in your REPL or an integrated debugger interface.

Further reading

https://github.com/gibiansky/IHaskell

6.1.7 Haskell for Mac

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



```haskell
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
```

```haskell
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!
```

```
sum :: Num a => [a] -> a
```

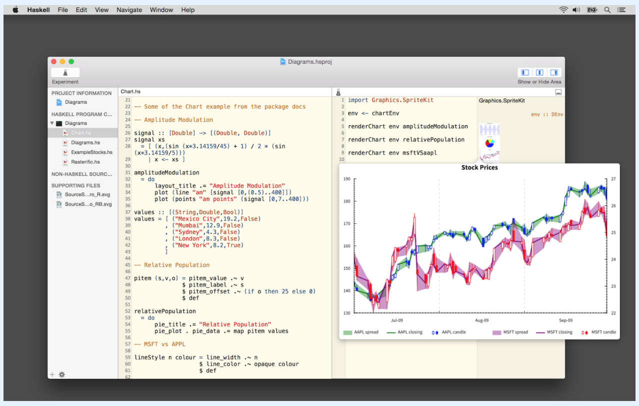
Haskell for Mac is an easy-to-use integrated programming environment for Haskell on OS X. It includes its own Haskell distribution and requires no further set up. It features interactive Haskell playgrounds to explore and experiment with code. Playground code is not only type-checked, but also executed 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 customisable themes. (Alternatively, users can also configure an external editor.) 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.

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 graphics. Haskell for Mac includes special support for the Rasterific, Diagrams, and Chart libraries. Graphical results are immediately displayed in the playground. They are also live and change as you modify the program code.

Moreover, Haskell for Mac includes a binding to the main parts of Apple's 2D animation and games framework SpriteKit. The Haskell binding to SpriteKit is purely functional using an immutable representation of the scene graph and pure transformation functions. SpriteKit scenes are rendered as interactive popover windows in the Haskell for Mac playground, where they react to keyboard, mouse, and trackpad events.


```
-- Some of the Chart example from the package docs

-- Amplitude Modulation

signal :: [Double] -> [(Double, Double)]
signal xs
  = [ (x,(sin (x*3.14159/45) + 1) / 2 * (sin (x*3.14159/5)))
    | x <- xs ]

amplitudeModulation
  = do
      layout_title .= "Amplitude Modulation"
      plot (line "am" [signal [0,(0.5)..400]])
      plot (points "am points" (signal [0,7..400]))

values :: [(String,Double,Bool)]
values = [ ("Mexico City",19.2,False)
         , ("Mumbai",12.9,False)
         , ("Sydney",4.3,False)
         , ("London",8.3,False)
         , ("New York",8.2,True)
         ]

-- Relative Population

pitem (s,v,o) = pitem_value .~ v
              $ pitem_label .~ s
              $ pitem_offset .~ (if o then 25 else 0)
              $ def

relativePopulation
  = do
      pie_title .= "Relative Population"
      pie_plot . pie_data .= map pitem values

-- MSFT vs APPL

lineStyle n colour = line_width .~ n
                   $ line_color .~ opaque colour
                   $ def
```

```
import Graphics.SpriteKit

env <- chartEnv

renderChart env amplitudeModulation

renderChart env relativePopulation

renderChart env msftVSaapl
```

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.

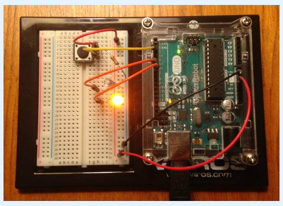
The current version of Haskell for Mac is based on GHC 7.8.4 and LTS Haskell 2.20. We are currently preparing an update to GHC 7.10 and LTS Haskell 3. We are working on several extensions to Haskell for Mac, which we will announce once we have concrete release dates.

Further reading

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

6.1.8 Haskino

Report by:	Mark Grebe
Participants:	Andrew Gill
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. Haskino presents two complimentary ways of developing programs for the Arduino.

The first method allows programming of an Arduino tethered to a host computer through a serial connection. This work started with Levent Erkök's hArduino package. To this we have added our strong Remote Monad concepts, which provide a more efficient method of communication with the board. We have also replaced the Firmata serial communication protocol and firmware with a new protocol and firmware which also allow for more efficient communication, and are expandable to meet the needs of our second programming method.

The second method of programming the Arduino uses a deep embedding to out-source entire groups of commands and control-flow idioms. These programs may then be stored in EEPROM on the board, and executed from startup, with no connection to the host computer required. A Haskell programmer can start program development with the first method, which allows for convenient prototyping and debugging. The program can then be moved to the second method, with the entire computation being performed on the Arduino, and not requiring the host computer.

The development has been active over the past 6 months and there is a paper accepted for publication at PADL 2016.

Further reading

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

6.2 Code Management

6.2.1 Darcs

Report by:	Eric Kow
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.

Our most recent release was Darcs 2.10 (April 2015). This release includes the new `darcs rebase` command (for merging and amending patches that would be hard to do with patch theory alone), numerous optimisations and performance improvements, a `darcs convert` command for switching to and from Git, as well as general improvements to the user interface.

In more recent news, we have gotten back into what we hope to be a habit of regular Darcs hacking sprints, with a recent one in Paris this September and in Seville this upcoming January. We also have a new maintainer, Guillaume Hoffmann. Guillaume has been with the project for five years now and has been working on bringing new developers into the project and making links between Darcs and other communities.

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/2.10

6.2.2 `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.

`cab` now supports GHC 7.10.

Further reading

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

6.3 Interfacing to other Languages

6.3.1 java-bridge

Report by:	Julian Fleischer
Status:	active development

The Java Bridge is a library for interfacing the Java Virtual Machine with Haskell code and vice versa. It comes with a rich DSL for discovering and invoking Java methods and allows to set up callbacks into the Haskell runtime. If exported via the FFI it is also possible to use Haskell libraries from within the JVM natively.

The package also offers a bindings generator which translates the API of a Java class or package into a Haskell API. Using the bindings generator it is possible to generate a Haskell module with a clean Haskell API that invokes Java behind the scenes. Typical conversions, for example byte arrays to lists or Java maps to lists of key value pairs, are taken care of. The generated bindings and predefined conversions are extensible by defining appropriate type class instances accordingly.

While the documentation for the bindings generator still needs improvement, the overall library is in a quite usable state.

The java bridge is published under the MIT license and available via hackage as `java-bridge`.

Further reading

If you want to know more about the inner workings: The Java Bridge has been created as part of a bachelor thesis which you can access at http://page.mi.fu-berlin.de/scravy/bridging-the-gap-between-haskell-and-java.pdf.

6.3.2 fficxx

Report by:	Ian-Woo Kim
Participants:	Ryan Feng
Status:	Actively Developing

fficxx (“eff fix”) is an automatic haskell Foreign Function Interface (FFI) generator to C++. While haskell has a well-specified standard for C FFI, interfacing C++ library to haskell is notoriously hard. The goal of fficxx is to ease making haskell-C++ binding and to provide relatively nice mapping between two completely different programming paradigms.

To make a C++ binding, one write a haskell model of the C++ public interfaces, and then fficxx automatically generates necessary boilerplate codes in several levels: C++-C shims, C-haskell FFI, low level haskell type representation for C++ class/object and high level haskell type and typeclass representation and some casting functions. The generated codes are organized into proper haskell modules to minimize name space collision and packaged up as cabal packages.

The tool is designed to adapt different configurations and unique needs, such as splitting bindings into multiple cabal packages and renaming classes and functions to resolve some obstacles that are originated from naming collision, which is quite inevitable in making an FFI library.

The information of a C++ library can be written in terms of simple haskell expressions, aiming at good usability for ordinary haskell users. For example, if we have a C++ library which has the following interface:

```
class A {
public:
  A();
  virtual void Foo();
};
class B : public A {
public:
  B();
  virtual void Bar();
};
```

one provide the model in terms of haskell data type defined in fficxx library:

```
a = myclass "A" [] mempty Nothing
    [ Constructor [] Nothing
    , Virtual void_ "Foo" [ ] Nothing ]
b = myclass "B" [a] mempty Nothing
    [ Constructor [] Nothing
    , Virtual void_ "Bar" [] Nothing ]
```

One of the projects that successfully uses fficxx is HROOT which is a haskell binding to the ROOT library. ROOT is a big C++ histogramming and statistical analysis framework. Due to fficxx, the HROOT package faithfully reflects the ROOT C++ class hierarchy, and the user from C++ can use the package relatively easily.

fficxx is available on hackage and being developed on the author's github (http://github.com/wavewave/fficxx). In 2013, with Ryan Feng, we tried to make fficxx more modernized with more transparent support of various C/C++ data types, including consistent multiple pointer/reference operations and function pointers. fficxx is still being in progress in incorporating the new pointer operations. C++ template support is now planned.

Further reading

http://ianwookim.org/fficxx

6.4 Deployment

6.4.1 Cabal and Hackage

Report by:	Duncan Coutts

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.

Hackage is a distribution point for Cabal packages. It is an online archive of Cabal packages which can be used via the website and client-side software such as cabal-install. Hackage enables users to find, browse and download Cabal packages, plus view their API documentation.

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.

Looking forward

We would like to encourage people considering contributing to take a look at the bug tracker on github, 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: http://www.haskell.org/cabal
- Hackage package collection: http://hackage.haskell.org/
- Bug tracker: https://github.com/haskell/cabal/

6.4.2 Stackage: the Library Dependency Solution

Report by:	Natalia Muska
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 700 packages available in Stackage. The official site is www.stackage.org.

Stackage Update: Stackage is an infrastructure to create stable builds of complete package sets referred to as "snapshots." Stackage provides users with the assurance that their packages will always build, will actually compile, all tests suites pass, and all will work across three GHC versions (7.8, 7.6, and 7.4). Users of a snapshot verified by Stackage can expect all packages to install the first time.

Each snapshot is given a unique hash which is a digest of that snapshot's package set. Snapshots don't change. Once a hash is provided, it refers only to that snapshot. So if a user writes a project using snapshot aba1b51af, and in two months switches to another machine and builds their project with aba1b51af, it will succeed.

For package authors, Stackage gives them the valuable knowledge that their package builds and tests successfully across the current, stable and old GHC versions. Library authors have guarantees that users of Stackage can easily use their library and can do so on a reasonable number of GHCs. Authors are also informed when a newly uploaded package breaks theirs, meaning it's time to update the package for it to be included in the latest snapshot.

Recently Stackage added some additional features including Haddock documentation and cabal.config files. By including Haddock documentation in Stackage all new exclusive snapshots have Haddock links allowing users to view documentation of all packages included in the snapshot. This means users can generally view everything in one place, on one high-availability service. By creating a cabal.config link on snapshot pages, Stackage users don't have to change their remote-repo field.

Stackage Server: Before Stackage Server, use of Stackage was limited to either manually downloading a project and building it all locally, or by using FP Haskell Center. With Stackage Server, users are able to go to the server web site and pick a snapshot. On the build is a simple copy/paste line to use as a Cabal repo, to replace the users existing remote-repo line.

When a new package is released and has been properly updated, users can go to the Stackage home page and get the latest snapshot and update their repo. The Stackage server also supports the uploading of custom snapshots, this allows a company, a Linux distribution, an organization, a university, or just as a general hacker who wants to keep all their projects under one package set, to maintain their own custom series of snapshots, and also make it available to other people. Then the burden will be on those users to make sure it builds, rather than the recommended and Stackage maintained snapshots.

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.

6.4.3 Haskell Cloud

Report by:	Gideon Sireling

Haskell Cloud is an OpenShift cartridge for deploying Haskell on Red Hat's open source PaaS cloud. It includes GHC 7.8, cabal-install, Gold linker, and a choice of pre-installed frameworks - a full list can be viewed on the Wiki.

Using a Haskell Cloud cartridge, existing Haskell projects can be uploaded, build, and run from the cloud with minimal changes. Ongoing development is focused on OpenShift's upcoming Docker release and GHC 7.10.

Further reading

- https://bitbucket.org/accursoft/haskell-cloud
- http://www.haskell.org/haskellwiki/Web/Cloud#OpenShift
- https://blog.openshift.com/functional-programming-in-the-cloud-how-to-run-haskell-on-openshift/

6.5 Others

6.5.1 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 (→ **??**) builds on ghc-heap-view.

Since version 0.5.3, ghc-heap-view also supports GHC 7.8.

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

6.5.2 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.

Further reading

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

6.5.3 Tasty

Report by:	Roman Cheplyaka
Participants:	Michael LaCorte, Sergey Vinokurov, and many others
Status:	actively maintained

Tasty is a modern testing framework for Haskell. As of May 2015, 230 hackage packages use Tasty for their tests. We've heard from several companies that use Tasty to test their Haskell software.

What's new since the last HCAR?

- Tasty now sets the number of parallel running tests equal to the number of available capabilities (i.e. the number set by `-N`) by default. As always, that can be changed with `-j`.
- Printing test results on Windows used to be slow, but now it's fast!
- Tasty-HUnit now has a new function, `testCaseSteps`, which lets you annotate a multi-step unit test. Here's an example:

```
main =
  defaultMain $
  testCaseSteps "Multi-step test" $
  \step -> do

  step "Step 1"
  -- do something

  step "Step 2"
  -- do something else
```

  As a reminder from the last HCAR, Tasty-HUnit no longer uses the original HUnit package; instead it reimplements the relelvant subset of its API.
- The way Tasty-Golden works internally has changed. There are a few consequences (see the CHANGELOG for details); an interesting one is that you can now update golden files in parallel.
  Also, if a golden file doesn't exist, it will be created automatically. You'll see a message like

```
UnboxedTuples:                    OK (0.04s)
  Golden file did not exist; created
```

  This is convenient when adding new tests.

Further reading

- For more information about Tasty and how to use it, please consult the README at http://bit.ly/tasty-home
- Tasty has a mailing list http://bit.ly/tasty-ml and an IRC channel (`#tasty` on FreeNode), where you can get help with Tasty.

6.5.4 Automatic type inference from JSON

Report by:	Michal J. Gajda
Status:	stable

This rapid software development tool `json-autotype` interprets JSON data and converts them into Haskell module with data type declarations.

```
$ json-autotype input.json -o JSONTypes.hs
```

The generated declarations use automatically derived Aeson class instances to read and write data directly from/to JSON strings, and facilitate interaction with growing number of large JSON APIs.

Generated parser can be immediately tested on an input data:

```
$ runghc JSONTypes.hs input.json
```

The software can be installed directly from Hackage.

It uses sophisticated union type unification, and robustly interprets most ambiguities using clever typing.

The tool has reached maturity this year, and thanks to automated testing procedures it seems to robustly infer types for all JSON inputs considered valid by Aeson.

The author welcomes comments and suggestions at ⟨mjgajda@gmail.com⟩.

Further reading

http://hackage.haskell.org/packages/json-autotype

6.5.5 Exference

Report by:	Lennart Spitzner
Status:	experimental, active development

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

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

Output:

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

Input:

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

Output:

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

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

There are two primary use-cases for Exference:

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

Since the last HCAR, development has slowed down but continued. Additions include minor optimizations, support for type declarations, improvements to the interface (simplifications of the expression, etc.) and expansion of the default environment.

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

6.5.6 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 0.1.6.2, the immediate goal being switching to pipes for searching the directory trees.

Further reading

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

6.5.7 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 have a paper in the 2015 Haskell Symposium that discusses these ideas in more detail. We have identified six variants of the design pattern, including a weak remote monad, which calls primitives one at a time, and a strong remote monad, which bundles primitives together in a robust way that amortizes the cost of calling remote capabilities. We have also identified over a dozen use-cases of this design pattern being used in practice.

Further reading

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

6.5.8 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:

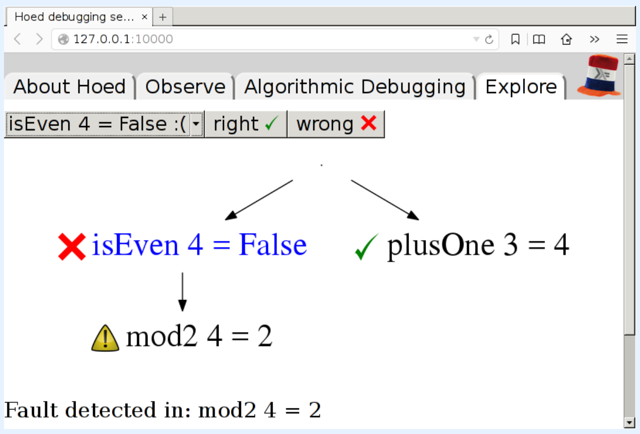
```
import Debug.Hoed.Pure

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.



✗ isEven 4 = False

✓ plusOne 3 = 4

⚠ mod2 4 = 2

Fault detected in: mod2 4 = 2

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.

Further reading

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

7 Libraries, Applications, Projects

7.1 Language Features

7.1.1 Conduit

Report by:	Michael Snoyman
Status:	stable

While lazy I/O has served the Haskell community well for many purposes in the past, it is not a panacea. The inherent non-determinism with regard to resource management can cause problems in such situations as file serving from a high traffic web server, where the bottleneck is the number of file descriptors available to a process.

The left fold enumerator was one of the first approaches to dealing with streaming data without using lazy I/O. While it is certainly a workable solution, it requires a certain inversion of control to be applied to code. Additionally, many people have found the concept daunting. Most importantly for our purposes, certain kinds of operations, such as interleaving data sources and sinks, are prohibitively difficult under that model.

The conduit package was designed as an alternate approach to the same problem. The root of our simplification is removing one of the constraints in the enumerator approach. In order to guarantee proper resource finalization, the data source must always maintain the flow of execution in a program. This can lead to confusing code in many cases. In conduit, we separate out guaranteed resource finalization as its own component, namely the ResourceT transformer.

Once this transformation is in place, data producers, consumers, and transformers (known as Sources, Sinks, and Conduits, respectively) can each maintain control of their own execution, and pass off control via coroutines. The user need not deal directly with any of this low-level plumbing; a simple monadic interface (inspired greatly by the pipes package) is sufficient for almost all use cases.

Since its initial release, conduit has been through many design iterations, all the while keeping to its initial core principles. Since the last HCAR, we've released version 1.2. This release introduces two changes: it adds a stream fusion implementation to allow much more optimized runs for some forms of pipelines, and uses the codensity transform to provide better behavior of monadic bind.

Additionally, much work has gone into `conduit-combinators` and `streaming-commons`, both of which are packages introduced in the last HCAR.

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

The library is available on Hackage. There is an interactive tutorial available on the FP Complete School of Haskell. You can find many conduit-based packages in the Conduit category on Hackage as well.

Further reading

- http://hackage.haskell.org/package/conduit
- https://www.fpcomplete.com/user/snoyberg/library-documentation/conduit-overview
- http://hackage.haskell.org/packages/archive/pkg-list.html#cat:conduit

7.1.2 GHC type-checker plugin for kind Nat

Report by:	Christiaan Baaij
Status:	actively developed

As of GHC version 7.10, GHC's type checking and inference mechanisms can be enriched by plugins. This particular plugin enriches GHC's knowledge of arithmetic on the type-level. Specifically it allows the compiler to reason about *equalities* of types of kind `GHC.TypeLits.Nat`.

GHC's type-checker's knowledge of arithmetic is virtually non-existent: it doesn't know addition is associative and commutative, that multiplication distributes over addition, etc. In a dependently-typed language, or in Haskell using singleton types, one can provide proofs for these properties and use them to type-check programs that depend on these properties in order to be (type-)correct. However, most of these properties of arithmetic over natural number are elementary school level knowledge, and it is cumbersome and tiresome to keep on providing and proving them manually. This type-checker plugin adds the knowledge of these properties to GHC's type-checker.

For example, using this plugin, GHC now knows that:

```
(x + 2)^(y + 2)
```

is equal to:

```
4*x*(2 + x)^y + 4*(2 + x)^y + (2 + x)^y*x^2
```

The way that the plugin works, is that it normalises arithmetic expressions to a normal form that very much resembles *Cantor normal form for ordinals*(http://en.wikipedia.org/wiki/Ordinal_arithmetic#Cantor_normal_form). Subsequently, it perform a simple syntactic equality of the two expressions. Indeed, in the example above, the latter expression is the normal form of the former expression.

The main test suite for the plugin can be found at: https://github.com/christiaanb/ghc-typelits-natnormalise/blob/master/tests/Tests.hs. It demonstrates what kind of *correct* code can be written without type equality annotations, or the use of `unsafeCoerce`.

One important aspect of this plugin is that it only enriches the type checker's knowledge of equalities, but not *in*equalities. That is, it does not allow GHC to solve constraints such as:

```
CmpNat (x + 2) (x + 3) ~ 'LT
```

The plugin is available on hackage, for GHC version 7.10 and higher:

```
$ cabal update
$ cabal install ghc-typelits-natnormalise
```

What's new since last HCAR:

- Support for interacting with other type-checker plugins, the first being http://hackage.haskell.org/package/ghc-typelits-extra.
- Prove more equalities (http://hackage.haskell.org/package/ghc-typelits-natnormalise-0.3.2/changelog).

Development focus for the plugin is on: proving more equalities, further testing, and improving its test suite.

Further reading

- http://hackage.haskell.org/package/ghc-typelits-natnormalise
- http://hackage.haskell.org/package/base/docs/GHC-TypeLits.html

7.1.3 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. On my branch [1] the Core language and type inferenace have already been updated according to the description in our ICFP'13 paper [2]. Accordingly, *all* type-level constructs are simultaneously kind-level constructs, as there is no distinction between types and kinds. Specifically, GADTs and type families will be 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.

As of the time of writing, the branch works on many examples but is still a bit buggy. I am currently in the process of merging into GHC's master branch; expect this to land in HEAD by the end of November.

After this phase, I will embark on working a proper Π-binder into the language, much along the lines of Adam Gundry's thesis on the topic [3]. Having Π would give us "proper" dependent types, and there would be no more need for singletons. A sampling of what I hope is possible when this work is done is online [4], excerpted here:

$$
\begin{array}{l}
\textbf{data}\ \mathsf{Vec} :: * \to \mathsf{Integer} \to *\ \textbf{where} \\
\quad \mathsf{Nil}\ \ :: \mathsf{Vec}\ a\ 0 \\
\quad (:::) :: a \to \mathsf{Vec}\ a\ n \to \mathsf{Vec}\ a\ (1\ '\!+ n) \\
\mathit{replicate} :: \pi\ n.\ \forall a.\ a \to \mathsf{Vec}\ a\ n \\
\mathit{replicate}\ @0\ \_ = \mathsf{Nil} \\
\mathit{replicate}\qquad x = x ::: \mathit{replicate}\ x
\end{array}
$$

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

Further reading

- [1]: https://github.com/goldfirere/ghc, the `nokinds` branch.
- [2]: *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
- [3]: *Type Inference, Haskell and Dependent Types*, by Adam Gundry. PhD Thesis, 2013. https://personal.cis.strath.ac.uk/adam.gundry/thesis/
- [4]: https://github.com/goldfirere/nyc-hug-oct2014/blob/master/Tomorrow.hs
- Haskell Implementors' Workshop 2014 presentation on Dependent Haskell. Slides: http://www.cis.upenn.edu/~eir/talks/2014/hiw-dependent-haskell.pdf; Video: https://www.youtube.com/watch?v=O805YjOsQjI
- Repo for presentation on Dependent Haskell at the NYC Haskell Users' Group: https://github.com/goldfirere/nyc-hug-oct2014
- Wiki page with elements of the design: https://ghc.haskell.org/trac/ghc/wiki/DependentHaskell

7.1.4 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. The core of Yampa includes combinators to create constant signals, apply pointwise (or time-wise) functions to signals, access the running time of a signal function, introduce delays and create loopbacks (carrying present output as future input). These systems can also be dynamic: their structure can change by using *switching* combinators, which enable the application of a different signal function at some point in the execution. Combined with combinators to deal with signal function collections, this enables a form of dynamic FRP in which new signals can be introduced, frozen, unfrozen, removed and altered at will.

Yampa is designed to guarantee *causality*: the value of an output signal at a time t can only depend on values of input signals at times $[0, t]$. Yampa restricts access to other signals only to the immediate past, by letting signals functions carry *state* for the future. FRP signal functions implement the Arrow and ArrowLoop typeclasses, making it possible to use both the arrow notation and arrow combinators. 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 the component's internal state. Components can be serialized, applied in parallel, etc.

Unlike other implementations of FRP, Yampa enforces a strict separation of effects and pure transformations. All IO code must exist outside the Signal Functions, making Yampa 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. Examples of the latter include Keera Studios' Magic Cookies! (https://goo.gl/0A8z6i), a board game for Android written in Haskell and avalible via Google Play for Android store.

Yampa is actively maitained. The last updates have focused on introducing documentation, structuring the code to facilitate navigation, eliminating legacy code superceeded by other Haskell libraries, and increasing code quality in general. Over the years, performance in FRP has been an active topic of discussion and Yampa has been optimised heavily (games like Haskanoid have been clocked at over 700 frames per second on a standard PC). Also because Yampa is *pure*, the introduction of parallelism is straightforward. In future versions, the benchmarking package `criterion` will be used to evaluate and increase performance. We encourage all Haskellers to participate 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.

Extensions to Arrowized Functional Reactive Programming are an active research topic. The Functional Programming Laboratory at the University of Nottingham is working on several extensions to make Yampa more general and modular, facilitate other uses cases, increase performance and work around existing limitations. To collaborate with our research on FRP, please contact Ivan Perez at and Henrik Nilsson at .

7.2 Education

7.2.1 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.

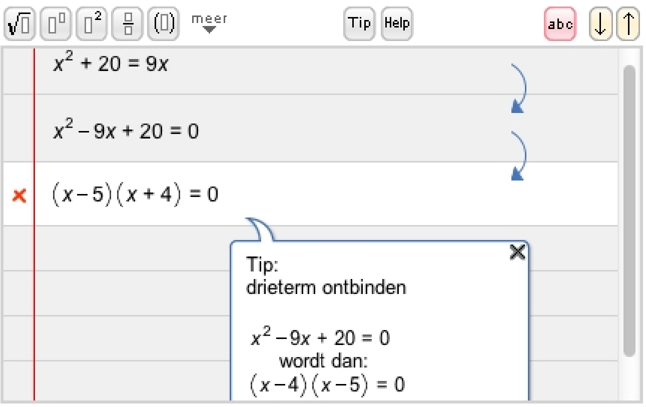
Contact

⟨J.Hage@uu.nl⟩

7.2.2 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) and learning environments. 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), MathDox, and the Math-Bridge system. We have developed several domain reasoners based on this framework, including reasoners for mathematics, linear algebra, logic, learning Haskell (the Ask-Elle programming tutor) and evaluating Haskell expressions, and for practicing communication skills (the serious game Communicate!, second/right screenshot).



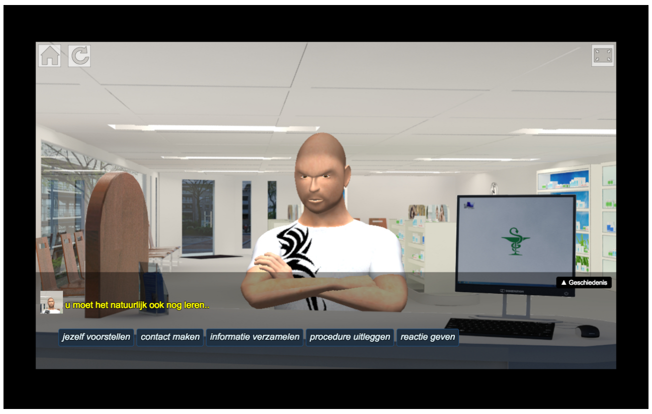
$x^2 + 20 = 9x$

$x^2 - 9x + 20 = 0$

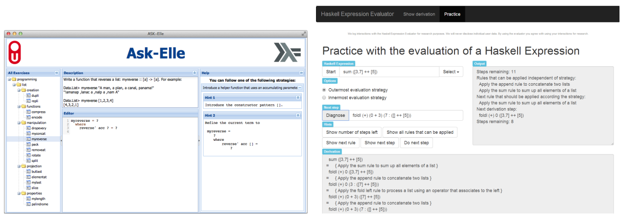
✗ $(x-5)(x+4) = 0$

Tip:
drieterm ontbinden

$x^2 - 9x + 20 = 0$
wordt dan:
$(x-4)(x-5) = 0$



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. Tim Olmer has developed a tutor in which a student can practice with evaluating Haskell expressions. Finally, Hieke Keuning has developed a programming tutor for imperative programming.


Ask-Elle
Haskell Expression Evaluator
Practice with the evaluation of a Haskell Expression

We are continuing our research in various directions. We are investigating feedback generation for axiomatic proofs for propositional logic, and are planning to add this to our logic tutor. We have just started on a statistics tutor. 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. We have also released our domain reasoner for mathematics and logic 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.
- Tim Olmer, Bastiaan Heeren, Johan Jeuring. Evaluating Haskell expressions in a tutoring environment. Trends in Functional Programming in Education 2014.
- Hieke Keuning, Bastiaan Heeren, Johan Jeuring. Strategy-based feedback in a programming tutor. Computer Science Education Research Conference (CSERC 2014).
- Johan Jeuring, Thomas van Binsbergen, Alex Gerdes, Bastiaan Heeren. Model solutions and properties for diagnosing student programs in Ask-Elle. Computer Science Education Research Conference (CSERC 2014).

7.3 Parsing and Transforming

7.3.1 HERMIT

Report by:	Andrew Gill
Participants:	Andrew Farmer, Neil Sculthorpe, Ryan Scott
Status:	active

The Haskell Equational Reasoning Model-to-Implementation Tunnel (HERMIT) is an NSF-funded project being run at KU (→9.7), which aims to improve the applicability of Haskell-hosted Semi-Formal Models to High Assurance Development. Specifically, HERMIT uses a Haskell-hosted DSL and a refinement DSL to perform rewrites directly on Haskell Core, the GHC internal representation.

Over the summer, we reworked our user-level refinement DSL to use Haskell, by making use of the remote monad (→6.5.7). This new shell, dubbed the Black Shell, replaced the REPL with GHCi, and brings the full power of Haskell DSLs to new API. The port has been completed, and we hope to release HERMIT, with the Black Shell, shortly.

Further reading

https://github.com/ku-fpg/hermit

7.3.2 Utrecht Parser Combinator Library: uu-parsinglib

Report by:	Doaitse Swierstra
Status:	actively developed

With respect to the previous version the code for building interleaved parsers was split off into a separate package `uu-interleaved`, such that it can be used by other parsing libraries too. Based on this another small package `uu-options` was constructed which can be used to parse command line options and files with preferences. The internals of these are described in a technical report: http://www.cs.uu.nl/research/techreps/UU-CS-2013-005.html.

As an example of its use we show how to fill a record from the command line. We start out by defining the record which is to hold the options to be possibly set:

```
data Prefers  = Agda | Haskell deriving Show
data Address = Address { city_ :: String
                       , street_ :: String }
               deriving Show
data Name    = Name { name_ :: String
                    , prefers_ :: Prefers
                    , ints_ :: [Int]
                    , address_ :: Address }
               deriving Show
$ (deriveLenses '' Name)
$ (deriveLenses '' Address)
```

The next thing to do is to specify a default record containing the default values:

```
defaults = Name "Doaitse" Haskell []
                (Address "Utrecht"
                         "Princetonplein")
```

Next we define the parser for the options, by specifying each option:

```
oName =
      name    `option` ( "name",    pString,
                         "Name")
   <> ints    `options`( "ints",    pNaturalRaw,
                         "Some numbers")
   <> prefers `choose` [("agda",     Agda,
                         "Agda preferred")
                       ,("haskell", Haskell,
                         "Haskell preferred")
                       ]
   <> address `field`
                    (  city   `option` ("city", pString,
                                        "Home city")
                    <> street `option` ("street", pString,
                                        "Home Street")
                    )
```

Finally when running this parser by the command *run* ((\$*defaults*) <\$> *mkP* *oName*) on the string ("-int=7 -city=Tynaarlo -i 5 -agda -i3 " ++ "-street=Zandlust") the result is

```
Name { name_    = Doaitse
     , prefers_ = Agda
     , ints_    = [7, 5, 3]
     , address_ = Address
                  { city_   = Tynaarlo
                  , street_ = Zandlust }
     }
```

If you make a mistake in the list of options, automatic error reporting and correction steps in and you get the following message:

```
./OptionsDemo --street=Zandlust -nDoaitse
-i3 --city=Tynaarlo
 --name     [Char]  optional  Name
 --ints     Int     recurring Some numbers
 Choose at least one from(
 --agda         required  Agda preferred
 --haskell      required  Haskell preferred
    )
 --city     [Char]  optional  Home city
 --street   [Char]  optional  Home Street
 --
 --  Correcting steps:
 --    Inserted  "-a" at position 70
 --    expecting one of
       [  "--agda", "--agda=", "--haskell",
          "--haskell=", "--ints=", "--ints",
          "-i", "-h", "-a"]
 --    Inserted  EOT at position 70
 --    expecting EOT
```

Features

- Combinators for easily describing parsers which produce their results online, do not hang on to the input and provide excellent error messages. As such they are "surprise free" when used by people not fully aware of their internal workings.
- Parsers "correct" the input such that parsing can proceed when an erroneous input is encountered.
- The library basically provides the to be preferred applicative interface and a monadic interface where this is really needed (which is hardly ever).
- No need for *try*-like constructs which make writing `Parsec` based parsers tricky.
- Scanners can be switched dynamically, so several different languages can occur intertwined in a single input file.
- Parsers can be run in an interleaved way, thus generalizing the merging and permuting parsers into a single applicative interface. This makes it e.g. possible to deal with white space or comments in the input in a completely separate way, without having to think about this in the parser for the language at hand (provided of course that white space is not syntactically relevant).

Future plans

Future versions will contain a check for grammars being not left-recursive, thus taking away the only remaining source of surprises when using parser combinator libraries. This makes the library even greater for use in teaching environments. Future versions of the library, using even more abstract interpretation, will make use of computed look-ahead information to speed up the parsing process further.

Contact

If you are interested in using the current version of the library in order to provide feedback on the provided interface, contact ⟨doaitse@swierstra.net⟩. There is a low volume, moderated mailing list which was moved to ⟨parsing@lists.science.uu.nl⟩ (see also http://www.cs.uu.nl/wiki/bin/view/HUT/ParserCombinators).

7.3.3 Generalized Algebraic Dynamic Programming

Report by:	Christian Höner zu Siederdissen
Status:	usable, active development

Generalized Algebraic Dynamic Programming 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. Below, we describe three highlights, our systems offers:

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 rigerous 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 (→ 7.11.1) 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. She can now easily encode problems based on set-like inputs and obtain 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:

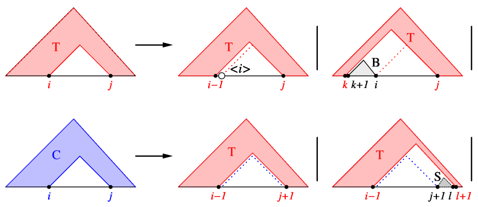
```
s (f <<< s % n ||| g <<< n ... h)
```

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.

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 (sub-optimal) 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.

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

7.4 Mathematics, Numerical Packages and High Performance Computing

7.4.1 Rlang-QQ

Report by:	Adam Vogt
Status:	active development

Rlang-QQ is intended to make it easier to call R from Haskell programs. This allows access to a large number of R packages for graphing, statistics or other uses. Rlang-QQ provides a quasiquoter which runs the R interpreter and tries to translate values between the two languages.

Haskell expressions can be referenced from R using syntax like `$(take 10 [1.0 .. ])`. Haskell variables can also be passed in by prefixing them with `hs_`: `hs_x` refers to `x`. Values that can be taken out of a Haskell `x :: Chan t` are accessible using `ch_x`. When the R code has an assignment such as `hs_x <- f()`, the quasiquote evaluates to an HList record which contains the result from `f()`.

Future work may include supporting the serialization of more data types between the two languages, passing data between the two runtimes in-memory instead of through files, and doing inference when possible on the R-code to restrict the types of the Haskell values that are serialized or deserialized.

Further reading

- http://hackage.haskell.org/package/Rlang-QQ
- http://www.r-project.org/
- http://www.haskell.org/haskellwiki/Quasiquotation

7.4.2 arb-fft

Report by:	Ian Ross
Status:	actively developed

This package started as an experiment to see how close a pure Haskell FFT implementation could get to FFTW ("the Fastest Fourier Transform in the West"). The result is a library that can do fast Fourier transforms for arbitrarily sized vectors with performance within a factor of about five of FFTW.

Future plans mostly revolve around making things go faster! In particular, the next thing to do is to write an equivalent of FFTW's `genfft`, a metaprogramming tool to generate fast straight-line code for transforms of specialised sizes. Other planned work includes implementing real-to-complex and real-to-real transforms, multi-dimensional transforms, and some low-level optimisation.

Further reading

- http://hackage.haskell.org/package/arb-fft
- http://www.skybluetrades.net/haskell-fft-index.html

7.4.3 hblas

Report by:	Carter Tazio Schonwald
Participants:	Stephen Diehl and Csernik Flaviu Andrei
Status:	Actively Developed

`hblas` is high level, easy to extend BLAS/LAPACK FFI Binding for Haskell.

`hblas` has several attributes that *in aggregate* distinguish it from alternative BLAS/LAPACK bindings for Haskell.

1. Zero configuration install
2. FFI wrappers are written in Haskell
3. Provides the fully generality of each supported BLAS/LAPACK routine, in a type safe wrapper that still follows the naming conventions of BLAS and LAPACK.
4. Designed to be easy to extend with further bindings to BLAS/LAPACK routines (because there are many many specialized routines!)
5. Adaptively choses between unsafe vs safe foreign calls based upon estimated runtime of a computation, to ensure that long running `hblas` ffi calls interact safely with the GHC runtime and the rest of an application.
6. `hblas` is not an end user library, but is designed to easily interop with any array library that supports storable vectors.

Further reading

- http://www.wellposed.com
- http://www.github.com/wellposed/hblas
- http://hackage.haskell.org/package/hblas

7.4.4 HROOT

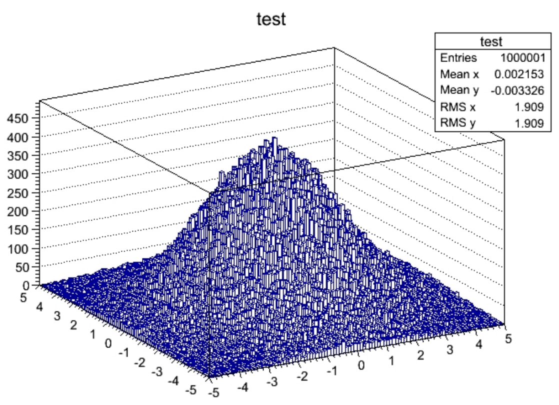
Report by:	Ian-Woo Kim
Status:	Actively Developing

HROOT is a haskell binding to ROOT framework by fficxx, a haskell-C++ binding generator tool. ROOT (http://root.cern.ch) is an OOP framework for data analysis and statistics, which is developed at CERN. The ROOT system provides a set of OO frameworks with all the functionality needed to handle and analyze large amounts of data in a very efficient way. ROOT is a de facto standard physics analysis tool in high energy physics experiments.

This haskell binding to ROOT provides an industrial-strength statistical analysis libraries to the haskell community. The haskell code for using HROOT is very straightforward to both haskell and C++ programmers thanks to the fficxx binding generator tool. The following is a sample code and a resultant histogram for histogramming a 2D gaussian distribution:

```
import Data.Random.Distribution.Normal
import HROOT

main :: IO ()
main = do
  tcanvas <- newTCanvas "Test" "Test" 640 480
  h2 <- newTH2F "test" "test"
               100 (-5.0) 5.0 100 (-5.0) 5.0
  let dist1 = Normal (0 :: Double)
                     (2 :: Double)
  let go n | n < 0 = return ()
           | otherwise = do
               histfill dist1 dist2 h2
               go (n-1)
  go 1000000
  draw h2 "lego"
  saveAs tcanvas "random2d.pdf" ""

histfill :: Normal Double -> TH2F -> IO ()
histfill dist1 hist = do
  x <- sample dist1
  y <- sample dist1
```

```
    fill2 hist x y
    return ()
```



Until ghc 7.6, HROOT cannot be used in interpreter mode of ghc, due to the linker problem. Now with ghc 7.8, ghci now uses the standard system linker for dynamically loaded library. Thus, our current focus is to have full ghc interpreter support for making HROOT a really useful analysis framework. In addition, we keep importing features from ROOT to available haskell functions.

Further reading

http://ianwookim.org/HROOT

7.4.5 Numerical

Report by:	Carter Tazio Schonwald
Status:	actively developed

The Numerical project, starting with the `numerical` package, has the goal of providing a general purpose numerical computing substrate for Haskell.

To start with, the `numerical` provides an extensible set of type classes suitable for both dense and sparse multi dimensional arrays, high level combinators for writing good locality code, and some basic matrix computation routines that work on both dense and sparse matrix formats.

The core Numerical packages, including `numerical`, are now in public pre-alpha as of mid May 2014, with on going active work as of November 2014.

Development of the numerical packages is public on github, and as they stabilize, alpha releases are being made available on hackage.

Further reading

- http://www.wellposed.com
- http://www.github.com/wellposed/numerical
- http://hackage.haskell.org/package/numerical

7.4.6 petsc-hs

Report by:	Marco Zocca
Status:	experimental, actively developed

PETSc (http://www.mcs.anl.gov/petsc/) is an extensive C library for scientific computation. It provides a unified interface to distributed datastructures and algorithms for parallel solution of numerical problems, e.g. (non-)linear equation systems, time integration of dynamical systems, nonlinear (constrained) optimization. It is built upon MPI but abstracts it “out of sight”; however the API lets advanced users interleave computation and communication in order to experiment with resource usage and performance.

Many applications using PETSc are concerned with the solution of discretized PDEs for modelling physical phenomena, but the numerical primitives offered can be applied in many other contexts as well.

The aim of `petsc-hs` is to provide a compositional, type- and memory-safe way to interact with this library. The bindings are based on `inline-c` (https://hackage.haskell.org/package/inline-c) for quick experimentation with the C side.

Development of `petsc-hs` is public on github as of October 2015.

At present (November 2015), bindings for most of the basic functionality are available, memory pointers have been made lexically scoped and rudimentary exception handling is in place; the library is dynamically linked and can be tested with GHCi.

The immediate development plans are to move out of the experimental phase: currently the effort is concentrated on representing distributed mutable array operations and overall giving the library a more declarative interface while at the same time encapsulating the C version’s best programming practices. Once this will be in place, a number of example PETSc programs will be provided and the API will be specialized to various use cases. Due to the multidisciplinary nature of this work, contributions, comments and test cases are more than welcome.

Further reading

https://github.com/ocramz/petsc-hs

7.4.7 combinat

Report by:	Balázs Kőműves
Status:	actively developed

The `combinat` package is a broad-reaching combinatorics library. It provides functions to generate, manipulate, count and visualize various combinatorial objects, for example: trees, partitions, compositions, lattice paths, power series, permutations, braids, Young tableaux, and so on.

There is ASCII visualization for most structures, which makes it convenient to work in GHCi, and also `graphviz` and/or `diagrams` for some of them (the latter ones in a separate package).

Development is mostly done in short bursts, based mainly on the current (always changing) interests of the author.

Further reading

- http://hackage.haskell.org/package/combinat
- http://hackage.haskell.org/package/combinat-diagrams

7.5 Data Types and Data Structures

7.5.1 HList — A Library for Typed Heterogeneous Collections

Report by:	Adam Vogt
Participants:	Oleg Kiselyov, Ralf Lämmel, Keean Schupke

HList is a comprehensive, general purpose Haskell library for typed heterogeneous collections including extensible polymorphic records and variants. HList is analogous to the standard list library, providing a host of various construction, look-up, filtering, and iteration primitives. In contrast to the regular lists, elements of heterogeneous lists do not have to have the same type. HList lets the user formulate statically checkable constraints: for example, no two elements of a collection may have the same type (so the elements can be unambiguously indexed by their type).

An immediate application of HLists is the implementation of open, extensible records with first-class, reusable, and compile-time only labels. The dual application is extensible polymorphic variants (open unions). HList contains several implementations of open records, including records as sequences of field values, where the type of each field is annotated with its phantom label. We and others have also used HList for type-safe database access in Haskell. HList-based Records form the basis of OOHaskell. The HList library relies on common extensions of Haskell 2010. HList is being used in AspectAG (http://www.haskell.org/communities/11-2011/html/report.html#sect5.4.2), typed EDSL of attribute grammars, and in Rlang-QQ.

The October 2012 version of HList library marks the significant re-write to take advantage of the fancier types offered by GHC 7.4 and 7.6. HList now relies on promoted data types and on kind polymorphism.

Since the last update, there have been several minor releases. These include features such as support for ghc-7.8 as well as additional syntax for the pun quasiquote.

Further reading

- HList repository: http://code.haskell.org/HList/
- HList: http://okmij.org/ftp/Haskell/types.html#HList
- OOHaskell: https://web.archive.org/web/20130129031410/http://homepages.cwi.nl/~ralf/OOHaskell

7.5.2 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.

7.5.3 fixplate

Report by:	Balázs Kőműves
Status:	experimental

The `fixplate` package is a re-implementation of Neil Mitchell’s `uniplate` generic programming library, to work on data types realized as fixed points of functors (as opposed to plain recursive data types). It turns out that Functor, Foldable and Traversable instances are enough for this style of generic programming.

The original motivation for this exercise was the ability to add extra data to the nodes of an existing tree, motivated by attribute grammars. Recursion schemes also fit here very well, though they are less powerful.

Apart from the standard traversals, the library also provides a generic zipper, generic tries, generic tree hashing, a generic expression pretty-printer and generic tree visualization. The library itself is fully Haskell98-compatible, though some GHC extensions can make it more convenient to use.

Further reading

http://hackage.haskell.org/package/fixplate

7.5.4 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 paper and a somewhat more recent, slightly longer, tutorial covering type-level programming as well as the use of this library, are available.

Further reading

- `generics-sop` package: https://hackage.haskell.org/package/generics-sop/
- Tutorial (summer school lecture notes): https://github.com/kosmikus/SSGEP/
- Paper: http://www.andres-loeh.de/TrueSumsOfProducts/

7.6 Databases and Related Tools

7.6.1 Persistent

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

The last HCAR announcement was for the release of Persistent 2.0, featuring a flexible primary key type.

Since then, 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 usefeul 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 (→ 7.6.2), a similar project.

Persistent is recommended to Yesod (→ 5.2.2) 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

7.6.2 Groundhog

Report by:	Boris Lykah
Status:	stable

Groundhog is a library for mapping user defined datatypes to the database and manipulating them in a high-level typesafe manner. It is easy to plug Groundhog into an existing project since it does not need modifying a datatype or providing detailed settings. The schema can be configured flexibly which facilitates integration with existing databases. It supports composite keys, indexes, references across several schemas. Just one line is enough to analyze the type and map it to the table. The migration mechanism can automatically check, initialize, and migrate database schema. Groundhog has backends for Sqlite, PostgreSQL, and MySQL.

Unlike Persistent (→ 7.6.1) it maps the datatypes instead of creating new ones. The types can be polymorphic and contain multiple constructors. It allows creating sophisticated queries which might include arithmetic expressions, functions, and operators. The database-specific operators, for example, array-related in PostgreSQL are statically guaranteed to run only for PostgreSQL connection. Its support for the natural and composite keys is implemented using generic embedded datatype mechanism.

Groundhog has got several commercial users which have positive feedback. Most of the recent changes were done to meet their needs. The new features include PostgreSQL geometric operators, Fractional, Floating, and Integral instances for lifted expressions, logging queries, references to tables not mapped to Haskell datatype, default column values, and several utility functions.

Further reading

- Tutorial,
  http://www.fpcomplete.com/user/lykahb/groundhog
- Homepage, http://github.com/lykahb/groundhog
- Hackage package,
  http://hackage.haskell.org/package/groundhog

7.6.3 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

7.6.4 HLINQ - LINQ for Haskell

Report by:	Mantas Markevicius
Participants:	Mike Dodds, Jason Reich
Status:	Experimental

HLINQ is a Haskell implementation of the LINQ database query framework [1] modelled on Cheney *et al*’s T-LINQ system for F# [2]. Database queries in HLINQ are written in a syntax close to standard Haskell do notation:

```
getAge people = do                    getAge = [||do
p <- people                           p <- people db
guard ((name p) == "Edna")            guard ((name p) == "Edna")
return (age p)                        return (age p)||]
```

Queries can be composed using Template Haskell splicing operators, while type-safety rules provide additional correctness guarantees. Additionally, HLINQ is built on the HDBC library and uses prepared SQL statements protecting it against most SQL injection type attacks. Furthermore queries are avalanche-safe, meaning that for any query only a single SQL statement will be generated. Our system is in prototype stage, but microbenchmarks show performance competitive with HaskellDB.

The project is hosted on GitHub [3], with a technical report planned soon.

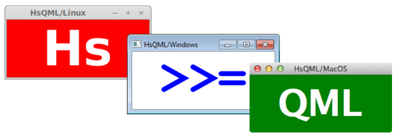
Further reading

1. Microsoft LINQ: https://msdn.microsoft.com/en-us/library/bb397926.aspx
2. Cheney, James, Sam Lindley, and Philip Wadler. "A practical theory of language-integrated query." ACM SIGPLAN Notices. Vol. 48. No. 9. ACM, 2013.
3. https://github.com/juventietis/HLINQ

7.7 User Interfaces

7.7.1 HsQML

Report by:	Robin KAY
Status:	active development



HsQML provides access to a modern graphical user interface toolkit by way of a binding to the cross-platform Qt Quick framework.

The library focuses on mechanisms for marshalling data between Haskell and Qt’s domain-specific QML language. The intention is that QML, which incorporates both a declarative syntax and JavaScript code, can be used to design and animate the front-end of an application while being able to easily interface with Haskell code for functionality.

Status The latest version at time of press is `0.3.3.0`. Changes released since the previous edition of this report include support for rendering custom OpenGL graphics onto QML elements, facilities for managing object life-cycles with weak references and finalisers, and a number of bug fixes. It has been tested on the major desktop platforms: Linux, Windows, and MacOS.

Further reading

http://www.gekkou.co.uk/software/hsqml/

7.7.2 Gtk2Hs

Report by:	Daniel Wagner
Participants:	Hamish Mackenzie, Axel Simon, Duncan Coutts, Andy Stewart, and many others
Status:	beta, actively developed

Gtk2Hs is a set of Haskell bindings to many of the libraries included in the Gtk+/Gnome platform. Gtk+ is an extensive and mature multi-platform toolkit for creating graphical user interfaces.

GUIs written using Gtk2Hs use themes to resemble the native look on Windows. Gtk is the toolkit used by Gnome, one of the two major GUI toolkits on Linux. On Mac OS programs written using Gtk2Hs are run by Apple's X11 server but may also be linked against a native Aqua implementation of Gtk.

Gtk2Hs features:

- Automatic memory management (unlike some other C/C++ GUI libraries, Gtk+ provides proper support for garbage-collected languages)
- Unicode support
- High quality vector graphics using Cairo
- Extensive reference documentation
- An implementation of the "Haskell School of Expression" graphics API
- Bindings to many other libraries that build on Gtk: gio, GConf, GtkSourceView 2.0, glade, gstreamer, vte, webkit

Recent efforts include increasing the coverage of the gtk3 bindings, as well as myriad miscellaneous bugfixes. Thanks to all who contributed!

Further reading

- News and downloads: http://haskell.org/gtk2hs/
- Development version: `darcs get` http://code.haskell.org/gtk2hs/

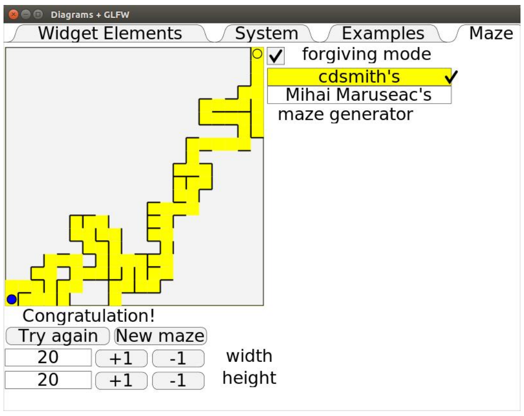
7.7.3 LGtk: Lens GUI Toolkit

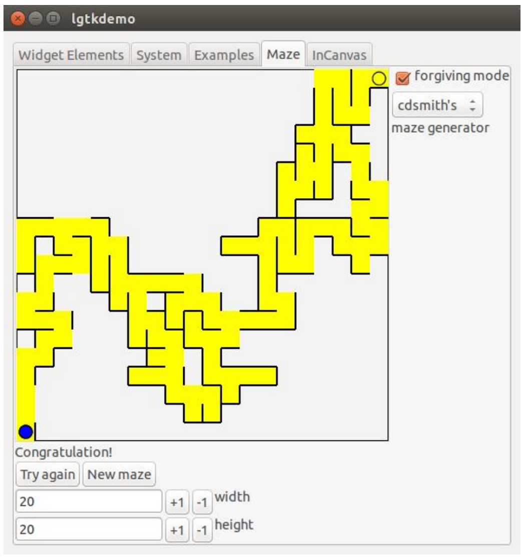
Report by:	Péter Diviánszky
Participants:	Csaba Hruska
Status:	experimental, actively developed

LGtk is a GUI Toolkit with the following goals:

- Provide a Haskell EDSL for declarative description of interactive graphical applications
- Provide an API for custom widget design
- Provide a playground for high-level declarative features like derived state-save and undo-redo operations and type-driven GUI generation

There is a demo application which presents the current features of LGtk.





Changes in lgtk-0.8 since the last official announcement:

- New features
  - New GLFW backend. One consequence is that the dependency on Gtk is not strict any more.
  - Canvas widgets rendering diagrams composed with the diagrams library. Mouse and keyboard events are also supported.
  - Widget toolkit generated with the diagrams library.
  - Slider widgets
- Architectural changes

  - Updated demo application
  - Switch from data-lens to Edward Kmett's lens library
  - Upgrade to work with GHC 8.2
  - Repository moved to GitHub
- Inner changes
  - Generalized and cleaned up interface of references
  - Cleaned up widget interface
  - More efficient reference implementation

Further reading

- haskell.org wiki page: http://www.haskell.org/haskellwiki/LGtk
- Haddock documentation on HackageDB: http://hackage.haskell.org/package/lgtk
- Wordpress blog: http://lgtk.wordpress.com/
- GitHub repository: https://github.com/divipp/lgtk

7.7.4 threepenny-gui

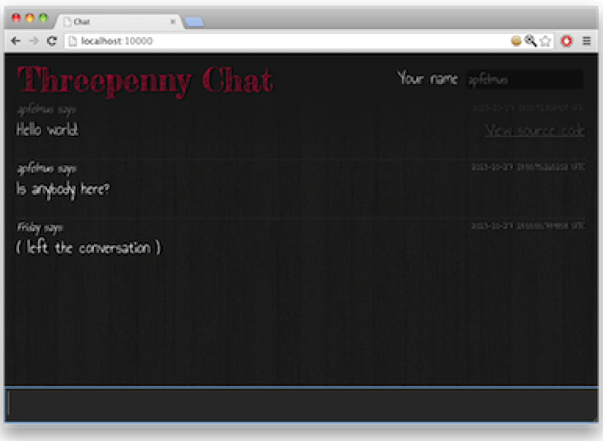
Report by:	Heinrich Apfelmus
Status:	active development

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.

Status

The project is alive and kicking, the latest release is version `0.6.0.3`. You can download the library from Hackage 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.

Compared to the previous report, no major changes have been made. A bug related to garbage collection of event handlers has been fixed, and the library has been updated to work with the current Haskell ecosystem.

Current development

The library is still very much in flux, significant API changes are likely in future versions. The goal is to make GUI programming as simple as possible, and that just needs some experimentation.

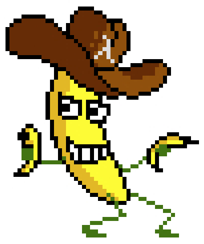
While Threepenny uses the web browser as a display, the goal was always to provide an environment for developing desktop applications. Recentely, a new platform for developing desktop applications with JavaScript has emerged, called Electron. I have successfully managed to connect Threepenny with the Electron platform, but I don't know how to best integrate this with the Haskell ecosystem, in particular Cabal. If you can offer any help with this, please let me know.

Further reading

- Project homepage: http://wiki.haskell.org/Threepenny-gui
- Example code: https://github.com/HeinrichApfelmus/threepenny-gui#examples
- Application gallery: http://wiki.haskell.org/Threepenny-gui#Gallery

7.7.5 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. With the release of version `1.0.0.0`, the development of the reactive-banana library has reached a milestone! I finally feel that the library does all the things that I wanted it to do.

In particular, compared to the previous report, the library now implements garbage collection for dynamically switched events. Also, the API no longer uses a phantom parameter to keep track of starting times; instead, a monadic approach is used. This simplifies the API for dynamic event switching, at the cost of requiring monadic types for some first-order combinators like `stepper`.

Additionally, there has been a small change concerning the semantics of the `Event` type: It is no longer possible to have multiple simultaneous occurrences in a single event stream. This forces the programmer to be more thoughtful about simultaneous event occurrences, a common source of bugs. The expressivity is the same, the old semantics can be recovered by using lists as occurrences.

Current development. With the library being complete, is there anything left to do? Well, of course, a library is never complete! However, my future focus will lie more on applications of FRP, rather than the implementation of the FRP primitives. For instance, I want to make more use of FRP in my `threepenny-gui` project, which is a library for writing graphical user interfaces in Haskell (→ 7.7.4). In turn, this will probably lead to improvements in the reactive-banana library, be it API revisions or performance tuning.

Further reading

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

7.7.6 fltkhs - GUI bindings to the FLTK library

Report by:	Aditya Siram
Status:	active

The `fltks` project is a set of bindings to the FLTK C++ toolkit (www.fltk.org). Coverage is fairly complete (85%) and it is easy to install and use. The main goal of this effort is to provide a low-cost, hassle-free way of creating self-contained, native GUI applications in pure Haskell that are portable to Windows, Linux and OSX.

FLTK was chosen because it is a mature toolkit and designed to be lightweight, portable and self-contained. In turn, `fltks` inherits these qualities with the additional benefit of having almost no dependencies outside of *base* and FLTK itself. This makes it very easy to get up and running with `fltks`.

`fltks` is also designed to be easy to use and learn. It tries to accomplish this by providing an API that matches the FLTK API as closely as possible so that a user can look up the pre-existing FLTK documentation for some function and in most cases be able to “guess” the corresponding Haskell function that delegates to it. Additionally `fltks` currently ships with 15 demos which are exact ports of demos shipped with the FLTK distribution so the user can study the code side-by-side. In most cases there is direct correspondence.

`fltks` is also extensible in a couple of ways. Firstly, the user can create custom GUI widgets in pure Haskell by simply overriding some key C++ functions with Haskell functions. Secondly, it is easy to add third-party widgets without touching the core bindings. Meaning if there is a useful FLTK widget that is not part of the FLTK distribution, the user can easily wrap it and publish it as a separate package without ever touching these bindings. Hopefully this fosters contribution allowing `fltks` to keep up with the FLTK ecosystem and even outpace it since users are now able to create new widgets in pure Haskell.

Ongoing work includes not only covering 100% of the API and porting all the demos but also adding support for FLUID (http://en.wikipedia.org/wiki/FLUID), the FLTK GUI builder. Haskellers will then be able to take any existing FLTK app which uses FLUID to build the user interface and migrate it to Haskell.

Contributions are welcome!

Further reading

https://hackage.haskell.org/package/fltkhs

7.7.7 wxHaskell

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

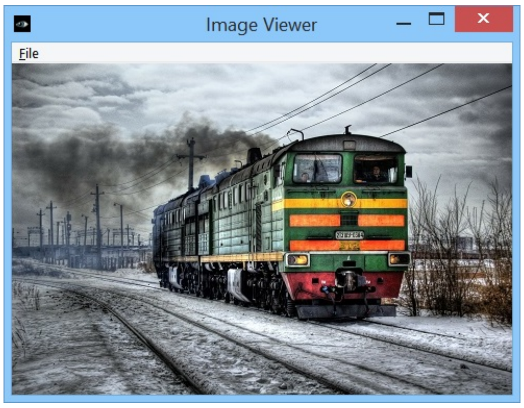
wxHaskell

Since the previous HCAR, wxHaskell versions 0.92 and 0.92.1 were released; functionality has been added and bugs were solved. Windows users may be glad to hear that wxHaskell is now very easy to install, so if you found it too hard to install, try again with one of the new installer packages. For further developments, check our GitHub repository. 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

7.8 Graphics and Audio

7.8.1 vect

Report by:	Balázs Kőműves
Status:	mostly stable

The `vect` package is low-dimensional linear algebra library intended specifically for computer graphics. It provides types and operations in 2, 3 and 4 dimensions, and is more-or-less feature-complete. OpenGL support is available as a separate package.

The library is intentionally monomorphic, providing as base fields the concrete types Float and Double. The monomorphicity makes life easier for both the user and the compiler, and we think that for graphics these two types cover most of the typical use cases. Nevertheless, a third, polymorphic version may be added in the future (until that happens, there is a polymorphic fork on Hackage).

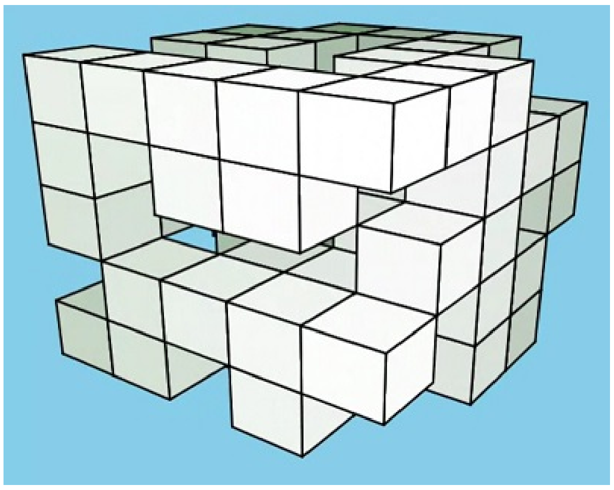
Further reading

◦ http://hackage.haskell.org/package/vect
◦ http://hackage.haskell.org/package/vect-opengl

7.8.2 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.



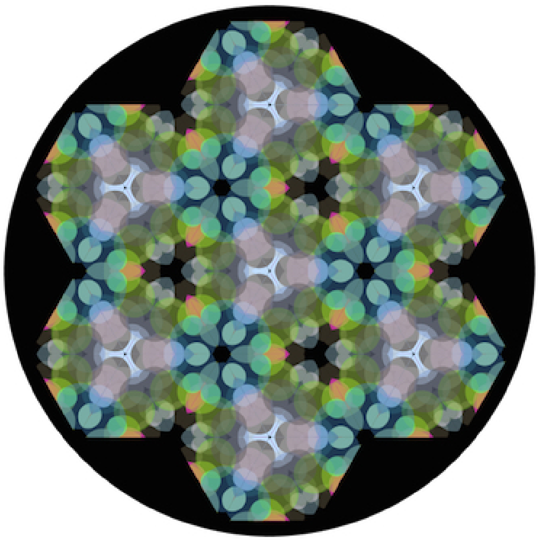
What’s new

There has not yet been a new major release of diagrams since version 1.3 in April, but work has continued apace. Here is a sampling of new features already in diagrams HEAD or currently being worked on:

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

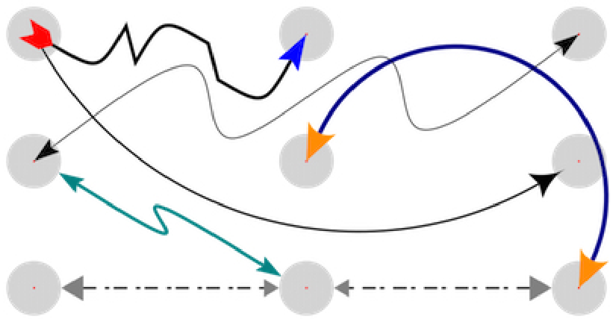
`diagrams-pandoc`, a pandoc filter which can automatically compile diagrams code included inline in pandoc documents, had its first release to Hackage.

We are also working on using `stack` to create a system for easier, more reproducible builds, which will benefit both users and developers, and form the basis for much more comprehensive continuous integration testing.



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.

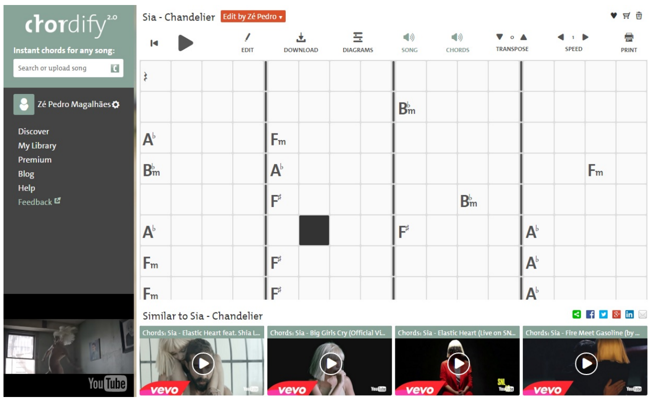


Further reading

- 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

7.8.3 Chordify

Report by:	José Pedro Magalhães
Participants:	W. Bas de Haas, Dion ten Heggeler, Gijs Bekenkamp, Tijmen Ruizendaal
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, the Haskell program HarmTrace takes these features and computes the chords. HarmTrace uses a model of Western tonal harmony to aid in the chord selection. At beat positions where the audio matches a particular chord well, this chord is used in final transcription. However, in case there is uncertainty about the sounding chords at a specific position in the song, the HarmTrace harmony model will select the correct chords based on the rules of tonal harmony.

We've recently completely redesigned Chordify and now showcase featured songs, popular songs in your country, and artist pages. We've also made some changes to the chord editing feature, making it easier to copy-paste edits, and letting users change the measure of the song. We plan to use the edits to improve the algorithm itself, and to implement a system that merges edits from various users into one single corrected version.

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

Further reading

http://chordify.net

7.8.4 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 synth Csound. It strives to be as simple and responsive as it can be. Features include almost all Csound build in audio units support, composable GUIs, FRP for event scheduling, MIDI and OSC support and many others. The library was updated for GHC-7.10.

200+ beautiful instruments are implemented. See the csound-catalog package. Each instrument is ready for real-time usage. Three drum machines are implemented. There is a library of standard effects. It can be used as a guitar processor.

With Csound it inherits many cutting edge sound synth techniques like granular synthesis or hyper vectorial synthesis, ambisonics.

The csound-expression is a Csound code generator. The flexible nature of Csound (it's written in C and has wonderful API) allows to use the produced code on any desktop OS, Android, iOS, Raspberry Pi, Unity, within many other languages. We can create audio engines with Haskell.

The library was presented at the Russian Function programming conference 2015 and at the International Csound Conference 2015.

The future plans for the library is to bring it on stage and make some audio installations with it, to improve documentation. I've created some music with the library. You can listen to it on the soundcloud https://soundcloud.com/anton-kho.

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

Further reading

https://github.com/anton-k/csound-expression

7.8.5 hmidi

Report by:	Balázs Kőműves
Status:	stable

The `hmidi` package provides bindings to the OS-level MIDI services, allowing Haskell programs to communicate with physical or virtual MIDI devices, for example MIDI keyboards, controllers, synthesizers, or music software.

Supported operating systems are Mac OS X and Windows. Linux (ALSA) support may be added at some future time.

An example application is provided by the `launchpad-control` package, which provides a high-level interface to the Novation Launchpad MIDI controller.

Further reading

- http://hackage.haskell.org/package/hmidi
- http://hackage.haskell.org/package/launchpad-control

7.8.6 Glome

Report by:	Jim Snow
Status:	New Version of Glome Raytracer

Glome is a ray tracer I wrote quite some time ago. The project had been dormant for about five years until a few months ago when I decided to fix some long-standing bugs and get it back into a state that compiles with recent versions of GHC. I got a little carried away, and ended up adding some major new features.

First, some background. Glome is a ray tracer, which renders 3d images by tracing rays from the camera into the scene and testing them for intersection with scene objects. Glome supports a handful of basic primitive types including planes, spheres, boxes, triangles, cones, and cylinders. It also has a number of composite primitives that modify the behavior of other primitives, such as CSG difference and intersection.

One of the more interesting composite primitives is a BIH-based accelleration structure, which sorts primitives into a hierarchy of bounding volumes. This allows for scenes with a very large number of primitives to be rendered efficiently.

Major new changes to Glome are a re-factoring of the shader code so that it is now possible to define textures in terms of user-defined types and write your own shader (though the default should be fine for most uses), a new tagging system, some changes to the front-end viewer application (which uses SDL now instead of OpenGL), and a new triangle mesh primitive type.

Tagging requires a bit of explanation. When a ray intersects with something in the scene, Glome returns a lot of information about the properties of the location where the ray hit, but until recently it didn't give much of a clue as to what exactly the ray hit. For 3D rendering applications, you don't usually care, but for many computational geometry tasks you do very much care.

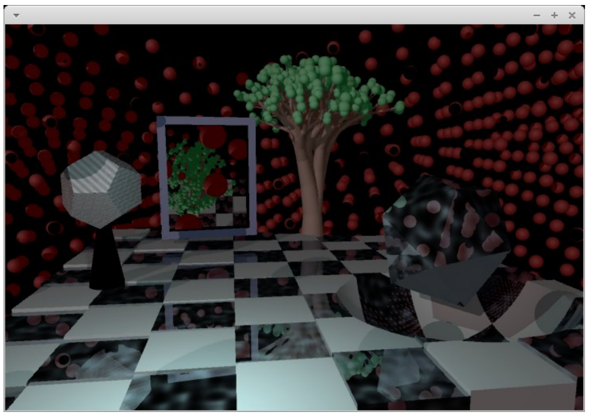
The new tagging system makes it possible to associate any 3D primitive with a tag, such that the tag is returned along with any ray intersection that hit the wrapped primitive. Tags are returned in a list, so that it's possible to have a heirarchy of tagged objects.

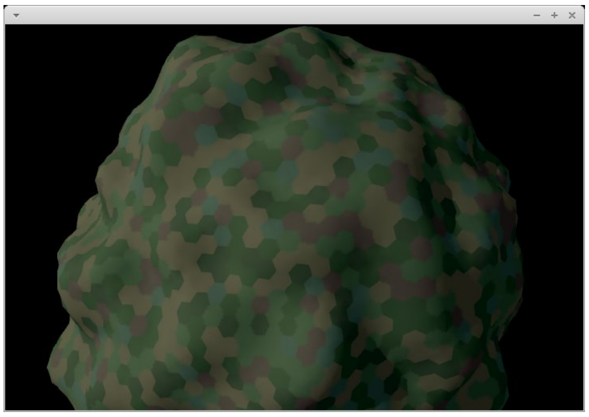
As an example of tags in action, I tagged some of the objects in Glome's default test scene, and instrumented the viewer so that clicking on the image causes a ray to be traced into the scene from the cursor's location, and then we print any tags returned by the ray intersection test. (Tags can be any type, but for illustrative purposes, the test scene uses strings.)

An interesting feature of the tagging system is that you don't necessarily have to click directly on the object to get back the tag; you could also click on the image of the object reflected off of some other shiny object in the scene.

Even though Glome is still a bit too slow for practical interactive 3D applications (I've been able to get around 2-3 FPS at 720x480 for reasonably complex scenes on a fairly fast machine), tags should at least make it easier to write interactive applications when Moore's law catches up.

Glome is split into three packages: GloveVec, a vector library, GlomeTrace, the ray-tracing engine, and GlomeView, a simple front-end viewer application. All are available on hackage or via github under a GPLv2 license.





Further reading

- https://github.com/jimsnow/glome
- http://www.haskell.org/haskellwiki/Glome

7.9 Text and Markup Languages

7.9.1 lhs2TEX

Report by:	Andres Löh
Status:	stable, maintained

This tool by Ralf Hinze and Andres Löh is a preprocessor that transforms literate Haskell or Agda code into 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

7.9.2 pulp

Report by:	Daniel Wagner
Participants:	Daniel Wagner, Michael Greenberg
Status:	Not yet released

Anybody who has used LATEX knows that it is a fantastic tool for typesetting; but its error reporting leaves much to be desired. Even simple documents that use a handful of packages can produce hundreds of lines of uninteresting output on a successful run. Picking out the parts that require action is a serious chore, and locating the right part of the document source to change can be tiresome when there are many files.

Pulp is a parser for LATEX log files with a small but expressive configuration language for identifying which messages are of interest. A typical run of pulp after successfully building a document produces no output; this makes it very easy to spot when something has gone wrong. Next time you want to produce a great paper, process your log with pulp!

Features

◦ LATEX log parser with special-case support for many popular packages and classes
◦ Expressive configuration language
  – Filter out document-specific unimportance
  – Increase verbosity as the document nears completion
◦ Uniform error reporting format with file and line information
◦ Instructions for use with latexmk
◦ Rudimentary Windows support

Further reading

http://github.com/dmwit/pulp

7.9.3 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, are 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.

Further reading

◦ http://lelf.lu/prose
◦ https://github.com/llelf/prose
◦ https://github.com/llelf/numerals

7.10 Natural Language Processing

7.10.1 NLP

Report by: Eric Kow

The Haskell Natural Language Processing community aims to make Haskell a more useful and more popular language for NLP. The community provides a mailing list, Wiki and hosting for source code repositories via the Haskell community server.

The Haskell NLP community was founded in March 2009. The list is still growing slowly as people grow increasingly interested in both natural language processing, and in Haskell.

At the present, the mailing list is mainly used to make announcements to the Haskell NLP community. We hope that we will continue to expand the list and expand our ways of making it useful to people potentially using Haskell in the NLP world.

New packages

- *Earley-0.8.0* (Olle Fredriksson)
  This (Text.Earley) is a library consisting of two parts:
  1. **Text.Earley.Grammar**: An embedded context-free grammar (CFG) domain-specific language (DSL) with semantic action specification in applicative style.

     An example of a typical expression grammar working on an input tokenized into strings is the following:

```
expr :: Grammar r String (Prod r String String Expr)
expr = mdo
  x1 ← rule $ Add <$> x1 <* namedSymbol "+" <*> x2
      <|> x2
      <?> "sum"
  x2 ← rule $ Mul <$> x2 <* namedSymbol "*" <*> x3
      <|> x3
      <?> "product"
  x3 ← rule $ Var <$> (satisfy ident <?> "identifier")
      <|> namedSymbol "("  *> x1 <* namedSymbol ")"
  return x1
  where
    ident (x: _) = isAlpha x
    ident _ = False
```

  2. **Text.Earley.Parser**: An implementation of (a modification of) the Earley parsing algorithm.

     To invoke the parser on the above grammar, run e.g. (here using words as a stupid tokeniser):

```
fullParses $ parser expr $ words "a + b * ( c + d )"
 = ([Add (Var "a") (Mul (Var "b")
(Add (Var "c") (Var "d")))]
   , Report {...}
   )
```

     Note that we get a list of all the possible parses (though in this case there is only one).

https://github.com/ollef/Earley

Further reading

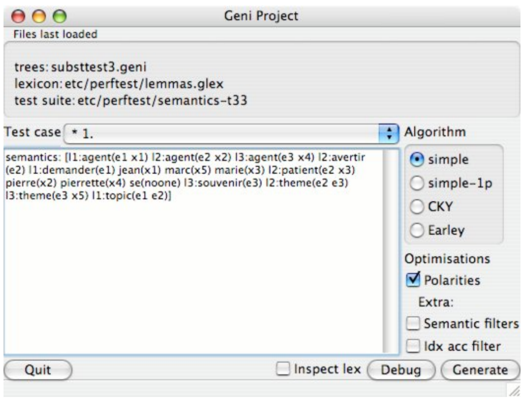
- The Haskell NLP page http://projects.haskell.org/nlp

7.10.2 GenI

Report by: Eric Kow

GenI is a surface realizer for Tree Adjoining Grammars. Surface realization can be seen a subtask of natural language generation (producing natural language utterances, e.g., English texts, out of abstract inputs). GenI in particular takes a Feature Based Lexicalized Tree Adjoining Grammar and an input semantics (a conjunction of first order terms), and produces the set of sentences associated with the input semantics by the grammar. It features a surface realization library, several optimizations, batch generation mode, and a graphical debugger written in wxHaskell. It was developed within the TALARIS project and is free software licensed under the GNU GPL, with dual-licensing available for commercial purposes.

GenI is now mirrored on GitHub, with its issue tracker and wiki and homepage also hosted there. The most recent release, GenI 0.24 (2013-09-18), allows for custom semantic inputs, making it simpler to use GenI in a wider variety for applications. This has recently been joined by a companion geni-util package which offers a rudimentary geniserver client and a reporting tool for grammar debugging.



Files last loaded

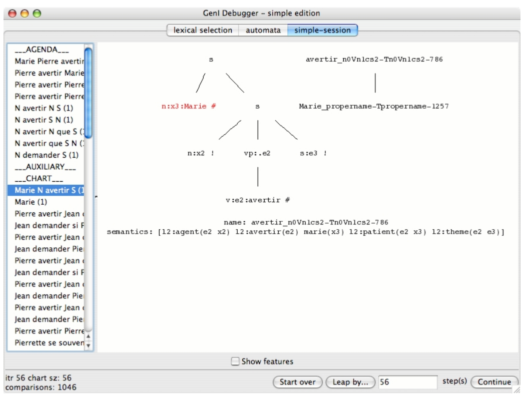
trees: substtest3.geni
lexicon: etc/perftest/lemmas.glex
test suite: etc/perftest/semantics-t33

Test case: * 1.

semantics: [l1:agent(e1 x1) l2:agent(e2 x2) l3:agent(e3 x4) l2:avertir(e2) l1:demander(e1) jean(x1) marc(x5) marie(x3) l2:patient(e2 x3) pierre(x2) pierrette(x4) se(noone) l3:souvenir(e3) l2:theme(e2 e3) l3:theme(e3 x5) l1:topic(e1 e2)]

Algorithm: [✓ simple] simple-1p CKY Earley

Optimisations: [✓ Polarities]
Extra: Semantic filters, Idx acc filter, Inspect lex



___AGENDA___
Marie Pierre avertir
Pierre avertir Marie
Pierre avertir Pierre
Pierre avertir Pierre
N avertir N S (1)
N avertir S N (1)
N avertir N que S (
N avertir que S N (
N demander S (1)
___AUXILIARY___
___CHART___
Marie N avertir S (1
Marie (1)
Pierre avertir Jean
Jean demander si P
Pierre avertir Jean
Jean demander Pie
Pierre avertir Jean
Jean demander si P
Pierre avertir Jean
Jean demander Pie
Pierre avertir Jean
Jean demander Pie
Pierre avertir Pierre
Pierrette se souven

name: avertir_n0Vn1cs2-Tn0Vn1cs2-786
semantics: [l2:agent(e2 x2) l2:avertir(e2) marie(x3) l2:patient(e2 x3) l2:theme(e2 e3)]

itr 56 chart sz: 56
comparisons: 1046
56 step(s)

GenI is available on Hackage, and can be installed via cabal-install, along with its GUI and HTTP server user interfaces. For more information, please contact us on the geni-users mailing list.

Further reading

- http://github.com/kowey/GenI
- http://projects.haskell.org/GenI
- Paper from Haskell Workshop 2006:
  http://hal.inria.fr/inria-00088787/en
- http://websympa.loria.fr/wwsympa/info/geni-users

7.11 Bioinformatics

7.11.1 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. Following ideas established in algebraic dynamic programming (ADP) 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, and sets (with boundaries, if necessary). Both linear and 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.

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 $p \to c\ p\ c \mid \epsilon$.

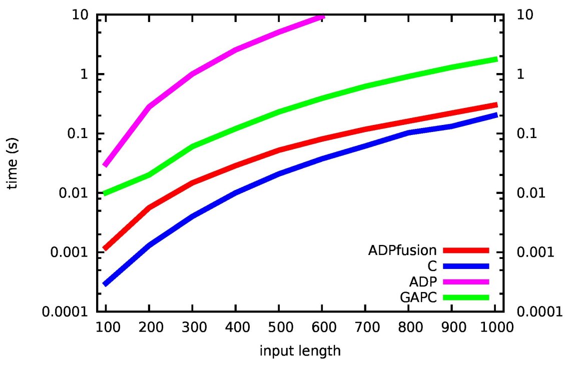
The corresponding ADPfusion code is similar:

```
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 machinary 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 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

7.11.2 Biohaskell

Report by:	Ketil Malde
Participants:	Christian Höner zu Siederdissen, Michal J. Gajda, Nick Ignolia, Felipe Almeida Lessa, Dan Fornika, Maik Riechert, Ashish Agarwal, Grant Rotskoff, Florian Eggenhofer, Sarah Berkemer, Niklas Hambüchen

Bioinformatics in Haskell is a steadily growing field, and the *Bio* section on Hackage now contains 69 libraries and applications. The biohaskell web site coordinates this effort, and provides documentation and related information. Anybody interested in the combination of Haskell and bioinformatics is encouraged to sign up to the mailing list (currently by emailing ⟨ketil@malde.org⟩Ketil), and to register and document their contributions on the http://biohaskell.org wiki.

In the summer of 2014, Sarah Berkemer was financed by Google's Summer of Code program to work on optimizing transalign. After a summer's work, Sarah was able to improve both space and time usage. Other new additions are parsers by Floran Eggenhofer for the NCBI Genbank format and for Clustal mulitiple sequence alignments. There is also a new library for working with EEG devices, written by Niklas Hambüchen and Patrick Chilton.

Further reading

- http://biohaskell.org
- http://blog.malde.org
- http://www.bioinf.uni-leipzig.de/~choener/haskell/
- https://bioinf.eva.mpg.de/biohazard/

7.11.3 arte-ephys: Real-time electrophysiology

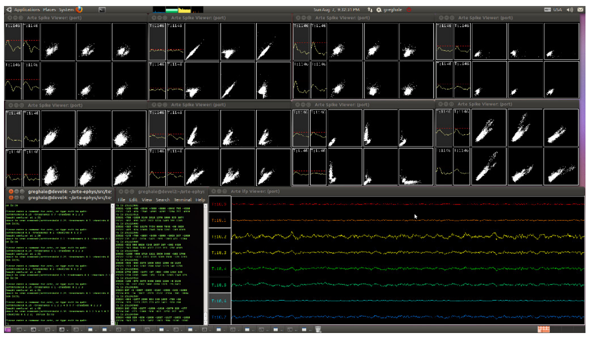
Report by:	Greg Hale
Participants:	Alex Chen
Status:	work in progress

Arte-ephys is a soft real-time neural recording system for experimental systems neuroscientists.

Our lab uses electrode arrays for brain recording in freely moving animals, to determine how these neurons build, remember, and use spatial maps.

We previously recorded and analyzed our data in two separate stages. We are now building a recording system focused on streaming instead of offline analysis, for real-time feedback experiments. For example, we found that activity in the brain of resting rats often wanders back to representations of specific parts of a recently-learned maze, and we would now like to automatically detect these events and reward the rat immediately for expressing them, to see if this influences either the speed of learning of a specific part of the maze or the nature of later spatial information coding.

We now have a proof-of-concept that streams recorded data from disk, performs the necessary preprocessing, and accurately decodes neural signals in realtime, while drawing the results with gloss. Our next goal is to integrate this into a sytem that streams raw neural data during the experiment.



Further reading

- http://github.com/ImAlsoGreg/arte-ephys
- http://github.com/ImAlsoGreg/haskell-tetrode-ephys
- http://web.mit.edu/wilsonlab/html/research.html

7.12 Embedding DSLs for Low-Level Processing

7.12.1 CλaSH

Report by:	Christiaan Baaij
Participants:	Jan Kuper, Arjan Boeijink, Rinse Wester
Status:	actively developed

The first line of the package description on hackage is:

> CλaSH (pronounced 'clash') is a functional hardware description language that borrows its syntax and semantics from the functional programming language Haskell.

In essence, however, it is a combination of:

- A Haskell library containing data types and functions for circuit design: http://hackage.haskell.org/package/clash-prelude.
- A compiler that transforms the Haskell code to low-level synthesisable VHDL or SystemVerilog: http://hackage.haskell.org/package/clash-ghc.

Of course, the compiler cannot transform arbitrary Haskell code to hardware, but only the *structural* subset of Haskell. This subset is vaguely described as the *semantic* subset of Haskell from which a *finite* structure can be inferred, and hence excludes unbounded recursion. The CλaSH compiler is thus a proper compiler (based on static analysis), and *not* an embedded Domain Specific Language (DSL) such as Kansas Lava (→ 7.12.3).

CλaSH has been in active development since 2010. Since then we have significantly improved stability, enlarged the subset of transformable Haskell, improved performance of the compiler, and added (System)Verilog generation. And, perhaps most importantly, vastly improved documentation.

CλaSH is available on Hackage, for GHC version 7.10 and higher:

```
$ cabal update
$ cabal install clash-ghc
```

What's new since last HCAR:

- CλaSH can now generate, next to VHDL-93 and SystemVerilog-2005, Verilog-2001.
- Support for memory primitives whose content can be initialised from a file.

◦ Major overhaul and extension of the `Vector` module. All functions in `Vector` are now synthesisable to VHDL/(System)Verilog.

Development plans for CλaSH are:

◦ Behavioural synthesis of unbounded recursion (by Ingmar te Raa).
◦ Use a dependently typed internal core language, so that we can use both Haskell/GHC and Idris http://http://www.idris-lang.org/ as *front-end* language for circuit design (by Christiaan Baaij).

Further reading

http://www.clash-lang.org

7.12.2 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 is currently being developed by projects at Chalmers University of Technology (→ 9.6), 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. The aim is to allow functions to be written in pure functional style in order to raise the abstraction level of the code and to enable more high-level optimizations. The current version consists of a library of numeric and array processing operations as well as a code generator producing C code for running on embedded targets.

The official packages feldspar-language and feldspar-compiler contain the language for pure computations and its C back end, respectively.

Additionally, feldspar-io (not yet released, but fully usable) adds an "IO-like" monad for making interactive Feldspar programs and binding to external C libraries. Ongoing work involves using `feldspar-io` to implement more high-level libraries for streaming and interactive programs. Two examples of such libraries are:

◦ feldspar-synch – a synchronous data-flow library
◦ zeldspar – a Ziria-like EDSL

Further reading

◦ Official home page: http://feldspar.github.io
◦ Recent paper (TFP 2015) about controlling the signatures of generated C functions

7.12.3 Kansas Lava

Report by:	Andrew Gill
Participants:	Bowe Neuenschwander
Status:	ongoing

Kansas Lava is a Domain Specific Language (DSL) for expressing hardware descriptions of computations, and is hosted inside the language Haskell. Kansas Lava programs are descriptions of specific hardware entities, the connections between them, and other computational abstractions that can compile down to these entities. Large circuits have been successfully expressed using Kansas Lava, and Haskell's powerful abstraction mechanisms, as well as generic generative techniques, can be applied to good effect to provide descriptions of highly efficient circuits.

◦ The Fabric monad is now a Monad transformer. The Fabric monad historically provided access to named input/output ports, and now also provides named variables, implemented by ports that loop back on themselves. This additional primitive capability allows for a *typed* state machine monad. This design gives an elegant stratospheric pattern: purely functional circuits using streams; a monad for layout over *space*; and a monad for state generation, that acts over *time*.
◦ On top of the Fabric monad, we are implementing an atomic transaction layer, which provides a BSV-like interface, but in Haskell. An initial implementation has been completed, and this is being reworked to include BSV's Ephemeral History Registers.

Further reading

http://www.ittc.ku.edu/csdl/fpg/Tools/KansasLava

7.13 Games

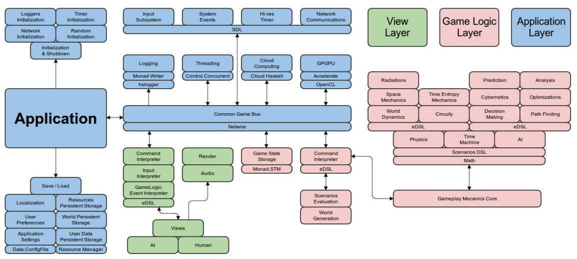
7.13.1 The *Amoeba-World* game project

Report by:	Alexander Granin
Status:	work in progress

In functional programming, there is a serious problem: there are no materials for the development of large applications. As we know, this field is well studied for imperative and object-oriented languages. There are books on design, architecture, design patterns and modeling practices. But we have no idea how this big knowledge can be adapted to functional languages.

I'm working on a game called "The Amoeba World". The goal of this project is to explore approaches to the development of large applications on Haskell. The results of my research are some articles which will be used to compose a book about functional design and architecture. Currently two articles are written out of the planned four (in Russian, but the articles will be translated to English soon). The first highlights the issue of whether the mainstream knowledge of architecture is applicable to the functional paradigm and what tools can be used for designing of architecture. It shows that the UML is ill-suited for the functional paradigm and the architecture is constructed using mind maps and concept cards. The second article talks about a low-level design of the application using the language Haskell. It has a theoretical part named *what makes a good design*, but there is also practical part describing of the some anti-patterns in Haskell. The third article is under development now. In it, the application design based on properties and scenarios is researched. The fourth article will be discussing the use of FRP.

Code of the game “The Amoeba World” should be written well to be a good example of the design concepts. These concepts are: using DSL, parsing, layering, using lenses, Inversion of Control, testing, FRP, SDL, usefulness of monads. The overall architecture of the game looks as follows:



At the moment, the game logic has been rewritten twice. The draft of game logic is ready. A special file format ’ARF’ (Amoeba Raw File) for the game objects is done. Parsec is used for parsing, and a custom safe translator is written, which works on rules. Now I’m are working on a Application Layer. Settings loading is done. A primitive renderer for the game world is created. A draft game cycle and IO event handler from SDL subsystem is done by using Netwire FRP library. The next objectives are to add an interaction within the game world and then move to the execution of scenarios on game objects.

Further reading

- https://github.com/graninas/The-Amoeba-World
- http://bit.ly/ArchitectureAndDesingInFP (in Russian)

7.13.2 EtaMOO

Report by:	Rob Leslie
Status:	experimental, active development

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

Unlike LambdaMOO which is a single-threaded server, EtaMOO seeks to offer a fully multi-threaded environment, including concurrent execution of MOO tasks. To retain backward compatibility with the general MOO code expectation of single-threaded semantics, EtaMOO makes extensive use of software transactional memory (STM) to resolve possible conflicts among simultaneously running MOO tasks.

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

While still under development, the current implementation supports loading a LambdaMOO-format database from a file, receiving client (telnet) connections from the network, and executing MOO code as a result of processing the commands received from each connection. Soon to be implemented will be the ability to save the changes made to the MOO object database back to a file, at which point the server should be largely usable.

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

7.13.3 scroll

Report by:	Joey Hess
Status:	stable, complete

Scroll is a roguelike game, developed in one week as an entry in the 2015 Seven Day Roguelike Challenge.

In scroll, you’re a bookworm that’s stuck on a scroll. You have to dodge between words and use spells to make your way down the page as the scroll is read. Go too slow and you’ll get wound up in the scroll and crushed.

This was my first experience with using Haskell for game development, and I found it quite an interesting experience, and a great crutch in such an intense coding sprint. Strong typing and purely functional code saved me from many late night mistakes, until I eventually became so exhausted that String → String seemed like a good idea. Even infinite lists found a use; one of scroll’s levels features a reversed infinite stream of consciousness based on Joyce’s Ulysses. . .

Scroll was written in continuation passing style, and this turned out to be especially useful in developing its magic system, with spells that did things ranging from creating other spells, to using a quick continuation based threading system to handle background tasks, to letting the player enter the altered reality of a dream, from which they could wake up later.

I had a great time creating a game in such a short time with Haskell, and documenting my progress in 7 blog posts, and it's been well received by players.

Further reading

http://joeyh.name/code/scroll/

7.13.4 Nomyx

Report by:	Corentin Dupont
Status:	stable, actively developed

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

At the beginning, the initial rules are describing:

◦ How to add new rules and change existing ones. For example a unanimity vote is necessary to have a new rule accepted.
◦ How to win the game. For example you win the game if you have 5 rules accepted.

But of course even that can be changed!

A first version has been released. A match is currently on-going, join us! A lot of learning material is available, including a video, a tutorial, a FAQ, a forum and API documentation.

If you like Nomyx, you can help! There is a development mailing list (check the website). The plan now is to create a new version were knowing haskell is not necessary to play.

Further reading

http://www.nomyx.net

7.13.5 Barbarossa

Report by:	Nicu Ionita
Status:	actively developed

Barbarossa is a UCI chess engine written completely in Haskell. UCI is one of 2 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 v0.3.0 from begin of October.

Further reading

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

7.14 Others

7.14.1 leapseconds-announced

Report by:	Björn Buckwalter
Status:	stable, maintained

The leapseconds-announced library provides an easy to use static LeapSecondTable 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 2015 of leapseconds-announced contains all leap seconds up to 2015-07-01. A new version will be uploaded if/when the IERS announces a new leap second.

Further reading

https://hackage.haskell.org/package/leapseconds-announced

7.14.2 hledger

Report by:	Simon Michael
Status:	ongoing development; suitable for daily use

hledger is a cross-platform program (and Haskell library) for tracking money, time, or any other commodity, using double-entry accounting and a simple, editable text file format. hledger aims to be a reliable, 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+.

hledger's HCAR entry was last updated in the November 2011 report, but development has continued steadily, with 2-3 major releases each year.

Many new features and improvements have been introduced, making hledger much more useful. These include:

◦ Easier installation, using stack, system packages, or downloadable Windows binaries.
◦ A simpler and more robust web interface, with built-in help, balance charts, flexible transaction entry, and automatic browser startup
◦ A new curses-style interface, hledger-ui, is now included and fully supported
◦ The command-line interface is more robust, and is aware of terminal width, COLUMNS, and wide characters
◦ New commands: accounts, balancesheet, cashflow, incomestatement
◦ New add-on packages: ledger-autosync, hledger-diff, hledger-interest, and hledger-irr
◦ hledger can now report current value based on market prices (-V)
◦ The journal format has become richer, supporting more Ledger features such as balance assertions
◦ hledger journals and reports can be exported as CSV
◦ hledger now reads CSV files directly, using flexible conversion rules
◦ The balance command can show multiple columns, with per-period changes or ending balances
◦ Depth-limiting now interacts well with other features, making it effective for summarising
◦ hledger-web's query language is richer and is also used by the command-line interface
◦ The Decimal library is used for representing amounts exactly
◦ Unicode is handled correctly
◦ Many commands are faster

Project updates include:

◦ hledger.org and the docs have been refreshed a few times, and now include many examples
◦ hledger's code repo and bug tracker have moved from darcs/darcs hub/google code to git/github
◦ hledger has its own IRC channel on freenode: #hledger, with logging and commit/issue/travis notifications

hledger is available from hledger.org, github, hackage, stackage, and is packaged for a number of systems including Debian, Ubuntu, Gentoo, Fedora, and NixOS. See http://hledger.org/download or http://hledger.org/developer-guide for guidance.

Immediate plans:

◦ 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.

Further reading

http://hledger.org

7.14.3 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!

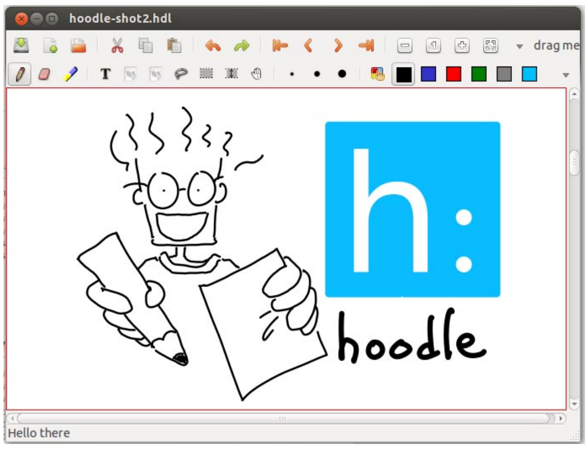
Further reading

◦ 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/

7.14.4 Hoodle

Report by:	Ian-Woo Kim
Status:	Actively Developing

Hoodle is a pen-notetaking programing written in haskell using Gtk2hs. The name Hoodle is from Haskell + doodle.



This project first started as making a haskell clone of Xournal, a notetaking program developed in C. But now Hoodle has more unique features, as well as basic pen notetaking function. Pen input is directly fed into from X11 events, which has sub-pixel level accuracy for the case of wacom tablets. Therefore, the resultant pen strokes are much smoother than other similar open-source programs such as Jarnal and Gournal.

Hoodle can be used for annotation on PDF files, and also supports importing images of PNG, JPG and SVG types, and exporting Hoodle documents to PDF. One of the most interesting features is "linking": each Hoodle document can be linked with each other by simple drag-and-drop operations. Then, the user can navigate linked Hoodle documents as we do in web browser. Another interesting feature is that one can edit a document in split views, so that a long Hoodle document can be easily edited. Hoodle can embed LaTeXtexts and the embedded text can be edited via network.

GUI programming is in general tightly tied into a GUI framework. Since most frameworks rely on callbacks for event processing, program logic is likely to be scattered in many callback functions. We cure this situation by using coroutines. In haskell, coroutine can be implemented in a straightforward way without relying on specific language feature. This abstraction enable us to reason through the program logic itself, not through an inverted logic in a GUI framework.

Hoodle is being very actively developed as an open-source project hosted on Github. The released versions are located on Hackage, and it can be installed by simple cabal install. On Linux, OS X, and Windows systems with Gtk2hs and Poppler, Hoodle can be installed without problems. Recently, it is packaged for NixOS. Making a Hoodle binary package for other linux distributions, OS X and window is planned.

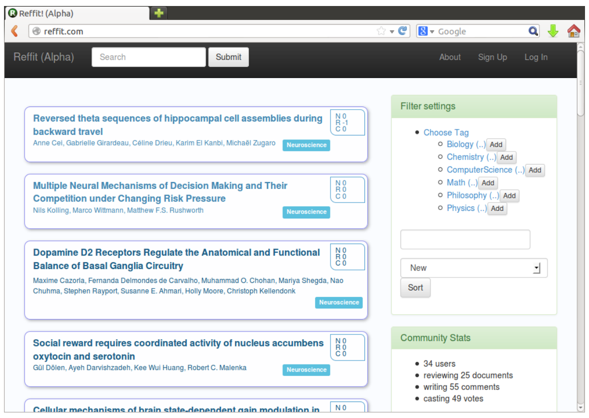
The development focus as of now is to have more flexible link features (link to arbitrary position of a document) and an internal database for document management. Hoodle manages documents with a unique UUID, but it does not have a good internal database yet. This feature can also be extended to saving Hoodle documents in cloud storage in a consistent way. Refining rendering with appropriate GPU acceleration is also planned. In the long run, we plan to support mobile platforms.

Further reading

http://ianwookim.org/hoodle

7.14.5 Reffit

Report by:	Greg Hale
Status:	work in progress



Reversed theta sequences of hippocampal cell assemblies during backward travel
Anne Cei, Gabrielle Girardeau, Céline Drieu, Karim El Kanbi, Michaël Zugaro
Neuroscience
N 0 R -1 C 0

Multiple Neural Mechanisms of Decision Making and Their Competition under Changing Risk Pressure
Nils Kolling, Marco Wittmann, Matthew F.S. Rushworth
Neuroscience
N 0 R 0 C 0

Dopamine D2 Receptors Regulate the Anatomical and Functional Balance of Basal Ganglia Circuitry
Maxime Cazorla, Fernanda Delmondes de Carvalho, Muhammad O. Chohan, Mariya Shegda, Nao Chuhma, Stephen Rayport, Susanne E. Ahmari, Holly Moore, Christoph Kellendonk
Neuroscience
N 0 R 0 C 0

Social reward requires coordinated activity of nucleus accumbens oxytocin and serotonin
Gül Dölen, Ayeh Darvishzadeh, Kee Wui Huang, Robert C. Malenka
Neuroscience
N 0 R 0 C 0

Cellular mechanisms of brain state-dependent gain modulation in
N 0

- Choose Tag
  - Biology (..) Add
  - Chemistry (..) Add
  - ComputerScience (..) Add
  - Math (..) Add
  - Philosophy (..) Add
  - Physics (..) Add

- 34 users
- reviewing 25 documents
- writing 55 comments
- casting 49 votes

Reffit is a Snap website for collecting and organizing short comments on peer reviewed papers, blog posts, and videotaped talks. We hope to attract a community and foster a culture of open discussion of papers, with a lighthearted attitude, informality, and gamification.

Further reading

- http://reffit.com
- http://github.com/ImAlsoGreg/reffit

7.14.6 Laborantin

Report by:	Lucas DiCioccio
Status:	Working, development for new features

Conducting scientific experiments is hard. Laborantin is a DSL to run and analyze scientific experiments. Laborantin is well-suited for experiments that you can run offline such as benchmarks with many parameters.

Laborantin encourages users to express experiments parameters, experiment results, as well as execution, startup, and teardown procedures in a methodical manner. For instance, the following snippet defines a network 'ping' experiment with a destination and packet-size parameters.

```
ping = scenario "ping" $ do
  describe "ping to a remote server"
  parameter "destination" $ do
    describe "a destination server (host or ip)"
    values [str "example.com", str "dicioccio.fr"]
  parameter "packet-size" $ do
    describe "packet size in bytes"
    values [num 50, num 1500]
  run $ do
    (StringParam srv) <- param "destination"
    (NumberParam ps) <- param "packet-size"
    liftIO (execPing srv ps) >>= writeResult "ping.out"

execPing :: Text -> Rational -> IO (Text)
execPing host pktSz =
  let args = [ "-c", "10"
             , "-s" , show (round pktSz) , T.unpack host]
  in fmap T.pack (readProcess "ping" args "")
```

Laborantin also lets users express dependencies between experiments. Laborantin is designed to allow multiple backend (where to run and store experiments) and multiple frontends (how a user interacts with Laborantin). The current backend stores experiment results on the filesystem and provides a command line frontend.

Contributions are welcome. In the future, we plan to enrich Laborantin with helper modules for common tasks such as starting and collecting outputs of remote processes, reformatting results, and generating plots (e.g., with Diagrams). Laborantin would also benefit from new backends (e.g., to store results in an SQL database or HDFS) and new frontends (e.g., an integration in IHaskell).

Further reading

- Hackage page: http://hackage.haskell.org/package/laborantin-hs
- Example of web-benchmarks: https://github.com/lucasdicioccio/laborantin-bench-web

7.14.7 Transient

Report by:	Alberto Gómez Corona
Status:	active development

Transient is a monad with batteries included that bringing the power of high level effects in order to reduce the learning curve and make haskell programmer productive. Effects include event handling/reactive, backtracking, extensible state, indetermism, concurrency, parallelism, thread control and distributed computing. It is possible to create combinators that permit newcomers to program at a higher level, that was not previously possible.

The impedance mismatch between specifications and programming comes from the fact that the technical requirements manage similar concepts than programming, but at a higher level: For example: this specification description: "the query processor will send request for each data source and filter the results according with the provided function"

A specification like this is described as a sequence of steps, as if the functionality were a single process, but really it may involve many threads, synchronizations, event handling, working with network nodes, possibly undoing actions under some conditions, stopping under some other condition etc.

Transient permits to write a monadic sequence that express this requirement with a one-to-one correspondence, since these effects are included and are managed automatically by a single monad. In particular a monadic or applicative expression in Transient may receive events in the middle of the sequence and may dispatch threads for these events and yet externally it is a single expression. Now It can also perform cloud computing in the same way.

Transient uses a different way to produce effects: A transient statement can access his own expression an his own continuation. Therefore it can re-execute them when some asynchronous input is received. Or this continuation can be stored in the state, so that other statements can make use of them later, so combinations of statements can edit the execution flow to produce effects.

Althoug it may be functionally equivalent to delimited continuations, in transient is more easy to program discrete effects and create powerful primitives that can be composed to create programs that are easier to understand without resorting to the ubiquous use of the all-powerful unrestricted ”callCC” For example, with some primitives like ”async” the Applicative operators can be used for concurrency and the alternative operator can be used for parallelism. the same operators can be used under distributed computing when combined with ”runAt”

What is new in Transient is the addition of some primitives for publish-suscribe events[4], parallel non-determinism[6], logging, distributed computing[5], streaming, distributed streaming and beginning a mapReduce functionality with datasets in the stlyle of Apache spark[7]

Future work:

Transient will be the base of a haskell embedded general purpose language that implement a cloud computing architecture where MFlow and hplayground will be the user interface.

Further reading

- Transient GIT repository https://github.com/agocorona/transient
- An EDSL for Hard-working IT programmers https://www.fpcomplete.com/user/agocorona/EDSL-for-hard-working-IT-programmers
- The hardworking programmer II: practical backtracking to undo actions https://www.fpcomplete.com/user/agocorona/the-hardworking-programmer-ii-practical-backtracking-to-undo-actions
- Publish-suscribe variables https://www.fpcomplete.com/user/agocorona/publish-subscribe-variables-transient-effects-v
- Moving processes between nodes https://www.fpcomplete.com/user/agocorona/moving-haskell-processes-between-nodes-transient-effects-iv
- Parallel non-determinism https://www.fpcomplete.com/user/agocorona/beautiful-parallel-non-determinism-transient-effects-iii
- streamimg, distributed streaming, mapReduce with distributed datasets https://www.fpcomplete.com/user/agocorona/estimation-of-using-distributed-computing-streaming-transient-effects-vi-1

7.14.8 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. The are programmed via binary files in a proprietary, undocumented data format.

We have reverse engineered the format, and created a tool to analze these files and generate your 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 idoms 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)

7.14.9 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.

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

7.14.10 Octohat (Stack Builders)

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

7.14.11 git-annex

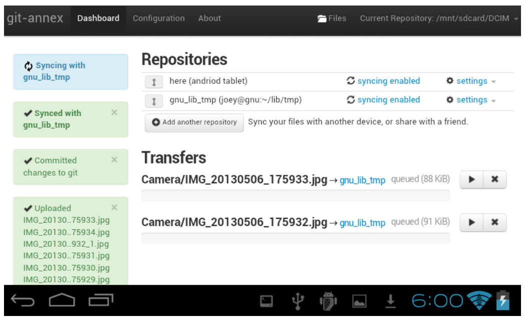
Report by:	Joey Hess
Status:	stable, actively developed

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

gnu_lib_tmp (joey@gnu:~/lib/tmp) syncing enabled

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/

7.14.12 openssh-github-keys (Stack Builders)

Report by:	Stack Builders
Participants:	Justin Leitgeb
Status:	A library to automatically manage SSH access to servers using GitHub teams

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 recently 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

7.14.13 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 NoInfo
```

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 http://propellor.branchable.com/posts/

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.

Another exciting possibility is using Propellor to not only configure existing Linux systems, but to manage their entire installation process. This has already been prototyped in a surprisingly small amount of added code (under 200 lines), which can replace arbitrary Linux systems with clean re-installs described entirely by Propellor's config.hs.

Further reading

http://propellor.branchable.com/

7.14.14 dimensional: Statically Checked Physical Dimensions

Report by:	Douglas McClean
Participants:	Björn Buckwalter, Alberto Valverde GonzÃąlez
Status:	active

Dimensional is a library providing data types for performing arithmetic with physical quantities and units. Information about the physical dimensions of the quantities/units is embedded in their types, and the validity of operations is verified by the type checker at compile time. The boxing and unboxing of numerical values as quantities is done by multiplication and division with units. The library is designed to, as far as is practical, enforce/encourage best practices of unit usage within the frame of the SI. Example:

$d :: \mathsf{Fractional}\ a \Rightarrow \mathsf{Time}\ a \rightarrow \mathsf{Length}\ a$
$d\ t = a\ /\ \_2 * t\ \hat{}\ pos2$
$\quad \textbf{where}\ a = 9.82 *\tilde{}\ (meter\ /\ second\ \hat{}\ pos2)$

We are pleased to announce the release of dimensional 1.0, based on the prototype dimensional-dk implementation. Using data kinds and closed type families, the new version includes improved Haddock documentation, unit names with many options for pretty-printing, exact conversion factors between units (even between degrees and radians!), types for manipulating units and quantities whose dimensions are not known statically, and support for unboxed vectors.

New users with access to GHC 7.8 or later are strongly encouraged to use dimensional 1.0.

The “classic” dimensional library as released in 2006 is based on multi-parameter type classes and functional dependencies. It is stable with units being added on an as-needed basis. The primary documentation is the literate Haskell source code. Any future maintenance releases will have version numbers < 1.0.

The dimensional-tf library released in January 2012 a port of dimensional using type families will continue to be supported but is not recommended for new development.

Further reading

https://github.com/bjornbm/dimensional-dk

7.14.15 igrf: The International Geomagnetic Reference Field

Report by:	Douglas McClean
Status:	active

The igrf library provides a Haskell implementation of the International Geomagnetic Reference Field, including the latest released model values.

Upcoming development efforts include a parser for the model files as released by the IAGA and a dimensionally-typed interface using the dimensional library.

Further reading

https://github.com/dmcclean/igrf

7.14.16 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

8 Commercial Users

8.1 Well-Typed LLP

Report by:	Andres Löh
Participants:	Duncan Coutts, Adam Gundry

Well-Typed is a Haskell services company. We provide commercial support for Haskell as a development platform, including consulting services, training, and bespoke software development. For more information, please take a look at our website or drop us an e-mail at ⟨info@well-typed.com⟩.

We have been working on a large number of different projects for various clients, most of which are unfortunately not publically visible.

Ben Gamari and Austin Seipp have been helping with GHC (→ 3.1).Adam Gundry is still trying to make overloaded record fields a reality.

On behalf of the Industrial Haskell Group (IHG), Duncan Coutts and Edsko de Vries have been working on Hackage security, which is currently in beta (see link below).

Duncan has also been working on an improved implementation of the `binary` package, called `binary-serialise-cbor`, which is now in production at one of our clients where it has dramatically improved performance and reduced memory use. It is still experimental, however, and currently available only on Github (link below).

Edsko de Vries and Andres Löh have been developing and improving `generics-sop`, a generic programming library based on n-ary sums of products (→ 7.5.4).

Andres has also helped with the development of the Haskell Servant web framework, culminating in a better approach to routing, to be included in the next release, and a paper at this year's Workshop on Generic Programming (link below).

We have been organizing various Haskell courses, mostly with the help of Skills Matter, in London and New York. We also helpd to organize the fourth Haskell eXchange in London in October, the first time spanning two days and two tracks, and directly followed by an infrastructure-themed Haskell Hackathon. We're happy that we had so many great speakers and participants. The videos are available online (link below). Registration for the Haskell eXchange 2016 is already open. If you're interested in speaking, please contact us (or Andres, specifically).

Our course dates for 2016 will be published within the next few weeks. We are always open to suggestions for extra locations and dates and on-demand courses.

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 just drop us an e-mail.

Further reading

- Company page: http://www.well-typed.com
- Blog: http://blog.well-typed.com/
- Hackage security beta announcement: http://www.well-typed.com/blog/2015/08/hackage-security-beta/
- `binary-serialise-cbor` package: https://github.com/well-typed/binary-serialise-cbor/
- `generics-sop` package: https://hackage.haskell.org/package/generics-sop/
- Paper and links on Servant: http://www.andres-loeh.de/Servant/
- Haskell eXchange 2015 (including videos): https://skillsmatter.com/conferences/7069-haskell-exchange-2015
- Haskell eXchange 2016 (registration): https://skillsmatter.com/conferences/7276-haskell-exchange-2016
- Training page: http://www.well-typed.com/services_training
- Skills Matter Haskell course overview: https://skillsmatter.com/explore?content=courses&location=&q=Haskell

8.2 Bluespec Tools for Design of Complex Chips and Hardware Accelerators

Report by:	Rishiyur Nikhil
Status:	Commercial product; free for academia

Bluespec, Inc. provides an industrial-strength language (BSV) and tools for high-level hardware design. Components designed with these are shipping in some commercial smartphones and tablets today.

BSV is used for all aspects of ASIC and FPGA design — specification, synthesis, modeling, and verification. Digital circuits are *described* using a notation with Haskell semantics, including algebraic types, polymorphism, type classes, higher-order functions and monadic elaboration. Strong static checking is also used to support discipline for multiple clock-domains and gated clocks. The dynamic semantics of a such circuits are described using Term Rewriting Systems (which are essentially atomic state transitions). BSV is applicable to all kinds of hardware systems, from algorithmic "datapath" blocks to complex control blocks such as processors, DMAs, interconnects, and caches, and to complete SoCs (Systems on a Chip).

Perhaps uniquely among hardware-design languages, BSV's rewrite rules enable design-by-refinement, where an initial executable approximate design is systematically transformed into a quality implementation by successively adding functionality and architectural detail.

Before synthesizing to hardare, a circuit description can be executed and debugged in *Bluesim*, a fast simulation tool. Then, the *bsc* tool compiles BSV into high-quality Verilog, which is then further synthesized into netlists for ASICs and FPGAs using standard synthesis tools. There are extensive libraries and infrastructure components to make it easy to build FPGA-based accelerators for compute-intensive software.

Bluespec also provides implementations and development environments for CPUs based on the U.C. Berkeley RISC-V instruction set (www.riscv.org).

Status and availability

BSV tools have been available since 2004, both commercially and free for academic teaching and research. It is used in a several leading universities (incl. MIT, U.Cambridge, and IIT Chennai) for computer architecture research.

Further reading

- *Types, Functional Programming and Atomic Transactions in Hardware Design*, R.S. Nikhil, in *In Search of Elegance in the Theory and Practice of Computation, Essays dedicated to Peter Buneman (Festschrift), Springer-Verlag Lecture Notes in Computer Science, LNCS 8000*, pp.418-431, 2013.
- *Abstraction in Hardware System Design*, R.S. Nikhil, in *Communications of the ACM*, 54:10, October 2011, pp. 36-44.
- *BSV by Example*, R.S. Nikhil and K. Czeck, 2010, book available on Amazon.com (or free PDF from Bluespec, Inc.)
- http://bluespec.com/SmallExamples/index.html: from *BSV by Example*.
- http://www.cl.cam.ac.uk/~swm11/examples/bluespec/: Simon Moore's BSV examples (U. Cambridge).
- http://csg.csail.mit.edu/6.375: *Complex Digital Systems*, MIT courseware.

8.3 Haskell in the industry in Munich

Report by: Haskell Consultancy Munich

Haskell is used by several companies specializing in the development of reliable software and hardware, for example for the automotive industry in Munich. It is also in use by the developers of medical software which needs assure the integrity of data processing algorithms. It is also used by new media and internet companies. You may contact the author of this report (⟨haskell.consultancy@gmail.com⟩) for details.

Haskell at Google Munich

Google is using Haskell in Ganeti (http://code.google.com/p/ganeti/), a tool for managing clusters of virtual servers built on top of Xen and KVM. There is a mailing list (http://groups.google.com/group/ganeti) which is the official contact to the team.

There are lots of presentations about Ganeti online (http://downloads.ganeti.org/presentations/), and some of them are accompanied by videos to be found with a quick search on the internet.

Energy Flow Analysis – Ingenieurbüro Guttenberg & Hördegen

The Engineering Office provides services and tools to companies designing and operating smart systems with energy management: Smart Grids, Smart Houses, Smart Production, and so on. Smart systems are complex: efficiency is only one aspect in a challenging system design. We want to make measurement and optimisation of overall system efficiency as comfortable and easy as possible. The objective is to provide support in choosing between system functionality, performance, safety, and reliability as well as energy efficiency. We provide a support service for the whole development chain, starting with specification, through system design and simulation to system implementation and validation. The advantage of our approach is that we can directly model, investigate and optimise energy flow. This opens new possibilities, such as better optimisation of efficiency, operation, and design for local grids containing electrochemical storage, thermal storage, heat pumps, block heat and power units and so on.

Since it combines good performance and parallelization features while providing a very high level of assurance, we have chosen to execute our technology with Haskell.

For more information, please visit http://www.energiefluss.info. There is an introductory document to the services provided (http://energiefluss.info/img/profile_gh.pdf).

Informatik Consulting Systems AG

ICS AG (http://ics-ag.de), with 11 offices in Germany, use Haskell for their software, as it is a good fit for their domain, which is simulation, safety, and business-critical systems. It affords ICS a competitive edge over the market. Industries ICS work with include advanced technologies, automotive, industrial solutions, and transportation and they have an impressive list of customers (http://ics-ag.de/kunden.html).

Haskell Consultancy Munich

The author of this report runs a Haskell consultancy. Established in 2008, the business provides full-stack support for industries ranging from finance and media to medical and electronics design and automation, with a permanent focus on functional programming. We have a strong background in statistics and operations research. The current trend in the industry is the migration of monolithic legacy software in C, C#, Python, Java, or PHP towards a functional, service-oriented architecture, with on-site training of personnel in the new programming paradigm. Another trend is design of hard realtime applications for industrial use. Further information can be requested via email (⟨haskell.consultancy@gmail.com⟩).

Funktionale Programmierung – Dr. Heinrich Hördegen

Funktionale Programmierung - Dr. Heinrich Hördegen (http://funktional.info) is a Haskell and functional programming software consultancy located in Munich.

Dr. Hördegen has a lot of experience in software engineering and has been an advocate of functional programming since 2005. It follows that during his doctoral thesis at the LORIA (http://www.loria.fr) he was able to design and implement compiler modules for the AVISPA project (http://www.avispa-project.org/) using OCaml.

Dr. Hördegen has been using Haskell as his main technology to implement robust and reliable software since 2009. In his role co-founder and CTO of Ingenieurbüro Guttenberg & Hördegen (http://www.energiefluss.info) he leads the development of proprietary software for energy flow analysis. This complex system is comprised of 50000 lines of code, distributed into 130 modules.

Some of Dr. Hördegen's favourite things about Haskell are algebraic data types, which simplify symbolic computation, the amazing speed Haskell can provide during number crunching, the powerful parallelization capabilities Haskell provides, and finally Cloud Haskell, which lets you easily distribute computations onto whole clusters.

Dr. Hördegen's consultancy sponsors and organizes the Haskell Meetup (http://www.haskell-munich.de/) and supports the Haskell community as a whole.

codecentric AG

Here at codecentric (https://www.codecentric.de/), we believe that more than ever it's important to keep our tools sharp in order to provide real value to our customers. The best way to do this is to provide software expertise and an environment in which people can freely express their ideas and develop their skills. One of the results is codecentric Data Lab, where mathematicians, data scientists and software developers join forces to live up to the big data hype. Another is the Functional Institute (http://clojureworkshop.com/), which helps to spread the word about functional programming with Clojure and Haskell.

We provide services in functional programming in Clojure and Haskell as well as services for Big Data projects, ranging from project support and knowledge sharing to bespoke software development and project management. We are over 200 employees strong in 10 offices around Germany and Europe. You may contact Alex Petrov (⟨alex.petrov@codecentric.de⟩) with any enquiries.

8.4 Better

Report by:	Carl Baatz

Better provides a platform for delivering adaptive online training to students and employees.

Companies and universities work with us to develop courses which are capable of adapting to individual learners. This adaptivity is based on evidence we collect about the learner's understanding of the course material (primarily by means of frequent light-weight assessments). These courses run on our platform, which exposes a (mobile-compatible) web interface to learners. The platform also generates course statistics so that managers/teachers can monitor the progress of the class taking the course and evaluate its effectiveness.

The backend is entirely written in Haskell. We use the `snap` web framework and we have a storage layer written on top of `postgres-simple` which abstracts data retrieval, modification, and versioning. The choice of language has worked out well for us: as well as the joy of writing Haskell for a living, we get straightforward deployment and extensive server monitoring courtesy of `ekg`. Using GHC's profiling capabilities, we have also managed to squeeze some impressive performance out of our deployment.

The application-specific logic is all written in Haskell, as is *most* of the view layer. As much rendering as possible is performed on the backend using `blaze-html`, and the results are sent to a fairly thin single-page web application written in Typescript (which, while not perfect, brings some invaluable static analysis to our front-end codebase).

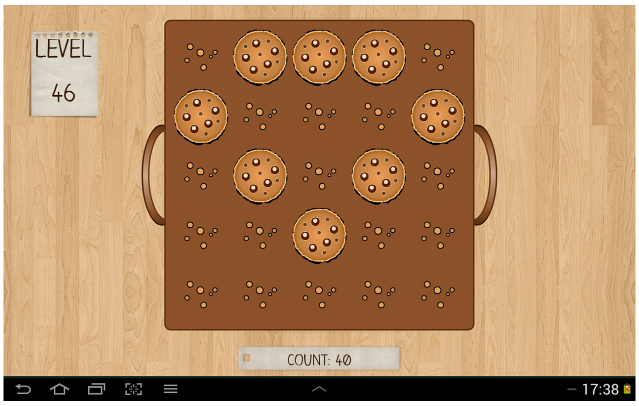
The company is based in Zurich, and the majority of the engineering team are Haskellers. We enjoy a high level of involvement with the Zurich Haskell community and are delighted to be able to host the monthly HaskellerZ user group meetups and the yearly ZuriHac hackathon.

8.5 Keera Studios LTD

Report by: Ivan Perez

Keera Studios Ltd. is a game development studio that uses Haskell to create mobile and desktop games. We have published Magic Cookies!, the first commercial game for Android written in Haskell, now available on Google Play™ (https://goo.gl/cM1tD8).

We have also shown a breakout-like game running on a Android tablet (http://goo.gl/53pK2x), using *hardware acceleration* and *parallelism*. The desktop version of this game additionally supports Nintendo Wiimotes and Kinect. This proves that Haskell truly is viable option for *professional game development*, both for mobile and for desktop. A new game is currently being developed for Android and iOS.



In order to provide more reliable code for our clients, we have developed a battery of small Haskell mobile apps, each testing only one feature. We have dozens of apps, covering SDL and multimedia including multi-touch support, accelerometers, and stereoscopy (for more realistic depth and 3D effects). Our battery also includes apps that communicate with Java via C/C++, used for Facebook/Twitter status sharing, to save game preferences using Android's built-in Shared Preferences storage system, or to create Android widgets. 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 continue to participate in Haskell meetings and engaging in the community, with a recent talk on Game Programming at the Haskell eXchange 2015.

We have developed GALE, a DSL for graphic adventures, together with an engine and a basic IDE that allows non-programmers to create their own 2D graphic adventure games without any knowledge of programming. Supported features include multiple character states and animations, multiple scenes and layers, movement bitmasks (used for shortest-path calculation), luggage, conversations, sound effects, background music, and a customizable UI. The IDE takes care of asset management, generating a fully portable game with all the necessary files. The engine is multi-platform, working *seamlessly* on Linux, Windows and Android. We are continue beta-testing GALE games on Google Play.

We have released Keera Hails, the *reactive* library we use for desktop GUI applications, as Open Source (http://git.io/vTvXg). Keera Hails is being *actively developed* and provides integration with Gtk+, network sockets, files, FRP Yampa signal functions and other external resources. Experimental integration with wxWidgets and Qt is also available, and newer versions include partial backends for Android (using Android's default widget system, communicating via FFI) and HTML DOM (via GHCJS). We are working on providing complete backends for all major GUI toolkits and platforms. Recent updates to our project are geared towards adding documentation, tests and benchmarks, in order to facilitate using, understanding and extending the framework and guaranteeing a high level of quality.

Apart from implementing a simple yet powerful form of reactivity, Keera Hails addresses common problems in Model-View-Controller, providing an application skeleton with a scalable architecture and thread-safe access to the application's internal model. Accompanying libraries feature standarised solutions for common features such as configuation files and internationalisation. We have used this framework in commercial applications (including but not limited to GALE IDE), and in the Open-Source posture monitor Keera Posture (http://git.io/vTvXy). Links to these applications, examples, demos and papers, including a recent paper on Reactive Values and Relations presented at the Haskell Symposium 2015, are available on our website.

We are committed to using Haskell for all our operations. For games we often opt for the Arrowized Functional Reactive Programming Domain-Specific Language Yampa (http://git.io/vTvxQ) or for Keera GALE. For desktop GUI applications we use our own Keera Hails (http://git.io/vTvXg). To create web applications and internal support tools we use Yesod, and continue developing our project management, issue tracking and invoicing web application to facilitate communication with our clients.

Screenshots, videos and details are published regularly on our Facebook page (https://www.facebook.com/keerastudios) and on our company website (http://www.keera.co.uk). If you want to use Haskell in your next game, desktop or web application, or to receive more information, please contact us at .

8.6 plaimi

Report by: Alexander Berntsen

plaimi's expertise lies in identifying problems, researching how to solve them, and developing the necessary software solutions. We have a principled approach to R&D with emphasis on correctness.

Haskell is one of our primary tools in aiming for correctness. We use it for all of our in-house development, and frequently contribute upstream to the free software libraries we use, as well as to the Gentoo Haskell project, being Gentoo users.

All of our own software is also free software. We are commited to free software and copyleft. We are opposed to academic paywalls, patents, and other antisocial activites, that makes information difficult to obtain and share. We exclusively develop free software, and our research is freely available under a copyleft licence, Creative Commons Attribution-ShareAlike.

We are a group of hackers. What this means is that we value the notion of playful cleverness. We enjoy working on challenging problems and finding elegant solutions to them, having lots of fun in the process. Our strong work morale and prinicpled ethics framework result in software that emphasises correctness and aims to liberate and empower users.

Our website is https://secure.plaimi.net/. We are currently looking for work. Do you have any? Get in touch! We'd love to collaborate with you on your projects or our projects, as well as define entirely new projects.

8.7 Stack Builders

Report by:	Stack Builders
Status:	software consultancy

Stack Builders is an international Haskell and Ruby agile software consultancy with 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.
- We maintain several packages in Hackage including hapistrano, inflections, octohat, openssh-github-keys, 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/

8.8 Optimal Computational Algorithms, Inc.

Report by:	Christopher Anand

OCA develops high-performance, high-assurance mathematical software using Coconut (COde CONstructing User Tool), a hierarchy of DSLs embedded in Haskell, which were originally developed at McMaster University. The DSLs encode declarative assembly language, symbolic linear algebra, and algebraic transformations. Accompanying tools include interpreters, simulators, instruction schedulers, code transformers (both rule-based and ad-hoc) and graph and schedule visualizers.

To date, Coconut math function libraries have been developed for five commercial architectures. Taking advantage of Cocont's symbolic code generation, software for reconstructing multi-coil Magnetic Resonance Images was generated from a high-level mathematical specification. The implementation makes full use of dual-CPUs, multiple cores and SIMD parallelism, and is licensed to a multi-national company. The specification is transformed using rules for symbolic differentiation, algebraic simplification and parallelization. The soundness of the generated parallelization can be verified in linear time (measured with respect to program size).

Further reading

- http://www.cas.mcmaster.ca/~kahl/Publications/TR/Anand-Kahl-2007a_DSL/
- http://www.cas.mcmaster.ca/~anand/papers/AnandKahlThaller2006.pdf
- http://www.cas.mcmaster.ca/sqrl/papers/SQRLreport50.pdf
- https://macsphere.mcmaster.ca/handle/11375/10755
- http://www.cas.mcmaster.ca/~anand/papers/CAS-14-05-CA.pdf

8.9 Snowdrift.coop

Report by:	Bryan Richter
Participants:	Aaron Wolf et al.
Status:	Work in progress

Snowdrift.coop is a web platform for funding and supporting free/libre/open projects. We are tackling the 'snowdrift dilemma' that limits contributions to non-rivalrous goods such as open-source software. The organization is a non-profit multi-stakeholder cooperative, and all code is available under OSI- and FSF-approved licenses. Haskell is our primary programming language, and we welcome any interested contributors to help us accelerate our progress.

In our current work we have recently focused on three main areas: 1) opening the project to greater participation through code refactoring and tool development, 2) firming up the co-op governance structure, and 3) creating a comprehensive design framework for the website. There is also plenty of ongoing feature development on various aspects of the live site.

One notable contribution Snowdrift has made to the Haskell ecosystem is a thorough 'getting started' experience for beginners, from text editor suggestions to introductions to git. As part of that effort, we have developed a foolproof build process, with a tip of our hats to the new tool `stack`, and have developed a database initialization tool and various Yesod integrations with ghci and text editors. Interested contributors will find many opportunities for progress in this area.

The funding mechanism is not yet functional but progressing. Once functional, Snowdrift.coop itself will be a supported project, and should prove to be an excellent test-case for the adoption and success of the concept. In the meanwhile, we are actively looking for ways to improve both productivity and opportunities for our distributed team of volunteers. Experienced Haskellers are invited to mentor volunteers, take ownership of component libraries, and provide opinions and insights. New Haskellers—not to mention designers, writers, economists, legal professionals, or anyone else philosophically inclined to our mission of freeing the commons—are especially welcome; we pride ourselves on being inclusive and approachable by (non-)programmers at any level of technical sophistication!

Further reading

- https://snowdrift.coop (main site, with many resources)
- https://lists.snowdrift.coop (mailing lists)
- https://git.gnu.io/snowdrift (code repository)

9 Research and User Groups

9.1 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

9.2 Artificial Intelligence and Software Technology at Goethe-University Frankfurt

Report by:	David Sabel
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. Our results include the correctness of strictness analysis by abstract reduction, results on the equivalence of the call-by-name and call-by-need semantics, correctness of program transformations, and investigations on the conservativity of language extensions. In recent research we analyzed the question whether program transformations are optimizations, i.e. whether they improve the time resource behavior. We showed that common subexpression elimination is an improvement and we also showed that our notion of improvement is (asymptotically) resource equivalent to the improvement theory developed by Moran & Sands. Ongoing work is to enhance the techniques to (preferably automatically) verify that program transformations are improvements.

We also 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. Also automated termination provers for term rewrite systems are used in a part of the automation. Future research goals are to automate correctness proofs of program translations as they appear in compilers.

Concurrency. We analyzed a higher-order functional language with concurrent threads, monadic IO, MVars and concurrent futures which models Concurrent Haskell. We proved correctness of program transformations, correctness of an abstract machine, and 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. Our goal is to reconstruct known algorithms on uncompressed data for their use on grammars without prior decompression. We implemented several algorithms as a Haskell library including efficient algorithms for fully compressed pattern matching.

Cycle Rewriting. Cycle rewrite systems perform string rewriting on cycles – a cycle is a string where start and end are connected. Recently, we developed techniques to prove termination of cycle rewrite systems. A tool (called cycsrs) was implemented in Haskell to combine several termination and nontermination techniques and tools to automatically prove cycle termination. The tool participated in 2015 edition of the Termination Competition.

Further reading

http://www.ki.informatik.uni-frankfurt.de/research/HCAR.html

9.3 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. Stephen Adams is working on advanced refactoring of Haskell programs, extending HaRe. Andreas Reuleaux is building a refactoring tool for a dependently typed functional language in Haskell. Maarten Faddegon is working on making tracing for Haskell practical and easy to use by building the lightweight tracer and debugger Hoed. Olaf Chitil is also working on tracing, including the further development of the Haskell tracer Hat, and on type error debugging. Meng Wang is working on lenses, bidirectional transformation and property-based testing (QuickCheck). Scott Owens is working on verified compilers for the (strict) functional language CakeML. Colin Runciman from the University of York visited the PLAS group from April to September 2015. In particular he and Stefan Kahrs worked on minimising regular expressions, implemented in Haskell.

We are always looking for more PhD students. We are particularly keen to recruit students interested in programming tools for verification, 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/
- Kazutaka Matsuda and Meng Wang: Applicative Bidirectional Programming with Lenses. ICFP 2015. https://kar.kent.ac.uk/49084/
- Maarten Faddegon and Olaf Chitil: Algorithmic Debugging of Real-World Haskell Programs: Deriving Dependencies from the Cost Centre Stack. PLDI 2015. https://kar.kent.ac.uk/49003/
- Haskell: the craft of functional programming: http://www.haskellcraft.com
- 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/

9.4 Haskell at KU Leuven, Belgium

Report by:	Tom Schrijvers

Functional Programming, and Haskell in particular, is an active topic of research and teaching in the Declarative Languages & Systems group of KU Leuven, Belgium.

Teaching Haskell is an integral part of the curriculum for both informatics bachelors and masters of engineering in computer science. In addition, we offer and supervise a range of Haskell-related master thesis topics.

Research We actively pursue various Haskell-related lines of research. Some recent and ongoing work:

◦ Steven Keuchel works on InBound, a Haskell-like DSL for specifying abstract syntax trees with binders.
◦ George Karachlias works on extending GHC's pattern match checker to deal with GADTs, in collaboration with Dimitrios Vytiniotis and Simon Peyton Jones.
◦ Alexander Vandenbroucke extends the nondeterminism monad with tabulation, a form of memoization "on steroids" from logic programming.
◦ With Nicolas Wu we have recently worked on fusion for free monads to obtain efficient algebraic effect handlers. See our forthcoming MPC 2015 paper.
◦ With Mauro Jaskelioff and Exequiel Rivas we launch a new slogan:

> Nondeterminism monads are just near-semirings in the category of endofunctors, what's the problem?

See our forthcoming paper at PPDP 2015.

Leuven Haskell User Group We host the Leuven Haskell User Group, which has held its first meeting on March 3, 2015. The group meets roughly every other week and combines formal presentations with informal discussion. For more information: http://groups.google.com/forum/#!forum/leuven-haskell

Further reading

http://people.cs.kuleuven.be/~tom.schrijvers/Research/

9.5 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/2015

9.6 Functional Programming at Chalmers

Report by:	Jean-Philippe Bernardy

Functional Programming is an important component of the CSE department at Chalmers and University of Gothenburg. In particular, Haskell has a very important place, as it is used as the vehicle for teaching and numerous research projects. Besides functional programming, language technology, and in particular domain specific languages is a common aspect in our projects. We have hosted ICFP 2014 in Gothenburg this September.

Property-based testing. QuickCheck, developed at Chalmers, is one of the standard tools for testing Haskell programs. It has been ported to Erlang and used by Ericsson, Quviq, and others. QuickCheck continues to be improved. Quickcheck-based tools and related techniques are currently being developed:

◦ We have shown how to successfully apply QuickCheck to test polymorphic properties.
◦ A new exhaustive testing tool (testing-feat on Hackage) has been developed. It is especially suited to generate test cases from large groups of mutually recursive syntax tree types. A paper describing it was presented at the Haskell Symposium 2012.
◦ Testing Type Class Laws: the specification of a class in Haskell often starts with stating, in comments, the laws that should be satisfied by methods defined in

instances of the class, followed by the type of the methods of the class. We have developed a library (ClassLaws) that supports testing such class laws using QuickCheck.

Parsing: BNFC. The BNF Converter (BNFC) is a frontend for various parser generators in various languages. BNFC is written in Haskell and is commonly used as a frontend for the Haskell tools Alex and Happy. BNFC has recently been extended in two directions:

- A Haskell backend, which offers incremental and parallel parsing capabilities, as well as the ability to parse context-free grammars in full generality, has been added to BNFC. The underlying concepts are described in a paper published at ICFP 2013.
- BNFC has been embedded in a library (called BNFC-meta on Hackage) using Template-Haskell. An important aspect of BNFC-meta is that it automatically provides quasi-quotes for the specified language. This includes a powerful and flexible facility for anti-quotation.

Parsing: Combinators. A new package for combinator-based parsing has been released on Hackage. The combinators are based on the paper Parallel Parsing Processes. The technique is based on parsing in parallel all the possibly valid alternatives. This means that the parser never "hold onto" old input. A `try` combinator is also superfluous.

Parsing: Natural languages. Grammatical Framework is a declarative language for describing natural language grammars. It is useful in various applications ranging from natural language generation, parsing and translation to software localization. The framework provides a library of large coverage grammars for currently fifteen languages from which the developers could derive smaller grammars specific for the semantics of a particular application.

Generic Programming. Starting with Polytypic Programming in 1995 there is a long history of generic programming research at Chalmers. Recent developments include fundamental work on parametricity. This work has led to the development of a new kind of abstraction, to generalize notions of erasure. This means that a new kind of generic programming is available to the programmer. A paper describing the idea was presented in ICFP 2013.

Our research on generic-programming is lively, as witnessed by a constant stream of publications: Testing Type Class Laws, Functional Enumeration of Algebraic Types (FEAT), Testing versus proving in climate impact research and Dependently-typed programming in scientific computing — examples from economic modelling. The last two are part of our effort to contribute to the emerging research programme in Global Systems Science.

Program Inversion/bidirectionalization. Program transformation systems that generate pairs of programs that are some sort of inverses of each other. The pairs are guaranteed to be consistent by construction with respect to certain laws. Applications include pretty-printing/parsing, XML transformation etc. The work is done in collaboration with University of Tokyo and University of Bonn.

Language-based security. SecLib is a light-weight library to provide security policies for Haskell programs. The library provides means to preserve confidentiality of data (i.e., secret information is not leaked) as well as the ability to express intended releases of information known as declassification. Besides confidentiality policies, the library also supports another important aspect of security: integrity of data. SecLib provides an attractive, intuitive, and simple setting to explore the security policies needed by real programs.

Type theory. Type theory is strongly connected to functional programming research. Many dependently-typed programming languages and type-based proof assistants have been developed at Chalmers. The Agda system (→ 4.1) is the latest in this line, and is of particular interest to Haskell programmers. While today's GHC incorporates much of the dependently-typed feature set, supporting plain old Haskell means a certain amount of clunkiness. Agda provides a cleaner language, while remaining close to Haskell syntax.

Embedded domain-specific languages. The functional programming group has developed several different domain-specific languages embedded in Haskell. The active ones are:

- **Feldspar** (→ 7.12.2) is a domain-specific language for digital signal processing (DSP).
- **Obsidian** is a language for data-parallel programming targeting GPUs.
  Most recently we used Obsidian to implement an interesting variation of counting sort that also removes duplicate elements. This work was presented at FHPC 2013.

We are also working on general methods for EDSL development:

- **Syntactic** is a library that aims to support the definition of EDSLs. The core of the library was presented at ICFP 2012. The paper presents a generic model of typed abstract syntax trees in Haskell, which can serve as a basis for a library supporting the implementation of deeply embedded DSLs.
- **Names For Free.** A new technique for representing names and bindings of object languages represented as Haskell data types has been developed.

The essence of the technique is to represent names using *typed* de Bruijn indices. The type captures exactly the context where the index is valid, and hence is as safe to use as a name. The technique was presented at Haskell Symposium 2013. We are currently extending the technique to work for proofs as well as programs.

- **Circular Higher-Order Syntax** We have also developed a light-weight method for generating names while building an expression with binders. The method lends itself to be used in the front end of EDSLs based on higher-order syntax. The technique was presented at ICFP 2013.
- **Simple and Compositional Monad Reification** A method for reification of monads (compilation of monadic embedded languages) that is both simple and composable. The method was presented at ICFP 2013.

Automated reasoning. We are responsible for a suite of automated-reasoning tools:

- **Equinox** is an automated theorem prover for pure first-order logic with equality. Equinox actually implements a hierarchy of logics, realized as a stack of theorem provers that use abstraction refinement to talk with each other. In the bottom sits an efficient SAT solver. Paradox is a finite-domain model finder for pure first-order logic with equality. Paradox is a MACE-style model finder, which means that it translates a first-order problem into a sequence of SAT problems, which are solved by a SAT solver.
- **Infinox** is an automated tool for analysing first-order logic problems, aimed at showing finite unsatisfiability, i.e., the absence of models with finite domains. All three tools are developed in Haskell.
- **QuickSpec** generates algebraic specifications for an API automatically, in the form of equations verified by random testing. http://www.cse.chalmers.se/~nicsma/quickspec.pdf
- **Hip** (the Haskell Inductive Prover) is a new tool to automatically prove properties about Haskell programs by using induction or co-induction. The approach taken is to compile Haskell programs to first order theories. Induction is applied on the meta level, and proof search is carried out by automated theorem provers for first order logic with equality.
- On top of Hip we built **HipSpec**, which automatically tries to find appropriate background lemmas for properties where only doing induction is too weak. It uses the translation and structural induction from Hip. The background lemmas are from the equational theories built by QuickSpec. Both the user-stated properties and those from QuickSpec are now tried to be proven with induction. Conjectures proved to be theorems are added to the theory as lemmas, to aid proving later properties which may require them. For more information, see http://web.student.chalmers.se/~danr/hipspec-atx.pdfthe draft paper.

Teaching. Haskell is present in the curriculum as early as the first year of the BSc programme. We have four courses solely dedicated to functional programming (of which three are MSc-level courses), but we also provide courses which use Haskell for teaching other aspects of computer science, such the syntax and semantics of programming languages, compiler construction, data structures and parallel programming.

9.7 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. There are three major Haskell projects at KU (as well as numerous smaller ones): the GHC rewrite plugin HERMIT (→ 7.3.1), the Wakarusa Project (→ 5.1.2), and the Haskino Project (→ 6.1.8). All three projects are using now using the remote monad design pattern (→ 6.5.7) as a key technology.

Further reading

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

9.8 Regensburg Haskell Meetup

Report by:	Andres Löh

Since autumn 2014 Haskellers in Regensburg, Bavaria, Germany have been meeting roughly once per month to socialize and discuss Haskell-related topics.

I'm happy to say that this meetup continues to thrive. We typically have between 10 and 15 attendees (which is really not bad if you consider the size of

Regensburg), and we often get visitors from Munich, Nürnberg and Passau.

New members are always welcome, whether they are Haskell beginners or experts. If you are living in the area or visiting, please join! Meetings are announced a few weeks in advance on our meetup page: http://www.meetup.com/Regensburg-Haskell-Meetup/.

9.9 Haskell in the Munich Area

Report by:	Haskell Consultancy Munich

Haskell in education

Haskell is widely used as an educational tool for both teaching students in computer science as well as for teaching industry programmers transitioning to functional programming. It is very well suited for that and there is a huge educational body present in Munich.

Haskell at the Ludwig-Maximilians-Universität, Munich

Following a limited test run last year which included 12 people, the Institut für Informatik (Institute for Computer Science) has switched their *Programming and Modelling* (http://www.tcs.ifi.lmu.de/lehre/ss-2014/promo) course from ML to Haskell. It runs during the summer semester and is frequented by 688 students. It is a mandatory course for Computer Science and Media Information Technology students as well as many students going for degrees related to computer science, e.g. Computer Linguistics (where lambda calculus is very important) or Mathematics. The course consists of a lecture and tutorial and is led by Prof. Dr. Martin Hofmann and Dr. Steffen Jost. It started on the 7th April, 2014. It is expected that 450 students will complete the course. Notably, the course is televised and is accessible at the LMU portal for Programming and Modelling (https://videoonline.edu.lmu.de/de/sommersemester-2014/5032).

Haskell is also used in *Advanced Functional Programming* (https://www.tcs.ifi.lmu.de/lehre/ss-2012/fun) which runs during the winter semester and is attended by 20-30 students. It is mandatory for Computer Science as well as Media Information Technology students.

Neither of these courses has any entry requirements, and you may enter the university during the summer semester, which makes them very accessible.

Any questions may be directed to Dr. Steffen Jost (⟨jost@tcs.ifi.lmu.de⟩).

Haskell at the Hochschule für angewandte Wissenschaften München (Academy for applied sciences Munich)

Haskell is taught in two courses at the College: Functional Programming and Compiler Design. Both courses consist of lectures and labs. Prof. Dr. Oliver Braun has brought Haskell to the school and has been using it during the last year for both courses; before that he taught Haskell at FH Schmalkalden Thüringen (http://www.fh-schmalkalden.de/) for 3.5 years.

Compiler Design (http://ob.cs.hm.edu/lectures/compiler) is a compulsory course taught, depending on the group, using Haskell, Scheme, or Java. The Haskell version is frequented by over 40 students. Part of the note depends on a compiler authored in Haskell.

Functional Programming (http://ob.cs.hm.edu/lectures/fun) is a new, non-compulsory course attended by 20 students, taught with Haskell. The grade depends among others on an exam in Haskell knowledge and a project authored in Haskell with the Yesod web framework. It is taught with Learn You a Haskell and teaches practical skills such as Cabal, Haddock, QuickCheck, HUnit, Git, and Yesod. The school department's website itself is in Snap.

Dr. Oliver Braun has started using Haskell in 1997, when it became the first programming language he's used during his studies. He has later used Haskell during his thesis and afterwards his dissertation. He finds Haskell great for teaching. Oliver Braun can be reached via email (⟨ob@cs.hm.edu⟩).

Haskell as a teaching tool in the industry

Haskell is used in Munich to teach functional programming to industrial programmers. Since it uses the same basic programming model, it can also be used as a simple learning tool to introduce people to Scala. That is because both are based on System F and Haskell has a very clean, minimal implementation of it. It has been successfully used to teach a team of 10 PHP programmers the basics of functional programming and Scala and, together with other educational tools, get them up and running within a couple months, during which time the team remained productive. This approach makes it easy for companies to switch from the likes of PHP, Java, .NET, or C# to functional programming (Haskell, Scala, Clojure). At the same time the project switched to SOA (service oriented architecture) using the Twitter scala libraries. Having understood the basics of FP in Haskell, the team could easily move onto the more complicated task of understanding the more unique and intricate parts of Scala that correspond to extensions to System F while being able to understand Scala's syntax. You may contact the author of this report (⟨haskell.consultancy@gmail.com⟩) for details.

Haskell community

There are several meetups dedicated to Haskell in Munich. The organizers have initiated cooperation in order to build and support the local community, as well as the community in Germany. There is something related to Haskell happening every week.

The organizers would like to establish contact with other Haskell communities in Germany as well as the whole world. You may write to the Haskell Hackathon organizer (⟨haskell.hackathon@gmail.com⟩). As of 2014, it is known that there is Haskell activity in Berlin, Cologne (Köln), Düsseldorf, Frankfurt am Main, Halle, Hamburg, and Stuttgart, as well as in Austria, Switzerland and the Czech Republic. If you're from one of those communities, please write us! The Munich community welcomes any new connections from other locations.

The community receives notable guests, such as:

- Reinhard Zumkeller, one of the regular contributors to the OEIS. Reinhard likes to use Haskell for work with integer sequences.
- Lars R. Hupel, the maintainer of scalaz. Lars teaches with Haskell at the local university and enjoys advanced topics in type systems and category theory.
- Andres Löh, co-founder of Well-Typed LLP. Andres always brings up very practical discussions on the use of Haskell. For example, he has recently held a presentation on the Par monad.
- Heiko Seeberger from . Heiko is interested in all sorts of functional programming and loves Haskell for its simplicity and consistency.
- many others which the author of this report could not reach for comment before the publication due to time constraints.

The community is very lively and there are many initiatives being worked on. For example, actively popularizing Haskell in the local industry, creating a network of companies, programmers, and informational events. The author of this report may be reached for more information (⟨haskell.consultancy@gmail.com⟩).

Haskell Hackathon

The Haskell Hackathon is a small meeting for people who would like to build their Haskell skillset. People bring their laptops and work on one of the proposed topics together, sharing experience and teaching each other. Topics range from very easy (if you don't know Haskell, you may come and the organizer will teach you the basics one on one) through intermediate (how to best set up the dev env, how to read the papers, how to use important libraries) to very advanced (free applicatives, comonads). Defocus is discouraged (subjects not related to Haskell are limited). The operating language is German but if you speak any other language you are welcome to join us.

The Hackathon is organized by the author of this report (⟨haskell.consultancy@gmail.com⟩) and is currently in its second year. It is frequented by the staff and students of the local universities, industry programmers, as well as Haskell enthusiasts. You may contact the Hackathon with any questions via email (⟨haskell.hackathon@gmail.com⟩).

We keep track of ideas we would like to explore during the Haskell Hackathon (http://haskell-hackathon.no-ip.org/ideen.html). Any and all new questions are welcome!

Haskell Meetup

The Haskell Meetup, also called Haskell Stammtisch (which directly translates to: Haskell regulars table) is a social event for the Haskell community. It is the original Haskell event in Munich. Everyone is welcome (even non-Haskell programmers!). It happens once a month, usually at Cafe Puck which is a pub in one of the cooler parts of Munich, where the members can eat schnitzel and drink beer while chatting about topics ranging from Haskell itself to abstract mathematics, industrial programming, and so on. The group is very welcoming and they make you feel right at home. The Meetup attracts between 15 and 20 guests and there's a large proportion of regulars. Attendance ranges from students, through mathematicians (notably the OEIS has a presence), industry programmers, physicists, and engineers. The Meetup receives international guests and sometimes we hold lectures.

The Haskell Meetup, established 29th September 2011 by Heinrich Hördegen. It is sponsored by Funktionale Programmierung Dr. Heinrich Hördegen (http://funktional.info) and Energy Flow Analysis – Ingenieurbüro Guttenberg & Hördegen (http://www.energiefluss.info).

Munich Lambda

Munich Lambda (http://www.meetup.com/Munich-Lambda/) was founded on Jun 28, 2013 by Alex Petrov. There have been 12 events so far, on topics including Haskell, Clojure, and generally functional programming, as well as Emacs. Meetups on the topic of Haskell occur every month to two months.

Typically, the meetup begins with a short introductory round where the visitors can talk about their work or hobbies and grab some food (provided by sponsors), followed by couple of presentations, and topped off by an informal discussion of relevant topics and getting to know each other. It is a great opportunity to meet other likeminded people who like Haskell or would like to start out with it.

Munich Lambda is sponsored by codecentric (http://www.codecentric.de/) and StyleFruits (http://www.stylefruits.de).

Mailing lists in Munich

There are two mailing lists in use: https://lists.fs.lmu.de/mailman/listinfo/high-order-munich and http://mailman.common-lisp.net/cgi-bin/mailman/listinfo/munich-lisp.

The lists are used for event announcements as well as to continue discussions stemming from recent events. It is usually expected that anyone subscribed to one is also on the other, but conversations normally happen only on one or the other. There are 59 subscribers to high-order-munich.

There is a mail distributor for the Haskell Hackathon (http://haskell-hackathon.no-ip.org). In order to receive emails, send mail to the Haskell Hackathon organizer (⟨haskell.hackathon@gmail.com⟩).

ZuriHac 2014, Budapest Hackathon 2014, and the Munich Hackathon

There is a group of people going to ZuriHac 2014 (http://www.haskell.org/haskellwiki/ZuriHac2014). We are currently planning the logistics. If you would like to join us, you may write to the high-order-munich mailing list (https://lists.fs.lmu.de/mailman/listinfo/high-order-munich). Some people going to ZuriHac want to visit Munich first and will be received by the Munich community. There will be events during the week before ZuriHac. Boarding in Munich is inexpensive; the bus to Zurich is only 15 Euro and you may travel with a group of Haskell enthusiasts. There is a lot to see and visit in Munich. It is an easy travel destination as the Munich Airport has direct connections with most large airports in the world. Zurich is 312 kilometers (194 miles) away and no passport is necessary to travel from Munich to Zurich.

In addition, there is a group going to the Budapest Hackathon (http://www.haskell.org/haskellwiki/BudapestHackathon2014), which is a week before ZuriHac. To connect those two together, both geographically and in time, a Munich Lambda event is planned for the 4th of June in Munich. The travel is very cheap (the bus tickets from Budapest to Munich and from Munich to Zurich are on the order of 30 Euro). This way people can attend all three, completing what has been nicknamed the Haskell World Tour 2014. For more information you may contact the organizer of the Haskell Hackathon in Munich (⟨haskell.hackathon@gmail.com⟩). You may have fun, meet people from three huge Haskell communities, travel together, and see the world, all in one week!

Halle

There is a group of Haskell members going to HaL-9 in Halle (http://www.haskell.org/pipermail/haskell/2014-March/024115.html), which is 439 kilometers (273 miles) away. Henning Thielemann (⟨schlepptop@henning-thielemann.de⟩), the event organizer, is in charge of car pooling for visitors coming from all locations.

9.10 HaskellMN

Report by:	Kyle Marek-Spartz
Participants:	Tyler Holien
Status:	ongoing

HaskellMN is a user group from Minnesota. We have monthly meetings on the third Wednesday in downtown Saint Paul.

Further reading

http://www.haskell.mn