Haskell Communities and Activities Report

http://tinyurl.com/haskcar

Twenty-Third Edition — November 2012

Janis Voigtländer (ed.)
Andreas Abel
Heinrich Apfelmus
Emil Axelsson
Doug Beardsley
Jean-Philippe Bernardy
Jeroen Bransen
Gwern Branwen
Joachim Breitner
Björn Buckwalter
Erik de Castro Lopo
Olaf Chitil
Duncan Coutts
Jason Dagit
Nils Anders Danielsson
Romain Demeyer
Daniel Díaz
Atze Dijkstra
Adam Drake
Sebastian Erdweg
Ben Gamari
Andy Georges
Patai Gergely
Brett G. Giles
Andy Gill
George Giorgidze
Torsten Grust
Jurriaan Hage
Bastiaan Heeren
Mike Izbicki
PÁLI Gábor János
Guillaume Hoffmann
Csaba Hruska
Paul Hudak
Oleg Kiselyov
Michal Konečný
Eric Kow
Ben Lippmeier
Andres Löh
Hans-Wolfgang Loidl
Rita Loogen
Ian Lynagh
Christian Maeder
José Pedro Magalhães
Ketil Malde
Antonio Mamani
Simon Marlow
Dino Morelli
JP Moresmau
Ben Moseley
Takayuki Muranushi
Jürgen Nicklisch-Franken
Tom Nielsen
Rishiyur Nikhil
Jens Petersen
David Sabel
Uwe Schmidt
Martijn Schrage
Tom Schrijvers
Andrew G. Seniuk
Jeremy Shaw
Christian Höner zu Siederdissen
Michael Snoyman
Doaitse Swierstra
Henning Thielemann
Sergei Trofimovich
Bernhard Urban
Marcos Viera
Janis Voigtländer
Daniel Wagner
Greg Weber
Kazu Yamamoto
Edward Z. Yang
Brent Yorgey

Preface

This is the 23rd edition of the Haskell Communities and Activities Report. As usual, fresh entries are formatted using a blue background, while updated entries have a header with a blue background. Entries for which I received a liveness ping, but which have seen no essential update for a while, have been replaced with online pointers to previous versions. Other entries on which no new activity has been reported for a year or longer have been dropped completely. Please do revive such entries next time if you do have news on them.

A call for new entries and updates to existing ones will be issued on the Haskell mailing list in April. Now enjoy the current report and see what other Haskellers have been up to lately. Any feedback is very welcome, as always.

Janis Voigtländer, University of Bonn, Germany, ⟨hcar@haskell.org⟩

Contents

1 Community

1.1 haskell.org

Report by:	Jason Dagit
Participants:	Ganesh Sittampalam, Edward Z. Yang, Vo Minh Thu, Mark Lentczner, Edward Kmett, Brent Yorgey
Status:	active

The haskell.org committee is in its second year of operation managing the haskell.org infrastructure and money. The committee's "home page" is at http://www.haskell.org/haskellwiki/Haskell.org_committee, and occasional publicity is via a blog (http://haskellorg.wordpress.com) and twitter account (http://twitter.com/#!/haskellorg) as well as the Haskell mailing list.

Since the last community report, the following has happened:

haskell.org incorporation

Haskell.org has now joined Software in the Public Interest (http://www.spi-inc.org). This allows haskell.org to accept donations as a US-based non-profit as well as pay for services with these donations. Currently, most of the money in the haskell.org account comes from GSoC participation.

We are currently in the process of establishing guidelines for fund raising and appropriate ways to spend funds. The main expense of haskell.org at this time is server hosting. The GSoC participant reimbursement is actually paid by Google and we do not consider this a normal expense as Google reimburses us for the full amount.

Assets

At the start of 2011 the haskell.org account had $7,261.73 USD, and by the end of the year the account balance was $13,056.32. The haskell.org expenses for 2011 include:

- GSoC Participant Reimbursements: $2,816.41
- Server Hosting Fees: $705.41

The haskell.org income for 2011 includes:

- GSoC Payments: $6,500.00
- Google GSoC Participant Reimbursements: $2,816.41

Note that the participant reimbursement paid by haskell.org matches the reimbursement given to haskell.org by Google. The haskell.org credits for 2011 include only GSoC payments of $9,316.41, leaving us with a balance of $13,056.32 at the end of 2011.

Haskell.org has the following server assets:

- abbot, kindly hosted by Galois
- sparky, kindly hosted by Chalmers but technically owned by Oxford Department of Computer Science
- lambda, commercially hosted
- lun, a VM hosted on lambda
- www, a VM hosted on lambda
- haskell.org domain name

General

The haskell.org infrastructure is becoming more stable, but still suffers from occasional hiccups. While the extreme unreliability we saw for a while has improved with the reorganisation, the level of sysadmin resource/involvement is still inadequate. The committee is open to ideas on how to improve the situation.

With the task of incorporation behind us, the haskell.org committee can now focus on establishing guidelines around donations, fund raising, and appropriate uses of funds.

1.2 Haskellers

Report by:	Michael Snoyman
Status:	experimental

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

Since the May 2011 HCAR, Haskellers has added polls, which provides a convenient means of surveying a large cross-section of the active Haskell community. There are now over 1300 active accounts, versus 800 one year ago.

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 In Japanese: Learn You a Haskell for Great Good!

Report by:	Takayuki Muranushi
Participants:	Hideyuki Tanaka
Status:	available

An official translation of the book "Learn You a Haskell for Great Good!" by Miran Lipovača (http://learnyouahaskell.com/) to Japanese is now available in stores.

The original book is an elaborate and popular introduction to the programming language Haskell. The reader will walk through the playland of Haskell decorated with funky examples and illustrations, and without noticing any difficulties, will become one with the core concepts of Haskell, say types, type classes, lazy evaluations, functors, applicatives and monads. The translators have added a short article on handling multi-byte strings in Haskell.

We are grateful to all the people's work that made this wonderful book available in Japanese, including the publisher, our kind reviewers, and the original author Miran. We wish for prosperity of the Haskell community in Japan and in many countries, and for those who don't read Japanese, we'd just like to let you know that we're doing fine in Japan!

Publication details:

◦ Published by Ohmsha, 2012. ISBN 4274068854.
◦ Original book published by No Starch Press, 2011. ISBN 1593272839.

Book website:

◦ http://ssl.ohmsha.co.jp/cgi-bin/menu.cgi?ISBN=978-4-274-06885-0

Further reading

http://www.amazon.co.jp/dp/4274068854/

2.2 The Monad.Reader

Report by:	Edward Z. Yang

There are many academic papers about Haskell and many informative pages on the HaskellWiki. Unfortunately, there is not much between the two extremes. That is where The Monad.Reader tries to fit in: more formal than a wiki page, but more casual than a journal article.

There are plenty of interesting ideas that might not warrant an academic publication—but that does not mean these ideas are not worth writing about! Communicating ideas to a wide audience is much more important than concealing them in some esoteric journal. Even if it has all been done before in the Journal of Impossibly Complicated Theoretical Stuff, explaining a neat idea about "warm fuzzy things" to the rest of us can still be plain fun.

The Monad.Reader is also a great place to write about a tool or application that deserves more attention. Most programmers do not enjoy writing manuals; writing a tutorial for The Monad.Reader, however, is an excellent way to put your code in the limelight and reach hundreds of potential users.

Since the last HCAR there has been one new issue, featuring articles on Haskell error reporting, enumerating tuples and the MapReduce monad.

Further reading

http://themonadreader.wordpress.com/

2.3 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:

Haskell with only one type class

Type classes are so ingrained in Haskell that one can hardly think about the language without them. And yet, if we remove the type class declaration and the standard type classes, leaving the language with a single, fixed, pre-defined type class with a single method, no expressivity is lost. We can still write all Haskell98 type-class programming idioms including constructor classes, as well as multi-parameter type classes and even some functional dependencies. Adding the equality constraint gives us all functional dependencies, bounded existentials, and even associated data types. Besides clarifying the role of type classes as method bundles, we propose a simpler model of overloading resolution than that of Hall et al.

http://okmij.org/ftp/Haskell/TypeClass.html#Haskell1

Higher-order polymorphic selection

This mini-tutorial shows an example of how looking at the problem in a different way turns it from impossible (at least, in System F) to elementary, expressible in the Hindley-Milner system. The problem is about selectors, such as tuple selectors:

$$\begin{aligned}&\mathit{fst} \;::\mathit{forall}\ a\ b\ .\ (a, b) \to a\\&\mathit{snd} ::\mathit{forall}\ a\ b\ .\ (a, b) \to b\end{aligned}$$

First, we would like to write a function g that takes a selector as an argument and applies it to several *heterogeneous* tuples. For example:

$$g\ \mathit{sel} = (\mathit{sel}\ (1, \texttt{'b'}), \mathit{sel}\ (\mathit{true}, \texttt{"four"}))$$

It is already a problem to type such a function in System F, let alone in the Hindley-Milner system. But we want more: a function that takes functions like g as an argument:

$$\begin{aligned}&\mathit{fs}\ g = (g\ \mathit{snd}, (), g\ \mathit{fst})\\&\mathit{test} = f\ (\lambda \mathit{sel} \to (\mathit{sel}\ (1, \texttt{'b'}), \mathit{sel}\ (\mathit{true}, \texttt{"four"})))\end{aligned}$$

The mini-tutorial first shows a brute-force solution, emulating the necessary higher-rank polymorphism. Then we change the point of view: we now represent the function as a 'table', 'indexed' by the selector argument. This change of representation is quite like applying the eta-rule for sums. In this new representation, the problem becomes trivial:

$$\begin{aligned}&g' = (\mathit{g\_\_fst}, \mathit{g\_\_snd})\\&\textbf{where}\\&\mathit{g\_\_fst} \;= (\mathit{fst}\ (1, \texttt{'b'}), \mathit{fst}\ (\mathit{True}, \texttt{"four"}))\\&\mathit{g\_\_snd} = (\mathit{snd}\ (1, \texttt{'b'}), \mathit{snd}\ (\mathit{True}, \texttt{"four"}))\\&\mathit{fs}\ g' = (\mathit{fst}\ g', (), \mathit{snd}\ g')\\&\mathit{test} = \mathit{fs}\ g'\end{aligned}$$

http://okmij.org/ftp/Computation/extra-polymorphism.html#ho-poly-sel

Parametric polymorphism over a type class

A Haskell-Cafe message posed a problem of parameterizing a function by a type class rather than by a type. Can we write the following two definitions as a single polymorphic definition, eliminating the code duplication?

$$\begin{aligned}&\mathit{foo} :: (\mathit{Num}\ c, \mathit{Num}\ d) \Rightarrow\\&\qquad (\mathit{forall}\ b\ .\ \mathit{Num}\ b \Rightarrow a \to b) \to a \to (c, d)\\&\mathit{foo}\ f\ x = (f\ x, f\ x)\\&\mathit{bar} :: (\mathit{Read}\ c, \mathit{Read}\ d) \Rightarrow\\&\qquad (\mathit{forall}\ b\ .\ \mathit{Read}\ b \Rightarrow a \to b) \to a \to (c, d)\\&\mathit{bar}\ f\ x = (f\ x, f\ x)\end{aligned}$$

The only difference between *foo* and *bar* is the type class constraint: *Num* vs *Read*. The problem thus is to parameterize over constraints. The mini-tutorial develops the solution without resorting to the recently added Constraint kind, showing that first-class constraints have always been available in Haskell.

The type class Apply

$$\begin{aligned}&\textbf{class}\ \mathit{Apply}\ l\ a\ b \mid l\ b \to a\ \textbf{where}\\&\quad \mathit{apply} :: l \to a \to b\\&\textbf{data}\ \mathit{LRead} = \mathit{LRead}\\&\textbf{instance}\ \mathit{Read}\ b \Rightarrow \mathit{Apply}\ \mathit{LRead}\ \mathit{String}\ b\ \textbf{where}\\&\quad \mathit{apply}\ \_ = \mathit{read}\end{aligned}$$

introduces the correspondence between constraints and ordinary types. For example, the shown instance maps the type *LRead* to the constraint *Read*. The type-class Apply also gets around higher-rank polymorphism; therefore, the types are inferrable. The mini-tutorial then explores the fact that the quantification over arbitrary type predicates (expressed as constraints) gives the unrestricted set comprehension.

http://okmij.org/ftp/Computation/extra-polymorphism.html#class-quantification

3 Implementations

3.1 Haskell Platform

Report by: Duncan Coutts

Background

The Haskell Platform (HP) is the name of the "blessed" set of libraries and tools on which to build further Haskell libraries and applications. It takes a core selection of packages from the more than 4500 on Hackage (→ 6.3.1). It is intended to provide a comprehensive, stable, and quality tested base for Haskell projects to work from.

Historically, GHC shipped with a collection of packages. For the last few years the task of shipping an entire platform has been transferred to the Haskell Platform.

Recent progress

The latest major version of the platform was released in June and includes GHC-7.4.1 and updates to tools like cabal and many libraries.

Mark Lentczner has taken over as the platform release manager, and has been doing a great job coordinating between package maintainers, and volunteers building the platform for different operating systems. Mark presented a status update and future plans at the recent Haskell implementors workshop. Mark also led an effort to rationalise our use of the haskell.org server resources.

Looking forward

Major releases take place on a 6 month cycle. The next release is due soon in mid November. It will be based on GHC-7.4.2. It includes three new libraries:

- The `async` package by Simon Marlow provides a simple abstraction for running IO actions asynchronously. This is much like the fork/join or futures libraries that are becoming popular in other languages. There are many existing examples of this pattern in various Haskell code bases, since it is quite easy to implement, in terms of forkIO and MVars. This new package should help to standardise existing practice, and cover corner cases like asynchronous exceptions.

- The `split` package provides functions for splitting lists (usually strings) on delimeters. While Haskell has had functions like `words` and `lines` for years, the standard libraries have never provided other convenient splitting functions, partly because there are so many different use cases and combinations. The `split` package provides a collection of functions you can combine to get just the kind of split you want, plus convenience functions for a number of special cases.

- The `vector` package provides flat sequences. Unlike the existing `arrays` package, vectors are not multi-dimensional and always indexed from 0. But in turn vectors provide a much larger collection of useful functions (many of which only make sense for sequnces, not for multi-dimensional arrays). The library is carefully tuned for compact in-memory representations and speed.

The timing of the GHC-7.6 release was such that we took the decision to go with 7.4.2. It is likely that the following platform release in the spring will use the by-then stable 7.6.x version. Haskell users need to get used to the idea that there will always be a significant lag between the latest major GHC release and when that version makes it into the latest stable platform release. This is due to the time to test and update other packages, and to get important fixes incorporated back into a GHC bug-fix release.

Our systems for coordinating and testing new releases remains too time consuming, involving too much manual work. Help from the community on automation would be very valuable.

While we did get several new packages into the platform this release, there are still improvements we could make to keep the process running smoothly. Mark and the platform steering committee will be proposing some modifications to lower the barrier to entry. Nevertheless, we would still like to encourage package authors to propose new packages. This can be initiated at any time. We also invite the rest of the community to take part in the review process on the libraries mailing list ⟨libraries@haskell.org⟩. The current procedure involves writing a package proposal and discussing it on the mailing list with the aim of reaching a consensus. Details of the procedure are on the development wiki.

Further reading

http://haskell.org/haskellwiki/Haskell_Platform
- Download: http://haskell.org/platform/
- Wiki: http://trac.haskell.org/haskell-platform/
- Adding packages: http://trac.haskell.org/haskell-platform/wiki/AddingPackages

3.2 The Glasgow Haskell Compiler

Report by: Simon Marlow
Participants: many others

We made a bug-fix release of GHC 7.4.2 in June, and a completely new release of GHC 7.6 in August. As well as the usual raft of general improvements, GHC 7.6 included some new features:

- Multi-way if, and **case**.
- Kind polymorphism and data kinds [7].
- Deferred type errors [6].
- Improved support for generic programming (→ 7.4.2), [8].
- The ability to change at runtime the number of processors running Haskell threads.
- The first supported GHC for 64-bit Windows.
- Type-level literal symbols.

We expect to do a 7.6.2 release quite soon, and a 7.8.1 release in a few months' time.
Here is what we have been up to in the last six months:

Kind polymorphism and data kinds is a major new feature of GHC 7.6. It's described in "Giving Haskell a promotion" [7], and has already been used in interesting ways ("The Right Kind of Generic Programming" [8], "Dependently Typed Programming with Singletons" [9]). Leading up to the GHC 7.6 release Simon PJ has been working hard on making kind polymorphism work properly, which was a lot more work than he anticipated.

There is plenty more to do here, such as exploiting kind polymorphism to make a better *Typeable* class.

Type holes. Thijs Alkemade and Sean Leather have been working on another variant of deferred error messages, that would allow you to write a program that contains as-yet-unwritten sub-terms, or "holes" and have GHC report a fairly precise type for the hole. The HEAD now has an initial implementation (`-XTypeHoles`), and there are ongoing discussions about how to make it better still. Details on their wiki page [10].

Data parallelism. We are currently completely rewriting our implementation of vectorisation avoidance [1] in GHC's vectoriser. This leads to an overall much simpler and more robust vectoriser. In particular, it will be more liberal in allowing scalar subcomputations imported from modules compiled without vectorisation (such as the standard Prelude). This should finally enable us to get rid of the specialised, mini-Prelude in the DPH libraries.

After having solved the problem of obtaining asymptotically work-efficient vectorisation [2], we are now turning to improving the constants in the DPH libraries, and in particular, to achieve more reliable fusion in the presence of segmented operations, folds, and parallelism. Ben Lippmeier has a few exciting ideas on major improvements in that direction that we will discuss in more detail once we have conducted more experiments. We plan to finish the new vectorisation-avoidance infrastructure in time for GHC 7.8, but the new fusion system will likely not be ready in time for that release.

Moreover, Trevor McDonell has made good progress in devising a novel fusion system for the embedded Accelerate GPU language. We hope to be able to release it around the same time as GHC 7.8.

Overlapping type family instances. Richard Eisenberg is close to finishing an implementation of overlapping type family instances. The overlap mechanism is distinct from overlapping type class instances, as the programmer has to give an explicit ordering to the overlapping instances. More information can be found on the wiki page [11].

Dynamic libraries by default. In GHC 7.8, it will be possible to build GHC in such a way that by default it will dynamically link against Haskell libraries, rather than statically linking as it does now. As well as smaller binary sizes, this has the big advantage that GHCi will be able to use the system linker to load libraries, rather than our own linker implementation. This will mean fewer GHCi bugs, and make it a lot easier to add GHCi support to other platforms. We plan to make this the default way of building GHC on as many platforms as possible.

The new code generator. Several years since this project was started, the new code generator is finally working [14], and is now switched on by default in master. It will be in GHC 7.8.1. From a user's perspective there should be very little difference, though some programs will be faster.

There are three important improvements in the generated code. One is that let-no-escape functions are now compiled much more efficiently: a recursive let-no-escape now turns into a real loop in `C--`. The second improvement is that global registers (R1, R2, etc.) are now available for the register allocator to use within a function, provided they aren't in use for argument passing. This means that there are more registers available for complex code sequences. The third improvement is that we have a new sinking pass that replaces the old "mini-inliner" from the native code generator, and is capable of optimisations that the old pass couldn't do.

Hand-written `C--` code can now be written in a higher-level style with real function calls, and most of the hand-written C-- code in the RTS has been converted into the new style. High-level C-- does not mention global registers such as R1 explicitly, nor does it manipulate the stack; all this is handled by the C-- code generator in GHC. This is more robust and simpler, and means that we no longer need a special calling-convention for primops — they now use the same calling convention as ordinary Haskell functions.

We're interested in hearing about both performance improvements and regressions due to the new code generator.

Improved floating point register allocation. On x86-64 there are now six machine registers available for any mixture of floating-point types. Previously a maximum of four values of type Float and two values of type Double could simultaneously be kept in machine registers.

SIMD primitives. The simd branch now supports passing SSE vector values in machine registers. We expect the simd branch to be merged in time for 7.8.

Type-nat solver. Iavor S. Diatchki has been working on the type-checker to add support for discharging constraints involving arithmetic operations at the type-level. This work is on the type-nats branch of GHC. The basic support for common operations is fairly stable, and now it is in the testing phase. The most externally visible changes to the solver are: experimental support for matching on type-level naturals, using an auxiliary type family [12], and the module *GHC.TypeLits* was refactored to make it compatible with Richard Eisenberg's singletons library [13]. Next, we plan to work on integration with the master branch, and experimental support for the inverse operations of what's currently in the solver (i.e., (-), (/), Log, Root).

As always there is far more to do than we can handle, and there is loads of space for people to contribute. Do join us!

Bibliography

- [1] Vectorisation avoidance, Gabriele Keller et al., HS'12, http://www.cse.unsw.edu.au/~chak/papers/KCLLP12.html
- [2] Work-efficient higher-order vectorisation, Ben Lippmeier et al., ICFP'12, http://www.cse.unsw.edu.au/~chak/papers/LCKLP12.html
- [6] Equality proofs and deferred type errors, Dimitrios Vytiniotis et al., ICFP'12, http://research.microsoft.com/en-us/um/people/simonpj/papers/ext-f/
- [7] Giving Haskell a promotion, Brent Yorgey et al., TLDI'12, http://research.microsoft.com/en-us/um/people/simonpj/papers/ext-f/
- [8] The Right Kind of Generic Programming, José Pedro Magalhães, WGP'12, http://dreixel.net/research/pdf/trkgp.pdf
- [9] Dependently typed programming with singletons, Richard Eisenberg and Stephanie Weirich, HS'12, http://www.cis.upenn.edu/~eir/pubs.html
- [10] Holes in GHC: http://hackage.haskell.org/trac/ghc/wiki/Holes
- [11] Overlapping type family instances: http://hackage.haskell.org/trac/ghc/wiki/NewAxioms
- [12] Matching on type nats: http://hackage.haskell.org/trac/ghc/wiki/TypeNats/MatchingOnNats
- [13] Singletons and kinds: http://hackage.haskell.org/trac/ghc/wiki/TypeNats/SingletonsAndKinds
- [14] The new codegen is nearly ready to go live: http://hackage.haskell.org/trac/ghc/blog/newcg-update

3.3 UHC, Utrecht Haskell Compiler

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

What is new? UHC is the Utrecht Haskell Compiler, supporting almost all Haskell98 features and most of Haskell2010, plus experimental extensions. The current focus is on the Javascript backend.

What do we currently do and/or has recently been completed? As part of the UHC project, the following (student) projects and other activities are underway (in arbitrary order):

- (completed) Jurriën Stutterheim and others: building web applications with the Javascript backend. See the below UHC Javascript url for more info.
- (ongoing) Jeroen Bransen (PhD): “Incremental Global Analysis”.
- (ongoing) Jan Rochel (PhD): “Realising Optimal Sharing”, based on work by Vincent van Oostrum and Clemens Grabmayer.
- (ongoing) Atze Dijkstra: overall architecture, type system, bytecode interpreter + java + javascript backend, garbage collector.

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 (→ 5.3.3).

Further reading

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

3.4 Specific Platforms

3.4.1 Haskell on FreeBSD

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

The FreeBSD Haskell Team is a small group of contributors 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 i386 and amd64) starting from GHC 6.12.1, hence one can always download vanilla binary distributions for each recent release.

We have a developer repository for Haskell ports that features around 400 ports of many popular Cabal packages. The updates committed to this repository are continuously integrated to the official ports tree on a regular basis. However, the FreeBSD Ports Collection already includes many popular and important Haskell software: GHC 7.4.1, Haskell Platform 2012.2.0.0, Gtk2Hs, wxHaskell, XMonad, Pandoc, Gitit, Yesod, Happstack, Snap, Agda, git-annex, and so on — all of them will be available as part of the soon-to-be-published FreeBSD 9.1-RELEASE.

If you find yourself interested in helping us or simply want to use the latest versions of Haskell programs on FreeBSD, check out our page at the FreeBSD wiki (see below) where you can find all important pointers and information required for use, contact, or contribution.

Further reading

http://wiki.FreeBSD.org/Haskell

3.4.2 Debian Haskell Group

Report by:	Joachim Breitner
Status:	working

The Debian Haskell Group aims to provide an optimal Haskell experience to users of the Debian GNU/Linux distribution and derived distributions such as Ubuntu. We try to follow the Haskell Platform versions for the core package and package a wide range of other useful libraries and programs. At the time of writing, we maintain 500 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 stable Debian release ("squeeze") provides the Haskell Platform 2010.1.0.0 and GHC 6.12, Debian testing ("wheezy") and unstable ("sid") contain the Platform version 2012.3.0.0 with GHC 7.4.1. Debian wheezy is currently frozen, so no new uploads to testing and unstable are happening at the moment. We are however working on the infrastructure: Full support for running hoogle to search all installed Haskell documentation is in the making.

Debian users benefit from the Haskell ecosystem on 13 architecture/kernel combinations, including the non-Linux-ports KFreeBSD and Hurd.

Further reading

http://wiki.debian.org/Haskell

3.4.3 Haskell in Gentoo Linux

Report by:	Sergei Trofimovich

Gentoo Linux currently officially supports GHC 7.4.1, GHC 7.0.4 and GHC 6.12.3 on x86, amd64, sparc, alpha, ppc, ppc64 and some arm platforms.

The full list of packages available through the official repository can be viewed at http://packages.gentoo.org/category/dev-haskell?full_cat.

The GHC architecture/version matrix is available at http://packages.gentoo.org/package/dev-lang/ghc.

Please report problems in the normal Gentoo bug tracker at bugs.gentoo.org.

There is also an overlay which contains almost 800 extra unofficial and testing packages. Thanks to the Haskell developers using Cabal and Hackage (→ 6.3.1), we have been able to write a tool called "hackport" (initiated by Henning Günther) to generate Gentoo packages with minimal user intervention. Notable packages in the overlay include the latest version of the Haskell Platform (→ 3.1) as well as the latest 7.4.1 release of GHC, as well as popular Haskell packages such as pandoc, gitit, yesod (→ 5.2.6) and others.

As usual GHC 7.4 branch required some packages to be patched. For a 6 months period we have got about 150 patches waiting for upstream inclusion.

Over the time more and more people get involved in gentoo-haskell project which reflects positively on haskell ecosystem health status.

More information about the Gentoo Haskell Overlay can be found at http://haskell.org/haskellwiki/Gentoo.

It is available via the Gentoo overlay manager "layman". If you choose to use the overlay, then any problems should be reported on IRC (`#gentoo-haskell` on freenode), where we coordinate development, or via email ⟨haskell@gentoo.org⟩ (as we have more people with the ability to fix the overlay packages that are contactable in the IRC channel than via the bug tracker).

As always we are more than happy for (and in fact encourage) Gentoo users to get involved and help us maintain our tools and packages, even if it is as simple as reporting packages that do not always work or need updating: with such a wide range of GHC and package versions to co-ordinate, it is hard to keep up! Please contact us on IRC or email if you are interested!

For concrete tasks see our perpetual TODO list: https://github.com/gentoo-haskell/gentoo-haskell/blob/master/projects/doc/TODO.rst

3.4.4 Fedora Haskell SIG

Report by:	Jens Petersen
Participants:	Lakshmi Narasimhan, Shakthi Kannan, Michel Salim, Ben Boeckel, and others
Status:	ongoing

The Fedora Haskell SIG works on providing good Haskell support in the Fedora Project Linux distribution.

Fedora 18 will ship in December with ghc-7.4.1 and haskell-platform-2012.2.0.0, and version updates also to many other packages. New packages added since the release of Fedora 17 include cabal-rpm, happstack-server, hledger, and a bunch of libraries. Cabal-rpm has been revamped to replace the previously used cabal2spec packaging shell-script.

At the time of writing there are now 205 Haskell source packages in Fedora. The Fedora package version numbers listed on the Hackage website refer to the latest branched version of Fedora (currently 18).

Fedora 19 work is starting now with ghc-7.4.2, haskell-platform-2012.4 and plans finally to package up Yesod.

If you want to help with package reviews and Fedora Haskell packaging, please join us on Freenode irc #fedora-haskell and our low-traffic mailing-list, or follow @fedorahaskell.

Further reading

- Homepage: http://fedoraproject.org/wiki/SIGs/Haskell
- 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/f17-f18.compare

4 Related Languages and Language Design

4.1 Agda

Report by:	Nils Anders Danielsson
Participants:	Ulf Norell, Andreas Abel, 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.

The next version of Agda is still under development. Some of the changes were mentioned in the last HCAR entry. More recently Stevan Andjelkovic has contributed a LaTeX backend, with the aim to support both precise, Agda-style highlighting, and lhs2TeX-style alignment of code.

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.

Recently, I have added more comfortable syntax for data type declarations and let-definitions. Data and codata types can now also be defined recursively. In the long run, I plan to evolve MiniAgda into a core language for Agda with termination certificates.

MiniAgda is available as Haskell source code and compiles with GHC 6.12.x – 7.4.1.

Further reading

http://www2.tcs.ifi.lmu.de/~abel/miniagda/

4.3 Disciple

Report by:	Ben Lippmeier
Participants:	Tran Ma, Amos Robinson, Erik de Castro Lopo
Status:	experimental, active development

Disciple Core is an explicitly typed language based on System-F2, intended as an intermediate representation for a compiler. In addition to the polymorphism of System-F2 it supports region, effect and closure typing. Evaluation order is left-to-right call-by-value by default, but explicit lazy evaluation is also supported. The language includes a capability system to track whether objects are mutable or constant, and to ensure that computations that perform visible side effects are not suspended with lazy evaluation.

The Disciplined Disciple Compiler (DDC) is being rewritten to use the redesigned Disciple Core language. This new DDC is at a stage where it will parse and type-check core programs, and compile first-order functions over lists to executables via C or LLVM backends. There is also an interpreter that supports the full language.

What is new?

- Tran Ma has extended the core language with witnesses of Distinctness, which encode the fact that two regions of memory cannot alias at runtime. This information is used during program transformation in DDC, as well as being converted to LLVM aliasing metadata. Aliasing metadata allows the LLVM compiler to perform alias dependent follow-on optimisations, such as Global Value Numbering (GVN).

- Amos Robinson has added a rewrite rule system which understands the Disciple effect typing mechanism. Rewrite rules can be given constraints that ensure they only fire when particular expressions have non-interfering effects. This enables rewrite rule based transformations such as build/fold fusion to work in the presence of (non-interfering) effects.

- Ben Lippmeier has been working on the compiler framework and code generators. DDC now supports cross module inlining and a few basic code transformations like let-floating. All code that constructs heap objects is written directly in our lowest-level intermediate language, Disciple Salt (a bit like Cmm). This language is a fragment of the full Disciple Core language, so we can use the same AST right up until final code generation, either via C or LLVM. All runtime system code is written directly in Disciple Salt, and then inlined into user-written programs during compilation.

Future plans

We are currently fixing bugs in preparation for a release at the end of November.

Further reading

http://disciple.ouroborus.net

4.4 SugarHaskell

Report by:	Sebastian Erdweg
Participants:	Tillmann Rendel, Felix Rieger, Klaus Ostermann
Status:	active

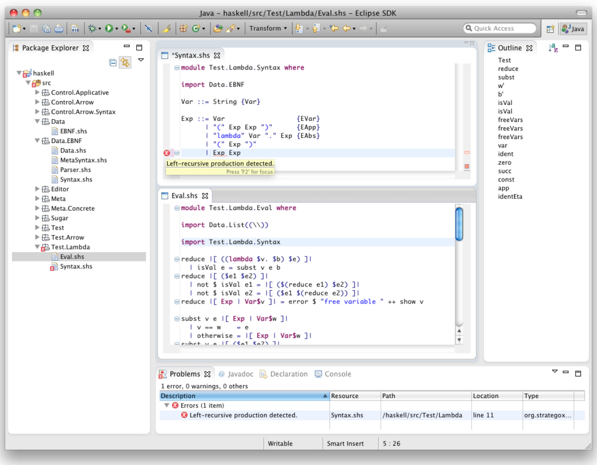
SugarHaskell is a generic extension of Haskell that enables programmers to define and use flexible syntactic extensions of Haskell. SugarHaskell extensions are organized as regular libraries, which define an extended syntax and a transformation of the extended syntax into Haskell's base syntax (or an extension thereof). To activate an extension, a SugarHaskell programmer simply imports the library that defines the extension; the extension is active in the remainder of the current file. Our Haskell Symposium paper [4] contains numerous examples, including arrow notation and, as illustrated in the following, idiom brackets:

import *Control.Applicative*
import *Control.Applicative.IdiomBrackets*

instance *Traversable Tree* **where**
  $traverse\ f\ Leaf \qquad = (\!|\ Leaf\ |\!)$
  $traverse\ f\ (Node\ l\ x\ r) =$
    $(\!|\ Node\ (traverse\ f\ l)\ (f\ x)\ (traverse\ f\ r)\ |\!)$

The library *Control.Applicative.IdiomBrackets* provides a syntactic extension for programming with applicatives, using idiomatic brackets (| ... |). Uses of idiom brackets are desugared in-place to produce plain Haskell code. Generally, the usage of syntactic extensions in a program is transparent to its clients.

SugarHaskell provides both a compiler and an Eclipse-based IDE. The SugarHaskell compiler is available as a Hackage package [2] and can be easily installed using cabal-install. Since our system is implemented in Java, the SugarHaskell package requires a preinstalled Java runtime. Moreover, we distribute the source code via github, and involvement of others is welcome. The SugarHaskell IDE is available as an Eclipse plugin and can be installed from our Eclipse update site [3]. The IDE provides some standard editor services such as code coloring or outlining for Haskell, and is also extensible itself to accommodate user-defined editor services for SugarHaskell extensions.



```
module Test.Lambda.Syntax where

import Data.EBNF

Var ::= String {Var}

Exp ::= Var                    {EVar}
      | "(" Exp Exp ")"        {EApp}
      | "lambda" Var "." Exp {EAbs}
      | "(" Exp ")"
      | Exp Exp
Left-recursive production detected.
                                Press 'F2' for focus
```

```
module Test.Lambda.Eval where

import Data.List((\\))

import Test.Lambda.Syntax

reduce |[ ((lambda $v. $b) $e) ]|
  | isVal e = subst v e b
reduce |[ ($e1 $e2) ]|
  | not $ isVal e1 = |[ ($(reduce e1) $e2) ]|
  | not $ isVal e2 = |[ ($e1 $(reduce e2)) ]|
reduce |[ Exp | Var$v ]| = error $ "free variable " ++ show v

subst v e |[ Exp | Var$w ]|
  | v == w     = e
  | otherwise = |[ Exp | Var$w ]|
```

```
1 error, 0 warnings, 0 others
Description                          Resource     Path                   Location   Type
Errors (1 item)
  Left-recursive production detected.  Syntax.shs   /haskell/src/Test/Lambda   line 11    org.strategox...
```

SugarHaskell is a research prototype that is under active development. We work both on the implementation and the conceptional foundation of the system. The feedback cycle is short and any feedback is appreciated.

Further reading

[1] http://sugarj.org
[2] http://hackage.haskell.org/package/sugarhaskell
[3] Eclipse update site: http://sugarj.org/update
[4] Sebastian Erdweg, Felix Rieger, Tillmann Rendel, and Klaus Ostermann. **Layout-sensitive Language Extensibility with SugarHaskell.** In *Haskell Symposium*, pages 149–160. ACM, 2012.

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, Oleg Lobachev, Rita Loogen, **in Copenhagen:** 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. The Eden logo

consists of four λ turned in such a way that they form the Eden instantiation operator (#). 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*. A *PA*-monad enables the *eager* execution of user defined sequences of *Parallel Actions* in Eden.

Web Pages

http://www.mathematik.uni-marburg.de/~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 libraries of parallel skeletons.

The current stable release of the Eden compiler is based on GHC 7.4.2. Binary packages and source code are available on our web pages, the Eden libraries (Haskell-level) are also available via Hackage.

A newer variant based on GHC-7.6.1 (and matching Eden libraries) are available as source code via git repositories at http://james.mathematik.uni-marburg.de:8080/gitweb. We plan the next full release of Eden with the next (minor or major) GHC release.

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.

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

- Oleg Lobachev: *Parallel Computation Skeletons with Premature Termination Property*, in Functional and Logic Programming (FLOPS) 2012, Springer LNCS 7294, pp. 197–212.
- P. Rabanal, I. Rodríguez, F. Rubio:, *A Functional Approach to Parallelize Particle Swarm Optimization*, Metaheurísticas, Algoritmos Evolutivos y Bioinspirados, MAEB'12, 2012.
- 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.

- Mischa Dieterle, Thomas Horstmeyer, Jost Berthold, Rita Loogen: *Iterating Skeletons — Structured Parallelism by Composition*, Draft Proceedings of the Symposium on the Implementation and Application of Functional Languages (IFL 2012), Technical Report RR–12–06, Oxford University, 2012.
- Thomas Horstmeyer and Rita Loogen: *Graph-Based Communication in Eden*, revised and extended version of TFP 2009 paper, submitted to Higher-Order Symbol Computation (HOSC), October 2012.

Further reading

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

5.1.2 GpH — Glasgow Parallel Haskell

Report by:	Hans-Wolfgang Loidl
Participants:	Phil Trinder, Patrick Maier, Mustafa Aswad, Malak Aljabri, Evgenij Belikov, Pantazis Deligianis, Robert Stewart, Prabhat Totoo (Heriot-Watt University); Kevin Hammond, Vladimir Janjic, Chris Brown (St Andrews University)
Status:	ongoing

Status

A distributed-memory, GHC-based implementation of the parallel Haskell extension GpH and of a fundamentally revised version of the evaluation strategies abstraction is available in a prototype version. In current research an extended set of primitives, supporting hierarchical architectures of parallel machines, and extensions of the runtime-system for supporting these architectures are being developed.

Main activities

We have been extending the set of primitives for parallelism in GpH, to provide enhanced control of data locality in GpH applications. Results from applications running on up to 256 cores of our Beowulf cluster demonstrate significant improvements in performance when using these extensions.

In the context of the SICSA MultiCore Challenge, we are comparing the performance of several parallel Haskell implementations (in GpH and Eden) with other functional implementations (F#, Scala and SAC) and with implementations produced by colleagues in a wide range of other parallel languages. The latest challenge application was the n-body problem. A summary of this effort is available on the following web page, and sources of several parallel versions will be uploaded shortly: http://www.macs.hw.ac.uk/sicsawiki/index.php/MultiCoreChallenge.

New work has been launched into the direction of inherently parallel data structures for Haskell and using such data structures in symbolic applications. This work aims to develop foundational building blocks in composing parallel Haskell applications, taking a data-centric point of view. Current work focuses on data structures such as append-trees to represent lists and quad-trees in an implementation of the n-body problem.

Another strand of development is the improvement of the GUM runtime-system to better deal with hierarchical and heterogeneous architectures, that are becoming increasingly important. We are revisiting basic resource policies, such as those for load distribution, and are exploring modifications that provide enhanced, adaptive behaviour for these target platforms.

GpH Applications

As part of the SCIEnce EU FP6 I3 project (026133) (April 2006 – December 2011) and the HPC-GAP project (October 2009 – September 2013) we use Eden, GpH and HdpH as middleware to provide access to computational Grids from Computer Algebra (CA) systems, in particular GAP. We have developed and released SymGrid-Par, a Haskell-side infrastructure for orchestrating heterogeneous computations across high-performance computational Grids. Based on this infrastructure we have developed a range of domain-specific parallel skeletons for parallelising representative symbolic computation applications. A Haskell-side interface to this infrastructures is available in the form of the Computer Algebra Shell CASH, which is downloadable from Hackage. We are currently extending SymGrid-Par with support for fault-tolerance, targeting massively parallel high-performance architectures.

Implementations

The latest GUM implementation of GpH is built on GHC 6.12, using either PVM or MPI as communications library. It implements a virtual shared memory abstraction over a collection of physically distributed machines. At the moment our main hardware platforms are Intel-based Beowulf clusters of multicores. We plan to connect several of these clusters into a wide-area, hierarchical, heterogenous parallel architecture.

Further reading

http://www.macs.hw.ac.uk/~dsg/gph/

Contact

⟨gph@macs.hw.ac.uk⟩

5.1.3 Parallel GHC project

Report by:	Duncan Coutts
Participants:	Duncan Coutts, Andres Löh, Mikolaj Konarski, Edsko de Vries
Status:	active

Microsoft Research funded a 2-year project, which is now coming to an end, to promote the real-world use of parallel Haskell. The project involved industrial partners working on their own tasks using parallel Haskell, and consulting and engineering support from Well-Typed (→ 8.1). The overall goal has been to demonstrate successful serious use of parallel Haskell, and along the way to apply engineering effort to any problems with the tools that the organisations might run into. In addition we have put significant engineering work into a new implementation of Cloud Haskell.

The participating organisations are working on a diverse set of complex real world problems:

- Dragonfly (New Zealand): Hierarchical Bayesian Modeling
- Los Alamos National Laboratory (USA): high performance Monte Carlo algorithms to model the flow of radiation and other physical phenomena
- IIJ Innovation Institute Inc. (Japan): network servers handling a massive number of concurrent connections
- Telefonica I+D: processing large graphs representing social networks

As the project winds down, we will be publishing more details about the outcomes of these projects.

On the engineering side, the two main areas of focus in the project recently have been ThreadScope and Cloud Haskell.

ThreadScope. The latest release of ThreadScope (version 0.2.2) provides detailed statistics about heap and GC behaviour. It is much like the output that can be obtained by running your program with `+RTS -s` but presented in a more friendly way and with the ability to see the same statistics for any period within the program, not just the entire program run. This work could be extended to show graphs of the heap size over time. Compared to GHC's traditional heap profiling this does not require recompiling in profiling mode and is very low overhead, but what is lost is the detailed breakdown of the heap by type, cost centre or retainer.

In addition there is a new feature to emit phase markers from user code and have these visualised in the ThreadScope timeline window.

These new features rely on the development version of GHC, and so will become generally available with GHC-7.8.

Finally, there is an alpha release of an ambitious new feature to integrate data from Linux's "perf" system into ThreadScope. The Linux "perf" system lets us see events in the OS such as system calls and other internal kernel trace points, and also to collect detailed CPU performance counters. Our work has focused on capturing and transforming this data source, and integrating it with the existing RTS event tracing system which we believe will enable many useful new visualisations. Our initial new visualisation in ThreadScope lets us see when system calls are occurring. We hope that this and other future work in this area will help developers who are trying to optimise the performance of applications like network servers.

Cloud Haskell. For about the last year we have been working on a new implementation of Cloud Haskell. This is the same idea for concurrent distributed programming in Haskell that Simon Peyton Jones has been telling everyone about, but it's a new implementation designed to be robust and flexible.

The summary about the new implementation is that it exists, it works, it's on hackage, and we think it is now ready for serious experiments.

Compared to the previous prototype:

- it is much faster;
- it can run on multiple kinds of network;
- has backends to support different environments (like cluster or cloud);
- has a new system for dealing with node disconnect and reconnect;
- has a more precisely defined semantics;
- supports composable, polymorphic serialisable closures;
- and internally the code is better structured and easier to work with.

By the time you read this, we will have also released a backend for the Windows Azure cloud platform. Backends for other environments should be relatively straightforward to develop.

Further details including papers, videos and blog posts are on the Cloud Haskell homepage.

Further reading

- Parallel GHC project homepage: http://www.haskell.org/haskellwiki/Parallel_GHC_Project
- Cloud Haskell homepage: http://www.haskell.org/haskellwiki/Cloud_Haskell
- ThreadScope homepage: http://www.haskell.org/haskellwiki/ThreadScope

5.1.4 Static Verification of Transactions in STM Haskell

Report by:	Romain Demeyer
Participants:	Wim Vanhoof
Status:	ongoing work

This PhD project targets the detection of concurrency bugs in STM Haskell. We focus on static analysis, i.e., we try to find errors by analyzing the source code of the program without executing it. Specifically, we target what we call *application-level* bugs, i.e., when the shared memory becomes inconsistent with respect to the design of the application because of an unexpected interleaving of the threads that access the memory. Our approach is to check that each transaction of the program preserves a given user-defined consistency property.

We have already defined, formalized and developed a framework of verification and, now, we try to evaluate which range of concurrency bugs we are able to detect. The ongoing work also includes the implementation of a prototype and the research in order to reduce the number of annotations the programmer has to provide for running the analysis.

Contact

Please feel free to contact me at rde@info.fundp.ac.be for further information.

5.2 Haskell and the Web

5.2.1 WAI

Report by:	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.

Since the last HCAR, WAI has switched to conduits (→ 7.1.1). WAI also added a vault parameter to the request type to allow middleware to store arbitrary data.

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.

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) and dingo (GUI). Since the last HCAR, another web framework called Scotty was released. 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.

The WAI standard has proven itself capable for different users and there are no outstanding plans for changes or improvements.

Further reading

http://www.yesodweb.com/book/wai

5.2.2 Warp

Report by:	Greg Weber

Warp is a high performance, easy to deploy HTTP server backend for WAI (→ 5.2.1). Since the last HCAR, Warp has switched from enumerators to conduits (→ 7.1.1), added SSL support, and websockets integration.

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

5.2.3 Holumbus Search Engine Framework

Report by:	Uwe Schmidt
Participants:	Timo B. Kranz, Sebastian Gauck, Stefan Schmidt
Status:	first release

Description

The Holumbus framework consists of a set of modules and tools for creating fast, flexible, and highly customizable search engines with Haskell. The framework consists of two main parts. The first part is the indexer for extracting the data of a given type of documents, e.g., documents of a web site, and store it in an appropriate index. The second part is the search engine for querying the index.

An instance of the Holumbus framework is the Haskell API search engine Hayoo! (http://holumbus.fh-wedel.de/hayoo/).

The framework supports distributed computations for building indexes and searching indexes. This is done with a MapReduce like framework. The MapReduce framework is independent of the index- and search-components, so it can be used to develop distributed systems with Haskell.

The framework is now separated into four packages, all available on Hackage.

- The Holumbus Search Engine
- The Holumbus Distribution Library
- The Holumbus Storage System
- The Holumbus MapReduce Framework

The search engine package includes the indexer and search modules, the MapReduce package bundles the distributed MapReduce system. This is based on two other packages, which may be useful for their on: The Distributed Library with a message passing communication layer and a distributed storage system.

Features

- Highly configurable crawler module for flexible indexing of structured data
- Customizable index structure for an effective search
- *find as you type* search
- Suggestions
- Fuzzy queries
- Customizable result ranking
- Index structure designed for distributed search
- Git repository containing the current development version of all packages under https://github.com/fortytools/holumbus
- Distributed building of search indexes

Current Work

Currently there are activities to optimize the index structures of the framework. In the past there have been problems with the space requirements during indexing. The data structures and evaluation strategies have been optimized to prevent space leaks. A second index structure working with cryptographic keys for document identifiers is under construction. This will further simplify partial indexing and merging of indexes.

There is a small project extracting the sources of the data structure used for the index to build a separate package. The search tree used in Holumbus is a space optimised version of a radix tree, which enables fast prefix and fuzzy search.

The second project, a specialized search engine for the FH-Wedel web site, has been finished http://w3w.fh-wedel.de/. The new aspect in this application is a specialized free text search for appointments, deadlines, announcements, meetings and other dates.

The Hayoo! and the FH-Wedel search engine have been adopted to run on top of the Snap framework (→ 5.2.7).

Further reading

The Holumbus web page (http://holumbus.fh-wedel.de/) includes downloads, Git web interface, current status, requirements, and documentation. Timo Kranz's master thesis describing the Holumbus index structure and the search engine is available at http://holumbus.fh-wedel.de/branches/develop/doc/thesis-searching.pdf. Sebastian Gauck's thesis dealing with the crawler component is available at http://holumbus.fh-wedel.de/src/doc/thesis-indexing.pdf The thesis of Stefan Schmidt describing the Holumbus MapReduce is available via http://holumbus.fh-wedel.de/src/doc/thesis-mapreduce.pdf.

5.2.4 Happstack

Report by:	Jeremy Shaw

Happstack is a fast, modern framework for creating web applications. Happstack is well suited for MVC and RESTful development practices. We aim to leverage the unique characteristics of Haskell to create a highly-scalable, robust, and expressive web framework.

Happstack pioneered type-safe Haskell web programming, with the creation of technologies including web-routes (type-safe URLS) and acid-state (native Haskell database system). We also extended the concepts behind formlets, a type-safe form generation and processing library, to allow the separation of the presentation and validation layers.

Some of Happstack's unique advantages include:

- a large collection of flexible, modular, and well documented libraries which allow the developer to choose the solution that best fits their needs for databases, templating, routing, etc.
- the most flexible and powerful system for defining type-safe URLs.
- a type-safe form generation and validation library which allows the separation of validation and presentation without sacrificing type-safety
- a powerful, compile-time HTML templating system, which allows the use of XML syntax

A recent addition to the Happstack family is the `happstack-foundation` library. It combines what we believe to be the best choices into a nicely integrated solution. `happstack-foundation` uses:

- happstack-server for low-level HTTP functionality
- acid-state for type-safe database functionality
- web-routes for type-safe URL routing
- reform for type-safe form generation and processing
- HSP for compile-time, XML-based HTML templates
- JMacro for compile-time Javascript generation and syntax checking

Future plans

Happstack is the oldest, actively developed Haskell web framework. We are continually studying and applying new ideas to keep Happstack fresh. By the time the next release is complete, we expect very little of the original code will remain. If you have not looked at Happstack in a while, we encourage you to come take a fresh look at what we have done.

Some of the projects we are currently working on include:

- a fast pipes-based HTTP and websockets backend with a high level of evidence for correctness
- a dynamic plugin loading system
- a more expressive system for weakly typed URL routing combinators
- a new system for processing form data which allows fine grained enforcement of RAM and disk quotas and avoids the use of temporary files
- a major refactoring of HSP (fewer packages, migration to Text/Builder, a QuasiQuoter, and more).

One focus of Happstack development is to create independent libraries that can be easily reused. For example, the core web-routes and reform libraries are in no way Happstack specific and can be used with other Haskell web frameworks. Additionally, libraries that used to be bundled with Happstack, such as IxSet, SafeCopy, and acid-state, are now independent libraries. The new backend will also be available as an independent library.

When possible, we prefer to contribute to existing libraries rather than reinvent the wheel. For example, our preferred templating library, HSP, was created by and is still maintained by Niklas Broberg. However, a significant portion of HSP development in the recent years has been fueled by the Happstack team.

We are also working directly with the Fay team to bring an improved type-safety to client-side web programming. In addition to the new happstack-fay integration library, we are also contributing directly to Fay itself.

For more information check out the happstack.com website — especially the “Happstack Philosophy” and “Happstack 8 Roadmap”.

Further reading

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

5.2.5 Mighttpd2 — Yet another Web Server

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

Mighttpd (called mighty) version 2 is a simple but practical Web server in Haskell. It is now working on Mew.org serving static files, CGI (mailman and contents search) and reverse proxy for back-end Yesod applications.

Mighttpd is based on Warp providing performance on par with `nginx`. You can use the `mightyctl` command to reload configuration files dynamically and shutdown Mighttpd gracefully.

You can install Mighttpd 2 (*mighttpd2*) from HackageDB.

Further reading

- http://www.mew.org/~kazu/proj/mighttpd/en/
- http://www.iij.ad.jp/en/company/development/tech/mighttpd/
- http://www.yesodweb.com/blog/2012/10/future-work-warp

5.2.6 Yesod

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

Yesod is a traditional MVC RESTful framework. By applying Haskell’s strengths to this paradigm, we have created a web framework that 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 (→ 7.7.2) — 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 (→ 7.7.2). View 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. Their return type shows which response types are allowed for the request.

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.

Yesod finally reached its 1.0 version. The last HCAR entry was for the 0.8 version. Some of the major changes since then are:

- Luite Stegemen contributed a faster and improved development environment that used the GHC API
- Nubis Bruno contributed yesod-test, a convenient testing framework.
- Flexible session interface
- Flexible placement of Javascript on the HTML page
- Switch from enumerators to conduits

We are excited to have achieved a 1.0 release. This signifies maturity and API stability and a web framework that gives developers all the tools they need for productive web development. Future directions for Yesod are now largely driven by community input and patches. Easier client-side interaction is definitely one concern that Yesod is working on going forward. The 1.0 release features better coffeescript support and even roy.js support

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

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

Further reading

http://www.yesodweb.com/

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, 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.

The Snap Framework has seen one major release (0.9) since the last HCAR. Some of the major features added are support for choosing different configurations based on user-specified execution environments (such as production or development), an improved project template demonstrating use of the session and auth snaplets, new functions allowing you to retrieve socket information for a running server, and of course a number of other bug fixes and minor features.

Another piece of exciting news is that we recently received funding for paid Snap development. We're using it to get another paid developer working with the core Snap team to write better and more comprehensive documentation and help with some specific implementation tasks.

Since the last HCAR we have done a LOT of behind-the-scenes work on some big improvements that will be coming out in upcoming releases. In keeping with our tradition, we're taking our time with these features to make sure they measure up to the high quality that Snap users have come to expect. When these features are finished Snap will have more than two and a half years of development, and we think it will be worthy of a 1.0 release.

Further reading

- Snaplet Directory: http://snapframework.com/snaplets
- http://snapframework.com

5.3 Haskell and Compiler Writing

5.3.1 MateVM

Report by:	Bernhard Urban
Participants:	Harald Steinlechner
Status:	active development

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 proccessing 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 think that Haskell is perfectly suitable for compiler challenges, as already well known. However, we have to jump between "Haskell world" and "native code world", due to the requirements of a Just-In-Time Compiler. This poses some special challenges when it comes to signal handling and other interesing 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.

While we are currently able to execute simple Java programs, many features are missing for a full JavaVM, most noteable are Classloader, Floating Point or Threads. We would like to use GNU Classpath as base library some day. Other hot topics are Hoopl and Garbage Collection at the moment. In the long-run, we would like to implement features known from adaptive compilation, e.g. method inlining or stack allocation of objects.

If you are interested in this project, do not hestiate 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 CoCoCo

Report by:	Marcos Viera
Participants:	Doaitse Swierstra, Arthur Baars, Arie Middelkoop, Atze Dijkstra, Wouter Swierstra
Status:	experimental

CoCoCo (Compositional Compiler Construction) is a set of libraries and tools in the form of a collection of embedded domain specific languages (EDSL) in Haskell for constructing extensible compilers, where compilers can be composed out of separately compiled and statically type checked language-definition fragments.

Our approach builds on:

- the introduction of a naming structure which makes it possible to represent mutually dependent structures and the possibility to inspect and manipulate such structures in a type-safe way
- the description of typed grammar fragments as first class Haskell values, and the typed Left-Corner Transform to remove left-recursion
- the possibility to construct a self-analysing, error correcting parser on the fly
- the possibility to deal with attribute grammars as first class Haskell values, which can be transformed, composed and finally evaluated.

As a case study we have implemented an Oberon0 compiler, which is available as a Hackage package:

- http://hackage.haskell.org/package/oberon0.

Its implementation is described in a technical report:

- Viera, M., Swierstra, S.D.: Compositional Compilers Construction: Oberon0. UU-CS 2012-016, Institute of Information and Computing Science (October 2012).

Related Libraries

- **murder:** The murder library is an EDSL for grammar fragments as first-class values. It provides combinators to define and extend grammars, and produce compilers out of them.
  http://hackage.haskell.org/package/murder
- **AspectAG:** Library of strongly typed Attribute Grammars implemented using type-level programming.
  http://hackage.haskell.org/package/AspectAG
- **TTTAS:** Library for Typed Transformations of Typed Abstract Syntax.
  http://hackage.haskell.org/package/TTTAS
- **uulib:** Fast Parser Combinators and Pretty Printing Combinators .
  http://hackage.haskell.org/package/uulib
- **uu-parsinglib:** New version of the Utrecht University parser combinator library, which provides online, error correction, annotation free, applicative style parser combinators.
  http://hackage.haskell.org/package/uu-parsinglib

Further reading

http://www.cs.uu.nl/wiki/Center/CoCoCo

5.3.3 UUAG

Report by:	Jeroen Bransen
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 (→ 3.3), 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 0.9.42.1 (November 2012), is extensively tested, and is available on Hackage. There is also a Cabal plugin for easy use of AG files in Haskell projects. Recently, we have improved the building procedure to make sure that the UUAGC can both be built from source as well as from the included generated Haskell sources, without the need of an external bootstrap program. Also, we added code generation for Ocaml.

We are working on the following enhancements of the UUAG system:

First-class AGs. We provide a translation from UUAG to AspectAG (→ 5.3.4). AspectAG is a library of strongly typed Attribute Grammars implemented using type-level programming. With this extension, we can write the main part of an AG conveniently with UUAG, and use AspectAG for (dynamic) extensions. Our goal is to have an extensible version of the UHC.

Ordered evaluation. We have implemented a variant of Kennedy and Warren (1976) for *ordered* AGs. For any absolutely non-circular AGs, this algorithm finds a static evaluation order, which solves some of the problems we had with an earlier approach for ordered AGs. A static evaluation order allows the generated code to be strict, which is important to reduce the memory usage when dealing with large ASTs. The generated code is purely functional, does not require type annotations for local attributes, and the Haskell compiler proves that the static evaluation order is correct.

Incremental evaluation. We are currently also running a Ph.D. project that investigates incremental evaluation of AGs. In this ongoing work we hope 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.

Further reading

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

5.3.4 AspectAG

Report by:	Marcos Viera
Participants:	Doaitse Swierstra, Wouter Swierstra
Status:	experimental

See: http://www.haskell.org/communities/11-2011/html/report.html#sect5.4.2.

5.3.5 LQPL — A Quantum Programming Language Compiler and Emulator

Report by:	Brett G. Giles
Participants:	Dr. J.R.B. Cockett and Rajika Kumarasiri
Status:	v 0.9.0 experimental released in July 2012

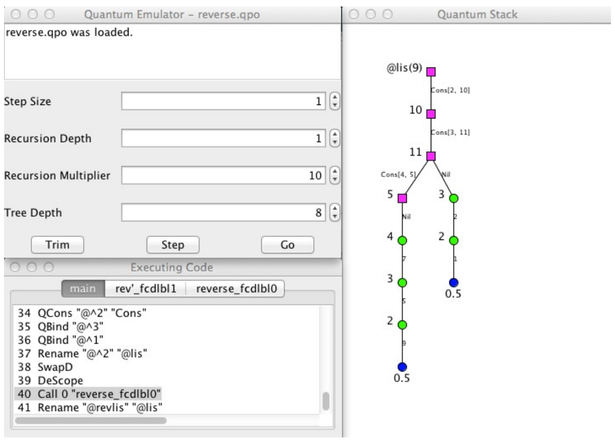
LQPL (Linear Quantum Programming Language) is a functional quantum programming language inspired by Peter Selinger's paper "Towards a Quantum Programming Language".

The LQPL system consists of a compiler, a GUI based front end and an emulator. Compiled programs are loaded to the emulator by the front end. LQPL incorporates a simple module / include system (more like C's include than Haskell's import), predefined unitary transforms, quantum control and classical control, algebraic data types, and operations on purely classical data.

The largest difference since the previous release of the package is that LQPL is now split into separate components. These consist of:

- The compiler (written in Haskell) — available at the command line and via a TCP/IP interface.
- The emulator (written in Haskell) — available as a server via a TCP/IP interface.
- The front end (Java/Swing)— with version 0.9, the front end was rewritten as a Java/Swing application, which connects to both the compiler and the emulator via TCP/IP. A text based / command line interface is being considered.

A screenshot of the new interface (showing a probabilistic list) is included below.



```
reverse.qpo was loaded.
```

Step Size: 1
Recursion Depth: 1
Recursion Multiplier: 10
Tree Depth: 8

```
34 QCons "@^2" "Cons"
35 QBind "@^3"
36 QBind "@^1"
37 Rename "@^2" "@lis"
38 SwapD
39 DeScope
40 Call 0 "reverse_fcdlbl0"
41 Rename "@revlis" "@lis"
```

Quantum programming allows us to provide a fair coin toss, as shown in the code example below.

```
qdata Coin      = {Heads | Tails},
toss ::( ; c:Coin) =,
{  q = |0>;     Had q;,
   measure q of ,
     |0> => {c = Heads},
     |1> => {c = Tails},
},
```

This allows programming of probabilistic algorithms, such as leader election.

Separation into modules was a preparatory step for improving the performance of the emulator and adding optimization features to the language.

Further reading

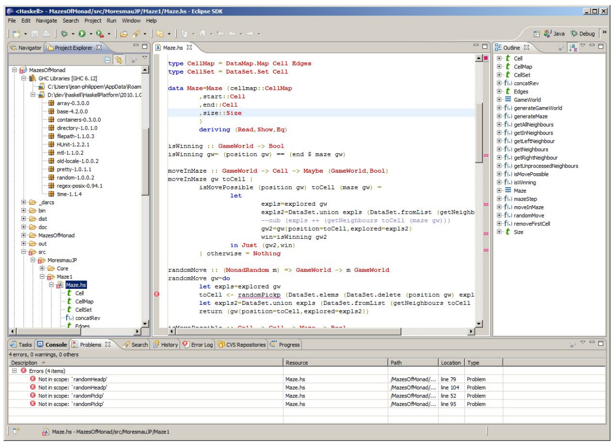
Documentation and executable downloads may be found at http://pll.cpsc.ucalgary.ca/lqpl/index.html. The source code, along with a wiki and bug tracker, is available at https://bitbucket.org/BrettGilesUofC/lqpl.

6 Development Tools

6.1 Environments

6.1.1 EclipseFP

Report by:	JP Moresmau
Participants:	building on code from B. Scott Michel, Alejandro Serrano, Thiago Arrais, Leif Frenzel, Thomas ten Cate, Martijn Schrage, Adam Foltzer and others
Status:	stable, maintained, and actively developed



```
type CellMap = DataMap.Map Cell Edges
type CellSet = DataSet.Set Cell

data Maze=Maze {cellmap::CellMap
        ,start::Cell
        ,end::Cell
        ,size::Size
        }
        deriving (Read,Show,Eq)

isWinning :: GameWorld -> Bool
isWinning gw= (position gw) == (end $ maze gw)

moveInMaze :: GameWorld -> Cell -> Maybe (GameWorld,Bool)
moveInMaze gw toCell |
        isMovePossible (position gw) toCell (maze gw) =
                let
                        expls=explored gw
                        expls2=DataSet.union expls (DataSet.fromList (getNeighb
                        --nub (expls ++ (getNeighbours toCell (maze gw)))
                        gw2=gw{position=toCell,explored=expls2}
                        win=isWinning gw2
                in Just (gw2,win)
        | otherwise = Nothing

randomMove :: (MonadRandom m) => GameWorld -> m GameWorld
randomMove gw=do
        let expls=explored gw
        toCell <- randomPickp (DataSet.elems (DataSet.delete (position gw) expl
        let expls2=DataSet.union expls (DataSet.fromList (getNeighbours toCell
        return (gw{position=toCell,explored=expls2})
```

4 errors, 0 warnings, 0 others

Description	Resource	Path	Location	Type
Errors (4 items)				
Not in scope: `randomHeadp'	Maze.hs	/MazesOfMonad/...	line 79	Problem
Not in scope: `randomHeadp'	Maze.hs	/MazesOfMonad/...	line 104	Problem
Not in scope: `randomPickp'	Maze.hs	/MazesOfMonad/...	line 52	Problem
Not in scope: `randomPickp'	Maze.hs	/MazesOfMonad/...	line 95	Problem

Maze.hs - MazesOfMonad/src/MoresmauJP/Maze1

EclipseFP is a set of Eclipse plugins to allow working on Haskell code projects. Its goal is to offer a fully featured Haskell IDE in a platform developers coming from other languages may already be familiar with. It features Cabal integration (.cabal file editor, uses Cabal settings for compilation, allows the user to install Cabal packages from within the IDE), and GHC integration. Compilation is done via the GHC API, syntax coloring uses the GHC Lexer. Other standard Eclipse features like code outline, folding, and quick fixes for common errors are also provided. HLint suggestions can be applied in one click. EclipseFP also allows launching GHCi sessions on any module including extensive debugging facilities: the management of breakpoints and the evaluation of variables and expressions uses the Eclipse debugging framework, and requires no knowledge of GHCi syntax. It uses the BuildWrapper Haskell tool to bridge between the Java code for Eclipse and the Haskell APIs. It also provides a full package and module browser to navigate the Haskell packages installed on your system, integrated with Hackage. The source code is fully open source (Eclipse License) on github and anyone can contribute. Current version is 2.3.2, released in October 2012 and supporting GHC 7.0 and above, and more versions with additional features are planned and actively worked on. Feedback on what is needed is welcome! The website has information on downloading binary releases and getting a copy of the source code. Support and bug tracking is handled through Sourceforge forums.

Further reading

http://eclipsefp.github.com/

6.1.2 ghc-mod — Happy Haskell Programming

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

`ghc-mod` is a backend command to enrich Haskell programming on editors including Emacs and Vim. The ghc-mod package on Hackage includes the `ghc-mod` command and Emacs front-end.

Emacs front-end provides the following features:

Completion. You can complete a name of keyword, module, class, function, types, language extensions, etc.

Code template. You can insert a code template according to the position of the cursor. For instance, "module Foo where" is inserted in the beginning of a buffer.

Syntax check. Code lines with error messages are automatically highlighted thanks to flymake. You can display the error message of the current line in another window. `hlint` can be used instead of GHC to check Haskell syntax.

Document browsing. You can browse the module document of the current line either locally or on Hackage.

Expression type. You can display the type/information of the expression on the cursor.

There are two Vim plugins:

- ghcmod-vim
- syntastic

`ghc-mod` now supports GHC 7.6.

Further reading

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

6.1.3 HEAT: The Haskell Educational Advancement Tool

Report by:	Olaf Chitil
Status:	active

Heat is an interactive development environment (IDE) for learning and teaching Haskell. Heat was designed for novice students learning the functional programming language Haskell. Heat provides a small number of supporting features and is easy to use. Heat is distributed as a single, portable Java jar-file and works on top of GHCi.

Heat provides the following features:

- Editor for a single module with syntax-highlighting and matching brackets.
- Shows the status of compilation: non-compiled; compiled with or without error.
- Interpreter console that highlights the prompt and error messages.
- If compilation yields an error, then the relevant source line is highlighted and no further expression can be evaluated in the console until the source has been changed and successfully recompiled.
- A tree structure provides a program summary, giving definitions of types and types of functions.
- Automatic checking of either Boolean or QuickCheck properties of a program; results shown in summary.

Further reading

http://www.cs.kent.ac.uk/projects/heat/

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, Darcs 2.8.2, was in September 2012. Some key changes in Darcs 2.8 include a faster and more readable `darcs annotate`, a `darcs obliterate -O` which can be used to conveniently "stash" patches, and hunk editing for the `darcs revert` command.

We have some fairly exciting work merged into mainline Darcs for the next release. First, we have a new rebase feature which should be a great help for darcs users that maintain long-term and conflict prone branches. The new rebase feature will also be useful for some more day to day use cases such as the "deep amend-record" feature many of us have long wished for, or even just more basic patch cleanups and reorganisation. Second, the 2012 Google Summer of Code project by BSRK Aditya has been successful, with the long-promised patch index optimisation now merged into mainline. The patch index will help Darcs users who need to search for changes to specific files within a large number of patches (particularly relevant to darcs hosting sites).

More generally, our work has emphasised two of our key priorities: code quality and Darcs hosting. For code quality we have embarked on an overhaul of our module organisation along with work towards deeper refactors such as abstracting over the use of IO to better capture some of our darcs-specific state.

Darcs hosting is also a hot area in Darcs development. Simon Michael has pushed forward development of the original Darcsden code by Alex Suraci, resulting in the recent darcsden 1.0 release (September 2012) and new public host http://hub.darcs.net. Feedback and help pushing forward this new Darcs hosting option will be greatly appreciated!

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://wiki.darcs.net/Development/Priorities

6.2.2 DarcsWatch

Report by:	Joachim Breitner
Status:	working

DarcsWatch is a tool to track the state of Darcs (→ 6.2.1) patches that have been submitted to some project, usually by using the `darcs send` command. It allows both submitters and project maintainers to get an overview of patches that have been submitted but not yet applied.

DarcsWatch continues to be used by the xmonad project (→ 7.8.2), the Darcs project itself, and a few developers. At the time of writing (November 2012), it was tracking 39 repositories and 4552 patches submitted by 238 users.

Further reading

- http://darcswatch.nomeata.de/
- http://darcs.nomeata.de/darcswatch/documentation.html

6.2.3 `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`, `cabal`, and `cabal-dev`.

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 provides the "ghci" subcommands.

Further reading

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

6.3 Deployment

6.3.1 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.

Recent progress

The Cabal packaging system has always faced growing pains. We have been through several cycles where we've faced chronic problems, made major improvements which bought us a year or two's breathing space while package authors and users become ever more ambitious and start to bump up against the limits again. In the last few years we have gone from a situation where 10 dependencies might be considered a lot, to a situation now where the major web frameworks have a 100+ dependencies and we are again facing chronic problems.

The Cabal/Hackage maintainers and contributors have been pursuing a number of projects to address these problems:

The IHG sponsored Well-Typed (→ 8.1) to work on cabal-install resulting in a new package dependency constraint solver. This was incorporated into the cabal-install-0.14 release in the spring, and which is now in the latest Haskell Platform release. The new dependency solver does a much better job of finding install plans. In addition the cabal-install tool now warns when installing new packages would break existing packages, which is a useful partial solution to the problem of breaking packages.

We had two Google Summer of Code projects on Cabal this year, focusing on solutions to other aspects of our current problems. The first is a project by Mikhail Glushenkov (and supervised by Johan Tibell) to incorporate sandboxing into cabal-install. In this context sandboxing means that we can have independent sets of installed packages for different projects. This goes a long way towards alleviating the problem of different projects needing incompatible versions of common dependencies. There are several existing tools, most notably `cabal-dev`, that provide some sandboxing facility. Mikhail's project was to take some of the experience from these existing tools (most of which are implemented as wrappers around the cabal-install program) and to implement the same general idea, but properly integrated into cabal-install itself. We expect the results of this project will be incorporated into a cabal-install release within the next few months.

The other Google Summer of Code project this year, by Philipp Schuster (and supervised by Andres Löh), is also aimed at the same problem: that of different packages needing inconsistent versions of the same common dependencies, or equivalently the current problem that installing new packages can break existing installed packages. The solution is to take ideas from the Nix package manager for a persistent non-destructive package store. In particular it lifts an obscure-sounding but critical limitation: that of being able to install multiple instances of the same version of a package, built against different versions of their dependencies. This is a big long-term project. We have been making steps towards it for several years now. Philipp's project has made another big step, but there's still more work before it is ready to incorporate into ghc, ghc-pkg and cabal.

Looking forward

Johan Tibell and Bryan O'Sullivan have volunteered as new release managers for Cabal. Bryan moved all the tickets from our previous trac instance into github, allowing us to move all the code to github. Johan managed the latest release and has been helping with managing the inflow of patches. Our hope is that these changes will increase the amount of contributions and give us more maintainer time for reviewing and integrating those contributions. Initial indications are positive. Now is a good time to get involved.

The IHG is currently sponsoring Well-Typed to work on getting the new Hackage server ready for switchover, and helping to make the switchover actually happen. We have recruited a few volunteer administrators for the new site. The remaining work is mundane but important tasks like making sure all the old data can be imported, and making sure the data backup system is comprehensive. Initially the new site will have just a few extra features compared to the old one. Once we get past the deployment hurdle we hope to start getting more contributions for new features. The code is structured so that features can be developed relatively independently, and we intend to follow Cabal and move the code to github.

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.

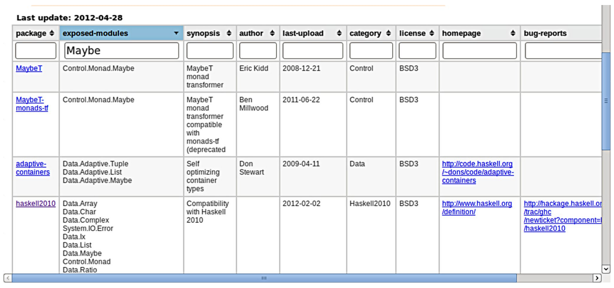
Further reading

- Cabal homepage: http://www.haskell.org/cabal
- Hackage package collection: http://hackage.haskell.org/
- Bug tracker: https://github.com/haskell/cabal/

6.3.2 Portackage — A Hackage Portal

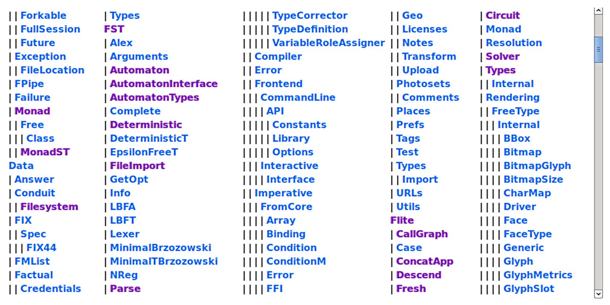
Report by:	Andrew G. Seniuk

Portackage (fremissant.net/portackage) is a web interface to all of hackage.haskell.org, which at the time of writing includes some 4000 packages exposing over 17000 modules. There are package and module views, as seen in the screenshots.



Last update: 2012-04-28

package	exposed-modules	synopsis	author	last-upload	category	license	homepage	bug-reports
	Maybe							
MaybeT	Control.Monad.Maybe	MaybeT monad transformer	Eric Kidd	2008-12-21	Control	BSD3		
MaybeT-monads-tf	Control.Monad.Maybe	MaybeT monad transformer compatible with monads-tf (deprecated	Ben Millwood	2011-06-22	Control	BSD3		
adaptive-containers	Data.Adaptive.Tuple Data.Adaptive.List Data.Adaptive.Maybe	Self optimizing container types	Don Stewart	2009-04-11	Data	BSD3	http://code.haskell.org/~dons/code/adaptive-containers	
haskell2010	Data.Array Data.Char Data.Complex System.IO.Error Data.Ix Data.List Data.Maybe Control.Monad Data.Ratio	Compatibility with Haskell 2010		2012-02-02	Haskell2010	BSD3	http://www.haskell.org/definition/	http://hackage.haskell.or/trac/ghc/newticket?component=l/haskell2010



```
| | Forkable        | Types                 | | | | | TypeCorrector            | | Geo         | Circuit
| | FullSession     FST                     | | | | | TypeDefinition           | | Licenses    | Monad
| | Future          | Alex                  | | | | | VariableRoleAssigner     | | Notes       | Resolution
| Exception         | Arguments             | | Compiler                   | | Transform   | Solver
| | FileLocation    | Automaton             | | Error                      | | Upload      | Types
| FPipe             | AutomatonInterface    | | Frontend                   | Photosets     | | Internal
| Failure           | AutomatonTypes        | | | CommandLine              | | Comments    | Rendering
| Monad             | Complete              | | | | API                    | Places        | | FreeType
| | Free            | Deterministic         | | | | | Constants              | Prefs         | | | Internal
| | | Class         | DeterministicT        | | | | | Library                | Tags          | | | | BBox
| | MonadST         | EpsilonFreeT          | | | | | Options                | Test          | | | | Bitmap
Data                | FileImport            | | | Interactive              | Types         | | | | BitmapGlyph
| Answer            | GetOpt                | | | | Interface              | | Import      | | | | BitmapSize
| Conduit           | Info                  | | Imperative                 | URLs          | | | | CharMap
| | Filesystem      | LBFA                  | | | FromCore                 | Utils         | | | | Driver
| FIX               | LBFT                  | | | | Array                  Flite           | | | | Face
| | Spec            | Lexer                 | | | | Binding                | CallGraph     | | | | FaceType
| | | FIX44         | MinimalBrzozowski     | | | | Condition              | Case          | | | | Generic
| FMList            | MinimalTBrzozowski    | | | | ConditionM             | ConcatApp     | | | | Glyph
| Factual           | NReg                  | | | | Error                  | Descend       | | | | GlyphMetrics
| | Credentials     | Parse                 | | | | FFI                    | Fresh         | | | | GlyphSlot
```

The package view includes links to the package, homepage, and bug tracker when available. Each name in the module tree view links to the Haddock API page. Control-hovering will show the fully-qualified name in a tooltip.

Portackage is only a few days old; imminent further work includes

- Tree branches will be collapsed by default.
- Cookies (as well as server DB) will maintain persistent state of which nodes you have open, since this information carries value, both in terms of cost to reconstruct manually, and of personal mnemonics — if nodes were collapsed, you would forget where things were, instead of having them right there filtered out.
- A flat list of modules with the filtering text input field would be good, but the full list of modules is too large for the present naïve JavaScript.

The code itself is mostly Haskell, but is still too green to expose on Hackage.

6.4 Others

6.4.1 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 1.18 and has been released in September 2012. The main change is compatibility with GHC 7.6. Development repository and bug tracker are on GitHub. There are still plans for a rewrite of lhs2TEX with the goal of 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

6.4.2 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.

Tracing a program with Hat consists of two phases: First the program needs to be run such that it additionally writes a trace to file. To add trace-writing, *hat-trans* translates all the source modules *Module* of a Haskell program into tracing versions *Hat.Module*. These are compiled as normal and when run the program does exactly the same as the original program except for additionally writing a trace to file. Second, after the program has terminated, you view the trace with a browsing tool. Hat comes with several tools to selectively view 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. Currently Hat supports Haskell 98 plus some language extensions such as multi-parameter type classes and functional dependencies. For portability all viewing tools use a textual interface; however, many tools use some Unix-specific features and thus run on Unix / Linux / OS X, but not on Windows.

Hat was mostly built around 2000–2004 and then disappeared because of lack of maintenance. Now it is back and new developments have started.

Currently the source-to-source transformation of hat-trans is being rewritten to use the haskell-src-exts parser. Thus small bugs of the current parser will disappear and in the future it will be easier to cover more Haskell language extensions.

When a traced program uses any libraries besides the standard Haskell 98 / 2010 ones, these libraries currently have to be manually transformed (in trusted mode). A new tool will be built to easily wrap any existing libraries such that they can be used by a traced program (without tracing the computations inside the libraries).

Feedback on Hat is welcome.

Further reading

- Initial website: http://olafchitil.github.com/hat
- Hackage package: http://hackage.haskell.org/package/hat

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.

Left fold enumerators have been the most common approach 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. The most recent major release — version 0.5 — was made in June of this year. This design is working efficiently and properly for conduit's use cases, and there are no plans for further breaking changes. The package can be considered mature and ready to be used by the general public.

There is a rich ecosystem of libraries available to be used with conduit, including cryptography, network communications, serialization, XML processing, and more. The Web Application Interface was the original motivator for creating the library, and continues to use it for expressing request and response bodies between servers and applications. As such, conduit is also a major player in the Yesod ecosystem.

The library is available on Hackage. The Haddocks contain a fairly detailed tutorial explaining common usage patterns. You can find many conduit-based packages in the Conduit category on Hackage as well.

Further reading

- http://hackage.haskell.org/packages/archive/conduit/0.5.2.7/doc/html/Data-Conduit.html
- http://hackage.haskell.org/packages/archive/pkg-list.html#cat:conduit

7.1.2 Free Sections

Report by:	Andrew G. Seniuk

Free sections (package `freesect`) extend Haskell (or other languages) to better support partial function application. The package can be installed from Hackage and runs as a preprocessor. Free sections can be explicitly bracketed, or usually the groupings can be inferred automatically.

```
   zipWith          ( f _ $ g _ z )   xs ys,
                      -- context inferred,
 = zipWith         _[ f _ $ g _ z ]_  xs ys,
                      -- explicit bracketing,
 = zipWith  (\ x y -> f x $ g y z )   xs ys,
                      -- after the rewrite,
```

Free sections can be understood by their place in a tower of generalisations, ranging from simple function application, through usual partial application, to free sections, and to named free sections. The latter (where _ wildcards include identifier suffixes) have the same expressivity as a lambda function wrapper, but the syntax is more compact and semiotic.

Although the rewrite provided by the extension is simple, there are advantages of free sections relative to explicitly written lambdas:

- lambda forces the programmer to invent fresh names for the wildcards
- lambda forces the programmer to repeat those names, and place them correctly
- freesect wildcards stand out vividly, indicating where the awaited expressions will go
- reading the lambda requires visual pattern-matching between left and right sides
- lambda is longer overall, and prefaces the expression of interest with boilerplate

On the other hand, the lambda (or named free section) is more powerful than the anonymous free section:

- it can achieve arbitrary permutations without further ado; but anonymous wildcards preserve their lexical order
- it is more expressive when nesting is involved, because the variables are not anonymous

Free sections (like function wrappers generally) are especially useful in refactoring and retrofitting exisitng code, although once familiar they can also be useful from the ground up. Philosophically, use of this sort of syntax promotes "higher-order programming", since any expression can so easily be made into a function, in numerous ways, simply by replacing parts of it with freesect wildcards. That this is worthwhile is demonstrated by the frequent usefulness of sections.

The notion of free sections emanated from an encompassing research agenda around vagaries of lexical syntax. Immediate plans specific to free sections include:

- possibly something could be prepared for academic publication
- implementing the named free sections extension-extension for completeness
- attempting to get it accepted into some project (maybe some Haskell compiler) which handles parsing (my code uses a fork of HSE, and divergence is accruing)

Otherwise, pretty much a one-off which will be deemed stable in a few months. Maybe I'll try extending some language which lacks lambdas (or where its lambda syntax is especially unpleasant).

Further reading

fremissant.net/freesect

7.2 Education

7.2.1 Holmes, Plagiarism Detection for Haskell

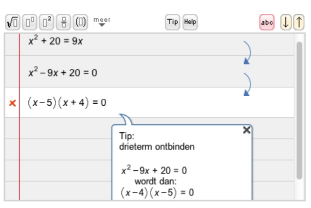
Report by:	Jurriaan Hage
Participants:	Brian Vermeer, Gerben Verburg

See: http://www.haskell.org/communities/11-2011/html/report.html#sect8.1.1.

7.2.2 Interactive Domain Reasoners

Report by:	Bastiaan Heeren
Participants:	Alex Gerdes, Johan Jeuring, Josje Lodder, Bram Schuur
Status:	experimental, active development

The IDEAS project (at Open Universiteit Nederland and Utrecht University) aims at developing interactive domain reasoners on various topics. These reasoners assist students in solving exercises incrementally by checking intermediate steps, providing feedback on how to continue, and detecting common mistakes. The reasoners are based on a strategy language, from which feedback is derived automatically. The calculation of feedback is offered as a set of web services, enabling external (mathematical) learning environments to use our work. We currently have a binding with the Digital Mathematics Environment of the Freudenthal Institute (first/left screenshot), the ActiveMath learning system of the DFKI and Saarland University (second/right screenshot), and our own online exercise assistant that supports rewriting logical expressions into disjunctive normal form.



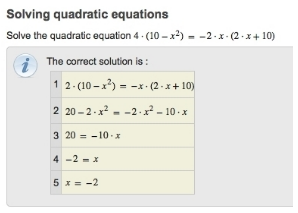
$x^2 + 20 = 9x$

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

× $(x-5)(x+4) = 0$

Tip:
drieterm ontbinden

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



Solving quadratic equations

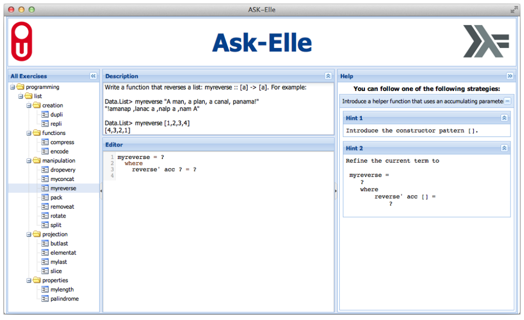
Solve the quadratic equation $4 \cdot (10 - x^2) = -2 \cdot x \cdot (2 \cdot x + 10)$

The correct solution is :

1	$2 \cdot (10 - x^2) = -x \cdot (2 \cdot x + 10)$
2	$20 - 2 \cdot x^2 = -2 \cdot x^2 - 10 \cdot x$
3	$20 = -10 \cdot x$
4	$-2 = x$
5	$x = -2$

We are adding support for more exercise types, mainly at the level of high school mathematics. For example, our domain reasoner now covers simplifying expressions with exponents, rational equations, and derivatives. We have investigated how users can interleave solving different parts of exercises. We have extended our strategy language with different combinators for interleaving, and have shown how the interleaving combinators are implemented in the parsing framework we use for recognizing student behavior and providing hints.

Recently, we have focused on designing the Ask-Elle functional programming tutor. This tool lets you practice introductory functional programming exercises in Haskell. The tutor can both guide a student towards developing a correct program, as well as analyse intermediate, incomplete, programs to check whether or not certain properties are satisfied. We are planning to include checking of program properties using QuickCheck, for instance for the generation of counterexamples. We have to guide the test-generation process to generate test-cases that do not use the part of the program that has yet to be developed. We also want to make it as easy as possible for teachers to add programming exercises to the tutor, and to adapt the behavior of the tutor by disallowing or enforcing particular solutions, and by changing the feedback. Teachers can adapt feedback by annotating the model solutions of an exercise. The tutor has an improved web-interface and is used in an introductory FP course at Utrecht University.



Write a function that reverses a list: myreverse :: [a] -> [a]. For example:

```
Data.List> myreverse "A man, a plan, a canal, panama!"
"!amanap ,lanac a ,nalp a ,nam A"

Data.List> myreverse [1,2,3,4]
[4,3,2,1]
```

```
myreverse = ?
  where
    reverse' acc ? = ?
```

You can follow one of the following strategies:

Introduce a helper function that uses an accumulating paramete

Introduce the constructor pattern [].

Refine the current term to

```
 myreverse =
    ?
   where
       reverse' acc [] =
           ?
```

The feedback services are available as a Cabal source package. The latest release is version 1.0 from September 1, 2011.

Further reading

- Online exercise assistant (for logic), accessible from our project page.
- 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. Interleaving Strategies. Conference on Intelligent Computer Mathematics, Mathematical Knowledge Management (MKM 2011).
- Johan Jeuring, Alex Gerdes, and Bastiaan Heeren. A Programming Tutor for Haskell. To appear in Lecture Notes Central European School on Functional Programming, (CEFP 2011). Try our tutor at http://ideas.cs.uu.nl/ProgTutor/.

7.3 Parsing and Transforming

7.3.1 epub-metadata

Report by:	Dino Morelli
Status:	stable, actively developed

See: http://www.haskell.org/communities/05-2011/html/report.html#sect6.2.4.

7.3.2 Utrecht Parser Combinator Library: uu-parsinglib

Report by:	Doaitse Swierstra
Status:	actively developed

The previous extension for recognizing merging parsers was generalized so now any kind of applicative and monadic parsers can be merged in an interleaved way. As an example take the situation where many different programs write log entries into a log file, and where each log entry is uniquely identified by a transaction number (or process number) which can be used to distinguish them. E.g., assume that each transaction consists of an *a*, a *b* and a *c* action, and that a digit is used to identify the individual actions belonging to the same transaction; the individual transactions can now be recognized by the parser:

$$\begin{aligned} pABC = \mathbf{do}\ d &\leftarrow mkGram\ (pa \ *\!> pDigit) \\ &mkGram\ (pb \ *\!> pSym\ d) \\ &\quad *\!> mkGram\ (pc \ *\!> pSym\ d) \end{aligned}$$

Now running many merged instances of this parser on the input returns the list of numbers, each identifying an occurrence of an "abc" subsequence:

```
run (pmMany(pABC)) "a2a1b1b2c2a3b3c1c3",
Result: "213",
```

Furthermore the library was provided with many more examples in two modules in the *Demo* directory.

Features

- Much simpler internals than the old library (http://haskell.org/communities/05-2009/html/report.html#sect5.5.8).
- 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 makes 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

Since the part dealing with merging is relatively independent of the underlying parsing machinery we may split this off into a separate package. This will enable us also to make use of a different parsing engines when combining parsers in a much more dynamic way. In such cases we want to avoid too many static analyses.

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 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.

Students are working on a package for processing options which makes use of the merging parsers, so that the various options can be set in a flexible but typeful way.

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).

7.4 Generic and Type-Level Programming

7.4.1 Unbound

Report by:	Brent Yorgey
Participants:	Stephanie Weirich, Tim Sheard
Status:	actively maintained

Unbound is a domain-specific language and library for working with binding structure. Implemented on top of the RepLib generic programming framework, it automatically provides operations such as alpha equivalence, capture-avoiding substitution, and free variable calculation for user-defined data types (including GADTs), requiring only a tiny bit of boilerplate on the part of the user. It features a simple yet rich combinator language for binding specifications, including support for pattern binding, type annotations, recursive binding, nested binding, set-like (unordered) binders, and multiple atom types.

Further reading

- http://byorgey.wordpress.com/2011/08/24/unbound-now-supports-set-binders-and-gadts/
- http://byorgey.wordpress.com/2011/03/28/binders-unbound/
- http://hackage.haskell.org/package/unbound
- http://code.google.com/p/replib/

7.4.2 A Generic Deriving Mechanism for Haskell

Report by:	José Pedro Magalhães
Participants:	Atze Dijkstra, Johan Jeuring, Andres Löh, Simon Peyton Jones
Status:	actively developed

Haskell's deriving mechanism supports the automatic generation of instances for a number of functions. The Haskell 98 Report only specifies how to generate instances for the Eq, Ord, Enum, Bounded, Show, and Read classes. The description of how to generate instances is largely informal. As a consequence, the portability of instances across different compilers is not guaranteed. Additionally, the generation of instances imposes restrictions on the shape of datatypes, depending on the particular class to derive.

We have developed a new approach to Haskell's deriving mechanism, which allows users to specify how to derive arbitrary class instances using standard datatype-generic programming techniques. Generic functions, including the methods from six standard Haskell 98 derivable classes, can be specified entirely within Haskell, making them more lightweight and portable.

We have implemented our deriving mechanism together with many new derivable classes in UHC (→3.3) and GHC. The implementation in GHC has a more convenient syntax; consider enumeration:

```
class GEnum a where
  genum :: [a]
  default genum :: (Representable a,
                    Enum' (Rep a)) ⇒ [a]
  genum = map to enum'
```

The *Enum'* and *GEnum* classes are defined by the generic library writer. The end user can then give instances for his/her datatypes without defining an implementation:

```
instance (GEnum a) ⇒ GEnum (Maybe a)
instance (GEnum a) ⇒ GEnum [a]
```

These instances are empty, and therefore use the (generic) default implementation. This is as convenient as writing **deriving** clauses, but allows defining more generic classes. This implementation relies on the new functionality of default signatures, like in *genum* above, which are like standard default methods but allow for a different type signature.

GHC 7.6.1 brings support for automatic derivation of *Generic1* instances, meaning that generic functions that abstract over type containers (such as *fmap*) are now also supported.

Further reading

http://www.haskell.org/haskellwiki/GHC.Generics

7.4.3 Optimising Generic Functions

Report by:	José Pedro Magalhães
Status:	actively developed

Datatype-generic programming increases program reliability by reducing code duplication and enhancing reusability and modularity. However, it is known that datatype-generic programs often run slower than type-specific variants, and this factor can prevent adoption of generic programming altogether. There can be multiple reasons for the performance penalty, but often it is caused by conversions to and from representation types that do not get eliminated during compilation.

Fortunately, it is also known that generic functions can be specialised to concrete datatypes, removing any overhead from the use of generic programming. We have investigated compilation techniques to specialise generic functions and remove the performance overhead of generic programs in Haskell. We used a representative generic programming library and inspected the generated code for a number of example generic functions. After understanding the necessary compiler optimisations for producing efficient generic code, we benchmarked the runtime of our generic functions against handwritten variants, and concluded that all the overhead can indeed be removed automatically by the compiler. More details can be found in the IFL'12 draft paper linked below.

Further reading

Optimisation of Generic Programs through Inlining

7.5 Proof Assistants and Reasoning

7.5.1 HERMIT

Report by:	Andy Gill
Participants:	Andy Gill, Andrew Farmer, Ed Komp, Neil Sculthorpe, Adam Howell, Robert Blair, Ryan Scott, Patrick Flor, Michael Tabone
Status:	active

The Haskell Equational Reasoning Model-to-Implementation Tunnel (HERMIT) is an NSF-funded project being run at KU (→ 9.8), which aims to improve the applicability of Haskell-hosted Semi-Formal Models to High Assurance Development. Specifically, HERMIT will use: a Haskell-hosted DSL; the Worker/Wrapper Transformation; and a new refinement user interface to perform rewrites directly on Haskell Core, the GHC internal representation.

This project is a substantial case study of the application of Worker/Wrapper on larger examples. In particular, we want to demonstrate the equivalences between efficient Haskell programs, and their clear specification-style Haskell counterparts. In doing so there are several open problems, including refinement scripting and managing scaling issues, data representation and presentation challenges, and understanding the theoretical boundaries of the worker/wrapper transformation.

The project started in Spring 2012, and is expected to run for two years. The HERMIT team currently consists of one assistant professor, one research engineer, one post-doc, one PhD student, one Masters student, two undergraduates, and three student and ex-student volunteers.

We have reworked KURE (http://www.haskell.org/communities/11-2008/html/report.html#sect5.5.7), a Haskell-hosted DSL for strategic programming, as the basis of our rewrite capabilities, and constructed the rewrite kernel making use of the GHC Plugins architecture. As for interfaces to the kernel, we currently have a command-line REPL, we are constructing a web-based API, and an Android version is planned. A detailed introduction to the HERMIT architecture and implementation can be found in our Haskell Symposium 2012 paper. Thus far, we have used HERMIT to successfully mechanize about a dozen small examples of program transformations, drawn from the literature on techniques such as concatenate vanishes, tupling transformation, and worker/wrapper. A discussion of our experiences mechanizing these examples can be found in our IFL 2012 paper.

Funded by the NSF REU initiative and ITTC (our research center), we are also working on an Android application interface for HERMIT, where gestures can be used to manipulate Haskell Core programs. Five KU undergraduates are working on the Android application.

Further reading

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

7.5.2 HTab

Report by:	Guillaume Hoffmann
Status:	active development

HTab is an automated theorem prover for hybrid logics based on a tableau calculus. It handles hybrid logic with nominals, satisfaction operators, converse modalities, universal modalities, the down-arrow binder, and role inclusion.

Main changes of version 1.6.0 are the switch to a better blocking mechanism called pattern-based blocking, and general effort to reduce and clean up the source code (removing some features in the process) to facilitate further experiments.

It is available on Hackage and comes with sample formulas to illustrate its input format.

Further reading

http://code.google.com/p/intohylo/

7.5.3 Free Theorems for Haskell

Report by:	Janis Voigtländer
Participants:	Daniel Seidel

See: http://www.haskell.org/communities/11-2011/html/report.html#sect8.6.3.

7.6 Mathematical Objects

7.6.1 dimensional: Statically Checked Physical Dimensions

Report by:	Björn Buckwalter
Status:	active, stable core with experimental extras

Dimensional is a library providing data types for performing arithmetics 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:

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

The dimensional library is stable with units being added on an as-needed basis. The primary documentation is the literate Haskell source code. The wiki on the project web site has several usage examples to help with getting started.

Ongoing experimental work includes:

- Support for user-defined dimensions and a proof-of-concept implementation of the CGS system of units.
- dimensional-vectors — a rudimentary linear algebra library which statically tracks the sizes of vectors and matrices as well as the physical dimensions of their elements on a per element basis, disallowing non-sensical operations. This library makes it very difficult to accidentally implement, e.g., a Kalman filter incorrectly. My work on dimensional-vectors is need-driven and tends to occur in spurts.
- dimensional-experimental — a library in heavy flux of which the most interesting feature is probably automatic differentiation of functions involving physical quantities. Example:

  $$\begin{aligned}&v :: \mathit{Fractional}\ a \Rightarrow \mathit{Time}\ a \rightarrow \mathit{Velocity}\ a\\&v\ t = \mathit{diff}\ \ d\ t\end{aligned}$$

- dimensional-tf — dimensional was originally implemented using functional dependencies but in January 2012 a port using type families was released. For the time being dimensional-tf is considered experimental but if it eventually proves itself to be a better dimensional it will be merged into the latter with a major version bump.

The core library, dimensional, as well as dimensional-tf, can be installed off Hackage using cabal. The other experimental packages can be cloned off of Github.

Dimensional relies on *numtype* for type-level integers (e.g., *pos2* in the above example), *ad* for automatic differentiation, and *HList* (→7.7.1) for type-level vector and matrix representations.

Further reading

- http://dimensional.googlecode.com
- https://github.com/bjornbm/dimensional-vectors
- https://github.com/bjornbm/dimensional-experimental
- http://flygdynamikern.blogspot.com/2012/02/announce-dimensional-tf-010-statically.html

7.6.2 AERN

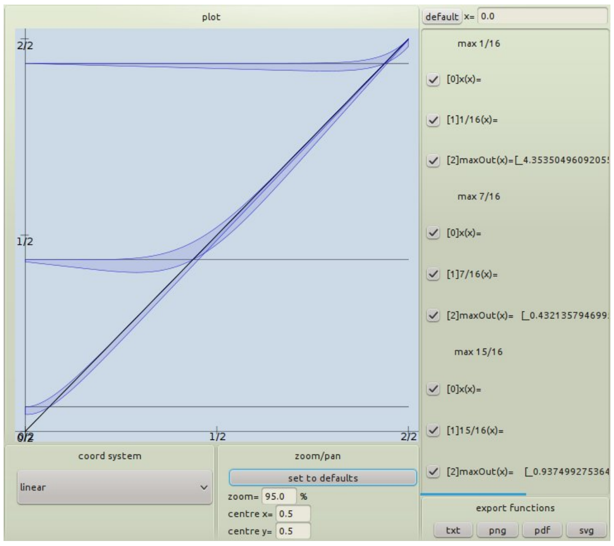
Report by:	Michal Konečný
Participants:	Jan Duracz
Status:	experimental, actively developed

AERN stands for Approximating Exact Real Numbers. We are developing a family of libraries that will provide:

- A reliable arbitrary-precision correctly rounded **interval arithmetic**, including both standard intervals and inverted intervals with Kaucher arithmetic. Reliability is achieved using extensive QuickCheck testing against a nearly-complete formalisation of the real numbers.
- Arbitrary-precision arithmetic of **polynomial intervals** (similar to but more general than Taylor Models). This is useful for example for:
  - Automatically reducing overestimations in interval computations.
  - Efficiently supporting validated numerical integration, specifically in the simulation of ordinary differential equation (ODE) and hybrid system initial value problems (IVPs).
  - Automatically deciding many inequalities and interval inclusions with non-linear and elementary functions that occur in numerical theorem proving and, specifically, in the verification of numerical programs.
- A type class hierarchy for validated and exact computation, featuring:
  - Standard mathematical structures such as posets and lattices extended to take account of rounding errors and partially decided relations such as equality.
  - Both numerical order and interval refinement order.
  - An ability to increase computational effort with the view to reduce the negative effects of rounding and of the partial ability to decide equality. The approximate operations and partially decided relations converge to exact operations and totally decided relations as effort approaches infinity.
  - Extensive set of QuickCheck properties for each type class, enabling automatic checking of, e.g., algebraic properties such as associativity, extended to take account of rounding.

- Benchmarks for comparing the efficiency of various versions of validated approximate arithmetic, e.g., various interval arithmetics and various function enclosure arithmetics.

- ◦ Tools for interactive plotting of univariate function enclosures (see figure below for a screenshot of an early prototype).
- ◦ A framework for distributed query-driven lazy dataflow validated numerical computation with denotational exact semantics based on Domain Theory.



x= 0.0

✓ [0]x(x)=
✓ [1]1/16(x)=
✓ [2]maxOut(x)=[_4.3535049609205

✓ [0]x(x)=
✓ [1]7/16(x)=
✓ [2]maxOut(x)= [_0.432135794699

✓ [0]x(x)=
✓ [1]15/16(x)=
✓ [2]maxOut(x)= [_0.937499275364

linear

zoom= 95.0 %
centre x= 0.5
centre y= 0.5

There are stable older versions of the libraries on Hackage but these lack the type classes described above.

We are still in the process of redesigning and rewriting the libraries. Out of the newly designed code, we have so far released libraries featuring:

- ◦ The type classes for approximate real number operations.
- ◦ Correctly rounded real interval arithmetic with Double endpoints.

A release of interval arithmetic with MPFR endpoints is planned in before the end of 2012 despite the fact that currently one has to recompile GHC to use MPFR safely.

We have made progress on implementing polynomial intervals and plan to release them by the end of 2012. The development files include demos that solve selected ODE and hybrid system IVPs using polynomial intervals.

All AERN development is open and we welcome contributions and new developers.

Further reading

http://code.google.com/p/aern/

7.6.3 Paraiso

Report by:	Takayuki Muranushi
Status:	active development

Paraiso is a domain-specific language (DSL) embedded in Haskell, aimed at generating explicit type of partial differential equations solving programs, for accelerated and/or distributed computers. Equations for fluids, plasma, general relativity, and many more falls into this category. This is still a tiny domain for a computer scientist, but large enough that an astrophysicist (I am) might spend even his entire life in it.

In Paraiso we can describe equation-solving algorithms in mathematical, simple notation using *builder monads*. At the moment it can generate programs for multicore CPUs as well as single GPU, and tune their performance via automated benchmarking and genetic algorithms. The first set of experiment have been performed and published as a paper (http://arxiv.org/abs/1204.4779), accepted to Computational Science & Discovery.

Anyone can get Paraiso from hackage (http://hackage.haskell.org/package/Paraiso) or github (https://github.com/nushio3/Paraiso).

Further reading

http://paraiso-lang.org/wiki/

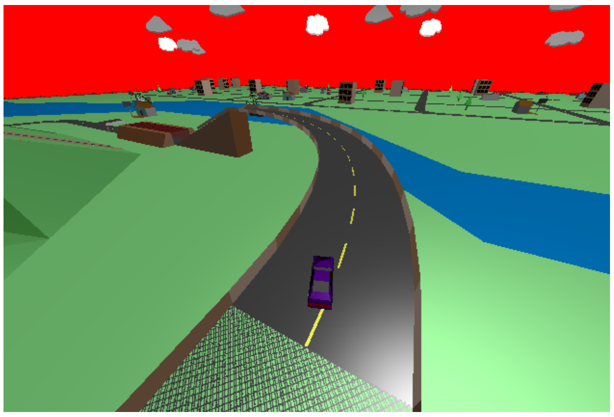
7.6.4 Bullet

Report by:	Csaba Hruska
Status:	experimental, active development

Bullet is a professional open source multi-threaded 3D Collision Detection and Rigid Body Dynamics Library written in C++. It is free for commercial use under the zlib license. The Haskell bindings ship their own (auto-generated) C compatibility layer, so the library can be used without modifications. The Haskell binding provides a low level API to access Bullet C++ class methods. Some bullet classes (Vector, Quaternion, Matrix, Transform) have their own Haskell representation, others are binded as class pointers. The Haskell API provides access to some advanced features, like constraints, vehicle and more.

At the current state of the project most common services are accessible from Haskell, i.e., you can load collision shapes and step the simulation, define constraints, create raycast vehicle, etc. More advanced Bullet features (soft body simulation, Multithread and GPU constaint solver, etc.) will be added later.

Currently we are developing a new high level FRP based API, which is built top of Bullet.Raw module using the Elerea library.



Further reading

http://www.haskell.org/haskellwiki/Bullet

7.7 Data Types and Data Structures

7.7.1 HList — A Library for Typed Heterogeneous Collections

Report by:	Oleg Kiselyov
Participants:	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 (→ 5.3.4), typed EDSL of attribute grammars, and in HaskellDB.

The October 2012 version of HList library marks the significant re-write to take advantage of the fancier types offered by GHC 7.4+. HList now relies on type-level booleans, natural numbers and lists, and on kind polymorphism. A number of operations are implemented as type functions. Another notable addition is unfold for heterogeneous lists. Many operations (projection, splitting) are now implemented in terms of unfold. Such a refactoring moved more computations to type-level, with no run-time overhead.

Currently the core of HList has been re-written: HList, HArray, TIP – up to records. In the near future, we will finish the re-writing and take advantage of the better kind polymorphism supported by GHC 7.6+.

Further reading

- HList: http://okmij.org/ftp/Haskell/types.html#HList
- OOHaskell: http://homepages.cwi.nl/~ralf/OOHaskell/

7.7.2 Persistent

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

Persistent is a type-safe data store interface for Haskell. Haskell has many different database bindings available. However, most of these have little knowledge of a schema and therefore do not provide useful static guarantees. Persistent is designed to work across different databases, and works on Sqlite, PostgreSQL, MongoDB, and MySQL. MySQL is a new edition since the last HCAR, thanks to Felipe Lessa.

Since the last report, Persistent has been structured into separate type-classes. There is one for storage/serialization, and one for querying. This means that anyone wanting to create database abstractions can re-use the battle-testsed persistent storage/serialization layer. Persistent's query layer is universal across different backends and uses combinators:

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

There are some drawbacks to the query layer: it doesn't cover every use case. Since the last HCAR report, Persistent has gained some very good support for raw SQL. One can run arbitrary SQL queries and get back Haskell records or types for single columns.

Persistent also gained the ability to store embedded objects. One can store a list or a Map inside a column/field. The current implementation is most useful for MongoDB. In SQL an embedded object is stored as JSON.

Future plans

Future directions for Persistent:
- Full CouchDB support
- A MongoDB specific query layer
- Adding key-value databases like Redis without a query layer.

Most of Persistent development occurs within the Yesod (→ 5.2.6) community. However, there is nothing specific to Yesod about it. You can have a type-safe, productive way to store data, even on a project that has nothing to do with web development.

Further reading

http://yesodweb.com/book/persistent

7.7.3 DSH — Database Supported Haskell

Report by:	Torsten Grust
Participants:	George Giorgidze, Tom Schreiber, Alexander Ulrich, Jeroen Weijers
Status:	active development

DSH :: → [SQL]

Database-Supported Haskell, DSH for short, is a Haskell library for database-supported program execution. Using the DSH library, a relational database management system (RDBMS) can be used as a coprocessor for the Haskell programming language, especially for those program fragments that carry out data-intensive and data-parallel computations. Rather than embedding a relational language into Haskell, DSH turns idiomatic Haskell programs into SQL queries. The DSH library and the FerryCore package it uses are available on Hackage (http://hackage.haskell.org/package/DSH).

Support for algebraic data types. Algebraic data types (ADTs) are the essential data modelling tool of a number of functional programming languages like Haskell, OCaml and F#. In recent work we added support for ADTs to DSH. ADTs may be freely constructed and deconstructed in queries and may show up in the result type. The number of relational queries generated is small and statically determined by the type of the query.

DSH in the Real World. We have used DSH for large scale data analysis. Specifically, in collaboration with researchers working in social and economic sciences, we used DSH to analyse the entire history of Wikipedia (terabytes of data) and a number of online forum discussions (gigabytes of data).

Because of the scale of the data, it would be unthinkable to conduct the data analysis in Haskell without using the database-supported program execution technology featured in DSH. We have formulated several DSH queries directly in SQL as well and found that the equivalent DSH queries were much more concise, easier to write and maintain (mostly due to DSH's support for nesting, Haskell's abstraction facilities and the monad comprehension notation, see below).

One long-term goal is to allow researchers who are not necessarily expert programmers or database engineers to conduct large scale data analysis themselves.

Towards a New Compilation Strategy. As of today, DSH relies on a query compilation strategy coined *loop-lifting*. Loop-lifting comes with important and desirable properties (*e.g.*, the number of SQL queries issued for a given DSH program only depends on the *static type* of the program's result). The strategy, however, relies on a rather complex and monolithic mapping of programs to the relational algebra. To remedy this, we are currently exploring a new strategy based on the *flattening transformation* as conceived by Guy Blelloch. Originally designed to implement the data-parallel declarative language NESL, we revisit flattening in the context of query compilation (which targets database kernels, one particular kind of data-parallel execution environment). Initial results are promising and DSH might switch over in the not too far future. We hope to further improve query quality and also address the formal correctness of DSH's program-to-queries mapping.

Related Work. Motivated by DSH we reintroduced the *monad comprehension* notation into GHC and also extended it for parallel and SQL-like comprehensions. The extension is available in GHC 7.2. We have also implemented a Haskell extension for *overloading the list notation*. This extension will be available in GHC in the near future.

Further reading

http://db.inf.uni-tuebingen.de/research/dsh

7.8 User Interfaces

7.8.1 Gtk2Hs

Report by:	Daniel Wagner
Participants:	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

The most recent release includes GHC 7.6 compatibility (thanks to John Lato) and several minor behavioral improvements.

Further reading

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

7.8.2 xmonad

Report by:	Gwern Branwen
Status:	active development

XMonad is a tiling window manager for X. Windows are arranged automatically to tile the screen without gaps or overlap, maximizing screen use. Window manager features are accessible from the keyboard; a mouse is optional. XMonad is written, configured, and extensible in Haskell. Custom layout algorithms, key bindings, and other extensions may be written by the user in config files. Layouts are applied dynamically, and different layouts may be used on each workspace. Xinerama is fully supported, allowing windows to be tiled on several physical screens.

Development since the last report has continued; XMonad founder Don Stewart has stepped down and Adam Vogt is the new maintainer. After gestating for 2 years, version 0.10 has been released, with simultaneous releases of the XMonadContrib library of customizations (which has now grown to no less than 216 modules encompassing a dizzying array of features) and the xmonad-extras package of extensions,

Details of changes between releases can be found in the release notes:
- http://haskell.org/haskellwiki/Xmonad/Notable_changes_since_0.8
- http://haskell.org/haskellwiki/Xmonad/Notable_changes_since_0.9
- the Darcs repositories have been upgraded to the hashed format
- XMonad.Config.PlainConfig allows writing configs in a more 'normal' style, and not raw Haskell
- Supports using local modules in xmonad.hs; for example: to use definitions from ~/.xmonad/lib/XMonad/Stack/MyAdditions.hs
- xmonad –restart CLI option
- xmonad –replace CLI option
- XMonad.Prompt now has customizable keymaps
- Actions.GridSelect - a GUI menu for selecting windows or workspaces & substring search on window names
- Actions.OnScreen
- Extensions now can have state
- Actions.SpawnOn - uses state to spawn applications on the workspace the user was originally on, and not where the user happens to be
- Markdown manpages and not man/troff
- XMonad.Layout.ImageButtonDecoration & XMonad.Util.Image
- XMonad.Layout.Groups
- XMonad.Layout.ZoomRow
- XMonad.Layout.Renamed
- XMonad.Layout.Drawer
- XMonad.Layout.FullScreen
- XMonad.Hooks.ScreenCorners
- XMonad.Actions.DynamicWorkspaceOrder
- XMonad.Actions.WorkspaceNames
- XMonad.Actions.DynamicWorkspaceGroups

Binary packages of XMonad and XMonadContrib are available for all major Linux distributions.

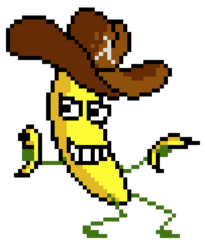
Further reading

- Homepage: http://xmonad.org/
- Darcs source:
  `darcs get` http://code.haskell.org/xmonad
- IRC channel: `#xmonad @@ irc.freenode.org`
- Mailing list: ⟨xmonad@haskell.org⟩

7.9 Functional Reactive Programming

7.9.1 reactive-banana

Report by:	Heinrich Apfelmus
Status:	active development



Reactive-banana is a practical 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.

- Writing *graphical user interfaces* with FRP is made easy. The library can be hooked into any existing event-based framework like wxHaskell or Gtk2Hs. A plethora of example code helps with getting started. You can mix FRP and imperative style. If you don't know how to express functionality in terms of FRP, just temporarily switch back to the imperative style.

- 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.

- It features an *efficient implementation.* No more spooky time leaks, predicting space & time usage should be straightforward.

Status. Version 0.7.0.0 of the reactive-banana library has been released on hackage.

Compared to the previous report, the library now features efficient *dynamic event switching*, also known as *first class events*. This means that events and behaviors can now be created on the fly, they no longer have to be specified fully at compilation time. For instance, it is now possible to implement a GUI where text entry widgets can be added and removed on user command. The source code example `BarTab.hs` demonstrates this.

This is a significant milestone because in very early approaches to FRP, dynamic event switching has been the cause of major inefficiencies, namely the so-called time leaks. By using the type system, reactive-banana can rule out these gross inefficiencies.

The API for dynamic event switching explores a different part of the design space than other packages for FRP, in particular the `sodium` library. There is a trade-off: reactive-banana is simpler when you don't use dynamic event switching, sodium is simpler for heavy uses of dynamic event switching. Hopefully, time will tell which approach provides the more pleasant overall FRP experience.

Current development. Programming GUIs for the world wide web has become very important in recent years. Fortunately, efforts to compile Haskell to JavaScript are reaching the point of becoming usable now, and I intend to make FRP with reactive-banana available for the web as soon as possible, for instance by reducing dependencies on GHC extensions and libraries.

Reactive-banana's implementation of dynamic event switching addresses the grossest of inefficiencies, but some more benign efficiency problems still remain, in particular concerning garbage collection of dynamic events. They will be addressed in a future version.

Notable use cases. In his reactive-balsa library, Henning Thielemann uses reactive-banana to control digital musical instruments with MIDI in real-time.

Further reading

- Project homepage: http://haskell.org/haskellwiki/Reactive-banana
- Example code: http://haskell.org/haskellwiki/Reactive-banana/Examples
- BarTab example: http://haskell.org/haskellwiki/Reactive-banana/Examples#bartab
- reactive-balsa: http://www.haskell.org/haskellwiki/Reactive-balsa
- sodium: http://hackage.haskell.org/package/sodium

7.9.2 Functional Hybrid Modelling

Report by:	George Giorgidze
Participants:	Joey Capper, Henrik Nilsson
Status:	active research and development

The goal of the FHM project is to gain a better foundational understanding of noncausal, hybrid modelling and simulation languages for physical systems and ultimately to improve on their capabilities. At present, our central research vehicle to this end is the design and implementation of a new such language centred around a small set of core notions that capture the essence of the domain.

Causal modelling languages are closely related to synchronous data-flow languages. They model system behaviour using ordinary differential equations (ODEs) in explicit form. That is, cause-effect relationship between variables must be explicitly specified by the modeller. In contrast, noncausal languages model system behaviour using differential algebraic equations (DAEs) in implicit form, without specifying their causality. Inferring causality from usage context for simulation purposes is left to the compiler. The fact that the causality can be left implicit makes modelling in a noncausal language declarative (the focus is on expressing the equations in a natural way, not on how to express them to enable simulation) and also makes the models more reusable.

FHM is an approach to modelling which combines purely functional programming and noncausal modelling. In particular, the FHM approach proposes modelling with first class models (defined by continuous DAEs) using combinators for their composition and discrete switching. The discrete switching combinators enable modelling of hybrid systems (i.e., systems that exhibit both continuous and discrete dynamic behaviour). The key concepts of FHM originate from work on Functional Reactive Programming (FRP).

We are implementing Hydra, an FHM language, as a domain-specific language embedded in Haskell. The method of embedding employs quasiquoting and enables modellers to use the domain specific syntax in their models. The present prototype implementation of Hydra enables modelling with first class models and supports combinators for their composition and discrete switching.

We implemented support for dynamic switching among models that are computed at the point when they are being "switched in". Models that are computed at run-time are just-in-time (JIT) compiled to efficient machine code. This allows efficient simulation of structurally dynamic systems where the number of structural configurations is large, unbounded or impossible to determine in advance. This goes beyond to what current state-of-the-art noncausal modelling languages can model. The implementation techniques that we developed should benefit other modelling and simulation languages as well.

We are also exploring ways of utilising the type system to provide stronger correctness guarantees and to provide more compile time reassurances that our system of equations is not unsolvable. Properties such as equational balance (ensuring that the number of equations and unknowns are balance) and ensuring the solvability of locally scoped variables are among our goals.

Furthermore, a minimal core language for FHM is being developed and formalised in the dependently-typed language Agda. The goals of the core language are to capture the essence of Hydra such that we can demonstrate its correctness and prove the existance of a number of desirable properties. Of particular interest is the soundness of the implementation with respect to the formal semantics, and properties such as termination and productivity for the structural dynamics.

Recently, George Giorgidze completed his PhD thesis featuring an in-depth description of the design and implementation of the Hydra language. In addition, the thesis features a range of example physical systems modelled in Hydra. The examples are carefully chosen to showcase those language features of Hydra that are lacking in other noncausal modelling languages.

Further reading

The implementation of Hydra and related papers (including George's PhD thesis) are available from http://db.inf.uni-tuebingen.de/team/giorgidze.

Implementation and articles relating to the formalisation of an FHM core language can be found at http://cs.nott.ac.uk/~jjc.

7.9.3 Elerea

Report by:	Patai Gergely
Status:	experimental, active

Elerea (Eventless reactivity) is a tiny discrete time FRP implementation without the notion of event-based switching and sampling, with first-class signals (time-varying values). Reactivity is provided through various higher-order constructs that also allow the user to work with arbitrary time-varying structures containing live signals.

Stateful signals can be safely generated at any time through a specialised monad, while stateless combinators can be used in a purely applicative style. Elerea signals can be defined recursively, and external input is trivial to attach. The library comes in three major variants, which all have precise denotational semantics:

- `Simple`: signals are plain discrete streams isomorphic to functions over natural numbers;
- `Param`: adds a globally accessible input signal for convenience;
- `Clocked`: adds the ability to freeze whole subnetworks at will.

The code is readily available via cabal-install in the `elerea` package. You are advised to install `elerea-examples` as well to get an idea how to build non-trivial systems with it. The examples are separated in order to minimize the dependencies of the core library. The experimental branch is showcased by Dungeons of Wor, found in the `dow` package (http://www.haskell.org/communities/05-2010/html/report.html#sect6.11.2). Additionally, the basic idea behind the experimental branch is laid out in the WFLP 2010 article *Efficient and Compositional Higher-Order Streams*.

Since the last report, the API was extended with effectful combinators that allow IO computations to be used in the definitions of the signals. The primary use for this functionality is to provide FRP-style bindings on top of imperative libraries. At the moment, a high-level Elerea based API for the Bullet physics library is under development.

Further reading

- http://hackage.haskell.org/package/elerea
- http://hackage.haskell.org/package/elerea-examples
- http://hackage.haskell.org/package/dow
- http://sgate.emt.bme.hu/documents/patai/publications/PataiWFLP2010.pdf
- http://babel.ls.fi.upm.es/events/wflp2010/video/video-08.html (WFLP talk)

7.10 Graphics

7.10.1 LambdaCube

Report by:	Csaba Hruska
Participants:	Gergely Patai
Status:	experimental, active development

LambdaCube 3D is a domain specific language and library that makes it possible to program GPUs in a purely functional style.

Programming with LambdaCube constitutes of composing a pure data-flow description, which is compiled into an executable module and accessed through a high-level API. The language provides a uniform way to define shaders and compositor chains by treating both streams and framebuffers as first-class values.

In its current state, LambdaCube is already functional, but still in its infancy. The current API is a rudimentary EDSL that is not intended for direct use in the long run. It is essentially the internal phase of a compiler backend exposed for testing purposes. To exercise the library, we have created two small proof of concept examples: a port of the old LambdaCube Stunts example, and a Quake III level viewer.

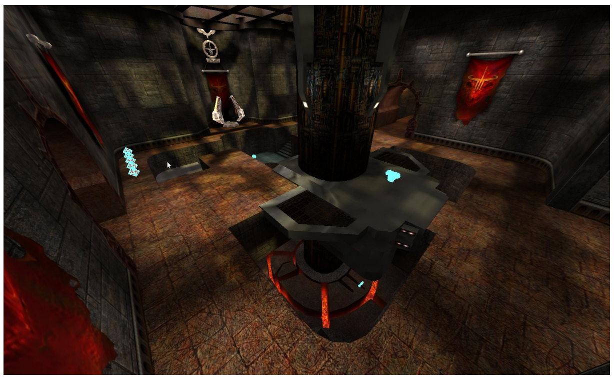
Over the last few months, we extended the implementation with some essential major features:

- texture support
- multi-pass rendering
- sharing detection and CSE in the shaders (through hash-consing)

We also improved the existing examples and created new ones: a showcase for variance shadow mapping and another for integration with the Bullet physics engine.

Last but not least, we finally started a new blog dedicated to LambdaCube. The blog is intended to be the primary source of information and updates on the project from now on.

Everyone is invited to contribute! You can help the project by playing around with the code, thinking about API design, finding bugs (well, there are a lot of them anyway), creating more content to display, and generally stress testing the library as much as possible by using it in your own projects.



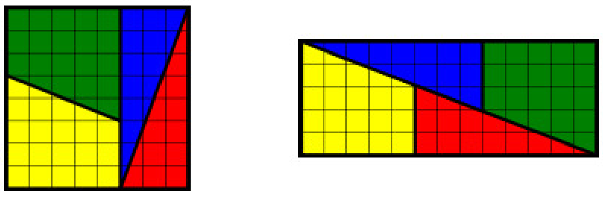
Further reading

- https://lambdacube3d.wordpress.com/
- https://github.com/csabahruska/lc-dsl
- http://www.haskell.org/haskellwiki/LambdaCubeEngine
- http://www.youtube.com/watch?v=kDu5aCGc8l4

7.10.2 diagrams

Report by:	Brent Yorgey
Participants:	Jan Bracker, Andy Gill, Chris Mears, Michael Sloan, Ryan Yates
Status:	active development

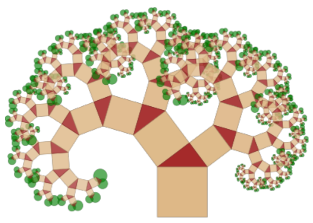
The diagrams framework provides an embedded domain-specific language for declarative drawing. The overall vision is for diagrams to become a viable alternative to DSLs like MetaPost or Asymptote, but with the advantages of being *declarative*—describing what to draw, not how to draw it—and *embedded*—putting the entire power of Haskell (and Hackage) at the service of diagram creation. There is still much more to be done, but diagrams is already quite fully-featured, with a comprehensive user manual, 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

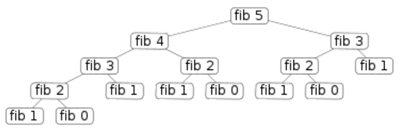
Development proceeds slowly (since most of the main developers are busy with other things) but passionately. The upcoming 0.6 release didn't make it out the door in time for the HCAR deadline, but look for a new release in early December! New features in 0.6 will include:

- All diagrams-related repositories have now moved to github, to foster increased contribution.
- "Traces", which give an easy way to find arbitrary points on the boundary of a diagram (useful for, *e.g.* drawing connecting lines between diagrams).
- Proper support for *subdiagrams*, making possible advanced techniques like constructing animations via keyframing.
- Nicer syntax for constructing literal points and vectors.
- More work on SVG, postscript, and HTML5 canvas backends. The status of various backends can be seen at http://projects.haskell.org/diagrams/backend-tests/all-index.html.
- Two new general-purpose libraries spun off from `diagrams-core`, namely `monoid-extras` (containing some special-purpose constructions on monoids) and `dual-tree` (a rose tree data structure with cached and accumulating monoidal annotations).
- Many other small improvements, new features, and bug fixes.



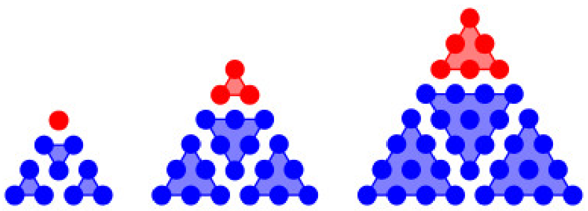
Some other exciting recent things in diagrams-land:

- The `diagrams-builder` package allows rendering diagrams dynamically, at run time, and has been used to support things like inline diagrams code in blog posts and LaTeX documents (see https://byorgey.wordpress.com/2012/08/28/creating-documents-with-embedded-diagrams/).
- Brent recently presented a paper at the Haskell Symposium, *Monoids: Theme and Variations*, based on diagrams and motivating the design of some of its core data structures: http://www.cis.upenn.edu/~byorgey/pub/monoid-pearl.pdf.



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.



Future plans

Some exciting work on animation, interactivity, and using diagrams as a high-powered presentation tool is underway—stay tuned!

A native SVG backend is under active development and almost ready to be used as a drop-in replacement for the cairo backend. The cairo backend will still be supported, but SVG will replace cairo as the default "out-of-the-box" backend, vastly simplifying installation for new useres. Other features planned for the near future include better arrow support, multi-page diagrams, and gradients. Longer-term plans include a richer API for drawing curved paths, interactive diagrams, and a custom Gtk application for interactively developing diagrams and animations.

Further reading

- http://projects.haskell.org/diagrams
- http://projects.haskell.org/diagrams/gallery.html
- http://haskell.org/haskellwiki/Diagrams
- http://github.com/diagrams
- https://byorgey.wordpress.com/2012/08/28/creating-documents-with-embedded-diagrams/
- http://www.cis.upenn.edu/~byorgey/pub/monoid-pearl.pdf
- http://www.youtube.com/watch?v=X-8NCkD2vOw

7.11 Audio

7.11.1 Audio Signal Processing

Report by:	Henning Thielemann
Status:	experimental, active development

This project covers many aspects of audio signal processing in Haskell. It is based on the Numeric Prelude framework (http://haskell.org/communities/05-2009/html/report.html#sect5.6.2). Over the time the project has grown to a set of several packages:

- `synthesizer-core`: Raw implementations of oscillators, noise generation, frequency filters, resampling, pitch and time manipulation, Fourier transformation. Support for several data structures like lists, signal generators, storable vectors and causal signal processing arrows that allow you to balance between efficiency and flexibility.
- `synthesizer-dimensional`: Type-safe physical units in signal processing and abstraction from sample rate.
- `synthesizer-midi`: Render audio streams from sequences of MIDI events.
- `synthesizer-alsa`: Everything that is needed for a real-time software synthesizer within the Advanced Linux Sound Architecture ALSA.
- `synthesizer-llvm`: Highly efficient signal processing by Just-In-Time compilation and vectorization through the Low-Level Virtual Machine (http://llvm.org/), including a real-time software synthesizer.
- `sample-frame`, `sample-frame-np`: Type classes shared between the packages for various sample formats (integer, float, logarithmic encoding, stereo).
- `alsa-core`, `alsa-pcm`, `alsa-seq`, `jack`: Bindings to audio input and output via ALSA and JACK.

◦ `sox`, `soxlib`: Reading and writing many audio file formats and play sounds via `sox` shell command or `libsox` binary interface.

Recent advances are:

◦ Extended and safer bindings to ALSA PCM and ALSA MIDI sequencer.
◦ MIDI, ALSA and LLVM code is now cleanly separated.
◦ Example program `split-record` that divides an audio file according to pauses.

Further reading

http://www.haskell.org/haskellwiki/Synthesizer

7.11.2 Live-Sequencer

Report by:	Henning Thielemann
Participants:	Johannes Waldmann
Status:	experimental, active

The Live-Sequencer allows to program music in the style of Haskore, but it is inherently interactive. You cannot only listen to changes to the music quickly, but you can alter the music while it is played. Changes to the music may not have an immediate effect but are respected when their time has come.

Additionally users can alter parts of the modules of a musical work via a WWW interface. This way multiple people including the auditory can take part in a live composition. This mode can also be used in education, when students shall solve small problems in an exercise.

Technical background: The music is represented as lazy list of MIDI events. (MIDI is the Musical Instrument Digital Interface). The MIDI events are sent via ALSA and thus can control any kind of MIDI application, be it software synthesizers on the same computer or external hardware synthesizers. The application can also receive MIDI events that are turned into program code. We need certain ALSA functionality for precise timing of events. Thus the sequencer is currently bound to Linux.

The Live-Sequencer can be run either as command-line program without editing functions or as an interactive program based on wxwidgets.

The used language is a much simplified kind of Haskell. It provides no sharing, misses many syntactic constructs and is untyped. However the intersection between Haskell and the Live-Sequencer language is large enough for algorithmic music patterns and we provide several examples that are contained in this intersection.

Future plans

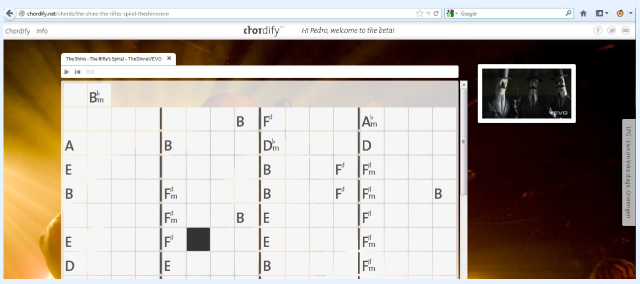
◦ Define proper semantics for live changes to a program
◦ Use of Helium's parser, module system and type checker
◦ Refined reduction steps for educational purposes
◦ Highlighting of active terms that better fits to the music

Further reading

http://www.haskell.org/haskellwiki/Live-Sequencer

7.11.3 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 Soundcloud, Youtube, 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 have recently entered an open beta testing phase, so we invite all users to visit chordify.net, request a beta account, and try Chordify. We are especially interested in feedback. 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

7.11.4 Euterpea

Report by:	Paul Hudak
Participants:	Donya Quick, Daniel Winograd-Cort
Status:	prototype release, active development

Overview

Euterpea is a Haskell library for computer music applications. It is a descendent of Haskore and HasSound, and is intended for both educational purposes as well as serious computer music development. Euterpea can be thought of as a "wide-spectrum" DSL, suitable for high-level music representation, algorithmic composition, and analysis; mid-level concepts such as MIDI; and low-level audio processing, sound synthesis, and instrument design. It also includes a *musical user interface* (MUI), a set of GUI widgets such as sliders, buttons, and so on.

The audio and MIDI-stream processing aspects of Euterpea are based on *arrows*, which makes programs analogous to signal processing diagrams. Using arrows prevents certain kinds of space leaks, and facilitates significant optimization strategies (in particular, the use of *causal commutative arrows*.

Euterpea is being developed at Yale in Paul Hudak's research group, where it has become a key component of Yale's new Computing and the Arts major. Hudak is teaching a two-term sequence in computer music using Euterpea, and is developing considerable pedagogical material, including a new textbook tentatively titled *The Haskell School of Music — From Signals to Symphonies* (HSoM). The name "Euterpea" is derived from "Euterpe", who was one of the nine Greek Muses (goddesses of the arts), specifically the Muse of Music.

Status

The system is stable enough for experimental computer music applications, and for use in coursework either to teach Haskell programming or to teach computer music concepts.

All source code, papers, and a draft of the HSoM textbook can be found on the Yale Haskell Group website at: http://haskell.cs.yale.edu/.

History

Haskore is a Haskell library developed over 15 years ago by Paul Hudak and his students at Yale for high-level computer music applications. *HasSound* was a later development that served as a functional front-end to csound's sound synthesis capabilities. Euterpea combines Haskore with a native Haskell realization of HasSound (i.e. no csound dependencies).

Future Plans

Euterpea is a work in progress, as is the HSoM textbook. Computer-music specific MUI widgets (such as keyboards and guitar frets), further optimization strategies, better support for real-time MIDI and audio processing, and a parallel (multicore) implementation are amongst the planned future goals.

Anyone who would like to contribute to the project, please contact Paul Hudak at paul.hudak@yale.edu.

FurtherReading

Please visit http://haskell.cs.yale.edu/. Click on "Euterpea" to learn more about the library, "Publications" to find our papers on computer music (including HSoM), and "CS431" or "CS432" to see the course material used in two computer music classes at Yale that use Euterpea.

7.12 Text and Markup Languages

7.12.1 HaTeX

Report by:	Daniel Díaz
Status:	Stabilizing and improving
Current release:	Version 3.3

Description

HaTeX is a Haskell implementation of LaTeX, with the aim to be a helpful tool to generate or parse LaTeX code.

From a global sight, it's composed of:

1. The `LaTeX` datatype, as an AST for LaTeX.
2. A set of combinators of LaTeX *blocks*.
3. A renderer of LaTeX code.
4. A parser of LaTeX code.
5. Methods to analyze the LaTeX AST.
6. A monadic implementation of combinators.
7. Methods for a subset of LaTeX packages.

What is new?

Since the release of the version 3 to the current 3.3, the most notable changes have been:

3.1 New module `Warnings`. Here we added methods to analyze a LaTeX AST.

3.2 Implemented the parser. Also support for greek letters and implementation of the `graphicx` package.

3.3 Tree rendering from a Haskell tree. A typeclass (`LaTeXC`) puts together monoid and monad interfaces.

Furthermore, now is available an open source user's guide.

Future plans

The next mission of HaTeX is to enhance what currently is. Fixing bugs, extend documentation, improve the guide, add useful functions.

Contact

If you are someway interested in this project, please, feel free to give any kind of opinion or idea, or to ask any question you have. A good place to take contact and stay tuned is the HaTeX mailing list:

`hatex <at> projects.haskell.org`

Of course, you always can mail to the maintainer.

Further reading

7.12.2 Haskell XML Toolbox

Report by:	Uwe Schmidt
Status:	eighth major release (current release: 9.3)

Description

The Haskell XML Toolbox (HXT) is a collection of tools for processing XML with Haskell. It is itself purely written in Haskell 98. The core component of the Haskell XML Toolbox is a validating XML-Parser that supports almost fully the Extensible Markup Language (XML) 1.0 (Second Edition). There is a validator based on DTDs and a new more powerful one for Relax NG schemas.

The Haskell XML Toolbox is based on the ideas of HaXml and HXML, but introduces a more general approach for processing XML with Haskell. The processing model is based on arrows. The arrow interface is more flexible than the filter approach taken in the earlier HXT versions and in HaXml. It is also safer; type checking of combinators becomes possible with the arrow approach.

HXT is partitioned into a collection of smaller packages: The core package is `hxt`. It contains a validating XML parser, an HTML parser, filters for manipulating XML/HTML and so called XML pickler for converting XML to and from native Haskell data.

Basic functionality for character handling and decoding is separated into the packages `hxt-charproperties` and `hxt-unicode`. These packages may be generally useful even for non XML projects.

HTTP access can be done with the help of the packages `hxt-http` for native Haskell HTTP access and `hxt-curl` via a libcurl binding. An alternative lazy non validating parser for XML and HTML can be found in `hxt-tagsoup`.

The XPath interpreter is in package `hxt-xpath`, the XSLT part in `hxt-xslt` and the Relax NG validator in `hxt-relaxng`. For checking the XML Schema Datatype definitions, also used with Relax NG, there is a separate and generally useful regex package `hxt-regex-xmlschema`.

The old HXT approach working with filter `hxt-filter` is still available, but currently only with hxt-8. It has not (yet) been updated to the hxt-9 mayor version.

Features

- Validating XML parser
- Very liberal HTML parser
- Lightweight lazy parser for XML/HTML based on Tagsoup (http://www.haskell.org/communities/05-2010/html/report.html#sect5.11.3)
- Binding to the expat parser via hexpat package
- Easy de-/serialization between native Haskell data and XML by pickler and pickler combinators
- XPath support
- Full Unicode support
- Support for XML namespaces
- Cabal package support for GHC
- HTTP access via Haskell bindings to libcurl and via Haskell HTTP package
- Tested with W3C XML validation suite
- Example programs
- Relax NG schema validator
- XML Schema validator (next release)
- Lightweight regex library with full support of Unicode and XML Schema Datatype regular expression syntax
- An HXT Cookbook for using the toolbox and the arrow interface
- Basic XSLT support
- GitHub repository with current development versions of all packages http://github.com/UweSchmidt/hxt

Current Work

The master thesis and project implementing an XML Schema validator started in October 2011 has been finished. The validator will be released in a separate module hxt-xmlschema. Integration with hxt has been prepared in hxt-9.3. The XML Schema datatype library has also been completed, all datatypes including date and time types are implemented. But there is still a need for testing the validator, especially with the W3C test suite. Hopefully testing will be done in the next few months. With the release of the schema validator the the master thesis will also be published on the HXT homepage. The current state of the validator can be found in the HXT repository on github.

Further reading

The Haskell XML Toolbox Web page (http://www.fh-wedel.de/~si/HXmlToolbox/index.html) includes links to downloads, documentation, and further information.

The latest development version of HXT can be found on github under (https://github.com/UweSchmidt/hxt).

A getting started tutorial about HXT is available in the Haskell Wiki (http://www.haskell.org/haskellwiki/HXT). The conversion between XML and native Haskell data types is described in another Wiki page (http://www.haskell.org/haskellwiki/HXT/Conversion_of_Haskell_data_from/to_XML).

7.12.3 epub-tools (Command-line epub Utilities)

Report by:	Dino Morelli
Status:	stable, actively developed

A suite of command-line utilities for creating and manipulating epub book files. Included are: epubmeta, epubname, epubzip.

epubmeta is a command-line utility for examining and editing epub book metadata. With it you can export, import and edit the raw OPF Package XML document for a given book. Or simply dump the metadata to stdout for viewing in a friendly format.

epubname is a command-line utility for renaming epub ebook files based on their OPF Package metadata. It tries to use author names and title info to construct a sensible name. epubname has recently undergone extensive redesign:

- ◦ Major change of the formatting rules system. Renaming machinery is now described in a domain-specific language, NOT in statically compiled code. Users are able to extend the functionality with custom naming rules in conf files.
- ◦ Added interactive mode to ask about each file rename as they happen, this is like darcs now!
- ◦ Added ability to specify target directory for books to be moved to as part of renaming.

epubzip is a handy utility for zipping up the files that comprise an epub into an .epub zip file. Using the same technology as epubname, it can try to make a meaningful filename for the book.

epub-tools is available from Hackage and the Darcs repository below.

Further reading

- ◦ Project page: http://ui3.info/d/proj/epub-tools.html
- ◦ Source repository: `darcs get` http://ui3.info/darcs/epub-tools

7.13 Natural Language Processing

7.13.1 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.

Recently released packages and projects

- ◦ *approx-rand-test* Approximate randomization test, eg. for testing whether differences in performance of parse disambiguation/fluency ranking models is significant or not. (https://github.com/danieldk/approx-rand-test)
- ◦ *GenI 0.22*: (a long overdue update), surface realiser for Natural Language Generation (https://projects.haskell.org/GenI) (Eric Kow)
- ◦ Latent Dirichlet Allocation, a hierarchical Bayesian admixture model commonly used for topic modeling and many other NLP applications (Grzegorz Chrupala)
  - – *lda* an experimental implementation of Latent Dirichlet Allocation (http://hackage.haskell.org/package/lda)
  - – *swift-lda* Gibbs sampler for Latent Dirichlet Allocation. The sampler can be used in an online as well as batch mode (http://hackage.haskell.org/package/swift-lda/).
  - – *colada* Colada implements incremental word class class induction using Latent Dirichlet Allocation (LDA) with a Gibbs sampler (http://hackage.haskell.org/package/colada):

New packages and projects in development

- ◦ *NubFinder*: Research project to develop technology to search and analyze user opinions on the Web. (https://sites.google.com/site/nubfinder)
- ◦ *alpinocorpus-server*: Server for the Alpino treebank library (https://github.com/danieldk/alpinocorpus-server) (Daniel de Kok)
- ◦ *alpinocorpus-haskell*: Haskell bindings for the Alpino treebank library. (https://github.com/danieldk/alpinocorpus-haskell) (Daniel de Kok)

At the present, the mailing list is mainly used to make announcements to the Haskell NLP community.

At the time of this writing, there is an ongoing Coursera online NLP class, for which some of list members have expressed an interest in doing the assingments in Haskell. 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.

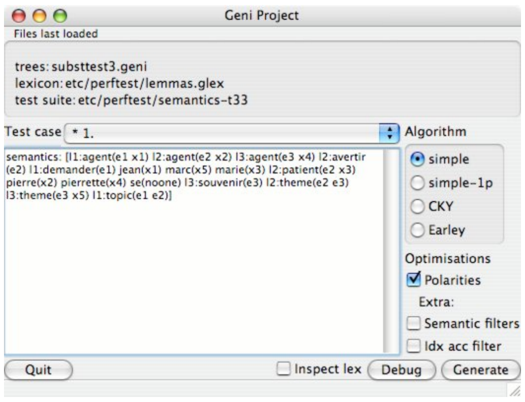
Further reading

http://projects.haskell.org/nlp

7.13.2 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.



Files last loaded

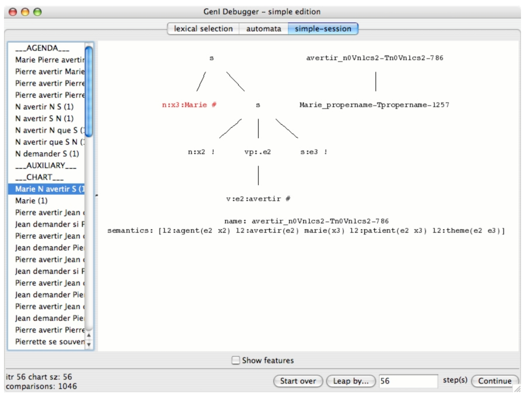
trees: substtest3.geni
lexicon: etc/perftest/lemmas.glex
test suite: etc/perftest/semantics-t33

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



___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_n0Vnlcs2-Tn0Vnlcs2-786
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

Leap by: 56

Since May 2011, Eric is working with Computational Linguistics Ltd and SRI international to develop new features for GenI and improve its scalability and performance for use in an interactive tutoring application. Most recently, we have released an long overdue update to GenI, featuring GHC 7 support, simpler installation, library cleanups, bugfixes, and a handful of new UI features.

GenI is available on Hackage, and can be installed via cabal-install. Our most recent release of GenI was version 0.22 (2012-04-22). For more information, please contact us on the geni-users mailing list.

Further reading

- 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

7.14 Machine Learning

7.14.1 Bayes-stack

Report by:	Ben Gamari
Participants:	Laura Dietz
Status:	stable, actively developed

Bayes-stack is a framework for inference on probabilistic graphical models. It supports hierarchical latent variable models, including Latent Dirichlet allocation and even more complex topic model derivatives. We focus on inference using blocked collapsed Gibbs sampling, but the framework is also suitable for other iterative update methods.

Bayes-stack is written for parallel environments running on multi-core machines. While many researchers see collapsed Gibbs sampling as a hindrance for parallelism, we embrace its robustness against mildly out-of-date state. In bayes-stack, a model is represented as blocks of jointly updated random variables. Each inference worker thread will repeatedly pick a block, fetch the current model state, and compute a new setting for its variables. It then pushes an update function to a thread responsible for updating the global state. This thread will accumulate state updates, committing them only periodically to manage memory bandwidth and cache pressure.

Unlike other approaches where sets of variables are evolved independently for several iterations, bayes-stack synchronizes the model state after only a few variables have been processed. This improves convergence properties while incurring minimal performance costs.

The project provides two packages. The core of the framework is contained in the `bayes-stack` package while `network-topic-models` demonstrates use of the framework, providing several topic model implementations. These include Latent Dirichlet Allocation

(LDA), the shared taste model for social network analysis, and the citation influence model for citation graphs.

Haskell's ability to capture abstraction without compromising performance has enabled us to preserve the purity of the model definition while safely utilizing concurrency. Tools like GHC's event log and Threadscope have been extremely helpful in evaluating the performance characteristics of the parallel sampler.

Currently our focus is on improving scalability of the inference. While our inference approach should allow us to find a reasonable trade-off between data-sharing and performance, much work still remains to realize this potential.

We thank Simon Marlow for both his discussions concerning parallel performance tuning with GHC as well as his continuing work in pushing forward the state of high-performance concurrency in Haskell. Furthermore, we are excited about work surrounding Threadscope by Duncan Coutts, Peter Wortmann, and others.

Further reading

- http://www.github.com/bgamari/bayes-stack
- http://www.cs.umass.edu/~dietz/delayer/

7.14.2 Homomorphic Machine Learning

Report by:	Mike Izbicki
Status:	preliminary

I have been exploring the algebraic properties of machine learning algorithms using Haskell. For example, the training of a Naive Bayes classifier turns out to be a semigroup homomorphism. This algebraic interpretation has two main advantages: First, all semigroup homomorphisms can be converted into an online and/or parallel algorithm for free using specially designed higher-order functions. Second, we can perform cross-validation on homomorphisms much faster than we can on non-homomorphic functions.

I am in the process of writing a prototype library for homomorphic learning called HLearn. Haskell was the natural choice for implementing the project due to its emphasis on algebra and its high performance. My goal is to have an initial release sometime in 2012. I can be contacted at ⟨mike@izbicki.me⟩.

7.15 Bioinformatics

7.15.1 ADPfusion

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

ADPfusion provides a domain-specific language (DSL) for the formulation of dynamic programs with a special emphasis on computational biology. 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.

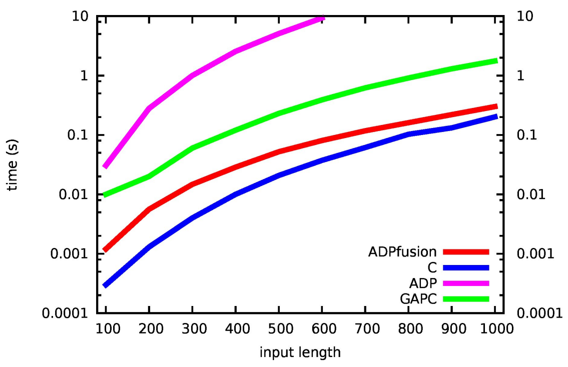
As an example, consider a grammar that recognizes palindromes. Given the non-terminal p, as well as parsers for single characters c and the empty input ϵ, the production rule for palindromes can be formulated as $p \rightarrow c\,p\,c \mid \epsilon$.

The corresponding ADPfusion code is similar:

```
(p, f <<< c % p % c ||| g <<< e ... h),
```

We need a number of combinators as "glue" and additional evaluation functions f, g, and h. With $f\ c_1\ p\ c_2 = p$ && $(c_1 \equiv c_2)$ scoring a candidate, $g\ e\ =$ `True`, and $h\ xs =$ `or` xs determining if the current substring is palindromic.

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 current (post-ICFP) code contains numerous improvements, including better grammar handling (fewer combinators!) and new possibilities for combining grammars and algebras.

Details can be found in the paper (ICFP'12 proceedings) with a preprint available on the ADPfusion homepage.

Further reading

- http://www.tbi.univie.ac.at/~choener/adpfusion
- http://hackage.haskell.org/package/ADPfusion
- http://dx.doi.org/10.1145/2364527.2364559

7.15.2 Biohaskell

Report by:	Ketil Malde
Participants:	Christian Höner zu Siederdissen, Nick Ignolia, Felipe Almeida Lessa, Dan Fornika, Maik Riechert, Ashish Agarwal, Grant Rotskoff

Bioinformatics in Haskell is a steadily growing field, and the *Bio* section on Hackage now contains 51 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, and to register and document their contributions on the http://biohaskell.org wiki.

Further reading

- http://biohaskell.org
- http://blog.malde.org
- http://www.tbi.univie.ac.at/~choener/haskell/
- http://adp-multi.ruhoh.com

7.16 Embedding DSLs for Low-Level Processing

7.16.1 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 developed in co-operation by Ericsson, Chalmers University of Technology (Göteborg, Sweden) and Eötvös Loránd (ELTE) University (Budapest, Hungary).

The motivating application of Feldspar is telecoms processing, but the language is intended to be useful for DSP in general. The aim is to allow DSP 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 an extensive library of numeric and array processing operations as well as a code generator producing C code for running on embedded targets.

The current version deals with the data-intensive numeric algorithms which are at the core of any DSP application. We have recently added support for the expression and compilation of parallel algorithms. As future work remains to extend the language to deal with interaction with the environment (e.g., processing of streaming data) and to support compilation to heterogeneous multi-core targets.

Further reading

- https://github.com/Feldspar/feldspar-language
- http://hackage.haskell.org/package/feldspar-language
- http://hackage.haskell.org/package/feldspar-compiler

7.16.2 Kansas Lava

Report by:	Andy Gill
Participants:	Andy Gill, 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.

Kansas Lava has undergone considerable changes over the last 12 months.

- The Patches idea (normalizing handshaking circuits) and the Wakarusa Monad (compiling state machines into untyped Fabrics) have been rejected. There was a PEPM'12 paper about Patches, including the design and implementation of a controller for an ST7066U-powered LCD display. Wakarusa was never written up.

- Patches and Wakarusa have been replaced with a unified enhancement of the Fabric monad. 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 Kansas Lava, we are developing Kansas Lava Cores. In hardware, a core is a component that can be realized as a circuit, typically on an FPGA. Kansas Lava Cores contains about a dozen cores, and basic board support for Spartan3e, as well as a high-fidelity emulator for the Spartan3e. The cores and the simulator has been rewritten to use the new Fabric and new state-machine generation monad.

- Using various components provided as Kansas Lava Cores, we continue developing the λ-bridge with implementations (in Haskell and Kansas Lava) of a simple protocol stack for communicating with FPGAs. This bridge is based on the best-effort, unreliable, but acknowledgment-centric access to an 8-bit WISHBONE-compliant hardware bus, and idempotent transaction requests. This has the advantage of moving the implementation of the complexity of providing reliable communications into the Haskell code. The net effect is that programmers can use Haskell to talk to real hardware over multiple possible physical connections (RS232, RJ-45, USB, etc), and communicate via standard Haskell idioms like file Handles, Chans, and MVars.

- Finally, with Iavor Diatchki (Galois), we are reworking our `sized-types` library to use the new kind `Nat` in GHC 7.6.

Further reading

- http://www.ittc.ku.edu/csdl/fpg/Tools/KansasLava
- http://www.youtube.com/playlist?list=PL211F8711E3B3DF9C

7.17 Others

7.17.1 Clckwrks

Report by:	Jeremy Shaw

clckwrks (pronounced "clockworks") is a blogging and content management system (CMS). It is intended to compete directly with popular PHP-based systems. Pages and posts are written in markdown and can be edited directly in the browser. The system can be extended via plugins and themes packages.

At present, clckwrks is still alpha, and requires Haskell knowledge to install and configure. However, the goal is to create an end user system that requires zero Haskell knowledge. It will be possible to one-click install plugins and themes and perform all other administrative functions via the browser.

Future plans

We are currently focused on four tasks:

1. Overhaul of the plugin system to support one-click installation of plugins and themes
2. Improvements to the user experience in the core blogging and page editing functionality
3. Simplifying installation
4. Improved documentation

Once the core is solid, we will focus development efforts on creating plugins to extend the core functionality.

Further reading

http://www.clckwrks.com/

7.17.2 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 2012 of leapseconds-announced contains all leap seconds up to 2012-07-01. A new version will be uploaded if/when the IERS announces a new leap second.

Further reading

- http://hackage.haskell.org/cgi-bin/hackage-scripts/package/leapseconds-announced
- http://github.com/bjornbm/leapseconds-announced

7.17.3 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!

By now, the data collected by some arbtt users has become quite large. This awoke the dormant development and the newly released version 0.6.4 sports processing in constant memory and faster time-related functions.

Further reading

- http://www.joachim-breitner.de/projects#arbtt

- http://www.joachim-breitner.de/blog/archives/336-The-Automatic-Rule-Based-Time-Tracker.html
- http://darcs.nomeata.de/arbtt/doc/users_guide/

7.17.4 sshtun (Wrapper daemon to manage an ssh tunnel)

Report by:	Dino Morelli
Status:	experimental, actively developed

This is a daemon that executes an ssh command to form a secure tunnel and then blocks on it. If the tunnel goes down, sshtun can attempt to reestablish it. It can also be set up to monitor a file on an http server to determine if the tunnel should be up or not, so you can switch it on or off remotely.

sshtun is available from Hackage and the Darcs repository below.

Further reading

- Project page: http://ui3.info/d/proj/sshtun.html
- Source repository: `darcs get` http://ui3.info/darcs/sshtun

7.17.5 hMollom — Haskell implementation of the Mollom API

Report by:	Andy Georges
Status:	active

Mollom (http://mollom.com) is a anti-comment-spam service, running in the cloud. The service can be used for free (limited number of requests per day) or paid, with full support. The service offers a REST based API (http://mollom.com/api/rest). Several libraries are offered freely on the Mollom website, for various languages and web frameworks – PHP, Python, Drupal, etc.

hMollom is an implementation of this API, communicating with the Mollom service for each API call that is made and returning the response as a Haskell data type, along with some error checking.

hMollom is currently under active development. The current release targets the Mollom REST API. We carefully track new developments in the Mollom API.

The development happens on GitHub, see http://github.com/itkovian/hMollom, packages are put on Hackage.

Further reading

http://github.com/itkovian/hMollom

7.17.6 hGelf — Haskell implementation of the Graylog extended logging format

Report by:	Andy Georges
Status:	active

Graylog (http://graylog2.org) is a log management framework that allows setting up event log monitoring and anlysis through various tools. The logging format used is GELF — The GrayLog Extended Logging Format.

At the moment of writing hGelf, there was no Haskell package available on Hackage that allows wrapping log messages in this format. hGelf aimed to fill this void.

The development of hGelf happens on GitHub, see https://github.com/itkovian/hGelf, packages are put on Hackage.

Further reading

http://github.com/itkovian/hGelf

7.17.7 Galois Open-Source Projects on GitHub

Report by:	Jason Dagit
Status:	active

Galois is pleased to announce the movement of our open source projects to GitHub!

As part of our commitment to giving back to the open source community, we have decided that we can best publish our work using GitHub's public website. This move should provide the open source community more direct access to our repositories, as well as more advanced collaboration tools.

Moved repositories include the widely-used XML and JSON libraries, our FiveUI extensible UI Analysis tool, our high-speed Cereal serialization library, our SHA and RSA crypto packages, the HaLVM, and more. For a list of our open source packages, please see our main GitHub page here: https://github.com/galoisinc

We are very excited to interact with the GitHub community and utilize all the great tools there. On the other hand, if you're not a GitHub user, please feel free to continue to send us any patches or suggestions as per usual.

For those currently hacking on projects using our old repositories at `code.galois.com`, we apologize for the inconvenience! The trees on GitHub hold the exact same trees, however, so you should be able to add a remote tree (git remote add) and push without too much difficulty.

8 Commercial Users

8.1 Well-Typed LLP

Report by:	Ian Lynagh
Participants:	Andres Löh, Duncan Coutts

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 are working for a variety of commercial clients, but naturally, only some of our projects are publically visible.

We continue to be involved in the development and maintenance of GHC (→ 3.2). Since the last HCAR, we have put out the 7.4.2 patch release as well as 7.6.1, the first release of a new stable branch. We are expecting to put out a 7.6.2 bug-fix release in the not too distant future, as well as a 7.8.1 release with the latest goodies.

On behalf of the Industrial Haskell Group (IHG) (→ 8.3), we have now completed the 64bit Windows port of GHC, and it was released as part of GHC 7.6.1. We have now turned our attention to the Hackage server, and are working on getting the new Hackage 2 implementation to the point where it is usable as the central hackage.haskell.org server.

The Parallel GHC Project (→ 5.1.3) is drawing to a close now. Information about the partner organisations' projects can be found on the wiki page (http://www.haskell.org/haskellwiki/Parallel_GHC_Project), and will also be presented in forthcoming talks and publications. There have also been a number of useful off-shoots of the project, most notably a much improved ThreadScope application (for debugging performance of concurrent and parallel programs), and a new full implementation of Cloud Haskell (an Erlang-like system for Haskell, now ready for early adopters).

We continue to be involved in the community, maintaining several packages on Hackage and giving talks at a number of conferences. More recently, we have partnered with Skills Matter to offer public beginners and advanced Haskell courses on a quarterly basis. See the "Training" section of our website for more details.

We are of course always looking for new clients and projects, so if you have something we could help you with, just drop us an e-mail.

Further reading

- http://www.well-typed.com/
- Blog: http://blog.well-typed.com/

8.2 Bluespec Tools for Design of Complex Chips and Hardware Accelerators

Report by:	Rishiyur Nikhil
Status:	commercial product

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. All hardware behavior is expressed using *rewrite rules* (Guarded Atomic Actions). BSV borrows many ideas from Haskell — algebraic types, polymorphism, type classes (overloading), and higher-order functions. Strong static checking extends into correct expression of multiple clock domains, and to gated clocks for power management. BSV is universally applicable, from algorithmic "datapath" blocks to complex control blocks such as processors, DMAs, interconnects, and caches.

Bluespec's core tool synthesizes (compiles) BSV into high-quality Verilog, which can be further synthesized into netlists for ASICs and FPGAs using third-party tools. Atomic transactions enable design-by-refinement, where an initial executable approximate design is systematically transformed into a quality implementation by successively adding functionality and architectural detail. The synthesis tool is implemented in Haskell (well over 100K lines).

Bluesim is a fast simulation tool for BSV. There are extensive libraries and infrastructure to make it easy to build FPGA-based accelerators for compute-intensive software, including for the Xilinx XUPv6 board popular in universities, and the Convey HC-1 high performance computer.

BSV is also enabling the next generation of computer architecture education and research. Students implement and explore architectural models on FPGAs, whose speed permits evaluation using whole-system software.

Status and availability

BSV tools, available since 2004, are in use by several major semiconductor and electronic equipment companies, and universities. The tools are free for academic teaching and research.

Further reading

- *Abstraction in Hardware System Design*, R.S. Nikhil, in *Communications of the ACM*, 54:10, October 2011, pp. 36-44.
- *Bluespec, a General-Purpose Approach to High-Level Synthesis Based on Parallel Atomic Transactions*, R.S. Nikhil, in *High Level Synthesis: from Algorithm to Digital Circuit, Philippe Coussy and Adam Morawiec (editors)*, Springer, 2008, pp. 129-146.
- *BSV by Example*, R.S. Nikhil and K. Czeck, 2010, book available on Amazon.com.
- 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.
- http://www.bluespec.com/products/BluDACu.htm: A fun example with many functional programming features — BluDACu, a parameterized Bluespec hardware implementation of Sudoku.

8.3 Industrial Haskell Group

Report by:	Andres Löh
Participants:	Duncan Coutts, Ian Lynagh

The Industrial Haskell Group (IHG) is an organization to support the needs of commercial users of Haskell.

The main activity of the IHG is to fund work on the Haskell development platform. It currently operates two schemes:

- The collaborative development scheme pools resources from full members in order to fund specific development projects to their mutual benefit.
- Associate and academic members contribute to a separate fund which is used for maintenance and development work that benefits the members and community in general.

In the past six months, the collaborative development scheme funded work on a Win64 port of GHC and on the new Hackage server (→ 6.3.1).

The Win64 port of GHC is now complete and released (as a part of GHC 7.6). There's an installer available for the 64-bit Windows version, so it should be easy enough to get started.

The main focus of current IHG work is the transition to the new Hackage server. In a first phase, we have been fixing a number of outstanding bugs and memory issues with the Hackage server. We are now confident that Hackage 2 is stable enough to go into production. We are continuing to work on a number of issues such as somewhat improved security (per-package maintainer groups that restrict who can upload new versions of a package), a good backup infrastructure, and providing a smooth migration from Hackage to Hackage 2. Among other things, we can now import all of the old account data into the new Hackage sever.

Snapshots of the new server are intermittently being made available at http://new-hackage.haskell.org.

Details of the tasks undertaken are appearing on the Well-Typed (→ 8.1) blog, on the IHG status page and on standard communication channels such as the Haskell mailing list.

The collaborative development scheme is running continuously, so if you are interested in joining as a member, please get in touch. Details of the different membership options (full, associate, or academic) can be found on the website.

We are particularly interested in new members who might be willing to fund further efforts on Cabal and Hackage.

If you are interested in joining the IHG, or if you just have any comments, please drop us an e-mail at ⟨info@industry.haskell.org⟩.

Further reading

- http://industry.haskell.org/
- http://industry.haskell.org/status/

8.4 Barclays Capital

Report by:	Ben Moseley

Barclays Capital has been using Haskell as the basis for our FPF (Functional Payout Framework) project for about seven years now. The project develops a DSL and associated tools for describing and processing exotic equity options. FPF is much more than just a payoff language — a major objective of the project is not just pricing but "zero-touch" management of the entire trade lifecycle through automated processing and analytic tools.

For the first half of its life the project focused only on the most exotic options — those which were too complicated for the legacy systems to handle. Over the past few years however, FPF has expanded to provide the trade representation and tooling for the vast majority of our equity exotics trades and with that the team has grown significantly in both size and geographical distribution. We now have eight permanent full-time Haskell developers spread between Hong Kong, Kiev and London (with the latter being the biggest development hub).

Our main front-end language is currently a deeply embedded DSL which has proved very successful, but we have recently been working on a new non-embedded implementation. This will allow us to bypass some of the traditional DSEL limitations (e.g., error messages and syntactical restrictions) whilst addressing some business areas which have historically been problematic. The new language is based heavily on arrows, but has a custom (restrictive but hopefully easier-to-use than raw arrow-notation) syntax. We are using a compiler from our custom DSL syntax into Haskell source (with standard transformers from Ross Paterson's "arrows" package) to provide the semantics for the language but plan to develop a number of independent backends. Our hope is that, over time, this will gradually replace our embedded DSL as the front end for all our tools. For the parsing part of this work we have been very impressed by Doaitse Swierstra's uu-parsinglib (→ 7.3.2).

We have been and remain very satisfied GHC users and feel that it would have been significantly harder to develop our systems in any other current language.

8.5 Oblomov Systems

Report by: Martijn Schrage

See: http://www.haskell.org/communities/11-2011/html/report.html#sect9.6.

8.6 madvertise Mobile Advertising

Report by: Adam Drake

madvertise Mobile Advertising, GmbH is Europe's leading marketplace for mobile app and web advertising, with traffic frequencies of up to 25.000 requests per second. madvertise was founded in 2009 and the recent purchase of Turkish mobile advertising firm Mobilike has raised the number of employees at madvertise to approximately 95.

Haskell is used in the Research and Data Science group at madvertise, especially to tackle problems in large scale data analysis and machine learning. One example of our use of Haskell is in the initial design for a real-time bidding system for ad impressions, including optimizations for publisher revenue and liquidity management. Such a system must support a high level of concurrency as each ad request results in a full-cycle auction taking place, and Haskell excels in such an environment. Another example of our usage of Haskell is in the toolchain for constructing a system to measure and act upon information theoretic entropy for high-frequency data in a real-time fashion.

Haskell is used at madvertise as a general purpose language that is preferred for making full use of multicore hardware, providing code correctness, and for providing clarity and stability through the type system. We plan to continue to use Haskell where appropriate, including the possibility of production systems in the future, and to open-source as many of our tools as possible.

Further reading

http://madvertise.com

8.7 OpenBrain Ltd.

Report by: Tom Nielsen

OpenBrain Ltd. is developing a new platform for statistical computing that enables optimal decisions taking into account all the available information. We have developed a new statistical programming language (BAYSIG) that augments a Haskell-like functional programming language with Bayesian inference and first-class ordinary and stochastic differential equations. BAYSIG is designed to support a declarative style of programming where almost all the work consists in building probabilistic models of observed data. Data analysis, risk assessment, decision, hypothesis testing and optimal control procedures are all derived mechanically from the definition of these models. We are targeting a range of application areas, including financial, clinical and life sciences data.

We are building a web application (http://BayesHive.com) to make this platform accessible to a wide range of users. Users can upload and analyse varied types of data using a point-and-click interface. Models and analyses are collected in literate programming-like documents that can be published by users as blogs.

We use Haskell for almost all aspects of implementing this platform. The BAYSIG compiler is written in Haskell, which is particularly well suited for implementing the recursive syntactical transformations underlying statistical inference. `BayesHive.com` is being developed in Yesod.

Contact

⟨tomn@openbrain.org⟩

Further reading

http://BayesHive.com

9 Research and User Groups

9.1 Haskell at Eötvös Loránd University (ELTE), Budapest

Report by:	PÁLI Gábor János
Status:	ongoing

Haskell
at ELTE

Education

There are many different courses on Haskell and Agda that run at Eötvös Loránd University, Faculty of Informatics.

◦ Programming for first-year BSc students using Haskell, it is officially in the curriculum. It is also taught for foreign language students as part of their program.

◦ Advanced functional programming using Haskell, it is an optional course for BSc and MSc students.

◦ Programming in Agda as an optional course for BSc and MSc students.

◦ Other Haskell-related courses on Lambda Calculus, Type Theory and Implementation of Functional Languages.

There is an interactive online evaluation and testing system, called ActiveHs. It contains several hundred systematized exercises and it may be also used as a teaching aid. There is also some experimenting going on about supporting SVG graphics, and extending the embedded interpreter and testing environment with safe emulation of IO values, providing support for Agda. ActiveHs is now also avaiable on Hackage.

We have been translating our course materials to English, some of the materials is already available.

Further reading

◦ Haskell course materials (in English): http://pnyf.inf.elte.hu/fp/Overview_en.xml
◦ Agda course materials (in English): http://pnyf.inf.elte.hu/fp/Index_a.xml
◦ ActiveHs: http://hackage.haskell.org/package/activehs

9.2 Artificial Intelligence and Software Technology at Goethe-University Frankfurt

Report by:	David Sabel
Participants:	Conrad Rau, Manfred Schmidt-Schauß

Semantics of programming languages. One of our research interests focuses on programming language semantics, especially on contextual equivalence and bisimilarity. Deterministic call-by-need lambda calculi with letrec provide a semantics for the core language of Haskell. For such extended lambda calculi we proved correctness of strictness analysis using abstract reduction, equivalence of the call-by-name and call-by-need semantics, and completeness of applicative bisimilarity w.r.t. contextual equivalence. We also explored several nondeterministic extensions of call-by-need lambda calculi and their applications. An important result is that for calculi with `letrec` and nondeterminism usual definitions of applicative similarity are unsound w.r.t. contextual equivalence.

A recent topic are core languages of concurrent programming languages, like Concurrent Haskell. We analyzed a higher-order functional language with concurrent threads, monadic IO, synchronizing variables and implicit, monadic, and concurrent futures. Using contextual equivalence based on may- and should-convergence, we have shown that several transformations preserve program equivalence. We also proved correctness of a Sestoft-like abstract machine for this language. An important result is that the language with concurrency conservatively extends the pure core language of Haskell, i.e. all program equivalences for the pure part also hold in the concurrent language.

Most recently, we analyzed Software Transactional Memory in a concurrent calculus with futures. An obviously correct big-step-semantics was used as a specification to show correctness of a highly concurrent implementation.

An ongoing project tries to automate correctness proofs of program transformations. These proofs require to compute so-called forking and commuting diagrams. We implemented an algorithm as a combination of several unification algorithms in Haskell which computes these diagrams. To conclude the correctness proofs we automated the corresponding induction proofs (which use the diagrams) using automated termination provers for term rewriting systems.

Grammar based compression. Another research topic of our group focuses on algorithms on grammar compressed strings and trees. One goal is to reconstruct known algorithms on strings and terms (unification, matching, rewriting etc.) for their use on grammars without prior decompression. We implemented several of those algorithms in Haskell which are available as a Cabal package.

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. This September we welcomed Scott Owens as a new group member and lecturer. We are a group of staff and students with shared interests in functional programming. While our work is not limited to Haskell — in particular our interest in Erlang has been growing — Haskell provides a major focus and common language for teaching and research.

Our members pursue a variety of Haskell-related projects, some of which are reported in other sections of this report, such as Simon Thompson's text book *Haskell: the craft of functional programming*. At the Haskell Implementors Workshop 2012 Thomas Schilling presented his work on trace-based dynamic optimisations for Haskell programs. Olaf Chitil presented at ICFP 2012 his practial lazy contracts for Haskell, which are available from Hackage. He also maintains the Haskell IDE Heat. Recently he resurrected the Haskell tracer Hat (→ 6.4.2). Hat is now on Hackage and new developments have started.

We are always looking for PhD students to work with us. We are particularly keen to recruit students interested in programming tools for 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.

Further reading

- PLAS group: http://www.cs.kent.ac.uk/research/groups/plas/
- Haskell: the craft of functional programming: http://www.haskellcraft.com
- Refactoring Functional Programs: http://www.cs.kent.ac.uk/research/groups/plas/hare.html
- A trace-based just-in-time compiler for Haskell http://www.youtube.com/watch?v=PtEcLs2t9Ws
- Scion: http://code.google.com/p/scion-lib/
- Hat, the Haskell Tracer: http://olafchitil.github.com/hat
- Practial Lazy Typed Contracts for Haskell: http://www.cs.kent.ac.uk/~oc/contracts.html
- Heat: http://www.cs.kent.ac.uk/projects/heat/

9.4 Formal Methods at DFKI and University Bremen

Report by:	Christian Maeder
Participants:	Mihai Codescu, Dominik Dietrich, Christoph Lüth, Till Mossakowski
Status:	active development

The activities of our group center on formal methods, covering a variety of formal languages and also translations and heterogeneous combinations of these.

We are using the Glasgow Haskell Compiler and many of its extensions to develop the Heterogeneous tool set (Hets). Hets consists of parsers, static analyzers, and proof tools for languages from the CASL family, such as the Common Algebraic Specification Language (CASL) itself (which provides many-sorted first-order logic with partiality, subsorting and induction), HasCASL, CoCASL, CspCASL, and an extended modal logic based on CASL.

Other languages supported include Isabelle, QBF, Maude, VSE, TPTP, THF, FPL (logic of functional programs), LF type theory and still Haskell (via Programatica). More recently, description logics like OWL, RDF, Common Logic, and DOL (the Distributed Ontology Language) have been integrated.

The user interface of the Hets implementation (about 200K lines of Haskell code) is based on some old Haskell sources such as bindings to uDrawGraph (formerly Davinci) and Tcl/TK that we maintain and also gtk2hs (→ 7.8.1), but we are moving to a mere web interface based on warp (→ 5.2.2).

HasCASL is a general-purpose higher-order language which is in particular suited for the specification and development of functional programs; Hets also contains a translation from an executable HasCASL subset to Haskell. There is a prototypical translation of a subset of Haskell to Isabelle/HOL.

Further reading

- Group activities overview:
  http://www.informatik.uni-bremen.de/agbkb/forschung/formal_methods/
- CASL specification language:
  http://www.cofi.info
- Heterogeneous tool set:
  http://www.dfki.de/cps/hets
  http://www.informatik.uni-bremen.de/htk/
  http://www.informatik.uni-bremen.de/uDrawGraph/

9.5 Haskell at Universiteit Gent, Belgium

Report by: Tom Schrijvers
Participants: Steven Keuchel

Haskell is one of the main research topics of the new Programming Languages Group at the Department of Applied Mathematics and Computer Science at the University of Ghent, Belgium.

Teaching. UGent is a great place for Haskell-aficionados:

- Make Haskell part of your studies with the elective course *Functional and Logic Programming Languages*.
- Explore the theory behind Haskell in the new master course on *Programming Language Fundamentals*.
- Explore Haskell in depth with one of our Haskell master thesis topics.
- Attend the thriving Ghent Functional Programming Group (→ 9.10).

Research. Haskell-related projects of the group members and collaborators are:

- *Meta-Theory à la Carte*:

  Formalizing meta-theory, or proofs about programming languages, in a proof assistant has many well-known benefits. However, the considerable effort involved in mechanizing proofs has prevented it from becoming standard practice. This cost can be amortized by reusing as much of an existing formalization as possible when building a new language or extending an existing one. Unfortunately reuse of components is typically ad-hoc, with the language designer cutting and pasting existing definitions and proofs, and expending considerable effort to patch up the results.

  This work presents a more structured approach to the reuse of formalizations of programming language semantics through the composition of modular definitions and proofs. The key contribution is the development of an approach to induction for extensible Church encodings which uses a novel reinterpretation of the universal property of folds. These encodings provide the foundation for a framework, formalized in Coq, which uses type classes to automate the composition of proofs from modular components.

  Several interesting language features, including binders and general recursion, illustrate the capabilities of our framework. We reuse these features to build fully mechanized definitions and proofs for a number of languages, including a version of mini-ML. Bounded induction enables proofs of properties for non-inductive semantic functions, and mediating type classes enable proof adaptation for more feature-rich languages.

  This is joint work with Ben Delaware and Bruno Oliveira.

- *Generic Conversions of Abstract Syntax Representations*:

  This work presents a datatype-generic approach to syntax with variable binding. A universe specifies the binding and scoping structure of object languages, including binders that bind multiple variables as well as sequential and recursive scoping. Two interpretations of the universe are given: one based on parametric higher-order abstract syntax and one on well-typed de Bruijn indices. The former provides convenient interfaces to embedded domain-specific languages, but is awkward to analyse and manipulate directly, while the latter is a convenient representation in implementations, but is unusable as a surface language. We show how to generically convert from the parametric HOAS interpretation to the de Bruijn interpretation thereby taking the pain from DSL developer to write the conversion themselves.

  This is joint work with Johan Jeuring.

- *Modular Reasoning about Incremental Programming*:

  Incremental Programming (IP) is a programming style in which new program components are defined as increments of other components. Examples of IP mechanisms include: *Object-oriented programming* (OOP) inheritance, *aspect-oriented programming* (AOP) advice and *feature-oriented programming* (FOP). A characteristic of IP mechanisms is that, while individual components can be independently defined, the composition of components makes those components become tightly coupled, sharing both control and data flows. This makes reasoning about IP mechanisms a notoriously hard problem: *modular reasoning* about a component becomes very difficult; and it is very hard to tell if two tightly coupled components *interfere* with each other's control and data flows.

  This work presents *modular reasoning about interference* (MRI), a *purely functional* model of IP embedded in Haskell. MRI models inheritance with mixins and side-effects with monads. It comes with a range of powerful reasoning techniques: equational reasoning, parametricity and reasoning with algebraic laws about effectful operations. These techniques enable modular reasoning about interference in the presence of side-effects.

  MRI formally captures *harmlessness*, a hard-to-formalize notion in the interference literature, in two theorems. We prove these theorems with a non-trivial combination of all three reasoning techniques.

This is joint work with Bruno Oliveira and William Cook.

- *Search Combinators*:

  Search heuristics often make all the difference between effectively solving a combinatorial problem and utter failure. Hence, the ability to swiftly design search heuristics that are tailored towards a problem domain is essential to performance improvement. In other words, this calls for a high-level domain-specific language (DSL).

  The tough technical challenge we face when designing a DSL for search heuristics, is to bridge the gap between a conceptually simple specification language (high-level, purely functional and naturally compositional) and an efficient implementation (typically low-level, imperative and highly non-modular). We overcome this challenge with a systematic approach in Haskell that disentangles different primitive concepts into separate monadic modular mixin components, each of which corresponds to a feature in the high-level DSL. The great advantage of mixin components to provide a semantics for our DSL is its modular extensibility.

  This is joint work with Guido Tack, Pieter Wuille, Horst Samulowitz and Peter Stuckey, following up on *Monadic Constraint Programming*, a monadic DSL for Constraint Programming in Haskell.

Further reading

- http://users.ugent.be/~tschrijv/haskell.html
- http://users.ugent.be/~tschrijv/SearchCombinators/
- http://hackage.haskell.org/package/Monatron
- http://hackage.haskell.org/package/monadiccp

9.6 fp-syd: Functional Programming in Sydney, Australia

Report by:	Erik de Castro Lopo
Participants:	Ben Lippmeier, Shane Stephens, and others

We are a seminar and social group for people in Sydney, Australia, interested in Functional Programming and related fields. Members of the group include users of Haskell, Ocaml, LISP, Scala, F#, Scheme and others. We have 10 meetings per year (Feb–Nov) and meet on the third Thursday of each month. We regularly get 20–30 attendees, with a 70/30 industry/research split. Talks this year have included material on compilers, theorem proving, type systems, Haskell web programming, OCaml and Jocaml. We usually have about 90 mins of talks, starting at 6:30pm, then go for drinks afterwards. All welcome.

Further reading

- http://groups.google.com/group/fp-syd
- http://fp-syd.ouroborus.net/
- http://fp-syd.ouroborus.net/wiki/Past/2012

9.7 Functional Programming at Chalmers

Report by:	Jean-Philippe Bernardy

Functional Programming is an important component of the Department of Computer Science and Engineering at Chalmers. In particular, Haskell has a very important place, as it is used as the vehicle for teaching and numerous projects. Besides functional programming, language technology, and in particular domain specific languages is a common aspect in our projects.

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; tools and related techniques:

- PULSE, the ProTest User-Level Scheduler for Erlang, which has been used to find race conditions in industrial software.

- We have shown how to sucessfully apply QuickCheck to polymorphic properties: http://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=99387.

Natural language technology. Grammatical Framework (http://www.haskell.org/communities/11-2010/html/report.html#sect9.7.3) 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.

Parser generator and template-haskell. BNFC-meta is an embedded parser generator, presented at the Haskell Symposium 2011. Like the BNF Converter, it generates a compiler front end in Haskell. Two aspects distinguish BNFC-meta from BNFC and other parser generators:
- BNFC-meta is not a program but a library (the parser description is embedded in a quasi-quote).
- BNFC-meta automatically provides quasi-quotes for the specified language. This includes a powerful and flexible facility for anti-quotation.

More info: http://hackage.haskell.org/package/BNFC-meta.

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 "Proofs for Free" (extensions of the parametricity & dependent types work from ICFP 2010, now published in JFP 2012). Patrik Jansson leads a work-package on DSLs within the EU project "Global Systems Dynamics and Policy" (http://www.gsdp.eu/, started Oct. 2010). If you want to apply DSLs, Haskell, and Agda to help modelling Global Systems Science, please get in touch! Jansson and Bernardy have also just started a new project called "Strongly Typed Libraries for Programs and Proofs".

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. We encourage you to experiment with programs and proofs in Agda as a "dependently typed Haskell".

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

- **Feldspar** (→ 7.16.1) is a domain-specific language for digital signal processing (DSP), developed in co-operation by Ericsson, Chalmers FP group and Eötvös Loránd (ELTE) University in Budapest.
- **Obsidian** is a language for data-parallel programming targeting GPGPUs.

The following languages are not actively developed at the moment:

- **Lava** is a language for structural hardware description. Circuits are modeled as ordinary Haskell functions, and many of Haskell's advantages (such as higher-order functions and polymorphism) are also available for Lava descriptions. There are several versions of Lava around. The version developed at Chalmers aims particularly at supporting formal verification in a convenient way.
- **Wired** is an extension to Lava, targeting (not exclusively) semi-custom VLSI design. A particular aim of Wired is to give the designer more control over on-chip wires' effects on performance. The most recent activity was to use Wired to explore the layout of multipliers (Kasyab P. Subramaniyan, Emil Axelsson, Mary Sheeran and Per Larsson-Edefors. Layout Exploration of Geometrically Accurate Arithmetic Circuits. *Proceedings of IEEE International Conference of Electronics, Circuits and Systems.* 2009). Home page: http://www.cse.chalmers.se/~emax/wired/.

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 analyzing 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 the draft paper http://web.student.chalmers.se/~danr/hipspec-atx.pdf

Teaching. Haskell is present in the curriculum as early as the first year of the Bachelors program. We have four courses solely dedicated to functional programming (of which three are Masters-level courses), but we also provide courses which use Haskell for teaching other aspects of computer science, such as programming languages, compiler construction, hardware description and verification, data structures and programming paradigms.

9.8 Functional Programming at KU

Report by:	Andy Gill
Status:	ongoing

Functional Programming is vibrant 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 Andy Gill) together form the core functional programming initiative at KU. Apart from Kansas Lava (→ 7.16.2) and HERMIT (→ 7.5.1), there are several other FP and Haskell related things going on, primarily in the area of web technologies.

We are interested in providing better support for interactive applications in Haskell by building on top of existing web technologies, like the fast Chrome browser, HTML5, and JavaScript. This is motivated partly by having easy tools to interactively teach programming in Haskell, and partly by the needs of the HERMIT (→ 7.5.1) project.

Towards this, we have developed a lightweight web framework called `Scotty`. Modeled after Ruby's popular Sinatra framework, Scotty is intended to be a cheap and cheerful way to write RESTful, declarative web applications. Scotty borrows heavily from the Yesod (→ 5.2.6) ecosystem, conforming to the WAI (→ 5.2.1) interface and using the fast Warp (→ 5.2.2) web server by default. More information can be found at the link below.

On top of `Scotty`, we are building `sunroof`, a deeply-embedded Javascript compiler, allowing for the handling of arbitrary asynchronous Javascript events directly on the browser. The initial design and implementation was written up in XLDI'12.

Finally, in August 2012 Nicolas Frisby successfully defended his PhD, and plans to start an internship at MSR Cambridge early next year.

Further reading

- The Functional Programming Group: http://www.ittc.ku.edu/csdl/fpg
- CSDL website: https://wiki.ittc.ku.edu/csdl/Main_Page
- http://www.ittc.ku.edu/csdl/fpg/Tools/Scotty
- http://www.ittc.ku.edu/csdl/fpg/Tools/BlankCanvas

9.9 San Simón Haskell Community

Report by:	Antonio Mamani
Participants:	Carlos Gomez

The San Simón Haskell Community from San Simón University Cochabamba-Bolivia, is an informal Spanish group that aims to learn, share information, knowledge and experience related to the functional paradigm.

On October last year, we participated on the XVIII National Congress of Computer Science of Bolivia (Congreso Nacional de Ciencias de la Computacíń de Bolivia), in which we organized two special activities: a Journal in Functional Programming (We had a very good introduction to functional paradigm and haskell [Msc. Vladimir Costas] and many short talks about the benefits of knowing Haskell and other functional languages [members of San Simon Haskell Community]) and the 2nd Open House Haskell Community (We showed some of the projects we were working on).

Projects in the 2nd Open House Haskell Community:

1. **L-System** — Application that renders an L-System using wxHaskell (http://hackage.haskell.org/package/lsystem) [Carlos Gomez]

2. **Compiler IDL - Java** — Generate code from IDL to Java. [Richard Jaldin]

3. **Mini Java** — Mini-Java compiler from scratch. [Antonio Mamani]

4. **3S Functional Web Browser** — Bachelor theses project about experimenting the implementation of a web browser with Haskell. (http://hsbrowser.wordpress.com/3s-functional-web-browser/) [Carlos Gomez]

This year, we are planning to organize the 2nd local Haskell Hackathon and the 3rd Open House Haskell Community. That's all for now, see you on facebook.

9.10 Ghent Functional Programming Group

Report by:	Andy Georges
Participants:	Jeroen Janssen, Tom Schrijvers, Jasper Van der Jeugt
Status:	active

The Ghent Functional Programming Group is a user group aiming to bring together programmers, academics, and others interested in functional programming located in the area of Ghent, Belgium. Our goal is to have regular meetings with talks on functional programming, organize functional programming related events such as hackathons, and to promote functional programming in Ghent by giving after-hours tutorials. While we are open to all functional languages, quite frequently, the focus is on Haskell, since most attendees are familiar with this language. The group has been active for two and a half years, holding meetings on a regular basis.

We have reported in previous HCARs on the first eleven meetings. Since May 2012, we had a single meeting. The GhentFPG #12 meeting took place on May 8, 2012 and involved two talks.

- Tom Schrijvers — Discussion on the Flemish Programming Contest 2012, with a focus on using the right Haskell data types for solving several of the given problems.
- Jasper Van der Jeugt — Tutorial on parallelisation in Haskell.

The attendance at the meetings usually varies between 10 to 15 people. We do have a number of Ghent University students attending. However, due to a shift in venue, the attendence has dropped slighty.

The plans for the fall 2012 Hackathon have shifted due to busy schedules of the GhentFPG organisers. In this academic year, we do plan to review the approach used during the meetings, because talks seem to attract more attendees compared to problem solving or coding events.

If you want more information on GhentFPG you can follow us on twitter (@ghentfpg), via Google Groups (http://groups.google.com/group/ghent-fpg), or by visiting us at irc.freenode.net in channel #ghentfpg.

Further reading

- http://www.haskell.org/haskellwiki/Ghent_Functional_Programming_Group
- http://groups.google.com/group/ghent-fpg