Haskell Communities and Activities Report

http://tinyurl.com/haskcar

Twenty-Fourth Edition — May 2013

Janis Voigtländer (ed.)

Andreas Abel, Heinrich Apfelmus, Emil Axelsson,
Doug Beardsley, Jean-Philippe Bernardy, Jeroen Bransen,
Joachim Breitner, Erik de Castro Lopo, Olaf Chitil,
Duncan Coutts, Nils Anders Danielsson, Atze Dijkstra,
Péter Diviánszky, Sebastian Erdweg, Dennis Felsing,
Julian Fleischer, Ben Gamari, Andy Georges,
Patai Gergely, Brett G. Giles, Andy Gill,
Torsten Grust, Jurriaan Hage, Bastiaan Heeren,
Mike Izbicki, PÁLI Gábor János, 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,
Mihai Maruseac, Kazutaka Matsuda, Dino Morelli,
JP Moresmau, Ben Moseley, Takayuki Muranushi,
Jürgen Nicklisch-Franken, Tom Nielsen, Rishiyur Nikhil,
Jens Petersen, Simon Peyton Jones, 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,
Daniel Wagner, Kazu Yamamoto, Edward Z. Yang,
Brent Yorgey

Preface

This is the 24th 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 October. 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' — Haskell 2014

Report by:	Ian Lynagh
Participants:	Carlos Camarão, Iavor Diatchki, Bas van Dijk, Ian Lynagh, John Meacham, Neil Mitchell, Ganesh Sittampalam, David Terei, Henk-Jan van Tuyl

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

The Haskell 2014 committee has now formed, and we would be delighted to receive your proposals for changes to the language. Please see http://hackage.haskell.org/trac/haskell-prime/wiki/Process for details on the proposal process.

The committee will meet 4 times a year, to consider proposals completed before:

- 1st August
- 1st November
- 1st February
- 1st May

So if you have been meaning to put the finishing touches to a proposal, then we would encourage you to do so by the end of July!

The source for the Haskell report will be updated as proposals are accepted, but new versions of the standard will only be released once a year, during January.

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 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 neural networks and computational quantum chemistry.

Further reading

http://themonadreader.wordpress.com/

2.2 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:

Ordinary and one-pass CPS transformation in the tagless-final style

We demonstrate ordinary and administrative-redex–less call-by-value Continuation Passing Style (CPS) transformation that assuredly produces well-typed terms and is patently total. It is natural to require the result of transforming a well-typed term be well-typed. In the tagless-final approach that requirement is satisfied automatically: after all, only well-typed terms are expressible. We impose a more stringent requirement that a transformation be total. In particular, the fact that the transformation handles all possible cases of the source terms must be patently, syntactically clear. The complete coverage must be so clear that the metalanguage compiler should be able to see that, without the aid of extra tools.

What makes CPS transform interesting is that the type of the result is different from the type of the source term: the CPS transform translates not only terms but also types. Moreover, the CPS type transform and the arrow type constructor do not commute.

Since the only thing we can do with tagless-final terms is to interpret them, the CPS transformer is written in the form of an interpreter. It interprets source terms yielding transformed terms, which can be interpreted in many ways. In particular, the terms can be interpreted by the CPS transformer again, yielding 2-CPS terms. CPS transformers are composable, as expected.

One particular interpretation of CPS-transformed terms is displaying them, so we can see the result of the transformation. We notice right away that the ordinary (Fischer or Plotkin) CPS transform introduces many administrative redices, which make the result too hard to read. Therefore, we implement Danvy and Filinski's one-pass CPS transform, which relies on the metalanguage to get rid of the administrative redices.

http://okmij.org/ftp/tagless-final/course/course.html#CPS

Embedding linear and affine lambda-calculi

One may think that only those domain-specific languages can be embedded in Haskell whose type system is a subset of that of Haskell. To counter that impression we show how to faithfully embed typed linear lambda calculus, so that only well-typed and well-formed linear terms are representable. Any bound variable must be referenced exactly once in the abstraction's body. Haskell as the metalanguage will statically reject as ill-typed the attempts to represent terms with a bound variable referenced several times — or, as in the K combinator, never.

A trivial modification turns the embedding into that of the affine lambda calculus, which allows to ignore the bound variable. K combinator becomes expressible. The tutorial code defines the typed linear lambda calculus and its two interpreters, to evaluate and to show linear lambda terms. Later we add general abstractions imposing no constraints on the use of bound

variables.

http://okmij.org/ftp/tagless-final/course/course.html#linear

Polyvariadic functions and keyword arguments: pattern-matching on the type of the context

Any function polymorphic in its return type is polyvariadic: The type can always be instantiated to the arrow type, letting the function accept one more argument. The identity function is hence polyvariadic, as one can easily check. Functions in the continuation-passing style are naturally polymorphic in the return, that is, the answer-type. Danvy's Functional Unparsing (typed, polyvariadic *printf*) is the clearest demonstration of the technique.

The tutorial collects more elaborate applications. The key idea is that the concrete type of a polymorphic term is determined by the context into which the term is placed. Pattern-matching on that concrete type lets the term determine the context of its use: specifically, the term can determine the number and the types of the values it receives as arguments, and what is expected in return. For example, given

$$\textbf{class } \mathsf{C}\ a \textbf{ where}$$
$$\quad f :: \mathsf{String} \to a$$

the term f `"Hello, "` has the polymorphic type $\mathsf{C}\ a \Rightarrow a$. If the term appears in the context $putStrLn\ \$\ f$ `"Hello, "` True `" world" '!'` the type variable a is instantiated to the concrete type $\mathsf{Bool} \to \mathsf{String} \to \mathsf{Char} \to \mathsf{String}$. Hopefully, there is an instance of C with this type, defining the required operation. Such instances can be built inductively:

$$\textbf{instance } \mathsf{C}\ x \Rightarrow \mathsf{C}\ (\mathsf{Char} \to x) \textbf{ where}$$
$$\quad f\ a\ x = f\ (a +\!\!+ [x])$$
$$\textbf{instance } \mathsf{C}\ x \Rightarrow \mathsf{C}\ (\mathsf{Bool} \to x) \textbf{ where}$$
$$\quad f\ a\ x = f\ (a +\!\!+ show\ x)$$

The class C thus lets f pattern-match on its continuation. This type class is the unifying pattern for previously described polyvariadic functions. The tutorial includes a new one: a generic transformation on a type-indexed collection (TIP).

TIP is a heterogeneous array whose elements have distinct types. Therefore, an element can be located based solely on its type. Given a function $t1 \to t2 \to \ldots \to tn \to tr$ we would like to apply it to a given TIP: locating the arguments in the TIP by their types $t1 \ldots tn$ and replacing the TIP element indexed by tr with the function's result. The number of arguments can be arbitrary, even zero. The types $t1 \ldots tn, tr$ do not have to be distinct – but they all must be in the domain of the TIP. Otherwise, a type error should be reported. The problem is a variation of the keyword argument problem, where the 'keyword' is the argument type.

http://okmij.org/ftp/Haskell/polyvariadic.html

2.3 Agda Tutorial

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

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

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

I currently work on the correction and completion of the existing material.

Further reading

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

3 Implementations

3.1 Haskell Platform

Report by:	Duncan Coutts

At the time of preparing this HCAR, everybody was poised for the 2013.2.0.0 release of the platform, but it did not appear in time to be included in the report. For the previous entry, see: http://www.haskell.org/communities/11-2012/html/report.html#sect3.1.

3.2 The Glasgow Haskell Compiler

Report by:	Simon Peyton Jones
Participants:	many others

The big event in late 2012 was the news of Simon Marlow's move to Facebook. Simon is the absolute master of huge tracts of GHC, including especially its runtime system, garbage collection, code generation, and support for parallelism. His contribution to GHC is a massive one, and this makes a good occasion for us to acknowledge it: **thank you Simon!**

Simon isn't going to disappear, of course, but he'll have less time to work on GHC than before. That means that everyone else, including you, gentle reader, has new opportunities to contribute to the huge shared community endeavour that we call GHC.

As planned, we made another minor release 7.6.2 from the 7.6 branch in January 2013. This included only bug and performance fixes; no new features were added. We plan to put out a new major release 7.8.1 soon after ICFP 2013, including some significant changes described below.

There remains more to do than we will ever have time for, so please do come and join in the fun!

Source language and type system:

Simon and Dimitrios overhauled the solver for type constraints, once more. No new functionality, but the result is smaller, faster, and lacks many of the bugs of its predecessor. You don't want to know all the details, but it reminded us again of how valuable it is that the constraint solver is now one coherent piece of code, with a well-defined task, rather than being spread out in bits and pieces across the type checker.

Meanwhile others have been adding new features.

Poly-kinded Typeable. The Typeable class is now kind-polymorphic, meaning we can finally drop the boilerplate TypeableN classes. The new definition of Typeable is as follows:

$$\textbf{class}\ \mathsf{Typeable}\ (a :: k)\ \textbf{where}$$
$$\quad \mathit{typeRep} :: \mathit{proxy}\ a \to \mathsf{TypeRep}$$

With this change comes the ability to derive Typeable instances for every user datatype, and even for type classes. This means user defined instances of Typeable are unnecessary. Furthermore, since ill-defined user instances can lead to runtime errors, they are now forbidden; the only way to get Typeable instances is by using the deriving mechanism. User-defined instances will be ignored, with a warning.

Migrating to this new Typeable is easy. Code that only derived Typeable instances, and did not mention any of the TypeableN classes, should work as before. Code that mentioned the TypeableN classes should be adapted to replace these by the poly-kinded Typeable class. User-defined instances of Typeable should be replaced by derived instances.

Additionally, a new compiler pragma `AutoDeriveTypeable` triggers automatic derivation of Typeable instances for all datatypes and classes defined in the module.

Type holes. A GHC extension called "type holes" [TYH] was added by Thijs Alkemade, under supervision of Sean Leather and with help from Simon Peyton Jones. When GHC encounters a hole in an expression, written as "_", it will generate an error message describing the type that is needed in place of that hole. It gives some helpful additional information, such as the origins of the type variables in the hole's type and the local bindings that can be used. Together with `-fdefer-type-errors` this should make it easier to write code step-by-step, using hints from the compiler about the unfinished parts.

Rebindable list syntax. A GHC extension called "overloaded lists" [OL] was added by Achim Krause, George Giorgidze, and colleagues. When this is turned on, the way GHC desugars explicit lists and lists in arithmetic sequence notation is changed. Instead of directly desugaring to built-in lists, a polymorphic witness function is used, similar to the desugaring of numeric literals. This allows for a more flexible use of list notations, supporting many different list-like types. In addition, the functions used in this desugaring process are completely rebindable.

Type level natural numbers. Iavor S. Diatchki has been working on a solver for equations involving type-level natural numbers. This allows simplifying

and reasoning about type-level terms involving arithmetic. Currently, the solver can evaluate equations and inequalities mentioning the type functions $(+)$, $(*)$, $(\uparrow)$, and $(\leqslant)$. The solver works pretty well when it can use evaluation to prove equalities (e.g., examples like $2 + 5 = x$, $2 + x = 5$). There is also some support for taking advantage of the commutativity and associativity of $(+)$, $(*)$. More experimental features include: support for $(-)$, which currently is implemented by desugaring to $(+)$; the type-level function FromNat1, which has special support for working with natural number literals, and thus can be used to expose some of their inductive structure. This work is currently on the `type-nats` branch, and the plan is to merge it into `HEAD` in the next few months.

Kinds without data. Trevor Elliott, Eric Mertens, and Iavor Diatchki have begun implementing support for "data kind" declarations, described in more detail on the GHC wiki [KD]. The idea is to allow a new form of declaration that introduces a new kind, whose members are described by the (type) constructors in the declaration. This is similar to promoting data declarations, except that no new value-level-constructors are declared, and it also allows the constructors to mention other kinds that do not have a corresponding type-level representation (e.g., $*$).

Ordered overlapping type family instances. Richard Eisenberg has implemented support for ordered overlapping type family instances, called branched instances. This allows type-level functions to use patterns in a similar way to term-level functions. For example:

```
type family Equals (x :: *) (y :: *) :: Bool
type instance where
  Equals x x = True
  Equals x y = False
```

Details can be found on the wiki page [OTF].

Back end and code generation:

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.

Support for vector (SSE/AVX) instructions. Support for SSE vector instructions, which permit 128-bit vectors, is now in `HEAD`. As part of this work, up to 6 arguments of type Double, Float, or vector can be passed in registers. Previously only 4 Float and 2 Double arguments could be passed in registers. AVX support will be added soon pending a refactoring of the code that implements vector primops.

Data Parallel Haskell:

Vectorisation Avoidance. Gabriele Keller and Manuel Chakravarty have extended the DPH vectoriser with an analysis that determines when expressions cannot profitably be vectorised. Vectorisation avoidance improves compile times for DPH programs, as well as simplifying the handling of vectorised primitive operations. This work is now complete and will be in GHC 7.8.

New Fusion Framework. Ben Lippmeier has been waging a protracted battle with the problem of array fusion. Absolute performance in DPH is critically dependent on a good array fusion system, but existing methods cannot properly fuse the code produced by the DPH vectoriser. An important case is when a produced array is consumed by multiple consumers. In vectorised code this is very common, but none of the "short cut" array fusion approaches can handle it — e.g. stream fusion used in Data.Vector, delayed array fusion in Repa, *foldr/build* fusion etc. The good news is that we've found a solution that handles this case and others, based on Richard Waters's series expressions, and are now working on an implementation. The new fusion system is embodied by a GHC plugin that performs a custom core-to-core transformation, and some added support to the existing Repa library. We're pushing to get the first

version working for a paper at the upcoming Haskell Symposium.

A faster I/O manager:

Andreas Voellmy performed a significant reworking of the IO manager to improve multicore scaling and sequential speed. The most significant problems of the old IO manager were (1) severe contention (under some workloads) on a single MVar holding the table of callbacks, (2) invoking a callback typically requires messaging across capabilities, (3) polling for ready files performs an OS context switch, causing excessive context switching. These problems contribute greatly to the response time of servers written in Haskell.

The redesigned IO manager addresses these problems in the following ways. We replace the single MVar for the callback table with a simple concurrent hash table, allowing for more concurrent registrations and callbacks. We use one IO manager service thread per capability, each with its own callback table and with the service thread for a given capability serving the waiting Haskell threads that were running (and will be woken up) on that capability. This further reduces contention on callback tables, ensures that notifying a thread is typically done without cross-capability messaging and allows the work of polling and notifying threads to be parallelized across cores. To reduce context switching, we modify the service loops to first poll without waiting, which can be done without releasing the HEC (which would typically incur an OS context switch).

The new IO manager also takes advantage of the edge-triggered and one-shot modes of epoll on Linux to achieve further performance improvements on Linux.

These changes result in substantial performance improvements in some applications. In particular, we implemented a minimal web server and found that performance with the new "parallel" IO manager improved by a factor of 19 versus the old IO manager; with the old IO manager, our server achieved a peak performance of roughly 45,000 http requests per second using 8 cores (performance degraded after 8 cores), while the same server using the parallel IO manager serves 860,000 requests/sec using 18 cores [PIO]. We have measured similar improvements in the response time of servers written in Haskell.

Kazu Yamamoto contributed greatly to the project by implementing the redesign for BSD-based systems using kqueue and by improving the code in order to bring it up to GHC's standards. In addition, Bryan O'Sullivan and Johan Tibell provided critical guidance and reviews.

Dynamic linking:

Ian Lynagh has changed GHCi to use dynamic libraries rather than static libraries. This means that we are now able to use the system linker to load packages, rather than having to implement our own linker. From the user's point of view, that means that a number of long-standing bugs in GHCi will be fixed, and it also reduces the amount of work needed to get a fully functional GHC port to a new platform. Currently, on Windows GHCi still uses static libraries, but we hope to have dynamic libraries working on Windows too by the time we release.

Cross compilation:

Three connected projects concerned cross-compilation

Registerised ARM support added using David Terei's LLVM compiler back end with Stephen Blackheath doing an initial ARMv5 version and LLVM patch and Karel Gardas working on floating point support, ARMv7 compatibility and LLVM headaches. Ben Gamari did work on the runtime linker for ARM.

General cross-compiling with much work by Stephen Blackheath and Gabor Greif (though many others have worked on this as well).

A cross-compiler for Apple iOS. iOS-specific parts [IOS] were mostly done by Stephen Blackheath with Luke Iannini on the Cabal patch, testing and supporting infrastructure, also with assistance and testing by Miëtek Bak and Jonathan Fischoff, and thanks to many others for testing; The iOS cross compiler was started back in 2009 by Stephen Blackheath with funding from Ryan Trinkle of iPwn Studios.

Thanks to Ian Lynagh for making it easy for us with integration, makefile refactoring and patience, and to David Terei for LLVM assistance.

Links:

- [TYH], http://www.haskell.org/haskellwiki/GHC/TypeHoles
- [OL], http://hackage.haskell.org/trac/ghc/wiki/OverloadedLists
- [KD], http://hackage.haskell.org/trac/ghc/wiki/GhcKinds/KindsWithoutData
- [OTF], Overlapping type family instances, http://hackage.haskell.org/trac/ghc/wiki/NewAxioms
- [CG], The new codegen is nearly ready to go live, http://hackage.haskell.org/trac/ghc/blog/newcg-update
- [PIO], The results are amazing, https://twitter.com/bos31337/status/284701554458640384
- [IOS], Building for Apple iOS targets, http://hackage.haskell.org/trac/ghc/wiki/Building/CrossCompiling/iOS

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 460 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 7.4.2, Haskell Platform 2012.4.0.0, Gtk2Hs, wxHaskell, XMonad, Pandoc, Gitit, Yesod, Happstack, Snap, Agda, git-annex, and so on – all of them will be available as part of the upcoming FreeBSD 8.4-RELEASE.

Note that Haskell ports are now built with dynamic linking on by default, and the GHC port uses the latest available version of GCC and binutils from the Ports Collection as GCC in base is obsolete and soon replaced with Clang. We have just started the preparations for Haskell Platform 2013.2.0.0, which will bring us GHC 7.6.3 to the Ports Collection soon. We also learned that our Haskell ports have been successfully imported to DPorts, an effort to use FreeBSD ports on DragonFly. In addition to this, there was some support added for LLVM-based code generation in our development repository.

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 628 source packages.

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

The just released stable Debian release ("wheezy") provides the Haskell Platform 2012.3.0.0 and GHC 7.4.1. We are already working on new features for the next release. Full support for running hoogle to search all installed Haskell documentation is in the making, and Debian "experimental" already ships GHC 7.6.3 and up-to-date versions of the libraries.

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 http://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.

In November Agda 2.3.2 was released, with the following changes (among others):

- Pattern synonyms (Stevan Andjelkovic and Adam Gundry).
- Modifications to the constraint solver (Andreas Abel).
- A LaTeX backend, with the aim to support both precise, Agda-style highlighting, and lhs2TeX-style alignment of code (Stevan Andjelkovic).
- The Emacs mode no longer depends on GHCi and haskell-mode (Peter Divianszky).
- The Emacs mode is more interactive: type-checking no longer blocks Emacs, and there is an option to highlight the expression that is currently being type-checked (Guilhem Moulin and Nils Anders Danielsson).

Further reading

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

4.2 MiniAgda

Report by:	Andreas Abel
Status:	experimental

See: http://www.haskell.org/communities/05-2012/html/report.html#sect4.2.

4.3 Disciple

Report by:	Ben Lippmeier
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?

- Over the last month we've been working on a new core language fragment, Disciple Core Flow, to support work on array fusion for Data Parallel Haskell (DPH). We're writing a GHC plugin that translates GHC core programs to Disciple Core Flow, performs array fusion, and translates back. We're using Disciple Core Flow instead of GHC Core directly because it has a simple (and working) external core format, which we use to test the fusion transform.

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

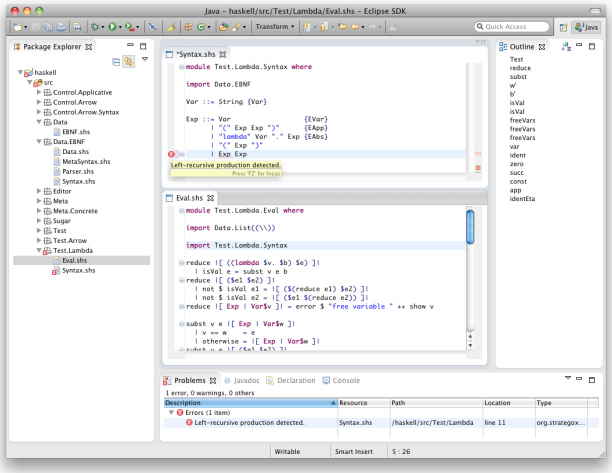
activate an extension, a SugarHaskell programmer simply imports the library that defines the extension; the extension is active in the remainder of the current file. Our Haskell Symposium paper [4] contains numerous examples, including arrow notation and, as illustrated in the following, idiom brackets:

```
import Control.Applicative
import Control.Applicative.IdiomBrackets

instance Traversable Tree where
  traverse f Leaf           = (| 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.



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

- Mischa Dieterle, Thomas Horstmeyer, Jost Berthold, Rita Loogen: *Iterating Skeletons — Structured Parallelism by Composition*, Selected Papers of the Symposium on the Implementation and Application of Functional Languages (IFL 2012), LNCS, Springer 2013, to appear.
- M. KH. Aswad, P. W. Trinder, A. D. Al-Zain, G. J. Michaelson, J. Berthold: *Comparing Low-Pain and No-Pain Multicore Haskells*, revised and extended version of TFP 2009 paper, in Special Issue of Higher-Order Symbol Computation (HOSC), to appear.

- Thomas Horstmeyer and Rita Loogen: *Graph-Based Communication in Eden*, revised and extended version of TFP 2009 paper, in Special Issue of Higher-Order Symbol Computation (HOSC), to appear.
- Oleg Lobachev, Michael Guthe, Rita Loogen: *Estimating parallel performance*, Journal of Parallel and Distributed Computing, Vol 73, No. 6, June 2013, pp. 876 - 887.

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

5.2 Haskell and the Web

5.2.1 WAI

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

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

WAI has mostly been stable since the last HCAR, with the exception of a newly added field to represent the request body length. This avoids repeatedly doing a costly integer parse, and correctly handling the case of chunked bodies at the type level. WAI has also been updated to allow the newest version of the conduit (→ 7.1.1) package.

WAI is also a platform for re-using code between web applications and web frameworks through WAI middleware and WAI applications. WAI middleware can inspect and transform a request, for example by automatically gzipping a response or logging a request. The Yesod (→ 5.2.6) web framework provides the ability to embed arbitrary WAI applications as subsites, making them a part of a larger web application.

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

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

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:	Michael Snoyman

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/
- https://github.com/snoyberg/posa-chapter/blob/master/warp.md

5.2.6 Yesod

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

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

Performance scalablity comes from the amazing GHC compiler and runtime. GHC provides fast code and built-in evented asynchronous IO.

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

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

- declarative routing with type-safe urls — say goodbye to broken links
- no XSS attacks — form submissions are automatically sanitized
- database safety through the Persistent library (→ 7.6.2) — 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 (→ 7.6.2). Views can be handled by the Shakespeare family of compile-time template languages. This includes Hamlet, which takes the tedium out of HTML. Both of these libraries are optional, and you can use any Haskell alternative. Controllers are invoked through declarative routing and can return different representations of a resource (html, json, etc).

Yesod is broken up into many smaller projects and leverages Wai (→ 5.2.1) to communicate with the server. This means that many of the powerful features of Yesod can be used in different web development stacks that use WAI such as Scotty.

The new 1.2 release of Yesod, introduces a number of simplifications, especially to the subsite handling. Most applications should be able to upgrade easily. Some of the notable features are:

- Much more powerful multi-representation support via the selectRep/provideRep API.
- More efficient session handling.
- All Handler functions live in a typeclass, providing you with auto-lifting.
- Type-based caching of responses via the cached function.
- More sensible subsite handling, switch to HandlerT/WidgetT transformers.
- Simplified dispatch system, including a lighter-weight Yesod.
- Simplified streaming data mechanism, for both database and non-database responses.
- Completely overhauled yesod-test, making it easier to use and providing cleaner integration with hspec.
- yesod-auth's email plugin now supports logging in via username in addition to email address.
- Refactored persistent module structure for clarity and ease-of-use.
- Easy asset combining for static javascript and css files
- Faster shakespeare template reloading and support for TypeScript templates.

Since the 1.0 release, Yesod has maintained a high level of API stability, and we intend to continue this tradition. The 1.2 release introduces a lot of potential code breakage, but most of the required updates should be very straightforward. Future directions for Yesod are now largely driven by community input and patches. We've been making progress on the goal of easier client-side interaction, and have high-level interaction with languages like Fay, TypeScript, and CoffeeScript.

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 continues to have a lot of activity since the last HCAR. We released Snap 0.10 which included a major redesign of the Heist template system with a huge performance improvement. That was followed by 0.11 as we continued to develop a higher level API on top of the new compiled Heist functionality that was introduced in 0.10.

The Snap team also released a new package called `io-streams` that provides a streaming I/O solution focused on simplicity and ease of use. The `io-streams` library includes a comprehensive test suite with 100% code coverage. The `io-streams` release was accompanied by a package providing OpenSSL support and a third-party HTTP client library called `http-streams`.

All in all, we are very happy with the continued growth of the Snap ecosystem. Going forward, we are working on a new Snap server built on `io-streams`. Also, several of the core Snap developers have full-time employment working with Snap in production systems and are continuing to develop new higher-level libraries and tools for commercial Haskell deployments. Join the team in the `#snapframework` IRC channel on Freenode to keep up with all the latest developments.

Further reading

- `io-streams` release announcement: http://snapframework.com/blog/2013/03/05/announcing-io-streams
- Snaplet Directory: http://snapframework.com/snaplets
- http://snapframework.com

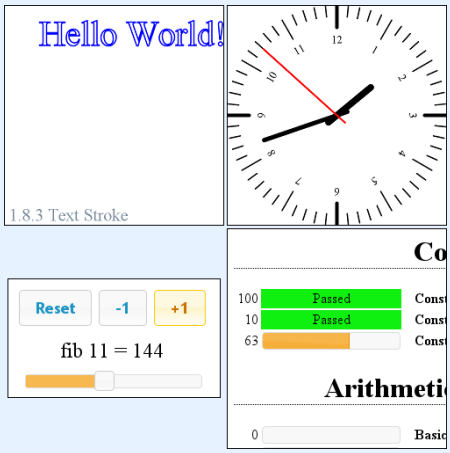
5.2.8 Sunroof

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

Sunroof is a Domain Specific Language (DSL) for generating JavaScript. It is built on top of the JS-monad, which, like the Haskell IO-monad, allows read and write access to external resources, but specifically JavaScript

resources. As such, Sunroof is primarily a feature-rich foreign function API to the browser's JavaScript engine, and all the browser-specific functionality, like HTML-based rendering, event handling, and drawing to the HTML5 canvas.

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



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

Further reading

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

5.3 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 suitable for compiler challenges, as already many times proven. 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.

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

Future plans are, to add an interpreter to gather profile information for the compiler and also do more aggressive optimizations (e.g. method inlining or stack allocation) , using the interpreter as fallback path via deoptimization if a assumption is violated.

Apart from that, many features are missing for a full JavaVM, most noteable are the concept of Classloaders, Floating Point or Threads. We would like to use GNU Classpath as base library some day. Other hot topics are Hoopl and Garbage Collection.

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 0.9.42.3 (April 2013), is extensively tested, and is available on Hackage. There is also a Cabal plugin for easy use of AG files in Haskell projects.

Some of the recent changes to the UUAG system are:

OCaml support. We have added OCaml code generation such that UUAG can also be used in OCaml projects.

Improved build system. 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.

First-class AGs. We provide a translation from UUAG to AspectAG (http://www.haskell.org/communities/11-2011/html/report.html#sect5.4.2). 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.

We are currently working on the following enhancements:

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 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

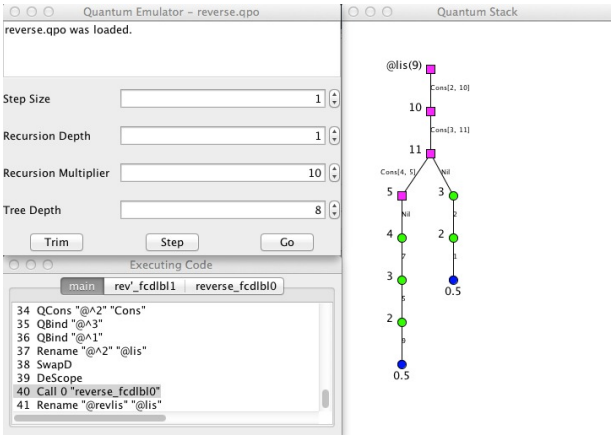
LQPL (Linear Quantum Programming Language) is a functional quantum programming language inspired by Peter Selinger's paper "Towards a Quantum Programming Language".

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.



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

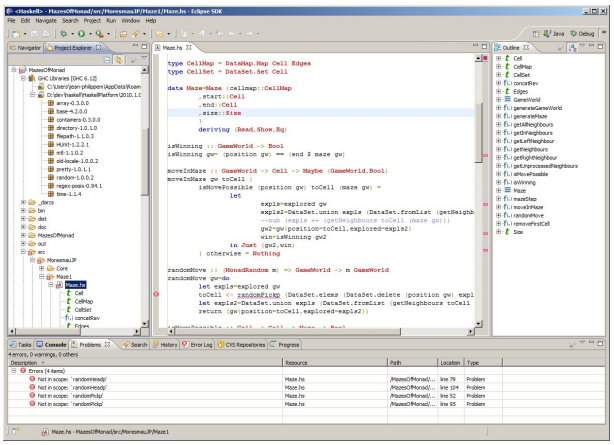
Documentation and executable downloads may be found at http://pll.cpsc.ucalgary.ca/lqpl/index.html. The source code, along with a wiki and bug tracker, is available at https://bitbucket.org/BrettGilesUofC/lqpl.

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



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, and imports can be organized automatically. 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. EclipseFP integrates with Haskell test frameworks, most notably HTF, to provide UI feedback on test failures. It can also use cabal-dev to provide sandboxing and project dependencies inside an Eclipse workspace. The source code is fully open source (Eclipse License) on github and anyone can contribute. Current version is 2.5.2, released in March 2013, 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 and github issues.

Further reading

- http://eclipsefp.github.com/
- http://jpmoresmau.blogspot.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

Here are new features:

- `ghc-mod` now analyses library dependencies from a cabal file.
- The "`check`" subcommand became faster than before unless Template Haskell is used.
- The "`debug`" subcommand is provided.
- The "`browse`" subcommand displays more information on functions etc if the "`-d`" option is specified.

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.

Version 5.05, with small improvements and bug-fixes, was released end of April 2013.

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 2.8.4 (with GHC 7.6 support), was in Februrary 2013. 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.

Our work on the next Darcs release continues. In our sights are the new 'darcs rebase' command (for merging and amending patches that would be hard to do with patch theory alone), the patch index optimisation (for faster local lookups on repositories with long histories), and the packs optimisation (for faster darcs get).

To accompany this work are some very interesting lines of development being contributed by Sebastian Fischer on behalf of factis research. The work is aimed at improving the patch management experience for teams of Darcs users, first with a mechanism to track the history of patches (who pulled them into the repository, or removed them, when) and second with possible commands that would allow patches to be easily split apart and merged together. We're very grateful to factis for funding this development work and look forward to using it ourselves.

Meanwhile, the Darcs hub at http://hub.darcs.net continues to grow in robustness and usage (at the time of this writing, 254 accounts, 294 repos). The Darcs hub is based on work by Simon Michael improving the original Darcsden by Alex Suraci, resulting in a 1.0 release earlier last year. 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, the Darcs project itself, and a few developers. At the time of writing (April 2013), it was tracking 39 repositories and 4579 patches submitted by 244 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 benchmark option ("`-b`") for the "`conf`" subcommand

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.

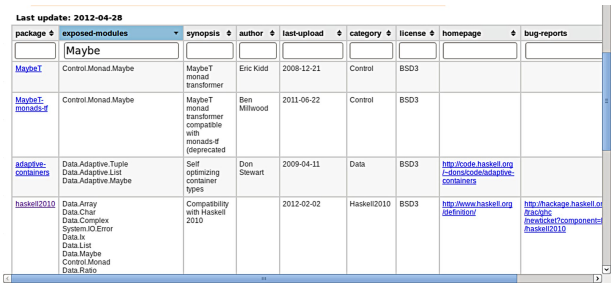
Further reading

- 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 (http://fremissant.net/portackage) is a web interface to all of http://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.



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 ghc-heap-view

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

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

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

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

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

The graphical tool ghc-vis (→ 6.4.3) builds on ghc-heap-view.

Further reading

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

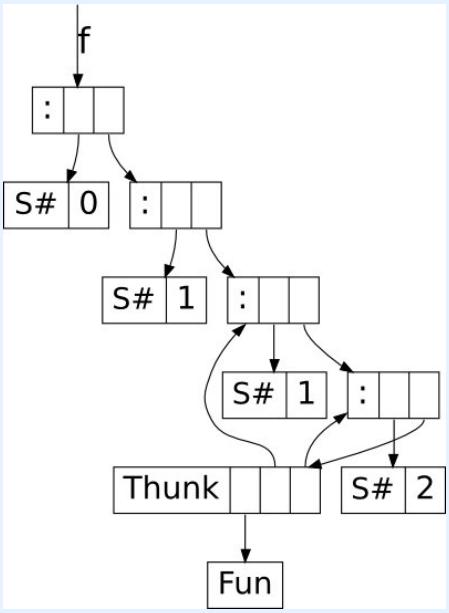
6.4.3 ghc-vis

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

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

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

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



At this point the visualization can be used interactively: To evaluate a thunk, simply click on it and immediately see the effects. You can even evaluate thunks which are normally not reachable by regular Haskell code.

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

Further reading

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

6.4.4 Hat — the Haskell Tracer

Report by: Olaf Chitil

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

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

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 tool. Hat comes with several tools for selectively viewing fragments of the trace in different ways: *hat-observe* for Hood-like observations, *hat-trail* for exploring a computation backwards, *hat-explore* for freely stepping through a computation, *hat-detect* for algorithmic debugging, ...

Hat is distributed as a package on Hackage that contains all Hat tools and tracing versions of standard libraries. Currently Hat supports Haskell 98 plus some language extensions such as multi-parameter type classes and functional dependencies. For portability all viewing tools have 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. This work is nearly finished and a new version of Hat will be released soon.

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://projects.haskell.org/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. Since the last HCAR, we've released version 1.0. This release introduces a simplification of the public facing API, optimizing for the common use cases. This was a minor change, and the conduit ecosystem has already caught up. The package has been in a mature state for quite some time now, and can be relied upon for most streaming data needs.

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

Further reading

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

7.1.2 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

http://fremissant.net/freesect

7.2 Education

7.2.1 Holmes, Plagiarism Detection for Haskell

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

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

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

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

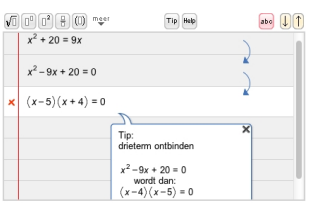
Contact

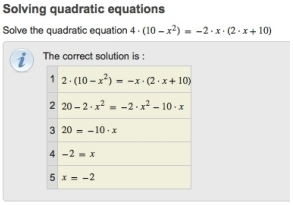
⟨J.Hage@uu.nl⟩

7.2.2 Interactive Domain Reasoners

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

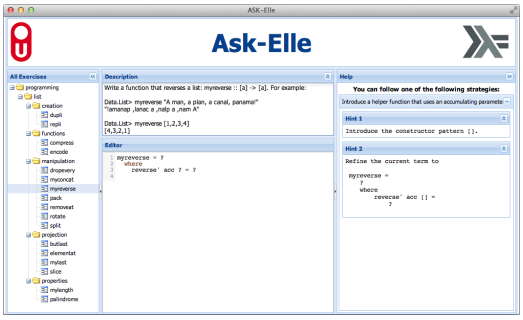
The Ideas project at Open Universiteit Nederland and Utrecht University aims at developing domain reasoners for stepwise exercises on various topics. These reasoners assist students in solving exercises incrementally by checking intermediate steps, providing feedback on how to continue, and detecting 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.





We have continued working on 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 analyze intermediate, incomplete programs and check whether or not certain properties are satisfied. We are currently extending the tutor with QuickCheck properties for

testing the correctness of student programs, and for the generation of counterexamples. Normalization of functional programs is used for dealing with all kinds of variations in programs; this is ongoing research. 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. 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.



We have modeled the artificial intelligence of a real-time video game on top of the strategy combinator language used in the domain reasoners. In the future we expect to develop a serious game for communication skills using a similar approach.

The feedback services are available as a Cabal source package. The latest release is version 1.0 from September 1, 2011. In the near future we will release a new version of the feedback services.

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.
- Johan Jeuring, Alex Gerdes, and Bastiaan Heeren. A Programming Tutor for Haskell. Lecture Notes Central European School on Functional Programming, (CEFP 2011). Try our tutor at http://ideas.cs.uu.nl/ProgTutor/.
- Tom Hastjarjanto, Johan Jeuring, and Sean Leather. A DSL for describing the artificial intelligence in real-time video games. Third International Workshop on Games and Software Engineering (GAS 2013).

7.3 Parsing and Transforming

7.3.1 FliPpr

Report by:	Kazutaka Matsuda
Participants:	Meng Wang
Status:	experimental, prototype

FliPpr ("flip" + "ppr" (pretty-printer)) is a program transformation tool that takes a pretty-printing program and returns a parser consistent with the pretty-printer. It is common that, when we implement a programming language, we have to write a pair of programs: a pretty-printer and a parser, and it is expected that, especially early in the development, the syntax of the language changes frequently. In such a case, we have to update both the pretty-printer and the parser so that they continue to work with each other. FliPpr removes this maintenance burden from the programmers through program inversion techniques such that a consistent parser (in the sense that pretty-printed code is always correctly parsed) is automatically generated.

Pretty-printers in FliPpr are written with Wadler's pretty-printing combinators as recursive functions on an AST datatype, which is very similar to what we normally do in Haskell. The differences are that there is a new combinator for embedding additional information for effective parsing, and there are syntactic restrictions in place. For details, please see our ESOP paper (doi: `10.1007/978-3-642-37036-6_6`).

Currently, the implementation is an experimental prototype. So bugs and unhelpful error messages are expected. However, you shall be able to play with the system by using the examples from the implementation page below.

Further reading

http://www-kb.is.s.u-tokyo.ac.jp/~kztk/FliPpr/

7.3.2 Utrecht Parser Combinator Library: uu-parsinglib

Report by:	Doaitse Swierstra
Status:	actively developed

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

As an example of its use we show how to fill a record from the command line. We start out by defining the

record which is to hold the options to be possibly set:

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

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

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

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

```
oName =
      name    `option` ("name", pString)
  <> ints     `options` ("int",  pNatural)
  <> prefers `flags`   [("agda",     Agda)
                       ,("haskell", Haskell)]
  <> address `field`
               (    city   `option` ("city"
                                    , pString)
                <> street `option` ("street"
                                    , pString)
               )
```

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

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

Features

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

Future plans

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

Contact

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

7.3.3 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 (→ 9.9), which aims to improve the applicability of Haskell-hosted Semi-Formal Models to High Assurance Development. Specifically, HERMIT uses a Haskell-hosted DSL and a 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.

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. A journal writeup of the KURE internals has been submitted to JFP, and is available on the group webpage. As for interfaces to the kernel, we currently have a command-line REPL, and an Android version is under development. Thus far, we have used HERMIT to successfully mechanize many smaller examples of program transformations, drawn from the literature on techniques such as concatenate vanishes, tupling transformation, and worker/wrapper. We are scaling up our capabilities, working on larger examples, and hope to submit a paper to the Haskell Symposium this summer.

HERMIT was also used in a large case study, led by Michael Adams from Portland State University in Oregon. Adams used HERMIT to mechanize the optimization of scrap your boilerplate generics, leading to execution speeds that were as fast as hand-optimized code (→ 7.4.3).

Further reading

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

7.4 Generic and Type-Level Programming

7.4.1 Unbound

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

See: http://www.haskell.org/communities/05-2012/html/report.html#sect7.4.1.

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
Participants:	Michael D. Adams, Andrew Farmer
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 paper linked below.

We have also investigated how to optimise the popular Scrap Your Boilerplate (SYB) generic programming library. Using a HERMIT (→ 7.3.3) script for implementing an optimisation pass in the compiler, we have removed all runtime overhead from SYB functions. More details can be found in the draft paper linked below.

Further reading

- Optimisation of Generic Programs through Inlining
- Optimizing SYB Is Easy!

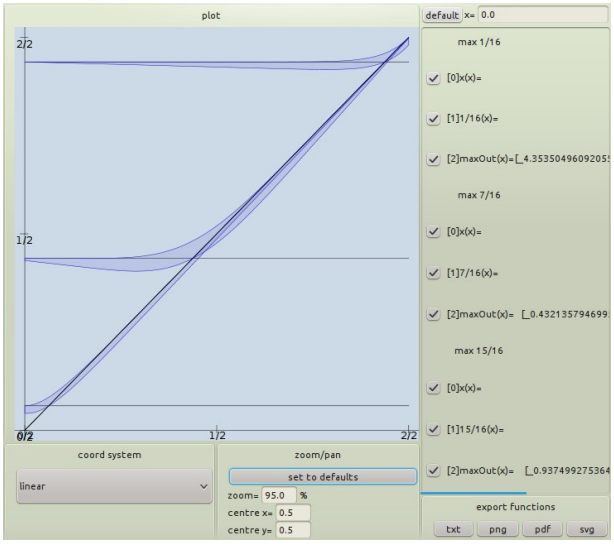
7.5 Mathematical Objects

7.5.1 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.



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/

7.5.2 Paraiso

Report by:	Takayuki Muranushi
Status:	active development

See: http://www.haskell.org/communities/05-2012/html/report.html#sect7.6.4.

7.5.3 bed-and-breakfast

Report by:	Julian Fleischer
Status:	active development

Bed and breakfast is a pure Haskell linear algebra library that I built between getting out of the bed and having breakfast — thus the name. It features basic operations on matrices, for which it makes use of boxed and unboxed mutable arrays as necessary to improve performance — everything is pure though, as stateful computations happen in the ST monad. It currently excels in inverting matrices and finding determinants.

Bed and breakfast is published under the MIT license and available via hackage as `bed-and-breakfast`. You are more than welcome to suggest improvements. Development happens at http://hub.darcs.net/scravy/bed-and-breakfast.

7.6 Data Types and Data Structures

7.6.1 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 (http://www.haskell.org/communities/11-2011/html/report.html#sect5.4.2), 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/

7.6.2 Persistent

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

Persistent is a type-safe data store interface for Haskell. Haskell has many different database bindings available, but they provide few usefeul static guarantees. Persistent uses knowledge of the data schema to provide a type-safe interface that re-uses existing database binding libraries. Persistent is designed to work across different databases, and works on Sqlite, PostgreSQL,

MongoDB, and MySQL, with an experimental backend for CouchDB.

The 1.2 release features a refactoring of the module hierarchy. We're taking this opporunity to clean up a few idiosyncracies in the API and make the documentation a bit more helpful, but otherwise the library is remaining unchanged.

The MongoDB backend features new helpers, query operators, and bug fixes for working with embedded/nested models. One can store a list of Maps or records inside a column/field. This is required for proper usage of MongoDB. In SQL an embedded object is stored as JSON, which is convenient as long as the column is not queried.

In order to accomodate various different backend types, Persistent is broken up into multiple components (separated by type classes). There is one for storage/serialization, one for uniqueness, and one for querying. This means that anyone wanting to create database abstractions can re-use the battle-tested persistent storage/serialization layer without having to implement the full query interface.

Persistent's query layer is the same for any backend that implement the query interface, although backends can define their own additional operators. The interface is a straightforward usage of combinators:

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

There are some drawbacks to the query layer: it doesn't cover every use case. Persistent has built-in some very good support for raw SQL. One can run arbitrary SQL queries and get back Haskell records or types for single columns. In addition, Felipe Lessa has created a library called esqueleto for having complete control over generating SQL but with type safety. persistent-MongoDB also has helpers for working with raw queries.

Future plans

Possible future directions for Persistent:

- Adding key-value databases such as Redis without a query layer.
- Full CouchDB support

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

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://www.yesodweb.com/book/persistent
- http://hackage.haskell.org/package/esqueleto

7.6.3 DSH — Database Supported Haskell

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

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

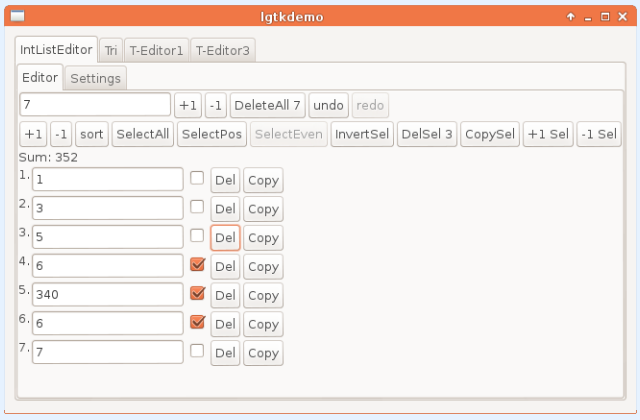
7.7 User Interfaces

7.7.1 LGtk: Lens-based Gtk API

Report by:	Péter Diviánszky
Status:	first release, experimental, actively developed

Most Haskellers would like to use a mature FRP-based API for creating graphical user interfaces. But FRP may not be the best tool for special user interfaces, like interfaces which consist mainly of buttons, check-boxes, combo boxes, text entries, tabs and menus. The goal of the LGtk project is to give a lens-based API which better fits these user interfaces. LGtk is built on Gtk2Hs.

The first release of LGtk was announced on 15 April 2013. Currently the first version of the API is available with a demo application.



LGtk is being actively developed. I currently work on the following items:

- Give an approximation of the semantics of LGtk and adjust the API to the given semantics.
- Support for asynchronous effects.
- Make a tutorial for developers with lots of small examples.

Further reading

http://people.inf.elte.hu/divip/LGtk/index.html

7.7.2 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

Since the last release, there have been many bug-fixes, and Peter Davies and Hamish Mackenzie have begun adding experimental Gtk3 support, enabled by the "gtk3" cabal flag.

Further reading

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

7.8 Functional Reactive Programming

7.8.1 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. The latest version of the reactive-banana library is `0.7.1.1`. Compared to the previous report, there has been no new public release as the API and its semantics have reached a stable plateau.

It turned out that the library suffered from a large class of space leaks concerning accumulated behaviors. The space semantics have been refined and the current development version eliminates these issues.

Current development. Programming graphical user interfaces has always been the driving force behind my efforts to implement an FRP library. Unfortunately, it appears that current Haskell GUI libraries, like Gtk2Hs and wxHaskell, are mostly dormant these days and not widely accessible due to installation woes on different platforms.

Moreover, in recent years, the web browser has emerged as an important platform for programming user interfaces. I would be happy to make reactive-banana available in this context as well, but unfortunately, efforts to compile Haskell to JavaScript are still in early stages of development.

To move forward, I have decided to work on implementing a small GUI framework, called `threepenny-gui`, which uses the web browser to display user interfaces written in Haskell. It is derived from Christopher Done's former `ji` project. While not directly related to FRP, I hope that this effort will make reactive-banana more widely accessible and help test its mettle in real-world tasks.

Concerning the development of reactive-banana itself, there are still some efficiency problems remaining, in particular concerning garbage collection of dynamic events. However, I feel that work on these remaining efficiency issues needs to be informed by practical applications and I wish to focus on the aforementioned GUI framework to lay a foundation for that first.

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
- threepenny-gui: https://github.com/HeinrichApfelmus/threepenny-gui

7.8.2 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)

7.9 Graphics

7.9.1 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.

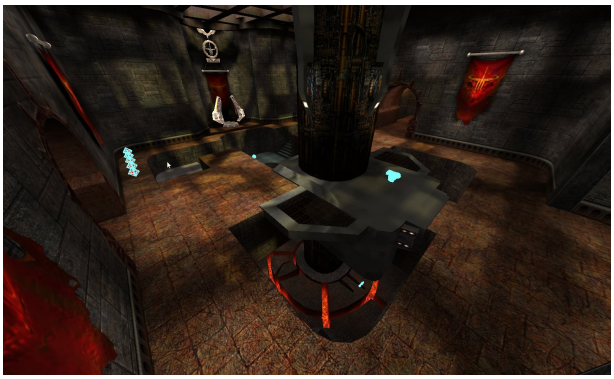
Over the last few months, we extended the implementation with some essential major features:

- 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

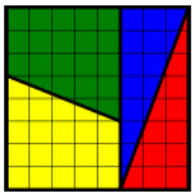
- https://lambdacube3d.wordpress.com/
- https://github.com/csabahruska/lc-dsl
- http://www.haskell.org/haskellwiki/LambdaCubeEngine
- http://www.youtube.com/watch?v=kDu5aCGc8l4

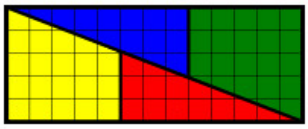
7.9.2 diagrams

Report by:	Brent Yorgey
Participants:	Daniel Bergey, 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.



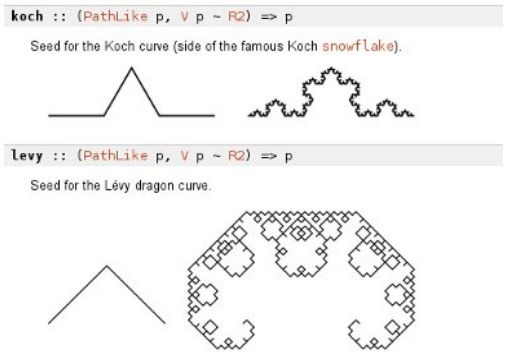


What's new

Since the last HCAR edition, version 0.6 was released in December. New features in 0.6 include:

- A new Haskell-native SVG backend now comes "out of the box"—making installation of diagrams far easier for many users, since it no longer depends on any external libraries via the FFI. There is also a new official Postscript backend. The cairo backend is still supported, but is no longer required to use diagrams.
- "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).

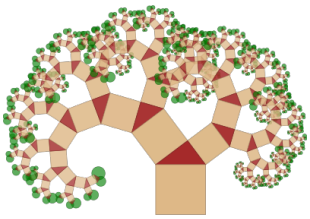
Perhaps the most exciting news since the last HCAR edition is the release of the `diagrams-haddock` package, which enables embedding diagrams code directly in Haddock documentation, with images automatically compiled and inserted into the Haddock output. This is an easy way to spruce up your documentation with declaratively constructed graphics. `diagrams-haddock` is already in use in a few packages on Hackage, most notably `diagrams-contrib`.

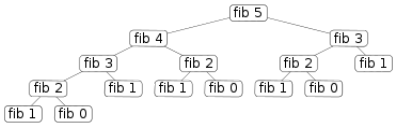


In conjunction with `diagrams-haddock`, there have also been significant improvements to the underlying `diagrams-builder` package, which renders diagrams dynamically at run time. The caching and conditional rebuilding of diagrams is now much smarter, so that it "does the right thing" in most situations (rebuilding a diagram when it has changed, and avoiding a rebuilt when it hasn't). This directly improves the experience of using `diagrams-haddock` as well as `BlogLiterately-diagrams` (for writing blog posts with embedded diagrams) and `diagrams-latex` (for LaTeX documents with embedded diagrams).

There have been many improvements and changes to the core diagrams libraries as well, with an 0.7 release planned for the not-too-distant future. Features slated for the upcoming release include:

- A nice API for drawing arrows between arbitrary points or diagrams.
- Additions to the `diagrams-contrib` library, including a symmetric layout algorithm for binary trees, circle packing layout, a generalized turtle drawing interface, factorization diagrams, and iterated subset fractals.
- Computing the curvature of path segments at a given point.
- Generating paths as a constant offset from another path.
- A generalized color API, allowing backends to use whatever color space they want.
- Many documentation improvements, using `diagrams-haddock` to generate example images.





Contributing

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

haskellwiki/Diagrams/Contributing, or come hang out in the `#diagrams` IRC channel on freenode.

Further reading

- http://projects.haskell.org/diagrams
- http://projects.haskell.org/diagrams/gallery.html
- http://haskell.org/haskellwiki/Diagrams
- http://github.com/diagrams
- 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

7.10 Audio

7.10.1 Audio Signal Processing

Report by:	Henning Thielemann
Status:	experimental, active development

See: http://www.haskell.org/communities/05-2012/html/report.html#sect7.11.1.

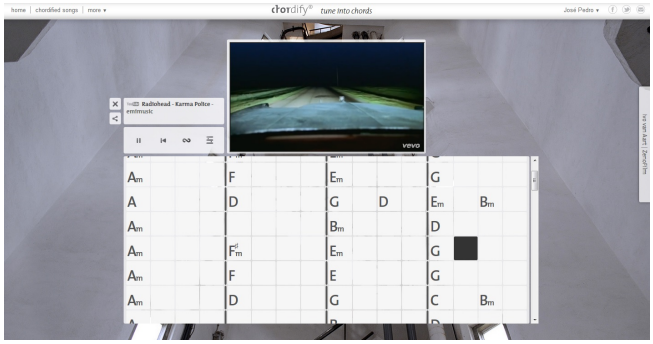
7.10.2 Live-Sequencer

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

See: http://www.haskell.org/communities/05-2012/html/report.html#sect7.11.2.

7.10.3 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.

Chordify is free for everyone to use. We have recently implemented a user account system, and keep adding new features on a regular basis. 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

7.10.4 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.

7.11 Text and Markup Languages

7.11.1 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).

7.11.2 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

7.12 Natural Language Processing

7.12.1 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.

New packages and projects in development

- *approx-rand-test 0.1* This version of the approximate randomization test package adds charting using Cairo. Charts show the outcomes of the test statistic for the original samples and randomized samples, as well as significance boundaries. If Cairo is not available, it can also draw ASCII charts.

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

Further reading

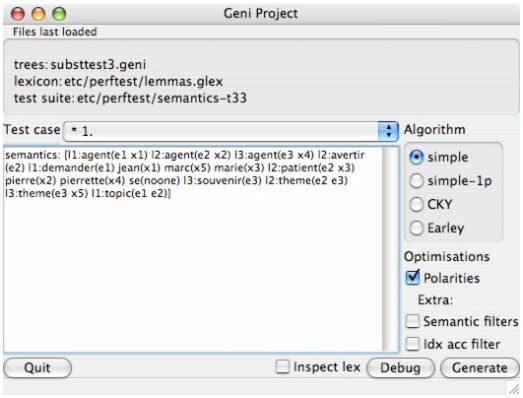
http://projects.haskell.org/nlp

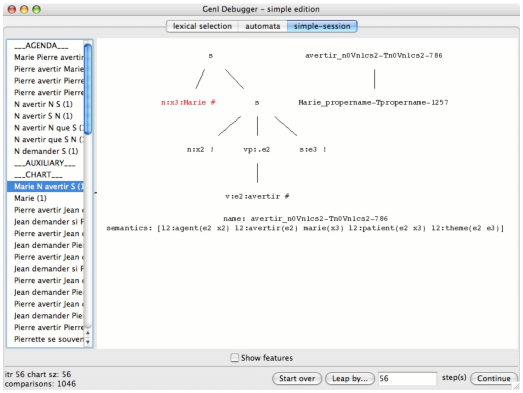
7.12.2 GenI

Report by:	Eric Kow

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

GenI is now mirrored on GitHub, with its issue tracker and wiki and homepage also hosted there. We are working on a new release (GenI 0.24) that allows for custom semantic inputs, making it simpler to use GenI in a wider variety for applications.





GenI is available on Hackage, and can be installed via cabal-install. Our most recent release of GenI was version 0.24 (2012-10-19). For more information, please contact us on the geni-users mailing list.

Further reading

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

7.13 Machine Learning

7.13.1 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/

7.13.2 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⟩.

7.14 Bioinformatics

7.14.1 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.

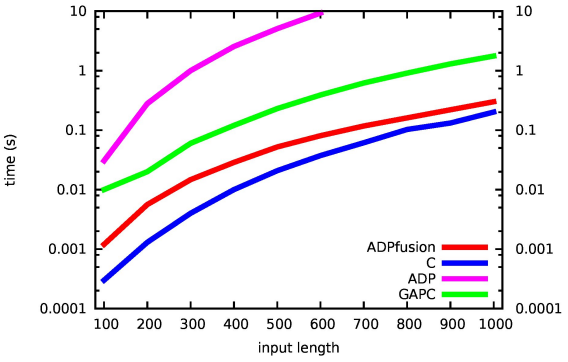
As an example, consider a grammar that recognizes palindromes. Given the non-terminal p, as well as parsers for single characters c and the empty input ϵ, the production rule for palindromes can be formulated as $p \rightarrow c\ p\ c \mid \epsilon$.

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*.



Starting with ADPfusion 0.2, dynamic programs on *more than one input* sequence can be written. This allows efficient dynamic programs that compute, say, the alignment of two or more inputs. More complicated algorithms of coupled context-free grammars also become possible with this new, *multi-dimensional* expansion. Together with generalised index spaces, more algorithms can be implemented efficiently, while at the same time reducing the effort required to implement these more complicated algorithms *correctly*.

Further reading

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

7.14.2 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 53 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

7.15 Embedding DSLs for Low-Level Processing

7.15.1 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

7.15.2 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.

- The Fabric monad is now a Monad transformer. The Fabric monad historically provided access to named input/output ports, and now also provides named variables, implemented by ports that loop back on themselves. This additional primitive capability allows for a *typed* state machine monad. This design gives an elegant stratospheric pattern: purely functional circuits using streams; a monad for layout over *space*; and a monad for state generation, that acts over *time*.
- On top of 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.

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

Further reading

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

7.16 Others

7.16.1 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/

7.16.2 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/

7.16.3 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

7.16.4 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

8 Commercial Users

8.1 Well-Typed LLP

Report by:	Andres Löh
Participants:	Ian Lynagh, 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 7.6.2 and 7.6.3 patch releases. The May 2013 release of the Haskell Platform will be based on 7.6.3. We are expecting a release of 7.8.1 for autumn.

On behalf of the Industrial Haskell Group (IHG) (→ 8.3), we are making good progress on getting Hackage 2 ready. We have already entered the transition period. Both Hackage and Hackage 2 are currently running in parallel, with Hackage 2 currently being in alpha-testing.

We continue to be involved in the community, maintaining several packages on Hackage and giving talks at a number of conferences. Some of our recent appearances are available online, such as Duncan's talk on Cloud Haskell at the Haskell eXchange, Duncan's interview with InfoQ on Parallelism, Concurrency and Distributed Programming in Haskell, and Andres's talk on Datatype-Generic Programming at Skills Matter (links below).

We are in the process of expanding our training activities. Please see the "Training" section of our website for more details and feel free to contact us if you are interested in Haskell training courses.

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

- Company page: http://www.well-typed.com
- Blog: http://blog.well-typed.com/
- Training page: http://www.well-typed.com/services_training
- Duncan's Cloud Haskell talk: http://skillsmatter.com/podcast/home/cloud-haskell
- Duncan's InfoQ interview: http://www.infoq.com/interviews/coutts-haskell
- Andres's Datatype-Generic Programming talk: http://skillsmatter.com/podcast/haskell/a-haskell-lecture-with-leading-expert-andres-loh

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 the new Hackage server (→ 6.3.1).

As an important milestone, the new Hackage server is currently in alpha testing: it is continuously running side-by-side with the old Hackage server, packages are mirrored every 30 minutes from the old server, user accounts have been ported from the old server. So it is in principle possible to use the new Hackage server right now and to play with it. You are invited to try it and provide feedback.

The new server (as well as any future updates) are for now 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 very interested in new members, in particular if they 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 questions or 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 uuparsinglib (→ 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

Oblomov Systems is a one-person software company based in Utrecht, The Netherlands. Founded in 2009 for the Proxima 2.0 project (http://www.haskell.org/communities/05-2010/html/report.html#sect6.4.5), Oblomov has since then been working on a number of Haskell-related projects. The main focus lies on web-applications and (web-based) editors. Haskell has turned out to be extremely useful for implementing web servers that communicate with JavaScript clients or iPhone apps.

Awaiting the acceptance of Haskell by the world at large, Oblomov Systems also offers software solutions in Java, Objective C, and C#, as well as on the iPhone/iPad. Last year, Oblomov Systems has worked together with Ordina NV on a substantial Haskell project for the Council for the Judiciary in The Netherlands.

Further reading

http://www.oblomov.com

8.6 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

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, and it is now also avaiable on Hackage. We also have an associated FreeBSD port available (www/hs-activehs) to make deployment and maintenance at our department server easier.

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://people.inf.elte.hu/divip/AgdaTutorial/Index.html (→ 2.3)
- 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. Extended call-by-need lambda calculi with `letrec` model the core language of Haskell. In several investigations we analyzed such calculi. Our obtained results include correctness of strictness analysis using abstract reduction, equivalence of the call-by-name and call-by-need semantics, completeness of applicative bisimilarity w.r.t. contextual equivalence, and unsoundness of applicative bisimilarity in nondeterminstic languages with `letrec`. Most recently, we showed that any semantic investigation of Haskell should include the `seq`-operator, since extending the lazy lambda calculus by `seq` (and also by data constructors and case, unless the core language is typed) is not conservative, i.e. the semantics changes.

Another recent result is that deciding (extended) α-equivalence in languages with bindings (like `let`) is graph isomorphism complete. However, if the expressions are free of garbage (i.e. have no unused bindings) the problem can be solved in polynomial time.

Recently, we analyzed a higher-order functional language with concurrent threads, monadic IO, synchronizing variables and concurrent futures which models Concurrent Haskell. We proved correctness of program transformations, correctness of an abstract machine, and we have shown that this language 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 proved correctness of a highly concurrent implementation of Software Transactional Memory (STM) in a similar program calculus. We also prototypically implemented a library of our STM-approach in Haskell.

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

Parallelization. Recently, we compared several approaches to parallelize the Davis-Putnam-Logemann-Loveland algorithm in Haskell using Parallel and Concurrent Haskell.

Grammar based compression. This research topic focuses on algorithms on grammar compressed strings and trees and 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.

Further reading

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

9.3 Functional Programming at the University of Kent

Report by:	Olaf Chitil

The Functional Programming group at Kent is a subgroup of the Programming Languages and Systems Group of the School of Computing. We are a group of staff and students with shared interests in functional programming. While our work is not limited to Haskell, we use for example also Erlang and ML, Haskell provides a major focus and common language for teaching and research.

Our members pursue a variety of Haskell-related projects, several of which are reported in other sections of this report. Thomas Schilling is writing up his PhD work on trace-based dynamic optimisations for Haskell programs. Olaf Chitil leads new developments of the Haskell tracer Hat.

We are expecting several new PhD students in our group in September, who will be working on refactoring, types and tracing, but we are always looking for more PhD students. 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, a library for building IDEs for Haskell: http://code.google.com/p/scion-lib/
- Hat, the Haskell Tracer: http://projects.haskell.org/hat/
- Practial Lazy Typed Contracts for Haskell: http://www.cs.kent.ac.uk/~oc/contracts.html
- Heat, an IDE for learning Haskell: http://www.cs.kent.ac.uk/projects/heat/

9.4 Formal Methods at DFKI and University Bremen

Report by:	Christian Maeder
Participants:	Mihai Codescu, 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 is a parsing, static analysis and proof management tool incorporating various provers and different specification languages, thus providing a tool for heterogeneous specifications. Logic translations are first-class citizens.

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

Hets can speak to the following provers:

- minisat, zChaff (SAT solvers),
- SPASS, Vampire, Darwin, KRHyper and MathServe (automated first-order theorem provers),
- Pellet and Fact++ (description logic tableau provers),
- Leo-II and Satallax (automated higher-order theorem provers),
- Isabelle (an interactive higher-order theorem prover),
- CSPCASL-prover (an Isabelle-based prover for CspCASL),
- VSE (an interactive prover for dynamic logic).

The user interface of the Hets implementation (about 200K lines of Haskell code) is based on some Haskell sources such as bindings to uDrawGraph (formerly Davinci) and Tcl/TK that we maintain and also gtk2hs (→ 7.7.2). Additionally we have a command line interface and a prototypcial web interface based on warp (→ 5.2.2) with a RESTful API.

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
- distributed ontology language DOL:
  http://www.ontoiop.org
- Heterogeneous tool set:
  http://hets.dfki.de
  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 Haskell in Romania

Report by: Mihai Maruseac
Participants: Dan Popa

In Romania, Haskell is taught at several universities across the country: in Bucharest at both University POLITEHNICA of Bucharest and University of Bucharest, in Bacău at "Vasile Alecsandri" University, in Braşov at "Transilvania" University, ... However, everywhere the courses are only centered on the theoretical aspects of functional programming and (sometimes) type systems. As a result, very few students will use this language after the exam is taken.

However, small communities are created to promote the language. That was the case of the Ro/Haskell group from Bacău or FPBucharest group. Right now, almost all of these groups have stopped being active.

The main reason behind these failures is that the point of view in presenting the language is too deeply concerned with presenting its features and the purely functional aspect while hiding away the fact that you have to do some IO in real world applications. Basically, every activity of the previous groups and the subjects taught at universities regard Haskell only as a laboratory language.

A small group of people from Faculty of Automatic Control and Computers, University POLITEHNICA of Bucharest, decided last year to change that. The new teachers and teaching assistants from the Programming Paradigm course organised the first "Functional Programming Summer School" in June 2012 where a few real-world topics were presented among more theoretical aspects.

This year, a small subgroup of the ROSEdu (http://rosedu.org/) community developed on the feedback from the summer school and created a plan towards making Haskell a known and usable language with a community around it. There were talks on Yesod and GHC at different events (OSOM, Talks by Softbinator) or companies (IXIA), there are some projects ready to be launched and there is a workshop called "Programming Haskell from N00b to Real World Programmer" to be organized in June, during ROSEdu Summer Workshops. Not to mention the possibility of the second edition of the "Functional Programming Summer School". Lastly, a ROSEdu member agreed to publish each month an article about real-world Haskell programming in the Today Software Magazine (http://www.todaysoftmag.com/tsm/en/).

9.7 fp-syd: Functional Programming in Sydney, Australia

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

We are a seminar and social group for people in Sydney, Australia, interested in Functional Programming and related fields. Members of the group include users of Haskell, Ocaml, LISP, Scala, F#, Scheme and others. We have 10 meetings per year (Feb–Nov) and meet on the third (usually, sometimes fourth) Wednesday 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, Haskell database libraries, Scala and the Free Monad. 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/2013

9.8 Functional Programming at Chalmers

Report by: Jean-Philippe Bernardy
Participants: John Hughes, Mary Sheeran, Aarne Ranta, Patrik Jansson, Koen Claessen, Björn von Sydow, Johan Nordlander, Nils Anders Danielsson, Alejandro Russo, Ulf Norell, Meng Wang, Josef Svenningsson, Emil Axelsson, Moa Johansson, ...

Functional Programming is an important component of the CSE department at Chalmers and U. of Gothenburg. 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. The last year FP research at Chalmers has been further strengthened by recruitment of four Ass. Prof. and several PhD students and a few more recruitments are in the pipeline. We also hope to see all HCAR readers at ICFP 2014 in Gothenburg — welcome!

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 and tools and related techniques are developed:

- PULSE, the ProTest User-Level Scheduler for Erlang, which has been used to find race conditions in industrial software.
- We have shown how to successfully apply QuickCheck to test polymorphic properties.
- A new exhaustive testing tool (testing-feat on Hackage) has been developed. It is especially suited to generate test cases from large groups of mutually recursive syntax tree types. A paper describing it was presented at the Haskell Symposium 2012.
- Testing Type Class Laws: the specification of a class in Haskell often starts with stating, in comments, the laws that should be satisfied by methods defined in instances of the class, followed by the type of the methods of the class. We have developed a library (ClassLaws) that supports testing such class laws using QuickCheck.

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.

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

- A Haskell backend, which offers incremental and parallel parsing capabilities, has been added to BNFC. The underlying concepts are described in this draft paper: http://www.cse.chalmers.se/~bernardy/PP.pdf.
- BNFC has been embedded in a library (called BNFC-meta on Hackage) using Template-Haskell. An important aspect of BNFC-meta is that it automatically provides quasi-quotes for the specified language. This includes a powerful and flexible facility for anti-quotation.

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

programmer. A draft paper describing the idea is available.

Jansson and Bernardy are also working on a project called "Strongly Typed Libraries for Programs and Proofs". This has led to publications and libraries for Testing Type Class Laws, Functional Enumeration of Algebraic Types (FEAT), Testing versus proving in climate impact research and Dependently-typed programming in scientific computing — examples from economic modelling. The last two are part of our effort to contribute to the emerging research programme in Global Systems Science (as part of the 2014–2020 European Union research funding scheme Horizon 2020).

Program Inversion. FliPpr (→ 7.3.1) is a program transformation system that through program inversion generates a consistent parser from a pretty-printer (written with Wadler's pretty-printing combinators), so that pretty-printed code is always correctly parsed. The work is done in collaboration with University of Tokyo, and a paper on it was presented at ESOP 2013.

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** (→ 7.15.1) 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.

Furthermore we are currently developing a framework for convenient definitions of DSLs (essentially a DSL for DSLs). A first result in this area is the syntactic library, whose core was presented at ICFP 2012. The paper presents a generic model of typed abstract syntax trees in Haskell, which can serve as a basis for a library supporting the implementation of deeply embedded DSLs.

- **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 BSc programme. We have four courses solely dedicated to functional programming (of which three are MSc-level courses), but we also provide courses which use Haskell for teaching other aspects of computer science, such as programming languages, compiler construction, data structures, parallel programming and programming paradigms.

9.9 Functional Programming at KU

Report by:	Andy Gill
Status:	ongoing

Functional Programming continues at KU and the Computer Systems Design Laboratory in ITTC! The System Level Design Group (lead by Perry Alexander) and the Functional Programming Group (lead by Andy Gill) together form the core functional programming initiative at KU. There are three major Haskell projects at KU (as well as numerous smaller ones): the GHC rewrite plugin HERMIT (→ 7.3.3), the VHDL generator Kansas Lava (→ 7.15.2) and the JavaScript generator Sunroof (→ 5.2.8).

Nicolas Frisby, who defended his PhD from KU last summer, spent the spring at MSR Cambridge, working on optimizations inside the Glasgow Haskell compiler. Andrew Farmer spent the spring in Portland, OR, working with functional programmers at Portland State.

Further reading

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

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